

Центр информационных технологий index.art

РУКОВОДСТВО ПРОГРАММИСТА
по конфигурированию и модификации
системы index.crm

редакция 1.5

Екатеринбург, 2006-2011

Содержание

Общая структура системы index.crm	3
ЧАСТЬ 1 АРХИТЕКТУРА ЯДРА СИСТЕМЫ	3
Движок Центра информационных технологий index.art, версия 3.0	3
Структура хранения информации	4
Синтаксис шаблонов программных модулей.....	5
Типы модулей.....	5
Теги уровня класса	5
Теги уровня функции	5
Библиотека классов движка	6
Authorisator	7
BindManager	8
Bookmarks.....	8
ConstantPointer	9
FieldEditor	9
GroupOperations.....	9
TableEditor	10
Общие функции	12
Основные сведения о ядре index.crm	14
Структура системы	14
Точки входа	14
Основные модули системы	14
Стандартные интерфейсы	16
Диалоговые окна	17
Программные средства index.crm	20
Система разграничения прав доступа	20
Система печати (экспорта в Microsoft Excel).....	20
Органы управления	21
Сортировка таблиц	21
Список и дерево	23
Выпадающее меню.....	24
Календарь.....	25
Дополнительные программные средства.....	25
UserSettings	26
Workflow	26
Security.....	27
Функции SQL	27
ЧАСТЬ 2 СОЗДАНИЕ МОДУЛЕЙ ДЛЯ INDEX.CRM	29
Конфигурации index.crm для конкретных заказчиков	29
Формирование конфигурации.....	29
Регламент обновления	29
Пакеты исправления ошибок.....	31
Базовые классы типов модулей	31
Справочники	32
Журналы документов.....	33
Отчеты	33
Создание модуля системы	34
Настройка модуля	34
Конструктор	34
ManagementForm	35
DisplayNavigation	36
Создание модуля интерфейса администратора.....	38
Пример создания конфигурации для заказчика	39
Формулировка задачи.....	39
Создание журнала документов «Заявки»	39
Модификация справочника	41
Создание отчета	42
Формирование пакетов обновления	44

Общая структура системы index.crm

Система index.crm построена на версии 3.0 стандартного ядра («движка») для разработки веб-сайтов, созданного Центром информационных технологий index.art. Система написана на языке PHP5 (ядро поставляется в бинарном виде, шифрование осуществляется системой IonCube). В качестве сервера баз данных используется MySQL 5.0 или старше. Общую архитектуру системы можно представить в виде схемы основных программных компонентов:

Конфигурация для конкретного заказчика

- Модифицированные и дополнительные программные модули

Базовая версия index.crm

- базовые классы для модулей системы
- набор средств навигации, печати, импорта/экспорта данных
- стандартные пользовательские интерфейсы
- набор основных модулей системы
- общие системы (проверка прав доступа, планировщик и т.д.)

Движок index.art для разработки сайтов, версия 3.0

- поддержка программных модулей и шаблонов различного уровня, алгоритм формирования страниц (парсер);
- библиотека базовых классов.

Первая часть настоящего руководства посвящена описанию возможностей настройки системы, доступных при помощи Интерфейса администратора (без вмешательства в программный код системы). Вторая часть документа посвящена рассмотрению архитектуры и программных интерфейсов ядра системы. В третьей части рассматриваются вопросы создания конфигураций index.CRM с вмешательством в программный код системы.

ЧАСТЬ 1 АРХИТЕКТУРА ЯДРА СИСТЕМЫ

Движок Центра информационных технологий index.art, версия 3.0¹

Основные факты о движке:

- Для доступа ко всем страницам сайта используется один скрипт – index.php. Этот скрипт должен быть документом по умолчанию для сайта.
- Адресация к конкретным веб-страницам сайта происходит по номеру. При этом ссылка на какую-либо страницу выглядит так: /?id=123, где 123 – идентификатор страницы.
- Движок использует разделение кода и дизайна. Скрипты PHP, являющиеся частью сайта, не должны (кроме определенных исключительных случаев) содержать HTML-код внутри себя. Весь HTML-код хранится в базе данных в виде шаблонов.
- Для управления веб-сайтом используется специальный интерфейс, расположенный в каталоге /config. В контексте index.crm этот интерфейс называется «Интерфейс администратора». Составляющие этого интерфейса являются отдельными PHP-файлами и не используют собственно движок, хотя могут использовать общую библиотеку классов системы.
- При необходимости хранения в БД данных, имеющих иерархическую структуру, используется связка из двух таблиц. Одна таблица хранит контейнеры, и может содержать ссылки на другие записи этой же таблицы (т.е. является иерархической сама по себе).

¹ При создании сайтов в index.art сейчас используется новая версия движка, 4.0. Ее основные отличия – отсутствие специального интерфейса администратора в папке /config (управляющие модули в нем реализованы как обычные модули движка), использование человеко-читаемых URL (/main/index вместо /?id=123), расширенной библиотеки органов управления и т.д.

Другая таблица хранит дочерние элементы и может содержать только ссылки на контейнеры. С такой структурой иерархических таблиц работают все программные элементы, составляющие систему index.cfm.

Структура хранения информации

Основной единицей хранения информации в движке является **документ**. Документы объединены в иерархическую структуру **разделов**.

Раздел (таблица sections) – контейнер, содержащий другие разделы и/или документы. Используется для организации хранения документов и (в некоторых случаях) для формирования меню.

Документ (таблица documents) – с точки зрения пользователя, представляет собой веб-страницу.

Основные свойства документа:

Название	Поле БД	Назначение
Идентификатор	id	Идентификатор. Используется в качестве номера документа при формировании ссылок на него (ссылки вида /?id=123)
Раздел	section	Хранит ссылку на контейнер – раздел, в котором содержится документ.
Название	name	Наименование документа, которое используется для его идентификации в интерфейсе администратора
Заголовок	name_visible	Используется в качестве заголовка страницы (тег <title>)
Шаблон	template	Ссылка на шаблон (таблица templates)
Показывать в меню	active	Флаг, определяющий, нужно ли показывать документ в различных меню
Код	code	HTML-код содержимого документа

Шаблон документа (таблица templates) – HTML-код страницы, в который вставляется содержимое документа. Шаблон хранит общую структуру страницы, в которой специальными тегами ({#ELEM0} и {#ELEM1}) обозначены места, куда нужно вставить, соответственно, заголовок и тело документа. Возможна вставка и других свойств документа – это зависит от конфигурации сайта (в контексте index.cfm другие элементы документа не используются).

Как шаблон документа, так и код конкретного документа могут содержать ссылки на программные модули. Для удобства представления в визуальном редакторе ссылки на программные модули оформляются так: , где module – имя модуля. При формировании страницы движок сайта, обнаружив такое включение, загружает файл модуля (/include/module.ext), проверяет его тип, загружает шаблон программного модуля, и выполняет слияние данных, сгенерированных модулем, с шаблоном. Получившийся в результате слияния HTML-код вставляется в код страницы на место ссылки на программный модуль.

Шаблон программного модуля (таблица modules) представляет собой HTML-код фрагмента страницы, который должен являться результатом работы модуля. Этот тип шаблонов может содержать специфические управляющие теги, указываемые в фигурных скобках, интерпретируемые при построении страницы. Синтаксис этих тегов описан далее.

Интерфейс (таблица interfaces) – часто используемый фрагмент шаблона программного модуля. Включение интерфейса в шаблоне программного модуля аналогично использованию директивы include в программном коде.

Программный модуль движка версии 3.0 представляет собой файл, содержащий PHP-код модуля. Модуль оформляется в виде класса, различные методы которого вызываются при интерпретации шаблона модуля. Каждый метод может вернуть ассоциативный массив данных, которые будут использованы при интерпретации фрагмента шаблона, относящегося к данному методу. Например, в шаблоне имеется следующее выражение:

```
{#ShowClient}Клиент: {!client}{#/ShowClient}
```

В этом случае программный модуль должен содержать следующий метод:

```
function ShowClient() {
```

```
return Array("client"=>$this->client);
}
```

При интерпретации шаблона модуля будет вызван метод ShowClient(), который вернет массив данных; данные из этого массива будут использованы при интерпретации фрагмента шаблона, содержащегося между тегами {#ShowClient} {#/ShowClient}, и тег {!client} заменится на значение, переданное в массиве. Результатом выполнения этого модуля будет строка вида:

Клиент: ООО «Альфа»

Таким образом, общий процесс формирования страницы выглядит так:

1. Запрос свойств документа по id, переданному через запрос
2. Получение из БД шаблона документа
3. Слияние шаблона документа и тела документа
4. Проход по HTML-коду страницы, вызов всех встречающихся программных модулей, подстановка результатов их выполнения в код страницы.

Синтаксис шаблонов программных модулей

Типы модулей

Модуль содержится в файле имя_модуля.ext и может содержать:

1. Функцию, возвращающую код HTML в виде строки (версия 1.0)
2. Функцию, возвращающую массив переменных для слияния с шаблоном (версия 2.0)
3. Класс, содержащий переменные и методы для слияния с шаблоном (версия 3.0)

Модули первого типа не имеют шаблона (переключатель «имеет шаблон» в свойствах модуля в интерфейсе администратора должен быть снят). Модули второго типа имеют шаблон, в котором можно использовать теги уровня функции. Модули третьего типа имеют шаблон, в котором можно использовать теги уровня класса и уровня функции.

Теги уровня класса

Тег	Описание	Пример
{%имя_интерфейса}	Включение общего фрагмента шаблона («интерфейса»). Интерфейсы хранятся в таблице interfaces и представляют собой часто используемые фрагменты шаблонов. Действие данного тега аналогично директиве #include в C++: он обрабатывается на первом проходе интерпретатора, до всех остальных тегов.	{%print_form}
{#имя_метода} шаблон метода {#/имя_метода}	Вызывает метод «имя_метода» класса модуля. Метод должен возвращать массив переменных для слияния с шаблоном данного модуля, указанным между тегами вызова метода.	{#ShowName} {?auth}Пользователь: {!name}{?/auth} {#/ShowName}
На уровне класса допустимо также использование всех тегов уровня функции, кроме циклов (см. ниже). При этом для подстановки будут использованы переменные - члены класса модуля, объявленные как public.		

Теги уровня функции

Тег	Описание	Пример
{!имя_переменной}	Заменяется на значение переменной из массива, возвращенного функцией, или переменной - члена класса	{!name}

	(если вызван с уровня класса).	
<code>{?имя_переменной[условие значение]}содержимое{~/имя_переменной}</code> <code>[условие значение]</code> – необязательный фрагмент		
	Содержимое обрабатывается движком только в том случае, если определена переменная «имя_переменной». Если заданы дополнительные параметры – «условие» и «значение», то переменная проверяется на соответствие условию. Допустимые условия: <code>==</code> , <code>!=</code> , <code><=</code> , <code>>=</code> , <code><</code> , <code>></code> . Значение – буквы и цифры.	<code>{?error}</code> Ошибка! <code>{~/error}</code> <code>{?error=fatal}</code> Фатальная ошибка! <code>{~/error}</code> <code>{?count>1}</code> Найдено более одного элемента <code>{~/count}</code>
<code>{~имя_переменной}</code> содержимое <code>{~/имя_переменной}</code>	Содержимое обрабатывается движком только в том случае, если НЕ определена переменная «имя_переменной».	<code>{~auth}</code> Вы не авторизованы! <code>{~/auth}</code>
<code>{:цикл}шаблон{:~/цикл}</code>	«Шаблон» обрабатывается движком столько раз, сколько элементов содержится в массиве «цикл».	<code>{:items}{!name}
{:~/items}</code>
<code>{>функция[?параметр]}</code> шаблон <code>{~/функция}</code>	Движок вызывает функцию «функция», объявленную вне классов, передает ей необязательный параметр, и сливает возвращенный функцией массив с указанным шаблоном. Функции в качестве первого параметра передается id документа.	<code>{>date?now}</code> <code>{!year} – {!mon} – {!mday}</code> <code>{~/date}</code>
<code>{<модуль[?параметр]}</code> <code>{~/модуль}</code>	Движок загружает и выполняет зарегистрированный в системе модуль «модуль» (любого типа) и сливает его с собственным шаблоном модуля. Параметр, если указан, передается в модуль как второй параметр конструктора или функции.	<code>{<login}{~/login}</code>

Теги уровня функции допускают бесконечное вложение друг в друга.

Модуль первого или второго типа может отдать команду движку заменить его вызов на вызов другого модуля, вернув элемент массива "redirect", равный имени модуля, который следует вызвать.

Библиотека классов движка

Помимо компонентов, отвечающих за формирование страницы, движок содержит библиотеку базовых классов. Эти классы широко используются как в интерфейсе администратора, так и в клиентской части CRM-системы. Многие классы (например, BindManager) расширяются под нужды конкретных заказчиков.

В списке методов классов перечислены только те из них, которые имеют смысл применительно к CRM системе.

Для отображения пользовательского форм, которые генерируют некоторые из перечисленных классов, используются определенные интерфейсы index.crm. Если классу соответствует какой-либо интерфейс, в описании класса приводится ссылка на него. Многие классы используются также и в Интерфейсе администратора, поэтому содержат наряду с методом, подготавливающим данные для слияния с шаблоном, и метод, выводящий форму в виде HTML-кода. Эти методы не должны использоваться в пользовательской части index.crm.

Authorisator

Управляет авторизацией пользователей. В качестве логина при входе на сайт используется e-mail адрес.

Пользователи хранятся в таблице users. Каждый пользователь принадлежит к какой-либо группе (поле igrroup, ссылается на таблицу usergroups). В контексте index.cfm каждый пользователь относится также к какому-либо организационному подразделению, но класс Authorisator с этой привязкой не работает.

Необходимость в расширении этого класса вряд ли может возникнуть.

В index.cfm создается только один экземпляр этого класса, хранящийся в глобальной переменной \$auth. Его должны использовать все модули системы.

Функция	Описание
LastError()	Возвращает строку - описание последней происшедшей ошибки
Authorisator(\$authmode)	Конструктор. Значение параметра - AUTH_APACHE (авторизация будет производиться средствами Apache) или AUTH_SITE (авторизация средствами сайта, значение по умолчанию). AUTH_APACHE можно использовать только в том случае, если на самом сайте авторизация не нужна. В index.cfm по умолчанию используется метод AUTH_SITE, но, если имеются повышенные требования к безопасности, можно использовать и AUTH_APACHE.
AuthoriseAutomatic(\$email)	Автоматически авторизует пользователя после регистрации. Если пользователь с адресом email еще ни разу не входил на сайт, текущий пользователь будет авторизован как пользователь с адресом email. Проверка пароля не производится. В index.cfm не используется. Использовать с осторожностью.
Authorise()	Пытается авторизовать пользователя. В зависимости от выбранного типа авторизации использует либо имя пользователя, прошедшего авторизацию Apache, либо имя и пароль, содержащиеся в \$_REQUEST["username"] и \$_REQUEST["password"].
CheckAuthorisation(\$external=false)	Проверяет, авторизован ли пользователь. Возвращает true или false. Параметр должен быть установлен в true, если вызывается из системы управления (требуется пускать только пользователей внутренних групп).
GetUserID()	Возвращает идентификатор текущего пользователя
GetUserName()	Возвращает имя текущего пользователя
GetUserEmail()	Возвращает e-mail адрес текущего пользователя
GetUserGroup()	Возвращает id группы текущего пользователя
GetUserLastLogin()	Возвращает время последнего входа в систему текущего пользователя
GetUserPrevLogin()	Возвращает предыдущего входа в систему текущего пользователя
IsWebmaster()	Возвращает true, если текущий пользователь является вебмастером. Идентификаторы пользователей-вебмастеров определены в /include/defines.ext. Вебмастер является «супер-администратором». В частности, в index.cfm только вебмастер имеет доступ к Интерфейсу администратора.
GetSubstitution()	Возвращает идентификатор пользователя, которого в данный момент замещает текущий пользователь
GetSubstitutionGroup()	Возвращает группу пользователя, которого в данный момент замещает текущий пользователь

BindManager

Используется для управления привязкой записей из одной таблицы к записям другой. Например, нам надо в каждом документе отображать контактную информацию сотрудников. Документы хранятся в таблице Documents, сотрудники в таблице Contacts; тогда привязки должны храниться в таблице DocumentsContacts, имеющей поля Documents и Contacts (совпадение имен таблиц и полей принципиально).

Функция	Описание
LastError()	Возвращает строку – описание последней произошедшей ошибки
BindManager()	Конструктор
DescribeBinding (\$desttable_, \$sourcetable_, \$bindtable_, \$id_field_, \$name_field_, \$id_source_, \$id_source_value_, \$scriptname_, \$scriptparams_)	Описывает условия работы класса Таблица которую привязываем (Contacts) Таблица к которой идет привязка (Documents) Таблица, хранящая привязки (DocumentsContacts) Название поля идентификатора в \$desttable ("id") Название поля имени в \$desttable ("name") Название поля идентификатора записи в \$sourcetable к которой привязываем ("id") Значение идентификатора записи в \$sourcetable Имя скрипта, выводящего список записей для привязки Параметры для передачи этому скрипту
ShowBindings()	Возвращает HTML-код, содержащий форму с перечнем существующих привязок, позволяющей удалить привязку или создать новую.
HandleForm()	Обработка submit этой формы

Этот класс является базовым для многих прикладных классов в различных конфигурациях. В базовой версии он расширяется классом **Binder**, который, вместе с соответствующим ему интерфейсом BindingForm, используется для отображения табличной части документов «Предложение», «Продажа», «Счет». В простейших случаях доработки табличной части этих документов (когда нужно просто добавить поля) достаточно внести изменения в интерфейс и добавить дополнительные поля в вызов функции SetExtendedParams, а сам класс можно оставить без изменений. В более сложных случаях, когда требуется выполнение каких-либо специфических операций с табличной частью, потребуется расширить и сам класс – прежде всего, метод HandleForm.

Функция	Описание
SetExtendedParams()	Получает и устанавливает список дополнительных параметров (строка, имена параметров – полей таблицы \$bindtable – указаны через запятую). Эти параметры передаются
SetAccess()	Устанавливает уровень доступа. При уровне доступа =1 класс выводит данные, но не разрешает их изменить, при уровне =0 и не выводит.
ShowBindingsEngine()	Аналогичен методу ShowBindings(), но возвращает не HTML-код, а массив данных для слияния с шаблоном.

Bookmarks

Используется для вывода закладок. В index.cfm для вывода закладок используется интерфейс LocalNav, который сливается с данными, полученными от одноименного класса. Закладки, расположенные в нижней части страницы, выводятся интерфейсом Bookmarks, который также получает данные от этого класса.

Функция	Описание
SetBookmarks(\$names_)	Получает массив с именами закладок
SetRecordId(\$name,\$value)	Устанавливает идентификатор текущей редактируемой записи для передачи при переходе с закладки на закладку. \$name – имя поля идентификатора, \$value – значение.
DrawBookmarks()	Возвращает HTML-код закладок
DrawBookmarksEngine()	Возвращает массив данных для слияния с шаблоном
HandleBookmark()	Обрабатывает переход по закладкам
GetCurrentBookmark()	Возвращает номер текущей закладки

ConstantPointer

Реализует управление полями типа «перечисление». Аналогичен классу FieldEditor. Используется классом TableEditor.

FieldEditor

Служебный класс, используемый классом TableEditor для управления полями разных типов. Описание этого класса не приводится, так как напрямую с ним модули index.crm не работают.

Теоретически, может понадобиться создание поля нового типа. В этом случае можно воспользоваться как образцом классом ConstantPointer.

GroupOperations

Позволяет совершать групповые операции с элементами любой иерархии: перенос, удаление, копирование. Этому классу соответствует интерфейс GroupForm. Используется для управления иерархическими справочниками (Персонал, Номенклатура).

Функция	Описание
LoadRightsControl (\$module,\$check_proc)	Устанавливает функцию проверки прав доступа к записям иерархии. Первый параметр – идентификатор модуля CRM (см. ниже описание системы разграничения прав), второй – имя процедуры, возвращающей уровень прав доступа к какой-либо конкретной записи. Такие процедуры для каждого типа модулей описаны в файле /include/custom_rights.ext и имеют вид [Module]RightControl(\$type,\$id), где [Module] – имя модуля, например Clients, \$type – тип записи (константа TreeFolder или TreeFile – контейнер или дочерняя запись), \$id – идентификатор записи.
LastError()	Возвращает строку – описание последней произошедшей ошибки
GroupOperations()	Конструктор
PermanentDelete(\$flag=false)	Если \$flag=false, при удалении объектов записи не будут реально удаляться из таблицы – для них будет выставляться флаг del=1.
KeepHistory(\$flag=true)	Если \$flag=true, любые изменения, производимые в полях таблиц БД, будут отражаться в таблице values_storage, которая хранит историю значений полей записей.
SetAuth(\$a)	Привязывает экземпляр класса Authorisator для проверки прав текущего пользователя для выполнения тех или иных операций
AllowDelete(\$flag)	Разрешает/запрещает операцию удаления
AllowMove(\$flag)	Разрешает/запрещает операцию перемещения

AllowCopy(\$flag)	Разрешает/запрещает операцию копирования
RequireUnique(\$require,\$field)	Устанавливает требование уникальности поля. Первый параметр – требовать ли уникальность, второй – имя поля.
CustomDeleteProc(\$proc,\$on_parent)	Устанавливает специальную процедуру обработки удаления записи. \$proc – функция, \$on_parent – флаг, true если применять к контейнерам, false если к содержимому. Функция будет вызываться для каждого удаляемого элемента перед его удалением из базы.
DescribeGroupOperations(\$parent_id_, \$parent_ref_, \$parent_id_name_, \$parent_name_name_, \$parent_name_parent_, \$parent_table_, \$child_table_, \$child_id_, \$child_name_, \$sortchild_, \$operating_child_, \$rootname_, \$ownerfield_="", \$ownervalue_="")	Устанавливает параметры работы класса Идентификатор исходного контейнера Название поля-ссылки на контейнер в таблице содержимого Название идентификатора в таблице контейнеров Название поля «имя» в таблице контейнеров Название поля-ссылки на родительский объект в таблице контейнеров Название таблицы контейнеров Название таблицы содержимого Название идентификатора в таблице содержимого Название поля «имя» в таблице содержимого Порядок сортировки содержимого Флаг, true если операции производятся с содержимым, false если со вложенными контейнерами. Очень важное поле! Влияет на все остальные функции. Название корня иерархии Название поля «владельца» записи Значение поля «владельца» записи
DrawList(\$dopField, \$dopSubmit)	Выводит список содержимого контейнера Список дополнительных полей Список дополнительных кнопок Array(name,value,)
DrawEngineList()	Выводит массив данных, предназначенный для слияния с шаблоном GroupForm
DrawNewItem(\$custom="", \$file_)	Выводит форму создания нового элемента File, image – загрузка файла или картинки
DrawNewItemEngine(\$custom="")	Возвращает массив данных для формы создания нового элемента
DrawForm()	Выводит форму, обеспечивающую выполнение операций удаления, перемещения, копирования
DrawFormEngine()	Возвращает массив данных для отображения формы, обеспечивающей выполнение операций удаления, перемещения, копирования
DeleteItem(\$iid)	Удаляет запись с идентификатором \$iid
CheckCanMove(\$section,\$destination)	Проверяет возможность перемещения объектов из контейнера \$section в \$destination. Возвращает false, если в результате разрушится иерархия (один из контейнеров вложен в другой).
HandleForm()	Обрабатывает submit формы

TableEditor

Реализует управление содержанием любых таблиц баз данных. От этого класса порожден класс TableEditorEngine, который широко используется в index.cfm – редактирование записей справочников и журналов документов (сам TableEditor «в чистом виде» используется только в Интерфейсе администратора).

Список возможных типов полей: INTEGER_TYPE, STRING_TYPE, TEXT, REAL, POINTER (указатель на запись в другой таблице – выбор при помощи выпадающего меню), HTML (в

index.crm не используется), TREE (указатель на запись в иерархической таблице – выбор при помощи органа управления «дерево»), CUSTOM (поле пользовательского типа, обслуживаемое внешним классом – аналогом FieldEditor; для примера см. класс ConstantEditor), FLAG, DATE_TYPE, DATETIME_TYPE, SELECT_POPUP (указатель на запись в другой таблице – выбор при помощи диалогового окна), RADIO.

Функция	Описание
LastError()	Возвращает строку – описание последней произошедшей ошибки
TableEditor()	Конструктор
LoadRightsControl(\$module)	Устанавливает функцию проверки прав доступа к записи. Параметр – идентификатор модуля CRM (см. ниже описание системы разграничения прав).
OverrideRightsControl(\$access,\$access_current)	Отключает контроль прав и устанавливает уровень доступа принудительно: \$access – права доступа к модулю, \$access_current – к текущей записи.
SetCurrentRecord(\$idrec)	Устанавливает идентификатор текущей редактируемой записи
EnableDelete(\$flag)	Позволять ли удалять запись
PermanentDelete(\$flag=false)	Если \$flag=false, при удалении объектов записи не будут реально удаляться из таблицы – для них будет выставляться флаг del=1.
SetDeleteDependent(&\$func)	Устанавливает пользовательскую функцию, которая будет вызываться при удалении записи. Функция получает id удаляемой записи и может использоваться для удаления зависимых записей.
KeepHistory(\$flag=true)	Если \$flag=true, любые изменения, производимые в полях таблиц БД, будут отражаться в таблице values_storage, которая хранит историю значений полей записей.
EnableUpdate(\$flag)	Позволять ли изменять/добавлять запись
EnableSubscripts(\$flag)	Показывать ли подписи полей
EnableReplace(\$flag)	
SetTableDescription(<ul style="list-style-type: none"> 1. \$table_, 2. \$field_names, 3. \$field_read_names, 4. \$field_types, 5. \$idfield_="id", 6. \$keyfield_="name", 7. \$orderfield_="name", 8. \$ownerfield_="", 9. \$ownervalue_="") 	Устанавливает описание редактируемой записи Имя таблицы Массив названий полей Массив подписей полей Массив типов полей Название поля «идентификатор» Название поля «имя записи» Поле, по которому производится сортировка Название поля «принадлежность записи» Значение поля «принадлежность записи»
AutomateDescription(\$table_)	Выполняет ту же работу что и SetTableDescription, но автоматически.
LoadExtraFields()	Загружает (из таблицы list_props) описания полей для данного модуля, созданные администратором.
OverrideField(\$field,\$type,\$readname)	Заменяет определение какого-либо поля
SetFieldExtra(\$field,\$value)	Определяет, является ли поле \$field (SQL-имя) дополнительным (\$value=true или false).
SetFieldValue(\$field,\$value)	Устанавливает значение по умолчанию для поля Имя поля Значение поля
DeleteField(\$field)	Удаляет поле из описания таблицы
DescribePointerField(\$field,\$targettable,\$targetid="id", \$targetname="name",	Описывает поле типа «указатель» SQL-имя поля Таблица, на которую оно указывает Название поля «идентификатор» в той таблице Название поля «имя записи» в той таблице

<code>\$condition=""</code> , <code>\$order=""</code> , <code>\$module=""</code>)	Условие выборки допустимых полей Поле для сортировки записей Идентификатор модуля, которому принадлежат записи таблицы <code>\$targettable</code> , для определения прав доступа к записям.
<code>CustomizeField(\$field,\$object)</code>	Устанавливает экземпляр специального класса для обработки поля <code>field</code>
<code>DescribeSelectPopupField(\$field,</code> <code>\$dialog,</code> <code>\$dialog_history,</code> <code>\$targettable,</code> <code>\$targetid="id",</code> <code>\$targetname="name",</code> <code>\$condition=""</code>)	Описывает поле типа «выбор с помощью диалога подбора» (например, поле Клиент) SQL-имя поля Имя диалога подбора Идентификатор записи в родительской таблице – если нужно хранить историю поля Таблица, из которой производится выбор записей Поле-идентификатор в ней Поле-название в ней Условие выборки из нее
<code>DescribeAdvancedTreeField</code> <code>(\$field,</code> <code>\$name_p,</code> <code>\$section_p,</code> <code>\$table_p,</code> <code>\$name_c,</code> <code>\$section_c,</code> <code>\$table_c,</code> <code>\$rootname_</code> , <code>\$check_proc,</code> <code>\$sortparent,</code> <code>\$sortchild,</code> <code>\$ownerfield,</code> <code>\$ownervalue,</code> <code>\$select_parent)</code>	Описывает поле типа «выбор из иерархического справочника» (например, поле Подразделение) SQL-имя поля Поле-название в таблице контейнеров иерархии Поле-ссылка на контейнер в ней же Имя таблицы контейнеров иерархии Поле-название в дочерней таблице иерархии Поле-ссылка на контейнер в ней же Имя дочерней контейнеров иерархии Название корневого элемента иерархии Имя функции проверки прав доступа Порядок сортировки контейнеров Порядок сортировки дочерних элементов Поле принадлежности записи (фактически, условие) Значение «хозяина» записи (значение условия) Флаг – выбирать контейнеры (true) или дочерние элементы (false)
<code>SetFieldMandatory(\$field,\$flag=true)</code>	Делает поле <code>\$field</code> обязательным для заполнения (если <code>\$flag=true</code>)
<code>RecordEditor(\$id=0)</code>	Выводит форму редактора записи Значение(id) записи
<code>HandleForm()</code>	Обрабатывает submit формы

Общие функции

Помимо классов, существует ряд отдельных функций, которые также используются практически всеми модулями `index.cfm`. Большая часть этих функций содержится в файле `/config/Engine/util.ext`; функции, относящиеся к безопасности и проверке прав доступа, содержатся в `/include/security.ext`.

Функция	Описание
<code>querystring()</code>	Возвращает полный адрес текущей страницы
<code>removeid(\$qs,\$id)</code>	Удаляет из строки адреса <code>\$qs</code> параметр с названием <code>\$id</code> и его значение (т.е. выражение <code>&[id]=[value]</code>)
<code>FormatPrice(\$price)</code>	Форматирует число в денежном формате
<code>BuildCondition(\$control,</code> <code>&\$config,</code> <code>\$table,</code> <code>\$ffield,</code> <code>\$fop,</code> <code>\$fvalue)</code>	Формирует SQL-условие фильтра Список (экземпляр класса <code>ListControl</code>) Запись (экземпляр класса <code>TableEditorEngine</code>) Имя таблицы Имя поля Операция Значение

BuildSort(&\$control, &\$config, \$table, \$sortf, \$sorto, \$def="name")	Формирует SQL-выражение сортировки Список (экземпляр класса ListControl) Запись (экземпляр класса TableEditorEngine) Имя таблицы Имя поля Порядок Порядок сортировки по умолчанию (если \$sortf и \$sorto не заданы)
GetPeriodForm(\$year="", \$smoonth="", \$sday="", \$year="", \$emoonth="", \$eday="")	Возвращает массив данных для формы выбора периода (интерфейсы PeriodForm и PeriodFormShort) Год даты начала Месяц даты начала День даты начала Год даты конца периода Месяц даты конца периода День даты конца периода Если параметры дат пропущены, они берутся из запроса – если форма выбора периода уже была заполнена. Если и в запросе этих значений нет, берется интервал от текущей даты до предшествующей ей на месяц.
SaveValues(\$to_keep, \$force_save, \$table, \$current_record, \$idfield)	Проверяет, нужно ли сохранить информацию об истории значений записи, и сохраняет ее Массив значений полей записи Флаг «сохранять в любом случае» Таблица БД Идентификатор записи Название поля с идентификатором Функция проверяет, совпадают ли значения в массиве \$to_keep со значениями, хранящимися в БД, и если не совпадают – сохраняет историю значений для тех полей, которые изменились.
num2str(\$L)	Возвращает сумму прописью от числа \$L
GetPP(\$sale)	Возвращает номер платежного поручения, которым была оплачена продажа \$sale (для печати в счет-фактуре)
CP1251toUTF8(\$str)	Преобразует из кодировки CP1251 в UTF8 (используется при экспорте в Excel)
countLines(\$str,\$width)	Разбивает текст \$str на строки таким образом, чтобы он вписался в прямоугольник шириной \$width, будучи написан шрифтом Arial 9. Используется при печати для определения высоты строк.
round(\$num)	Функция округления до двух знаков после запятой, которая учитывает специфику хранения чисел в базе данных.

Основные сведения о ядре index.crm

Структура системы

CRM-система index.crm представляет собой надстройку над движком, описанным выше. В index.crm вводится дополнительное понятие – модуль системы. Модуль системы представляет собой интерфейс, позволяющий пользователю выполнять операции с какими-либо данными. Для пользователя, модуль системы представляет собой пункт верхнего меню. С точки зрения движка, модуль системы – это определенный документ, в котором содержится вызов

Index.crm содержит модули трех основных типов (таблица `modules_types`): справочники, журналы документов и отчеты. Каждый программный модуль вызывается из отдельного документа. Кроме того, существует несколько дополнительных документов (страница входа, главная страница и др.).

Index.crm хранит список самостоятельных модулей системы (из которых, в частности, формируется верхнее навигационное меню) в таблице `modules_menu`. Типы модулей хранятся в таблице `modules_types`. Таблица `modules_menu` связывает модуль системы с документом, в котором он реализован. Идентификатор из таблицы `modules_menu` используется как идентификатор модуля системы, в частности, при проверке прав доступа.

Точки входа

Система имеет несколько точек входа (PHP-скриптов), к которым может обратиться пользователь.

`/index.php` – основная точка входа. Через этот скрипт отображаются все страницы системы.

`/list_getdata.php`, `/tree_getdata.php` – скрипты подкачки данных в органы управления «список» и «дерево»

`/tree/getclients.php`, `/tree/getdocuments.php`, `/tree/getgoods.php` – скрипты подкачки данных в диалоговые окна выбора клиента, документа, подбора товаров

`/dialogs/GetFile.php` – отправляет пользователю файл, хранящийся в каталоге файлов.

`/dialogs/PrintDocument.php` – интерфейс для печати одного документа

`/dialogs/PrintDocuments.php` – интерфейс для печати группы документов

`/dialogs/PrintReport.php` – интерфейс для печати отчета

`/dialogs/drawchart.php` – скрипт, выдающий изображение диаграммы (используется библиотека `gd`)

`/dialogs/остальные файлы.php` – модальные диалоги выбора или создания чего-либо

`/config/*.php` – страницы Интерфейса администратора.

Скрипты подкачки данных вызываются из соответствующих органов управления при помощи метода `XMLHttpRequest` и возвращают данные в формате XML.

Модальные окна используются в основном для реализации действий, выполняемых при нажатии каких-либо кнопок в интерфейсе системы, например вызова фильтров и т.д.

Интерфейсы печати работают во многом аналогично основному интерпретатору движка – скрипту `index.php`. Они получают на входе идентификатор(ы) документов, которые должны быть отправлены на печать, получают в БД соответствующие шаблоны (таблица `print_templates`), вызывают нужный программный модуль из каталога `/dialogs/print_forms` для получения данных, и сливают шаблон и данные в XML-документ в формате Microsoft Excel, который и отправляется пользователю.

Основные модули системы

Название	Id модуля	Имя программного модуля	Таблицы базы данных	Файл класса
Справочники				
Клиенты	4	clients	Clients – клиенты Clients_groups –	clients.ext

			группы Clients_files – присоединенные файлы	
Персонал	3	personnel	Branches – подразделения Users – пользователи Branches_modules и groups_modules – права доступа	Personnel.ext
Номенклатура	5	goods	Categories – разделы Goods – товары Measures – единицы измерения	Goods.ext
Подчиненные справочники справочника «Клиенты»				
Договоры	Нет		Contracts – договоры Contracts_goods – подключенные системы	Clients.ext
Персоны	Нет		Persons – персоны	Clients.ext
Журналы документов				
Контакты	12	contacts	Contacts – контакты Contact_types – типы контакта	Contacts.ext
Предложения	6	offers	Offers – предложения Offers_goods – товары в них Offers_files – присоединенные файлы	Offers.ext
Счета	18	invoices	Invoices – счета Invoices_goods – товары в них	Invoices.ext
Продажи	7	sales	Sales – продажи Sales_goods – товары в них	Sales.ext
Платежи	9	payments	Payments – платежи	payments.ext
Отчеты				
Выполнение задач		tasks_report		tasks_report.ext
Задолженность контрагентов		debts		debts.ext
Изменения в реквизитах клиента		client_props_rep		client_props_rep.ext
Клиенты, требующие внимания		problem_clients		problem_clients.ext
Отчет по продажам		repsales		repsales.ext
Продуктивность персонала		productivity		productivity.ext

Статистика продаж		sales_stages		sales_stages.ext
Агрегированный отчет по взаимодействию с клиентом		report_client		report_client.ext
Все документы		alldocs		alldocs.ext
История баланса клиента		client_payments		client_payments.ext

Остальные таблицы базы данных (не перечисленные ранее):

Таблица	Назначение
eventlog	Журнал действий пользователей
events	Задачи
files	Иногда используется для хранения информации о файлах, загруженных в систему
filesections	Иногда используется для хранения информации о файлах, загруженных в систему
Filters_conditions	В этой таблице хранятся «условия фильтрации» – параметры, по которым можно фильтровать и сортировать справочники и журналы документов.
images	Иногда используется для хранения информации об изображениях, загруженных в систему
imgsections	Иногда используется для хранения информации об изображениях, загруженных в систему
list_props	В этой таблице хранится описание полей справочников и журналов документов, в т.ч. созданных пользователем.
specialsections	Список разделов интерфейса администратора, зависящих от конфигурации
specialsectionsusers	Рудимент системы управления сайтом; не используется
stages	Стадии цикла продаж; в настоящее время не используется
standard_list	Список стандартных справочников (практически не используется)
standard_list_options	Значения стандартных справочников (практически не используется)
updates	Список установленных пакетов (обновлений)
user_settings	Различные настройки пользователя
values_storage	История значений для всех таблиц

Стандартные интерфейсы

Интерфейс	Класс или функция, подготавливающий данные для него	Описание
BindingForm	Binder (/include/Binder.class)	Используется для формирования табличной части документов
Bookmarks	Bookmarks (/config/Engine/Bookmarks.class)	Закладки для навигации внутри страницы
LocalNav	Bookmarks (/config/Engine/Bookmarks.class)	Закладки для навигации внутри страницы (немного отличается внешним видом от предыдущего интерфейса)
GroupForm	GroupOperations (/config/Engine/GroupOperations.class)	Выводит форму для выполнения групповых операций с элементами любой иерархии – удаления, перемещения, создания новых элементов
PeriodForm	функция GetPeriodForm (/config/Engine/util.ext)	Форма выбора периода (для отчетов)
PeriodFormShort	функция GetPeriodForm (/config/Engine/util.ext)	Форма выбора периода (для фильтрации журналов документов)

PrintButton	метод ShowPrintButton класса reports (/include/reports.class)	Кнопка для отправки отчета на печать
PrintForm	Каждый модуль самостоятельно	Выпадающее меню со списком печатных форм для данного документа/элемента справочника
StandardForm	TableEditor (/config/Engine/TableEditor.class)	Форма редактирования свойств записи таблицы
Status	Каждый модуль самостоятельно	Строка статуса в верхней части страницы
TaskListForDocument	Метод GetTasksForDocument класса Workflow (/include/workflow.class)	Список задач, связанных с документом
Toolbar	ListControl (/include/ListControl.class)	Панель инструментов, расположенная над списком

Диалоговые окна

Диалоговые окна хранятся в каталоге /dialogs и используются для выполнения различных операций многими модулями системы.

Примечание: здесь и далее под «типом документов» подразумевается идентификатор (из таблицы modules_menu) модуля, который работает с документами данного типа.

Диалог	Описание
AddContact	Форма регистрации контакта. Получает через запрос идентификатор клиента (\$_REQUEST["client"]), возвращает через window.dialogArguments объект с информацией о новом контакте, а также о новом договоре и контактной персоне, которые могут быть созданы в процессе регистрации контакта. Свойства объекта: Client – идентификатор клиента Date – дата контакта Contract – id договора, или -1, если создан новый newContract – название нового договора (если создан) Person – id персоны, или -1, если создана новая newPerson – имя новой персоны (если создана) newPersonPosition – должность новой персоны (если создана) type – тип контакта ball – оценка успешности контакта comment – комментарий пользователя addTask – флаг «создать задачу по результатам контакта»
AddContract	Используется диалогом AddContact для добавления нового договора
AddPerson	Используется диалогом AddContact для добавления новой персоны
List_picker_md	Используется диалогами подбора товаров для ввода количества товара
AddGood	Используется диалогами подбора товаров для создания нового товара
AddTask	Форма создания новой задачи. Входящие параметры: клиент (client) тип документа-основания (doctype) идентификатор документа-основания (document) идентификатор контакта (contact; указание контакта аналогично указанию идентификатора документа-контакта в параметре document). Свойства возвращаемого объекта: User – пользователь, которому назначена задача Date – дата, на которую она назначена Task – тип задачи Title – название задачи Comment – комментарий планирующего

	Client – идентификатор клиента, с которым связана задача
ChooseClientParam	Выбор параметра справочника клиентов, по которому осуществляется подбор клиента. Используется диалогом FindClient.
ChooseColumns	Выбор столбцов, которые будут отображаться в навигационном списке. Получает через запрос три списка, в которых элементы разделены точкой с запятой: sets – список выбранных полей (SQL-имена), full – список SQL-имен полей (может содержать произвольные названия, которые должны интерпретироваться вызывающим модулем), fullr – список названий полей, которые будут показаны пользователю. Возвращает объект с одним свойством – sets, который содержит список выбранных полей.
ChooseDocument	Диалог выбора документов из списка документов, удовлетворяющих каким-либо критериям. Параметры запроса: client – идентификатор клиента, которому принадлежат документы, doctype – тип выбираемых документов, can_add – флаг «допустимо создание нового документа». Диалог загружает при помощи XMLHttpRequest (обращение к скрипту /tree/getdocuments.php) список документов указанного типа, относящихся к указанному клиенту. Возвращает объект, свойство id которого содержит идентификатор выбранного документа (или -1, если пользователь нажал на кнопку «Создать новый»). Свойство name содержит отображаемое название документа.
ChooseGoods	Диалог подбора товаров. Для подбора используется дерево с Norton-подобной навигацией. Пользователь имеет возможность перемещаться по дереву при помощи нажатия стрелок на клавиатуре или мыши. При выборе какого-либо товара отображается окно ввода количества товара, и выбранный товар добавляется к списку в правой части окна. После выбора всех нужных товаров пользователь нажимает кнопку «Принять», и информация о выбранных товарах передается в вызывающий документ. Пользователь имеет возможность создавать новые товары в каталоге в ходе подбора. Для этого используется диалоговое окно AddGood и скрипт /tree/makegood.php, обращение к которому происходит методом XMLHttpRequest. Свойства возвращаемого объекта: arr – массив товаров. Каждый элемент массива является массивом именованных элементов: num – количество, name – название, price – цена. Идентификатор товара является индексом массива.
ChooseGoodsAlt	Альтернативный подбор товаров. Подбор осуществляется по части названия (или любого другого свойства) товаров. Иерархия справочника номенклатуры при этом игнорируется. Для подкачки данных используется скрипт /tree/getgoods.php. Возвращаемый объект – такой же, как у диалога ChooseGoods.
ChooseGoodsParam	Используется диалогом ChooseGoodsAlt для выбора параметра, по которому происходит подбор товаров.
ChoosePrintForm	Выбор формы печати. Используется при пакетной печати документов в том случае, если для документов данного типа имеется несколько печатных форм. Получает тип печатаемых документов через параметр doctype. Возвращает идентификатор выбранного шаблона печати через window.returnValue.
Filter	Простой фильтр. Получает параметры: Fields – список полей (SQL-имена), через запятую Names – отображаемые имена полей Types – типы полей (0=строка, 1=указатель, 2=число) List – название модуля (используется для загрузки списка дополнительных полей из таблицы List_props) Свойства возвращаемого объекта:

	<p>Field – SQL-имя поля Op – операция Value – значение Поле «операция» передается в виде строки, которая заменяется на реальный символ операции функцией BuildCondition. Возможные значения: « EQUAL », « NOT EQUAL », « LESS », « MORE », « LIKE »</p>
FindClient	<p>Выбор клиента. Выбор осуществляется путем поиска подстроки в названии или любом другом поле клиента. Возвращает идентификатор выбранного клиента через window.returnValue, название – через свойство name объекта, переданного через window.dialogArguments.</p>
SelectUser	<p>Выбор пользователя (используется при переходе на календарь какого-либо пользователя). Возвращает идентификатор выбранного пользователя через window.returnValue.</p>
ShowHistory	<p>Отображает историю значений какого-либо реквизита какой-либо записи справочника. Параметры: Object_type – тип объекта (идентификатор модуля) Object_id – идентификатор объекта Field_name – имя реквизита (поля), история которого отображается Id_field – поле «идентификатор» в таблице справочника Name_field – поле «название» в таблице справочника Req_name – название реквизита для отображения пользователю</p>
Sort	<p>Выбор параметра сортировки. Получает параметры: Fields – список полей (SQL-имена), через запятую Names – отображаемые имена полей Types – типы полей (0=строка, 1=указатель, 2=число) List – название модуля (используется для загрузки списка дополнительных полей из таблицы List_props) Свойства возвращаемого объекта: Sortf – поле Sorto – направление сортировки (1=по убыванию, 2=по возрастанию)</p>

Программные средства index.crm

Система разграничения прав доступа

В системе существует два механизма назначения прав: на основе ролей пользователей и организационных подразделений. Права первого типа связывают модуль системы и группу пользователей (таблица `groups_modules`), второго типа – модуль системы и подразделение организации (`branches_modules`).

При определении прав доступа к каждой конкретной записи какого-либо журнала документов или справочника используется следующий алгоритм.

Для пользователя строится список правил доступа к данному разделу. Правил может быть несколько. Первое из них – всегда правило доступа по группе, остальные – по подразделению. По каждому правилу определяется уровень прав:

- Если проверка принадлежности записи не предусмотрена – берется уровень прав, установленный по умолчанию.

- Проверка принадлежности записи заключается в сравнении значения поля, указанного при установке прав или создании правила (обычно это поле `manager`) со значением идентификатора текущего пользователя. Если проверка принадлежности записи предусмотрена, и она выполняется успешно – берется тот уровень прав, который предусмотрен для «своих» записей; иначе берется уровень прав, установленный по умолчанию.

Среди всех полученных по списку правил уровней выбирается минимальный. Этот уровень является результирующим уровнем доступа для пользователя.

Если пользователь замещает какого-либо другого пользователя, такая же проверка проводится и для второго пользователя. Среди двух результатов по разным пользователям выбирается максимальный.

С технической точки зрения, проверка прав осуществляется двумя способами: при помощи PHP-функций (определение прав доступа к отдельной записи – см. описание функции `GetAccessLevel`), либо при помощи `stored function` SQL-сервера (проверка доступа к группе записей). Хранимая функция создается PHP-скриптом в ходе выполнения (см. описание функции `CreateAccessCheckSF`). Некоторые модули могут допускать кэширование прав доступа для ускорения построения списка доступных элементов. Обработка этой ситуации также происходит в функции `CreateAccessCheckSF`.

Система печати (экспорта в Microsoft Excel)

Система печати функционирует путем экспорта данных в Microsoft Excel. Для экспорта данные подготавливаются в виде XML данных, соответствующих схеме этого приложения, и отправляются пользователю как файл, имеющий тип `application/vnd.ms-excel`.

Для каждого типа документов или справочников в таблице `print_templates` может храниться набор шаблонов печати (в поле `doctype` этой таблице хранится идентификатор модуля системы – `id` из таблицы `modules_menu`). Модуль, допускающий печать содержимого, должен построить список доступных шаблонов печати, и отобразить его при помощи интерфейса `PrintForm`. После выбора шаблона пользователем в отдельном окне вызывается скрипт `/dialogs/PrintDocument.php` с параметрами `doctype=идентификатор модуля`, `idi=идентификатор записи`, `form=идентификатор шаблона печати`. Этот скрипт вызывает для подготовки данных функцию, имя которой соответствует имени модуля. Функция должна находиться в одноименном файле, расположенном в каталоге `/dialogs/print_forms`. Эта функция получает три параметра – `doctype=идентификатор модуля`, `id=идентификатор записи`, и `form_name=название формы печати` (из поля `name` таблицы `print_forms`; параметр опциональный). Функция должна подготовить массив данных для слияния с шаблоном. Особым элементом этого массива является элемент `"rowcount"` – он содержит число строк в документе.

Шаблоны печати создаются следующим образом. В Microsoft Excel создается файл-прототип, который сохраняется в формате XML. Полученный файл вставляется как код шаблона печати в разделе «Шаблоны печати» Интерфейса администратора. Нужно иметь в виду, что русские буквы при сохранении Excel преобразуются в кодировку UTF8, а в шаблоне они должны быть в обычной кодировке cp1251 – поэтому все русские надписи после вставки шаблона придется переписать (или перекодировать шаблон перед вставкой). Так как шаблон может

использоваться для печати как одного, так и нескольких документов сразу, в шаблон сразу после последнего тега <Column> должен быть вставлен цикл `{:documents}` (который закончится перед закрывающим тегом </Table>). У тега <Table> обязательно должен быть указан параметр `ss:ExpandedRowCount="{!rowcount}"` – сюда будет подставлено общее число строк в документе. Наконец, в конце шаблона (перед закрывающим тегом </Worksheet>) должен быть вставлен следующий блок:

```
{?pagebreaks}
  <PageBreaks xmlns="urn:schemas-microsoft-com:office:excel">
{:pagebreaks}
  <RowBreaks>
    <RowBreak>
      <Row>{!row}</Row>
    </RowBreak>
  </RowBreaks>
{::/pagebreaks}
  </PageBreaks>
{?/pagebreaks}
```

Этот блок нужен для того, чтобы разбить Excel-документ на страницы в том случае, если на печать выводится несколько элементов сразу.

Во внутренней части шаблона могут использоваться все теги уровня функции, описанные в начале настоящего руководства.

Если происходит печать нескольких документов одновременно (пользователь нажал кнопку «Печатать все» на панели инструментов какого-либо списка), то вызывается скрипт `/dialogs/PrintDocuments.php`. Если для данного типа документов предусмотрено несколько вариантов печатных форм, он запрашивает у пользователя, какую форму нужно использовать. Затем происходит печать всех документов, которые на момент нажатия кнопки отображались в списке.

Особым случаем печати является печать реестра документов (кнопка «Печатать список»). Она происходит при помощи специального шаблона «Список», а данные для него должен подготовить сам вызывающий модуль. Массив данных должен содержать следующие элементы: `ncols` – количество столбцов, `cols` – массив названий столбцов, `rows` – массив строк. Каждая строка, в свою очередь, представляет собой массив массивов `columns`, каждый из которых содержит элемент `data`, в котором находятся данные какой-либо ячейки ;-).

Для примера см. функцию `HandlePrintList` в `clients.ext`, которую с минимальными изменениями можно использовать и в любом другом модуле.

Похожим образом происходит и печать отчетов, для которых используется отдельный скрипт `/dialogs/PrintReport.php`. Функция, подготавливающая данные для печати отчета, должна воспроизводить процедуру генерации отчета, либо включать и использовать класс отчета из каталога `/include`.

Органы управления

В системе используется несколько органов управления («контролов»), разработанных специально для нее:

- список;
- дерево;
- календарь;
- выпадающее меню (замена <select>).

Эти органы управления используются всеми модулями системы.

Сортировка таблиц

Кроме того, имеется файл `table_sorted.js`, содержащий библиотеку, позволяющую легко организовать сортировку любых таблиц в системе. Ниже приводится описание его API.

function tableSorted(int номер столбца,object HTMLTBody,mixed порядок сортировки,string тип сортировки)

Одиночная функция tableSorted служит для автоматизации сортировки HTML-таблицы. Сделана в виде функции, т.к. возможно, что она будет являться компонентом.

первый параметр - это номер поля, по которому сортировать таблицу;

второй параметр - это непосредственно объект TBODY;

третий параметр - определяет сортировку по возрастанию и может принимать параметры:

1) true - по возрастанию

2) false - по убыванию

3) 'auto' - по первому срабатыванию триггера она сортировать будет по возрастанию, по второму - по убыванию и т.д.

по умолчанию выбирается 'auto'

четвертый параметр - служит для определения типа поля для сортировки:

1) 'string' - сортирует как строки

2) 'number' - сортирует значения как числовые

3) 'auto' - автоматически определяет - по какому типу сортировать поля

по умолчанию выбирается 'auto'

Функция при сортировке не нарушает триггеры, свойства DOM элементов.

Не работает со столбцами, имеющими свойства colspan и rowspan.

Пример работы tableSorted

```
<table>
  <thead>
    <tr>
      <th onclick="tableSorted(1,document.getElementById('content'))">№</th>
      <th
onclick="onclick="tableSorted(2,document.getElementById('content'))">Имя</th>
    </tr>
  </thead>
  <tbody id="content">
    <tr>
      <td>1</td>
      <td>Петр</td>
    </tr>
    <tr>
      <td>2</td>
      <td>Иван</td>
    </tr>
    <tr>
      <td>3</td>
      <td>Владимир</td>
    </tr>
  </tbody>
</table>
```

Так же "в комплекте" к функции tableSorted идет функция checkedCheckboxFromTable:

function checkedCheckboxFromTable(int номер столбца,object HTMLTBody,mixed отметка столбцов)

функция помечающая в таблице все чекбоксы в определенном столбце

Принимает следующие параметры:

первый параметр - это номер поля, в котором надо отмечать чекбоксы

второй параметр - это непосредственно объект TBODY

третий параметр - определяет, отмечать все чекбоксы, или нет. Может принимать параметры:

1) true - все отмечать

2) false - все отключать

3) 'auto' - по первому срабатыванию триггера либо все отмечает, либо нет, в зависимости от заполнения. По второму срабатыванию - инвертирует все, и т.д. По умолчанию выбирается 'auto'.

Список и дерево

Эти органы управления используются для навигации по спискам каких-либо записей (одноуровневым или иерархическим). Эти органы управления представляют собой два возможных вида навигационного контроля, который используется для выбора записи в некоторых справочниках.

Список и дерево строятся JavaScript'ом, который динамически (после загрузки страницы) создает визуальное представление органа управления в предназначенном для этого элементе div. Подкачка данных в список и дерево осуществляется при помощи метода XMLHttpRequest. Некоторые функции, облегчающие работу с подкачкой данных, находятся в файле /tree/xml_loader.js, который используется не только органами управления, но и другими частями системы – например, диалогами выбора клиента или подбора товаров.

Собственно функциональность списка и дерева реализуется классами BList и BTree, объявленными в /tree/NewList.php и /tree/NewTree.php (а также классами JavaScript List и Tree, объявленными в /tree/xml_list.js и /tree/xml_tree.js), но модули системы работают с ними при помощи промежуточного класса ListControl (/include/ListControl.class). Приведем описание методов класса ListControl.

Функция	Описание
ListControl(\$table, \$name, \$title, \$width="", \$align="", \$show_icons=true)	Конструктор Таблица БД, записи из которой отображает список SQL-имя поля первой колонки Заголовок первой колонки Ширина первой колонки Выравнивание первой колонки Флаг, показывать ли иконки перед записями
GetQuery()	Возвращает SQL-запрос, построенный с учетом всех фильтров, наложенных на список, и параметров сортировки
ShowPanel(\$show)	Устанавливает флаг, нужно ли показывать панель инструментов над контролем.
AddButton(\$width, \$icon, \$icon_act, \$func, \$tooltip, \$id="", \$href="")	Добавляет кнопку к панели инструментов контроля Ширина кнопки URL картинки URL картинки при наведении JS-функция, вызываемая при нажатии на кнопку Подсказка к кнопке Идентификатор кнопки Ссылка на кнопку (может использоваться вместо \$func)
SetClickHandler(\$func)	Устанавливает имя JS-функции, которая вызывается при щелчке на запись в списке или дереве.
SetCurrentSelection(\$id)	Устанавливает идентификатор текущей выбранной записи
SetOrder(\$order)	Устанавливает порядок сортировки (SQL-выражение)
SetOwner(\$field_owner, \$value_owner)	Устанавливает условие фильтра (вызов может повторяться несколько раз для наложения нескольких условий) Имя поля в SQL Значение
SetRightsControl(\$func_name)	Устанавливает имя PHP-функции, которая проверяет уровень прав доступа к каждой записи в списке. Этот метод проверки прав доступа не рекомендуется использовать; вместо него нужно пользоваться функцией CreateAccessCheckSF, которая создает stored функцию проверки прав – эту функцию нужно установить как одно из условий выборки записей из БД.
SetCustomIcon	Устанавливает имя PHP-функции, которая будет использоваться

(\$icon,\$func_name="")	для определения того, какую иконку надо отображать перед каждой записью. На момент написания руководства эта возможность в системе реально не использовалась.
AddColumn(\$title,\$field,\$width="", \$align="")	Добавляет столбец в список Заголовок столбца SQL-выражение для выборки (может быть подзапросом) Ширина столбца Выравнивание столбца
AddPointerColumn(\$title,\$field,\$table,\$idfield,\$namefield,\$width="", \$align="")	Добавляет столбец-указатель на другую таблицу Заголовок столбца Поле-ссылка в таблице-источнике Таблица, откуда берутся названия для отображения Имя поля «идентификатор» в ней Имя поля «название записи» в ней Ширина столбца Выравнивание столбца
DeletePointerColumn(\$field)	Удаляет столбец из списка
FilterList(\$condition)	Накладывает условие на список (параметр – SQL-выражение)
ShowToolbar()	Возвращает массив для интеграции с интерфейсом Toolbar, отображающим панель кнопок
PrintHTML()	Формирует и возвращает HTML-код, отображающий контрол. Также вносит необходимые изменения в глобальные переменные \$head_code и \$onload_code, которые подставляются соответственно в блоки <head> и <body onload=""> HTML-страницы.

Выпадающее меню

Этот орган управления не имеет инкапсулирующего PHP-класса, поэтому модули системы должны работать с ним напрямую, включая необходимый JavaScript-код в свой шаблон. Также работающие с ним модули должны проверять, имеется ли в \$head_code включение файла /tree/select.js, и если нет – включать его.

Необходимость реализации этого органа управления вызвана тем, что стандартный <select> перекрывает любые div'ы, создаваемые страницей, в т.ч. с абсолютным позиционированием. Это делает невозможной работу меню системы.

Приведем описание класса CSelect, реализованного на JavaScript.

Функция	Описание
CSelect(container, name, width, height, bwidth, lwidth, lheight, nodef_val)	Конструктор Объект-контейнер, в котором будет создан контрол (обычно div) Название (идентификатор) элемента input type=hidden, куда будет помещено выбранное пользователем значение Ширина контрола Высота контрола Ширина кнопки раскрытия выпадающей части Ширина выпадающей части Высота выпадающей части Значение, которое присваивается hidden'у, если ничего не выбрано
addOption(data, val, selected, classname)	Добавляет опцию в выпадающую часть Текст Значение Выбрана ли (необязательный параметр) Имя класса (необязательный параметр) Возвращает объект опции.
insertBeforeOption(before, data, val)	Добавляет опцию в выпадающую часть перед другой опцией Опция, перед которой будет вставлена данная опция Текст Значение

selected, classname)	Выбрана ли (необязательный параметр) Имя класса (необязательный параметр) Возвращает объект опции.
setSelectedByObj(o, notify, user)	Устанавливает выбранную опцию (по объекту) Объект опции, которая должна быть выбрана Вызвать ли обработчик onchange Если изменение вызвано вмешательством пользователя =true
setSelectedByValue(val, notify, user)	Устанавливает выбранную опцию (по значению) Значение опции, которая должна быть выбрана Вызвать ли обработчик onchange Если изменение вызвано вмешательством пользователя =true
setOnChangeFunc(func)	Устанавливает функцию-обработчик изменения выбранной опции (аналогично обработчику onchange у стандартного <select>)
Enable(state)	Разрешает или запрещает контрол. control.Enable(false) аналогично control.disabled=true для стандартного <select>.

Календарь

Этот орган управления предназначен для выбора даты.

Функция	Описание
Calendar(CalContainer, WeekFromMonday, Lang, mnYear, mxYear)	Конструктор Объект-контейнер, в котором будет создан контрол (обычно div) Флаг «начинать неделю с понедельника» Язык (русский/английский) Минимальный год Максимальный год
RebuildCalendar(year, month, day, markedDays)	Строит календарь в объекте-контейнере (удаляя старый, если есть) Выбранный год Выбранный месяц Выбранный день Массив «отмеченных» (визуально выделенных) дней – размер массива равен числу дней в месяце, для каждого дня элемент равен 0 или 1
SetMarkedDays(markedDays)	Установить «отмеченные» дни. Параметр – массив, размер которого равен числу дней в месяце, для каждого дня элемент равен 0 или 1.
SetOnEventsFunction(onclickfunc, onchangefunc)	Устанавливает пользовательские обработчики событий onclick и onchange. Обработчик onclick Обработчик onchange Параметры функций-обработчиков: function(calo,prevcell,cell,year,month,day) calo – объект календаря prevcell – ранее выделенный день cell – новый выделенный день year,month,day – дата, по которой щелкнули function(calo,year,month,status) year,month,day – установленная дата status – =true если вызвана перед изменением отображения календаря, =false если после
Clear()	Удаляет календарь из контейнера

Дополнительные программные средства

Система содержит ряд дополнительных классов и функций, которые также могут использоваться всеми модулями системы.

UserSettings

Этот класс предназначен для хранения и получения информации о настройках, сделанных в системе каждым конкретным пользователем. В частности, этот класс используется для хранения информации о том, какие периоды установлены пользователем в журналах документов, какие столбцы выбраны для отображения в справочнике «Клиенты» и т.д.

Функция	Описание
UserSettings(\$module)	Конструктор. Параметр – идентификатор модуля (строка)
SetValue(\$param,\$value)	Устанавливает значение параметра \$param равным \$value и сохраняет это значение в базу данных
<p>Для доступа к установленным значениям используется прямой доступ к именованному массиву \$values, являющемуся общедоступным членом класса. Т.е. для хранения и извлечения величин используется такой код:</p> <pre>\$v=new UserSettings("clients"); \$v->SetValue("width",10); \$a=\$v->values["width"]; // \$a=10;</pre>	

Workflow

Этот класс используется для выполнения операций, связанных с назначением и выполнением задач для пользователей.

Функция	Описание
OnCreateDocument(\$module, \$doc, \$getuser="")	Вызывается при создании документа. Автоматически закрывает задачи, которые можно считать выполненными при создании этого документа. Идентификатор модуля Идентификатор созданного документа Флаг «получить идентификатор пользователя из поля manager созданного документа»
closeTasks(\$tasklist, \$tasks, \$doc)	Закрывает конкретные задачи при создании документа Список идентификаторов задач (через запятую) Не используется Идентификатор документа
CloseSpecificTask(\$task, \$document)	Закрывает конкретную задачу \$task документом \$document
CreateTask(\$type, \$comment, \$title, \$user, \$base_doctype="", \$base_doc="", \$client="", \$date="")	Создает новую задачу Тип задачи Комментарий Название Идентификатор пользователя – исполнителя Тип документа-основания Идентификатор документа-основания Идентификатор клиента Дата, на которую создается задача
CloseTaskByContact(\$contact)	Автоматически закрывает задачи, которые могут быть закрыты только что созданным контактом \$contact
GetTasksForDocument(\$doctype, \$document)	Возвращает список задач, связанных с документом \$document (имеющим тип \$doctype). Возвращаемое значение – массив, предназначенный для слияния с интерфейсом TaskList.

Security

Файл /include/security.ext не является классом, а содержит функции, используемые модулями системы для проверки прав доступа.

Функция	Описание
GetAccessLevel(\$module, \$record=0)	Возвращает уровень доступа (0=нет, 1=только чтение, 2=полный) текущего пользователя к записи \$record из таблицы, обслуживаемой модулем \$module (или к модулю в целом, если id записи не указан).
CreateAccessCheckSF(\$module)	Создает stored функцию для проверки прав доступа текущего пользователя к записям таблицы, обслуживаемой модулем \$module, и возвращает ее имя. Может использоваться для формирования SQL-условий выборки. Stored функции нужно передать один параметр – идентификатор записи. Если модуль допускает кэширование прав, может вместо имени функции вернуть массив, содержащий два элемента: подзапрос для выборки из таблицы кэшированных прав и запрос для выборки количества доступных текущему пользователю записей.
GetCanCreate(\$module)	Проверяет, может ли текущий пользователь создавать новые записи в таблице модуля \$module.
GetAccessByGroup(\$module)	Возвращает массив, содержащий информацию об уровне доступа текущего пользователя к записям таблицы модуля \$module: элемент "access" содержит значение прав доступа к модулю в целом, "query" – часть SQL-запроса для проверки принадлежности записи, "access_private" – уровень доступа к записям, удовлетворяющим условиям проверки.
GetUserBranch(\$id)	Возвращает идентификатор подразделения, к которому принадлежит текущий пользователь.
ModuleRightControl(\$type, \$id, \$module, \$table, \$parentname)	Возвращает уровень доступа к конкретной записи какой-либо таблицы Тип записи (TreeFolder=родительский элемент, TreeFile=дочерний – имеет смысл для модулей, работающих с иерархическими таблицами) Идентификатор записи Идентификатор модуля Имя таблицы Имя поля-ссылки на родительскую запись Эта функция использовалась только для контроля прав доступа к записям, отображаемым в навигационном контроле. В настоящее время считается устаревшей.

Также имеется файл /include/custom_rights.ext, который содержит функции-псевдонимы для функции ModuleRightControl, определенные для каждого конкретного модуля системы (например, ClientsRightControl).

Функции SQL

Многие модули используют stored функции, входящие в базовую версию. Приведем их описание.

Функция	Описание
FUNCTION `getsum_invoices` (invoice INT)	Возвращает сумму счета с идентификатором

RETURNS double(10,2)	invoice. Эта функция должна быть переопределена, если пакет обновления вносит какие-то изменения в табличную часть счета и в алгоритм хранения/расчета цен.
FUNCTION `getsum_offers` (offer INT) RETURNS double(10,2)	Аналогичная функция для предложений
FUNCTION `getsum_sales` (invoice INT) RETURNS double(10,2)	Аналогичная функция для продаж
FUNCTION `getuserbalance` (userid int) RETURNS double(10,2)	Возвращает текущий баланс клиента userid. Рассчитывается как сумма платежей минус сумма продаж.
FUNCTION `getuserbalancefordate` (userid int,df date,dt date) RETURNS double(10,2)	Возвращает баланс клиента userid за период с даты df по дату dt.
FUNCTION `getvaluefordate` (list varchar(32),fieldname varchar(32),id_value int,for_date datetime) RETURNS varchar(255)	Возвращает значение поля fieldname записи id_value справочника list, актуальное на дату for_date. Внимание – текущее значение эта функция не возвращает, если оно не было сохранено в истории! Поэтому в системе везде используется конструкция select ifnull(getvaluefordate(...),field) from table, которая возвращает текущее значение поля в случае, если истории для него нет.
FUNCTION `getvaluefordocument` (list varchar(32),fieldname varchar(32),id_value int,for_document int,doc_type varchar(32)) RETURNS varchar(255)	Возвращает значение поля fieldname записи id_value справочника list, актуальное на дату создания документа for_document типа doc_type.
FUNCTION `getvalueforevent` (list varchar(32),fieldname varchar(32),id_value int,for_event int) RETURNS varchar(255)	Возвращает значение поля fieldname записи id_value справочника list, актуальное на дату, на которую назначена задача for_event.

ЧАСТЬ 2 СОЗДАНИЕ МОДУЛЕЙ ДЛЯ INDEX.CRM

Конфигурации index.crm для конкретных заказчиков

Формирование конфигурации

Конфигурация системы, разработанная для какого-либо конкретного заказчика, представляет собой надстройку над базовой версией index.crm, которая может включать следующие элементы:

1. Дополнительные поля справочников и журналов документов. Эти поля создаются при помощи интерфейса администратора и автоматически добавляются в соответствующие таблицы базы данных. Описания полей хранятся в таблице list_props.
2. Условия фильтрации по справочникам и журналам документов. Условиями фильтрации могут быть собственно поля, либо произвольные SQL-выражения.
3. Группы пользователей, организационные подразделения, собственно пользователи (справочник структуры организации – Персонал), и набор прав доступа.
4. Шаблоны печати.
5. Типы задач.
6. Дополнительные модули системы и требуемые для их работы новые таблицы БД.
7. Расширения существующих модулей системы.

ВНИМАНИЕ! Идентификаторы

- дополнительных полей справочников и журналов документов,
- модулей системы,
- условий фильтрации,
- типов задач,
- а также номера пакетов и идентификаторы документов (см. ниже),
нужно получать в index.art! Иначе будут возникать конфликты между различными пакетами.

Изменения объединяются в «пакеты обновления», которые с соблюдением определенного регламента (см. ниже) устанавливаются на базовую версию системы. После их установки получается конфигурация системы, специфичная для данного заказчика. Об автоматическом создании пакетов обновления см. раздел **«Ошибка! Источник ссылки не найден.»**.

Расширение существующих модулей происходит путем порождения на основе существующих классов новых, содержащих реализацию конкретных функций, и подмены вызова оригинального модуля на вызов измененного. Таким образом, при обновлении базовой версии исходный модуль может быть заменен, не нарушая функциональность порожденного модуля.

При расширении модуля функция обновления производит также изменения в шаблоне модуля и в шаблонах печати. Также при необходимости заменяется/добавляется модуль, подготавливающий данные для печати.

Регламент обновления

Существует три типа обновлений системы: класса А, В и С.

1. Доработки, которые войдут в базовую версию системы (код класса А). Фактически, такие пакеты обновления предназначены для апгрейда с одной подверсии системы на другую.
2. Доработки, которые не войдут в базовую версию системы, но будут предлагаться клиентам в качестве дополнительной функциональности (код класса В).
3. Доработки, которые разработаны эксклюзивно для какого-либо клиента (код класса С). Это наиболее распространенный тип доработок, постоянно встречающийся при внедрении системы у заказчиков.

Ниже приведены правила формирования пакетов обновления.

1. Каждая логически целостная доработка должна оформляться в виде пакета обновления, состоящего из:

- Word'овского файла с описанием сути доработки, перечнем всех измененных файлов.
- каталога Update, содержащего измененные файлы, находящиеся в соответствующих подкаталогах. Например, если изменен файл include/sales.ext, он должен находиться в папке Update/include/sales.ext.
- скрипта update.sql, содержащего sql-запросы, производящие необходимые изменения в базе данных (включая изменения таблицы modules, т.е. добавление/изменение шаблонов модулей). Выполнение этих запросов не должно уничтожать никакие пользовательские данные в БД. Скрипт должен учитывать возможность того, что данный пакет обновления уже установлен, и перед попыткой записать новые данные в базу должен удалять записи, которые, возможно, существуют (например, в таблице modules).
- при необходимости – скрипта update.php, который будет запущен после update.sql и может выполнить любые дополнительные операции по работе с БД или файлами.
- при необходимости – скрипта check_install.php, который проверяет возможность установки данного пакета обновления (необходимо в случае, если данный пакет зависит от других пакетов).

Скрипты update.php и check_install.php должны быть рассчитаны на работу из подкаталога Update. Т.е. при установке пакета обновления инженер скопирует каталог Update в каталог системы, и выполнит оба скрипта. Для доступа к БД эти скрипты должны включать файл /include/defines.ext.

2. Каждому пакету обновления присваивается уникальный, последовательный номер (номера получать в index.art). Для пакетов класса А, после установки пакета система получает соответствующее изменение версии (об этом должен заботиться скрипт update.sql; номер версии хранится в таблице constants, запись version), т.е. после установки пакета 24 система будет иметь версию 1.0.24. Пакеты обновления могут устанавливаться только в последовательности, т.е. пакет 24 не может быть установлен на систему с версией 1.0.22. Пакеты классов В и С должны создавать запись в таблице updates, занося туда следующую информацию: идентификатор пакета (получать в index.art), название пакета (строка), версия пакета, дата первичной установки, дата последнего обновления

3. Если для обновления требуется создание документа в таблице documents, идентификатор документа получать в index.art. Идентификатор фиксированный, и в скрипте обновления БД должен быть прописан явным образом. Идентификаторы начинаются с 1.

Пакетам обновления классов В и С запрещается модифицировать файлы программных модулей, входящих в базовую версию системы, но разрешается модифицировать документы базовой версии. Если необходимо добавить дополнительную функциональность к модулю, входящему в базовую версию, пакет обновления должен сделать следующее (на примере расширения функциональности модуля sales):

1. Создать новый программный модуль, например sales_packagename. В документе, откуда вызывается модуль sales, заменить его вызов на sales_packagename (SQL-запросом UPDATE documents SET code=REPLACE(...) из update.sql).

2. Файл модуля package.ext должен включать директиву include_once("sales.ext"). Класс sales_packagename должен расширять класс sales:

```
Class sales_packagename extends sales {
...
}
```

3. Скрипт update.sql или update.php должен автоматически модифицировать шаблон модуля sales в соответствии с нуждами модуля sales_packagename, и записывать модифицированный шаблон модулю sales_packagename.

Пакеты класса В или С могут быть рассчитаны на то, что какой-то другой модуль класса В или С уже установлен. В этом случае пакет обновления должен содержать скрипт check_install.php, который проверит наличие необходимых пакетов и выведет соответствующие сообщения об ошибке в случае их отсутствия. Если пакет зависит от другого пакета, расширяющего тот же базовый модуль, что и данный, то пакет должен расширять уже расширенный класс (т.е. class sales_package2 extends sales_packagename).

Если пакет модифицирует именно базовый класс (sales), во избежание конфликтов с другими пакетами обновлений в скрипт check_install.php желательно включить проверку на то, что базовый класс уже не расширен каким-либо другим пакетом.

Также скрипт check_intall.php может содержать проверку базовой версии системы (например, не меньше 1.0.24).

Порядок установки пакетов обновления: сначала устанавливаются пакеты класса А (номер версии системы доводится до актуального), затем устанавливаются все необходимые данному заказчику пакеты класса В, затем пакеты класса С.

Пакеты исправления ошибок

Особым классом пакетов являются пакеты исправления ошибок. Они не имеют номеров и рассчитаны на однократную установку на уже работающую систему. Состав пакета может быть таким же, как для пакета обновления, т.е. может включать измененные файлы модулей, а также скрипты update.sql и update.php.

Пакеты исправления могут быть предназначены для базовой версии системы, или для установки поверх модифицированной конфигурации. Исправления для базовой версии выпускаются только index.art, исправления для пакетов обновления могут выпускаться авторами пакетов. В любом случае, пакет исправления должен составляться с учетом того, что он может устанавливаться и на систему, модифицированную какими-то другими пакетами, и проверять наличие всех изменяемых данных. При составлении пакета исправления нужно придерживаться принципа минимизации вносимых изменений.

Базовые классы типов модулей

Для понимания способа работы модулей системы необходимо ознакомиться с информацией, приведенной в разделе ЧАСТЬ 1 АРХИТЕКТУРА ЯДРА СИСТЕМЫ.

Система index.cfm имеет три типа модулей: справочники, журналы документов и отчеты. Конкретные модули системы расширяют базовый класс соответствующего типа модулей. Эти базовые классы, в свою очередь, основаны на интерфейсе module:

```
interface module {  
    function GetUID();  
    function PrintLabel();  
}
```

Отметим, что в системе могут существовать страницы, на которых работают программные модули, не относящиеся ни к одному из указанных типов. Например, дополнительный пакет «Управление проектами», наряду с журналом документов «Проекты», содержит программный модуль «работа с проектом», который функционирует на отдельной странице. Специальных правил построения таких модулей не существует, они могут реализовывать любую функциональность. К работе системы разграничения доступа эти модули отношения не имеют, в случае обращения к данным каких-то других модулей системы они должны заботиться о безопасности самостоятельно.

Рассмотрим сначала способ работы модулей – справочников и журналов документов.

В начале формирования страницы, на которой расположен тот или иной модуль, сначала происходит вызов конструктор класса модуля. Конструктор выполняет ряд операций, важнейшие из которых – проверка прав доступа и обработка submit'a (т.е. того запроса пользователя, который привел к перезагрузке страницы).

Затем начинается обработка шаблона модуля. Шаблон состоит из нескольких фрагментов, соответствующих функциональным областям страницы. Два фрагмента должны присутствовать в шаблоне модуля обязательно:

DisplayNavigation – вывод списка записей (слева);

ManagementForm – вывод формы редактирования (справа).

Для документов также является обязательным фрагмент

LinksSet – область, содержащая ссылки «Вы можете перейти к...»

Среди других часто встречающихся фрагментов –

Bookmarks – закладки;

Linkages – список задач;

BindingForm – форма привязки записи.

Программист может также создавать любые другие фрагменты.

Каждому фрагменту соответствует одноименный метод в классе модуля (о языке шаблонов см. раздел «Синтаксис шаблонов программных модулей»). Этот метод подготавливает массив данных, который сливается с шаблоном, в результате чего образуется HTML-код фрагмента.

Смысл разбиения страницы на фрагменты состоит в том, что после выполнения каких-либо операций (выбор записи в списке, сохранение свойств записи) можно не перезагружать всю страницу, а обновить только некоторые фрагменты. Это делается автоматически средствами системы. Программный модуль должен только указать список фрагментов, которые должны обновляться при выполнении операции смены записи или ее сохранения.

Приведем определения базовых абстрактных классов типов модулей.

Справочники

```
abstract class lists implements module {
    protected $item;           // идентификатор текущей записи
    protected $errorcode;     // описание последней ошибки (выводится в строке статуса)
    protected $config;        // экземпляр класса TableEditor – редактор записи
    protected $settings;      // экземпляр класса Settings – настройки интерфейса пользователя
    protected $control;       // экземпляр класса ListControl – навигационный список
    protected $access;        // уровень доступа к модулю в целом
    protected $access_current; // уровень доступа к текущей записи
    public $nav;               // текущая закладка
    public $name;              // название текущей записи
    public $loaded;           // флаг «запись загружена»

    public $table;            // название основной таблицы БД для справочника
    public $first_col;        // идентификатор (из filters_conditions) первой колонки списка
    public $first_col_name;   // заголовок первой колонки списка записей
    public $default_cols;     // перечень (через ;) идентификаторов других колонок списка по
    // умолчанию
    public $fragments;        // перечень фрагментов шаблона, обновляемых при загрузке
    // записи
    public $support_dfilters;  // флаг «поддерживает динамические фильтры»
    public $iddoc;            // идентификатор документа, в котором работает модуль

    public function lists($iddoc);
    // Конструктор: обрабатывает динамические фильтры, проверяет права доступа, загружает
    // текущую запись, пользовательские настройки для модуля, вызывает обработчики групповых
    // операций с записями и управления списком записей.

    public function DisplayNavigation();
    // Обработчик фрагмента шаблона, соответствующего списку записей. Подготавливает и
    // отображает список записей и панель инструментов.

    abstract public function ManagementForm();
    // это объявление требует, чтобы в каждом модуле была реализована функция
    // ManagementForm, соответствующая фрагменту шаблона, содержащему форму
    // редактирования записи.

    public function HandleSetCols();
    // обрабатывает операцию выбора столбцов для отображения в списке

    protected function HandlePrintList(); // Обработчик операции «Печать списка записей»

    protected function HandlePrintAll(); // Обработчик операции «Печатать все»

    function HandleDeleteAll(); // Обработчик операции «Удалить все»
```

```
function serializeQueryString($filter=false,$order=false,$date=false);
// Возвращает строку URL, соответствующую текущему состоянию справочника (включает
текущую выбранную запись, закладку, условия фильтра и сортировки).

}
```

Журналы документов

Класс полностью аналогичен предыдущему, за исключением следующих отличий:

```
abstract class documents implements module {

protected $syear; protected $smoonth; protected $sday;
protected $eyear; protected $emoonth; protected $eday;
// переменные, в которых хранятся даты начала и окончания периода, установленного для
вывода документов в списке.

protected function HandleAddTask();
// обрабатывает операцию создания задачи на основе выбранного документа

function Linkages();
// Метод, подготавливающий данные для фрагмента шаблона «Список задач по документу»
(в нижней части страницы)

}
```

Принципиального отличия между этими классами нет, т.к. с точки зрения программной архитектуры разделение модулей на справочники и журналы документов достаточно условно (фактически оно сводится к небольшой разнице в наборе свойств – записи справочников обязательно имеют название, а документы дату, – и к различиям в роли этих записей в системе (документы могут быть основанием и результатом для задач)).

Отчеты

Базовый класс отчетов называется reports.

Функция	Описание
Абстрактные методы: ShowForm(); ShowReport(); getParams();	Отображение формы настройки отчета Отображение собственно отчета Сериализация параметров отчета для передачи в модуль печати
Конструктор(\$iddoc)	Инициализирует переменные-члены класса, хранящие даты начала и конца периода отчета
GetPeriodForm()	Возвращает массив данных для слияния с интерфейсом PeriodForm – формой выбора периода
GetPeriodCondition(\$field)	Возвращает SQL-условие, которая форма выбора периода налагает на поле \$field (которое имеет тип date или datetime)
GetUTSPeriodCondition(\$field)	Возвращает SQL-условие, которая форма выбора периода налагает на поле \$field (которое имеет тип int и хранит дату и время в формате UTS)
GetFromTo()	Возвращает массив данных для вывода фразы «с ... по ...»
ShowPrintButton()	Возвращает массив данных для фрагмента шаблона, который выводит кнопку экспорта отчета в Excel
ShowError()	Возвращает массив с информацией об ошибке, происшедшей при построении отчета.

Создание модуля системы

Мы рекомендуем создавать модули при помощи автоматической процедуры, описанной выше. Создание модуля системы вручную содержит следующие шаги.

1. Создание программного модуля (Интерфейс администратора -> Программные модули). Имя модуля должно быть уникальным (для примера возьмем имя `modulename`). Шаблон модуля можно создать на основе существующего модуля, наиболее близкого к создаваемому по функциональности.
2. Создание класса модуля. В каталоге `/include` надо создать файл `modulename.ext`, который должен содержать класс `modulename`, порожденный от одного из трех базовых классов – `lists`, `documents` или `reports` – в зависимости от типа модуля. Как правило, можно взять за основу существующий модуль, наиболее близкий создаваемому по функциональности.
3. Далее в Интерфейсе администратора, разделе «Разделы и документы», необходимо в контейнере, соответствующем типу модуля, создать новый документ. Имя документа будет отображаться как название модуля в меню системы.
4. В тело документа нужно вставить вызов программного модуля: ``. В свойствах документа установить переключатель «Активен», чтобы новый модуль отображался в меню системы. Шаблон изменить на «Обычная страница».
5. Новый модуль нужно зарегистрировать в разделе «Разделы разработчика -> Модули системы» Интерфейса администратора. Нужно создать новую запись, в поле «Название» указать имя модуля, которое будет видеть пользователь, в поле «Код» указать имя программного модуля (`modulename`), ввести имя основной таблицы БД, с которой работает модуль (эта таблица будет использоваться при проверке прав доступа), выбрать тип, указать документ, из которого вызывается модуль.
6. В разделе «Права доступа групп» Интерфейса администратора предоставить группам пользователей права доступа к новому модулю.

После выполнения этих операций новый модуль станет доступен в системе.

Настройка модуля

Здесь мы кратко прокомментируем основные действия, выполняемые методами программного модуля, которые могут быть модифицированы программистом с целью изменения логики работы модуля.

Конструктор

Конструктор нуждается в настройке в том случае, если требуется обрабатывать какие-либо параметры, которые могут быть переданы в модуль через URL. Пример такой настройки приведен ниже, при описании настройки метода `DisplayNavigation` (в панель инструментов добавляется новая кнопка, нажатие на которую обрабатывается в конструкторе).

Кроме того, в конструкторе, перед вызовом конструктора родительского класса, могут быть изменены переменные массива `$_REQUEST`. Это имеет смысл, например, в том случае, если нужно при создании новой записи присвоить какому-нибудь полю значение по умолчанию:

```
If($_REQUEST["record_create"]) {  
    If(!$_REQUEST["field_name"])  
        $_REQUEST["field_name"]="значение по умолчанию";  
}
```

или произвести любые другие манипуляции с введенными пользователем данными до того, как они попадут в базу данных.

Также в конструктор можно вставить отправку различных уведомлений, автоматическое создание задач при определенных условиях и т.д.

В случае, если справочник или журнал документов содержит закладки, в конструктор должно быть добавлено создание этих закладок и обработка `submit'a` на закладках. Обычно это делается примерно так:

```
$bms=Array("Продажа","Товары");           // список названий закладок  
$this->bm=new Bookmarks();                 // создание экземпляра класса Bookmarks  
$this->bm->SetBookmarks($bms);             // установка закладок
```

```

$this->bm->HandleBookmark(); // обработка перехода по закладкам
$this->bookmark=$this->bm->GetCurrentBookmark(); // получение текущей закладки
$this->setup_config(); // вызов вспомогательного метода, создающего
и настраивающего экземпляр класса RecordEditor для текущей записи
switch($this->nav=$this->bm->GetCurrentBookmark()) { // в зависимости от закладки...
  case 1:
    $this->bookmark_1(); // вызываем конструктор первой закладки
    break;
  case 2:
    $this->bookmark_2(); // или второй
    break;
}
$this->setup_navigation(); // фрагмент конструктора, относящийся к
обработке навигации

```

При существенной доработке конструктора желательно разбить эти доработки на части. В приведенном выше примере используются методы `setup_config`, `setup_navigation`, `bookmark_1` и `bookmark_2`. В дальнейшем при расширении функциональности модуля новыми порожденными классами будет проще переопределить эти методы, нежели весь конструктор в целом.

ManagementForm

Этот метод формирует данные для фрагмента страницы, содержащего форму редактирования записи. Настройка метода включает указание некоторых значений в массиве данных и может выглядеть, например, так (для журнала документов):

```

function ManagementForm() {
  global $connid,$auth; // Идентификатор соединения с БД и класс Authorisator
  $arr=Array();
  $arr["typename"]="Продажа"; // Название типа записи
  $arr["qs"]=$this->serializeQueryString(true,true); // Подготовка строки URL
  $arr["iddoc"]=$this->iddoc; // Идентификатор страницы модуля
  $arr["client"]=$this->client; // Текущий клиент (для ссылки)
  $arr["cleantarget"]="idi"; // Название параметра URL, где хранится id текущей записи.
  // Очищается JS-скриптами при переходе к другой записи.
  if($this->access==1) $arr["view"]=true;
  // Флаг «только чтение»
  if($this->item>0) {
    // Если выбрана текущая запись – формируем название
    // записи и помещаем его в заголовок формы
    $SQLStmt="SELECT number,DATE_FORMAT(date,'%d.%m.%Y') `sdate` FROM sales WHERE
id='".$this->item."'";
    if(!$result=sql_query($SQLStmt,$connid)) report_error("Ошибка получения названия
продажи");
    if($row=sql_fetch_array($result)) {
      if($row["number"])
        $title=$row["number"];
      $title.=" от ".$row["sdate"];
    }
  }
  if(!$title)
    $title="Новый"; // Иначе – запись называется «Новый» (документ)
  $arr=array_merge($arr,$this->config->RecordEditor($this->item,false,$title));
  // Сливаем имеющийся массив с тем, что выдает TableEditor
  $arr["title"]=$title; // Присваиваем заголовок форме
  $arr["skipform"]=true; // Очень важный флаг. Его нужно установить в случае
  // использования метода «В» перезагрузки страницы
  // (см. ниже в описании метода DisplayNavigation)
}

```

```
return $arr;  
}
```

DisplayNavigation

Один из наиболее сложных моментов, касающихся работы модулей справочников и журналов документов – настройка метода DisplayNavigation, управляющего отображением списка записей. Общий алгоритм навигации по странице какого-либо модуля состоит в следующем:

1. Когда пользователь выбирает в списке новую запись, может происходить одно из следующих действий:

А) перезагрузка страницы с передачей в URL параметра `idi`=идентификатор записи;

Б) загрузка в форму редактирования данных информации о новой записи при помощи AJAX

В) и/или загрузка фрагментов страницы (включая форму редактирования – фрагмент ManagementForm), без перезагрузки страницы.

Обычно схема применения этих методов такова: если страница модуля не содержит закладок и других вспомогательных элементов (только форму редактирования записи), используется метод Б. Если требуется обновить закладки, список задач по документу или другие фрагменты страницы, используется метод В. Если требуется полностью обновить страницу из-за смены типа записи или автоматического перехода на другую закладку, используется метод А.

2. Если пользователь выбирает какую-либо функцию при помощи кнопки, расположенной на панели инструментов, срабатывающая при этом JS-функция может отобразить какой-либо диалог, а затем произвести перезагрузку страницы с каким-либо параметром. Обработчики передаваемых параметров вызываются в конструкторе модуля, причем их выполнение может привести к еще одному обновлению страницы.

3. При переходе на другую закладку всегда происходит полная перезагрузка страницы.

Как правило, требуется в той или иной степени перегрузить метод DisplayNavigation, объявленный в родительском классе, причем полной замены этого метода желательно избежать. Более того: самостоятельно работать метод родительского класса не может, т.к. должен получить определенную информацию от метода дочернего класса. Поэтому код метода в дочернем классе делится на две части: до вызова `parent::DisplayNavigation()` и после.

```
function DisplayNavigation() {  
    global $head_code,$onload_code,$script_code,$auth,$connid;  
    // глобальные переменные движка: код, включаемый в раздел HEAD страницы, включаемый  
    в обработчик body onload, включаемый в тег <script> в HEAD (обычно содержит функции  
    JavaScript). Также экземпляр класса Authorisator и идентификатор соединения с БД.
```

```
    $this->default_cols="14;23";
```

```
    // Здесь через точку с запятой указываются идентификаторы столбцов, которые пользователь  
    увидит в списке записей по умолчанию. Идентификаторы соответствуют условиям  
    фильтрации, определенным для данного модуля, и берутся из таблицы filters_conditions.  
    Пользователь может выбрать другие столбцы для отображения, поэтому в методе базового  
    класса происходит запрос настроек пользователя, относящихся к этому модулю, и если там  
    выбраны какие-либо столбцы – отображаются именно выбранные.
```

```
    $this->first_col="DATE_FORMAT(date,'%d.%m.%Y)";
```

```
    // SQL-выражение для отображения в первой колонке. Первая колонка списка не может быть  
    отключена или перемещена на другую позицию. Должно существовать условие фильтрации,  
    соответствующее данной колонке, и указываемое здесь SQL-выражение должно  
    соответствовать выражению этого условия фильтрации.
```

```
    $this->first_col_name="Дата";
```

```
    // Заголовок первого столбца списка
```

```
    $this->first_col_id=85;
```

```
// Идентификатор (из filters_conditions) выражения для первого столбца списка
```

```
$this->fragments=""ManagementForm','Linkages','Bookmarks','LinksSet"";  
// Перечень фрагментов, которые должны быть перезагружены при выборе новой записи в  
списке или сохранении свойств записи.
```

ВНИМАНИЕ! Фрагменты, содержащие таблицу свойств какой-либо записи (формируемые при помощи метода TableEditor->RecordEditor()), должны иметь название, начинающееся на «Manage»: ManagementForm, ManagePerson и т.д. Это необходимо для их корректной обработки динамическим загрузчиком.

После этих операций обычно следует вызов метода родительского класса:
\$arr=parent::DisplayNavigation();

Затем должна быть объявлена JS-функция refreshList, которая вызывается после изменения свойств записи для обновления списка записей.

```
AddScriptCode(" function refreshList(id) {\n  
  if(id<=0) List".$this->table.".Build(0,null,null);  
  else List".$this->table.".Build(0,\"folder\",id);  
  \n } \n var fragments=[".$this->fragments."];\n");
```

Здесь же, как мы видим, в JavaScript передается список фрагментов, подлежащих обновлению при смене/модификации записи.

Для модулей, работающих с записями нескольких типов (например, с иерархическими деревьями, где есть контейнеры и дочерние элементы), а также для модулей, содержащих закладки, может потребоваться модификация поведения страницы при выборе новой записи. Для иерархических деревьев необходимо произвести перезагрузку страницы в случае, если выбрана запись другого типа (был выбран контейнер, а выбрали дочерний элемент) – в этом случае надо заменить форму редактирования записи и ряд свойств, хранящихся в теле страницы. Для обычных справочников и документов нужно перезагружать страницу в случае, если пользователь находится не на первой закладке, а при выборе другой записи требуется перейти на первую (т.е. например пользователь просматривает табличную часть счета, а при выборе другого счета должен попасть на первую закладку со свойствами этого счета). В таких случаях надо переопределить JS-функцию show".\$this->table, которую записывает в глобальную переменную \$script_code метод DisplayNavigation родительского класса. В приведенном ниже примере функция переопределяется для двух случаев: когда пользователь на первой закладке, и на любой другой. В функции URL-адрес текущей страницы обрабатывается (из него удаляются параметры, которые могут передаваться в модуль через URL – такие как printall, передаваемый при вызове функции печати всех документов, и др). В случае нахождения на первой закладке вызывается JS-функция LoadTarget, которая обращается к серверу при помощи AJAX и загружает данные в форму редактирования записи. В случае нахождения на любой другой закладке происходит перезагрузка страницы с выбором новой записи.

```
if($this->nav==1)  
  $script_code=preg_replace("/show".$this->table."\\(TreeId,ClickType,id,Type,Open\\)  
  \\{.*\\}/isU","showsales(TreeId,ClickType,id,Type,Open) {\n  
    var target=document.location.toString();  
    target=target.replace(/&created_onbase=[^&]*/ig,"");  
    target=target.replace(/&close_task=[^&]*/ig,"");  
    target=target.replace(/&printall=[^&]*/ig,"");  
    target=target.replace(/&result=[^&]*/ig,"");  
    target=target.replace(/&errorcode=[^&]*/ig,"");  
    LoadTable(id,'idi',true,target);  
  }\n",$script_code);  
else
```

```

    $script_code=preg_replace("/show".$this->table."\\(TreeId,ClickType,id,Type,Open\\)
    \{.*\}/isU","showsales(TreeId,ClickType,id,Type,Open) {\n
    var target=document.location.toString();
    target=target.replace(/&created_onbase=[^&]*/ig,"");
    target=target.replace(/&close_task=[^&]*/ig,"");
    target=target.replace(/&printall=[^&]*/ig,"");
    target=target.replace(/&idi=[^&]*/ig,"");
    target=target.replace(/&bookmark=[^&]*/ig,"");
    target=target.replace(/&result=[^&]*/ig,"");
    target=target.replace(/&errorcode=[^&]*/ig,"");
    document.location=target+'&idi='+id;
    }\n",$script_code);

```

Завершающая часть метода DisplayNavigation содержит вызов нескольких необходимых функций:

```

    $arr["control"]=$this->control->PrintHTML();
    // подготовка HTML-кода органа управления «список»
    $arr["view"]=$this->access_current==2?false:true;
    // установка флага, управляющего правами доступа – только просмотр или редактирование
    $arr["buttons"]=$this->control->ShowToolbar();
    // вывод HTML-кода панели инструментов

```

Осталось рассмотреть случай, когда программисту требуется добавить новую кнопку для отображения на панели инструментов модуля. В этом случае после вызова parent::DisplayNavigation() нужно поместить следующий код:

```

    $this->control->AddButtonAfter("Выбрать
    столбцы",23,"/img/m_buts/order_1.gif","img/m_buts/order_2.gif","addShortOrder()","Зарегистр
    ировать заявку");
    // Эта функция добавит кнопку «Зарегистрировать заявку» после кнопки «Выбрать столбцы».
    Для кнопки нужно создать два изображения: обычное (order_1.gif) и появляющееся при
    наведении (order_2.gif). Ширина кнопки – 23 пиксела. При нажатии на кнопку будет
    вызываться JS-функция addShortOrder().

```

Далее нужно определить саму функцию addShortOrder(). Пусть в нашем случае она отображает диалоговое окно, а затем обновляет страницу с параметрами, введенными пользователем в окне:

```

    AddHeaderCode(" function addShortOrder() {
    var fp=new Object();
    result=showModalDialog('/dialogs/AddOrder.php',fp,\"resizable: yes; help: no; status: no;
    scroll: no;\");
    if(fp.name)
    document.location="".$this-
    >serializeQueryString(true,false,true)."&addorder=true&name='"+fp.name;
    }\n");

```

Теперь нужно добавить обработчик параметра addorder в конструктор модуля:

```

    if($_REQUEST["addorder"]) {
    ...
    header("Location: /?id=".$this->iddoc."&idi=".$this->item);
    exit;
    // выполнить что требуется и снова обновить страницу, чтобы параметры исчезли из
    строки URL
    }

```

Создание модуля интерфейса администратора

Модуль интерфейса администратора лучше всего создавать на основе какого-либо существующего модуля. Наиболее типичной задачей является создание модуля, предназначенного для управления каким-либо справочником. Для примера можно взять модуль `/config/measures.php`, который реализует управление таблицей «Единицы измерения». Этот модуль содержит простейшую проверку прав доступа (является ли пользователь вебмастером) и обращение к классу `TableEditor` (не `TableEditorEngine!`), для вывода и обработки формы редактирования записи справочника. В качестве шаблона вывода используется файл `/config/standardcode.ext`, который выводит HTML-код, сгенерированный классом `TableEditor`, в виде страницы интерфейса администратора.

Для выбора записей может использоваться либо модуль, написанный пользователем, либо стандартный модуль `/config/Engine/lister.php`, который выводит все записи какой-либо таблицы. Этот модуль получает следующие параметры:

<code>listname</code>	Заголовок списка
<code>idfield</code>	Поле-идентификатор в отображаемой таблице
<code>namefield</code>	Поле-название в отображаемой таблице
<code>orderfield</code>	Порядок сортировки записей
<code>tablename</code>	Название таблицы, из которой выводятся записи
<code>targetscript</code>	Скрипт, предназначенный для редактирования записей
<code>targetframe</code>	Фрейм, в котором он отображается (<code>main</code>)
<code>findname</code>	Имя, введенное в форму поиска записи по имени
<code>filtertable</code>	Это и следующие поля предназначены для отображения дополнительного <code><select></code> 'а в форме поиска записи. Этот селект содержит записи из какой-либо другой таблицы, на которую ссылается текущая (например, группы пользователей, если мы работаем с таблицей пользователей).
<code>filterkey</code>	Поле из исходной таблицы, ссылающееся на <code>filtertable</code>
<code>filtername</code>	Поле «название» в <code>filtertable</code>
<code>filtercond</code>	Условие выборки записей из <code>filtertable</code>
<code>filterdispl</code>	Отображаемое название поля-указателя

Для отображения нового модуля в меню «Конфигурация» интерфейса администратора нужно добавить его в раздел «Специальные разделы». Для этого нужно создать там новую запись, заполнив поля формы следующим образом: «Название» – имя, под которым раздел появится в меню «Конфигурация», имя модуля – имя файла модуля без расширения (например, `measures`), вызываемый скрипт – скрипт, строящий список записей во фрейме навигации (это поле может содержать вызов `Engine/lister.php` со всеми необходимыми параметрами), установить флаг «Меню «Конфигурация»».

Пример создания конфигурации для заказчика

Для примера рассмотрим создание простой конфигурации, которая подразумевает выполнение наиболее часто встречающихся действий по доработке системы.

Формулировка задачи

Заказчику нужно отслеживать заявки на поставку продукции, поступающие от клиентов. Каждая заявка имеет следующие свойства: дата, заказчик, текстовое описание требуемой продукции, менеджер (сотрудник компании заказчика, который должен обработать заявку). После ввода заявки выбранному менеджеру должна создаваться задача на формирование коммерческого предложения клиенту, которая появится в его календаре.

В справочнике «Клиенты» нужно добавить дополнительное информационное поле «Область деятельности». Под формой редактирования клиента нужно вывести количество полученных от него заявок и сделать ссылку «Зарегистрировать заявку от клиента».

Кроме того, нужно создать отчет, который будет отображать количество заявок, выданных на обработку каждому менеджеру.

Создание журнала документов «Заявки»

Первое действие, которое мы должны выполнить для создания такой конфигурации – создать журнал документов «Заявки». Прделав все операции, указанные выше в разделе «Создание модуля системы», мы получим новый журнал документов «Заявки клиентов», имя модуля – orders. Новый модуль будем создавать на основе существующего журнала документов payments, имеющего похожую на требуемый модуль структуру. Кроме того, нам нужно создать таблицу базы данных orders:

```
CREATE TABLE `orders` (  
  `id` int(11) NOT NULL auto_increment,  
  `date` date default NULL,  
  `client` int(11) default NULL,  
  `manager` int(11) default NULL,  
  `descr` text,  
  `del` int(11) default '0',  
  PRIMARY KEY (`id`)  
) ENGINE=InnoDB DEFAULT CHARSET=cp1251;
```

Этот SQL-скрипт необходимо будет включить в файл update.sql нашего пакета.

Скопировав файл payments.ext в orders.ext, приступаем к изменениям. В первую очередь, заменяем строку "payment" на "order", чтобы автоматически переименовать сам класс и изменить все названия, связанные с платежами.

Первым методом, который нуждается в изменении, является метод setup_config, где настраивается форма свойств записи. Изменим первые три строки этого метода:

```
$fields=Array("id","date","client","descr","manager");  
$names=Array("", "Дата", "Клиент", "Описание", "Менеджер");  
$types=Array(INTEGER_TYPE,DATE_TYPE,SELECT_POPUP,TEXT,POINTER);
```

(На самом деле, можно ограничиться описанием в скрипте только полей date, client и manager, а поле descr сделать пользовательским – создать его в разделе «Конфигурация -> Справочники» Интерфейса администратора, а затем включить в файл update.php запрос на добавление записи в таблицу list_props. При этом приведенное выше описание таблицы (запрос CREATE TABLE) нужно оставить без изменений).

Описание поля «клиент» уже имеется ниже:

```
$this->config->DescribeSelectPopupField("client","/dialogs/FindClient.php",$this->  
>item.",'orders',"clients","id","name");
```

А описание поля «Менеджер» придется добавить:

```
$this->config->DescribePointerField("manager","users","id","name"," WHERE ugroup  
IN($manager_groups) ");
```

(не забыв добавить в начало метода global \$manager_groups – эта константа содержит идентификаторы групп пользователей, которые выполняют функции менеджеров). Если не нужно ограничивать доступных для выбора в списке пользователей только менеджерами, вводить последний параметр не надо.

Описание поля type, наоборот, удалим.

Из следующего метода, set_defaults, удалим блок if(\$_REQUEST["client"]) { } и строку \$this->config->SetFieldValue("type","Выписка");, относящиеся к полям журнала «платежи».

В следующий метод, process_after_submit, нам нужно добавить код, создающий задачу для менеджера. Добавим туда вызов функции CreateTask перед закрытием блока if(\$_REQUEST["record_create"]) { }:

```
$SQLStmt="SELECT * FROM `orders` WHERE id='".$this->item.'";  
if(!($result=sql_query($SQLStmt,$connid))) report_error("Ошибка создания задачи");  
if($row=sql_fetch_array($result))
```

```
$contact=$wf->CreateTask(6,"Задача создана автоматически","Отправить предложение по заявке",$row["manager"],$this->module,$this->item,$row["client"],$row["date"]." 10:00");
```

Число 6 здесь – тип создаваемой задачи. Возможные типы задач перечислены в таблице event_types. Пользователь может создавать свои типы задач при помощи раздела «Справочники -> Типы задач» Интерфейса администратора. Типы задач, создаваемые пакетами, должны иметь фиксированные идентификаторы, полученные в index.art.

Далее нужно сделать ручную замену цифры 9 (идентификатор модуля «платежи») на выражение \$this->module – идентификатор нашего нового модуля.

В функции PrintLabel заменяем слово «Платежи» на слово «Заказы», в ManagementForm – «Платеж» на «Заказ».

В функции DisplayNavigation нужно заменить идентификаторы полей, отображаемых по умолчанию в списке записей:

```
$this->default_cols="16;17";
```

Идентификаторы (из таблицы filters_conditions) должны принадлежать диапазону, выделенному для этого пакета. Несколько идентификаторов указываются через точку с запятой.

Далее необходимо указать параметры первой колонки списка:

```
$this->first_col="DATE_FORMAT(date,'%d.%m.%Y')";
```

```
$this->first_col_name="Дата";
```

```
$this->first_col_id=83;
```

Первая строка – SQL-выражение для отображения в столбце (должно соответствовать выражению, существующему в filters_conditions), вторая – заголовок столбца, третья – идентификатор выражения из таблицы filters_conditions.

Нужно иметь в виду, что навигационный контрол попытается отображать в списке заявок наименования клиентов, актуальные на дату создания заявки. Чтобы историю нашей новой таблицы можно было получить, нужно добавить такую строку в оператор case в stored function SQL-сервера getvaluefordocument:

```
when 'orders' then SELECT date into tdate FROM orders WHERE orders.id=for_document;
```

При автоматическом создании пакета эта операция выполняется автоматически.

В данном случае больше ничего модифицировать в PHP-файле не нужно.

Шаблон модуля берем от модуля payments без изменений, только заменяем слово «платеж» на «заявка».

Теперь модуль готов! Зайдя в него и создав новую заявку, убеждаемся, что при этом автоматически создается задача, с которой менеджер может перейти на создание предложения.

Модификация справочника

Нам нужно модифицировать существующий справочник «Клиенты», добавив туда новое поле и разместив новую ссылку на странице «Свойства клиента».

Задача добавления нового поля решается без использования программирования. Нужно зайти в Интерфейс администратор, раздел «Конфигурирование справочников», выбрать справочник «Клиенты». В форме создания поля заполнить следующие поля: Название – «Область деятельности», имя поля в SQL – "area", тип – «Строка». Новое поле будет автоматически создано в базе данных и станет доступным для пользователей. Если созданных таким образом полей несколько, порядок их следования можно изменять, варьируя числовое значение в поле «Приоритет» – чем больше приоритет, тем выше будет расположено поле. Если установить переключатель «Дополнительное поле», такое поле не будет отображаться на странице при ее открытии, а станет видимым только после щелчка по ссылке «Дополнительное поле >>».

Чтобы разместить на странице справочника «Клиенты» новую ссылку, нужно внести изменения в шаблон программного модуля clients (Интерфейс администратора -> Программные модули -> clients).

Первая закладка («Свойства клиента») начинается после условия `{?nav==1}`. Сначала идет включение интерфейса формы управления записью в справочнике, затем, если какая-то запись выбрана, отображается форма печати и список дополнительных ссылок. Вставим нашу ссылку перед формой печати:

```
{?id}
```

```
<a href="/?id=34&client={!id}">Создать заявку</a><br><br>
```

```
{%PrintForm}<br><br>
```

34 – идентификатор документа, в котором вызывается модуль заявок. Напомним, идентификаторы документов распределяются централизованно.

Теперь добавим вывод сообщения о количестве уже зарегистрированных заявок от данного клиента. Для этого нужно расширить функциональность существующего модуля clients. Так как вносить изменения в классы, входящие в базовую версию, запрещено, нам нужно создать новый класс, расширяющий класс clients. Сделаем это в файле clients_custom.ext (обычно вместо custom используется название заказчика).

Так как фрагмент, который нам нужно изменить, находится внутри фрагмента шаблона, относящегося к методу ManagementForm, нам нужно перегрузить этот метод. Шаблон изменим таким образом:

```
{?id}
```

```
Получено заявок: {!orders}<br>
```

```
<a href="/?id=34&client={!id}">Создать новую заявку</a><br><br>
```

```
{%PrintForm}<br><br>
```

А в файле clients_custom.ext напишем следующее:

```
include_once("clients.ext");
```

```
class clients_custom extends clients {
```

```
function ManagementForm() {
    global $connid;
    $arr=parent::ManagementForm();
    $SQLStmnt="SELECT COUNT(*) FROM orders WHERE client='".$this->item.'";
    if(!($result=sql_query($SQLStmnt,$connid))) report_error("Error getting print templates");
    if($row=sql_fetch_array($result))
        $arr["orders"]=$row[0];
    return $arr;
}
```

```
}
```

Теперь нужно переименовать программный модуль clients в clients_custom, и заменить вызов модуля в документе id=5 на ``.
Задача выполнена!

Создание отчета

Осталось выполнить последнюю часть задачи – создать отчет. Повторяем процедуру создания модуля системы, которую уже делали при создании журнала документов. Пусть отчет называется report_orders. За основу возьмем отчет «Продуктивность персонала».

Дальнейшая задача сводится к упрощению существующего модуля. Так как в форме параметров отчета не будет выбора менеджера, из метода ShowForm убираем выборку менеджеров. Также убираем заполнение выпадающего меню для формы построения графика. В методе GetReport пишем нужный нам запрос:

```
function GetReport() {
    global $connid,$manager_groups;
    $arr=Array();
    if($this->access<2) return $arr;
    if(!$this->year) $this->GetPeriodForm();
    $SQLStmt="SELECT name,
    (SELECT COUNT(*) FROM orders WHERE manager=users.id AND ".$this-
    >GetPeriodCondition("date").") `orders`
    FROM users WHERE id IN ( ".$manager_groups.") ORDER BY name";
    $arr["report"]=Array(); $nr=0;
    if(!($result=sql_query($SQLStmt,$connid))) report_error("Error building report");
    while($row=sql_fetch_array($result))
        $arr["report"][$nr++]=$row;
    $arr=array_merge($arr,$this->GetFromTo());
    $arr["nsmmonth"]=$this->smonth;
    $arr["nemoth"]=$this->emonth;
    $arr["iddoc"]=$this->iddoc;
    $this->Ok = (@$_REQUEST["show"] && $nr > 0);
    return $arr;
}
```

В остальных методах убираем ненужные строки.

В шаблоне убираем выбор менеджера из формы параметров отчета, полностью убираем форму построения графика, модифицируем таблицу данных под наш отчет:

```
{#ShowReport}
{?report}
<table border=0 cellpadding=5 cellspacing=0 class="reptable">
<tr>
<th>Менеджер</th>
<th>Заявки</th>
</tr>
{:report}
<tr{?odd} bgcolor="#E9E8E8"{?/odd}>
<td nowrap>{!name}</td>
<td align=right>{!orders}</td>
</tr>
{/report}
</table>
{?/report}
{#/ShowReport}
```

Осталось реализовать экспорт отчета в Excel. Открываем раздел «Шаблоны печати» Интерфейса администратора, копируем шаблон отчета «Производительность пользователя», создаем новый шаблон «Отчет по заявкам» и вставляем туда скопированный шаблон. Таблица отчета для нашего случая будет выглядеть так:

```
<Table>
<Column ss:AutoFitWidth="0" ss:Width="200"/>
<Column ss:AutoFitWidth="0" ss:Width="80"/>
<Row>
<Cell ss:StyleID="s46"><Data ss:Type="String">Отчет по заявкам
за период с {!sday} {!smonth} {!year} по {!eday} {!emonth} {!eyear}</Data></Cell>
</Row>
```

```

<Row>
  <Cell ss:StyleID="s35"><Data ss:Type="String">Менеджер</Data></Cell>
  <Cell ss:StyleID="s35"><Data ss:Type="String">Заявки</Data></Cell>
</Row>
{:report}
<Row>
  <Cell ss:StyleID="s41"><Data ss:Type="String">{!name}</Data></Cell>
  <Cell ss:StyleID="s41"><Data ss:Type="String">{!orders}</Data></Cell>
</Row>
{:/report}
</Table>

```

Файл report_orders.ext, подготавливающий данные для отчета и находящийся в /dialogs/print_forms, должен содержать следующую функцию:

```

function report_orders ($iddoc) {
  global $connid, $auth, $connid, $manager_groups;
  $mnp = array("
"Января", "Февраля", "Марта", "Апреля", "Мая", "Июня", "Июля", "Августа", "Сентября", "Октября", "
Ноября", "Декабря");
  $gd=getdate();
  $arr = array();
  $arr['sday'] = empty($_REQUEST['sday']) ? 1 : $_REQUEST['sday'];
  $smnth = empty($_REQUEST['smnth']) ? 1 : $_REQUEST['smnth'];
  $arr['smnth'] = $mnp[$smnth];
  $arr['syear'] = empty($_REQUEST['syear']) ? 1 : $_REQUEST['syear'];
  $arr['eday'] = empty($_REQUEST['eday']) ? $gd['day'] : $_REQUEST['eday'];
  $emonth = empty($_REQUEST['emonth']) ? $gd['mon'] : $_REQUEST['emonth'];
  $arr['emonth'] = $mnp[$emonth];
  $arr['eyear'] = empty($_REQUEST['eyear']) ? $gd['year'] : $_REQUEST['eyear'];
  $SQLstmt="SELECT name,
  (SELECT COUNT(*) FROM orders WHERE manager=users.id
  AND DATE(date) >= DATE('{$_arr['syear']}-{$_smnth}-{$_arr['sday']}')
  AND DATE(date) <= DATE('{$_arr['eyear']}-{$_emonth}-{$_arr['eday']}')
  ) `orders` FROM users WHERE id IN (". $manager_groups.") ORDER BY name";
  $arr["report"]=Array(); $nr=0;
  if(!($result=sql_query($SQLstmt,$connid))) report_error("Error building report");
  while($row=sql_fetch_array($result))
    $arr["report"][$nr++]=$row;
  $arr["rowcount"]=3+$nr;
  return $arr;
}

```

Все поставленные задачи нами выполнены. Теперь нужно оформить все сделанные изменения в виде пакета.

Формирование пакетов обновления

Первая задача – выделить все файлы, который войдут в наш пакет:

```

/include/clients_custom.ext
/include/report_orders.ext
/include/orders.ext
/dialogs/print_forms/report_orders.ext

```

Вторая задача – экспортировать из базы данных все добавленные и измененные шаблоны, и сформировать из них файл update.sql (при этом, так как обновление может устанавливаться многократно, нужно сначала удалить существующие записи)

```
DELETE FROM modules WHERE name='orders' OR name='report_orders';
```

```
INSERT INTO modules (name,template,has_tpl) VALUES('orders','...','1');
INSERT INTO modules (name,template,has_tpl) VALUES('report_orders','...','1');
```

Теперь нужно обновить шаблон модуля clients (который может быть уже переименован в clients_custom):

```
UPDATE modules SET template=REPLACE(template,' {?id}
{%PrintForm}<br><br>',' {?id}
Получено заявок: {!orders}<br>
<a href="/?id=34&client={!id}">Создать новую заявку</a><br><br>
{%PrintForm}<br><br>') WHERE name='clients' OR name='clients_custom';
```

Далее нужно добавить новое поле в таблицу clients и информацию о нем в таблицу list_props.

```
DELETE FROM list_props WHERE field='area' AND list='clients';
INSERT INTO `list_props` (name,field,type,list) VALUES ('Область
деятельности','area','varchar(255)','clients');
ALTER TABLE `clients` ADD COLUMN `area` VARCHAR(255) NULL DEFAULT NULL;
```

Далее вставляем код создания таблицы orders, приведенный выше. Теперь осталось внести нужные записи в таблицы documents и modules_menu, связанные с созданными нами модулями.

```
DELETE FROM documents WHERE id=35 OR id=34;
INSERT INTO `documents` VALUES(35,'Отчет по заявкам',NULL,'<img
src=\"config/images/report_orders.gif\" />',3,3,NULL,1,',',NULL,NULL);
INSERT INTO `documents` VALUES(34,'Заявки',NULL,'<img src=\"config/images/orders.gif\"
/>',3,2,NULL,1,',',<img src=\"images/orders.gif\" />',NULL);
```

```
DELETE FROM modules_menu WHERE id=44 OR id=43;
INSERT INTO `modules_menu` VALUES(44,'Заявки','orders',6,34,'orders',NULL);
INSERT INTO `modules_menu` VALUES(43,'Изменения в реквизитах
клиента','client_props_rep',7,28,'clients',NULL);
```

Здесь мы можем обращаться к записям по id, так как эти идентификаторы являются фиксированными.

Теперь займемся формированием файла update.php. В шаблоне этого файла есть код, который переименовывает расширенные модули – нам нужно только воспользоваться им для модификации модуля clients. Для этого нужно указать их в массиве \$modules_rename:

```
$modules_rename=Array(Array("from"=>"clients","to"=>"clients_custom"));
```

Осталось только присвоить значения переменным \$package_id, \$package_version, \$package_name, полученные при регистрации пакета.

Пакет обновления сформирован, наша работа завершена.