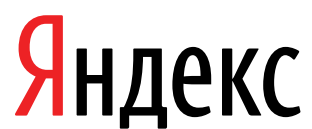


API Яндекс.Карт

Руководство разработчика

18.12.2012



API Яндекс.Карт. Руководство разработчика. Версия 2.x

Дата сборки документа: 18.12.2012.

Этот документ является составной частью технической документации Яндекса.

Сайт справки к сервисам Яндекса: <http://help.yandex.ru>

© 2008—2012 ООО «ЯНДЕКС». Все права защищены.

Предупреждение об исключительных правах

Яндекс (а также указанному им правообладателю) принадлежат исключительные права на все результаты интеллектуальной деятельности и приравненные к ним средства индивидуализации, используемые при разработке, поддержке и эксплуатации сервиса API Яндекс.Карт. К таким результатам могут относиться, но не ограничиваясь указанными, программы для ЭВМ, базы данных, изображения, тексты, другие произведения, а также изобретения, полезные модели, товарные знаки, знаки обслуживания, коммерческие обозначения и фирменные наименования. Эти права охраняются в соответствии с Гражданским кодексом РФ и международным правом.

Вы можете использовать сервис API Яндекс.Карт или его составные части только в рамках полномочий, предоставленных вам Пользовательским соглашением сервиса API Яндекс.Карт или специального соглашения.

Нарушение требований по защите исключительных прав правообладателя влечет за собой дисциплинарную, гражданско-правовую, административную или уголовную ответственность в соответствии с российским законодательством.

Контактная информация

ООО «ЯНДЕКС»

<http://www.yandex.ru>

Тел.: +7 495 739 7000

Email: pr@yandex-team.ru

Главный офис: 119021, Россия, г. Москва, ул. Льва Толстого, д. 16

Содержание

О Руководстве	4
Общие сведения	4
Подключение API	9
Карта	12
Объекты на карте	16
Активные области	22
Общие сведения	22
Картиночный слой	25
Описания активных областей	26
Источник данных	29
Создание слоя активных областей	31
Элементы управления	33
Макеты и шаблоны	37
События	38
Поиск по карте	40
Определение местоположения пользователя	42
Geo XML	43
Маршрутизатор	43

О Руководстве

Данный документ предназначен для разработчиков, использующих JavaScript API Яндекс.Карт.

Руководство рассчитано на разработчиков, знакомых с программированием на JavaScript и объектно-ориентированной парадигмой разработки.

В документе описаны основные понятия, используемые в API и описаны методы решения типовых задач, возникающих при создании интерактивных карт на страницах сайтов.

Руководство не является полным описанием API, его методы и объекты подробно описаны в Справочнике по программному интерфейсу.

Информация, представленная в документе, актуальна для JavaScript API версии 2.0.13.

Общие сведения

JavaScript API Яндекс.Карт представляет собой набор JavaScript-компонентов, предназначенных для размещения интерактивных карт на веб-страницах.

Компоненты API реализованы в виде классов, функций, статических объектов и интерфейсов.

В API используется следующее соглашение: названия всех классов начинаются с заглавной буквы ([Map](#), [Balloon](#)), при этом классы определяющие интерфейсы, начинаются с заглавной латинской буквы «I» ([IGeoObject](#)). Все остальные объекты принято именовать с маленькой буквы.

Пространства имен

Все компоненты API попадают в единое пространство имён (по умолчанию «ymaps»). Компоненты могут логически объединяться в подпространства имён второго и третьего уровней — например, `ymaps.map.Hint`, `ymaps.map.Balloon`.

Изменить пространство имен можно при [загрузке API](#) с помощью параметра `ns` (например, `ns=myNameSpace`).

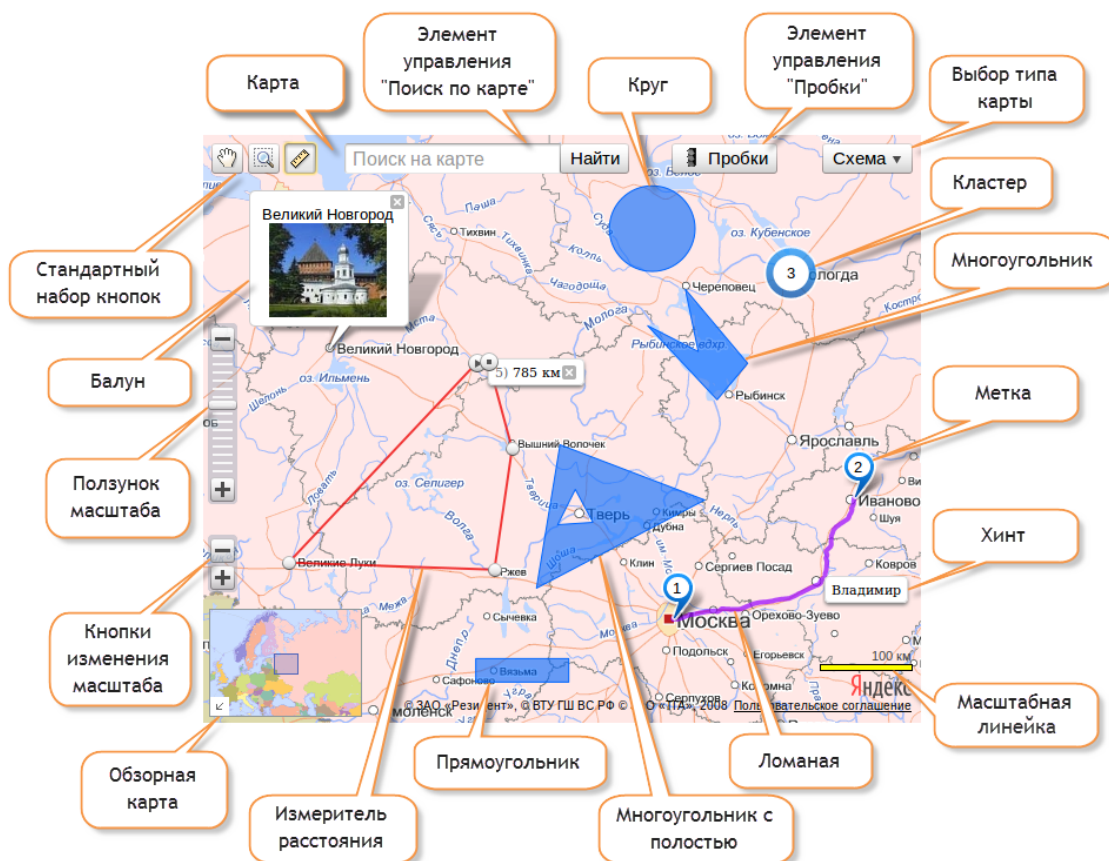
Использование глобального пространства имен позволяет избежать «пересечения» в именах компонентов API Яндекс.Карт и пользовательском коде или сторонних библиотеках.

Верстка

При формировании интерактивной карты происходит динамическое формирование HTML-кода и внедрение его в DOM-дерево страницы, на которой размещена карта.

Там, где это целесообразно, в генерируемом коде используются не стандартные HTML-элементы (`img`, `div` и т. д.), а элемент `ymaps`. Кроме того, все стилевое оформление генерируемого API кода базируется на CSS-классах, имеющих префикс «`ymaps-`», а CSS API использует селекторы только этих классов. Это позволяет свести вероятность конфликтов верстки к минимуму. Если в HTML-коде пользователя не используется элемент `ymaps`, а пользовательские CSS-определения не содержат классов с префиксом «`ymaps-`», конфликтов верстки не будет. То есть CSS пользователя не будет влиять на отображение карты и ее элементов, а CSS API не будет влиять на верстку пользователя.

Объекты карты



Иерархия объектов карты

Корневым элементом в иерархии объектов API является карта.

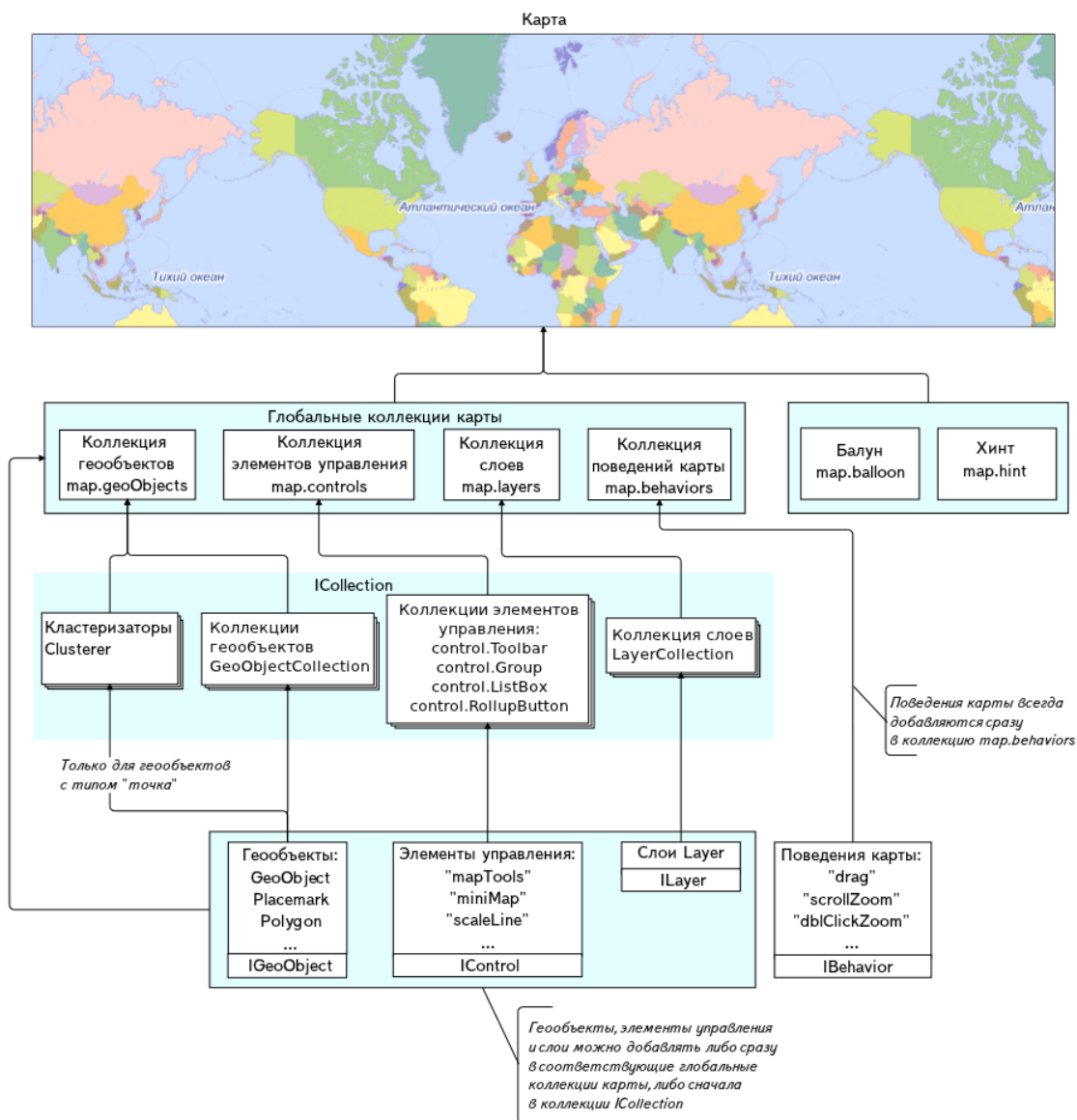
У карты есть четыре глобальных коллекции: коллекция [геообъектов](#), коллекция [элементов управления](#), коллекция [слоев](#) и коллекция [поведений](#) карты. На карте отображаются только те объекты, которые были добавлены в эти глобальные коллекции (то есть добавление объектов на карту осуществляется только через их добавление в глобальные коллекции). Соответственно для удаления объектов с карты их нужно удалить из глобальных коллекций:

`myMap.geoObjects.add(myGeoObject)` — добавление объекта на карту,

`myMap.geoObjects.remove(myGeoObject)` — удаление объекта с карты.

Перед тем как добавить объекты в глобальные коллекции, их можно сначала объединить в коллекции [ICollection](#). Например, метки ([Placemark](#)) можно объединить в коллекцию [GeoObjectCollection](#) и затем эту коллекцию добавить в [глобальную коллекцию геообъектов карты](#). После этого метки будут отображены на карте. Исключения составляют поведения, которые всегда добавляются сразу в [глобальную коллекцию поведений карты](#) и не объединяются в коллекции более низкого уровня.

Также у карты есть две отдельные сущности — [балун](#) и [хинт](#), которые не принадлежат глобальным коллекциям. На одну карту может быть добавлен только один балун (хинт).



Опции, данные и состояния

Среди свойств программных компонентов API можно выделить *опции*, *данные* и *состояние*, которые обычно обозначаются как `options`, `data` (или `properties`) и `state` соответственно.

Например, конструкторы классов [Hint](#) и [control.TrafficControl](#) выглядят следующим образом:

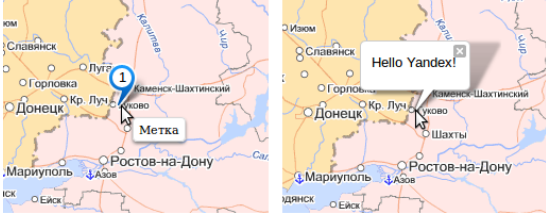
```
Hint (map, data, options)
control.TrafficControl (state)
```

Опции отвечают за визуальное отображение объектов (например, значок метки, ширина и цвет линий ломаной или многоугольника). Опции могут принимать строго определенные значения.

Данные `data` (или `properties`) — это произвольные данные, которые содержат любую информацию об объекте (например, его название, текст балуна и хинта, длина маршрута или время проезда). Данные могут содержать в себе любые поля.

Состояние описывает те свойства объекта, которые могут быть изменены с помощью визуального интерфейса (например, нажата или отжата кнопка).

Опции, данные и состояние представляются в виде JavaScript-объекта, состоящего из набора пар «ключ:значение».

Описание	Пример
<p>При наведении указателя мыши на метку показывается всплывающая подсказка с контентом <code>hintContent</code>.</p> <p>При нажатии на метку открывается балун с содержимым <code>balloonContent</code>.</p> 	<pre>var data = { balloonContent: 'Hello Yandex!', hintContent: 'Метка', iconContent: '1' }, options = {balloonHasCloseButton: true}, myPlacemark = new ymaps.Placemark([48, 40], data, options); // Добавление объекта на карту myMap.geoObjects.add(myPlacemark);</pre>

Наследование опций

Для опций реализовано иерархическое наследование. Это означает, что программный объект рекурсивно наследует опции всех своих родителей, однако в общем случае интерпретирует только часть из них.

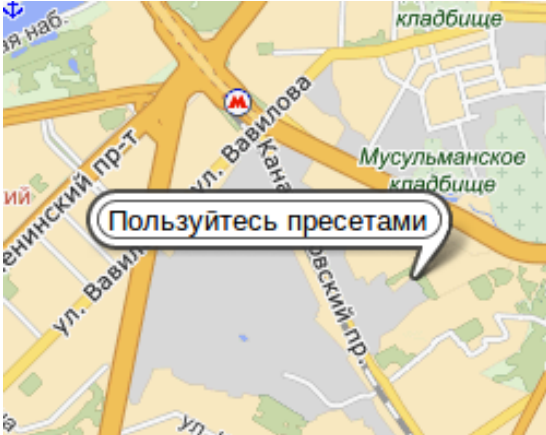
Элементом верхнего уровня в этой иерархии является карта. Через карту можно задать опции как самой карте, так и всем объектам, относящимся к ней: геообъектам, элементам управления, слоям, а также их коллекциям и т.д. Например, в опциях карты можно задать значок для отображения метки и этот значок будет использоваться по умолчанию для всех меток, размещаемых на карте.

При задании опций объектам не через карту область действия этих опций сужается. Например, если задать опцию балуна через коллекцию геообъектов [GeoObjectCollection](#) и открыть балун через экземпляр класса карты (`myMap.balloon.open()`), то опция не будет применена.

В API реализована поддержка заранее предустановленных наборов опций — т. н. *пресетов* (от англ. [preset](#)).

Для хранения пресетов существует специальное хранилище [option.presetStorage](#), позволяющее поставить пресету в соответствие именованный ключ. Хранилище представляет собой статический объект, уже содержащий пресеты, которые могут использоваться (или используются) некоторыми компонентами API.

Чтобы использовать значения, определенные в пресете, необходимо при задании опций сослаться на идентификатор пресета из этого хранилища с помощью ключа `preset`.

Описание	Пример
<p>Метка с белой иконкой, растягивающейся под контент</p> 	<pre>var myPlacemark = new ymaps.Placemark([55.7, 37.6], { iconContent: 'Пользуйтесь пресетами' }, { preset: 'twirl#whiteStretchyIcon' } // Иконка будет белой и растянется под iconContent);</pre>

Таким образом сослаться можно только на один пресет.

Пресеты являются всего лишь удобным способом задания наборов опций. Для опций, заданных с их помощью, действует стандартный иерархический механизм наследования.

Список доступных пресетов можно посмотреть в [справочнике](#).

Перед тем как объект будет отображён на карте, API сначала пытается определить опции, которые были заданы объекту напрямую. Если таковых нет, API ищет установленный для объекта пресет. Если и пресет не был установлен, то опции ищутся по иерархическому дереву вверх, до уровня карты.

Приоритетной опцией объекта является опция, заданная через него напрямую. Если объекту задать одну и ту же опцию на разных уровнях, то для этого объекта будет применена та опция, которая задана на более низком уровне.

```
// Задание опции метке через карту
myMap.options.set({
  geoObjectZIndexHover: 200 // z-index геообъекта при наведении на него
  указателя мыши
})

// Задание опции метке напрямую через объект Placemark
myMap.Placemark.options.set({
  zIndexHover: 100 // для этой метки значение опции будет 100
})
```

Использование префиксов в названиях опций

В API существует множество объектов, для которых могут быть заданы опции одинаковых назначений (например, макет балуна карты и макет балуна геообъекта). Для различия таких опций можно было бы однозначно определить их названия в рамках всего API. Однако это привело бы к образованию огромного списка опций с длинными названиями.

Поэтому для различия опций одинакового назначения в названиях этих опций используются *префиксы* (например, *balloonLayout* или *iconLayout*).

При задании опций дочерним объектам карты на уровне карты (то есть через саму карту) префиксы нужно использовать всегда. При проходе вниз по дереву иерархии на каждом уровне часть соответствующих префиксов отбрасывается. Префиксы не используются при задании опций объекту напрямую через этот объект.

Ниже приведены примеры использования префиксов для опции *layout* при задании макета определенным объектам (меткам, балуну, хинту и кластерам).

Для задания макета объекту напрямую через этот объект (например, хинту карты) опция *layout* указывается без префикса:

```
myMap.hint.show([56, 37], {'Я хинт!'}, {
  layout: MyHintLayoutClass // макет хинта карты
});
```

При задании опций определенным объектам через их коллекции (то есть на уровне выше), нужно указывать соответствующие префиксы: *icon*, *balloon*, *hint* и т.д.:

```
// Задание макета геообъектам через их коллекции
myGeoObjectCollection.options.set({
  hintLayout: MyHintLayoutClass // макет хинтов всех геообъектов из данной коллекции
  balloonLayout: MyBalloonLayoutClass // макет балунов всех геообъектов из данной коллекции
  iconLayout: MyIconLayoutClass // макет иконок всех меток из данной коллекции
  ...
});
```

Через кластеризатор можно задавать опции для меток и кластеров, входящих в него, а так же для их балунов и хинтов. Поэтому для различия опций объектов, относящихся только к кластерам используется префикс «*cluster*».

Примечание:

Если через кластеризатор все опции заданы без префикса «cluster», то эти опции применяются ко всем объектам кластеризатора, в том числе и к кластерам.

```
// Задание макета иконок меток и кластеров через кластеризатор
myClusterer.options.set({
    iconLayout: MyIconLayoutClass // макет иконок меток и кластеров, входящих в
    данный кластеризатор
    balloonLayout: ... // макет балунов всех объектов кластеризатора
    clusterBalloonLayout: ... // макет балунов только кластеров
    ...
});
```

При задании опций объектам через глобальные коллекции указываются аналогичные префиксы:

```
myMap.geoObjects.options.set({
    iconLayout: MyIconLayoutClass, // макет иконок всех объектов карты
    balloonLayout: MyBalloonLayoutClass, // макет балунов всех объектов карты
    clusterBalloonLayout: ... // макет балунов кластеров
    ...
});
```

При задании опций геообъектам, балуну и хинту через карту (то есть на самом верхнем уровне) помимо соответствующего определенному объекту префикса (icon, balloon и т.д.) нужно добавлять префикс geoObject:

```
// Задание опции объектам через глобальную коллекцию геообъектов
myMap.options.set({
    geoObjectIconLayout: MyIconLayoutClass, // макет всех меток на карте
    geoObjectBalloonLayout: MyBalloonLayoutClass, // макет балунов геообъектов
    карты
    balloonLayout: ... // макет всех балунов карты
    geoObjectClusterBalloonLayout: ... // макет балунов кластеров
    ...
});
```

Подключение API

Чтобы использовать API Яндекс.Карт, необходимо чтобы компоненты API были загружены вместе с кодом страницы как обычный внешний JavaScript-файл. Наиболее распространенным способом подключения внешних скриптов является использование элемента script в заголовке HTML-документа.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script src="http://api-maps.yandex.ru/2.0-stable/?
load=package.standard&lang=ru-RU" type="text/javascript"> </script>
  </head>
</html>
```

URL для загрузки имеет вид

```
(http|https)://api-maps.yandex.ru/<номер версии>/?lang=<идентификатор языка>
&<дополнительные параметры>
```

Нумерация версий описана в разделе [Версии API](#).

Компоненты API могут быть загружены как по протоколу HTTP, так и по HTTPS. Если сайт поддерживает работу по обоим протоколам, можно опустить явное указание схемы в атрибуте src элемента script.

```
<script src="//api-maps.yandex.ru/2.0-stable/?load=package.standard&lang=ru-RU"
type="text/javascript">
</script>
```

Параметры загрузка API

Параметр	Обязательный параметр	Описание
lang	✓	<p>Идентификатор языка — локаль.</p> <p>Задается в виде <code><language>.<region></code> в соответствии с RFC-3066.</p> <p>В настоящий момент поддерживаются следующие локали:</p> <ul style="list-style-type: none"> ru-RU — русский язык; en-US — английский язык; tr-TR — турецкий язык; uk-UA — украинский язык. <p>Задание локали определяет язык, на котором отображаются надписи на карте и элементах управления, предпочтительный язык, на котором возвращаются результаты поиска по карте и используемые по умолчанию единицы измерения.</p>
coordorder		<p>Порядок задания географических координат в функциях API, принимающих на вход пары долгота-широта (например, Placemark).</p> <p>Возможные значения:</p> <ul style="list-style-type: none"> latlong — [широта, долгота] — используется по умолчанию; longlat — [долгота, широта]. <p>Значение по умолчанию: latlong.</p>
key		API-ключ .
load	✓	<p>Список загружаемых пакетов.</p> <p>Имена пакетов перечисляются через запятую. Например, <code>load=package.standard,package.geoObjects</code>.</p> <p>Могут быть загружены как все компоненты API (<code>load=package.full</code>) так и отдельные пакеты. Это позволяет минимизировать объем трафика, передаваемого клиентскому приложению.</p> <hr/> <p>Примечание: Рекомендуется загружать пакет <code>package.standard</code>, содержащий стандартный набор компонентов для работы с API.</p> <hr/> <p>Компоненты также можно загружать «по требованию», используя функцию load.</p>

Параметр	Обязательный параметр	Описание
mode		<p>Режим загрузки API.</p> <p>Код API может быть загружен в упакованном виде для минимизации трафика и скорости исполнения в браузере (<code>mode=release</code>), а также в виде исходного кода (<code>mode=debug</code>).</p> <p>Загрузка в виде исходного кода удобна для отладки JavaScript-компонентов — код всех загруженных компонентов доступен для просмотра. Кроме того, в этом режиме в консоль выводятся сообщения об ошибках и исключениях. При загрузке в упакованном виде эти сообщения не выводятся.</p> <p>Значение по умолчанию: <code>release</code>.</p>
ns		<p>Пространство имен, в котором локализованы программные компоненты API.</p> <p>По умолчанию все объекты принадлежат пространству имен <code>ymaps</code> (например, <code>ymaps.load</code>, <code>ymaps.Map</code>). Если при загрузке API указать <code>ns=myNameSpace</code>, то объекты будут доступны уже как <code>myNameSpace.load</code>, <code>myNameSpace.Map</code>.</p> <p>Использование пространства имен позволяет избежать пересечения названий функций и прочих программных компонентов, используемых в API и пользовательском/стороннем коде.</p> <p>Значение по умолчанию: <code>ymaps</code>.</p>
onload		<p>Имя функции, которую необходимо вызвать после того, как компоненты API будут загружены и готовы к использованию (<code>callback</code>).</p> <p>Допускается использование вложенных пространств имен:</p> <pre>onload=myfunction onload=myapp.dosmth</pre> <p>Пример использования приведен в таблице ниже.</p>

Загрузка API по условию

Компоненты API также можно подключать с помощью функции [load](#), которую удобно использовать в том случае, если загрузку необходимо производить в соответствии с какими-то условиями.

```

<script src="http://api-maps.yandex.ru/2.0-stable/?load=package.map&lang=ru-
RU" type="text/javascript"></script>
<script type="text/javascript">
    if (window.location.pathname == '/traffic-page') {
        // На этой странице нужно показать пробки и инструмент поиска по карте
        ymaps.load(['package.traffic', 'package.search'], addControls);
    }
    function addControls(map) {
        map.controls.add('trafficControl').add('searchControl');
    }
</script>

```

Готовность API

Компоненты API Яндекс.Карт всегда загружаются асинхронно. Это происходит даже в том случае, если для подключения API используется тег `<script>` и никаких специальных действий для асинхронной загрузки не производилось.

Чтобы быть уверенным, что компоненты загружены и готовы к использованию, необходимо использовать функцию [ready](#) или параметр загрузки [onload](#).

Использование функции <code>ready()</code>	Использование параметра загрузки <code>onload</code>
<pre> <script src="http://...? load=package.standard&lang=ru-RU"></script> <script type="text/javascript"> var myMap; ymaps.ready(function () { myMap = new ymaps.Map("YMapsID", { center: [55.76, 37.64], zoom: 10 }); ... }); </script> <div id="YMapsID" style="width: 450px; height: 350px;"></div> </pre>	<pre> // Формируем div-контейнер карты <div id="YMapsID" style="width: 450px; height: 350px;"></div> <script type="text/javascript"> var myMap; function init() { myMap = new ymaps.Map("YMapsID", { center: [55.87, 37.66], zoom: 10 }); ... } </script> // Сразу после загрузки API будет вызвана функция init. На момент ее исполнения div- контейнер карты уже будет готов. <script src="http://...? load=package.standard&lang=ru- RU&onload=init"></script> </pre>

Возникновение событий загрузки DOM-дерева или документа не сигнализирует об окончании загрузки API. То есть использование обработчиков событий типа `document.ready`, `window.onload`, `jQuery.ready` и пр. не позволяет определить, готовы ли компоненты для использования.

Для инициализации карты необходимо, чтобы в DOM-дереве находился элемент, в котором она размещается.

Функция `ready` исполняет включенный в нее код после того, как будет загружены компоненты API и DOM-дерево документа.

Функция, переданная в параметр `onload` вызывается после загрузки API, но не отслеживает готовность DOM-дерева. В этом случае отслеживать доступность HTML-элемента, в который помещается карта, необходимо самостоятельно. Например, при помощи обработчиков событий, перечисленных выше.

Использование параметра `onload` дает возможность инициализировать карту, не дожидаясь, пока DOM будет сформирован полностью. Поэтому данный способ является самым быстрым способом загрузки API.

Карта

Создание и удаление карты

Для создания карты предназначен класс [Map](#). В конструкторе класса необходимо указать центр и коэффициент масштабирования карты и HTML-элемент, в котором она будет размещена.

Для указания HTML-элемента можно использовать как ссылку на элемент, так и его идентификатор.

```
<head>
<script type="text/javascript">
    var moscow_map,
        piter_map;

    ymaps.ready(function() {
        moscow_map = new ymaps.Map("first_map", {
            center: [55.76, 37.64],
            zoom: 10
        });
        piter_map = new ymaps.Map(document.getElementsByTagName('p')[2], {
            center: [55.76, 37.64],
            zoom: 9
        });
    });
</script>
</head>
<body>
    <p>Карта Москвы</p>
    <div id="first_map" style="width:400px; height:300px"></div>
    <p>Карта Санкт-Петербурга</p>
    <p style="width:400px; height:200px"></p>
</body>
```

Карта может быть размещена в любом [блочном HTML-элементе](#) и полностью заполняет занимаемую им прямоугольную область.

Вычисление размеров области производится в момент инициализации. Если во время вызова конструктора контейнер для размещения не сформирован в DOM-дереве или его размеры не определены, будет создана карта нулевых размеров (т.е. фактически она не будет отображена). Такая ситуация часто возникает при [размещении карты в изначально скрытом контейнере](#).

Заданные при инициализации [параметры карты](#) могут быть изменены.

Для удаления карты используется метод [destroy](#).

См. [Пример](#).

Параметры карты

Основными параметрами карты являются DOM-элемент, в котором размещается карта ([область показа](#)), участок отображаемой местности ([область картографирования](#)) и способ ее изображения ([тип карты](#)).

При создании карты необходимо обязательно задать область картографирования путем указания центра и уровня масштабирования. В дальнейшем область картографирования можно изменить с помощью методов [setCenter](#), [panTo](#), [setGlobalPixelCenter](#), [setZoom](#), [setBounds](#).

```
myMap.setCenter([55.7, 37.6], 6);
myMap.panTo([50.451, 30.522], {duration: 2000});
myMap.setBounds([[50.1, 30.2], [60.3, 20.4]]);
```

Тип карты можно указать в конструкторе или с помощью метода `setType`. Можно использовать как встроенные, так и собственные типы карт. К встроенным типам относятся (в скобках указаны соответствующие данным типам ключи):

- схема ('yandex#map'),
- спутник ('yandex#satellite'),
- гибрид ('yandex#hybrid'),
- народная карта ('yandex#publicMap'),
- народная карта в гибридном представлении ('yandex#publicMapHybrid').

Список городов, для которых доступны подробные (с точностью до домов) карты встроенных типов размещен на странице <http://maps.yandex.ru/?index>.

```
myMap.setType('yandex#publicMap');
```

См. [Пример](#).

Поведения

Карта обладает набором поведений, определяющих реакцию карты на действия, производимые пользователем. Например, при перемещении курсора мыши в области показа при нажатой левой кнопке карта сдвигается вслед за курсором.

При инициализации карты ей присваивается набор поведений, который в дальнейшем можно изменить. Доступ к поведением карты предоставляется полем `behaviors`.

```
myMap.behaviors
// Отключаем некоторые включенные по умолчанию поведения:
// - drag - перемещение карты при нажатой левой кнопки мыши;
// - rightMouseButtonMagnifier - увеличение области, выделенной
//   правой кнопкой мыши.
.disable(['drag', 'rightMouseButtonMagnifier'])
// Включаем измеритель расстояний, активирующийся при
// щелчке левой кнопкой мыши.
.enable('ruler');
```

Собственные поведения определяются с помощью реализации интерфейса `IBehavior`. Для создания собственного поведения достаточно создать класс и определить его свойства: менеджер опций поведения и менеджер событий. Также необходимо определить методы класса: `enable`, `disable`, `setParent` и `getParent`. Затем класс добавляется в хранилище поведений карты `ymaps.behavior.storage`, после чего менеджер поведений `map.behaviors` создаст экземпляр этого класса — новое поведение.

```
// Создадим и добавим на карту поведение, при котором
// при щелчке на карте происходит ее центрирование по месту клика.
// Для этого создадим класс MyBehavior и определим его свойства и методы
function MyBehavior() {
    // Определим свойства класса
    this.options = new ymaps.option.Manager(); // Менеджер опций
    this.events = new ymaps.event.Manager(); // Менеджер событий
}

// Определим методы
MyBehavior.prototype = {
    constructor: MyBehavior,
    // Когда поведение будет включено, добавится событие щелчка на карту
    enable: function () {
        /* this._parent - родителем для поведения является менеджер поведений;
        this._parent.getMap() - получаем ссылку на карту;
        this._parent.getMap().events.add - добавляем слушатель события на
        карту. */
        this._parent.getMap().events.add('click', this._onClick, this);
    },
    disable: function () {
        this._parent.getMap().events.remove('click', this._onClick, this);
    },
    // Устанавливает родителя для исходного поведения
```

```

    setParent: function (parent) { this._parent = parent; },
    // Получает родителя
    getParent: function () { return this._parent; },
    // При щелчке на карте происходит ее центрирование по месту клика
    _onClick: function (e) {
        var coords = e.get('coordPosition');
        this._parent.getMap().setCenter(coords);
    }
};

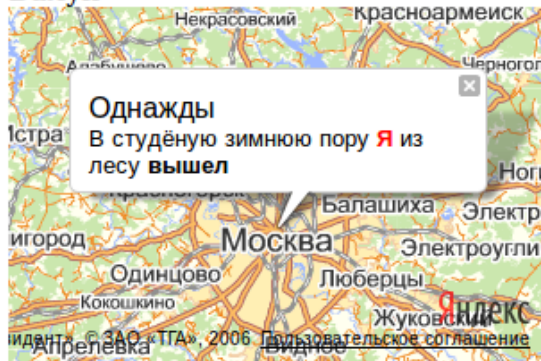
// Помещаем созданный класс в хранилище поведений. Далее данное поведение будет
// доступно по ключу 'mybehavior'.
ymaps.behavior.storage.add('mybehavior', MyBehavior);
// Включаем поведение
myMap.behaviors.enable('mybehavior');

```

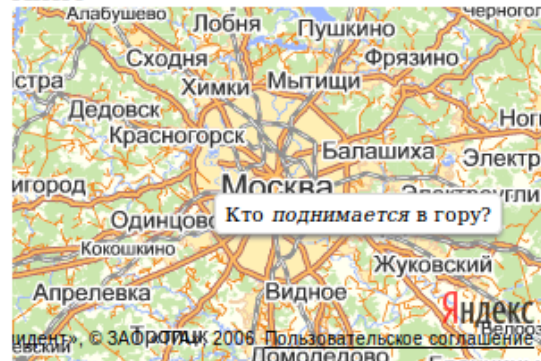
Балун и хинт

На карте может быть отображен один балун (всплывающее окно) и один хинт (подсказка).

Балун



Хинт



Балун и хинт отображаются в точке с заданными координатами, а их содержимое может содержать HTML-разметку. Ссылки на экземпляры классов балуна и хинта карты содержатся в полях [balloon](#) и [hint](#) объекта карты.

```

myMap.balloon.open(myMap.getCenter(), {
    contentHeader: 'Однажды',
    contentBody: 'В студеную зимнюю
пору' +
    ' <span style="color:red; font-
weight:bold">Я</span>' +
    ' из лесу <b>вышел</b>',
});

```

```

myMap.hint.show([55.76, 37.38],
    'Кто <em>поднимается</em> в гору?'
);

```

Доступ к балуну и хинту карты имеют геообъекты и активные области. То есть балун/хинт может быть открыт над геообъектом или активной областью без непосредственного указания координат балуна/хинта.

```

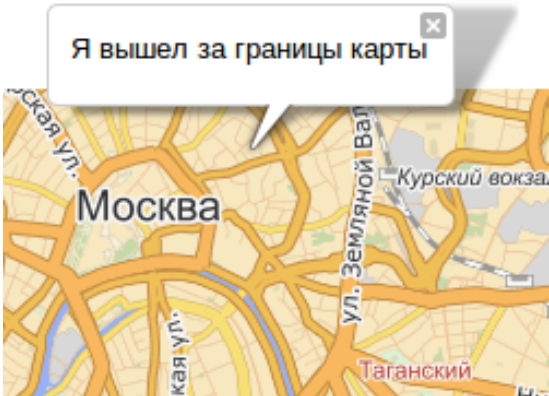
var myPlacemark = new ymaps.Placemark([55.7, 37.6], {
    balloonContentHeader: 'Однажды',
    balloonContentBody: 'В студеную зимнюю пору',
    balloonContentFooter: 'Мы пошли в гору',
    hintContent: 'Зимние происшествия'
});
// Балун откроется в точке «привязки» балуна — т. е. над меткой.
myPlacemark.balloon.open();

```

Если в момент открытия балуна на карте уже есть открытый балун, то старый балун закрывается, и открывается новый. То же самое относится и к хинту.

Внешний вид балуна и хинта удобно настраивать с помощью [опций](#) или [шаблонов](#).

Возможна ситуация, что балун выходит за границы карты. В таком случае балуну нужно задать дополнительные опции `balloonPane` и `balloonShadowPane`):

Иллюстрация примера	Пример
	<pre>var myPlacemark = new ymaps.Placemark([55.76, 37.64], { balloonContent: 'Я вышел за границы карты' }, { balloonPane: 'movableOuters', // теперь балун лежит в контейнере 'movableOuters' (не в контейнере overlays) и выходит за пределы карты balloonShadowPane: 'movableOuters' });</pre>

См. [Примеры](#).

Объекты на карте

Географическим объектам реального мира ставятся в соответствие программные объекты — *геообъекты*. К геообъектам относятся метки, круги, ломаные, прямоугольники, многоугольники, а также их коллекции.

Географические свойства геообъекта описываются с помощью [геометрии](#), которая задается геометрическим типом и координатами.

Базовым классом, реализующим геообъект является [GeoObject](#).

Экземпляр карты размещает геообъекты в собственном хранилище, реализованном в виде [коллекции](#), ссылка на которую находится в поле [geoObjects](#). Добавление геообъекта на карту, его изменение и удаление производится с помощью обращения к этой коллекции.

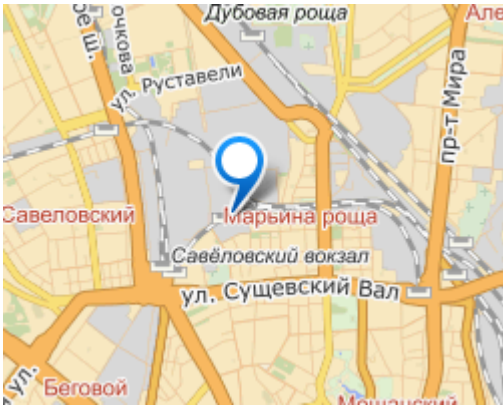
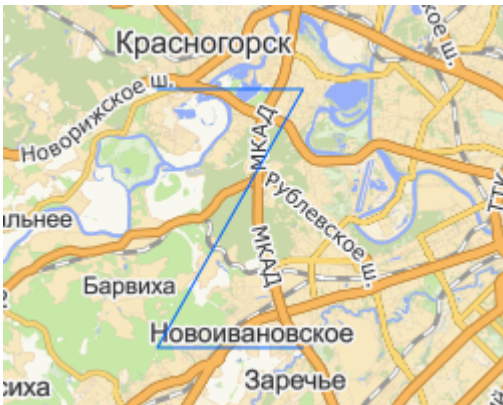
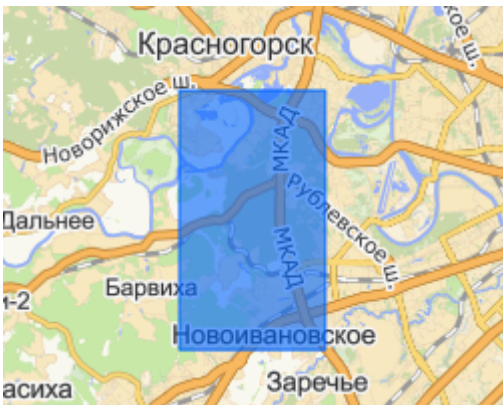
```
var myMap = new ymaps.Map("map", {
    center: [55.76, 37.64],
    zoom: 10
}),
myGeoObject = new ymaps.GeoObject({
    geometry: {
        type: "Point", // тип геометрии - точка
        coordinates: [55.8, 37.8] // координаты точки
    }
});
myMap.geoObjects.add(myGeoObject); // Размещение геообъекта на карте.
```

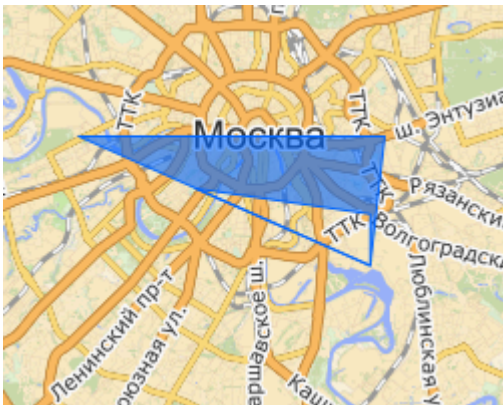
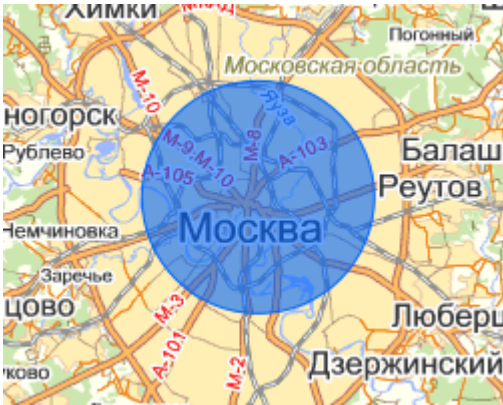
Для каждого типа геометрии определен вспомогательный класс, предоставляющий упрощенный синтаксис для создания геообъекта.

```
// Вспомогательный класс, который можно использовать
// вместо GeoObject с типом геометрии «Point» (см. предыдущий пример)
var myPlacemark = new ymaps.Placemark([55.8, 37.6]);
myMap.geoObjects.add(myPlacemark);
```


Геометрия геообъекта

В следующей таблице приведен список геометрий, их идентификаторов, используемых в классе `GeoObject` и соответствующих вспомогательных классов.

Тип геометрии / Идентификатор/ Вспомогательный класс	Пример
Точка / Point / Placemark 	Создание метки с помощью класса <code>GeoObject</code> : <pre>var myPlacemark = new ymaps.GeoObject({ geometry: { type: "Point", coordinates: [55.76, 37.56] } });</pre> Создание метки с помощью вспомогательного класса <code>Placemark</code> : <pre>var myPlacemark = new ymaps.Placemark([55.8, 37.6]);</pre>
Ломаная линия / LineString / Polyline 	Создание ломаной с помощью класса <code>GeoObject</code> : <pre>var myPolyline = new ymaps.GeoObject({ geometry: { type: "LineString", coordinates: [[...]] } });</pre> Создание ломаной с помощью вспомогательного класса <code>Polyline</code> : <pre>var myPolyline = new ymaps.Polyline([[55.80, 37.30], [55.80, 37.40], [55.70, 37.30], [55.70, 37.40]]);</pre>
Прямоугольник / Rectangle / Rectangle 	Создание прямоугольника с помощью класса <code>GeoObject</code> : <pre>var myRectangle = new ymaps.GeoObject({ geometry: { type: "Rectangle", coordinates: [[...]] } });</pre> Создание прямоугольника с помощью вспомогательного класса <code>Rectangle</code> : <pre>var myRectangle = new ymaps.Rectangle([[55.70, 37.30], [55.80, 37.40]]);</pre>
Многоугольник / Polygon / Polygon	Создание многоугольника с помощью класса <code>GeoObject</code> :

Тип геометрии / Идентификатор/ Вспомогательный класс	Пример
	<pre>var myPolygon = new ymaps.GeoObject({ geometry: { type: "Polygon", coordinates: [[...]], [...]]] });</pre> <p>Создание многоугольника с помощью вспомогательного класса Polygon:</p> <pre>var myPolygon = new ymaps.Polygon([[[55.75, 37.50], [55.75, 37.71], [55.70, 37.70]], [[55.73, 37.58], [55.72, 37.70], [55.70, 37.70]]]);</pre>
<p>Круг / Circle / Circle</p> 	<p>Создание круга с помощью класса GeoObject:</p> <pre>var myCircle = new ymaps.GeoObject({ geometry: { type: "Circle", coordinates: [55.76, 37.64], radius: 10000 } });</pre> <p>Создание круга с помощью вспомогательного класса Circle:</p> <pre>var myCircle = new ymaps.Circle([55.76, 37.64], 10000);</pre>

Тип геометрии присваивается геообъекту в момент его создания и не может быть изменен в дальнейшем. При этом координаты геометрии могут быть изменены как программно, так и пользователем с помощью [визуального редактора](#).

```
var myPlacemark = new ymaps.Placemark(
  [55.754952, 37.615319],
  {},
  {
    draggable: true, // метку можно перемещать
    preset: 'twirl#whiteStretchyIcon'
  }
);
myPlacemark.events.add('dragend', function(e) {
  // Получение ссылки на объект, который был передвинут.
  var thisPlacemark = e.get('target');
  // Определение координат метки
  var coords = thisPlacemark.geometry.getCoordinates();
  // и вывод их при щелчке на метке
  thisPlacemark.properties.set('balloonContent', coords);
});
```

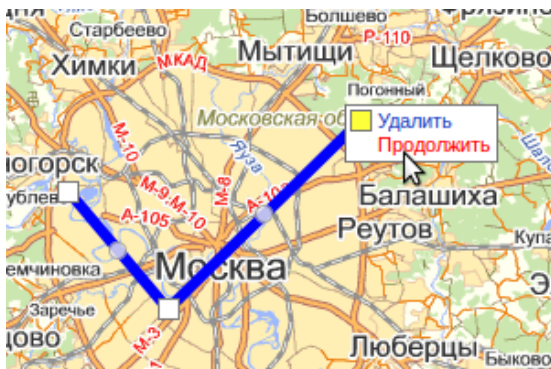
Если геообъект был инициализирован без указания геометрии, то он не будет отображаться на карте. Присвоить тип геометрии такому геообъекту также будет невозможно.

Визуальное редактирование

Географические свойства геообъекта могут быть изменены пользователем в визуальном режиме. Для этого API предоставляет два способа: перемещение геообъекта как целого и визуальный редактор.

За возможность перетаскивания геообъекта как целого отвечает опция `draggable`, которая может принимать значения `true` и `false`.

Визуальный редактор доступен для геообъектов с типом геометрии точка, ломаная и многоугольник. Ссылка на редактор содержится в поле `editor` экземпляра соответствующего класса.



```
var myPolyline = new ymaps.Polyline(
  [[55.86, 37.84], [55.70, 37.55], [55.8, 37.4]],
  {},
  {
    strokeWidth: 6,
    strokeColor: '#0000FF',
    draggable: true
  }
);

myPolyline.editor.startEditing();
```

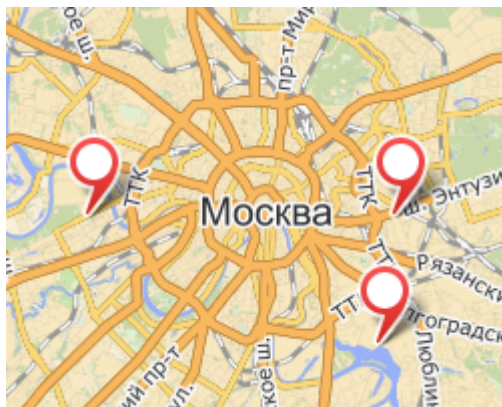
Коллекции геообъектов

Геообъекты могут объединяться в коллекции, которые позволяют определять свойства сразу всех своих членов и проводить групповые операции над ними.

В API Яндекс.Карт реализовано два вида коллекций: упорядоченные (`GeoObjectArray`) и неупорядоченные (`GeoObjectCollection`). Первые реализованы на основе массивов, вторые — на основе двусвязных списков.

Геообъект может принадлежать только одной коллекции. При добавлении геообъекта, принадлежащего одной коллекции, в другую коллекцию, он будет удален из первой.

При реализации собственных коллекций на основе этих классов рекомендуется использовать неупорядоченные коллекции во всех случаях, когда нет необходимости обращаться к элементу по порядковому номеру. Удаление элементов из неупорядоченной коллекции производится значительно быстрее, чем из упорядоченной.



```
var coords = [
  [55.75, 37.50], [55.75, 37.71], [55.70, 37.70]
],
myCollection = new
ymaps.GeoObjectCollection({}, {
  preset: 'twirl#redIcon', // все метки красные
  draggable: true // и их можно перемещать
});
// или myCollection = new
ymaps.GeoObjectArray(...);

for (var i = 0; i < coords.length; i++) {
  myCollection.add(new
ymaps.Placemark(coords[i]));
}

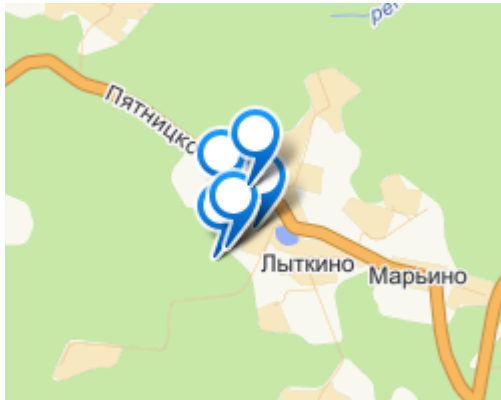
myMap.geoObjects.add(myCollection);

// При клике на карту все метки будут удалены.
myCollection.getMap().events.add('click',
function() {
  myCollection.removeAll();
});
```

Класс коллекции, которая может быть размещена на карте, должен реализовывать интерфейс `IMapObjectCollection`. Все упомянутые коллекции реализуют этот интерфейс и могут быть отображены на карте.

Кластеры

Точечные геообъекты могут находиться настолько близко друг другу, что на каком-то масштабе их метки начинают накладываться друг на друга. В этом случае пользователю приходится тщательно «прицеливаться», чтобы попасть курсором мыши на видимый фрагмент нужной метки. Существует даже специальный термин, описывающий эффекты такого рода — **pixel hunting** (**пиксель-хантинг**).



```
var coords = [
  [56.023, 36.988],
  [56.025, 36.981],
  [56.020, 36.981],
  [56.021, 36.983],
  [56.027, 36.987]
]

var myCollection = new
ymaps.GeoObjectCollection();

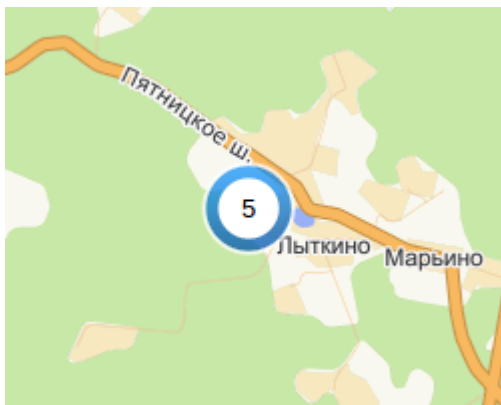
for (var i = 0; i < coords.length; i++) {
  myCollection.add(new
ymaps.Placemark(coords[i]));
}

myMap.geoObjects.add(myCollection);
```

Если метки находятся в одной точке и размеры их иконок совпадают, то наведение на перекрытые метки невозможно в принципе.

Стандартным способом решения проблемы является объединение близко расположенных объектов в группу (*кластер*) и использование для группы специальной кластерной иконки. Часто на иконке кластера указывается число содержащихся в нем элементов.

Для кластеризации объектов в API Яндекс.Карт используется класс [Clusterer](#).

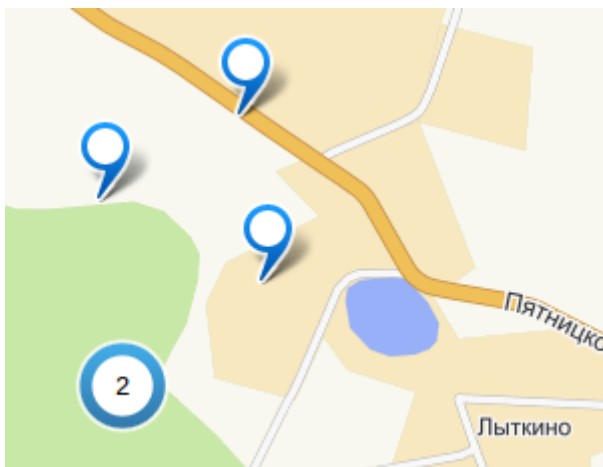


```
var myGeoObjects = [];

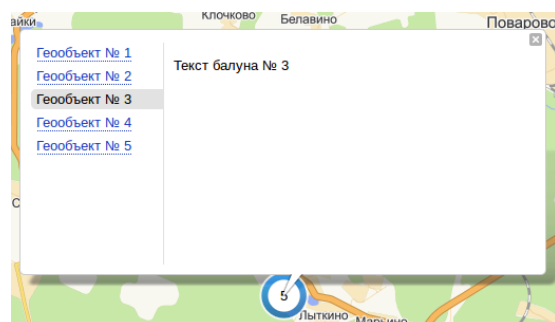
for (var i = 0; i < coords.length; i++) {
  myGeoObjects[i] = new ymaps.GeoObject({
    geometry: {
      type: "Point",
      coordinates: coords[i]
    }
  });
}

var myClusterer = new ymaps.Clusterer();
myClusterer.add(myGeoObjects);
myMap.geoObjects.add(myClusterer);
```

При увеличении масштаба кластер визуально распадается на отдельные метки и/или другие кластеры.



При щелчке на иконке кластера может произойти либо увеличение коэффициента масштабирования либо открытие балуна, в котором будет отображен список геообъектов кластера. Это поведение контролируется с помощью опции `clusterDisableClickZoom` (`true/false`).



```
var myGeoObjects = [];

for (var i = 0; i < coords.length; i++) {
    myGeoObjects[i] = new ymaps.GeoObject({
        geometry: {
            type: "Point",
            coordinates: coords[i]
        },
        properties: {
            clusterCaption: 'Геообъект № '+(i+1),
            balloonContentBody: 'Текст балуна № '+(i+1)
        }
    });
}

var myClusterer = new ymaps.Clusterer(
    {clusterDisableClickZoom: true}
);

myMap.geoObjects.add(myClusterer);
```

Примечание:

Если при отображении кластера коэффициент масштабирования карты является максимальным, то кластер не будет распадаться на отдельные метки при щелчке на его маркере. В этом случае независимо от значения параметра `clusterDisableClickZoom` всегда будет открываться балун.

Таким образом, отобразить информацию о близко расположенных геообъектах можно даже в том случае, если их метки накладываются друг на друга при максимальном коэффициенте масштабирования.

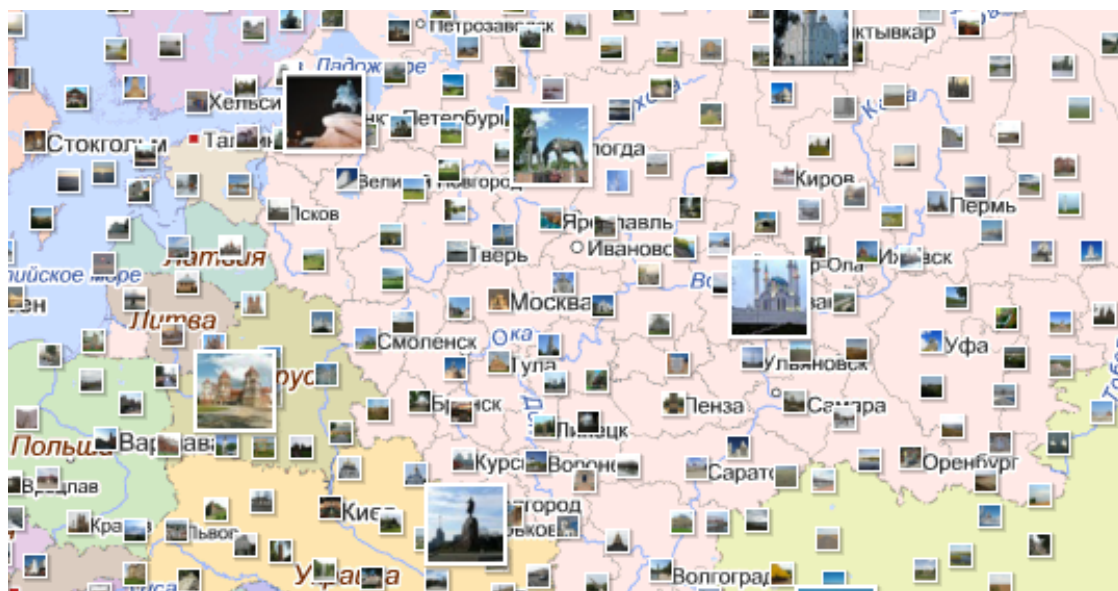
Использование кластеризатора позволяет значительно увеличить производительность при отображении большого количества геообъектов в браузере.

Отрисовка и обработка геообъекта — дорогостоящая с точки зрения потребления ресурсов операция, и чем больше объектов кластеризуется, тем больше ресурсов экономится. Сильное снижение производительности наблюдается уже при отображении нескольких сотен объектов без кластеризации.

Активные области

Общие сведения

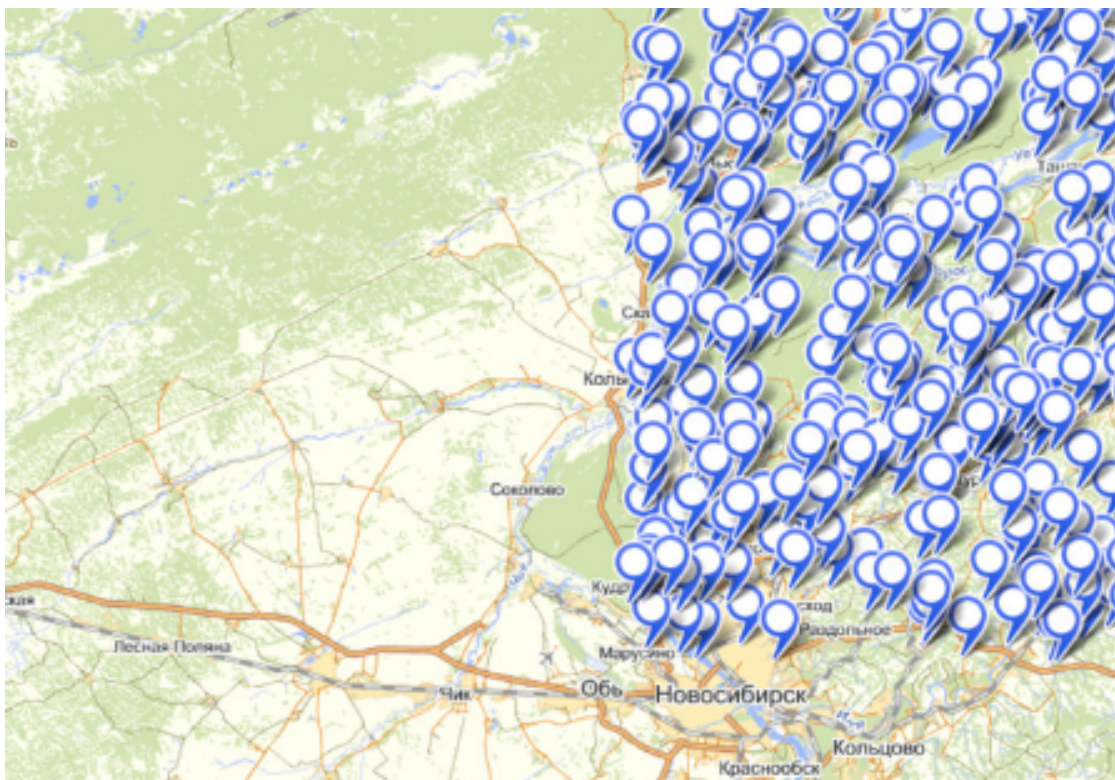
Довольно часто перед пользователем API стоит задача отобразить на карте сотни и даже тысячи геообъектов. Добавление на карту такого большого числа объектов может привести к значительной потере производительности.



Чтобы уменьшить нагрузку на браузер клиента целесообразно загружать не все данные сразу, а только те, которые нужны. Наиболее часто используется метод, при котором с сервера грузятся только те объекты карты, которые попадают в ее область просмотра.

На первый взгляд данный метод кажется достаточно эффективным, так как не происходит загрузки всех данных сразу. Однако у этого метода есть ряд недостатков.

Во-первых, перестроение объектов при смене границ карты занимает время. Каждый раз при перемещении карты приходится ждать, когда объекты появятся на карте.



Во-вторых, если границы карты сдвигаются всего на несколько пикселей, то после запроса к серверу могут вернуться те же самые данные. В связи с этим невозможно использование кэша браузера. К тому же частое перемещение карты приводит к сильной загрузке сервера.

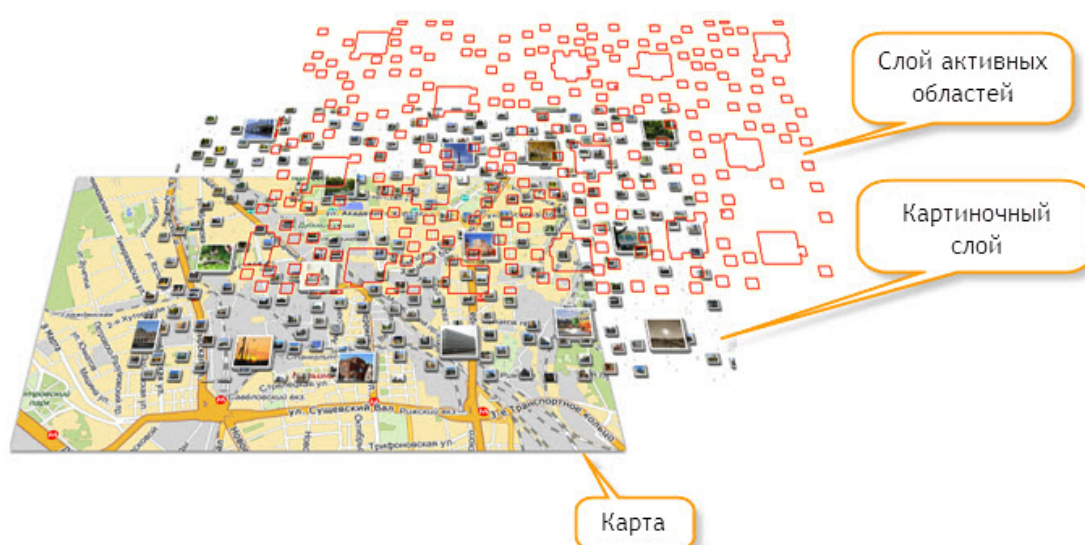
Технология активных областей — это иной способ загрузки и отображения большого числа объектов, позволяющий добиться высокой производительности.

При использовании активных областей вместо показа тысячи отдельных объектов отображается один слой, содержащий изображения всех этих объектов. Поверх слоя с изображениями объектов располагается еще один слой — с информацией об этих объектах и их границах. Он позволяет сделать объекты интерактивными, то есть запрограммировать реакцию объекта на действия пользователя.

Преимущество этого метода в том, что с сервера подгружается информация об объектах только того тайла, на котором находится указатель мыши. При этом является возможным использование кэширования данных.

Слои активных областей используются, например, при отображении на картах Яндекса [пробок](#) и [фотографий](#).

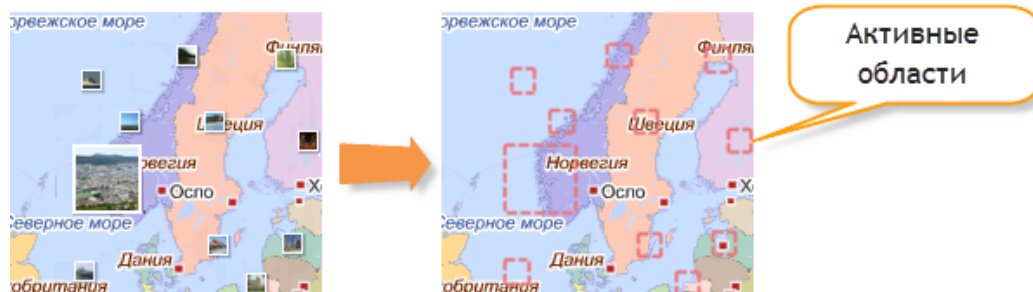
Технология активных областей



Слой, содержащий изображения объектов, называется **картиночным слоем** и формируется, как и любой слой, из тайлов. Для создания этого слоя необходимо сформировать изображения тех тайлов, в которые попадают объекты при отображении на карте. При этом фон тайлов должен быть прозрачным.

Для того, чтобы объекты стали интерактивными, то есть могли взаимодействовать с пользователем, поверх картиночного слоя на карту добавляется **слой активных областей**. Он предоставляет API сведения о том, какие участки карты должны быть интерактивными.

Интерактивный участок карты называется **активной областью** и представляется одной или несколькими простейшими геометрическими фигурами. Активная область является абстрактным понятием, на карте она не видна.



Слой активных областей содержит **информацию об активных областях** — их позицию, типы их геометрий, а также некоторые свойства (например, содержание их балуна или хинта). Эта информация представляется в формате JSON.

Для хранения данных слоя активных областей используются те же самые тайловые технологии, что и для обычных слоев карты. Это означает, что данные раскладываются по квадратам 256x256 пикселей, нарезанным для всех уровней масштабирования. Данные об активных областях хранятся на сервере.

Слой активных областей обменивается данными с сервером с помощью специального объекта — **источника данных**. Источник данных формирует и отправляет запрос за информацией об активных областях нужного тайла и возвращает данные слою.

Слой хранит данные об активных областях только того тайла, на котором находится указатель мыши. Когда пользователь перемещает указатель мыши на другой тайл, слой обращается к серверу за данными для нового тайла. Как только сервер вернет ответ, слой удалит данные предыдущего тайла. Таким образом, слой не хранит загруженные ранее данные.

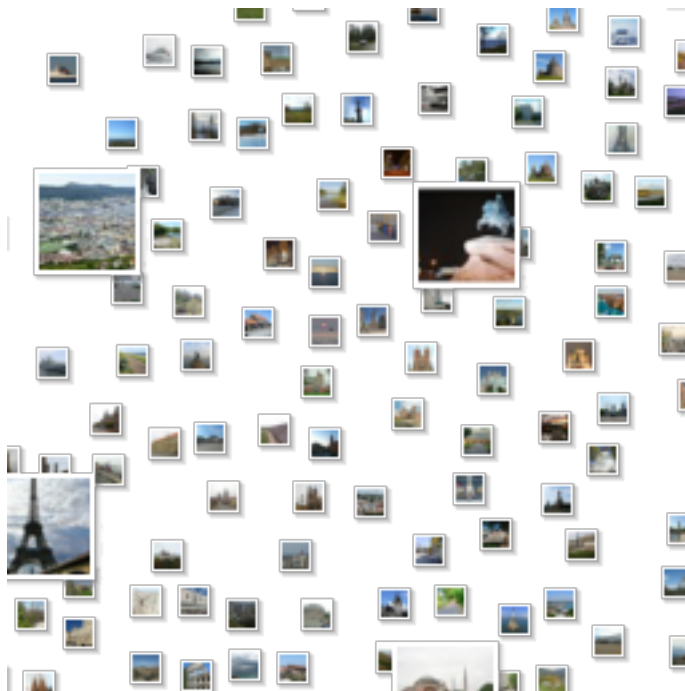
Однако есть возможность использования кэша браузера. Подробнее об этом см. в разделе [Использование кэша браузера](#).

Слой активных областей и картиночный слой существуют независимо друг от друга, и порядок их добавления на карту неважен. В некоторых случаях слой активных областей может использоваться и без картиночного (например, если нужно сделать интерактивным изображение какого-то дома на карте). На карту может быть добавлено несколько слоев активных областей.

Картиночный слой

Картиночный слой содержит изображения объектов, размещаемых на карте.

Как и любой другой слой, картиночный слой состоит из тайлов. Перед созданием этого слоя достаточно сформировать только те тайлы, в которые попадают изображения объектов. Изображения должны быть нарисованы на прозрачном фоне.



Тайлы должны быть сформированы для всех уровней масштабирования.

Для создания картиночного слоя используется класс [Layer](#). Этот класс принимает следующие параметры:

- строковый шаблон, по которому строится URL тайла или функция, генерирующая соответствующий URL (подробнее см. в [справочнике](#));
- [опции](#) слоя.

```
var imgLayer = new ymaps.Layer('http://server.domain/images/%z/%x/%y.png', {
    tileTransparent: true // слой является прозрачным
});
```

Ссылка на слои находится в поле [layers](#) объекта карты. Для добавления слоя на карту используется метод [add\(\)](#):

```
myMap.layers.add(imgLayer);
```

Картиночный слой можно поместить в хранилище слоев карты. Для этого нужно вызывать метод [add\(\)](#) хранилища [layer.storage](#) с параметрами:

- строковый ключ, по которому добавляемый слой будет доступен;
- функция-конструктор, создающая экземпляры соответствующего слоя.

```
var MyLayer = function() {
    return new ymaps.Layer('http://server.domain/images/%z/%x/%y.png', {
        tileTransparent: true
    })
}

// Добавляем слой в хранилище по ключу 'my#layer'
ymaps.layer.storage.add('my#layer', MyLayer);
```

Описания активных областей

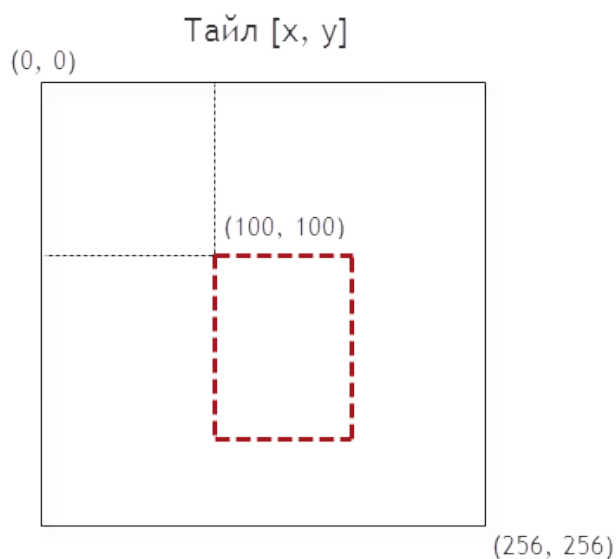
Описания активных областей должны быть сформированы отдельно для каждого тайла и для всех уровней масштабирования. В стандартной реализации API в качестве формата описания данных используется JSON.

В JSON-описании активной области необходимо указать:

- идентификатор активной области;
- её позицию в тайле в пиксельных координатах;
- тип геометрической фигуры, которой представлена активная область.




Идентификатор активной области должен быть сгенерирован самостоятельно. Он должен являться уникальным в пределах одного слоя.

Позиция фигуры задается в пиксельных координатах относительно левого верхнего угла соответствующего тайла.



Типы геометрии активных областей в API строго определены. В таблице ниже представлены допустимые типы:

Пример	Тип геометрии	Описание
	Rectangle	Прямоугольник.
	Polygon	Многоугольник.*

Пример	Тип геометрии	Описание
	ConvexPolygon	Выпуклый многоугольник.
	Multipolygon	Сложная фигура. Состоит из нескольких многоугольников.
	MultiConvexPolygon	Сложная фигура. Состоит только из выпуклых многоугольников.

* Многоугольники могут иметь внутренний контур. Если многоугольник является выпуклым, то внутренний его контур тоже должен быть выпуклым.

Также в описании активной области можно указать ряд ее свойств, например, содержание балуна или хинта, или z-index области относительно других областей.

Ниже приведен пример описания активных областей:

```

"data": {
  "type": "FeatureCollection",
  // Массив активных областей.
  "features": [{
    "type": "Feature",
    // Данные активной области.
    "properties": {
      "hintContent": "Содержимое текстовой подсказки.",
      "balloonContentBody": "Содержимое балуна.",
      "balloonContentHeader": "Заголовок балуна.",
      "balloonContentFooter": "Нижняя часть балуна.",
      // Можно задавать свойство balloonContent вместо Body/Header/Footer

      // Обязательное поле
      "HotspotMetaData": {
        // Идентификатор активной области.
        "id": 10469893,

        // Данные, на основе которых создается геометрия активной
        // области.
        // Обязательное поле.
        "RenderedGeometry": {
          "type": "Polygon",
          // Координаты многоугольника.
          "coordinates": [
            // Первый контур многоугольника.
            [
              [-315, 280], [32, 442], [141, 208], [-206, 46], [-315,
280]
            ],
            // Второй контур многоугольника.
            [
              [-186, 155], [-238, 265], [-152, 306], [-100, 196],
[-186, 155]
            ]
          ]
        }
      }
    }
  ]
}
```

```

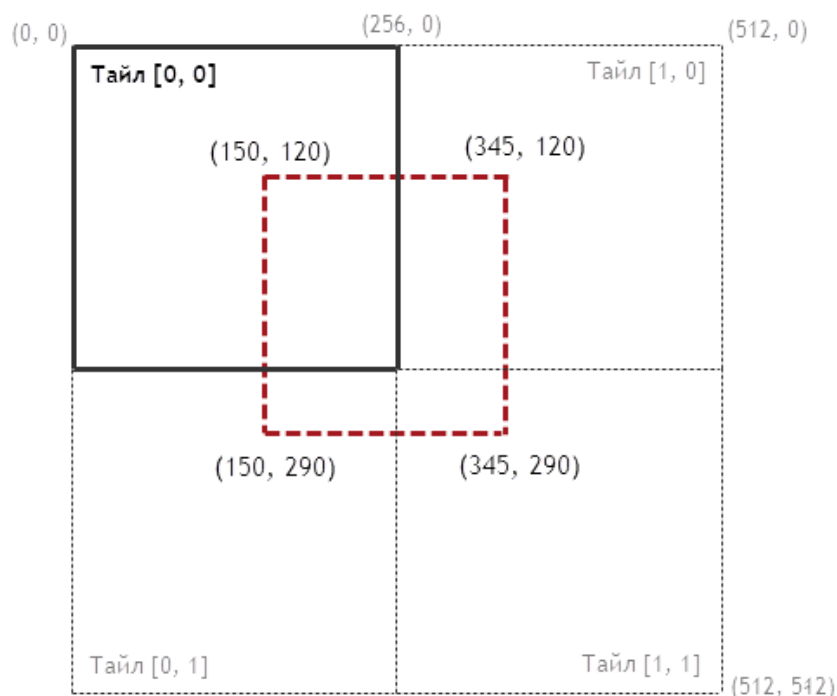
    }
  },
  {
    "type": "Feature",
    "properties": {
      // описание следующего хотспотного объекта...
    }
  }
}

```

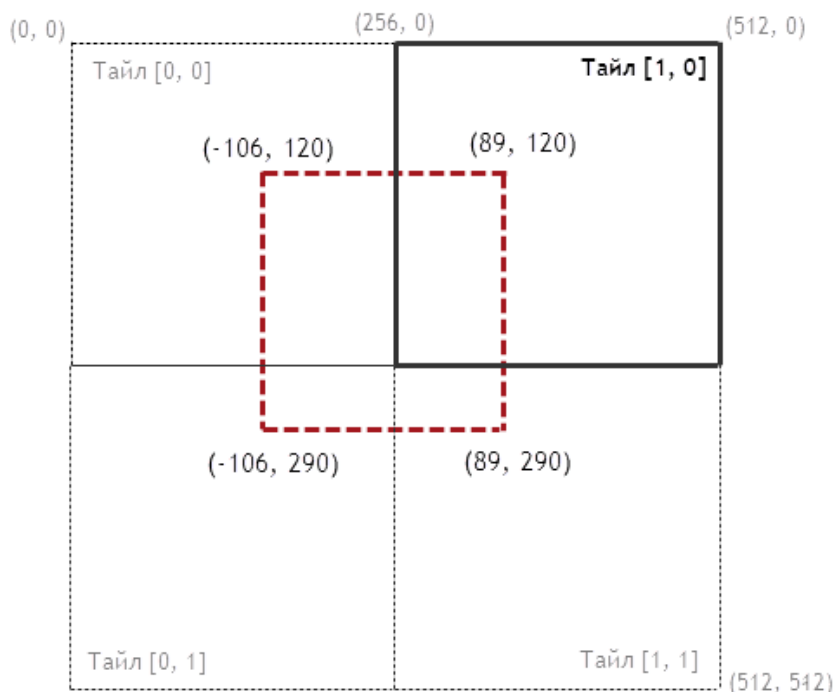
Бывают ситуации, когда активная область попадает сразу в несколько тайлов. При формировании описания такой области не нужно разбивать ее на фигуры по границам тайлов. Описание задается для всей фигуры целиком. Это означает, что в описании активной области для каждого тайла нужно указывать одинаковый идентификатор. При этом координаты вершин фигуры нужно указывать относительно того тайла, для которого формируется описание.

На рисунках ниже приведен пример указания координат вершин прямоугольника, попадающего в тайлы [0,0], [1,0], [0,1] и [1,1] (примеры приведены для первых двух тайлов).

Координаты вершин прямоугольника относительно тайла [0,0]:



Координаты вершин прямоугольника относительно тайла [1,0]:



Хранение данных об активных областях

Описание активных областей для каждого тайла можно хранить в отдельном файле. Эти файлы располагаются на сервере, и API будет обращаться к нему каждый раз, когда необходимо получить данные для нужного тайла.

API предоставляет возможность использования шаблонов для именования файлов данных. Если файлы именовать в соответствии с определенным шаблоном, то для получения URL данных для каждого тайла API достаточно лишь знать, какой шаблон используется.

Например, пусть шаблон URL данных тайла имеет следующий вид:

```
http://server.domain/%z/%x/%y
```

API выполнит следующую подстановку значений: вместо %z — значение коэффициента масштабирования, вместо %x и %y — номер тайла по x и номер тайла по y соответственно. Затем по получившемуся адресу API будет обращаться к серверу за данными для соответствующего тайла.

Например, для тайла [2,3] при z=4 URL данных будет следующим: `http://server.domain/4/2/3`.

Подробнее об использовании шаблонов можно посмотреть в [справочнике](#).

Примечание:

Пользователю API необходимо самостоятельно сформировать описания всех активных областей и позаботиться о том, чтобы сервер возвращал API эти данные, обернутые в json-callback (подробнее см. в разделе [Источник данных](#)).

Источник данных

Перед созданием слоя активных областей необходимо сформировать источник данных. Основная его задача — запрашивать у сервера данные нужного тайла.

API и сервер общаются между собой в формате JSONP. Это означает, что сервер должен оборачивать возвращаемые данные в callback-функцию, название которой источник указывает в параметре запроса.

Шаблон URL запросов, отправляемых источником, выглядит следующим образом:

```
'{URL данных тайла}?callback={jsonp-callback}',
```

где {URL данных тайла} — шаблон URL данных тайла (см. в разделе [Описания активных областей](#)), {jsonp-callback} — имя функции, в которую сервер должен обернуть ответ.

Для создания источника данных предназначен класс [hotspot.ObjectSource](#). В конструкторе этого класса необходимо передать шаблон URL данных тайлов. Также можно указать дополнительные параметры:

- шаблон имени jsonp-callback, в который сервер должен оборачивать возвращаемые данные нужного тайла. По этому шаблону источник будет формировать для каждого тайла название соответствующей функции и в запросе укажет это название в качестве параметра callback;
- [опции источника данных](#);

```
// Зададим шаблон URL, по которому будут доступны данные для тайла на сервере
var hotspotUrl = 'http://server.domain/%z/%x/%y',
    // Зададим шаблон jsonp-callback
    myCallback = 'myCallback_%c', // вместо '%c' API выполнит подстановку
    'x=<номер тайла по x>&y=<номер тайла по y>&z=<коэффициент масштабирования>'
    // Создадим источник тайлов
    objSource = new ymaps.hotspot.ObjectSource(hotspotUrl, myCallback, {
        noCach: true // не использовать кеш браузера
    });
```

Тогда, например, URL запроса для тайла с номером [2,3] и z=4 будет выглядеть следующим образом:

```
http://server.domain/4/2/3?callback=myCallback_x_2_y_3_z_4.
```

Если при создании источника не задан шаблон имени jsonp-callback, то API будет автоматически генерировать новые названия этих функций каждый раз перед отправкой запроса. В этом случае сервер не будет знать заранее имени jsonp-callback. Поэтому на серверной стороне придется формировать ответ динамически, подставляя переданный jsonp-callback.

Если же параметр callback задан, то сервер будет оборачивать данные для одного тайла всегда в один и тот же jsonp-callback. Это может быть полезным по следующей причине.

Так как название jsonp-callback для тайла заранее известно, на сервере не придется писать скрипт, определяющий это название. Описание активных областей тайла можно сформировать сразу обернутыми в заданную функцию. Например, если шаблон названия jsonp-callback задан как 'myCallback_%c', то для тайла с номером [1, 2] и z=5 на сервере можно создать файл данных:

```
myCallback_x_1_y_2_z_5({
    "data": {
        "type": "FeatureCollection",
        ...
    }
});
```

На основе полученных данных источник создает массив объектов (экземпляров класса [hotspot.Shape](#)), описывающих активные области. Эти объекты представляются в специальном формате, который является удобным для вычисления некоторых геометрических задач (например, определение попадания указателя мыши в активную область).

Источник возвращает слою массив преобразованных объектов. Если тайл не содержит активных областей, то источник возвращает пустой массив.

Стандартная реализация hotspot.ObjectSource предполагает, что источник будет запрашивать данные для любого тайла, на который переместился указатель (даже если активных областей там нет). Однако, бывают ситуации, когда можно заранее определить, в каких тайлах находятся активные области. Тогда источник можно перепрограммировать так, что он будет запрашивать данные только для этих тайлов.

О том, как можно переопределить стандартную реализацию hotspot.ObjectSource, можно посмотреть в [справочнике](#).

Использование кэша браузера

Если URL каждого тайла будет оставаться неизменным, то браузер может запоминать в кэше загруженные данные для этих URL.

Если заранее известно, что данные на сервере меняться не будут, то можно жестко кэшировать данные браузером. При таком кэшировании все обращения к ранее загруженным данным будут напрямую обслуживаться из кэша браузера, без обращения к серверу. Для этого в выдаче ответа сервера необходимо указать заголовки Expires и Cache-Control.

Если данные тайлов обновляются часто (например, в пробках данные обновляются раз в несколько минут), то можно использовать обычное кэширование браузера. Для этого в ответе сервера необходимо указывать заголовок Etag.

Если же кэш браузера использовать не нужно, то следует задать [опцию](#) источника noCache со значением true (либо не задавать шаблон имени собственной callback-функции).

Создание слоя активных областей

Для создания слоя активных областей используется класс [hotspot.Layer](#). В его конструкторе следует передать определенный ранее [источник данных](#) и, если необходимо, дополнительные [опции](#):

```
var hotspotLayer = new ymaps.hotspot.Layer(objSource, {
    hasBalloon: false // у слоя не будет создано поле balloon
});
```

После этого слой нужно добавить на карту:

```
myMap.layers.add(hotspotLayer);
```

Ниже приведен полный текст [примера](#) (с добавлением на карту картиночного слоя):

```
ymaps.ready(init);
function init() {

    var myMap = new ymaps.Map('map', {
        center: [55.709243, 37.500737],
        zoom: 9
    }, {
        // В нашем примере активные области есть только для 9 и 10 масштаба.
        // Поэтому ограничим диапазон коэффициентов масштабирования карты.
        minZoom: 9,
        maxZoom: 10
    });

    // Добавим на карту элемент управления коэффициентом масштабирования.
    myMap.controls.add('smallZoomControl', { top: 5 });

    // Шаблон URL для данных активных областей.
    // Источник данных будет запрашивать данные через URL вида:
    // '.../hotspot_layer/hotspot_data/9/tile_x=1&y=2', где
    // x, y - это номер тайла, для которого запрашиваются данные,
    // 9 - значение коэффициента масштабирования карты.
    var tileUrlTemplate = 'examples/maps/ru/hotspot_layer/hotspot_data/%z/tile_x=%x&y=%y',

    // Шаблон callback-функции, в которую сервер будет оборачивать данные тайла.
    // Пример callback-функции после подстановки -
    'testCallback_tile_x_1_y_2_z_9'.
    keyTemplate = 'testCallback_tile_%c',

    // URL тайлов картиночного слоя.
    // Пример URL после подстановки -
    // '.../hotspot_layer/images/9/tile_x=1&y=2.png'.
    imgUrlTemplate = 'examples/maps/ru/hotspot_layer/images/%z/tile_x=%x&y=%y'
};
```

```
%y.png',

    // Создадим источник данных слоя активных областей.
    objSource = new ymaps.hotspot.ObjectSource(tileUrlTemplate,
keyTemplate),

    // Создаем картиночный слой и слой активных областей.
    imgLayer = new ymaps.Layer(imgUrlTemplate, {tileTransparent: true}),
    hotspotLayer = new ymaps.hotspot.Layer(objSource, {cursor: 'help'});

    // Добавляем слои на карту.
    myMap.layers.add(hotspotLayer);
    myMap.layers.add(imgLayer);
}
```


Элементы управления

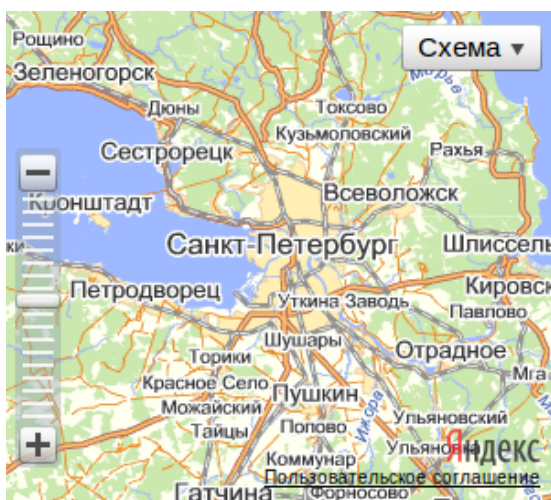
Элемент управления — визуальный объект, связанный с картой и предназначенный для взаимодействия с пользователем. В большинстве случаев элементы управления размещаются в области показа карты.

API карт включает в себя набор встроенных элементов управления (выбор типа карты, ползунок изменения масштаба и пр.) и предоставляет возможность определения собственных.

Компоненты, предназначенные для создания элементов управления, реализованы в виде классов, находящихся в пространстве имен `control` (например, `control.Button`).

Элементы управления программно связаны с объектом карты и хранятся в виде коллекции, ссылка на которую содержится в поле `controls`.

Существует [специальное хранилище](#), позволяющее поставить конструктору класса элемента управления в соответствие строковый ключ. Для добавления и удаления элементов управления карты можно указывать как ссылки на экземпляр элемента управления, так и ключ из этого хранилища.



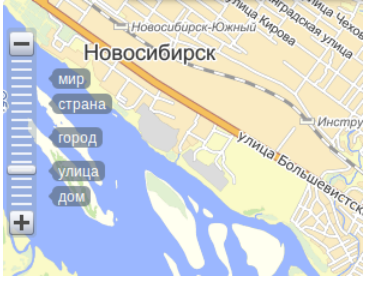
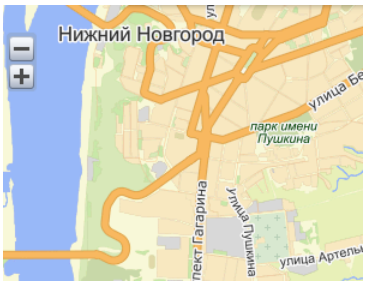

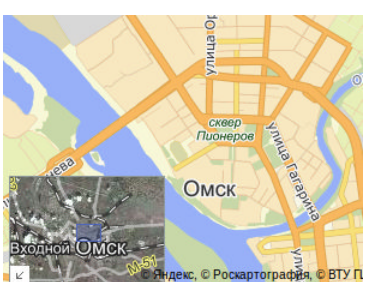
```
var myMap = new ymaps.Map("map", {
  center: [59.93772, 30.313622],
  zoom: 8
});

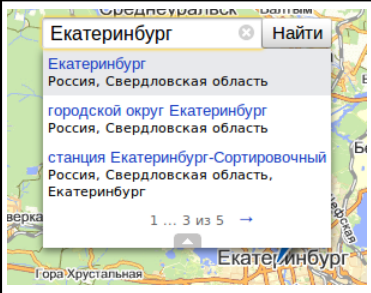
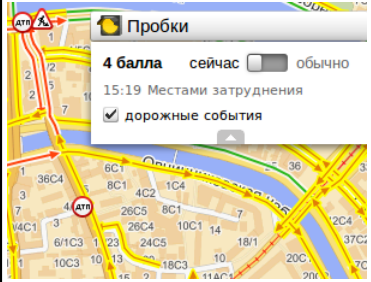
// Создание экземпляра элемента управления
myMap.controls.add(
  new ymaps.control.ZoomControl()
);

// Обращение к конструктору класса элемента
// управления по ключу
myMap.controls.add('typeSelector');
```

Стандартные элементы управления

Элемент управления	Описание	Пример
Стандартный набор кнопок 	Панель инструментов со стандартным набором кнопок: перетягивание карты (drag), увеличение выделенной области, измерение расстояний.	Добавление на карту панели инструментов со стандартным набором кнопок: <pre>myMap.controls.add('mapTools');</pre> или <pre>myMap.controls.add(new ymaps.control.MapTools());</pre>
Выбор типа карты 	Переключатель отображаемого типа карты .	Добавление на карту переключателя типа карты (в выпадающем списке будут отображены все допустимые типы): <pre>myMap.controls.add('typeSelector');</pre> Добавление на карту переключателя типа карты (в выпадающем списке будут отображены только указанные типы):

Элемент управления	Описание	Пример
		<pre>myMap.controls.add(new ymaps.control.TypeSelector(['yandex#map', 'yandex#publicMap']));</pre>
<p>Ползунок масштаба</p> 	<p>Изменение коэффициента масштабирования.</p> <p>Две кнопки для увеличения/уменьшения коэффициента масштабирования на единицу и ползунок масштаба.</p>	<p>Добавление на карту панели изменения масштаба:</p> <pre>myMap.controls.add('zoomControl');</pre> <p>или</p> <pre>myMap.controls.add(new ymaps.control.ZoomControl());</pre>
<p>Кнопки изменения масштаба</p> 	<p>Изменение коэффициента масштабирования — компактный вариант.</p> <p>Две кнопки, одна из которых увеличивает коэффициент масштабирования на единицу, вторая — уменьшает на единицу.</p>	<p>Добавление на карту кнопок изменения масштаба:</p> <pre>myMap.controls.add('smallZoomControl');</pre> <p>или</p> <pre>myMap.controls.add(new ymaps.control.SmallZoomControl());</pre>
<p>Масштабная линейка</p> 	<p>Отображает на карте масштабный отрезок и показывает его длину в используемых географических единицах измерениях.</p> <p>Позволяет визуально оценить расстояние между объектами.</p>	<p>Добавление на карту масштабной линейки:</p> <pre>myMap.controls.add('scaleLine');</pre> <p>или</p> <pre>myMap.controls.add(new ymaps.control.ScaleLine());</pre>
<p>Обзорная карта</p> 	<p>Уменьшенная карта местности, коэффициент масштаб которой на заданное количество единиц меньше основного.</p> <p>Тип обзорной карты может отличаться от типа основной карты.</p>	<p>Добавление обзорной карты:</p> <pre>myMap.controls.add('miniMap');</pre> <p>Добавление обзорной карты с заданными параметрами:</p> <pre>myMap.controls.add(new ymaps.control.MiniMap({type: 'yandex#hybrid', {zoomOffset: 3}));</pre>
<p>Поиск по карте</p>	<p>Поиск географических объектов по названию, адресу или географическим координатам.</p>	<p>Добавление на карту поисковой строки:</p> <pre>myMap.controls.add('searchControl');</pre> <p>Добавление на карту поисковой строки с заданными параметрами:</p>

Элемент управления	Описание	Пример
		<pre>myMap.controls.add(new ymaps.control.SearchControl({provider: 'yandex#publicMap', useMapBounds: true}));</pre>
	<p>Отображение пробок на карте.</p> <p>Позволяет включать/отключать отображение на карте слоя пробок.</p> <p>Может быть показана как текущая (пробки «сейчас»), так и усредненная по времени загруженность дорог (пробки «обычно»).</p>	<p>Добавление на карту элемента управления «Пробки»:</p> <pre>myMap.controls.add('trafficControl');</pre> <p>Добавление на карту элемента управления «Пробки» с указанным источником пробок (будут показаны пробки «обычно»):</p> <pre>myMap.controls.add(new ymaps.control.TrafficControl({providerKey: 'traffic#archive'}));</pre>

Пользовательские элементы управления

API предоставляет возможность создания собственных элементов управления, позволяющих выполнить произвольные действия.

Наиболее часто элементы управления реализуются в виде кнопок. Для создания кнопочного элемента управления предназначен класс [control.Button](#).

Поддерживается два типа реакций на нажатие кнопки: после нажатия кнопка остается нажатой (поведение по умолчанию) или возвращается в исходное состояние. Тип реакции на нажатие задается опцией `selectOnClick` (`true/false`).

При нажатии на кнопку всегда генерируется событие [click](#). Для кнопок, запоминающих свое состояние после нажатия, дополнительно генерируется событие [select](#) или [deselect](#) (в зависимости от того, в каком состоянии кнопка находилась в момент нажатия).



```
var myButton = new
ymaps.control.Button('<b>Я<b>');
myButton.events
.add('click', function ()
{ alert('Щёлк'); })
.add('select', function () { alert('На-
жата'); })
.add('deselect', function ()
{ alert('Отжата'); });
myMap.controls.add(myButton);
```

```
var myButton = new ymaps.control.Button({
data: {
content: 'Или <em>Я</em>',
title: 'отжимаюсь сама'
}, {
selectOnClick: false
}
});
myButton.events
.add('click', function () {
alert('Щёлк'); });
myMap.controls.add(myButton);
```

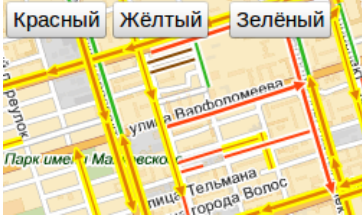


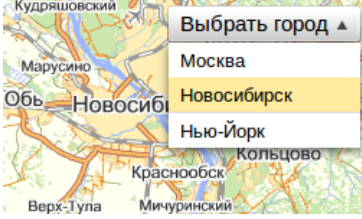
Класс [control.Button](#) реализует элемент управления, для которого установлен стандартный внешний вид, реакция на нажатие и методы, позволяющие задать и определить текущее состояние. Кнопка может быть включена в любой составной элемент управления.

Для определения собственных элементов управления предназначен интерфейс [IControl](#). Для элементов управления, которые могут находиться в только в двух состояниях удобно использовать интерфейс [ISelectableControl](#), расширяющий [IControl](#).

Составные элементы управления и группировка

API Яндекс.Карт включает в себя набор компонентов, позволяющих создавать составные элементы управления, визуально объединяя включенные в них элементы.

Список классов для создания составных элементов приведен в таблице.

Элемент управления	Описание	Пример
<p>Горизонтальная панель</p> 	<p>Панель инструментов, элементы которых расположены горизонтально.</p> <p>Чаще всего на горизонтальной панели размещаются кнопки.</p> <p>Классы:</p> <p><code>control.ToolBar</code> — контейнер элементов;</p> <p><code>control.ToolBarSeparator</code> — разделитель элементов панели.</p>	<pre>var myToolBar = new ymaps.control.ToolBar(); myToolBar.add(new ymaps.control.Button('Красный')) .add(new ymaps.control.Button('Жёлтый')) .add(new ymaps.control.ToolBarSeparator()) .add(new ymaps.control.Button('Зелёный')); myMap.controls.add(myToolBar);</pre>
<p>Панель переключаемых элементов</p> 	<p>Панель элементов, каждый из которых может находиться только в двух состояниях (<code>ISelectableControl</code>).</p> <p>Класс <code>control.RadioGroup</code>.</p>	<pre>var myRadioGroup = new ymaps.control.RadioGroup({ items: [new ymaps.control.Button('Наше'), new ymaps.control.Button('Маяк'), new ymaps.control.Button('Европа +')] }); myMap.controls.add(myRadioGroup);</pre>
<p>Раскрывающаяся панель</p> 	<p>Панель элементов, вертикально раскрывающаяся при наведении курсора на первый элемент.</p> <p>Класс <code>control.RollupButton</code>.</p>	<pre>var myRollupButton = new ymaps.control.RollupButton({ items: [new ymaps.control.Button('Раз'), new ymaps.control.Button('Два'), new ymaps.control.Button('Три')] }); myMap.controls.add(myRollupButton);</pre>
<p>Выпадающий список</p> 	<p>Выпадающий список.</p> <p>Классы:</p> <p><code>control.ListBox</code> — контейнер элементов;</p> <p><code>control.ListBoxItem</code> — элемент списка;</p> <p><code>control.ListBoxSeparator</code> — разделитель элементов списка.</p>	<pre>var myListBox = new ymaps.control.ListBox({ data: { title: 'Выбрать город' }, items: [new ymaps.control.ListBoxItem('Москва'), new ymaps.control.ListBoxItem('Новосибирск'), new</pre>

Элемент управления	Описание	Пример
		<pre> ymaps.control.ListBoxSeparator(), new ymaps.control.ListBoxItem('Н ью-Йорк'),] }); myMap.controls.add(myListBox) </pre>

В общем случае для группировки элементов управления предназначен класс [control.Group](#) (все приведенные в таблице составные элементы его наследуют). С его помощью можно задавать общие для всех элементов группы свойства и проводить групповые операции над ними.

```

var myControlGroup = new ymaps.control.Group({
    items: [
        new ymaps.control.Button('Первый'),
        new ymaps.control.Button('Второй'),
        new ymaps.control.Button('Расчет окончен')
    ]
}, {selectOnClick: false});

myControlGroup.events.add(['click'], function (e) {
    alert(e.get('target').data.get('content'));
});

```

Макеты и шаблоны

Внешний вид любых отображаемых элементов (балун, хинт, метка и пр.) определяется с помощью *макетов*. Макет представляет из себя объект, реализующий интерфейс [ILayout](#), который генерирует код, отвечающий за конечное представление объекта. Макеты могут изменять DOM-дерево документа.

Чтобы изменить/задать внешний вид отображаемого объекта необходимо изменить/задать связанный с ним макет. Объекты связываются со своими макетами с помощью соответствующих опций.

```

var myPlacemark = new ymaps.Placemark(
    [55.8, 37.6],
    {},
    {balloonContentLayout: MyBalloonContentLayoutClass}
    // MyBalloonContentLayoutClass — класс макета балуна.
);

```

В отличие от опций, которые позволяют изменить только некоторые параметры отображения элемента, макет позволяет сформировать какое угодно его представление.

Реализация собственного макета «с нуля» на основе интерфейса [ILayout](#) является относительно трудоемкой задачей, решение которой выходит за рамки данного руководства. Однако API предоставляет удобный механизм, позволяющий генерировать макеты на основе *шаблонов*.

Шаблон представляет собой HTML-код, в который могут быть включены подстановки. Для создания макетов с помощью языка шаблонов предназначена фабрика макетов — статический объект [templateLayoutFactory](#). Этот объект имеет единственный метод [createClass](#), в который передается код шаблона.

```

var myBalloonContentLayoutClass = ymaps.templateLayoutFactory.createClass(
    '<h3>Макет</h3><p>Создан на основе шаблона.</p>'
);

```

Часто возникает необходимость использовать данные объекта в связанном с ним макете. Например, вывести название метки в ее балуне.

Шаблоны имеют доступ к некоторым полям родительского объекта. В большинстве случаев доступны значения полей `data` (либо `properties`), `options` и `state`. Значения этих полей могут быть ис-

пользованы в т. н. подстановках. Типы возможных подстановок описаны в спецификации объекта [templateLayoutFactory](#).

```
var myPlacemark = new ymaps.Placemark(
  [55.8, 37.6],
  {
    name: 'Москва',
    description: 'Столица. Много людей.',
    metro: true
  },
  {balloonContentLayout: myBalloonContentLayoutClass}
);
var myBalloonContentLayoutClass = ymaps.templateLayoutFactory.createClass(
  '<h3>${properties.name}</h3>' +
  '<p>Описание: ${properties.description}</p>' +
  '<p>Население: ${properties.population|неизвестно}</p>' +
  '<p>Метрополитен: [if properties.metro]да[else]нет[endif]</p>'
);
```

Макеты можно размещать в специальном хранилище [layout.storage](#) и в дальнейшем извлекать их оттуда по ключу.

```
var myBalloonContentLayoutClass = ymaps.templateLayoutFactory.createClass(...);
ymaps.layout.storage.add('my#simplestBCLayout', myBalloonContentLayoutClass);
var myPlacemark = new ymaps.Placemark(
  ...
  {balloonContentLayout: 'my#simplestBCLayout'}
);
```

Шаблоны могут включать в себя определенные ранее макеты. В приведенных примерах переопределяется макет содержимого балуна, который изначально включает в себя три макета: для заголовка, тела и нижнего колонтитула.

Для включения в шаблон макета используются подстановки вида `[[название_класса_макета]]` и `[[ключ_макета_в_хранилище]]`.

```
var myBalloonContentLayoutClass = ymaps.templateLayoutFactory.createClass(
  '${[options.balloonContentLayout]}<p>...</p>${[my#simplestBCLayout]}'
);
```

События

В API Яндекс.Карт реализован собственный механизм работы с событиями, расширяющий стандартные средства управления событиями JavaScript. Это означает, что для работы с событиями браузера, окна браузера и DOM-дерева применимы стандартные подходы, используемые в JavaScript, для работы с объектами API — событийная модель API Яндекс.Карт.

В API реализована традиционная модель работы с событиями. Некоторые объекты могут генерировать события, на которые можно подписываться и передавать в функцию-обработчик.

Все объекты API, которые могут генерировать события, реализуют интерфейс [IEventEmitter](#) и имеют поле `events`. Это поле содержит ссылку на менеджер событий, с помощью которого можно подписаться на нужные события или удалить подписку.

```
myPlacemark.events.add('click', function () {
  alert('О, событие!');
});
```

Возникшее [событие](#) распространяется вверх по всей [иерархии](#) родительских объектов вплоть до родительской коллекции объекта карты (т. н. «всплывание»). До самого объекта карты события дочерних элементов не доходят.

```
myPlacemark.events.add('click', function () {
    alert('О, событие!');
});
myMap.geoObjects.events.add('click', function (e) {
    alert('Дošло до коллекции объектов карты');
    // Получение ссылки на дочерний объект, на котором произошло событие
    var object = e.get('target');
});
myMap.events.add('click', function () {
    alert('Событие на карте'); // Возникнет при щелчке на карте, но не на
    маркере.
});
```

Для остановки распространения события предназначен метод [stopPropagation](#).

```
myPlacemark.events.add('click', function (e) {
    alert('О, событие!');
    e.stopPropagation();
});
myMap.geoObjects.events.add('click', function () {
    alert('Граница на замке');
});
```

Для большинства событий определены стандартные действия, которые будут произведены после возникновения события. Эти действия могут быть отменены с помощью метода [preventDefault](#).

```
myMap.events.add('dblclick', function (e) {
    e.preventDefault(); // При двойном щелчке зума не будет.
});
```

Структура объекта-события

События, генерируемые API Яндекс.Карт представляют собой объекты класса [Event](#). Исключение составляют события карты, описываемые классом [MapEvent](#), расширяющим [Event](#).

Объект, описывающий событие, содержит в себе информацию как о самом событии, так и об объекте, который его сгенерировал. Извлечение данных из объектов [Event](#) и [MapEvent](#) производится с помощью метода [get](#), в который передается имя поля внутреннего объекта этого класса, описывающего событие.

Для любого события может быть получена ссылка на объект, который его сгенерировал. Эта ссылка содержится в поле `target`.

```
myMap.events.add('click', function (e) {
    var eMap = e.get('target'); // Получение ссылки на объект, сгенерировавший
    событие (карта).
    eMap.setType('yandex#hybrid');
});
```

Тип любого события может быть получен путем обращения к полю `type`.

```
myMap.events.add(['click', 'contextmenu'], function (e) {
    var eType = e.get('type');
    eType == 'click' ? alert('left button') : alert('right button');
});
```

Событие API, возникновение которого инициируется событием DOM, в поле `domEvent` внутреннего объекта содержит ссылку на объект API, описывающий DOM-событие (класс [DomEvent](#)). Этот объект содержит ссылку на оригинальное событие DOM в поле `originalEvent`.

```
myPlacemark.events.add('mousedown', function (e) {
    // 0, 1 или 2 в зависимости от того, какая кнопка мыши нажата (В IE значение
    может быть от 0 до 7).
    alert(e.get('domEvent').originalEvent.button);
});
```

Класс [DomEvent](#) предоставляет прокси-доступ к полям и методам оригинального DOM-события и адаптирован с учетом особенностей различных браузеров. Поэтому удобнее не обращаться

к оригинальному DOM-событию, а использовать класс `DomEvent`: метод `get` для получения свойства события и метод `callMethod` для обращения к его методу.

```
myPlacemark.events.add('click', function (e) {
    // Вместо alert(e.get('domEvent').originalEvent.pageX);
    alert(e.get('domEvent').get('pageX'));
});
```

Кроме того, `DomEvent` позволяет получить координаты курсора сразу в виде массива — `get('position')`.

Архитектура API спроектирована таким образом, что большую часть информации, содержащейся в объекте, описывающем событие, можно узнать из свойств объекта, который сгенерировал это событие.

Контекст обработки события

При назначении обработчика события с помощью менеджера событий, можно указать не только функцию-обработчик, но и контекст, в котором она будет выполняться. Чаще всего в качестве контекста передается ссылка на объект, внутренние данные которого необходимы функции-обработчику.

```
function myCounter(start) {
    this.counter = start;
}
myCounter.prototype.go = function () {
    var myMap = new ymaps.Map("map", {
        center: [59.93, 30.31],
        zoom: 12
    });
    myMap.events.add('click', this.up, this);
}
myCounter.prototype.up = function () {
    this.counter++;
    alert(this.counter)
};
var c = new myCounter(2012);
c.go();
```

Поиск по карте

API предоставляет сервис *геокодирования*, который позволяет определять географические координаты объекта по его названию (прямое геокодирование) и, наоборот, название объекта по его координатам (обратное геокодирование).

Оба вида геокодирования (прямое и обратное) производятся с помощью функции `geocode`, в которую можно передать название объекта в виде строки или координаты в виде массива.

```
var myGeocoder = ymaps.geocode("Петрозаводск");
var myReverseGeocoder = ymaps.geocode([61.79, 34.36]);
```

Функция `geocode` является асинхронной, то есть во время ее выполнения производится обмен данными с сервером.

Асинхронное взаимодействие реализовано с помощью обещаний (`promises`). Вызов функции `geocode` приводит к (немедленному) созданию объекта типа `util.Promise`, который выполнит заданные функции в тот момент, когда от сервера вернутся или результаты геокодирования, или сообщение об ошибке.

Результат функции `geocode` передается в функцию-обработчик в виде коллекции `GeoObjectCollection`. Этот объект реализует интерфейс `IGeoObject`, то есть может быть размещен на карте.


```
var myGeocoder = ymaps.geocode("Петрозаводск");
myGeocoder.then(
  function (res) {
    alert('Координаты объекта :' +
res.geoObjects.get(0).geometry.getCoordinates());
  },
  function (err) {
    alert('Ошибка');
  }
);
```

Результаты геокодирования передаются в функцию-обработчик либо в виде коллекции [GeoObjectArray](#) (по умолчанию) либо в виде JSON. Формат возвращаемых данных задается опцией `json` (`true/false`). Подробнее о формате возвращаемых данных можно посмотреть [Поиск по карте](#).

Поиск можно производить как по всей карте мира, так и по заданной прямоугольной области. При этом прямоугольная область может накладывать как жесткие ограничения (поиск будет производиться только среди объектов внутри области), так и мягкие. В последнем случае поиск будет производиться по всей карте, но чем ближе будет находиться найденный объект к центру области, тем более высоко он будет ранжироваться.

При обратном геокодировании можно уточнить тип объектов, которые необходимо найти (дом, улица, район города, населенный пункт, станция метро).

Найденные объекты ранжируются по степени удаленности от заданной точки.

```
var myCoords = [55.754952, 37.615319];
myMap.geoObjects.add(
  new ymaps.Placemark(myCoords,
    {iconContent: 'Где метро?'},
    {preset: 'twirl#greenStretchyIcon'}
  )
);
var myGeocoder = ymaps.geocode(myCoords, {kind: 'metro'});
myGeocoder.then(
  function (res) {
    var nearest = res.geoObjects.get(0);
    var name = nearest.properties.get('name');
    nearest.properties.set('iconContent', name);
    nearest.options.set('preset', 'twirl#redStretchyIcon');
    myMap.geoObjects.add(res.geoObjects);
  },
  function (err) {
    alert('Ошибка');
  }
);
```



При поиске улиц, районов и населенных пунктов не учитываются отнесенные к ним здания, каких бы размеров они ни были.

```
var myGeocoder = ymaps.geocode('Новый Арбат, 10');
myGeocoder.then(
  function (res) {
    var coords = res.geoObjects.get(0).geometry.getCoordinates();
    var myGeocoder = ymaps.geocode(coords, {kind: 'street'});
    myGeocoder.then(
      function (res) {
        var street = res.geoObjects.get(0);
        var name = street.properties.get('name');
        // Будет выведено «улица Большая Молчановка»,
        // несмотря на то, что обратно геокодируются
        // координаты дома 10 на ул. Новый Арбат.
        alert(name);
      }
    );
  }
);
```

Определение местоположения пользователя

В API включена возможность определения местоположения пользователя. Одним из основных параметров, используемых для оценки местоположения, является IP-адрес устройства, обращающегося к API. [Geolocation API](#) устройства при этом не используется.

Во время загрузки API производится запрос к серверу, и в момент [готовности API](#) данные о предполагаемом местоположении пользователя доступны для использования. Доступ к этим данным предоставляет статический объект [geolocation](#).

```
myMap.geoObjects.add(
    new ymaps.Placemark(
        [ymaps.geolocation.latitude, ymaps.geolocation.longitude],
        {
            balloonContentHeader: ymaps.geolocation.country,
            balloonContent: ymaps.geolocation.city,
            balloonContentFooter: ymaps.geolocation.region
        }
    )
);
```

Geo XML

API Яндекс.Карт позволяет загружать XML-файлы с географическими данными в форматах [YMapsML](#), [KML](#) и [GPX](#).

Для загрузки данных используется функция [geoXml.load](#), в которую передаётся URL XML-файла. Загрузка производится асинхронно (аналогично [геокодированию](#)), а формат загружаемого файла определяется функцией `geoXml.load` автоматически. Данные о геообъектах преобразуются в коллекцию [GeoObjectCollection](#) и размещаются в поле `geoObjects` объекта, передаваемого в функцию-обработчик. Этот объект может быть размещен на карте.

```
ymaps.geoXml.load('http://openflights.org/demo/openflights-sample.kml').then(function (res) {
    myMap.geoObjects.add(res.geoObjects);
});
```

Загрузка XML-данных из источника с указанным URL производится сервером Яндекса. Поэтому XML-данные следует размещать на ресурсе, к которому предоставлен публичный доступ (или позаботиться о том, чтобы серверный загрузчик Яндекса смог получить к доступ к нему).

Формат YMapsML позволяет описать не только координаты, геометрические свойства геообъектов и их отображение, но и включить в себя описание параметров карты. Параметры карты (если они заданы) помещаются в поле `mapState` в виде объекта, позволяющего применить эти параметры к любой карте.

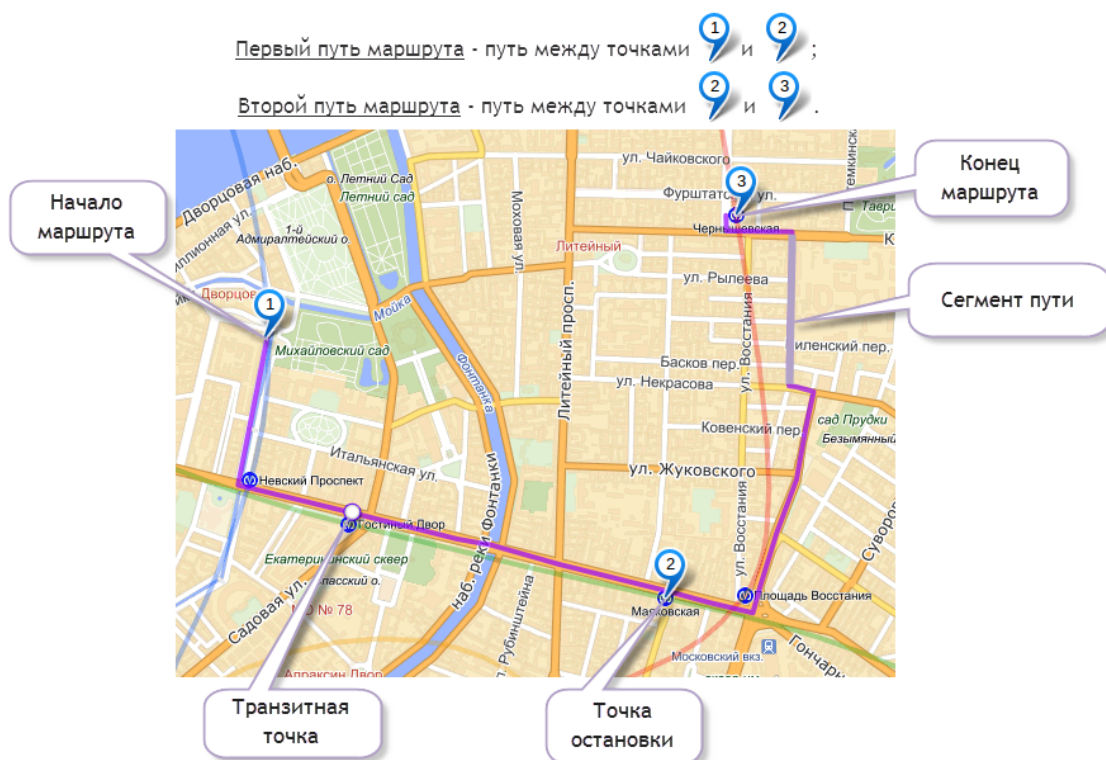
```
ymaps.geoXml.load('http://maps.yandex.ru/export/usermaps/93jfWjoXws37exPmKH-OFIu3IQduHal/').then(function (res) {
    myMap.geoObjects.add(res.geoObjects);
    if (res.mapState) {
        res.mapState.applyToMap(myMap);
    }
});
```

Для коллекции, в которую при загрузке преобразуется GPX-трек, определено два [пресета](#) с ключами `gpx#interactive` и `gpx#plain`. Первый изменяет свойства ломаных, которыми соединены GPX-точки, таким образом, что при клике на любой её точке показывается дополнительная информация (время, скорость и пр.). Этот пресет используется по умолчанию. Во втором случае вывод дополнительной информации не производится.

```
ymaps.geoXml.load('http://karmatsky.narod2.ru/MskChel2.xml').then(function (res) {
    res.geoObjects.options.set({ 'preset': 'gpx#plain' });
    myMap.geoObjects.add(res.geoObjects);
});
```

Маршрутизатор

API предоставляет возможность прокладывания автомобильных маршрутов. Маршрут между начальным и конечным пунктом вычисляется автоматически, при этом можно задать произвольное количество *точек остановки* и *транзитных точек* маршрута.



Для построения маршрута предназначена функция [route](#), в которую передается массив точек, через которые нужно проложить маршрут и, при необходимости, дополнительные опции его построения.

Вычисление маршрута производится асинхронно (аналогично [геокодированию](#)). Результат передается в функцию-обработчик в виде коллекции [GeoObjectCollection](#). Этот объект реализует интерфейс [IGeoObject](#), то есть может быть размещен на карте.

```
ymaps.route(['Москва', 'Санкт-Петербург']).then(
  function (route) {
    myMap.geoObjects.add(route);
  },
  function (error) {
    alert('Возникла ошибка: ' + error.message);
  }
);
```

Существует три способа задания точки маршрута:

1. Пара координат в виде массива ([59.94, 30.31]).
2. Строка ('Санкт-Петербург'). Производится автоматическое [геокодирование](#).
3. Объект с полями type и point ({type: 'wayPoint', point: 'Санкт-Петербург'}, {type: 'viaPoint', point: [59.94, 30.31]}).

Этот способ позволяет в явном виде указать тип точки маршрута: wayPoint — точка остановки, viaPoint — транзитная точка. Если в поле point передана строка, производится автоматическое геокодирование.

Если тип точки не указан, она считается точкой остановки.

```

ymaps.route([
  'Кронштадт, Якорная площадь',
  {
    type: 'viaPoint',
    point: [59.93328, 30.342791] // или 'Аничков мост'
  },
  'Санкт-Петербург, Финляндский вокзал' // или [59.956084, 30.356849]
]).then(
  function (route) {
    myMap.geoObjects.add(route);
  },
  function (error) {
    alert("Возникла ошибка: " + error.message);
  }
);

```

Прокладывать маршрут можно как с учетом так и без учета пробок. Игнорирование или учет пробок регулируется опцией `avoidTrafficJams` (`true/false`). По умолчанию пробки не учитываются. Маршрут, проложенный без учета пробок, может отличаться от маршрута, проложенного между теми же точками, но с учетом пробок.

```

ymaps.route(['Кронштадт, Якорная площадь', 'Санкт-Петербург, Финляндский вокзал', {avoidTrafficJams: true}]);

```

Построенный маршрут представляется в виде [упорядоченной коллекции](#) путей ([router.Path](#)), попарно последовательно соединяющих точки остановки. Получить пути маршрута можно с помощью метода [getPaths](#). Для предыдущего примера с промежуточной точкой «Аничков мост» код `route.getPaths().getLength()` возвратит единицу, так как промежуточные точки не влияют на разбиение маршрута на пути.

Каждый путь разбивается на сегменты ([router.Segment](#)), получить которые из пути можно с помощью метода [getSegments](#). Концами сегментов являются начальные и конечные точки путей, транзитные точки, точки вынужденной остановки (например, паром) и точки возможного изменения направления движения (развилка, въезд, съезд, поворот, разворот, перекресток).

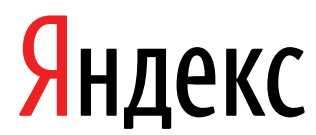
Сегмент описывается ломаной, координаты точек ломаной возвращаются методом [getCoordinates](#).

```

var routeLength = route.getLength(); // Длина маршрута
var firstPath = route.getPaths().get(0); // Первый путь
var firstPathLength = firstPath.getLength(); // Длина первого пути
var firstPathTime = firstPath.getTime(); // Время без учета пробок
var firstPathFirstSegment = firstPath.getSegments()[0]; // Первый сегмент первого пути
var firstPathFirstSegmentJamsTime = firstPathFirstSegment.getJamsTime(); //
Время с пробками

```

Для сегментов также доступны данные, позволяющие построить маршрутный лист: направление и угол поворота в конце сегмента, а также название улицы, на которой он начинается.



API Яндекс.Карт
Руководство разработчика

18.12.2012