

Front-end разработка

Операции с данными (front-end)

Версия 8.0



Эта документация предоставляется с ограничениями на использование и защищена законами об интеллектуальной собственности. За исключением случаев, прямо разрешенных в вашем лицензионном соглашении или разрешенных законом, вы не можете использовать, копировать, воспроизводить, переводить, транслировать, изменять, лицензировать, передавать, распространять, демонстрировать, выполнять, публиковать или отображать любую часть в любой форме или посредством любые значения. Обратный инжиниринг, дизассемблирование или декомпиляция этой документации, если это не требуется по закону для взаимодействия, запрещены.

Информация, содержащаяся в данном документе, может быть изменена без предварительного уведомления и не может гарантировать отсутствие ошибок. Если вы обнаружите какие-либо ошибки, сообщите нам о них в письменной форме.

Содержание

Операции с данными (front-end)	4
Сформировать пути к колонкам относительно корневой схемы	4
Добавить колонки в запрос	5
Получить результат запроса	6
Управлять фильтрами в запросе	6
Примеры формирования путей к колонкам	7
Путь к колонке корневой схемы	7
Путь к колонке схемы по прямым связям	7
Путь к колонке схемы по обратным связям	8
Примеры добавления колонок в запрос	8
Колонка из корневой схемы	8
Агрегирующая колонка	9
Колонка-параметр	9
Колонка-функция	9
Примеры получения результатов запроса	10
Строка набора данных по заданному первичному ключу	10
Результирующий набор данных	11
Примеры управления фильтрами в запросе	12
Класс EntitySchemaQuery	14
Методы	14
Класс DataManager	29
Класс DataManager	29
Класс DataManagerItem	33

Операции с данными (front-end)

 Средний

Работа с данными во front-end модулях Creatio реализована с помощью высокоуровневого класса `EntitySchemaQuery`, который предназначен для построения запросов на выборку из базы данных.

Особенности класса `EntitySchemaQuery`:

- Запрос на выборку данных `EntitySchemaQuery` строится с учетом прав доступа текущего пользователя.
- Механизм кэширования позволяет оптимизировать выполнение операций за счет обращения к кэшированным результатам запроса без дополнительного обращения к базе данных.

Алгоритм работы с данными во front-end модулях Creatio:

1. Создайте экземпляр класса `EntitySchemaQuery`.
2. Укажите корневую схему.
3. Сформируйте путь к колонке корневой схемы для добавления колонки в запрос.
4. Создайте экземпляры фильтров.
5. Добавьте фильтры в запрос.
6. Отфильтруйте результаты запроса.

Сформировать пути к колонкам относительно корневой схемы

Основой механизма построения запроса `EntitySchemaQuery` является корневая схема. **Корневая схема** — таблица базы данных, относительно которой строятся пути к колонкам в запросе, в том числе к колонкам присоединяемых таблиц. Для использования в запросе колонки из произвольной таблицы необходимо корректно задать путь к этой колонке.

Сформировать путь к колонке по прямым связям

Шаблон формирования пути к колонке по **прямым связям**

ИмяСправочнойКолонки.ИмяКолонкиСхемыИзСправочнойСхемы.

Например, есть корневая схема `[City]` из справочной колонкой `[Country]`, которая через колонку `Id` связана со справочной схемой `[Country]`.



Путь к колонке с наименованием страны, которой принадлежит город по прямым связям `Country.Name`.
Здесь:

- `Country` — имя справочной колонки корневой схемы `[City]` (ссылается на схему `[Country]`).
- `Name` — имя колонки из справочной схемы `[Country]`.

Сформировать путь к колонке по обратным связям

Шаблон формирования пути к колонке по **обратным связям**

```
[ИмяПрисоединяемойСхемы:ИмяКолонкиДляСвязиПрисоединяемойСхемы:ИмяКолонкиДляСвязиТекущейСхемы].
ИмяКолонкиПрисоединяемойСхемы
```

Путь к колонке с именем контакта, который добавил город по обратным связям

```
[Contact:Id:CreatedBy].Name . Здесь:
```

- `Contact` — имя присоединяемой схемы.
- `Id` — имя колонки схемы `[Contact]` для установки связи присоединяемой схемы.
- `CreatedBy` — имя справочной колонки схемы `[City]` для установки связи текущей схемы.
- `Name` — значение справочной колонки схемы `[City]`.

Если в качестве колонки для связи у текущей схемы выступает колонка `[Id]`, то ее можно не указывать: `[ИмяПрисоединяемойСхемы:ИмяКолонкиДляСвязиПрисоединяемойСхемы].ИмяКолонкиКорневойСхемы`.

Например, `[Contact:City].Name`.

Добавить колонки в запрос

Колонка запроса `EntitySchemaQuery` представляет собой экземпляр класса `Terrasoft.EntityQueryColumn`. В свойствах экземпляра колонки необходимо указать основные **характеристики**:

- Заголовок.
- Значение для отображения.
- Признаки использования.
- Порядок и позицию сортировки.

Для добавления колонок в запрос предназначен метод `addColumn()`, который возвращает экземпляр добавленной в запрос колонки. Имя колонки относительно корневой схемы в методах `addColumn()` формируется согласно правилам, описанным в статье [Сформировать пути к колонкам корневой схемы](#). Разновидности метода `addColumn()` позволяют добавлять в запрос колонки с различными параметрами и представлены в таблице ниже.

Разновидности метода `addColumn()`

Тип колонки	Метод
Колонка по заданному пути относительно корневой схемы	<code>addColumn(column, columnAlias)</code>
Экземпляр класса колонки запроса	
Колонка — параметр	<code>addParameterColumn(paramValue, paramDataType, columnAlias)</code>
Колонка — функция	<code>addFunctionColumn(columnPath, functionType, columnAlias)</code>
Колонка — агрегирующая функция	<code>addAggregationSchemaColumnFunctionColumn(columnPath, aggregationType, columnAlias)</code>

Получить результат запроса

Результат выполнения запроса `EntitySchemaQuery` — коллекция сущностей Creatio. Каждый экземпляр коллекции — строка набора данных, возвращаемого запросом.

Способы получения результатов запроса:

- Вызов метода `getEntity()` для получения конкретной строки набора данных по заданному первичному ключу.
- Вызов метода `getEntityCollection()` для получения всего результирующего набора данных.

Управлять фильтрами в запросе

Фильтр — набор условий, применяемых при отображении данных запроса. В терминах SQL фильтр представляет собой отдельный предикат (условие) оператора `WHERE`.

Для создания простого фильтра в `EntitySchemaQuery` используется метод `createFilter()`, который возвращает созданный объект фильтра `Terrasoft.CompareFilter`. В `EntitySchemaQuery` реализованы методы создания фильтров специального вида.

Экземпляр `EntitySchemaQuery` имеет свойство `filters`, которое представляет собой коллекцию фильтров данного запроса (экземпляр класса `Terrasoft.FilterGroup`). Экземпляр класса `Terrasoft.FilterGroup` представляет собой коллекцию элементов `Terrasoft.BaseFilter`.

Алгоритм добавления фильтра в запрос:

- Создайте экземпляр фильтра для данного запроса (метод `createFilter()`, методы создания фильтров специального вида).
- Добавьте созданный экземпляр фильтра в коллекцию фильтров запроса (метод `add()` коллекции).

По умолчанию фильтры, добавляемые в коллекцию `filters`, объединяются между собой логической операцией `AND`. Свойство `logicalOperation` коллекции `filters` позволяет указать логическую операцию,

которой необходимо объединять фильтры. Свойство `logicalOperation` принимает значения перечисления `Terrasoft.core.enums.LogicalOperatorType` (`AND` — И, `OR` — ИЛИ).

В запросах `EntitySchemaQuery` реализована возможность управления фильтрами, участвующими в построении результирующего набора данных. Каждый элемент коллекции `filters` имеет свойство `isEnabled`, которое определяет, участвует ли данный элемент в построении результирующего запроса (`true` — участвует, `false` — не участвует). Аналогичное свойство `isEnabled` также определено для коллекции `filters`. Установив это свойство в `false`, можно отключить фильтрацию для запроса, при этом коллекция фильтров запроса останется неизменной. Таким образом, изначально сформировав коллекцию фильтров запроса, в дальнейшем можно использовать различные комбинации, не внося изменений в саму коллекцию.

Формирование путей колонок в фильтрах `EntitySchemaQuery` осуществляется в соответствии с общими [правилами формирования путей](#) к колонкам относительно корневой схемы.

Примеры формирования путей к колонкам

 Средний

Путь к колонке корневой схемы

- Корневая схема: `[Contact]`.
- Колонка с адресом контакта: `Address`.

Пример создания запроса `EntitySchemaQuery`, возвращающего значения этой колонки

```
/* Создаем экземпляр класса EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем колонку Address, задаем для нее псевдоним Address. */
esq.addColumn("Address", "Address");
```

Путь к колонке схемы по прямым связям

- Корневая схема: `[Contact]`.
- Колонка с названием контрагента: `Account.Name`.
- Колонка с именем основного контакта у контрагента: `Account.PrimaryContact.Name`.

Пример создания запроса `EntitySchemaQuery`, возвращающего значения этих колонок

```
/* Создаем экземпляр класса EntitySchemaQuery с корневой схемой Contact. */
```

```
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем справочную колонку Account. Затем добавляем колонку Name из схемы Account, на котор
esq.addColumn("Account.Name", "AccountName");
/* Добавляем справочную колонку Account. Затем добавляем справочную колонку PrimaryContact из сх
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
```

Путь к колонке схемы по обратным связям

- Корневая схема: `[Contact]`.
- Колонка с именем контакта, который добавил город: `[Contact:Id:CreatedBy].Name`.

Пример создания запроса EntitySchemaQuery, возвращающего значения этой колонки

```
/* Создаем экземпляр класса EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Присоединяем к корневой схеме еще одну схему Contact по колонке Owner и выбираем из нее коло
esq.addColumn("[Contact:Id:Owner].Name", "OwnerName");
/* К справочной колонке Account присоединяем схему Contact по колонке PrimaryContact и выбираем
esq.addColumn("Account.[Contact:Id:PrimaryContact].Name", "PrimaryContactName");
```

Примеры добавления колонок в запрос

 Средний

Колонка из корневой схемы

Пример. Добавить в коллекцию колонок запроса колонку из корневой схемы.

Пример добавления в коллекцию колонок запроса колонки из корневой схемы

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addColumn("DurationInMinutes", "ActivityDuration");
```


Аггрегирующая колонка

Пример 1. Добавить в коллекцию колонок запроса агрегирующую колонку с типом агрегации `sum`, применяющимся ко всем записям таблицы.

Пример добавления в коллекцию колонок запроса агрегирующей колонки

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addAggregationSchemaColumn("DurationInMinutes", Terrasoft.AggregationType.SUM, "ActivitiesDu
```

Пример 2. Добавить в коллекцию колонок запроса агрегирующую колонку с типом агрегации `count`, применяющимся к уникальным записям таблицы.

Пример добавления в коллекцию колонок запроса агрегирующей колонки

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addAggregationSchemaColumn("DurationInMinutes", Terrasoft.AggregationType.COUNT, "UniqueActi
```

Колонка-параметр

Пример. Добавить в коллекцию колонок запроса колонку-параметр с типом данных `TEXT`.

Пример добавления в коллекцию колонок запроса колонки-параметра

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addParameterColumn("DurationInMinutes", Terrasoft.DataValueType.TEXT, "DurationColumnName");
```

Колонка-функция

Пример 1. Добавить в коллекцию колонок запроса колонку-функцию с типом функции `LENGTH` (размер значения в байтах).

Пример добавления в коллекцию колонок запроса колонки-функции

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addFunctionColumn("Photo.Data", Terrasoft.FunctionType.LENGTH, "PhotoLength");
```

Пример 2. Добавить в коллекцию колонок запроса колонку-функцию с типом `DATE_PART` (часть даты). В качестве значения используется день недели.

Пример добавления в коллекцию колонок запроса колонки-функции

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addDatePartFunctionColumn("StartDate", Terrasoft.DatePartType.WEEK_DAY, "StartDay");
```

Пример 3. Добавить в коллекцию колонок запроса колонку-функцию с типом `MACROS`, который не требует параметризации — `PRIMARY_DISPLAY_COLUMN` (первичная колонка для отображения).

Пример добавления в коллекцию колонок запроса колонки-функции

```
var esq = this.Ext.create(Terrasoft.EntitySchemaQuery, {
    rootSchemaName: "Activity"
});
esq.addMacrosColumn(Terrasoft.QueryMacroType.PRIMARY_DISPLAY_COLUMN, "PrimaryDisplayColumnValue");
```

Примеры получения результатов запроса



Средний

Строка набора данных по заданному первичному ключу

Пример. Получить конкретную строку набора данных по заданному первичному ключу.

Пример получения конкретной строки набора данных

```
/* Получаем Id объекта карточки. */
var recordId = this.get("Id");
/* Создаем экземпляр класса Terrasoft.EntitySchemaQuery с корневой схемой Contact. */
var esq = this.Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем колонку с именем основного контакта контрагента. */
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
/* Получаем одну запись из выборки по Id объекта карточки и отображаем ее в информационном окне.
esq.getEntity(recordId, function(result) {
    if (!result.success) {
        /* Например, обработка/логирование ошибки. */
        this.showInformationDialog("Ошибка запроса данных");
        return;
    }
    this.showInformationDialog(result.entity.get("PrimaryContactName"));
}, this);
```

На заметку. При получении справочных колонок метод `this.get()` возвращает объект, а не идентификатор записи в базе данных. Чтобы получить идентификатор, необходимо использовать свойство `value`, например, `this.get('Account').value`.

Результирующий набор данных

Пример. Получить весь результирующий набор данных.

Пример получения всего набора данных

```
var message = "";
/* Создаем экземпляр класса Terrasoft.EntitySchemaQuery с корневой схемой Contact. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
/* Добавляем колонку с названием контрагента, который относится к данному контакту. */
esq.addColumn("Account.Name", "AccountName");
/* Добавляем колонку с именем основного контакта контрагента. */
esq.addColumn("Account.PrimaryContact.Name", "PrimaryContactName");
```

```

/* Получаем всю коллекцию записей и отображаем ее в информационном окне. */
esq.getEntityCollection(function (result) {
    if (!result.success) {
        /* Например, обработка/логирование ошибки. */
        this.showInformationDialog("Ошибка запроса данных");
        return;
    }
    result.collection.each(function (item) {
        message += "Account name: " + item.get("AccountName") +
            " - primary contact name: " + item.get("PrimaryContactName") + "\n";
    });
    this.showInformationDialog(message);
}, this);

```

Примеры управления фильтрами в запросе

 Средний

Пример управления фильтрами в запросе

```

/* Создание экземпляра запроса с корневой схемой Contact. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
esq.addColumn("Name");
esq.addColumn("Country.Name", "CountryName");

/* Создание экземпляра первого фильтра. */
var esqFirstFilter = esq.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL, "CountryName", "Russia");

/* Создание экземпляра второго фильтра. */
var esqSecondFilter = esq.createColumnFilterWithParameter(Terrasoft.ComparisonType.EQUAL, "CountryName", "USA");

/* Фильтры в коллекции фильтров запроса будут объединяться логическим оператором OR. */
esq.filters.logicalOperation = Terrasoft.LogicalOperatorType.OR;

/* Добавление созданных фильтров в коллекцию запроса. */
esq.filters.add("esqFirstFilter", esqFirstFilter);
esq.filters.add("esqSecondFilter", esqSecondFilter);

/* В данную коллекцию попадут объекты - результаты запроса, отфильтрованные по двум фильтрам. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {

```

```

        // Обработка элементов коллекции.
    });
}
}, this);

/* Для второго фильтра указывается, что он не будет участвовать в построении результирующего запроса. */
esqSecondFilter.isEnabled = false;

/* В данную коллекцию попадут объекты - результаты запроса, отфильтрованные только по первому фильтру. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Обработка элементов коллекции. */
        });
    }
}, this);

```

Пример использования других методов создания фильтров

```

/* Создание экземпляра запроса с корневой схемой Contact. */
var esq = Ext.create("Terrasoft.EntitySchemaQuery", {
    rootSchemaName: "Contact"
});
esq.addColumn("Name");
esq.addColumn("Country.Name", "CountryName");

/* Выбираем все контакты, в которых не указана страна. */
var esqFirstFilter = esq.createColumnIsNullFilter("Country");

/* Выбираем все контакты, даты рождения которых находятся в промежутке между 1.01.1970 и 1.01.1980. */
var dateFrom = new Date(1970, 0, 1, 0, 0, 0, 0);
var dateTo = new Date(1980, 0, 1, 0, 0, 0, 0);
var esqSecondFilter = esq.createColumnBetweenFilterWithParameters("BirthDate", dateFrom, dateTo);

/* Добавление созданных фильтров в коллекцию запроса. */
esq.filters.add("esqFirstFilter", esqFirstFilter);
esq.filters.add("esqSecondFilter", esqSecondFilter);

/* В данную коллекцию попадут объекты - результаты запроса, отфильтрованные по двум фильтрам. */
esq.getEntityCollection(function (result) {
    if (result.success) {
        result.collection.each(function (item) {
            /* Обработка элементов коллекции. */
        });
    }
}, this);

```

Класс EntitySchemaQuery

 Средний

Класс `EntitySchemaQuery` предназначен для построения запросов на выборку из базы данных.

Методы

`abortQuery()`

Прерывает выполнение запроса.

`addAggregationSchemaColumn(columnPath, aggregationType, [columnAlias], aggregationEvalType)`

Создает и добавляет в коллекцию колонок запроса экземпляр функциональной колонки `Terrasoft.FunctionQueryColumn` с заданным типом `AGGREGATION`.

Параметры

<code>{String} columnPath</code>	Путь колонки для добавления (указывается относительно <code>rootSchema</code>).
<code>{Terrasoft.AggregationType} aggregationType</code>	<div>Тип используемой агрегирующей функции.</div> <div>Возможные значения (<code>Terrasoft.AggregationType</code>)</div> <div><div>AVG</div><div>Среднее значение всех элементов.</div></div> <div><div>COUNT</div><div>Количество всех элементов.</div></div> <div><div>MAX</div><div>Максимальное значение среди всех элементов.</div></div> <div><div>MIN</div><div>Минимальное значение среди всех элементов.</div></div> <div><div>NONE</div></div>

<div>Тип агрегирующей функции не определен.</div> <div>SUM</div> <div>Сумма значений всех элементов.</div>	
<div>{String} columnAlias</div>	<div>Псевдоним колонки (необязательный параметр).</div>
<div>{Terrasoft.Aggregation EvalType} aggregation EvalType</div>	<div>Область применения агрегирующей функции.</div> <div>Возможные значения (Terrasoft.AggregationEvalType)</div> <div>NONE</div> <div>Область применения агрегирующей функции не определена.</div> <div>ALL</div> <div>Применяется для всех элементов.</div> <div>DISTINCT</div> <div>Применяется к уникальным значениям.</div>

addColumn(column, [columnAlias], [config])

Создает и добавляет в коллекцию колонок запроса экземпляр колонки Terrasoft.EntityQueryColumn .

Параметры

<div>{String/Terrasoft.Base QueryColumn} column</div>	<div>Путь колонки для добавления (указывается относительно rootSchema) или экземпляр колонки запроса Terrasoft.BaseQueryColumn .</div>
<div>{String} columnAlias</div>	<div>Псевдоним колонки (необязательный параметр).</div>
<div>{Object} config</div>	<div>Конфигурационный объект колонки запроса.</div>

addDatePartFunctionColumn(columnPath, datePartType, [columnAlias])

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции Terrasoft.FunctionQueryColumn с типом DATE_PART .

Параметры

<code>{String} columnPath</code>	Путь колонки для добавления (указывается относительно <code>rootSchema</code>).
<code>{Terrasoft.DatePartType} datePartType</code>	<div>Часть даты, используемая в качестве значения</div> <div>Возможные значения (<code>Terrasoft.DatePartType</code>)</div> <div><div>NONE</div><div>Пустое значение.</div></div> <div><div>DAY</div><div>День.</div></div> <div><div>WEEK</div><div>Неделя.</div></div> <div><div>MONTH</div><div>Месяц.</div></div> <div><div>YEAR</div><div>Год.</div></div> <div><div>WEEK_DAY</div><div>День недели.</div></div> <div><div>HOURL</div><div>Час.</div></div> <div><div>HOURL_MINUTE</div><div>Минута.</div></div>
<code>{String} columnAlias</code>	Псевдоним колонки (необязательный параметр).


```
addDatePeriodMacrosColumn(macrosType, [macrosValue], [columnAlias])
```

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции `Terrasoft.FunctionQueryColumn` с типом `MACROS`, который требует параметризации. Например, следующие N дней, 3-й квартал года, и т. д.

Параметры

<code>{Terrasoft.QueryMacrosType} macrosType</code>	Тип макроса колонки.
<code>{Number/Date} macrosValue</code>	Вспомогательная переменная для макроса (необязательный параметр).
<code>{String} columnAlias</code>	Псевдоним колонки (необязательный параметр).

```
addFunctionColumn(columnPath, functionType, [columnAlias])
```

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции `Terrasoft.FunctionQueryColumn`.

Параметры

{String} columnPath	Путь колонки для добавления (указывается относительно rootSchema).
{Terrasoft.FunctionType} functionType	<div>Тип функции.</div> <div>Возможные значения (Terrasoft.FunctionType)</div> <div><div>NONE</div><div>Тип функционального выражения не определен.</div></div> <div><div>MACROS</div><div>Подстановка по макросу.</div></div> <div><div>AGGREGATION</div><div>Агрегирующая функция.</div></div> <div><div>DATE_PART</div><div>Часть даты.</div></div> <div><div>LENGTH</div><div>Размер значения в байтах. Используется для бинарных данных.</div></div>
{String} columnAlias	Псевдоним колонки (необязательный параметр).

addMacrosColumn(macroType, [columnAlias])

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-функции Terrasoft.FunctionQueryColumn с типом MACROS , который не требует параметризации (например, текущий месяц, текущий пользователь, первичная колонка, и т. д.).

Параметры

{Terrasoft.QueryMacrosType} macroType	<div>Тип макроса колонки.</div> <div>Возможные значения (Terrasoft.QueryMacroType)</div> <div><div>NONE</div></div>
---------------------------------------	---

Тип макроса не определен.

CURRENT_USER

Текущий пользователь.

CURRENT_USER_CONTACT

Контакт текущего пользователя.

YESTERDAY

Вчера.

TODAY

Сегодня.

TOMORROW

Завтра.

PREVIOUS_WEEK

Предыдущая неделя.

CURRENT_WEEK

Текущая неделя.

NEXT_WEEK

Следующая неделя.

PREVIOUS_MONTH

Предыдущий месяц.

CURRENT_MONTH

Текущий месяц.

NEXT_MONTH	Следующий месяц.
PREVIOUS_QUARTER	Предыдущий квартал.
CURRENT_QUARTER	Текущий квартал.
NEXT_QUARTER	Следующий квартал.
PREVIOUS_HALF_YEAR	Предыдущее полугодие.
CURRENT_HALF_YEAR	Текущее полугодие.
NEXT_HALF_YEAR	Следующее полугодие.
PREVIOUS_YEAR	Предыдущий год.
CURRENT_YEAR	Текущий год.
PREVIOUS_HOUR	Предыдущий час.
CURRENT_HOUR	Текущий час.

	<div>NEXT_HOUR</div> <div>Следующий час.</div> <div>NEXT_YEAR</div> <div>Следующий год.</div> <div>NEXT_N_DAYS</div> <div>Следующие N дней. Требуется параметризации.</div> <div>PREVIOUS_N_DAYS</div> <div>Предыдущие N дней. Требуется параметризации.</div> <div>NEXT_N_HOURS</div> <div>Следующие N часов. Требуется параметризации.</div> <div>PREVIOUS_N_HOURS</div> <div>Предыдущие N часов. Требуется параметризации.</div> <div>PRIMARY_COLUMN</div> <div>Первичная колонка.</div> <div>PRIMARY_DISPLAY_COLUMN</div> <div>Первичная колонка для отображения.</div> <div>PRIMARY_IMAGE_COLUMN</div> <div>Первичная колонка для отображения изображения.</div>
{String} columnAlias	Псевдоним колонки (необязательный параметр).

```
addParameterColumn(paramValue, paramDataType, [columnAlias])
```

Создает и добавляет в коллекцию колонок запроса экземпляр колонки-параметра `Terrasoft.ParameterQueryColumn`.

Параметры

<code>{Mixed} paramValue</code>	Значение параметра. Значение должно соответствовать типу данных.
<code>{Terrasoft.DataValueType} paramDataType</code>	Тип данных параметра.
<code>{String} columnAlias</code>	Псевдоним колонки (необязательный параметр).

```
createBetweenFilter(leftExpression, rightLessExpression, rightGreaterExpression)
```

Создает экземпляр `Between`-фильтра.

Параметры

<code>{Terrasoft.BaseExpression} leftExpression</code>	Выражение, проверяемое в фильтре.
<code>{Terrasoft.BaseExpression} rightLessExpression</code>	Начальное выражение диапазона фильтрации.
<code>{Terrasoft.BaseExpression} rightGreaterExpression</code>	Конечное выражение диапазона фильтрации.

```
createColumnBetweenFilterWithParameters(columnPath, lessParamValue, greaterParamValue, paramData
```

Создает экземпляр `Between`-фильтра для проверки попадания колонки в заданный диапазон.

Параметры

<code>{String} columnPath</code>	Путь к проверяемой колонке относительно корневой схемы <code>rootSchema</code> .
<code>{Mixed} lessParamValue</code>	Начальное значение фильтра.
<code>{Mixed} greaterParamValue</code>	Конечное значение фильтра.
<code>{Terrasoft.DataValueType} paramDataType</code>	Тип данных параметра.

`createColumnFilterWithParameter(comparisonType, columnPath, paramValue, paramDataType)`

Создает экземпляр `Compare`-фильтра для сравнения колонки с заданным значением.

Параметры

<code>{Terrasoft.ComparisonType} comparisonType</code>	Тип операции сравнения.
<code>{String} columnPath</code>	Путь к проверяемой колонке относительно корневой схемы <code>rootSchema</code> .
<code>{Mixed} paramValue</code>	Значение параметра.
<code>{Terrasoft.DataValueType} paramDataType</code>	Тип данных параметра.

`createColumnInFilterWithParameters(columnPath, paramValues, paramDataType)`

Создает экземпляр `In`-фильтра для проверки совпадения значения заданной колонки со значением одного из параметров.

Параметры

<code>{String} columnPath</code>	Путь к проверяемой колонке относительно корневой схемы <code>rootSchema</code> .
<code>{Array} paramValues</code>	Массив значений параметров.
<code>{Terrasoft.DataValueType} paramDataType</code>	Тип данных параметра.

`createColumnIsNotNullFilter(columnPath)`

Создает экземпляр `IsNull`-фильтра для проверки заданной колонки.

Параметры

<code>{String} columnPath</code>	Путь к проверяемой колонке относительно корневой схемы <code>rootSchema</code> .
----------------------------------	--

`createColumnIsNullFilter(columnPath)`

Создает экземпляр `IsNull` -фильтра для проверки заданной колонки.

Параметры

<code>{String} columnPath</code>	Путь к проверяемой колонке относительно корневой схемы <code>rootSchema</code> .
----------------------------------	--

`createCompareFilter(comparisonType, leftExpression, rightExpression)`

Создает экземпляр `Compare` -фильтра.

Параметры

<code>{Terrasoft.ComparisonType} comparisonType</code>	Тип операции сравнения.
<code>{Terrasoft.BaseExpression} leftExpression</code>	Выражение, проверяемое в фильтре.
<code>{Terrasoft.BaseExpression} rightExpression</code>	Выражение фильтрации.

`createExistsFilter(columnPath)`

Создает экземпляр `Exists` -фильтра для сравнения типа `[Существует по заданному условию]` и устанавливает в качестве проверяемого значения выражение колонки, расположенной по заданному пути.

Параметры

<code>{String} columnPath</code>	Путь к колонке, для выражения которой строится фильтр.
----------------------------------	--

`createFilter(comparisonType, leftColumnPath, rightColumnPath)`

Создает экземпляр фильтра класса `Terrasoft.CompareFilter` для сравнения значений двух колонок.

Параметры

<code>{Terrasoft.ComparisonType} comparisonType</code>	Тип операции сравнения.
<code>{String} leftColumnPath</code>	Путь к проверяемой колонке относительно корневой схемы <code>rootSchema</code> .
<code>{String} rightColumnPath</code>	Путь к колонке-фильтру относительно корневой схемы <code>rootSchema</code> .

`createFilterGroup()`

Создает экземпляр группы фильтров.

`createInFilter(leftExpression, rightExpressions)`

Создает экземпляр `In` -фильтра.

Параметры

<code>{Terrasoft.BaseExpression} leftExpression</code>	Выражение, проверяемое в фильтре.
<code>{Terrasoft.BaseExpression[]} rightExpressions</code>	Массив выражений, которые будут сравниваться с <code>leftExpression</code> .

`createIsNullFilter(leftExpression)`

Создает экземпляр `IsNull` -фильтра.

Параметры

<code>{Terrasoft.BaseExpression} leftExpression</code>	Выражение, которое проверяется по условию <code>IS NOT NULL</code> .
--	--

`createIsNullFilter(leftExpression)`

Создает экземпляр `IsNull` -фильтра.

Параметры

<code>{Terrasoft.Base Expression} left Expression</code>	Выражение, которое проверяется по условию <code>IS NULL</code> .
--	--

`createNotExistsFilter(columnPath)`

Создает экземпляр `Exists`-фильтра для сравнения типа `[Не существует по заданному условию]` и устанавливает в качестве проверяемого значения выражение колонки, расположенной по заданному пути.

Параметры

<code>{String} columnPath</code>	Путь к колонке, для выражения которой строится фильтр.
----------------------------------	--

`createPrimaryDisplayColumnFilterWithParameter(comparisonType, paramValue, paramDataType)`

Создает объект фильтра для сравнения первичной колонки для отображения со значением параметра.

Параметры

<code>{Terrasoft.Comparison Type} comparisonType</code>	Тип сравнения.
<code>{Mixed} paramValue</code>	Значение параметра.
<code>{Terrasoft.DataValue Type} paramDataType</code>	Тип данных параметра.

`destroy()`

Удаляет экземпляр объекта. Если экземпляр уже удален, запишет в консоль сообщение об ошибке. Вызывает виртуальный метод `onDestroy` для переопределения в подклассах.

`enablePrimaryColumnFilter(primaryColumnValue)`

Включает фильтрацию по первичному ключу.

Параметры

<code>{String/Number} primaryColumnValue</code>	Значение первичного ключа.
---	----------------------------

`error(message)`

Записывает сообщение об ошибке в журнал сообщений.

Параметры

<code>{String} message</code>	Сообщение об ошибке, которое будет записано в журнал сообщений.
-------------------------------	---

`execute(callback, scope)`

Запрос на выполнение запроса на сервере.

Параметры

<code>{Function} callback</code>	Функция, которая будет вызвана при получении ответа от сервера.
<code>{Object} scope</code>	Контекст, в котором будет вызвана функция <code>callback</code> .

`{Object} getDefSerializationInfo()`

Возвращает объект с дополнительной информацией для сериализации.

`getEntity(primaryColumnValue, callback, scope)`

Возвращает экземпляр сущности по заданному первичному ключу `primaryColumnValue`. После получения данных вызывает функцию `callback` в контексте `scope`.

Параметры

<code>{String/Number} primaryColumnValue</code>	Значение первичного ключа.
<code>{Function} callback</code>	Функция, которая будет вызвана при получении ответа от сервера.
<code>{Object} scope</code>	Контекст, в котором будет вызвана функция <code>callback</code> .

`getEntityCollection(callback, scope)`

Возвращает коллекцию экземпляров сущности, представляющих результаты выполнения текущего запроса. После получения данных вызывает функцию `callback` в контексте `scope`.

Параметры

<code>{Function} callback</code>	Функция, которая будет вызвана при получении ответа от сервера.
<code>{Object} scope</code>	Контекст, в котором будет вызвана функция <code>callback</code> .

`{Object} getTypeInfo()`

Возвращает информацию о типе элемента.

`log(message, [type])`

Записывает сообщение в журнал сообщений.

Параметры

<code>{String Object} message</code>	Сообщение, будет записано в журнал сообщений.
<code>{Terrasoft.LogMessage Type} type</code>	Тип журнала сообщений <code>callback</code> (необязательный параметр). По умолчанию <code>console.log</code> . Возможные значения заданы перечислением <code>Terrasoft.core.enums.LogMessageType</code> .

`onDestroy()`

Удаляет все подписки на события и уничтожает объект.

`serialize(serializationInfo)`

Сериализует объект в json.

Параметры

<code>{String} serialization Info</code>	Результат в json.
--	-------------------

`setSerializableProperty(serializableObject, propertyName)`

Задаёт имя свойства объекту, если он не пустой или не является функцией.

Параметры

<code>{Object} serializable Object</code>	Сериализуемый объект.
<code>{String} propertyName</code>	Имя свойства.

`warning(message)`

Записывает предупреждающее сообщение в журнал сообщений.

Параметры

<code>{String} message</code>	Сообщение, будет записано в журнал сообщений.
-------------------------------	---

Класс DataManager

 Средний

Класс DataManager

Класс `DataManager` — синглтон, который доступен через глобальный объект `Terrasoft`. Данный класс предоставляет хранилище `dataStore`. В хранилище может быть загружено содержимое одной или нескольких таблиц базы данных.

```
dataStore: {
  /* Коллекция данных типа DataManagerItem схемы SysModule. */
  SysModule: sysModuleCollection,
  /* Коллекция данных типа DataManagerItem схемы SysModuleEntity. */
  SysModuleEntity: sysModuleEntityCollection
}
```

Каждая запись коллекции представляет запись из соответствующей таблицы в базе данных.

Свойства

`{Object} dataStore`

Хранилище коллекций данных.

`{String} itemClassName`

Имя класса записи. Содержит значение `Terrasoft.DataManagerItem`.

Методы

`{Terrasoft.Collection} select(config, callback, scope)`

Если `dataStore` не содержит коллекцию данных с именем `config.entitySchemaName`, то метод формирует и выполняет запрос в базу данных, затем возвращает полученные данные, иначе возвращает коллекцию данных из `dataStore`.

Параметры

<code>{Object} config</code>	<p>Конфигурационный объект.</p> <p>Свойства конфигурационного объекта</p> <hr/> <p><code>{String} entitySchemaName</code></p> <p>Имя схемы.</p> <hr/> <p><code>{Terrasoft.FilterGroup} filters</code></p> <p>Условия.</p>
<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения функции обратного вызова.

`{Terrasoft.DataManagerItem} createItem(config, callback, scope)`

Создает новую запись типа `config.entitySchemaName` со значениями колонок `config.columnValues`.

Параметры

<code>{Object} config</code>	Конфигурационный объект. Свойства конфигурационного объекта <hr/> <code>{String} entitySchemaName</code> Имя схемы. <hr/> <code>{Object} columnValues</code> Значения колонок записи.
<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения функции обратного вызова.

```
{Terrasoft.DataManagerItem} addItem(item)
```

Добавляет запись `item` в коллекцию данных схемы.

Параметры

<code>{Terrasoft.DataManagerItem} item</code>	Добавляемая запись.
---	---------------------

```
{Terrasoft.DataManagerItem} findItem(entitySchemaName, id)
```

Возвращает запись из коллекции данных схемы с именем `entitySchemaName` и с идентификатором `id`.

Параметры

<code>{String} entitySchemaName</code>	Название коллекции данных.
<code>{String} id</code>	Идентификатор записи.

```
{Terrasoft.DataManagerItem} remove(item)
```

Устанавливает признак `isDeleted` для записи `item`, при фиксации изменений запись будет удалена из базы данных.

Параметры

<code>{Terrasoft.DataManagerItem} item</code>	Удаляемая запись.
---	-------------------

`removeItem(item)`

Удаляет запись из коллекции данных схемы.

Параметры

<code>{Terrasoft.DataManagerItem} item</code>	Удаляемая запись.
---	-------------------

`{Terrasoft.DataManagerItem} update(config, callback, scope)`

Обновляет запись со значением первичной колонки `config.primaryColumnValue` значениями из `config.columnValues`.

Параметры

<code>{Object} config</code>	<p>Конфигурационный объект.</p> <p>Свойства конфигурационного объекта</p> <hr/> <p><code>{String} entitySchemaName</code></p> <p>Имя схемы.</p> <hr/> <p><code>{String} primaryColumnValue</code></p> <p>Значение первичной колонки.</p> <hr/> <p><code>{Mixed} columnValues</code></p> <p>Значения колонок.</p>
<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения функции обратного вызова.

`{Terrasoft.DataManagerItem} discardItem(item)`

Отменяет изменения для записи `item`, сделанные в текущей сессии работы с `DataManager`.

Параметры

{Terrasoft.DataManager
Item} item

Запись, по которой будут отменены изменения.

{Object} save(config, callback, scope)

Выполняет сохранение в базу данных коллекций данных схем, указанных в `config.entitySchemaNames`.

Параметры

{Object} config

Конфигурационный объект.

Свойства конфигурационного объекта

{String[]} entitySchemaName

Имя схемы, сохранение которой необходимо выполнить.
Если свойство не заполнено, сохранение выполняется для всех схем.

{Function} callback

Функция обратного вызова.

{Object} scope

Контекст выполнения функции обратного вызова.

Класс DataManagerItem

Свойства

{Terrasoft.BaseViewMode} viewModel

Объектная проекция записи в базе данных.

Методы

setColumnValue(columnName, columnValue)

Устанавливает новое значение `columnValue` для колонки с именем `columnName`.

Параметры

<code>{String} columnName</code>	Название колонки.
<code>{String} columnValue</code>	Значение колонки.

```
{Mixed} getColumnValue(columnName)
```

Возвращает значение колонки с именем `columnName`.

Параметры

<code>{String} columnName</code>	Название колонки.
----------------------------------	-------------------

```
{Object} getValues()
```

Возвращает значения всех колонок записи.

```
remove()
```

Устанавливает признак `isDeleted` для записи.

```
discard()
```

Отменяет изменения для записи, сделанные в текущей сессии работы с `DataManager`.

```
{Object} save(callback, scope)
```

Фиксирует изменения в базе данных.

Параметры

<code>{Function} callback</code>	Функция обратного вызова.
<code>{Object} scope</code>	Контекст выполнения функции обратного вызова.

```
{Boolean} getIsNew()
```

Возвращает признак того, что запись новая.

```
{Boolean} getIsChanged()
```

Возвращает признак того, что запись была изменена.

Примеры использования

Получение записей из таблицы [Contact]

```

/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaName: "Contact",
    /* Убирать дубли в результирующем наборе данных. */
    isDistinct: true
};
/* Получение данных. */
Terrasoft.DataManager.select(config, function (collection) {
    /* Сохранение полученных записей в локальное хранилище. */
    collection.each(function (item) {
        Terrasoft.DataManager.addItem(item);
    });
}, this);

```

Добавление новой записи в объект `DataManager`

```

/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaName: "Contact",
    /* Значения колонок. */
    columnValues: {
        Id: "00000000-0000-0000-0000-000000000001",
        Name: "Name1"
    }
};
/* Создание новой записи. */
Terrasoft.DataManager.createItem(config, function (item) {
    Terrasoft.DataManager.addItem(item);
}, this);

```

Получение записи и изменение значения колонки

```

/* Получение записи. */
var item = Terrasoft.DataManager.findItem("Contact",
    "00000000-0000-0000-0000-000000000001");
/* Установка нового значения "Name2" колонке Name. */
item.setColumnValue("Name", "Name2");

```

Удаление записи из DataManager

```

/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaName: "Contact",
    /* Значение первичной колонки. */
    primaryColumnValue: "00000000-0000-0000-0000-000000000001"
};
/* Устанавливает признак isDeleted для записи item. */
Terrasoft.DataManager.remove(config, function () {
}, this);

```

Отмена изменений, сделанных в текущей сессии работы с DataManager

```

/* Получение записи. */
var item = Terrasoft.DataManager.findItem("Contact",
    "00000000-0000-0000-0000-000000000001");
/* Отмена изменений для записи. */
Terrasoft.DataManager.discardItem(item);

```

Фиксация изменений в базе данных

```

/* Определение конфигурационного объекта. */
var config = {
    /* Название схемы сущности. */
    entitySchemaNames: ["Contact"]
};
/* Сохранение изменений в базу данных. */
Terrasoft.DataManager.save(config, function () {
}, this);

```