



Платформа Битрикс

Бизнес-процессы. Документация для разработчиков





Содержание:

Введение	3
Глава 1. Что такое бизнес-процессы	4
ШАБЛОН БИЗНЕС-ПРОЦЕССА.....	4
БИЗНЕС-ПРОЦЕСС	6
Глава 2. Типы бизнес-процессов.....	7
ПОСЛЕДОВАТЕЛЬНЫЙ БИЗНЕС-ПРОЦЕСС	7
БИЗНЕС-ПРОЦЕССА СО СТАТУСАМИ.....	8
ВЫБОР ТИПА БИЗНЕС-ПРОЦЕССА	10
Глава 3. Действия.....	11
МЕТОДЫ КЛАССА СВРАCTIVITY	11
Члены класса СВRActivity	14
СВОЙСТВА ДЕЙСТВИЙ	15
ОСНОВНЫЕ СТАНДАРТНЫЕ ДЕЙСТВИЯ	17
ИНТЕРФЕЙСЫ ДЕЙСТВИЙ.....	18
СОЗДАНИЕ СОБСТВЕННЫХ ДЕЙСТВИЙ.....	24
Глава 4. Исполняющая среда бизнес-процессов	28
ОБЪЕКТ БИЗНЕС-ПРОЦЕССА.....	29
Основные методы объекта-оболочки.....	29
СЕРВИСЫ ИСПОЛНЯЮЩЕЙ СРЕДЫ БИЗНЕС-ПРОЦЕССОВ.....	31
ДОКУМЕНТ И ТИП ДОКУМЕНТА.....	32
Методы интерфейса IBPWorkflowDocument.....	33
КЛАСС-ОБЕРТКА СВРDOCUMENT	39
Заключение	49



Введение

В состав "1С-Битрикс: Управление сайтом" и "1С-Битрикс: Корпоративный портал" разных редакций входят два модуля, обеспечивающих пользователям работу с информацией в рамках бизнес-процессов.

Модуль **Бизнес-процессы** представляет собой технологию определения, выполнения и управления бизнес-процессами (англ. workflow - рабочий поток). Модуль ориентирован на визуальное программирование и использует декларативную модель программирования.

⚠ Примечание. Модуль Бизнес-процессы работает только на PHP 5 и выше. Перед его установкой проверьте соответствие вашей установки этому требованию.

Цель документа – дать представление о возможностях этих модулей, принципах их работы, способах создания новых действий.

Руководство предназначено для разработчиков проектов на основе продуктов компании "1С-Битрикс". При составлении документа подразумевалось, что читатель владеет терминологией и основными приемами работы с продуктами компании "1С-Битрикс".



Глава 1. Что такое бизнес-процессы

Технология бизнес-процессов позволяет специалисту в своей предметной области (не программисту) автоматизировать свою деятельность, решить возникшую задачу наглядным способом. Причем сделать это таким образом, чтобы другой специалист в этой области мог понять и изменить данное решение.

Программистам технология бизнес-процессов позволяет сделать решения более понятными, универсальными и масштабируемыми.

Шаблон бизнес-процесса

Шаблон бизнес-процесса – это схема (программа), в которой задана одна точка входа, последовательность действий (шагов, этапов, функций), совершаемых в заданном порядке и направленных на достижение некоторой цели, а так же одна или несколько точек выхода, определяющих завершение выполнения.

Шаблон создается в специальном модуле **Дизайнер бизнес-процессов** с помощью визуального конструктора. Конструктор позволяет перетаскивать действия из панели инструментов на Основную рабочую область конструктора, создавая шаблон бизнес-процесса визуальным образом. Шаблон создается в виде блок-схемы, которая наглядно отображает логику работы бизнес-процесса.

Шаблон бизнес-процесса конструируется из ряда действий (англ. activities). Действием может быть отправка электронной почты, обновление строки в базе данных, публикация документа и т.п. Существует ряд встроенных действий, которые могут быть использованы для выполнения работ общего назначения, и, кроме того, при необходимости есть возможность создавать собственные действия.

Во внутренней архитектуре бизнес-процесса шаблон бизнес-процесса представляется в виде многомерного массива, содержащего иерархию действий и значения их свойств. Именно с таким представлением шаблона бизнес-процесса работает API модуля **Бизнес-процессы**. Пример простого массива, представляющего шаблон бизнес-процесса:

```
array(  
  array(  
    "Type" => "SequentialWorkflowActivity",  
    "Name" => "SequentialWorkflowActivity1",  
    "Properties" => array(),  
    "Children" => array(  
      array(  
        "Type" => "SetFieldActivity",  
        "Name" => "SetFieldActivity1",  
        "Properties" => array("Field" => "XML_ID", "Value" => "В рассмотрении"),  
      ),  
      array(  
        "Type" => "IfElseActivity",  
        "Name" => "IfElseActivity1",
```




)

Бизнес-процесс

Бизнес-процесс – это конкретный экземпляр шаблона бизнес-процесса. Он создается по требованию, запускается с точки входа и заканчивает работу по достижении одной из точек выхода. Для одного шаблона может быть одновременно запущено неограниченное число бизнес-процессов.

После завершения бизнес-процесс перестает существовать. Но его статус сохраняется и доступен для использования.

Бизнес-процесс всегда выполняется над определенным документом, определяющим его кодом. При этом документ может не иметь физического представления (т.е. быть виртуальным). Бизнес-процесс может быть настроен на автоматический запуск при добавлении или изменении документа.

Каждый бизнес-процесс уникально идентифицирован с помощью его кода, который может быть назначен исполняющей средой или же задан программистом. По коду можно обратиться к определенному бизнес-процессу.

Бизнес-процессу может быть отправлено событие с помощью методов исполняющей среды. Сообщение отправляется бизнес-процессу по его уникальному коду.

При запуске бизнес-процесс может принимать на вход значения параметров, список которых задается при конструировании шаблона бизнес-процесса. Например, это могут быть идентификатор заказа или код текущего пользователя. Любое действие бизнес-процесса будет иметь доступ к этим параметрам.

Бизнес-процесс может не выполняться постоянно. Например, если в бизнес-процессе встречается действие **CBPDelayActivity** (реализует ожидание, откладывая выполнение на определенный срок), то бизнес-процесс входит в состояние ожидания, сохраняется в базе данных и удаляется из памяти. По истечении заданного времени бизнес-процесс считывается из базы данных, восстанавливается в памяти и продолжает выполняться с места остановки.



Глава 2. Типы бизнес-процессов

При помощи модуля **Бизнес-процессы** могут быть описаны два типа бизнес-процессов: последовательный и со статусами.

Последовательный бизнес-процесс

Последовательный бизнес-процесс — действия выполняются одно за другим от точки входа до точки выхода (Рис. 2.1).

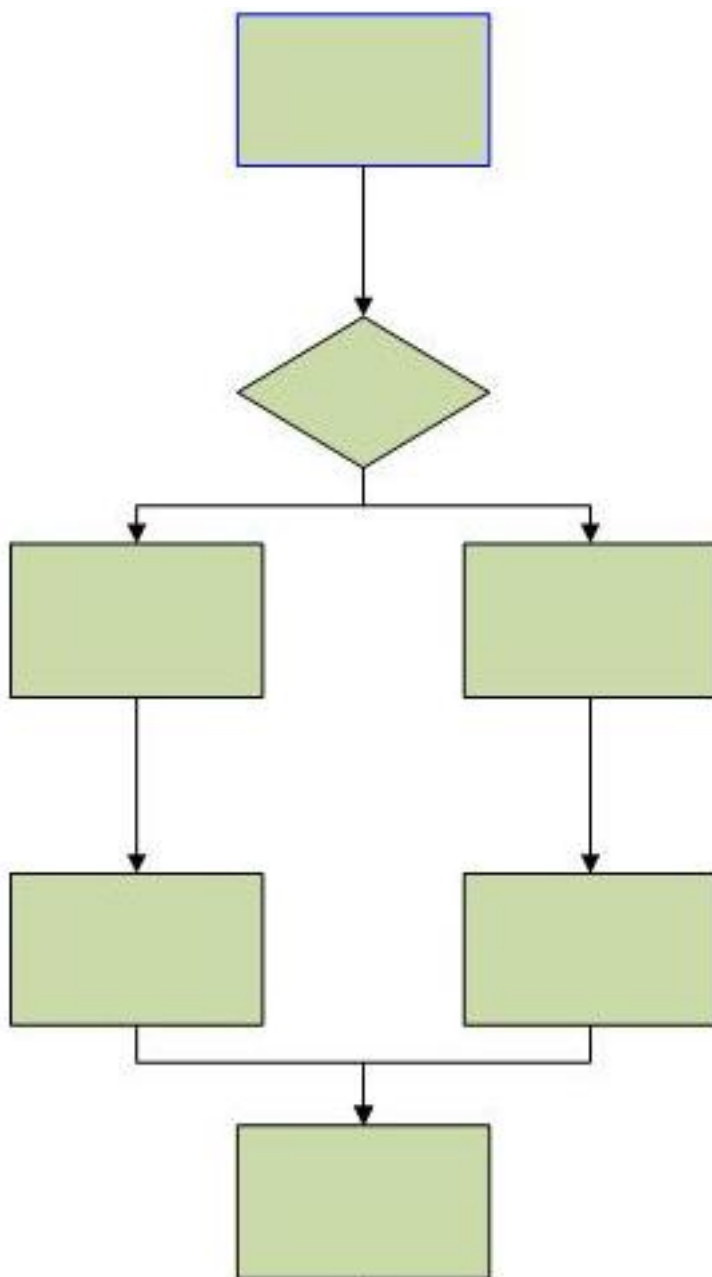


Рис. 2.1 Схема последовательного бизнес-процесса

Последовательный бизнес-процесс представляет собой бизнес-процесс как набор шагов, которые следует выполнять по порядку до тех пор, пока они все не завершатся. Он похож на обычную блок-схему, описывающую алгоритм решения задачи.



Последовательный бизнес-процесс начинает работу, выполняя находящееся в нем первое дочернее действие, и продолжается до тех пор, пока не выполнит все остальные дочерние действия.

Пример последовательного бизнес-процесса (Рис. 2.2):

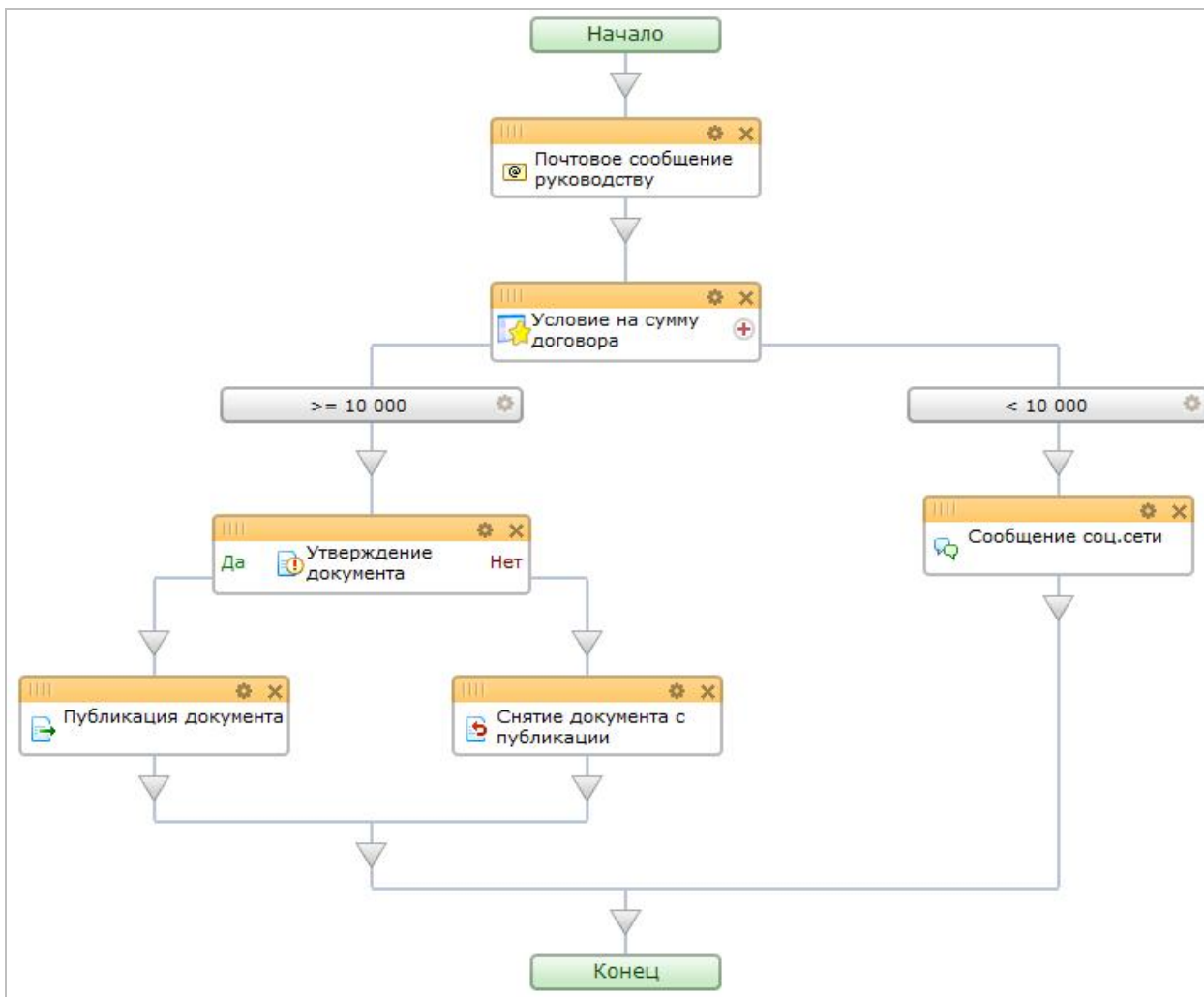


Рис. 2.2 Пример последовательного бизнес-процесса

Бизнес-процесса со статусами

Бизнес-процесс со статусами (так же известный как **машина состояний** или **автомат на состояниях**) — начала и конца не имеет, в процессе работы происходит переход из одного состояния в другое (Рис. 2.3).

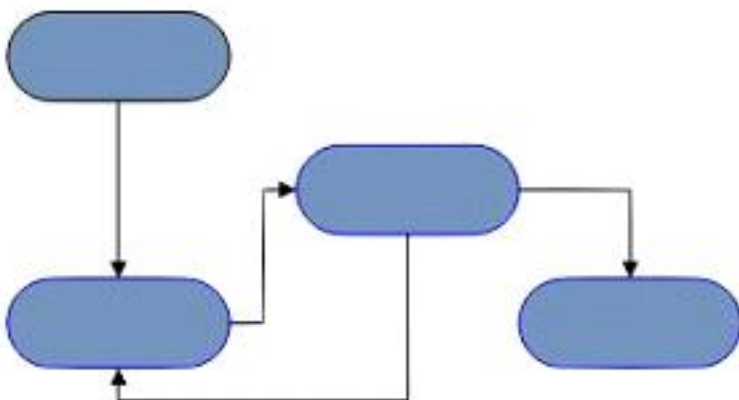


Рис. 2.3 Схема бизнес-процесса со статусами

Бизнес-процесс со статусами представляет собой набор состояний, переходов и действий. Одно состояние обозначается как начальное состояние. В процессе выполнения процесс переходит из одного состояния в другое, основываясь на событиях.

Пример бизнес-процесса со статусами (Рис. 2.4):

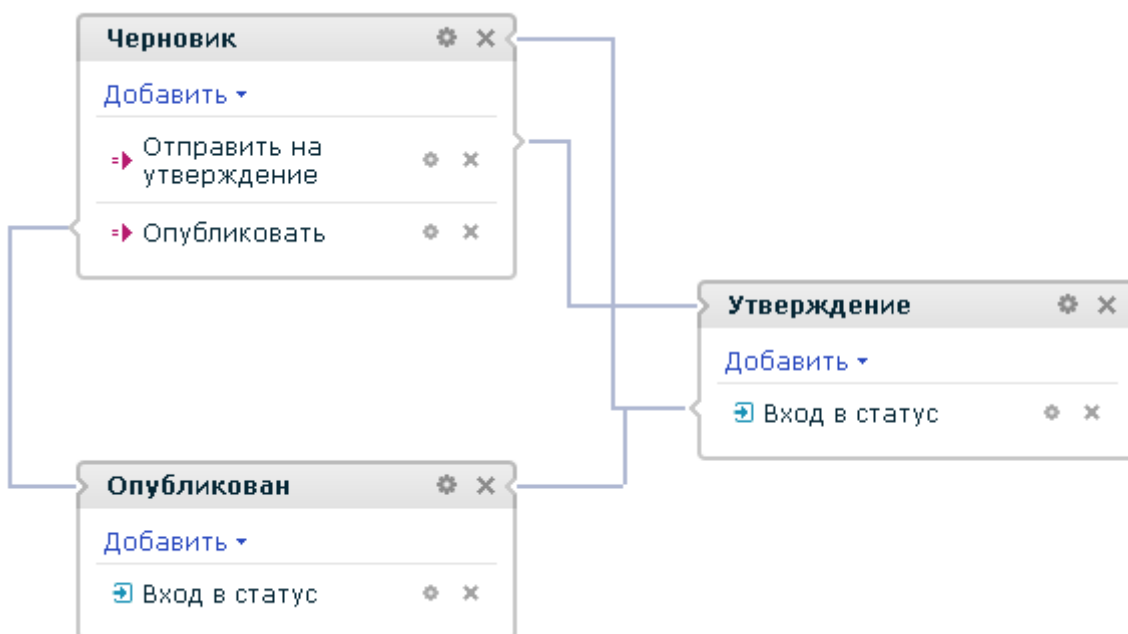


Рис. 2.4 Пример бизнес-процесса со статусами

При переходе бизнес-процесса в любой из статусов выполняется последовательный подпроцесс для инициализации этого статуса. После чего бизнес-процесс становится в ожидание одного из определенных в данном статусе событий. Список возможных событий и их названия могут быть получены для отображения в пользовательском интерфейсе. На каждое событие может быть получен список групп пользователей, которые имеют право отправлять данное событие.

При возникновении события выполняется последовательный подпроцесс, соответствующий принятому событию (связанный с ним).



Среди действий подпроцесса может быть действие по установке нового статуса. В этом случае бизнес-процесс переводится в новый статус. Если такого действия нет, то текущий статус считается конечным и бизнес-процесс завершается.

Если для данного статуса определен последовательный подпроцесс для финализации этого статуса, то он выполняется непосредственно перед выходом из данного статуса (перед переходом в другой статус).

Выбор типа бизнес-процесса

Практически любая возникающая задача может быть решена как с помощью последовательного бизнес-процесса, так и с помощью бизнес-процесса со статусами. Но выбор не соответствующего типа бизнес-процесса существенно усложнит шаблон бизнес-процесса.

Тип **последовательный бизнес-процесс** выбирается в том случае, если просто необходимо выполнить определенную последовательность действий.

Тип **бизнес-процесс со статусами** выбирается в том случае, если бизнес-процесс (документ) может находиться в разных состояниях, переходы между которыми осуществляются по определенным правилам. При этом появляется возможность автоматически контролировать права на доступ к документу в разных состояниях и на выполнение переходов между состояниями. Для каждого состояния может быть задан набор событий, при возникновении которых выполняются соответствующие подпроцессы.



Глава 3. Действия

Все, что происходит в бизнес-процессе — это действия. Сам бизнес-процесс представляет собой составное действие, которое позволяет определять внутри себя дочерние действия. Каждое действие в рамках бизнес-процесса должно иметь уникальное имя.

Действие — это класс, который наследуется от абстрактного класса **CBPActivity** или его потомков. Название класса должно начинаться с подстроки "CBP" и может состоять из латинских букв и цифр.

```
<?
if (!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true)die();

class CBPMyActivity1
    extends CBPActivity
{
    ...
}
?>
```

Непосредственно от класса **CBPActivity** наследуются действия, которые не могут содержать внутри себя другие действия. Этот класс определяет набор базовых методов, которые необходимы любому действию. Некоторые методы, определенные в классе **CBPActivity** могут или должны быть переопределены в классе-наследнике.

Методы класса CBPActivity

Класс **CBPActivity** содержит следующие методы:

Execute

Метод **Execute** по умолчанию ничего не делает и должен быть переопределен в классе-наследнике:

```
public function Execute()
```

Этот метод вызывается исполняющей средой при выполнении действия. Он непосредственно реализует поведение действия и должен быть переопределен в каждом действии. Метод **Execute** может содержать произвольный код.

Метод **Execute** должен вернуть статус выполнения действия, который используется исполняющей средой. Статус используется, чтобы определить, как обстоят дела с данным действием:

- завершилось ли действие успешно,
- все еще продолжается.



Возможные значения статуса выполнения действия описаны в классе **CBPActivityExecutionStatus**:

- **CBPActivityExecutionStatus::Closed** - действие завершило свою работу и может быть закрыто;
- **CBPActivityExecutionStatus::Executing** - у действия еще есть работа, которую нужно доделать. Например, составное действие должно выполнить свои дочерние элементы. В этом случае составное действие может запланировать запуск каждого своего дочернего действия и ожидать завершения их работы прежде, чем известить исполняющую среду о том, что данное действие завершилось.

На изменение статуса выполнения действия можно подписываться. Например, составное действие может узнать о завершении содержащегося в нем дочернего действия, подписавшись на изменение его статуса выполнения в **CBPActivityExecutionStatus::Closed**.

```
public function Execute()  
{  
    $this->WriteToTrackingService("Test record: ". $this->MyProperty);  
    ...  
    return CBPActivityExecutionStatus::Closed;  
}
```

Initialize

Метод **Initialize** по умолчанию ничего не делает и может быть переопределен в классе-наследнике:

```
public function Initialize()
```

Этот метод вызывается исполняющей средой для инициализации действия, которое происходит до запуска бизнес-процесса на выполнение. Большинство действий не переопределяют этот метод.

HandleFault

Метод **HandleFault** по умолчанию ничего не делает и может быть переопределен в классе-наследнике:

```
public function HandleFault(Exception $exception)
```

Этот метод вызывается исполняющей средой при возникновении ошибки выполнения действия. Метод может быть переопределен, если при возникновении ошибки выполнения действия необходимо выполнить какой-либо код.

Класс **CBPActivity** содержит набор уже реализованных методов, которые можно применять в действиях-наследниках:



CBPActivity GetRootActivity()

Метод возвращает корневое действие бизнес-процесса. Корневое действие реализует интерфейс **IBPRootActivity**. Например, если необходимо узнать код документа, для которого запущен бизнес-процесс, то можно выполнить следующий код:

```
$rootActivity = $this->GetRootActivity();  
$documentId = $rootActivity->GetDocumentId();
```

GetName

```
string GetName()
```

Метод возвращает имя действия. Имя действия уникально в рамках бизнес-процесса.

GetWorkflowInstancelId

```
string GetWorkflowInstancelId()
```

Метод возвращает код бизнес-процесса.

WriteToTrackingService

```
void WriteToTrackingService($message = "", $modifiedBy = 0)
```

Метод записывает в лог произвольное сообщение. Кроме того может быть записано, от имени какого пользователя отправляется это сообщение.

AddStatusChangeHandler

```
void AddStatusChangeHandler($event, $eventHandler)
```

Метод добавляет новый обработчик события изменения статуса действия. Первым параметром метод принимает одну из констант **CBPActivity::ExecutingEvent**, **CBPActivity::ClosedEvent**, **CBPActivity::FaultingEvent**. Вторым параметром метод принимает обработчик события, который реализует интерфейс **IBPActivityEventListener**. Например, действие может подписаться на завершение одного из своих дочерних действий **\$a** с помощью кода

```
$a->AddStatusChangeHandler(self::ClosedEvent, $this);
```

RemoveStatusChangeHandler

```
void RemoveStatusChangeHandler($event, $eventHandler)
```

Метод удаляет обработчик события изменения статуса действия. Параметры аналогичны параметрам метода **AddStatusChangeHandler**.



Члены класса CBPActivity

Класс **CBPActivity** содержит следующие члены, которые можно применять в действиях-наследниках:

- **workflow** – содержит объект-оболочку типа **CBPWorkflow** для данного бизнес-процесса,
- **parent** – содержит родительское действие,
- **executionStatus** – статус выполнения действия,
- **executionResult** – результат выполнения действия.

Пример действия

Например, необходимо действие, которое создаст файл с указанным в свойствах действия именем. Это можно реализовать следующим образом:

```
class CBPMyActivity
    extends CBPActivity
{
    public function __construct($name)
    {
        parent::__construct($name);
        // Определим свойство FileName, в котором будет
        // содержаться имя файла
        $this->arProperties = array("Title" => "", "FileName" => "");
    }

    // Исполняемый метод действия
    public function Execute()
    {
        // Если свойство с именем файла задано, осуществим в него запись
        // Обратите внимание, что для упрощения кода здесь не добавлены
        // необходимые проверки безопасности
        if (strlen($this->FileName) > 0)
        {
            $f = fopen($this->FileName, "w");
            fwrite($f, "Какой-то текст");
            fclose($f);
        }
        // Вернем указание исполняющей среде, что действие завершено
        return CBPActivityExecutionStatus::Closed;
    }
}
```

Составные действия

Составные действия наследуются от абстрактного класса **CBPCompositeActivity**, который в свою очередь наследуется от класса **CBPActivity**. Класс **CBPCompositeActivity** обеспечивает поддержку возможности включать внутрь действия



дочерние действия. Например, составным действием является стандартное действие **CBPParallelActivity** (параллельное выполнение), которое содержит в себе дочерние действия, соответствующие веткам параллельного выполнения.

Класс **CBPCompositeActivity** содержит член **arActivities**, с помощью которого можно обращаться к дочерним действиям.

Например, при запуске действия необходимо запустить первое дочернее действие и дождаться его завершения. Для этого можно использовать следующий код:

```
class CBPMyActivity
    extends CBPCompositeActivity // наследуем, так как составное действие
    implements IBPEventActivity // обработка события завершения дочернего //действия
{
    // Исполняемый метод действия
    public function Execute()
    {
        // Возьмем первое дочернее действие
        $activity = $this->arActivities[0];
        // Подпишемся на событие изменения статуса дочернего действия
        // (завершение)
        $activity->AddStatusChangeHandler(self::ClosedEvent, $this);
        // Отправим дочернее действие исполняющей среде на выполнение
        $this->workflow->ExecuteActivity($activity);
        // Вернем указание исполняющей среде, что действие еще выполняется
        return CBPActivityExecutionStatus::Executing;
    }

    // Обработчик события изменения статуса интерфейса IBPEventActivity
    // Параметром передается действие, изменившее статус
    protected function OnEvent(CBPActivity $sender)
    {
        // Отпишемся от события изменения статуса дочернего действия
        // (завершения)
        $sender->RemoveStatusChangeHandler(self::ClosedEvent, $this);
        // Дочернее действие завершено, выполняем другой необходимый нам код
        // Например завершаем действие
        $this->workflow->CloseActivity($this);
    }
}
```

Свойства действий

Действие может иметь свойства, значения которых настраиваются при добавлении действия в шаблон бизнес-процесса. Значениями свойств могут быть как константы, так и ссылки на свойства других действий бизнес-процесса.

Свойства действия описываются в конструкторе класса действия определением массива в члене класса **arProperties**:



```
public function __construct($name)
{
    parent::__construct($name);
    $this->arProperties = array("Title" => "", "MyProperty" => "");
}
```

Ключами в массиве определения свойств являются названия свойств, а значениями – значения по умолчанию.

При выполнении действия свойства доступны как члены класса:

```
$this->MyProperty
```

Входные параметры (свойства) бизнес-процесса доступны как свойства корневого действия бизнес-процесса. Любое действие бизнес-процесса может обратиться к входным параметрам бизнес-процесса.

Например, если бизнес-процесс был запущен с помощью кода:

```
// Код шаблона бизнес-процесса
$workflowTemplateId = 12;

// Бизнес-процесс запускается для документа - элемента инфоблока с кодом 358
$documentId = array("iblock", "CIBlockDocument", 358);

// Входные параметры бизнес-процесса
$arParameters = array("MyProperty" => "Красный");

$runtime = CBPRuntime::GetRuntime();
$wi = $runtime->CreateWorkflow($workflowTemplateId, $documentId, $arParameters);
$wi->Start();
```

то в любом действии этого бизнес-процесса значение параметра можно будет получить с помощью кода:

```
$rootActivity = $this->GetRootActivity();
if ($rootActivity->IsPropertyExists("MyProperty"))
    $val = $rootActivity->MyProperty;
// $val == "Красный"
```

Свойства действий описываются разработчиком при написании кода действия. Входные параметры бизнес-процесса (они же свойства корневого действия бизнес-процесса) описываются пользователем при создании шаблона бизнес-процесса.

Значениями свойств могут быть как константы, так и ссылки на свойства других действий бизнес-процесса при условии, что эти действия выполнялись раньше.

Чтобы в момент выполнения бизнес-процесса значением свойства одного действия являлось значение свойства выполненного выше другого действия, необходимо при



создании шаблона бизнес-процесса в качестве значения свойства первого действия задать массив вида

```
array("название действия, на свойство которого ссылаются", "название свойства")
```

В случае простых типов данных свойств в качестве значения свойства первого действия можно задать строку вида:

```
"{=название_действия, название_свойства}"
```

Чтобы сослаться на входной параметр (свойство) бизнес-процесса, которое доступно как свойство корневого действия бизнес-процесса, следует в качестве названия действия использовать слово **Template**:

```
array("Template", "название свойства")
"{=Template, название_свойства}"
```

Вообще в качестве названия действия можно использовать слова

- **Document** – для обращения к произвольному полю документа, над которым запущен бизнес-процесс;
- **Template** – для обращения к входному параметру (свойству) бизнес-процесса (корневого действия);
- **Variable** – для обращения к переменной бизнес-процесса;
- **User** – для получения кода текущего пользователя (в качестве названия свойства должно быть указано "ID");
- **System** – обращение к системным переменным, в настоящее время доступно только свойство **Now** – текущая дата в формате сайта;
- любое другое имя – обращение к свойству действия с этим именем.

Например, если во время разработки в качестве значения свойства установить строку:

```
"Документ [url={=Template:PathTemplate}]{=Document:NAME}[/url] был одобрен"
```

и при этом рабочий поток запустится с входящим параметром **PathTemplate** равным "file.php" над документом с названием "План счетов", то при выполнении действия значением свойства будет строка

```
"Документ [url=file.php]План счетов[/url] был одобрен"
```

Основные стандартные действия

<i>CBPActivity</i>	Абстрактный базовый класс всех действий.
<i>CBPCompositeActivity</i>	Абстрактный базовый класс составных действий, т.е. действий, которые могут содержать в себе дочерние действия.
<i>CBPCodeActivity</i>	Запускает на выполнение произвольный PHP-код.



<i>CBPSetVariableActivity</i>	Устанавливает значения переменных бизнес-процесса.
<i>CBPDelayActivity</i>	Реализует ожидание, откладывая выполнение на определенный срок.
<i>CBPHandleExternalEventActivity</i>	Реализует действие, которое ожидает внешнее событие. Бизнес-процесс останавливается до получения данного внешнего события.
<i>CBPEventDrivenActivity</i>	Служит контейнером для действий, выполнение которых происходит по событию.
<i>CBPIfElseActivity</i>	Реализует функционал условия.
<i>CBPIfElseBranchActivity</i>	Реализует функционал ветки условия.
<i>CBPWhileActivity</i>	Реализует функционал цикла.
<i>CBPListenActivity</i>	Реализует ожидание одного из нескольких возможных событий. Когда одно из событий происходит, остальные перестают ожидать событий и отменяются.
<i>CBPParallelActivity</i>	Параллельно запускает набор дочерних действий.
<i>CBPSequenceActivity</i>	Последовательно запускает набор дочерних действий.
<i>CBPSequentialWorkflowActivity</i>	Представляет собой бизнес-процесс, который выполняет действия последовательно (последовательный бизнес-процесс).
<i>CBPSetStateActivity</i>	Устанавливает статус в бизнес-процессе со статусами.
<i>CBPStateActivity</i>	Представляет собой статус в бизнес-процессе со статусами.
<i>CBPStateMachineWorkflowActivity</i>	Представляет собой бизнес-процесс со статусами.

Любое действие наследуется от базового класса действий или одного из его наследников.

Интерфейсы действий

Бизнес-процесс определяет набор интерфейсов, которые расширяют функциональность действий.

interface IBPActivityEventListener

```
interface IBPActivityEventListener
{
    public function OnEvent(CBPActivity $sender, $arEventParameters = array());
}
```



Действия, реализующие интерфейс **IBPActivityEventListener** могут обрабатывать внутренние по отношению к бизнес-процессу события изменения статуса других действий. Для реализации интерфейса необходимо реализовать метод обработки события изменения статуса действия. В параметрах метода передаются действие, которое меняет свой статус, и параметры события, если таковые есть.

Например, создадим действие, которое может содержать в себе дочерние действия и запускать их одно за другим по очереди. Для простоты опустим обработку ошибок, вопросы безопасности и «защиты от дурака».

```
class CBPMyActivity2
    extends CBPCompositeActivity
    implements IBPActivityEventListener
{
    public function Execute()
    {
        // Если дочерних действий нет, то просто завершаем работу нашего
        // действия
        if (count($this->arActivities) == 0)
            return CBPActivityExecutionStatus::Closed;

        // Подписываемся на событие завершения первого
        // по порядку дочернего действия
        $this->arActivities[0]->AddStatusChangeHandler(self::ClosedEvent, $this);

        // Отправляем на запуск первое по порядку дочернее действие
        $this->workflow->ExecuteActivity($this->arActivities[0]);

        // Говорим исполняющей среде, что наше действие еще выполняется
        return CBPActivityExecutionStatus::Executing;
    }

    // Реализация интерфейса IBPActivityEventListener. Метод будет
    // вызван при возникновении события на завершение дочернего
    // действия, на которое мы подписались.
    public function OnEvent(CBPActivity $sender, $arEventParameters = array())
    {
        // Удаляем обработчик события на завершение дочернего действия,
        // которое изменило статус
        $sender->RemoveStatusChangeHandler(self::ClosedEvent, $this);

        // Если не удалось запустить очередное дочернее действие, то
        // завершаем наше действие
        if (!$this->ExecuteNextChild())
            $this->workflow->CloseActivity($this);
    }

    private function ExecuteNextChild ()
    {
        // Найдем первое не завершенное дочернее действие, отправим его на
```



```
//запуск и вернем true. Если не завершенных дочерних действий больше
//нет, то вернем false.
$num = 0;
for ($i = count($this->arActivities) - 1; $i >= 0; $i--)
{
    if ($this->arActivities[$i]->executionStatus
        CBPActivityExecutionStatus::Closed)
    {
        if ($i == (count($this->arActivities) - 1))
            return false;

        $num = $i + 1;
        break;
    }
}

$this->arActivities[$num]->AddStatusChangeHandler(self::ClosedEvent, $this);
$this->workflow->ExecuteActivity($this->arActivities[$num]);
return true;
}
```

Данный функционал реализуется в стандартном действии **CBPSequenceActivity**.

interface IBPEventActivity

```
interface IBPEventActivity
{
    public function Subscribe(IBPActivityExternalEventListener $eventHandler);
    public function Unsubscribe(IBPActivityExternalEventListener $eventHandler);
}
```

Действия, реализующие интерфейс **IBPEventActivity** имеют функционал подписки на внешние по отношению к бизнес-процессу события. Для реализации интерфейса необходимо реализовать методы подписки и отписки. В параметрах методов передается обработчик, реализующий интерфейс **IBPActivityExternalEventListener**. Этот обработчик выполнится при возникновении события.

interface IBPActivityExternalEventListener

```
interface IBPActivityExternalEventListener
{
    public function OnExternalEvent($arEventParameters = array());
}
```

Интерфейс **IBPActivityExternalEventListener** реализуют классы, которые являются обработчиками внешних по отношению к бизнес-процессу событий. Для реализации интерфейса необходимо реализовать метод обработки внешнего события. В параметре метода передаются параметры события, если таковые есть.



Например, создадим действие голосования списка заданных пользователей. Для простоты опустим обработку ошибок, вопросы безопасности и «защиты от дурака».

```
class CBPMyActivity3
    extends CBPActivity
    implements IBPEventActivity, IBPActivityExternalEventListener
{
    private $taskId = 0;
    private $arMyActivityResults = array();

    public function __construct($name)
    {
        parent::__construct($name);

        // Определяем свойство действия, содержащее массив кодов
        // голосующих пользователей
        $this->arProperties = array("Title" => "", "Users" => null);
    }

    public function Execute()
    {
        // Подписываемся на событие голосования
        $this->Subscribe($this);

        // Сообщаем исполняющей среде, что действие еще выполняется
        return CBPActivityExecutionStatus::Executing;
    }

    public function Subscribe(IBPActivityExternalEventListener $eventHandler)
    {
        // Создаем новое задание для пользователей из свойства $this->Users
        $taskService = $this->workflow->GetService("TaskService");
        $this->taskId = $taskService->CreateTask(
            array(
                "USERS" => $this->Users,
                "WORKFLOW_ID" => $this->GetWorkflowInstanceId(),
                "ACTIVITY" => "MyActivity",
                "ACTIVITY_NAME" => $this->name,
                "NAME" => "Голосуем за документ",
                "DESCRIPTION" => "",
                "PARAMETERS" => array()
            )
        );

        // Подписываемся на внешнее событие - задание с голосованием
        $this->workflow->AddEventHandler($this->name, $eventHandler);
    }

    public function Unsubscribe(IBPActivityExternalEventListener $eventHandler)
    {

```



```
// Удаляем задание и отписываемся от внешнего
// события - задания с голосованием
$taskService = $this->workflow->GetService("TaskService");
$taskService->DeleteTask($this->taskId);

$this->workflow->RemoveEventHandler($this->name, $eventHandler);

$this->taskId = 0;
}

// Метод вызывается, когда действию приходит внешнее событие
public function OnExternalEvent($arEventParameters = array())
{
    // Метод запускается на выполнение, если произошло
    // внешнее событие - голосование по заданию
    if ($this->executionStatus == CBPActivityExecutionStatus::Closed)
        return;

    $taskService = $this->workflow->GetService("TaskService");
    $taskService->MarkCompleted($this->taskId, $arEventParameters["USER_ID"]);

    // Запоминаем результаты голосования
    $this->arMyActivityResults[$arEventParameters["USER_ID"]] =
    $arEventParameters["VOTE"];

    // Проверяем, все ли пользователи проголосовали
    $allUsersVoted = true;
    foreach ($this->Users as $u)
    {
        if (!isset($this->arMyActivityResults[$u]))
        {
            $allUsersVoted = false;
            break;
        }
    }

    if ($allUsersVoted)
    {
        // Все пользователи проголосовали - можно отписаться
        // от события, обработать результаты и завершить действие
        $this->Unsubscribe($this);

        // Здесь мы должны как-то обработать результаты
        // голосования $this->arMyActivityResults

        $this->workflow->CloseActivity($this);
    }
}

// Метод, который рисует форму задания
```



```
public static function ShowTaskForm($arTask, $userId, $userName = "")
{
    $form =
        '<tr><td valign="top" width="40%" align="right">Голосование:</td>'.
        '<td valign="top" width="60%">'.
        '<select name="vote"><option value="1">1</option>'.
        '<option value="2">2</option><option value="3">3</option></select>'.
        '</td></tr>';

    $buttons =
        '<input type="submit" name="do_vote" value="Голосовать"/>';

    return array($form, $buttons);
}

// Метод, который обрабатывает форму задания
public static function PostTaskForm($arTask, $userId, $arRequest, &$arErrors,
    $userName = "")
{
    $arErrors = array();

    try
    {
        $userId = intval($userId);

        // Собираем массив параметров события. Этот массив будет
        // доступен в качестве параметра в обработчике
        // события OnExternalEvent
        $arEventParameters = array(
            "USER_ID" => $userId,
            "USER_NAME" => $userName,
            "VOTE" => $arRequest["vote"],
        );

        // Отправляем внешнее событие указанному действию
        // указанного бизнес-процесса
        CBPRuntime::SendExternalEvent(
            $arTask["WORKFLOW_ID"],
            $arTask["ACTIVITY_NAME"],
            $arEventParameters
        );

        return true;
    }
    catch (Exception $e)
    {
        $arErrors[] = array(
            "code" => $e->getCode(),
            "message" => $e->getMessage(),
            "file" => $e->getFile(). " [" . $e->getLine(). "]",
        );
    }
}
```



```
        );  
    }  
  
    return false;  
}  
}
```

Создание собственных действий

Пользовательские действия создаются в папке */bitrix/activities/custom* относительно корня сайта. Каждое действие располагается в отдельной папке. Название папки действия должно совпадать с именем класса действия, но без первых символов "CBP". Кроме того имя папки должно быть записано строчными буквами (в нижнем регистре).

В папке действия должен располагаться файл класса действия. Название файла класса действия должно совпадать с названием папки действия и иметь расширение **php**. Кроме того в папке действия могут располагаться другие необходимые действию файлы. Например, файл с описанием действия, файлы с локализацией действия, изображения, файлы с ресурсами и т.п.

Файл с описанием действия располагается в папке действия и имеет имя **.description.php**. В этом файле содержится описание действия, которое необходимо для корректной работы системы. В файле описания действия должен содержаться код типа:

```
<?  
if (!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true) die();  
  
$arResultDescription = array(  
    "NAME" => GetMessage("MYACTIVITY_DESCR_NAME"),  
    "DESCRIPTION" => GetMessage("MYACTIVITY_DESCR_DESCR"),  
    "TYPE" => "activity",  
    "CLASS" => "MyActivity",  
    "JSCLASS" => "BizProcActivity",  
    "CATEGORY" => array(  
        "ID" => "other",  
    ),  
);  
?>
```

Здесь определен тип действия в элементе "TYPE", который имеет два возможных значения: "activity" для действий и "condition" для условий. Кроме того задаются название и описание действия, Java-скриптовый класс для отрисовки в визуальном редакторе, категория и т.п.

В подпапке **lang** папки действия располагаются файлы с локализацией фраз действия на различные языки.

Файл с классом действия имеет вид типа



```
<?
if (!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true) die();

class CBPMyActivity
    extends CBPActivity
{
    public function __construct($name)
    {
        parent::__construct($name);
        // Определим свойство действия MyText
        // Оно может быть задано в визуальном редакторе при
        // помещении действия в шаблон бизнес-процесса
        $this->arProperties = array("Title" => "", "MyText" => "");
    }

    // Исполняющийся метод действия
    public function Execute()
    {
        // Суть действия – запись значения свойства в файл
        if (strlen($this->MyText) > 0)
        {
            $f = fopen($_SERVER["DOCUMENT_ROOT"]."/dump.txt", "a");
            fwrite($f, $this->MyText);
            fclose($f);
        }

        // Возвратим исполняющей системе указание, что действие завершено
        return CBPActivityExecutionStatus::Closed;
    }

    // Статический метод возвращает HTML-код диалога настройки
    // свойств действия в визуальном редакторе. Если действие не имеет
    // свойств, то этот метод не нужен
    public static function GetPropertiesDialog($documentType, $activityName,
        $arWorkflowTemplate, $arWorkflowParameters, $arWorkflowVariables,
        $arCurrentValues = null, $formName = "")
    {
        $runtime = CBPRuntime::GetRuntime();

        if (!is_array($arWorkflowParameters))
            $arWorkflowParameters = array();
        if (!is_array($arWorkflowVariables))
            $arWorkflowVariables = array();

        // Если диалог открывается первый раз, то подгружаем значение
        // свойства, которое было сохранено в шаблоне бизнес-процесса
        if (!is_array($arCurrentValues))
        {
            $arCurrentValues = array("my_text" => "");
        }
    }
}
```



```
        $arCurrentActivity= &CBPWorkflowTemplateLoader::FindActivityByName(
            $arWorkflowTemplate,
            $activityName
        );
        if (is_array($arCurrentActivity["Properties"]))
            $arCurrentValues["my_text "] =
$arCurrentActivity["Properties"]["MyText"];
    }

    // Код, формирующий диалог, расположен в отдельном файле
    // properties_dialog.php в папке действия.
    // Возвращаем этот код.
    return $runtime->ExecuteResourceFile(
        __FILE__,
        "properties_dialog.php",
        array(
            "arCurrentValues" => $arCurrentValues,
            "formName" => $formName,
        )
    );
}

// Статический метод получает введенные в диалоге настройки свойств
// значения и сохраняет их в шаблоне бизнес-процесса. Если действие не
// имеет свойств, то этот метод не нужен.
public static function GetPropertiesDialogValues($documentType, $activityName,
    &$arWorkflowTemplate, &$arWorkflowParameters, &$arWorkflowVariables,
    $arCurrentValues, &$arErrors)
{
    $arErrors = array();

    $runtime = CBPRuntime::GetRuntime();

    if (strlen($arCurrentValues["my_text "] <= 0)
    {
        $arErrors[] = array(
            "code" => "emptyCode",
            "message" => GetMessage("MYACTIVITY_EMPTY_TEXT"),
        );
        return false;
    }

    $arProperties = array("MyText" => $arCurrentValues["my_text "]);


    $arCurrentActivity = &CBPWorkflowTemplateLoader::FindActivityByName(
        $arWorkflowTemplate,
        $activityName
    );
    $arCurrentActivity["Properties"] = $arProperties;
```



```
        return true;
    }
}
?>
```

Код в файле **properties_dialog.php**, формирующий диалог настройки свойств действия в визуальном редакторе, может выглядеть примерно так:

```
<?
if (!defined("B_PROLOG_INCLUDED") || B_PROLOG_INCLUDED!==true)die();
?>
<tr>
    <td align="right" width="40%"><span style="color:#FF0000;">*</span> <?=
    GetMessage("MYACTIVITY_PD_TEXT") ?>:</td>
    <td width="60%">
        <textarea name="my_text" id="id_my_text " rows="5" cols="40"><?=
        htmlspecialchars($arResult["my_text"]) ?></textarea>
        <input type="button" value="..." onclick="BPAShowSelector('id_my_text', 'string');">
    </td>
</tr>
```

Пользователь может ввести в поле **my_text** явное значение или выбрать одно из значений с помощью диалога, открывающегося по кнопке . Во втором случае пользователь может установить, что значением свойства будет являться значение свойства корневого действия, которое задается как входящий параметр при запуске бизнес-процесса.



Глава 4. Исполняющая среда бизнес-процессов

Исполняющая среда, реализованная в классе **CBPRuntime**, управляет бизнес-процессами. Она позволяет создавать и запускать бизнес-процессы, отправлять им события. Кроме того исполняющая среда управляет сервисами рабочей среды.

Исполняющая среда в рамках одного хита существует в единственном экземпляре. Получить экземпляр исполняющей среды можно с помощью кода

```
$runtime = CBPRuntime::GetRuntime();
```

Основные методы исполняющей среды:

CBPWorkflow

```
CBPWorkflow CreateWorkflow($workflowTemplateId, $documentId, $workflowParameters = array())
```

Создает новый бизнес-процесс на основании шаблона бизнес-процесса, заданного его кодом. Возвращается вновь созданный бизнес-процесс.

CBPWorkflow

```
CBPWorkflow GetWorkflow($workflowId)
```

Возвращает существующий бизнес-процесс по его идентификатору.

CBPRuntimeService

```
CBPRuntimeService GetService($name)
```

Возвращает экземпляр доступного сервиса по его названию.

SendExternalEvent

```
static void SendExternalEvent($workflowId, $eventName, $arEventParameters = array())
```

Статический метод отправляет указанное внешнее событие бизнес-процессу, заданному его идентификатором.

Статический класс **CBPDocument** содержит статические методы-оболочки, упрощающие использование исполняющей среды. Например, для того, чтобы запустить бизнес-процесс по коду его шаблона, можно использовать код:

```
$runtime = CBPRuntime::GetRuntime();  
$wi = $runtime->CreateWorkflow($workflowTemplateId, $documentId, $arParameters);  
$wi->Start();
```

Или можно использовать метод:

```
string CBPDocument::StartWorkflow($workflowTemplateId, $documentId, $arParameters, &$arErrors)
```



который кроме того обработает исключения, собрав их в массив `$arErrors`, и вернет идентификатор бизнес-процесса.

Объект бизнес-процесса

При создании бизнес-процесса создается объект-оболочка типа **CBPWorkflow**, который управляет бизнес-процессом, умеет отправлять на запуск его действия, транслировать и обрабатывать события.

Для каждого бизнес-процесса существует свой собственный объект-оболочка.

Получить объект-оболочку можно с помощью методов **CreateWorkflow** и **GetWorkflow** исполняющей среды:

```
$runtime = CBPRuntime::GetRuntime();  
$wi = $runtime->CreateWorkflow($workflowTemplateId, $documentId, $arParameters);
```

В коде действия объект-оболочка для бизнес-процесса, в который входит это действие, доступна через переменную-член **workflow**:

```
$this->workflow->ExecuteActivity($activity);
```

Основные методы объекта-оболочки

CBPRuntimeService

```
CBPRuntimeService GetService($name)
```

Возвращает экземпляр доступного сервиса по его названию. Этот метод сводится к вызову соответствующего метода исполняющей среды.

AddEventHandler

```
void AddEventHandler($eventName, IBPActivityExternalEventListener $eventHandler)
```

Добавляет обработчик внешнего события.

RemoveEventHandler

```
void RemoveEventHandler($eventName, IBPActivityExternalEventListener $eventHandler)
```

Удаляет обработчик внешнего события.

CloseActivity

```
void CloseActivity(CBPActivity $activity, $arEventParameters = array())
```

Завершает указанное действие.



ExecuteActivity

```
void ExecuteActivity(CBPActivity $activity, $arEventParameters = array())
```

Отправляет действие на выполнение.

GetInstanceld

```
string GetInstanceld()
```

Возвращает идентификатор бизнес-процесса.

Start

```
void Start()
```

Запускает бизнес-процесс на выполнение.

Terminate

```
void Terminate(Exception $e = null)
```

Прерывает выполнение бизнес-процесса.

Например, создадим составное действие, которое запустит параллельно все свои дочерние действия, дождется окончания их выполнения и завершится:

```
class CBPMyActivity4
    extends CBPCompositeActivity
    implements IBPActivityEventListener
{
    // Метод, который запускается при запуске действия
    public function Execute()
    {
        // Если дочерних действий нет, то просто завершим наше действие
        if (count($this->arActivities) == 0)
            return CBPActivityExecutionStatus::Closed;

        // Подпишемся на события завершения всех дочерних
        // действий и отправим эти дочерние действия на выполнение
        for ($i = 0; $i < count($this->arActivities); $i++)
        {
            $activity = $this->arActivities[$i];
            $activity->AddStatusChangeHandler(self::ClosedEvent, $this);
            $this->workflow->ExecuteActivity($activity);
        }

        // Говорим исполняющей среде, что наше действие еще работает
        return CBPActivityExecutionStatus::Executing;
    }
}
```



```
// Метод – обработчик события
public function OnEvent(CBPActivity $sender, $arEventParameters = array())
{
    // Отпишемся от события на завершение
    // дочернего действия $sender
    $sender->RemoveStatusChangeHandler(self::ClosedEvent, $this);

    // Проверим, есть ли дочерние действия, которые еще не завершены
    // то есть статус выполнения которых не Closed
    $flag = true;
    for ($i = 0; $i < count($this->arActivities); $i++)
    {
        $activity = $this->arActivities[$i];
        if (($activity->executionStatus != CBPActivityExecutionStatus::Initialized)
            && ($activity->executionStatus !=
                CBPActivityExecutionStatus::Closed))
        {
            $flag = false;
            break;
        }
    }

    // Если незавершенных дочерних действий нет, то завершаем наше
    // действие
    if ($flag)
        $this->workflow->CloseActivity($this);
}
}
```

Сервисы исполняющей среды бизнес-процессов

Сервис исполняющей среды — это класс, экземпляр которого создается при запуске исполняющей среды. Запущенный экземпляр предоставляет действиям бизнес-процесса какой-либо функционал. Например, сервис **CBPDocumentService** дает действиям возможность вызывать методы документа, а **CBPHistoryService** дает возможность сохранять документ в истории.

Любой сервис представляет собой класс, который наследуется от абстрактного класса **CBPRuntimeService**. При запуске исполняющей среды (явном или не явном – при первом вызове какого-либо метода исполняющей среды) создаются экземпляры всех классов сервисов. В дальнейшем эти экземпляры можно получить по их имени с помощью метода **GetService** исполняющей среды или объекта-оболочки бизнес-процесса.

Набор стандартных сервисов включает в себя:

- Сервис с именем **SchedulerService**, являющийся экземпляром класса **CBPSchedulerService**. Дает возможность подписываться на внешнее событие – истечение периода времени (работает с помощью функционала агентов).



- Сервис с именем **StateService**, являющийся экземпляром класса **CBPStateService** Служит для работы со статусами бизнес-процесса (документа).
- Сервис с именем **TrackingService**, являющийся экземпляром класса **CBPTrackingService**. Дает возможность записывать информацию в лог. В лог записываются системные события времени выполнения бизнес-процесса. Например, какие действия и с каким результатом были выполнены. Кроме того в лог может записываться любая произвольная информация.
- Сервис с именем **TaskService**, являющийся экземпляром класса **CBPTaskService**. Дает возможность работать с заданиями для пользователей.
- Сервис с именем **HistoryService**, являющийся экземпляром класса **CBPHistoryService**. Дает возможность сохранять документ в историю и восстанавливать его из истории.
- Сервис с именем **DocumentService**, являющийся экземпляром класса **CBPDocumentService**. Дает возможность вызывать методы документа.

Документ и тип документа

Тип документа и документ – абстрактные для бизнес-процессов объекты, физический смысл которых в рамках бизнес-процессов не определен. Тип документа есть некоторое объединение документов. Тип документа и документ могут не иметь физического представления, т.е. быть виртуальными.

Тип документа и документ определяются своими идентификаторами, которые имеют вид кортежа из трех элементов: кода модуля, имени класса документа и некоторого кода (как правило, кода элемента).

Например,

```
array("iblock","CIBlockDocument","458")
```

где:

- *iblock* – код модуля инфоблоков,
- *CIBlockDocument* – имя класса документа,
- *458* – ID элемента инфоблока.

Шаблон бизнес-процесса привязан к типу документа. При связывании устанавливается, будет ли бизнес-процесс запускаться автоматически на создание нового документа.

Бизнес-процесс всегда выполняется над определенным документом. На один и тот же документ может быть одновременно запущено произвольное число бизнес-процессов.

Класс документа должен реализовывать методы интерфейса **IBPWorkflowDocument**. Этот интерфейс содержит методы, которые необходимы бизнес-процессу для работы с документом.



Методы интерфейса IBPWorkflowDocument

Интерфейс **IBPWorkflowDocument** содержит методы:

GetDocument

```
public function GetDocument($documentId)
```

Метод возвращает свойства (поля) документа в виде ассоциативного массива вида

```
array(
    код_свойства => значение,
    ...
)
```

Определены все свойства, которые возвращает метод **GetDocumentFields**. Параметры метода:

<code>string \$documentId</code>	код документа
----------------------------------	---------------

GetDocumentFields

```
public function GetDocumentFields($documentType)
```

Метод возвращает массив свойств (полей), которые имеет документ данного типа. Метод **GetDocument** возвращает значения свойств для заданного документа. Возвращаемый массив имеет вид

```
array(
    код_свойства => array(
        "NAME" => название_свойства,
        "TYPE" => тип_свойства
    ),
    ...
)
```

Параметры метода:

<code>string \$documentType</code>	тип документа
------------------------------------	---------------

CreateDocument

```
public function CreateDocument($arFields)
```

Метод создает новый документ с указанными свойствами (полями) и возвращает его код. Параметры метода:

<code>array \$arFields</code>	массив значений свойств документа в виде:
-------------------------------	---



	<pre>array(код_свойства => значение, ...)</pre>
--	--

Коды свойств соответствуют кодам свойств, возвращаемым методом **GetDocumentFields**.

CreateDocument

```
public function UpdateDocument($documentId, $arFields)
```

Метод изменяет свойства (поля) указанного документа на указанные значения. Параметры метода:

<i>string \$documentId</i>	код документа
<i>array \$arFields</i>	массив новых значений свойств документа в виде: <pre>array(код_свойства => значение, ...)</pre>

Коды свойств соответствуют кодам свойств, возвращаемым методом **GetDocumentFields**.

DeleteDocument

```
public function DeleteDocument($documentId)
```

Метод удаляет указанный документ. Параметры метода:

<i>string \$documentId</i>	код документа
----------------------------	---------------

PublishDocument

```
public function PublishDocument($documentId)
```

Метод публикует документ, то есть делает его доступным в публичной части сайта. Параметры метода:

<i>string \$documentId</i>	код документа.
----------------------------	----------------



UnpublishDocument

```
public function UnpublishDocument($documentId)
```

Метод снимает документ с публикации, то есть делает его недоступным в публичной части сайта. Параметры метода:

<i>string \$documentId</i>	код документа
----------------------------	---------------

LockDocument

```
public function LockDocument($documentId, $workflowId)
```

Метод блокирует указанный документ для указанного бизнес-процесса. Заблокированный документ может изменяться только указанным бизнес-процессом. Если удалось заблокировать документ, то возвращается **true**, иначе – **false**. Параметры метода:

<i>string \$documentId</i>	код документа
<i>string \$workflowId</i>	код бизнес процесса

UnlockDocument

```
public function ($documentId, $workflowId)
```

Метод разблокирует указанный документ. При разблокировке вызываются обработчики события вида **Сущность_OnUnlockDocument**, которым входящим параметром передается код документа. Если удалось разблокировать документ, то возвращается **true**, иначе - **false**. Параметры метода:

<i>string \$documentId</i>	код документа;
<i>string \$workflowId</i>	код бизнес процесса.

IsDocumentLocked

```
public function IsDocumentLocked($documentId, $workflowId)
```

Метод проверяет, заблокирован ли указанный документ для указанного бизнес-процесса. Т.е. если для данного бизнес-процесса документ не доступен для записи из-за того, что он заблокирован другим бизнес-процессом, то метод должен вернуть **true**, иначе - **false**. Параметры метода:

<i>string \$documentId</i>	код документа;
<i>string \$workflowId</i>	код бизнес-процесса.

CanUserOperateDocument

```
public function CanUserOperateDocument($operation, $userId, $documentId, $arParams = array())
```



Метод проверяет права на выполнение операций над заданным документом. Проверяются операции:

- 0 - просмотр данных бизнес-процесса,
- 1 - запуск бизнес-процесса,
- 2 - право изменять документ,
- 3 - право смотреть документ.

Если права есть, то возвращается **true**, иначе – **false**. Параметры метода:

<i>int \$operation</i>	операция
<i>int \$userId</i>	код пользователя, для которого проверяется право на выполнение операции
<i>string \$documentId</i>	код документа, к которому применяется операция
<i>array \$arParams</i>	ассоциативный массив вспомогательных параметров. Используется для того, чтобы не рассчитывать заново те вычисляемые значения, которые уже известны на момент вызова метода. Стандартными являются ключи массива DocumentStates - массив состояний бизнес-процессов данного документа, WorkflowId - код бизнес-процесса (если требуется проверить операцию на одном бизнес-процессе). Массив может быть дополнен другими произвольными ключами.

CanUserOperateDocumentType

```
public function CanUserOperateDocumentType($operation, $userId, $documentType, $arParams = array())
```

Метод проверяет права на выполнение операций над документами заданного типа. Проверяются операции:

- 2 - право изменять документ,
- 4 - право изменять шаблоны бизнес-процессов для данного типа документа.

Если права есть, то возвращается **true**, иначе – **false**. Параметры метода:

<i>int \$operation</i>	операция
<i>int \$userId</i>	код пользователя, для которого проверяется право на выполнение операции
<i>string \$documentId</i>	код документа, к которому применяется операция
<i>array \$arParams</i>	ассоциативный массив вспомогательных параметров. Используется для того, чтобы не рассчитывать заново те вычисляемые значения, которые уже известны на момент



	вызова метода. Стандартными являются ключи массива DocumentStates - массив состояний бизнес-процессов данного документа, WorkflowId - код бизнес-процесса (если требуется проверить операцию на одном бизнес-процессе). Массив может быть дополнен другими произвольными ключами.
--	---

GetDocumentAdminPage

```
public function GetDocumentAdminPage($documentId)
```

Метод по коду документа возвращает ссылку на страницу документа в административной части. Параметры метода:

<i>string \$documentId</i>	код документа.
----------------------------	----------------

GetDocumentForHistory

```
public function ($documentId)
```

Метод возвращает массив произвольной структуры, содержащий всю информацию о документе. По этому массиву документ восстанавливается методом **RecoverDocumentFromHistory**. Параметры метода:

<i>string \$documentId</i>	код документа
----------------------------	---------------

RecoverDocumentFromHistory

```
public function RecoverDocumentFromHistory($documentId, $arDocument)
```

Метод восстанавливает указанный документ из массива. Массив создается методом **RecoverDocumentFromHistory**. Параметры метода:

<i>string \$documentId</i>	код документа
<i>array \$arDocument</i>	массив

GetAllowableOperations

```
public function GetAllowableOperations($documentType)
```

Метод для типа документа возвращает массив доступных операций в виде

```
array(
    "код_операции" => "название_операции_на_текущем_языке",
    ...
)
```

Параметры метода:



<i>string \$documentType</i>	код типа документа
------------------------------	--------------------

GetAllowableUserGroups

```
public function GetAllowableUserGroups($documentType)
```

Метод для типа документа возвращает массив возможных групп пользователей в виде

```
array(  
    "код_группы" => "название_группы_на_текущем_языке",  
    ...  
)
```

Параметры метода:

<i>string \$documentType</i>	код типа документа
------------------------------	--------------------

GetUsersFromUserGroup

```
public function GetUsersFromUserGroup($group, $documentId)
```

Метод возвращает пользователей указанной группы для указанного документа в виде массива кодов пользователей. Параметры метода:

<i>string \$group</i>	код группы пользователей
<i>string \$documentId</i>	код документа

Чтобы бизнес-процесс мог работать с каким-либо объектом, необходимо выбрать:

- что будет являться документом,
- что будет являться типом документа,
- какие будут идентификаторы у документа и типа документа.

Кроме того нужно реализовать класс документа в соответствии с интерфейсом **IBPWorkflowDocument**.

Для полноценной работы с бизнес-процессами, необходимо так же при создании документа организовать создание и запуск бизнес-процессов, настроенных на автозапуск. Нужно подключить интерфейс для создания шаблонов бизнес процессов. А так же нужно предоставить пользователям интерфейс для управления запущенными бизнес-процессами. Для создания указанного функционала существует готовое API.



Класс-обертка CBPDocument

Статический класс **CBPDocument** содержит методы-обертки для облегчения пользования методами бизнес-процессов. Доступные статические методы класса:

GetDocumentStates

```
array GetDocumentStates($documentType, $documentId = null)
```

Метод возвращает массив всех бизнес-процессов и их состояний для данного документа. Если задан код документа, то метод возвращает массив всех запущенных для данного документа бизнес-процессов (в том числе и завершенных), а так же шаблонов бизнес-процессов, настроенных на автозапуск при изменении документа. Если код документа не задан, то метод возвращает массив шаблонов бизнес-процессов, настроенных на автозапуск при создании документа.

Параметры метода:

<i>array</i> <i>\$documentType</i>	тип документа в виде массива <i>array</i> (модуль, сущность, тип_документа_в_модуле)
<i>mixed</i> <i>\$documentId</i>	код документа в виде массива <i>array</i> (модуль, сущность, код_документа_в_модуле). Если новый документ, то null .

Возвращается массив вида:

```

array(
    код_бизнес-процесса_или_шаблона => array(
        "ID" => код_бизнес-процесса,
        "TEMPLATE_ID" => код_шаблона_бизнес-процесса,
        "TEMPLATE_NAME" => название_шаблона_процесса,
        "TEMPLATE_DESCRIPTION" => описание_шаблона_процесса,
        "TEMPLATE_PARAMETERS" => массив_параметров_запуска_процесса
    _из_шаблона,
        "STATE_NAME" => текущее_состояние_процесса,
        "STATE_TITLE" => название_текущего_состояния_процесса,
        "STATE_MODIFIED" => дата_изменения_статуса_процесса,
        "STATE_PARAMETERS" =>
    массив_событий_принимаемых_процессом_в_данном_состоянии,
        "STATE_PERMISSIONS" =>
    права_на_операции_над_документом_в_данном_состоянии,
        "WORKFLOW_STATUS" => статус_процесса,
    ),
    ...
)

```

В зависимости от того, бизнес-процесс это или шаблон, часть полей может быть не установлена. Для шаблона бизнес-процесса со статусами состоянием является его начальное состояние.



Массив параметров запуска бизнес-процесса из шаблона (*TEMPLATE_PARAMETERS*) имеет вид:

```
array(  
    "param1" => array(  
        "Name" => "Параметр 1",  
        "Description" => "",  
        "Type" => "int",  
        "Required" => true,  
        "Multiple" => false,  
        "Default" => 8,  
        "Options" => null,  
    ),  
    "param2" => array(  
        "Name" => "Параметр 2",  
        "Description" => "",  
        "Type" => "select",  
        "Required" => false,  
        "Multiple" => true,  
        "Default" => "v2",  
        "Options" => array(  
            "v1" => "V 1",  
            "v2" => "V 2",  
            "v3" => "V 3",  
            ...  
        ),  
    ),  
    ...  
)
```

Допустимые типы параметров: *int*, *double*, *string*, *text*, *select*, *bool*, *date*, *datetime*, *user*.

Массив событий, принимаемых процессом в данном состоянии (*STATE_PARAMETERS*) имеет вид:

```
array(  
    array(  
        "NAME" => принимаемое_событие,  
        "TITLE" => название_принимаемого_события,  
        "PERMISSION" =>  
        массив_групп_пользователей_могущих_отправить_событие  
    ),  
    ...  
)
```

Права на операции над документом в данном состоянии (*STATE_PERMISSIONS*) имеют вид:

```
array(  
    операция => массив_групп_пользователей_могущих_осуществлять_операцию,
```




```
) ...
```

GetDocumentState

```
array GetDocumentState($documentId, $workflowId)
```

Метод для данного документа возвращает состояние указанного бизнес-процесса. Результирующий массив аналогичен массиву метода **GetDocumentStates**.

Параметры метода:

<i>array \$documentId</i>	код документа в виде массива <i>array(модуль, сущность, код_документа_в_модуле)</i>
<i>string \$workflowId</i>	код бизнес-процесса.

GetAllowableEvents

```
array GetAllowableEvents($userId, $arGroups, $arState)
```

Метод возвращает массив событий, которые указанный пользователь может отправить бизнес-процессу в указанном состоянии. Параметры метода:

<i>int \$userId</i>	код пользователя
<i>array \$arGroups</i>	массив групп пользователя
<i>array \$arState</i>	состояние бизнес-процесса

Возвращается массив событий вида

```
array(
    array(
        "NAME" => событие,
        "TITLE" => название_события
    ),
    ...
)
```

GetAllowableOperations

```
array GetAllowableOperations($userId, $arGroups, $arStates)
```

Метод возвращает массив операций, которые указанный пользователь может совершить, если документ находится в указанных состояниях. Если среди состояний нет ни одного бизнес-процесса типа конечных автоматов, то возвращается **null**. Если пользователь не может выполнить ни одной операции, то возвращается *array()*. Иначе возвращается массив доступных для пользователя операций в виде *array(операция, ...)*.



Параметры метода:

<i>int \$userId</i>	код пользователя
<i>array \$arGroups</i>	массив групп пользователя
<i>array \$arStates</i>	массив состояний бизнес-процессов документа

CanOperate

```
bool CanOperate($operation, $userId, $arGroups, $arStates)
```

Метод проверяет, может ли конкретный пользователь совершить выбранную операцию, если документ находится в указанных состояниях. Если среди состояний нет ни одного бизнес-процесса со статусами, то возвращается **true**. Если пользователь не может выполнить операцию, то возвращается **false**. Иначе возвращается **true**.

Параметры метода:

<i>string \$operation</i>	операция
<i>int \$userId</i>	код пользователя
<i>array \$arGroups</i>	массив групп пользователя
<i>array \$arStates</i>	массив состояний бизнес-процессов документа

StartWorkflow

```
string StartWorkflow($workflowTemplateId, $documentId, $arParams, &$arErrors)
```

Метод запускает бизнес-процесс по коду его шаблона. Возвращается код запущенного бизнес-процесса. В случае ошибки при запуске в последнем параметре возвращается массив ошибок. Параметры метода:

<i>int \$workflowTemplateId</i>	код шаблона бизнес-процесса
<i>array \$documentId</i>	код документа в виде массива <i>array(модуль, сущность, код_документа_в_модуле)</i>
<i>array \$arParams</i>	массив параметров запуска бизнес-процесса
<i>array \$arErrors</i>	массив ошибок, которые произошли при запуске бизнес-процесса в виде: <pre><i>array(array("code" => код_ошибки,</i></pre>



	<pre> "message" => сообщение, "file" => путь_к_файлу), ...) </pre>
--	--

AutoStartWorkflows

```
void AutoStartWorkflows($documentType, $autoExecute, $documentId, $arParams, &$arErrors)
```

Метод запускает бизнес-процессы, настроенные на автозапуск. Параметры метода:

<i>array \$documentType</i>	код типа документа в виде массива <i>array(модуль, сущность, код_типа_документа_в_модуле)</i>
<i>int \$autoExecute</i>	флаг типа автозапуска (CBPDocumentEventType::Create - автозапуск на создание документа, CBPDocumentEventType::Edit - автозапуск на изменение)
<i>array \$documentId</i>	код документа в виде массива <i>array(модуль, сущность, код_документа_в_модуле)</i>
<i>array \$arParams</i>	массив параметров запуска бизнес-процесса
<i>array \$arErrors</i>	массив ошибок, которые произошли при запуске бизнес-процесса в виде: <pre> array(array("code" => код_ошибки, "message" => сообщение, "file" => путь_к_файлу), ...) </pre>

SendExternalEvent

```
void ($workflowId, $workflowEvent, $arParams, &$arErrors)
```

Метод отправляет внешнее событие бизнес-процессу. Параметры метода:



<i>string \$workflowId</i>	код бизнес-процесса
<i>string \$workflowEvent</i>	событие
<i>array \$arParams</i>	параметры события
<i>array \$arErrors</i>	массив ошибок, которые произошли при отправке события в виде: <pre>array(array("code" => код_ошибки, "message" => сообщение, "file" => путь_к_файлу), ...)</pre>

TerminateWorkflow

```
void TerminateWorkflow($workflowId, $documentId, &$arErrors)
```

Метод останавливает выполнение бизнес-процесса. Параметры метода:

<i>string \$workflowId</i>	код бизнес-процесса
<i>array \$documentId</i>	код документа в виде массива <i>array(модуль, сущность, код_документа_в_модуле)</i>
<i>array \$arErrors</i>	массив ошибок, которые произошли при остановке бизнес-процесса в виде: <pre>array(array("code" => код_ошибки, "message" => сообщение, "file" => путь_к_файлу), ...)</pre>



)
--	---

OnDocumentDelete

```
void OnDocumentDelete($documentId, &$arErrors)
```

Метод удаляет все связанные с документом записи. Параметры метода:

<i>array \$documentId</i>	код документа в виде массива <i>array</i> (модуль, сущность, код_документа_в_модуле)
<i>array \$arErrors</i>	массив ошибок, которые произошли при удалении в виде: <pre>array(array("code" => код_ошибки, "message" => сообщение, "file" => путь_к_файлу), ...)</pre>

GetWorkflowTemplatesForDocumentType

```
array GetWorkflowTemplatesForDocumentType($documentType)
```

Метод возвращает массив шаблонов бизнес-процессов для данного типа документа. Параметры метода:

<i>array \$documentType</i>	код типа документа в виде массива <i>array</i> (модуль, сущность, код_типа_документа_в_модуле)
-----------------------------	--

Возвращается массив в виде:

```
array(
    array(
        "ID" => код_шаблона,
        "NAME" => название_шаблона,
        "DESCRIPTION" => описание_шаблона,
        "MODIFIED" => дата_изменения_шаблона,
        "USER_ID" => код_пользователя_изменившего_шаблон,
        "USER_NAME" => имя_пользователя_изменившего_шаблон,
        "AUTO_EXECUTE" => флаг_автовыполнения_CBPDocumentEventType,
        "AUTO_EXECUTE_TEXT" => текст_автовыполнения,
```



```

    ),
    ...
)

```

Например, выберем все шаблоны бизнес-процессов для информационного блока с ID равным 28:

```

$arWorkflowTemplates = CBPDocument::GetWorkflowTemplatesForDocumentType(
    array("iblock", "CIBlockDocument", "iblock_28")
);

```

DeleteWorkflowTemplate

```

void DeleteWorkflowTemplate($id, $documentType, &$arErrors)

```

Метод удаляет шаблон бизнес-процесса. Параметры метода:

<i>int \$id</i>	код шаблона бизнес-процесса
<i>array \$documentType</i>	код типа документа в виде массива <i>array(модуль, сущность, код_типа_документа_в_модуле)</i>
<i>array \$arErrors</i>	массив ошибок, которые произошли при выполнении в виде: <pre> array(array("code" => код_ошибки, "message" => сообщение, "file" => путь_к_файлу), ...) </pre>

Например, удалим шаблон бизнес-процесса с кодом 132 для инфоблока 18:

```

CBPDocument::DeleteWorkflowTemplate(
    132,
    array("iblock", "CIBlockDocument", "iblock_18"),
    $arErrorTmp
)

```

UpdateWorkflowTemplate

```

void UpdateWorkflowTemplate($id, $documentType, $arFields, &$arErrors)

```



Метод изменяет параметры шаблона бизнес-процесса. Параметры метода:

<i>int \$id</i>	код шаблона бизнес-процесса
<i>array \$documentType</i>	код типа документа в виде массива <i>array(модуль, сущность, код_типа_документа_в_модуле)</i>
<i>array \$arFields</i>	массив новых значений параметров шаблона бизнес-процесса
<i>array \$arErrors</i>	массив ошибок, которые произошли при выполнении в виде: <pre>array(array("code" => код_ошибки, "message" => сообщение, "file" => путь_к_файлу), ...)</pre>

Например, изменим флаг автозапуска шаблона бизнес-процесса с кодом 132 для инфоблока с кодом 32:

```
CBPDocument::UpdateWorkflowTemplate(
    132,
    array("iblock", "CIBlockDocument", "iblock_32"),
    array(
        "AUTO_EXECUTE" => CBPDocumentEventType::Create
    ),
    $arErrorsTmp
);
```

GetUserTasksForWorkflow

```
array GetUserTasksForWorkflow($userId, $workflowId)
```

Метод возвращает массив заданий для данного пользователя в данном бизнес-процессе. Возвращаемый массив имеет вид:

```
array(
    array(
        "ID" => код_задания,
        "NAME" => название_задания,
        "DESCRIPTION" => описание_задания,
    ),
    ...
)
```



) ...

Параметры метода:

<i>int \$userId</i>	код пользователя
<i>string \$workflowId</i>	код бизнес процесса



Заключение

В руководстве были рассмотрены некоторые моменты работы модуля **Бизнес-процессы** и описание API для программирования в рамках этого модуля.

Если у вас возникнут вопросы, то их можно задавать в форуме на сайте компании "1С-Битрикс":

<http://dev.1c-bitrix.ru/community/forums/>

или же решать в рамках **Технической поддержки** компании:

<http://dev.1c-bitrix.ru/support/>