

1С-Битрикс: Управление сайтом 6.x

Руководство по использованию технологии AJAX



1С-БИТРИКС

Содержание

Введение	3
Уровень компонентов	4
Как это работает?.....	4
Что нужно сделать, чтобы это заработало для моих компонентов?.....	5
Локальный уровень	7
Объект jsAjaxUtil	8
Класс CAjax	11
Класс CAjaxForm.....	11
Объект jsStyle	12
Объект jsEvent	12
API сервера	14
Tips&Tricks. Кастомизируем визуальные эффекты	16

Введение

Технология AJAX в системе «1С-Битрикс: Управление сайтом» реализована на двух уровнях:

- § Локальный уровень – это ситуации, когда AJAX-запросы инициируются клиентскими сценариями на странице или внутри компонента, и обработчик AJAX-запроса самостоятельно обрабатывает полученные данные, например, обновляет некоторую рабочую область страницы.
- § Уровень компонентов – это ситуация, когда AJAX-запросы инициируются клиентскими сценариями, созданными ядром продукта, и обработчик AJAX-запроса обновляет рабочую область, занимаемую компонентом.

Уровень компонентов

Технология AJAX на уровне компонентов внешне работает очень просто: в параметрах компонента выставляется параметр 'AJAX_MODE' => 'Y', и компонент работает без перезагрузки страницы. Снаружи все просто, но рассмотрим этот вопрос подробно изнутри.

Как это работает?

Когда обработчик компонентов встречает на своем пути компонент с таким параметром, то создается экземпляр служебного класса **CComponentAjax**. Этот экземпляр вычисляет для компонента уникальный идентификатор сессии на основе результата функции **debug_backtrace()** (что поднимает планку требуемой версии PHP до 5.0.0). Далее, перехватывается вывод компонента и следующим образом обрабатывается:

1. Прежде всего, все ссылки, встреченные внутри компонента, преобразуются в AJAX-вызовы. То есть, если в выводе встречается строка вида

```
<a href="my.php?ID=123&action=display">Посмотреть</a>
```

она заменяется на следующую:

```
<a href="my.php?ID=123&action=display"
onclick="jsAjaxUtil.InsertDataToNode('my.php?ID=123&action=display&bxajaxid=1ecd88a66ec8f6aca9f700e8f0e7ef13', 'comp_1ecd88a66ec8f6aca9f700e8f0e7ef13', true);
return false;">Посмотреть</a>
```

Мы имеем ту же самую ссылку, что и была, только с некоторыми «вкраплениями». Строка **1ecd88a66ec8f6aca9f700e8f0e7ef13** – это и есть тот самый уникальный идентификатор сессии; **comp_1ecd88a66ec8f6aca9f700e8f0e7ef13** – это идентификатор контейнера, в который заключается вывод компонента, и куда мы будем вставлять результат AJAX-запроса. Про метод `jsAjaxUtil.InsertDataToNode()` речь пойдет ниже.

При этом игнорируются некоторые ссылки:

- § ссылки, ведущие куда-либо, кроме текущей страницы или на адрес URL, который будет обрабатываться чем-либо, кроме текущего скрипта (для режима ЧПУ);
- § ссылки, содержащие атрибут `onclick`;
- § ссылки, содержащие атрибут `target`.

Все остальные атрибуты ссылки остаются неизменными.

2. Аналогично все формы, встречающиеся в выводе, также преобразуются к AJAX-виду:

```
<form name="snmf" action="index.php" method="POST" enctype="multipart/form-data">
```

превращаются в

```
<form name="snmf" action="index.php" method="POST" enctype="multipart/form-data"
onsubmit="return jsAjaxUtil.InsertFormDataToNode(this, 'comp_1ecd88a66ec8f6aca9f700e8f0e7ef13', true);">
<input type="hidden" name="bxajaxid" value="1ecd88a66ec8f6aca9f700e8f0e7ef13" />
```

Как конкретно происходит отправка формы, мы также поговорим ниже.

3. После этого, вывод «отпускается» и отправляется клиенту.


Когда клиент совершает переход по ссылке или отправляет форму, инициализируется AJAX-запрос к серверу на ту же самую страницу (другие ссылки игнорируются обработчиком), содержащий дополнительные параметры, принадлежащие компоненту, а также, содержащий параметр **bxajaxid** (идентификатор сессии). Начальная цепочка обработки начинается снова, за исключением того факта, что у нас уже есть идентификатор компонента, который инициализировал запрос. Когда обработчик встречает компонент, идентификатор которого не совпадает с переданным, он его игнорирует. Самое интересное начинается, когда находится соответствие. Обработчик очищает весь накопленный к тому моменту вывод и совершает над результатом компонента все вышеперечисленные действия. Затем, к выводу компонента прикрепляется служебный «пакет», содержащий:

- § новый заголовок страницы;
- § список клиентских сценариев, подключаемых компонентом при помощи **\$APPLICATION->AddHeadScript()**;
- § список файлов стилей, подключаемых компонентом.

Отдельно обрабатываются ситуации, когда в компоненте выполняется **RestartBuffer** или **LocalRedirect**.

После выполнения этих операций обработка страницы прерывается, вызывается **epilog_after.php** и результат отправляется клиенту в качестве ответа.

4. Менеджер AJAX, работающий на клиенте принимает запрос и, в свою очередь, обрабатывает его. Из результата аккуратно выделяются все строки, содержащие клиентские сценарии. Очищенный вывод помещается в контейнер, содержащий компонент, замещая собой текущую страницу компонента. Вырезанные сценарии выполняются в контексте окна. Переданный список дополнительных файлов со стилями и сценариями также обрабатывается и прикрепляется к документу. Заодно, заменяется заголовок окна документа.

 **Внимание!** Для того чтобы можно было заменить заголовок на странице, пришлось ввести стандарт: элемент шаблона сайта, содержащий заголовок, должен иметь идентификатор **pagetitle**. Если элемента с таким идентификатором на странице не найдено, будет обновлен только заголовок окна.

Что нужно сделать, чтобы это заработало для моих компонентов?

Прежде всего, проверить, как работает компонент в режиме AJAX, вставив параметр **'AJAX_MODE' => 'Y'** на странице с вызовом компонента. Особенно критична проверка, если в компоненте присутствуют какие-либо сложные клиентские сценарии. Здесь могу привести несколько рекомендаций:

1. не использовать в сценариях компонента `document.write()` в текущее окно;
2. следить за уникальностью CSS-стилей на разных страницах комплексного компонента. Это указано в руководстве по созданию компонентов 2.0, но все-таки повторю: стили, общие для страниц комплексного компонента, должны быть вынесены в стили компонента. Стили остальных страниц должны быть непересекающимися и используемыми только этой страницей.

3. следить за порядком выполнения клиентских сценариев компонента. Сценарии выполняются в строгом порядке, описанном ниже.
4. стоит делать так, чтобы ссылки, ведущие на страницы компонента, возвращающие не стандартный HTML-код, а, например, XML, не обрабатывались парсером. Типичный пример – ссылки, ведущие на RSS-ленту, создаваемую компонентом. То же самое стоит делать со ссылками, которые будут обработаны компонентом, но после обработки произойдет переадресация на другую страницу. Например, кнопка "купить" в компоненте `bitrix:catalog`. Это можно сделать, добавив ссылке атрибуты `target="_self"` или `onclick="void(0)"`.

После того, как проверка пройдена, нужно в файле `.parameters.php` компонента добавить в список параметров строку

```
'AJAX_MODE' => array()
```


После этого при настройке параметров компонента в визуальном редакторе или в режиме редактирования сайта появится дополнительная группа параметров «**Управление режимом AJAX**», содержащая следующие настройки:

- I Включить режим **AJAX** ('AJAX_MODE' => 'Y') – собственно, включение режима AJAX для компонента.
- I Включить затенение – если Вы смотрели, как работает компонент в режиме AJAX, то должны были заметить, что содержимое компонента накрывается «тенью». Она выполняет двоякую функцию: во-первых, показывает область, которая сейчас изменится, а, во-вторых, блокирует клики по этой области. Здесь ее можно включить. В коде страницы это выразится в появлении параметра `'AJAX_OPTION_SHADOW' => 'Y'`.
- I Включить прокрутку к началу компонента – когда пользователь совершает AJAX-переход, то по завершении загрузки происходит прокрутка к началу компонента. Это также можно включить: `'AJAX_OPTION_JUMP' => 'Y'`.
- I Включить подгрузку стилей – как уже было сказано выше, вместе с ответом компонента подгружается и обрабатывается список стилей, затребованных компонентом. Включение подгрузки стилей определяется параметром `'AJAX_OPTION_STYLE' => 'Y'`.
- I Включить эмуляцию навигации браузера – когда пользователь выполняет AJAX-переходы, то при включенной опции можно использовать кнопки браузера «Вперед» и «Назад» (данная возможность пока работает только в IE и Mozilla Firefox). Включение опции определяется параметром `'AJAX_OPTION_HISTORY' => 'Y'`.

Теперь опишем базу этого механизма, которая, по сути, является *Локальным уровнем* технологии AJAX.

Локальный уровень

Локальный уровень технологии AJAX в «1С-Битрикс: Управление сайтом» представлен клиентской библиотекой, которая находится в файле `/bitrix/js/main/ajax.js`, и несколькими серверными методами для удобства использования. Распишем несколько функций, предоставляемых этой библиотекой.

 **Внимание!** При выполнении AJAX-запросов любым из нижеперечисленных способов из результата автоматически вырезаются и **выполняются** в контексте окна клиентские сценарии. Обработчики получают уже очищенный вывод. Если в сценариях ответа создаются обработчики события **window.onload**, они также выполняются.

Вообще, обработка ответа происходит в такой последовательности:

- § менеджер AJAX получает ответ сервера;
- § ответ обрабатывается, вырезаются сценарии;
- § результирующий ответ передается обработчику запроса;
- § выполняется подключение всех внешних файлов сценариев;
- § выполняется подключение дополнительных стилей и файлов сценариев (для обработки уровня компонентов);
- § выполняются сценарии, переданные непосредственно в коде ответа;
- § выполняются новые обработчики события `window.onload`.

Объект jsAjaxUtil

jsAjaxUtil.InsertDataToNode(string url, string|object DOMNode container_id, bool bShadow)

jsAjaxUtil.AppendDataToNode(string url, string|object DOMNode container_id, bool bShadow)

Два аналогичных метода, реализующих загрузку данных с адреса **url** и вставку их в элемент с идентификатором **container_id**. Вместо идентификатора элемента допустимо передавать ссылку на объект требуемого узла DOM-структуры. Первый метод заменяет содержимое элемента новыми данными, второй – добавляет новые данные к имеющемуся содержимому элемента. Флаг **bShadow** показывает, вызывать ли затенение элемента. Подробнее см. [объект CAjax](#).

Пример:

```
<a class="news-more-link" href="/news/news.php?ID=104"
onclick="jsAjaxUtil.InsertDataToNode('/news/news.php?ID=104&ajax=Y',
'news_body_104', true); return false;">подробнее...</a>
<div id="news_body_104" class="news-body">текст новости откроется здесь</div>
```

string jsAjaxUtil.LoadData(string url, object Function obHandler)

string jsAjaxUtil.PostData(string url, object arData, object Function obHandler)

Два метода, дающих расширенные возможности контроля над результатом запроса. Первый метод осуществляет AJAX-запрос к адресу **url** и передает результат объекту **obHandler**. Второй осуществляет отправку данных, хранящихся в ассоциативном массиве **arData** на адрес **url** методом POST. Результат передается обработчику **obHandler**. Никаких визуальных эффектов, сопровождающих запрос, не выводится. Методы возвращают уникальный идентификатор запроса. Подробнее см. [объект CAjax](#).

Пример:

```
<script type="text/javascript">
function run()
{
jsAjaxUtil.ShowLocalWaitWindow('wait_id', 'ajax_container');
  jsAjaxUtil.LoadData('test.php?' + Math.random(), PutData);
}
function PutData(data)
{
  var obContainer = document.getElementById('ajax_container');
  var obDiv = obContainer.appendChild(document.createElement('DIV'));
obDiv.style.backgroundColor = '#40FF04';
obDiv.style.paddingBottom = '3px';
obDiv.appendChild(document.createTextNode(data));
  jsAjaxUtil.CloseLocalWaitWindow('wait_id', obContainer);
}
}
```



```

</script>
<a href="javascript:void(0)" onclick="run()">Загрузить</a>
<div id="ajax_container" style="height: 200px; width: 400px; border: solid 1px
#909090; overflow: auto;></div>

```

bool **jsAjaxUtil.InsertFormDataToNode**(*string|object Form* obForm, *string|object DOMNode* container_id, *bool* bShadow)

bool **jsAjaxUtil.AppendFormDataToNode**(*string|object Form* obForm, *string|object DOMNode* container_id, *bool* bShadow)

Два метода, осуществляющих динамическую отправку формы **obForm** с отображением результата в контейнере **container_id**. Вместо идентификатора формы или идентификатора элемента допустимо передавать ссылку на объект требуемого узла **DOM-структуры**. Первый метод заменяет содержимое элемента новыми данными, второй – добавляет новые данные к имеющемуся содержимому элемента. Флаг **bShadow** показывает, вызывать ли затенение элемента. Методы всегда возвращают **true**. Подробнее см. [класс CAjaxForm](#).

Использовать методы стоит следующим образом:

```

<form name="snmf" action="index.php" method="POST" enctype="multipart/form-data"
onsubmit="return MyFormCheck() && jsAjaxUtil.InsertFormDataToNode(this, 'result',
true);">

```

В примере **MyFormCheck()** - это собственный метод, осуществляющий валидацию полей формы.

bool **jsAjaxUtil.SendForm**(*string|object Form* obForm, *object Function* obHandler)

Метод осуществляет динамическую отправку формы **obForm** с передачей результата обработчику **obHandler**. Методу можно передавать как идентификатор формы, так и ссылку на объект формы в документе. Никаких визуальных эффектов, сопровождающих запрос, не выводится. Метод всегда возвращает **true**. Использование аналогично предыдущему. Подробнее см. [класс CAjaxForm](#).

jsAjaxUtil.RemoveAllChild(*object DOMNode* pNode)

Удаление всех дочерних узлов DOM-узла **pNode**.

jsAjaxUtil.EvalGlobal(*string* script)

Выполнение строки **script** в контексте окна (объекта **window**).

jsAjaxUtil.EvalExternal(*string* script_src)

Подключение внешнего файла сценария, расположенного по ссылке **script_src**. Проверяется уникальность списка загруженных файлов – один и тот же сценарий повторно грузиться не будет.

string **jsAjaxUtil.urlencode**(*string* str)

Аналог функции **urlencode** из PHP.

string **jsAjaxUtil.trim**(*string* str)

Аналог функции *trim* из PHP.

object **jsAjaxUtil.GetRealPos**(*object DOMNode* obNode)

Получение координат узла DOM-структуры относительно окна. Результатом является ассоциативный массив (объект), содержащий свойства **top**, **bottom**, **left**, **right**.

bool **jsAjaxUtil.IsIE**()

Определяет, загружена ли страница в браузере Internet Explorer.

bool **jsAjaxUtil.IsOpera**()

Определяет, загружена ли страница в браузере Opera.

string **jsAjaxUtil.GetStyleValue**(*object DOMNode* obElement, *string* property)

Получение значения CSS-свойства property элемента **obElement**. Учитываются как inline-стили, так и стили, создаваемые CSS-таблицей.

jsAjaxUtil.ShowLocalWaitWindow(*string* TID, *string|object DOMNode* obContainer, *bool* bShadow)

Вывод визуализации аяксового запроса. **TID** – уникальный идентификатор, **obContainer** – идентификатор целевого узла DOM-структуры или ссылка на него. **bShadow** – флаг, обозначающий, показывать ли затенение на целевом узле. Выводится пиктограмма в левом верхнем углу элемента, соответствующего целевому узлу, элемент затеняется (если bShadow = true).

jsAjaxUtil.CloseLocalWaitWindow(*string* TID, *string|object DOMNode* obContainer)

Убирает визуальные эффекты, созданные **jsAjaxUtil.ShowLocalWaitWindow()**. Эффекты также убираются нажатием кнопки Esc. Если в этот момент происходил AJAX-запрос, он также отменяется.


jsAjaxUtil.UpdatePageData(*object* arData)

Обновление параметров страницы на основе ассоциативного массива **arData**. Элементы массива:

- § **TITLE** – новый заголовок страницы. Для того чтобы можно было заменить заголовок на странице, пришлось ввести стандарт: элемент шаблона сайта, содержащий заголовок, должен иметь идентификатор **pagetitle**. Если элемента с таким идентификатором на странице не найдено, будет обновлен только заголовок окна.
- § **CSS** – массив ссылок на файлы стилей, которые требуется загрузить на страницу (см. [объект jsStyle](#))
- § **SCRIPTS** – массив ссылок на внешние файлы сценариев, которые нужно загрузить на страницу и выполнить (см. **jsAjaxUtil.EvalExternal()**).

Класс CAjax.

Класс CAjax – это менеджер AJAX-запросов. По умолчанию, создается экземпляр класса с именем **jsAjax**, который и следует использовать. Кроме того, выше описаны методы InsertDataToNode, AppendDataToNode, LoadData, PostData объекта jsAjaxUtil, которые составляют интерфейс, позволяющий решить большинство требуемых задач без прямого обращения к объекту класса.

 **Внимание!** Из результата автоматически вырезаются и выполняются в контексте окна клиентские сценарии. Обработчики получают уже очищенный вывод. Если в сценариях ответа создаются обработчики события **window.onload**, они также выполняются.

string **CAjax.InitThread()** – инициализация AJAX-запроса. Метод генерирует и возвращает идентификатор запроса, а также создает экземпляры классов CAjaxThread и XMLHttpRequest.

CAjax.AddAction(*string* TID, *object Function* obHandler) - добавление обработчика результата AJAX-запроса. Метод назначает обработчик результата запросу с идентификатором TID.

object CAjaxThread **CAjax.GetThread**(*string* TID) – метод возвращает ссылку на экземпляр класса CAjaxThread, соответствующий AJAX-запросу с идентификатором TID.

CAjax.Send(*string* TID, *string* url, *object* arData) – отправка AJAX-запроса методом GET с идентификатором TID на адрес url. Свойства объекта arData прикрепляются к параметрам url в виде параметров GET-запроса.

CAjax.Post(*string* TID, *string* url, *object* arData) – отправка POST-данных AJAX-запросом с идентификатором TID на адрес url. Свойства объекта arData используются в качестве полей POST-данных.

Класс CAjaxForm.

Основная задача класса CAjaxForm – это преобразование формы для динамической отправки. Преобразование заключается в следующем: на странице создается скрытый плавающий фрейм, на него перенаправляется отправка формы. Также, к форме добавляется скрытый параметр **AJAX_CALL=Y**. Результат берется из фрейма и передается обработчику. Единственное условие — результат не должен быть пустым. При работе на уровне компонентов, из результата *на сервере* вырезаются все клиентские сценарии (чтобы избежать выполнения сценариев в контексте окна фрейма), которые собираются в единый пакет, отправляемый серверу.

Выше описаны методы InsertFormDataToNode, AppendFormDataToNode, SendForm объекта jsAjaxUtil, которые составляют интерфейс, позволяющий решить большинство требуемых задач, минуя непосредственную работу с объектами класса CAjaxForm.

CAjaxForm(*object Form* obForm, *object Function* obHandler, *bool* bFirst) – конструктор класса. Входные параметры: **obForm** – объект формы, **obHandler** – функция-обработчик результата, **bFirst** – флаг, означающий, должно ли преобразование формы быть перманентным, или должно быть отменено после первого вызова.

CAjaxForm.process() – подготовка формы, указанной в конструкторе экземпляра класса, к динамической отправке.

CAjaxForm.setProcessedFlag(*bool* flag) – установка для формы флага «обработана».

bool **CAjaxForm.isFormProcessed(*object Form* obForm)** – проверка флага «обработана» для формы obForm.

Объект jsStyle

Объект jsStyle представляет собой менеджер стилей документа. Новые стили добавляются с минимальным приоритетом и могут быть перезаписаны стилями, ранее подключенными при помощи менеджера или в коде страницы. Повторно файлы стилей не загружаются, а включаются с тем же приоритетом, что и загруженные в первый раз (см. также jsAjaxUtil.UpdatePageData()).

jsStyle.Load(*string* cssurl) – загрузка и подключение к документу внешнего файла стилей, расположенного по адресу cssurl.

jsStyle.Unload(*string* cssurl) – отключение файла стилей cssurl. **Внимание!** Некоторые браузеры обновляют отображение документа только при динамическом изменении или при добавлении новых стилей.

jsStyle.UnloadAll() – отключение всех динамически подгруженных файлов стилей. **Внимание!** Некоторые браузеры обновляют отображение документа только при динамическом изменении или при добавлении новых стилей.

Объект jsEvent

Объект jsEvent представляет собой кроссбраузерный менеджер событий. При назначении событию объекта нового обработчика, все имеющиеся обработчики данного события также включаются в список. Идентификаторы событий указываются без префикса 'on', т.е., для того чтобы добавить свой обработчик MyOnload на событие onload окна, нужно вызвать **jsEvent.addEvent(*window*, 'load', MyOnload)**. В отличие от браузерных методов, attachEvent/addEventListener менеджер выполняет обработчики событий строго в том порядке, в каком они были назначены. Если у объекта уже есть обработчики, назначенные при помощи атрибута или сценария, они будут поставлены в начало списка выполнения и также выполнятся при инициализации события.

jsEvent.addEvent(*object* obElement, *string* event, *object Function* obHandler) – назначение обработчика **obHandler** событию **event** объекта **obElement**.

jsEvent.removeEvent(*object* obElement, *string* event, *object Function* obHandler) – удаление обработчика **obHandler** для события **event** объекта **obElement**.

jsEvent.removeAllHandlers(*object* obElement, *string* event) – удаление всех обработчиков события **event** объекта **obElement**. **Внимание!** Обработчики удаляются только в том случае, если объекту был назначен хоть один обработчик средствами объекта **jsEvent**.

jsEvent.removeAllEvents(*object* obElement) – удаление всех обработчиков всех событий объекта **obElement**. **Внимание!** Обработчики удаляются только для тех событий, для которых объекту был назначен хоть один обработчик средствами объекта **jsEvent**.

jsEvent.clearObject(*object* obElement) – удаление всей известной менеджеру информации объекте **obElement**. Рекомендуется вызывать перед удалением объекта.

Пример:

```
<script type="text/javascript">
function Greetings()
{
    alert('Greetings, visitor!');
}
function ChangeBGColor()
{
    document.body.style.backgroundColor = 'rgb(' + parseInt(Math.random() * 256) + ', '
+ parseInt(Math.random() * 256) + ', ' + parseInt(Math.random() * 256) + ')';
}
jsEvent.addEvent(window, 'load', Greetings);
jsEvent.addEvent(window, 'load', ChangeBGColor);
</script>
```

API сервера

На сервере доступно несколько дополнительных средств для работы с AJAX:

1. Константа **BX_AJAX_PARAM_ID** содержит название **REQUEST**-параметра, используемого в качестве идентификатора AJAX-запроса.
2. Стилизовое оформление визуальных эффектов AJAX настроено в файле **/bitrix/templates/default/ajax/ajax.css** и, соответственно, может быть переопределено для любого шаблона сайта.
3. **CAjax::Init()** – подключение клиентской библиотеки и требуемых ей файлов стилей.
4. *bool|string* **CAjax::GetSession()** – получение значения AJAX-идентификатора сессии. Если такого нет, функция возвращает **false**.
5. *string* **CAjax::GetSessionParam([string|bool \$ajax_id = false])** – получение строки вставки идентификатора AJAX-сессии в URL. Если идентификатор не передан в параметрах функции, он берется из параметров запроса к текущей странице. Если идентификатор отсутствует, функция возвращает пустую строку.
6. *string* **CAjax::AddSessionParam(string \$url[, string|bool \$ajax_id = false])** – вставка идентификатора AJAX-сессии в URL. Если идентификатор не передан в параметрах функции, он берется из параметров запроса к текущей странице. Если идентификатор отсутствует, функция возвращает **\$url** без изменений. Если в запросе к текущей странице присутствует флаг динамически отправленной формы (**AJAX_CALL=Y**), он также будет добавлен к **\$url**.
7. *string* **CAjax::GetLink(string \$url, string \$text, string \$container_id[, string \$additional = "", bool \$bReplace = true, bool \$bShadow = true])** – генерация тэга ссылки, для отправки посредством AJAX. Параметры: **\$url** – URL ссылки; **\$text** – текст ссылки (будет пропущен через **htmlspecialchars**); **\$container_id** – идентификатор контейнера на странице, куда должны быть вставлены полученные с сервера данные; **\$additional** – дополнительные атрибуты ссылки; **\$bReplace** – флаг, показывающий, должно ли содержимое контейнера замениться новыми данными, или добавиться к ним; **\$bShadow** – флаг, показывающий, должен ли контейнер накрываться «затенением».

Пример:

```
<?=CAjax::GetLink('script.php', 'Link "TEST"', 'ajax_container');?>
```

Результат:

```
<a href="script.php" onclick="jsAjaxUtil.InsertDataToNode('script.php', 'ajax_container', true); return false;">Link &quot;TEST&quot;</a>
```

8. *string* **CAjax::GetLinkEx(string \$real_url, string \$public_url, string \$text, string \$container_id[, string \$additional = "", bool \$bReplace = true, bool \$bShadow = true])** – расширенная версия функции **CAjax::GetLink()**. Параметры: **\$real_url** – URL, по которому пойдет AJAX-запрос; **\$public_url** – URL, который будет записан в атрибут **href** ссылки; **\$text** – текст ссылки (будет вставлен непосредственно, без **htmlspecialchars**); **\$container_id** – идентификатор контейнера на странице, куда должны быть вставлены полученные с сервера данные; **\$additional** – дополнительные атрибуты ссылки; **\$bReplace** – флаг, показывающий, должно ли содержимое контейнера замениться новыми данными, или добавиться к ним; **\$bShadow** – флаг, показывающий, должен ли контейнер накрываться «затенением».

Пример:

```
<?=CAjax::GetLinkEx('script.php?AJAX=Y', 'script.php', 'Link
&quot;<b>TEST</b>&quot;', 'ajax_container');?>
```

Результат:

```
<a href="script.php" onclick="jsAjaxUtil.InsertDataToNode('script.php?AJAX=Y',
'ajax_container', true); return false;">Link &quot;<b>TEST</b>&quot;</a>
```

9. *string CAjax::GetForm(string \$form_params, string \$container_id, string \$ajax_id[, bool \$bReplace = true, bool \$bShadow = true])* – получение тега формы, который будет отправлен на сервер без перезагрузки страницы. Параметры: *\$form_params* – строка, содержащая атрибуты тега формы; *\$container_id* – идентификатор контейнера на странице, куда должны быть вставлены полученные с сервера данные; *\$ajax_id* – идентификатор AJAX-сессии; *\$bReplace* – флаг, показывающий, должно ли содержимое контейнера замениться новыми данными, или добавиться к ним; *\$bShadow* – флаг, показывающий, должен ли контейнер накрываться «затенением».

Пример:

```
<?=CAjax::GetForm('name="ajaxform3" action="formtest.php" method="POST"
enctype="multipart/form-data"', 'formtest3', '1')?>
```

Результат:

```
<form name="ajaxform3" action="formtest.php" method="POST"
enctype="multipart/form-data" onsubmit="return
jsAjaxUtil.InsertFormDataToNode(this, 'formtest3', true);"><input type="hidden"
name="bxajaxid" value="1" />
```

Если атрибуты формы содержат обработчик события **onsubmit**, то это будет учтено:

```
<?=CAjax::GetForm('name="ajaxform3" action="formtest.php" method="POST"
enctype="multipart/form-data" onsubmit="return CheckForm()", 'formtest3', '1')?>
```

Результат:

```
<form name="ajaxform3" action="formtest.php" method="POST"
enctype="multipart/form-data" onsubmit="return CheckForm() &&
jsAjaxUtil.InsertFormDataToNode(this, 'formtest3', true)"><input type="hidden"
name="bxajaxid" value="1" />
```

10. *string CAjax::GetFormEvent(string \$container_id[, bool \$bReplace = true, bool \$bShadow = true])* – получение атрибута **onsubmit** тега формы для динамической отправки. Параметры: *\$container_id* – идентификатор контейнера на странице, куда должны быть вставлены полученные с сервера данные; *\$bReplace* – флаг, показывающий, должно ли содержимое контейнера замениться новыми данными, или добавиться к ним; *\$bShadow* – флаг, показывающий, должен ли контейнер накрываться «затенением».

Tips&Tricks. Кастомизируем визуальные эффекты

Как уже было сказано ранее, стили визуальных эффектов AJAX располагаются в каталоге `/bitrix/templates/.default/ajax`. Соответственно, для простой модификации цветовой схемы достаточно скопировать этот каталог в каталог нужного шаблона и исправить цвета или заменить файлы изображений своими собственными. Мы же рассмотрим менее тривиальный вариант. Реализуем визуальный эффект AJAX-перехода в виде затенения всей страницы с выводом сообщения о загрузке в виде блока, расположенного посередине страницы.

Для этого, скопируем каталог с шаблонами дизайна в шаблон, для которого мы хотим модифицировать вывод. В примере это будет шаблон `web20` стандартной поставки. В каталог `ajax/images` скопируем картинку [progress.gif](#). Затем модифицируем файл `ajax/ajax.css` следующим образом:

```
iframe.waitwindowlocal { position:absolute; border:0px; z-index:9999; }
div.waitwindowlocal { position:absolute; width:180px; padding:25px 5px 10px 5px; z-
index:10000; background-color:#DADADA; border:1px solid #666666; background-
image:url(images/progress.gif); background-position:center 7px; background-repeat:no-
repeat; text-align:center; font-weight:bold; color:#666666; font-size:9px; }
div.waitwindowlocalshadow { position:absolute; top:0px; left:0px; height:100%; width:
100%; z-index:9998; background-color:white; }
div.waitwindowlocalshadow { opacity:0.5; -moz-opacity:0.5; -khtml-opacity:0.5;
filter:progid:DXImageTransform.Microsoft.Alpha(opacity=50); }
```

Теперь очередь за кастомизацией сценария. Создадим каталог `/bitrix/templates/web20/js` и поместим в него файл `customize_ajax.js` со следующим содержанием:

```
if (window.jsAjaxUtil)
{
    // переопределим метод jsAjaxUtil.ShowLocalWaitWindow()
    jsAjaxUtil.ShowLocalWaitWindow = function (TID, cont, bShadow)
    {
        if (typeof cont == 'string' || typeof cont == 'object' && cont.constructor ==
String)
            var obContainerNode = document.getElementById(cont);
        else
            var obContainerNode = cont;

        if (null == bShadow) bShadow = true;
        var container_id = obContainerNode.id;

        if (bShadow)
        {
            // если нужно отображать затенение - отобразим
```



```

        var obWaitShadow =
document.body.appendChild(document.createElement('DIV'));
        obWaitShadow.id = 'waitshadow_' + container_id + '_' + TID;
        obWaitShadow.className = 'waitwindowlocalshadow';

        if (jsAjaxUtil.IsIE())
        {
            // для MSIE – раскроем тень по всей высоте содержимого окна
            obWaitShadow.style.height = document.body.scrollHeight + 'px';
        }
        else
        {
            // для остального сделаем ее фиксированной
            obWaitShadow.style.position = 'fixed';
        }
    }

    // создадим сообщение о загрузке
    var obWaitMessage =
document.body.appendChild(document.createElement('DIV'));
    obWaitMessage.id = 'wait_' + container_id + '_' + TID;
    obWaitMessage.className = 'waitwindowlocal';

    if (jsAjaxUtil.IsIE())
    {
        // для MSIE – разместим его посередине текущего отображаемого
контента окна
        var left = parseInt(document.body.scrollLeft +
document.body.clientWidth/2 - obWaitMessage.offsetWidth/2);
        var top = parseInt(document.body.scrollTop +
document.body.clientHeight/2 - obWaitMessage.offsetHeight/2);
    }
    else
    {
        // для остального сделаем ее посередине окна и с фиксированным
положением
        var left = parseInt(document.body.clientWidth/2 -
obWaitMessage.offsetWidth/2);
        var top = parseInt(document.body.clientHeight/2 -
obWaitMessage.offsetHeight/2);
        obWaitMessage.style.position = 'fixed';
    }
    obWaitMessage.style.top = top;

```

```

        obWaitMessage.style.left = left;

        // добавим текст
        obWaitMessage.innerHTML = 'Подождите, идет загрузка...';

        if(jsAjaxUtil.IsIE())
        {
            // для IE6 и ниже создадим под сообщением плавающий фрейм
            var frame = document.createElement("IFRAME");
            frame.src = "javascript:";
            frame.id = 'waitframe_' + container_id + '_' + TID;
            frame.className = "waitwindowlocal";
            frame.style.width = obWaitMessage.offsetWidth + "px";
            frame.style.height = obWaitMessage.offsetHeight + "px";
            frame.style.left = obWaitMessage.style.left;
            frame.style.top = obWaitMessage.style.top;
            document.body.appendChild(frame);
        }

        // добавим обработчик нажатия клавиши Esc.
        function __Close(e)
        {
            if (!e) e = window.event
            if (!e) return;
            if (e.keyCode == 27)
            {
                jsAjaxUtil.CloseLocalWaitWindow(TID, cont);
                jsEvent.removeEvent(document, 'keypress', __Close);
            }
        }

        jsEvent.addEvent(document, 'keypress', __Close);
    }
}

```

Как видно, мы переопределяем метод **jsAjaxUtil.ShowLocalWaitWindow()** под свои цели. Поскольку общая структура элементов остается прежней (см. оригинальный **jsAjaxUtil.ShowLocalWaitWindow()**), то переопределять **jsAjaxUtil.CloseLocalWaitWindow** нам не требуется. Остается только подключить этот файл сценария в шаблон (файл **/bitrix/templates/web20/header.php**) так, чтобы он шел после подключаемых оригинальный библиотек AJAX, то есть, после строки

```
<?APPLICATION->ShowHeadScripts()?>
```

и можно посмотреть результат.

! *Внимание!* Если нам не нужно полное изменение **ShowLocalWaitWindow**, то можно поступить проще:

```
if (window.jsAjaxUtil)
{
    jsAjaxUtil._ShowLocalWaitWindow = jsAjaxUtil.ShowLocalWaitWindow;
    jsAjaxUtil.ShowLocalWaitWindow = function (TID, cont, bShadow)
    {
        if (null == bShadow) bShadow = true;
        jsAjaxUtil._ShowLocalWaitWindow(TID, cont, bShadow);
        /*
            И затем производим любые манипуляции с уже созданными объектами
        */
    }
}
```

Например, можно совершить следующие кастомизации:

- § замена идущей в поставке продукта "тени" на полупрозрачное затемнение;
- § изменение цветов индикатора аjax-загрузки и добавление к нему сообщения;
- § изменение стиля курсора мыши на время аjax-запроса;
- § визуализация процесса загрузки в виде «бегающей» за курсором анимированной картинки в стиле некоторых тем Windows.