Многофункциональный модуль компании «Индивид»

Оглавление

Предисловие

Данный модуль содержит наиболее часто используемые шаблоны и приёмы проектирования при разработке приложений в компании.

Модуль призван

- ускорить разработку приложения:
  - на этапе начального развёртывания;
  - на этапе разработки;
  - на этапе отладки;
- обеспечить преемственность кода в рамках компании;
- унифицировать организацию инфраструктуры приложения.

Установка модуля

Модуль должен находиться в папке *local* корневной папки сайта.

Модуль устанавливается как решение для Marketplace 1С-Битрикс (установка производится в разделе «Установленные решения»).

Важный момент: все содержащие символы национальных алфавитов файлы модуля находятся в кодировке UTF-8. Поэтому, если сайт, на который ставится модуль, работает в однобайтной кодировке, то перед его установкой необходимо все вышеупомянутые файлы в эту кодировку сконвертировать.

Структура папок

- **lib** — стандартная папка модуля 1С-Бирикс, содержит PHP классы модуля в стандарте архитектуры D7. В данной папке следует создавать новые классы, требующиеся для реализации БЛ приложения;
- **examples** — нестандартная папка, содержит типовые примеры использования функциональностей модуля. В данной папке при разработке ничего создавать не нужно;
- **admin** — нестандартная папка, содержит дополнения для ядра;
- **install** — стандартная папка модуля 1С-Бирикс, содержит файлы и папки, используемые при установке модуля:
    - **cron** — папка копируется в каталог */local/*, содержит стандартный набор файлов, облегчающих запуск периодических задач через crontab;
    - **components** — папка копируется в каталог */local/*, содержит собственные компоненты компании, используемые при разработке приложения;
    - **sql** — папка содержит инсталляционный дамп БД модуля;
    - **doc_root** — папка копируется в корень сайта, содержит стандартный набор файлов, используемый в типовой структуре приложения на ранней стадии разработки;
    - **js** — папка копируется в каталог */bitrix/*, содержит js-скрипты для ядра;
    - **php_interface** — папка копируется в каталог */bitrix/*, содержит дополнения для ядра;
    - **templates** — папка копируется в каталог */local/*, содержит инсталляционный шаблон сайта с уже внедренным набором решений, используемых на ранней стадии разработки приложения;
- **lang** — стандартная папка модуля 1С-Бирикс, содержит языковые файлы модуля. В данной папке следует создавать языковые файлы для новых классов;
- **views** — нестандартная папка, содержит файлы PHP, используемые на сайте в качестве шаблонов представлений (view) парадигмы MVC. В данной папке следует создавать шаблоны представлений, требующиеся для реализации БЛ приложения.

Инфраструктура

PHP

Автозагрузка классов

В модуле используется автозагрузка классов средствами ядра D7 1С-Битрикс.

Автозагрузка происходит при первой попытке обратиться к классу, до этого момента файл не подключается.

Система именования классов в папке lib

Для корректной автозагрузки достаточно правильно называть класс в соответствии с его положением в файловой системе модуля, принцип следующий:

- папка — это пространство имен в модуле;
- файл — это класс в пространстве имен.

Примеры:

Название класса	**Файл класса**
\Indi\Main\Util	lib/util.php
\Indi\Main\Mvc\Controller\Form	lib/mvc/controller/form.php

При разработке новых классов и модулей рекомендуется придерживаться данной системы наименований.

Инициализация приложения, класс Indi\Main\Module

Класс обеспечивает установку начального состояния приложения. При установке модуля добавляется обработчик события *OnPageStart* модуля *main*, который реализован в методе *onPageStart.* В этом методе уже предусмотрен ряд действий: установка констант, создание дополнительных обработчиков.

Если в процессе разработки проекта возникает необходимость добавления каких-либо своих обработчиков или запуска всяких проектно-специфичных «фич», то это нужно делать именно в этом классе.

Работа с инфоблоками, класс Indi\Main\Iblock\Prototype

Базовые сведения

Файл класса: *lib/iblock/prototype.php*

Класс служит для облегчения рутинных операций с инфоблоками. Наследники класса служат для реализации бизнес-логики конкретных инфоблоков.

В классе используются шаблоны проектирования Singleton и Factory, что дает нам следующие возможности:

- для каждого инфоблока создается только один экземпляр обслуживающего его класса со всеми плюсами и минусами паттерна Singleton;
- для каждого инфоблока по его символьному коду (либо по идентификатору) можно получить

обслуживающий класс;

- инстанциируемые классы и, как следствие, динамические методы позволяют решить проблему позднего статического связывания при работе с наследованием в версиях PHP < 5.3.

Прочие «фишки»:

1. Метод *getInstance()* используется для получения singleton-экземпляра класса, отвечающего за БЛ данного инфоблока. Аргументом метода является идентификатор или символьный код инфоблока. Если передан символьный код, то он разбивается по заглавным буквам, и таким образом определяется имя класса. Например, символьный код *Content_News* дает имя класса *Indi\Main\Iblock\Content\News*. Если такой класс есть — возвращается экземпляр этого класса. Если нет — то экземпляр *Indi\Main\Iblock\Prototype.*
2. Чтобы было удобнее работать в IDE, нужно переопределить метод getInstance*()* в классе-наследнике *Indi\Main\Iblock\Prototype* с указанием в формате PHPDoc, что метод возвращает уже класс-наследник. Пример можно посмотреть в файле *lib/iblock/content/news.php*.
3. Предопределённые константы для связи символьного кода и идентификатора инфоблока: метод *Indi\Main\Iblock\Prototype::defineConstants()* создает глобальные константы вида \Indi\Main\Iblock*ID_{CODE}*, значением каждой является идентификатор инфоблока. Используя эти константы можно в любом месте приложения определить идентификатор инфоблока по его символьному коду. Например, для инфоблока с ID=1 и символьным кодом *ContentNewsRu* будет создана константа *\Indi\Main\Iblock\ID_ContentNewsRu* со значением 1.

   Аналогично создаются константы вида *\Indi\Main\Iblock\CODE_{ID}* для определения символьного кода инфоблока по его идентификатору. Например, для инфоблока с ID=1 и символьным кодом *ContentNewsRu* будет создана константа *\Indi\Main\Iblock\CODE_1* со значением ContentNewsRu.

   Вызов метода *Indi\Main\Iblock\Prototype::defineConstants()* осуществляется автоматически при первом вызове *Indi\Main\Iblock\Prototype::getInstance()*.
4. При работе с классом *Indi\Main\Iblock\Prototype* не нужно 100`500 раз писать *CModule::IncludeModule('iblock')*. Это делается автоматически при первом обращении к классу.
5. Внутренняя автоматизированная система кэширования:
   для кэширования данных в динамических методах класса *Indi\Main\Iblock\Prototype* и его наследников следует использовать метод *getCache()* (пример можно посмотреть в методе *Indi\Main\Iblock\Prototype::getData()*).

Важный момент: т. к. символьный код инфоблока используется в именовании обслуживающего его класса, то это накладывает ограничение на используемые в символьном коде символы. В частности нельзя использовать знак «минус», т. к. в PHP этот знак означает математический оператор. В идеале символьные коды инфоблоков следует формировать по принципу UpperCamelCase в формате {ТипИнфоблока}{СущностьИнфоблока}[{ЯзыкИнфоблока}].

Например: для типа инфоблока «Содержимое» и инфоблока «Новости» с этим типом символьный код будет «*ContentNews*». Имя класса соответственно - *Indi\Main\Iblock\Content\News*.

Все методы класса *Indi\Main\Iblock\Prototype* более чем достаточно документированы, поэтому для детального изучения его возможностей следует ознакомиться с исходным кодом.

Примеры работы с классом доступны в файле:

examples\php\iblock.php

Реализация бизнес-логики отдельных инфоблоков

Для примера модуль содержит реализацию класса инфоблока «Новости», файл — *lib/iblock/content/news.php*

Допустим, для этого инфоблока требуется реализовать какую-то специфичную логику. Например, в качестве бэкграунда на сайте используется детальная картинка самой последней новости, для которой эта детальная картинка загружена 8-).

Для этого пишем динамический метод *getLastElementWithPicture()*, который возвращает нам нужную новость (и при этом использует встроенную в класс систему кэширования).

Чтобы в любом приложения месте получить эту новость, достаточно выполнить следующий код:

```
$elementForBG = \Indi\Main\Iblock\Content\News::getInstance()-
>getLastElementWithPicture();
```

либо, что равнозначно:

```
$elementForBG = \Indi\Main\Iblock\Prototype::getInstance('ContentNews')-
>getLastElementWithPicture();
```

Ещё один пример часто решаемой задачи — исключение дублирующегося кода.

Допустим, есть инфоблок «События», данные которого выводятся во множестве мест сайта, причём разными компонентами и разными шаблонами. При этом выводимые данные обладают рядом общих признаков сущности «событие», которые необходимо дополнительно определить. Например:

- количество дней до начала события;
- отформатированный период действия события (в зависимости от наличия дат окончания/начала);
- отформатированная ссылка на сайт организатора (текстовое отображение ссылки не должно содержать протокольный префикс, а адрес ссылки — наоборот. Причем это должно работать корректно вне зависимости от того, указан ли протокольный префикс в БО или нет;
- иконки, выводимые на картинке в зависимости от ряда признаков;
- и т. п.

Как в этом случае зачастую поступает разработчик? Он в каждом шаблоне компонента создает файл *result_modifier.php*, в котором пишет по сути один и тот же код: считает кол-во дней до начала события, форматирует данные и т.п. В результате получаем негативные последствия (дублированный код затрудняет исправление и отслеживание ошибок). Исправляем ошибку в одном файле — она остается в другом. Вынуждены править кучу файлов, которые сначала еще надо найти.

Как поступить правильно? Нужно исключить дублирование кода. В нашем случае правильно для инфоблока «События» создать обслуживающий класс. В нем реализовать метод, реализующий специфику представления элемента инфоблока. Например, назовём его *normalizeElement*. А файлы *result_modifier.php* во всех шаблонах компонентов приведём примерно к следующему виду:

```
$iblock = \Indi\Main\Iblock\Prototype::getInstance($arParams['IBLOCK_ID']);
if (method_exists($iblock, 'normalizeElement')) {
      foreach ($arResult['ITEMS'] as &$arItem) {
            $iblock->normalizeElement($arItem);
      }
      unset($arItem);
}
```

В классе инфоблока, пишем метод *normalizeElement.* Выглядеть он может так:

```
/**
 * Настраивает данные элемента инфоблока в соотв-ии с БЛ
 *
 * @param array $data Данные элемента
 * @param boolean $detail Для детальной страницы
 * @return void
 */
```

```
public function normalizeElement(&$data, $detail = false)
{
	//Нормализуем адрес сайта
	if ($data['PROPERTIES']['WEBSITE']['VALUE']) {
		$website = explode('://', $data['PROPERTIES']['WEBSITE']['VALUE']);
		if (count($website) == 1) {
			$data['PROPERTIES']['WEBSITE']['VALUE'] = 'http://' . $website[0];
			$data['PROPERTIES']['WEBSITE']['LABEL'] = $website[0];
		} else {
			$data['PROPERTIES']['WEBSITE']['LABEL'] = $website[1];
		}
	}

}
```

В итоге для исправления ошибок, либо для модификации схемы вывода данных, нам понадобится внести изменения только в метод *normalizeElement* класса инфоблока.

Работа с консолью браузера, класс Indi\Main\Console

Для облегчения отладки приложения, а также для «прозрачной» отладки на боевых серверах в случаях, когда это требуется, можно выводить отладочную информацию в консоль браузера. Для этого используются два метода: *\Indi\Main\Console::log()* и *\Indi\Main\Console::error()*. Первым аргументом передаются любые данные, которые нужно вывести в консоль.

Для вывода на клиентской стороне используется JavaScript-объект *console*. Если браузер не поддерживает объект console, то обращение к нему не происходит и т. о. не возникает ошибок в коде на клиентской стороне.

Никаких глобальных переменных, класс Indi\Main\Registry

При переходе от процедурного программирования к объектно-ориентированному PHP разработчику зачастую бывает сложно отказаться от использования глобальных переменных. Но т. к. глобальные переменные — это зло, то пользоваться ими ни в коем случае нельзя.

В модуле вместо глобальных переменных предусмотрен класс *Indi\Main\Registry*, представляющий собой реестр обще используемых «объектов». Концепция такова: если существует необходимость обращения к некоторым общим данным в различных функциональных частях приложения, то эти данные должны храниться централизованно, чтобы можно было легко отследить, кто и откуда к ним обращается.

Для сохранения «объекта» в реестр используется метод *\Indi\Main\Registry::set()*.

Для получения «объекта» из реестра используется метод *\Indi\Main\Registry::get()*.

Пример практического использования

На сайте используется геотаргетинг. Данную функциональность можно условно разделить на две части:

- при запуске приложения по каким-либо признакам (сохраненные данные в сессии, cookies, IP-адрес) определяется географическое положение посетителя. Результат записывается в реестр. Например так:

```
\Indi\Main\Registry::set('geoLocation', $location);
```

- все функциональности, которые используют геотаргетинг, обращаются в реестр за ранее определенным географическим положением. Например так:

```
$location = \Indi\Main\Registry::get('geoLocation');
```

Важный момент: не следует увлекаться сохранением чего ни попадя в реестр, использовать только в случаях действительной необходимости.

Работа с поисковым индексом, класс Indi\Main\Search

Одна из часто решаемых задач при разработке сайта — изменение схемы поисковой индексации. Особенно часто — для инфоблоков. Для облегчения решения такой задачи реализован класс *Indi\Main\Search*.

Суть работы в случае инфоблоков такова — для каждого класса инфоблока должен быть реализован метод класса с именем *onBeforeSearchIndexElement* – для элементов, и *onBeforeSearchIndexSection* – для разделов. Эти методы отвечают за индексацию элементов и разделов конкретного инфоблока.

Пример практического использования

В инфоблоке «Новости» для элементов заведено свойство «URL». Если у элемента это свойство содержит адрес, ведущий на внешний ресурс, то в результатах поисковой выдачи такие элементы должны выводиться со ссылкой на этот самый ресурс.

Реализация: пишем метод *onBeforeSearchIndexElement*, который подменяет ссылку у элемента поискового индекса. Код метода см. в файле *lib\iblock\content\news.php*.

Парадигма MVC, классы Indi\Main\Mvc\Controller*, Indi\Main\Mvc\View*

В случаях, когда требуется обеспечить какую-либо интерактивную функциональность на клиентской части приложения, не привязанную к компонентам, в модуле предусмотрен свой ~~велосипед~~ вариант реализации парадигмы MVC (правильный, а не как это преподносят разработчики 1С-Битрикс).

Схема работы

1. Приложение клиентской части обращается на сервер путем AJAX запроса по URL, сформированному определенным образом.
2. Приложение серверной части по URL определяет какой контроллер и какой метод контроллера должен обработать запрос (так называемый routing). Создает инстанцию контроллера и вызывает запрашиваемый метод (экшн).
3. Метод контроллера обращается к модели/моделям и получает от них необходимые данные. Также метод назначает представление, отвечающее за вывод данных.
4. Полученные данные передаются представлению, которое формирует итоговый вариант ответа на запрос.
5. Ответ передается на клиентскую часть и там обрабатывается.

Принцип формирования URL и роутинг

При установке модуля в корне сайта создаётся папка *ajax*, в которой лежит файл-роутер. Данный файл отвечает за разбор URL, который должен формироваться следующим образом:

/ajax/{Controller}/{Action}/[?param1=val1[&param2=val2[...]]],

где:

- *Controller* — название контроллера;
- *Action* — название экшена.

Роутер разбирает URL и обращается к «фабрике» контроллеров — методу *factory* класса *\Indi\Main\Mvc\Controller\Prototype*, которой передаёт название контроллера:

```
$controller = \Indi\Main\Mvc\Controller\Prototype::factory(isset($urlParts[2]) ?
$urlParts[2] : 'default');
```

Фабрика по названию контроллера создаёт инстанцию обслуживающего его класса. Принцип формирования имени класса следующий: \Indi\Main\Mvc\Controller*{Controller}*.

Если класс с таким именем существует, то создаётся его инстанция. В противному случае — создаётся исключение.

Затем роутер запускает экшн контроллера путем вызова метода *doAction* класса *\Indi\Main\Mvc\Controller\Prototype*. Данному методу передаётся название экшена:

```
$controller->doAction(isset($urlParts[3]) ? $urlParts[3] : 'default');
```

По имени экшена формируется имя метода контроллера, отвечающего за его выполнение. Принцип формирования имени метода следующий: *{Action}Action*.

Если метод с таким именем существует, то он вызывается. В противном случае — создаётся исключение.

Контроллеры

Все новые контроллеры модуля должны складываться в папку *lib/mvc/controller/*. Названия файлов и контроллеров должны совпадать.

Например, если название контроллера «*news*», то файл контроллера должен называться *news.php*, а класс контроллера должен называться *Indi\Main\Mvc\Controller\News*.

Каждый класс контроллера должен являться наследником *Indi\Main\Mvc\Controller\Prototype*.

Представления

Для хранения шаблонов представлений в модуле выделена отдельная папка *views*. Для каждого контроллера шаблоны представлений желательно складывать в отдельную папку с названием, совпадающим с названием контроллера.

Например, если название контроллера «*news*», то папка для шаблонов представлений этого контроллера должна называться *news* (т. е. находится в модуле по пути *views/news*).

В модуле реализованы классы, отвечающих за разные виды представлений:

- *Indi\Main\Mvc\View\Json* — выводит данные в формате JSON (для данного вида представлений файлы шаблонов не используются);
- *Indi\Main\Mvc\View\Php* — выводит данные через php-шаблон;
- *Indi\Main\Mvc\View\Html* — выводит данные в формате HTML (для данного вида представлений файлы шаблонов не используются);
- *Indi\Main\Mvc\View\Xml* — выводит данные в формате XML (для данного вида представлений файлы шаблонов не используются).

По умолчанию все контроллеры выводят данные с использованием *Indi\Main\Mvc\View\Json*.

Чтобы сменить вид представления, в методе экшена контроллера следует его переназначить. Например, чтобы использовать *Indi\Main\Mvc\View\Php* с нужным файлом шаблона, можно написать следующее:

```
$this->view = new Mvc\View\Php('form/result.php');
```

где *form/result.php* — файл шаблона представления, находящийся в каталоге *views*.

Пример реализации

Пример реализации контроллера можно найти в файле *lib\mvc\controller\form.php*

Клиентская часть приложения

Пример вызова контроллера из клиентской части приложения:

```
$.get(
	'/ajax/form/feedback/',
	function(response) {
		$('#super-form').html(response);
	}
);
```

Локализация клиентской части приложения, класс Indi\Main\Locale

Для локализации JavaScript части приложения существует класс *\Indi\Main\Locale*, который реализован по паттерну Singleton. Решает 2 задачи:

1. Хранит форматы даты/времени текущего сайта, а также необходимые языковые сообщения (для этого используются языковые файлы модуля, например для русского языка файл локализации класса *\Indi\Main\Locale* находится в каталоге модуля *lang/ru/lib/locale.php*).
2. Преобразует хранимые данные в формат JSON, который является «родным» для JavaScript. Для этого используется метод *toJSON()*.

Для автоматической проекции PHP класса *\Indi\Main\Locale* в JavaScript объект *indi.app.locale* (см. раздел «Инфраструктура — Javascript») в инсталляционный шаблон сайта добавлена связка:

```
<script>indi.utils.apply(indi.app.locale, <?=\Indi\Main\Locale::getInstance()->toJSON()?>);</script>
```

Набор полезных утилит, класс Indi\Main\Util

Все полезные инструменты, необходимые при разработке, но недостаточно крупные, чтобы иметь собственный класс, сосредоточены в классе *Indi\Main\Util*.

Как наиболее часто используемые, можно выделить следующие методы этого класса:

- *getNumEnding()* — спрягает существительное и числительное в тех языках, где грамматика предусматривает такие словоформы. Например, когда нужно вывести подобные фразы: «1 комментарий», «2 комментария», «5 комментариев»;
- *getFileSize()* — выводит размер файла в адаптивном формате (килобайты, мегабайты и т. д.);
- *getURL()* — возвращает содержимое ресурса по указанному URL (используется модуль curl);
- *convertCharset()* — переводит кодировку из одного формата в другой, причем может выполнять это для массивов и объектов;
- *debug()* — выводит отладочный дамп в «эфир» через print_r();
- *dump()* — выводит отладочный дамп в «эфир» через var_dump();
- *log()* — выводит отладочный дамп в файл;
- *parseTemplate()* — заменяет в шаблоне конструкции #VAR_NAME# на значения.

Работа со страницей через AJAX

Часто при AJAX вызовах требуется получить текущую страницу с компонентом, но без содержимого header-а и footer-а шаблона сайта. Для автоматизации решения подобной задачи шаблон сайта отслеживает специальную константу модуля - Indi\Main\IS_AJAX, которая указывает, что страница запрашивается через AJAX. Если эта константа true, то содержимое header-а и footer-а не отдается.

JavaScript

Зачастую язык JavaScript используется самым примитивным образом — в режиме процедурного программирования. Однако он в достаточной степени поддерживает парадигму ООП, и более того — практически всё в этом языке является объектом. Для «просвещения» очень рекомендую почитать информацию на сайте проекта «JavaScript Garden».

Инсталляционный шаблон сайта

В инсталляционном шаблоне сайта в папке *js* находятся часто используемые JavaScript библиотеки.

Файл indi.*js*

Данный файл содержит «скелет» JavaScript-приложения.

В глобальном пространстве имен создается объект с именем *indi*. Он содержит множество других объектов и методов, позволяющих решить следующие задачи:

- выделение функциональных частей приложения (функциональные блоки);
- интеграция виджетов, управляемых через html-разметку;
- локализация клиентской части, исходя из настроек сервера;
- работа с HTML5 history API;
- определение возможностей user-agent-а пользователя;
- предоставление набора утилит для решений различного спектра задач.

Также файл *indi.js* содержит несколько jQuery плагинов.

В процессе разработки сайта в файл *indi.js* вносить изменения не следует. Делать это нужно только с целью исправления ошибок.

Файл *template.js*

В данном файле следует прописывать всю БЛ приложения: функциональные блоки, дополнительные виджеты, весь остальной специфичный для приложения функционал.

Стратегия разработки JavaScript приложения

При разработке приложения следует выделять в нем отдельные функциональные блоки. Каждый блок должен реализовывать какую-либо функциональность и быть по возможности независимым от других таких же блоков.

С точки зрения программирования для решения этой задачи следует обеспечить максимальную изолированность блоков друг от друга. Т. е. «снаружи» не должны быть видны никакие методы и тем более поля, если этого не требует логика работы блока. Реализуется подобное с помощью ограничения области видимости переменных и методов — все они должны объявляться через *var* и быть видны только в контексте блока. Таким образом физически обеспечивается невозможность вмешательства в логику работы функционального блока из контекстов вне его.

Если же требуется управлять блоком извне, то это следует делать через публичные методы. Например, через геттеры-сеттеры. Т. е. нужно реализовать интерфейс доступа к функциональному блоку (принцип инкапсуляции).

Все стандартные блоки приложения должны находиться в общем пространстве имен, чтобы обеспечить сквозной доступ к своему интерфейсу из любой точки приложения.

Зачем это нужно? Подобная организация кода облегчает внесение правок в код. В случае, когда новый человек должен произвести изменения в коде шаблона, он может быть уверен, что все вносимые им изменения коснутся только логики работы этого блока. И если он даже целиком её перепишет, то это не отразится на работе сайта, потому что блок изолирован и никакие другие скрипты не вмешиваются в его работу, либо вмешиваются только через предоставляемый интерфейс.

Попутно мы избавляемся от глобальных функций и переменных в коде. Для каждого интерфейсного элемента логика вынесена в изолированный код, предоставляющий регламентированный интерфейс управления.

Объект indi.app.blocks

indi.app.blocks — объект, содержащий структурно-логические блоки страницы сайта.

В общем случае на сайте можно выделить следующие структурно-логические блоки: шапка/header,

подвал/footer, sidebar и т. п. Все эти блоки обычно присутствуют на любой странице сайта.

Конструкторы всех блоков, указанных в *indi.app.blocks,* вызываются по мере готовности DOM.

Чтобы добавить структурно-логический блок, достаточно описать его конструктор в объекте *indi.app.blocks*, например:

```
/**
 * Блок шаблона: шапка
 */
indi.app.blocks.header = function()
{
	...
}
```

Если конструктор блока следует вызывать только при его присутствии на странице, то необходимо определить специальный метод *exists*, который отвечает за признак наличия блока на странице. Если метод *exists* не определен, то считается, что блок всегда присутствует на странице. Пример:

```
/**
 * Проверяет наличие блока
 *
 * @return boolean
 */
indi.app.blocks.header.exists = function()
{
	return $('#header').length > 0;
};
```

Методы, обеспечивающие интерфейс доступа к блоку, должны быть публичными (определяться через объект *this* в конструкторе). Пример:

```
/**
 * Блок шаблона: шапка
 */
indi.app.blocks.header = function()
{
	/**
	 * Ширина документа
	 *
	 * @var integer
	 */
	var docWidth = 0;

	/**
	 * Возвращает ширину документа
	 *
	 * @return integer
	 */
	this.getDocWidth = function()
	{
		return docWidth;
	};
	...
```

Все остальные методы (а они не должны быть доступны снаружи) должны быть приватными и также определяться в конструкторе. Пример:

```
/**
 * Блок шаблона: шапка
 */
indi.app.blocks.header = function()
{
	/**
	 * Ширина документа
	 *
	 * @var integer
	 */
```

```
	var docWidth = 0;

	/**
	 * Обработчик события изменения размеров окна браузера
	 *
	 * @return void
	 */
	var onWindowResize = function()
	{
		docWidth = $(document).width();
	};
	...
```

Все переменные блока также должны быть приватными. Пример:

```
/**
 * Блок шаблона: шапка
 */
indi.app.blocks.header = function()
{
	/**
	 * Приватное свойство, ширина документа
	 *
	 * @var integer
	 */
	var docWidth = 0;
	...
```

Пример реализации набора структурно-логических блоков и их взаимодействия находится в файле *examples/js/template.js*

При необходимости можно добавлять блоки, отвечающие за контент каких-либо страниц целиком (если на страницах не используются компоненты, js-часть которых можно оставить в скрипте компонента). Название подобных блоков лучше заканчивать суффиксом «*Page*», чтобы можно было отделить их от структурных блоков страницы. Например, *indi.app.blocks.aboutPage*, *indi.app.blocks.teamPage*.

Функциональные блоки вне объекта indi.app.blocks

Если система функциональных блоков, заложенная в indi.app.blocks, по каким-либо причинам не подходит для использования (например, в случае использования отдельного файла script.js шаблона компонента), и код в отдельном js-файле получается достаточно объемным либо сложным, то в нем также следует выделять функциональные блоки, а не писать код в виде сплошной «простыни».

Если функциональные блоки не взаимодействуют друг с другом, то можно их изолировать друг от друга через анонимные функции. Пример:

```
$(function() {
	/**
	* News box block
	*/
	(function() {
		/**
		 * News box DOM node
		 *
		 * @var object
		 */
		var domNode = $('#news-box');

		/**
		 * New box initialization
		 *
		 * @return void
		 */
		var init = function() {
```

```
            ...
      })();



      /**
      * News slider block
      */
      (function() {
            /**
             * News slider DOM node
             *
             * @var object
             */
            var domNode = $('#news-slider');

            /**
             * New slider initialization
             *
             * @return void
             */
            var init = function() {
            ...
      })();
});
```

Если же требуется обеспечить взаимодействие различных функциональных частей, то в таком случае необходимо создавать объекты с интерфейсом доступа. Пример:

```
$(function() {
      /**
      * News box block
      */
      var newsBox = new function() {
            /**
             * News box DOM node
             *
             * @var object
             */
            var domNode = $('#news-box');

            /**
             * Rotation enabled
             *
             * @var boolean
             */
            var rotation = true;

            /**
             * Enable news rotation
             *
             * @return void
             */
            this.enableRotation = function() {
                  rotation = true;
            }

            /**
             * Disable news rotation
             *
             * @return void
             */
            this.disableRotation = function() {
                  rotation = false;
            }
```

```
            /**
             * New box initialization
             *
             * @return void
             */
            this.init = function() {
            ...
      };



      /**
      * News slider block
      */
      (function() {
            /**
             * News slider DOM node
             *
             * @var object
             */
            var domNode = $('#news-slider');

            /**
             * New slider initialization
             *
             * @return void
             */
            var init = function() {
                  newsBox.disableRotation();
            ...
      })();
});
```

Виджеты, объект indi.ui.widgets

В клиентской части приложения предусмотрен механизм виджетов.

Виджет — некий визуальный элемент приложения, обладающий определенным поведением.

Чтобы сделать какой-то элемент DOM виджетом, достаточно указать ему класс widget и класс конкретного виджета.

Пример виджета, создающего переключатель на основе табов:

```
<div class="widget tabpane">
...
```

Тот же виджет, но уже с настройками в виде атрибутов data-*:

```
<div class="widget tabpane" data-tabs-selector="> h2 > .tabs">
...
```

При загрузке приложения по мере готовности DOM все виджеты на странице инициализируются.

Если нужно проинициализировать виджеты в каком-то конкретном элементе DOM (например, в случае загрузки содержимого через AJAX), следует использовать метод *indi.ui.widgets.init* с указанием первым аргументов селектора либо jQuery объекта DOM-элемента:

```
indi.ui.widgets.init('#news-box');
```

Новые виджеты, разработанные специально для конкретного сайта, следует прописывать в файле *template.js*.

Пример:

```
/**
```

```
 * Customized select field widget
 *
 * @param string|object selector DOM nodes selector
 * @return void
 */
indi.ui.widgets.items.select = function(selector) {
      $(selector).customSelect();
};
```

Прочие возможности объекта indi

Часто используемый функционал объекта *indi.app*:

- *indi.ui.loading* — отображает/прячет индикатор загрузки (удобно использовать при выполнении AJAX запросов);
- *indi.cookie* — работа с Cookie;
- *indi.ua* — свойства User-agent-а пользователя;
- *indi.utils* — набор утилит.
- *indi.app.locale* — локаль приложения. Содержит набор локализованных текстовых сообщений, а также утилиты форматирования даты и времени;
- *indi.history()* — обеспечивает поддержку HTML5 history;

CSS

В общем случае для задания стилей следует использовать файлы *styles.css* и *template_styles.css* шаблона сайта, а также стилевые файлы компонентов, как это рекомендуют делать разработчики системы 1С-Битрикс.

В случаях, когда сайт обладает уникальным дизайном отдельных статических страниц, имеет смысл стилевое оформление таких страницы вынести в отдельные файлы. Для этой цели в инсталляционном шаблоне сайта предусмотрен механизм автоподключения стилевых файлов разделов сайта (см. *Indi\Main\Util::addCSSLinksByPath()*).

Если в в папке *css* шаблона сайта существует иерархия папок, совпадающая с текущим путем, запрашиваемым у сервера, а в последней папке иерархии существует файл *style.css*, то этот файл автоматически подключится.

Примеры:

Путь в URL	Автоподключаемый файл
/about/	/bitrix/templates/{name}/css/about/style.css
/about/team/	/bitrix/templates/{name}/css/about/team/style.css

Панель импорта

Применение

В административной части сайта при установке модуля появляется «Панель управления импортами».

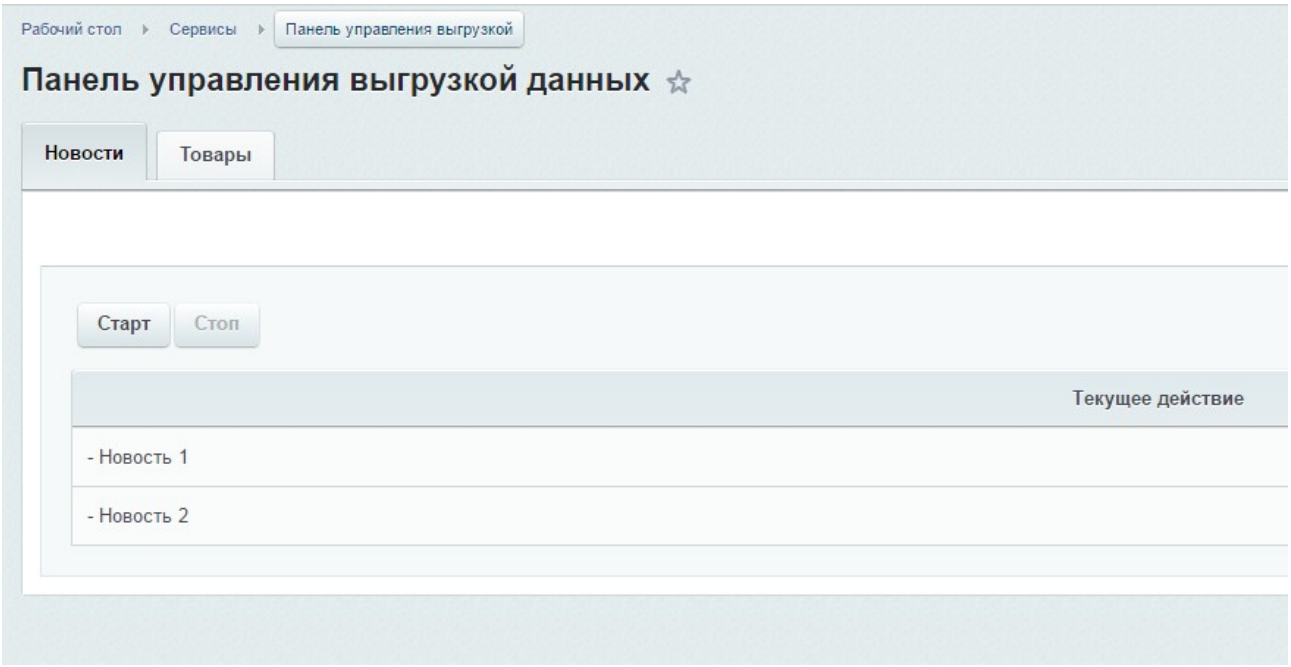
Она может быть применена при решении следующих задач:

- Перенос данных из БД текущего сайта;
- Импорт данных из csv, xls, xml.

Основные достоинства сервиса:

- При импорте данных на сайт снижает нагрузку на сервер (используется технология Ajax);
- Сокращение время на работы программиста (функционал сам реализует всю интерфейсную часть, программист только программирует логику).

Пример работы доступен по ссылке.



Установка и настройка

1. Для того, чтобы в БО (на вкладке «Сервисы») появился пункт меню, необходимо раскомментировать файл, находящийся по пути «/local/modules/indi.main/admin.menu.php».

Принцип работы

Скрипт через ajax обращается сам к себе и делает какую-то работу, возвращая get-строку для

следующего этапа, или ничего не возвращая. За основу взято данное решение.

Работа сервиса базируется на трех основных методах класса \Indi\Main\Import.php.

1. **getImportObject()**

Функция получает массив из выгрузки в следующем формате:

```
$importObject = array(
   array(
      "name" => "Новость 1",
      "code" => "New1",
      "picture" => "/upload/images/1.jpg"
   ),
   array(
      "name" => "Новость 2",
      "code" => "New1",
      "picture" => "/upload/images/3.jpg"
   )
);
```

Требования к массиву:

- Элементы массива должны быть также массивами, содержащими ключ «NAME» или «name»;
- Массив должен быть «проиндексированным» (см. пример выше), допустимы только числовые ключи 0,1,2,3....n

2. **importElement()**

Описывает логику непосредственного импорта 1 элемента массива, который возвращает метод getImportObject. На вход поступает массив с данными об одном элементе импорта, далее программист реализует любую бизнес-логику по импорту этих данных.

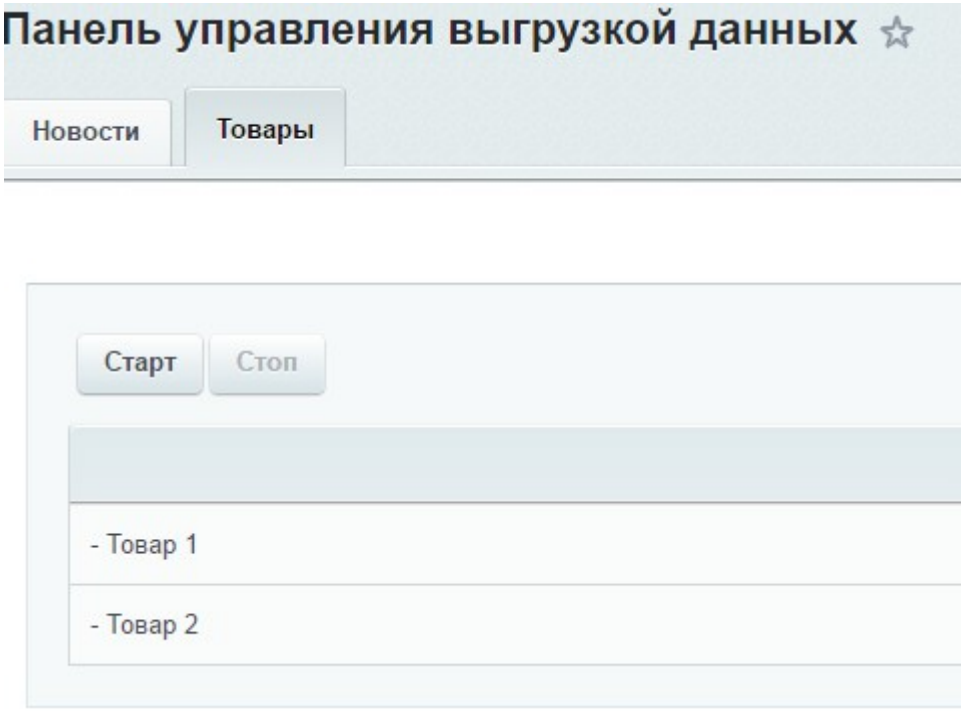
3. **importEpilog()**

Описывает действия которые должны быть выполнены после окончания импорта. Например, удаление всех записей, которые есть в инфоблоке, но нет в файле выгрузки.

Для облегчения понимания в дистрибутив модуля включены заготовки для описаных функций.

Дополнительные возможности

Для удобства импорты можно разбивать по вкладкам и запускать независимо друг от друга.



Настройка вкладок доступна в файле /local/modules/indi.main/admin/indi_data_import.php.

Для облегчения понимания в дистрибутиве модуля созданы в заготовки для вкладок панели.