# Руководство пользователя

# Pilot-ComponentKit

2024

Pilot-ComponentKit ..... 7
Системные требования ..... 9
ПЕРВЫЕ ШАГИ ..... 10
Добавить компоненты на HTML страницу ..... 11
Инициализация компонентов ..... 13
Загрузка частей модели ..... 14
Загрузка документа ..... 15
Создание расширения ..... 16
API Reference 3D ..... 18
Configuration ..... 19
SettingsNames ..... 19
Viewer3DConfiguration ..... 20
ViewerConfiguration ..... 20
ViewerSettings ..... 21
DisplayMode ..... 22
Events ..... 23
Extensions ..... 25
Extension ..... 25
ExtensionLoader ..... 27
ExtensionManager ..... 28
GizmoControl ..... 30
GizmoAxis ..... 30
GizmoBuilder ..... 30
GizmoObject ..... 30
GizmoAxis ..... 31
GizmoBuilder ..... 37
GizmoControl ..... 41
GizmoMaterials ..... 44
GizmoRotationAxis ..... 45
GizmoScaleAxis ..... 46
GizmoTranslationAxis ..... 47
IGizmoObject ..... 48
GizmoObject ..... 48
GuiViewer3D ..... 51

IfcType ..... 52
Model ..... 73
ModelElement ..... 79
ModelElement ..... 79
ModelElementIds ..... 81
ModelElementProperty ..... 82
ModelElementPropertySet ..... 83
ModelElementPropertyValue ..... 84
ModelElementTree ..... 85
ModelLoadingOptions ..... 87
ModelPart ..... 88
Navigation ..... 89
CameraNavigationMode ..... 89
CameraOrientation ..... 90
CameraParameters ..... 91
DesktopNavigation ..... 92
ICameraControl ..... 93
INavigation ..... 97
INavigationAgent ..... 100
INavigationEventSource ..... 102
NavigationEventSourceEventMap ..... 102
NavigationEvent ..... 102
NavigationEventOptions ..... 103
NavigationHandlerPriority ..... 103
INavigationTool ..... 104
MobileNavigation ..... 106
NavigationTool ..... 107
Point3 ..... 108
PilotWeb3D ..... 109
Render ..... 111
Color ..... 111
I3DRenderer ..... 114
IntersectionChecker ..... 117
ISceneIntersectionChecker ..... 120

IModelIntersectionChecker ....120
IRenderOperationContext....122
IRenderViewer3D ....125
IUserScene ....128
LabelSprite ....132
MeshLine....136
MeshLineMaterial ....136
MeshLineGeometry ....139
UpdateType....142
ViewObject....143
SelectionMode ....151
User Interface....152
ButtonBuilder....152
Control ....154
Toolbar ....155
ToolbarBuilder ....156
ViewerToolbar....157
Viewer3D....158
API Reference 2D....161
Configuration ....162
Viewer2DConfiguration ....162
ViewerConfiguration ....163
ViewerSettings ....164
ViewerToolbar....165
PilotWeb2D ....166
GuiViewer2D ....167
Viewer2D....168
Events....170
Extensions ....171
Extension....171
ExtensionLoader....173
ExtensionManager ....174
IAnnotationLayer....175
IDocument....177

IDocumentPage ..... 179
IPageUpdateParams ..... 181
DocumentLoadingOptions ..... 182
Point2 ..... 183
Common API Reference ..... 184
Button ..... 185
Control ..... 187
ControlState ..... 189
Dialog ..... 190
ElementClass ..... 190
IconsSet ..... 194
ISettings ..... 196
Localization ..... 197
Toolbar ..... 199
ViewerConfiguration ..... 201
ToolbarDirection ..... 202
ToolbarContentAlignment ..... 203
SettingsTheme ..... 203
ViewerSettings ..... 203
WindowStater ..... 204
IWindowStyle ..... 204
IWindowStateOptions ..... 204
ExtensionLoader ..... 206
IEventsDispatcher ..... 207
ComboButton ..... 209
SubMenu ..... 210
IControl ..... 212
Select ..... 213
ISelectItem ..... 213
Checkbox ..... 215
ExtensionActivationController ..... 216
Базовые расширения 3D ..... 217
BoxSelection ..... 218
ClippingPlaneExtension ..... 220

DeleteButtonExtension ....223
FullScreenExtension ....224
MeasurementToolsExtension ....226
ModelsBrowserExtension ....227
RemarksExtension ....228
RemarkManager ....229
RemarkEventMap ....232
RemarkViewObject ....232
RemarkObjectParameters ....234
RemarkStatusParameters ....235
ViewerSettingsExtension ....237
WasdNavigationExtension ....238
Базовые расширения 2D ....241
RemarksExtension ....242
RemarksManager ....243
Remark ....245
RemarkHtmlContainer ....246
RemarkParameters ....246
RemarkStatus ....248
ZoomExtension ....249
Примеры расширений 3D ....250
CameraChangeExtension ....251
ElementPropertiesExtension ....252
RemarksUIExtension ....253
SceneObserverExtension ....254
SelectionExtension ....255
SetPivotPositionExtension ....256
Примеры расширений 2D ....257
RemarksUIExtension ....258

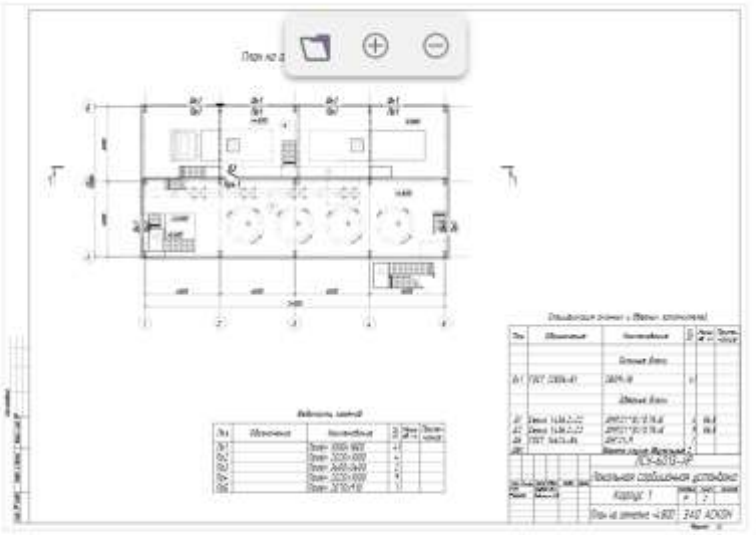
**Pilot-ComponentKit** — это облачная платформа для создания индивидуальных BIM-решений на любом этапе жизненного цикла объекта капитального строительства от проектирования до эксплуатации.

Pilot-ComponentKit функционирует как PaaS (платформа как сервис), предоставляя компоненты для работы с BIM-моделью и XPS-документами. Используя компоненты системы, такие как комплекс сервисов и API, вы можете разрабатывать собственные инженерные решения.

**PilotWeb3D**, **PilotBimDataProvider** и **PilotWeb2D** – это компоненты Pilot-ComponentKit. Они включают в себя библиотеки JavaScript, стили, методы API и документацию, интегрируются в любой веб-сайт, портал или мобильное приложение. С помощью этих компонентов решаются задачи отображения в браузерах информационных моделей (BIM) и документов, навигации по ним, работы с данными трёхмерных моделей, в том числе скрытия элементов модели, изменения их цвета, получения свойств, выполнения измерений.

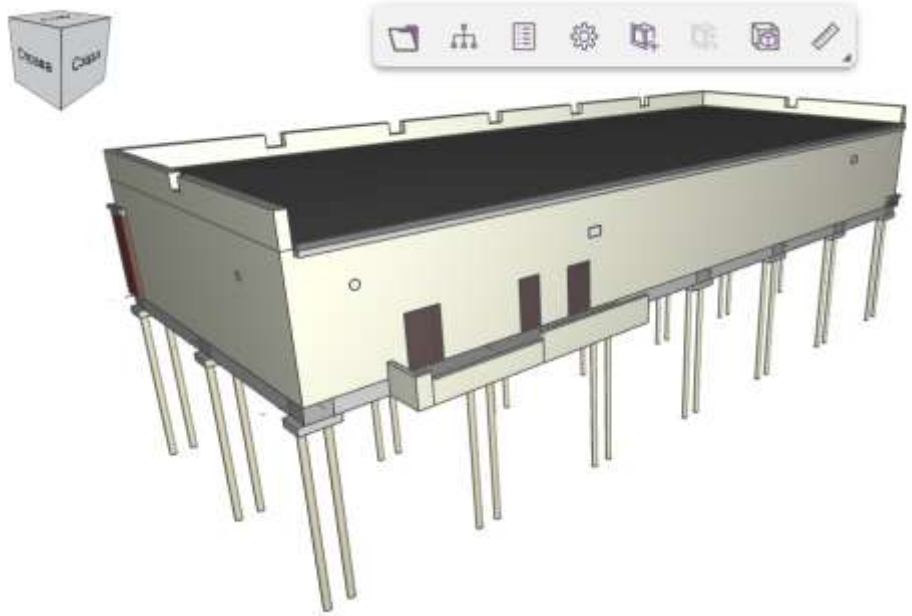
## Документы

Для загрузки документов используйте файлы в формате **.xps** (Open XML Paper Specification, ECMA-388).



## Информационные модели

Для загрузки моделей в компоненты **PilotWeb3D** и **PilotBimDataProvider** используются файлы **.bm**, формируемые сервисом **Pilot-BIM-Server**. Файлы **.bm** – это контейнер данных информационной модели. Он может содержать как изменение данных относительно предыдущей версии, так и актуальное состояние модели, включающее все её изменения до необходимой версии.



Файлы **.bm** получаются в результате работы системы Pilot в редакции Pilot-BIM. Это клиент-серверная система для управления данными на основе технологии информационного моделирования, состоящая из сервера **Pilot-Server**, сервера **Pilot-BIM-Server**, приложения для администрирования **Pilot-myAdmin**, компонента **CadFarmApp** и клиентского приложения **Pilot-BIM**.

**Pilot-BIM-Server** в составе системы решает следующие задачи:

- отслеживает появление и обновление IFC-файлов на Pilot-Server, обеспечивая актуальность информационной модели;
- осуществляет построение тесселяций: преобразует B-rep, описанный в IFC-файле, в триангуляционное представление BIM-объекта;
- создаёт и изменяет индексированные BIM-объекты в базе данных модели (Model DB);
- при обновлении IFC-файла сравнивает версии изменившихся моделей, определяет различия в базе данных модели (вычисляет diff Model DB), осуществляет перестроение тесселяций и изменение индексированных объектов в базе данных только для изменённых объектов.

# Системные требования

## Программное обеспечение

Для работы с компонентами рекомендуется использовать браузер, совместимый с WebGL-canvas:

- Google Chrome 104+
- Firefox 106+
- Microsoft Edge 104+
- Яндекс.Браузер 22+

## Аппаратное обеспечение

Наличие аппаратного графического ускорителя.

# ПЕРВЫЕ ШАГИ

## Добавить компоненты на HTML страницу

Следующий пример показывает как добавить компонент для работы с **BIM-моделью** на вашу HTML страницу.

```
<head>
    <meta name="viewport" content="width=device-width, minimum-scale=1.0,
initial-scale=1, user-scalable=no" />
    <meta charset="utf-8">

    <link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/pilotweb3d/style.css"
type="text/css">
    <script
src="https://pilotcloud.ascon.net/components/24.2.0/pilotweb3d/pilotweb3d.min
.js"></script>

    <style>
        body {
            margin: 0;
        }
        #pilotViewer {
            width: 100%;
            height: 100%;
            margin: 0;
        }
    </style>
</head>
<body>
  <div id="pilotViewer"></div>
</body>
```

Следующий пример показывает, как добавить компонент для работы с **документами** на вашу HTML страницу.

```
<head>
    <meta name="viewport" content="width=device-width, minimum-scale=1.0,
initial-scale=1, user-scalable=no" />
    <meta charset="utf-8">

    <link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/pilotweb2d/style.css"
type="text/css">
    <script
src="https://pilotcloud.ascon.net/components/24.2.0/pilotweb2d/pilotweb2d.min
.js"></script>

    <style>
        body {
            margin: 0;
        }
        #pilotViewer {
            width: 100%;
            height: 100%;
            margin: 0;
        }
    </style>
</head>
```

```
<body>
  <div id="pilotViewer"></div>
</body>
```

**Пакеты** PilotWeb3D **и** PilotWeb2D **имеют довольно большой размер, поэтому мы рекомендуем подключать эти библиотеки к странице HTML как можно позже.**

## Инициализация компонентов

Инициализация компонента **PilotWeb3D** для работы с BIM-моделью:

```
var viewer;
var options = {};

PilotWeb3D.Initializer(options, async () => {
    var htmlDiv = document.getElementById('pilotViewer');
    viewer = PilotWeb3D.CreateViewer(htmlDiv);
    await viewer.start();
    console.log('Initialization complete, loading a model next...');
  });
```

Инициализация компонента **PIlotWeb2D** для работы с документами:

```
var viewer;
var options = {};

PilotWeb2D.Initializer(options, async () => {
    var htmlDiv = document.getElementById('pilotViewer');
    viewer = PilotWeb2D.CreateViewer(htmlDiv);
    await viewer.start();
    console.log('Initialization complete, loading a document next...');
  });
```

Функцию инициализации достаточно вызвать один раз.

### Пример создания экземпляров компонентов

Как только функция обратного вызова инициализации `Initializer` была вызвана, мы можем создать экземпляр `GuiViewer3D`.

Пример создания компонента для BIM-моделей:

```
  let htmlDiv = document.getElementById('pilotViewer');
  let viewer = PilotWeb3D.CreateViewer(htmlDiv);
```

Пример создания компонента для документов:

```
  let htmlDiv = document.getElementById('pilotViewer');
  let viewer = PilotWeb2D.CreateViewer(htmlDiv);
```

Далее необходимо вызвать метод `start()`, который инициализирует компонент:

```
  await viewer.start();
```

### Освобождение ресурсов

Если компоненты больше не нужны на странице, следует завершить их работу:

```
  viewer.finish();
  viewer = null;
```

## Загрузка частей модели

Перед тем, как загрузить часть модели в компонент **PilotWeb3D**, её необходимо получить из системы Pilot-BIM.

Пример загрузки одной части модели:

```
  let options = {
    Guid: bmFileId
  };
  let buffer = // ArrayBuffer from .bm file

  viewer.loadModelPart(buffer, options, () => {
    console.log(`${options.Guid}: The model loaded successfully`);
  },(e) => {
    console.error(`${options.Guid}: An error occured while loading model
part: ${e}`);
  });
```

Пример загрузки нескольких частей модели:

```
  let options = {
    isConsolidatedModel: true,
    Guid: bmFileId
  };
  let buffer = // ArrayBuffer from .bm file

  viewer.loadModelPart(duffer, options, () => {
    console.log(`${options.Guid}: The model loaded successfully`);
  },(e) => {
    console.error(`${options.Guid}: An error occured while loading model
part: ${e}`);
  });
```

**Чтобы добавить часть модели к уже загруженным частям в `options` следует задать параметр `isConsolidatedModel : true`**

**`Guid` – следует указывать уникальный идентифкатор части модели в рамках одной консолидированной модели .**

**`buffer` - это массив байт полученный из файла** .bm **из системы** Pilot-BIM**.**

## Загрузка документа

Компонент **PilotWeb2D** предназначен для просмотра документов в формате **XPS**. Никаких предварительных требований для работы с документом нет.

Пример загрузки документа:

```
let options = {};
let buffer = // ArrayBuffer from xps document
let viewer.loadDocument(buffer, {}, () => {
  console.log("The document loaded successfully");
  /* place your code here */
},
(e) => {
  console.log("An error occured while loading document: " + e)
});
```

**`buffer` - это массив байт полученный из файла .xps.**

## Создание расширения

С помощью расширений можно дополнять функциональность компонентов работы с BIM-моделями и документами.

### *Шаг 1. Подключение файла расширения*

Расширение должно быть подключено после подключения всех классов ядра компонента **PilotWeb3D** или **PilotWeb2D**. Пример для подключения расширения `my-extension.js`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/pilotweb3d/pilotweb3d.min
.js"></script>
<script src="my-extension.js"></script>
```

### *Шаг 2. Пишем код расширения*

Чтобы написать свое расширение для компонентов необходимо:

1. Унаследоваться от класса `PilotWeb3D.Extension` или `PilotWeb2D.Extension`.
2. Зарегистрировать расширение с уникальным именем в системе.

Пример:

```
class My3DExtension extends PilotWeb3D.Extension {

  constructor(viewer) {
    super(viewer);
  }

  getName() {
    return 'My3DExtension';
  }

  load() {
    super.load();
    alert('My3DExtension is loaded!')
    return true;
  }

  unload() {
    super.unload();
    alert('My3DExtension is unloaded!')
    return true;
  }
}

PilotWeb3D.theExtensionManager.registerExtensionType('My3DExtension',
My3DExtension);
```

### *Шаг 3. Загрузка расширения*

Пример загрузки расширения для компонента **PilotWeb3D**:

```
var htmlDiv = document.getElementById('pilotViewer')
```

```
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("My3DExtension");
...
viewer.loadModelPart(...);
```

Пример загрузки расширения для компонента **PilotWeb2D**:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb2D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("My2DExtension");
...
viewer.loadDocument(...);
```

# API Reference 3D

# Configuration

## SettingsNames

**SettingsNames** – список настроек для просмотрщика **PilotWeb3D**.
Для задания настроек используйте класс Viewer3DConfiguration.

```
class SettingsNames {
  static TELEMETRY = "telemetry";
  static ANTI_ALIASING = "antiAliasing";
  static HIDE_EDGES_WHEN_NAVIGATING = "hideEdgesWhenNavigation";
  static DISPLAY_MODE = "displayMode";
  static NAVIGATION_CUBE = "navigationCube";
}
```

### TELEMETRY

Свойство для отображения отладочной информации в просмотрщике. Настройка может иметь значения `true/false/undefined`.

```
static TELEMETRY = "telemetry";
```

### ANTI_ALIASING

Свойство для настроек сглаживания 3D модели. Настройка может иметь значения `true/false/undefined`.

```
static ANTI_ALIASING = "antiAliasing";
```

### HIDE_EDGES_WHEN_NAVIGATING

Свойство для указания настройки скрытия ребер при навигации по 3D модели. Настройка может иметь занчения `true/false/undefined`.

```
static HIDE_EDGES_WHEN_NAVIGATING = "hideEdgesWhenNavigation";
```

### DISPLAY_MODE

Свойство для указания настройки отображения моделей в просмотрщике. Настройка может иметь значения `DisplayMode.FACES_AND_EDGES` / `DisplayMode.FACES` или `undefined`. Подробнее: DisplayMode.

```
static DISPLAY_MODE = "displayMode";
```

### NAVIGATION_CUBE

Свойство для указания настройки скрытия навигационного куба. Настройка может иметь занчения `true/false/undefined`.

```
static NAVIGATION_CUBE = "navigationCube";
```

## Viewer3DConfiguration

**Viewer3DConfiguration** – класс, описывающий настройки компонента **PilotWeb3D**. Этот класс наследуется от базового класса настроек ViewerConfiguration.

```
class Viewer3DConfiguration extends ViewerConfiguration {
  settings?: ViewerSettings;
}
```

### *Свойства*

#### settings

Необязательное поле для задания настроек отображения просмотрщика 3D моделей. Подробнее: ViewerSettings.

```
settings?: ViewerSettings;
```

## ViewerConfiguration

**ViewerConfiguration** – базовый класс, описывающий настройки компонентов **PilotWeb2D** и **PilotWeb3D**.

```
class ViewerConfiguration {
  appearance: ViewerSettings = {
    [BaseSettingsNames.TOOLBAR] : {
      direction: ToolbarDirection.TOP_FLUENT,
      content: ToolbarContentAlignment.CENTER
    } as ToolbarStyle
  };
}
```

### *Свойства*

#### appearance

Свойство для изменения внешнего вида просмотрщика.

```
appearance: ViewerSettings;
```

#### direction

Свойство управляет положением панели инструментов.

```
direction: string;
export enum ToolbarDirection {
```

```
  TOP_FIXED = 'ascn-toolbar-fixed-top',
  TOP_FLUENT = 'ascn-toolbar-top',
  BOTTOM_FIXED = 'ascn-toolbar-fixed-bottom',
  BOTTOM_FLUENT = 'ascn-toolbar-bottom',
}
```

### content

Свойство управляет расположением кнопок на панели инструментов.

```
content: string;
export enum ToolbarContentAlignment {
  CENTER = 'ascn-toolbar-content-center',
  START = 'ascn-toolbar-content-start',
  END = 'ascn-toolbar-content-end'
}
```

### settingsPrefix

Свойство задает префикс для настроек, хранящихся в браузере клиента.

```
settingsPrefix: string;
```

## ViewerSettings

**ViewerSettings** – класс, описывающий настройки просмотрщика компонента **PilotWeb3D**.

```
type ViewerSettings = Record<string, any>;
```

### Настройки просмотрщика по умолчанию

Компонент имеет следующие настройки, установленные по умолчанию:

```
const defaultViewer3DSettings: ViewerSettings = {
  [SettingsNames.TELEMETRY] : false,
  [SettingsNames.AXES] : true,
  [SettingsNames.CAMERA_ANIMATION]: true,
  [SettingsNames.HIDE_SMALL_ELEMENTS_WHEN_NAVIGATING]: false,
  [SettingsNames.GLOBAL_LIGHT]: true,
  [SettingsNames.LIGHT_SOURCE]: true,
  [SettingsNames.ANTI_ALIASING]: true,
  [SettingsNames.HIDE_SMALL_ELEMENTS_MOVING]: false,
  [SettingsNames.SMALL_ELEMENT_SIZE]: 10,
  [SettingsNames.LABEL_LINE_LENGTH]: 1000,
  [SettingsNames.HIDE_EDGES_WHEN_NAVIGATING]: true,
  [SettingsNames.DISPLAY_MODE]: DisplayMode.FACES_AND_EDGES,
  [SettingsNames.NAVIGATION_CUBE]: true
}
```

# DisplayMode

## Режим отображения 3D модели

```
enum DisplayMode {
  FACES_AND_EDGES = 0, // отображать ребра и грани
  FACES = 1 // отображать только грани
}
```

# Events

## Системные события

### Имена общих событий

```
class CoreEventTypes {
  // Имя события изменения размера просмотрщика.
  static VIEWER_RESIZE_EVENT: string;
  // Имя события нажатия левой клавиши мыши.
  static VIEWER_MOUSE_DOWN_EVENT: string;
  // Имя события перемещения мыши.
  static VIEWER_MOUSE_MOVE_EVENT: string;
  // Имя события отпускания левой клавиши мыши.
  static VIEWER_MOUSE_UP_EVENT: string;
  // Имя события изменения настройки.
  static SETTING_CHANGED_EVENT: string;
  // Имя события восстановления настройки в значение по умолчанию.
  static SETTING_RESET_EVENT: string;
  // Имя события загрузки расширения. Вызывается после загрузки расширения.
  static EXTENSION_LOADED;
  // Имя события выгрузки расширения. Вызывается непосредственно перед
выгрузкой расширения.
  static EXTENSION_UNLOADING;
  // Имя события выгрузки расширения. Вызывается после выгрузки расширения.
  static EXTENSION_UNLOADED;
}
```

### Имена событий для 3D

```
class EventTypes extends CoreEventTypes {
  // Имя события изменения селектированного элемента.
  static SELECTION_CHANGED_EVENT: string;
  // Имя события загрузки части консолидированной модели.
  static MODEL_PART_LOADED: string;
  // Имя события выгрузки части консолидированной модели.
  static MODEL_PART_UNLOADED: string;
  // Имя события обновления части консолидированной модели.
  static MODEL_PART_UPDATED: string;
  // Имя события изменения положения виртуального начала координат.
  static VIRTUAL_ORIGIN_CHANGED: string;
  // Имя события изменения положения камеры.
  static CAMERA_CHANGE_EVENT: string;
  // Имя события изменения типа навигации камеры.
  static CAMERA_NAVIGATION_MODE_CHANGED_EVENT: string;
  // Имя события клика по отрисованному элементу.
  static RENDER_CLICK_EVENT: string;
  // Имя события ховера отрисованного элемента.
  static RENDER_HOVER_EVENT: string;
  // Имя события двойного клика по отрисованному элементу.
  static RENDER_DOUBLE_CLICK_EVENT: string;
  // Имя события удаления объектов со сцены.
  static DELETE_OBJECTS_EVENT: string;
}
```

### Классы событий для 3D

```
// Класс события изменения селектированного элемента.
class SelectionChangedEvent extends Event {
  selectedIds: ModelElementIds[]; // Массив идентификаторов селектированных
элементов.
}

// Класс события загрузки или выгрузки части консолидированной модели.
class ModelPartEvent extends Event {
  modelPartId: string; // Идентификатор части консолидированной модели.
}

// Класс события обновления части консолидированной модели.
class ModelPartUpdateEvent extends ModelPartEvent {
  updatedElementIds?: string[]; // Идентификаторы обновлённых элементов
модели.
  removedElementIds?: string[]; // Идентификаторы удалённых элементов модели.
  addedElementIds?: string[]; // Идентификаторы добавленных элементов модели.
}

// Класс события изменения виртуального начала координат.
class VirtualOriginEvent extends Event {
  virtualOrigin: Point3; // Обновлённое положение виртуального начала
координат.
  delta: Point3; // Смещение объектов на сцене: oldOrigin – newOrigin.
}

// Класс события изменения положения камеры.
class CameraEvent extends Event {}

// Класс события клика по отрисованному элементу.
class ClickedEvent extends Event {
  modelId: string; // Идентификатор модели.
  modelElementId: string; // Идентификатор элемента модели.
  ctrlKey: boolean; // Флаг нажатия клавиши Ctrl.
}

// Класс события ховера отрисованного элемента.
class HoverEvent extends Event {
  modelId: string; // Идентификатор модели.
  modelElementId: string; // Идентификатор элемента модели.
}

// Класс события загрузки или выгрузки расширения.
class ExtensionEvent extends Event {
  extensionName: string;
}

// Класс события удаления объектов со сцены.
class DeleteEvent extends Event {
  deletedIds: ModelElementIds[]; // Идентификаторы объектов для удаления.
}
```

# Extensions

## Extension

**Extension** – это базовый класс описания расширения.

### *Свойства*

#### _viewer

```
protected _viewer: Viewer3D;
```

### *Методы*

#### load()

Метод вызвается, когда расширение было загружено.

```
load() : boolean | Promise<boolean>;
```

#### unload()

Метод вызвается, когда расширение было выгружено.

```
unload(): boolean;
```

#### activate()

Метод активирует работу модуля расширения.

```
activate() : boolean;
```

Возвращает `true`, если активация прошла успешно.

#### deactivate()

Метод деактивирует работу модуля расширения.

```
deactivate(): boolean;
```

Возвращает `true`, если деактивация прошла успешно.

#### getName()

Метод вызывается, когда расширение пытается получить имя расширения.

```
getName(): string;
```

## onToolbarCreated()

Метод вызывается, когда панель инструментов построилась, и расширение имеет возможность добавить/изменить/удалить элементы управления.

```
onToolbarCreated(builder: ToolbarBuilder): void;
```

где:
`builder` – построитель панели инструментов.

## onMouseDown()

Метод вызывается, когда произошло событие нажатия левой клавиши мыши.

```
onMouseDown(event: MouseEvent): void;
```

где:
`event` – событие мыши.

## onMouseMove()

Метод вызывается, когда произошло событие перемещения мыши.

```
onMouseMove(event: MouseEvent): void;
```

где:
`event` – событие мыши.

## onMouseUp()

Метод вызывается, когда произошло событие отпускания левой клавиши мыши.

```
onMouseUp(event: MouseEvent): void;
```

где:
`event` – событие мыши.

## ExtensionLoader

`ExtensionLoader` – это класс, предназначенный для загрузки и инициализации расширений, которые были зарегистрированы в компонентах. Перед тем, как загрузить расширение, его необходимо зарегистрировать с помощью `ExtensionManager`.

Пример:

```
// описываем расширение
class MyExtension extends PilotWeb3D.Extension {
  ...
}
// регистрируем
PilotWeb3D.theExtensionManager.registerExtensionType('myExtension',
MyExtension);

// загружаем в компонент
let viewer = PilotWeb3D.CreateViewer(div);
viewer.extensionLoader.loadExtension('myExtension');
```

### *Методы*

### loadExtension()

Метод загружает зарегистрированное расширение в компонент.

```
loadExtension(extensionId: string): Promise<Extension>;
```

где:
`extensionId` – уникальное имя расширения.

### unloadExtension()

Метод выгружает расширение.

```
unloadExtension(extensionId: string) : Promise<boolean>;
```

где:
`extensionId` – уникальное имя расширения.

### getExtensions()

Метод получает все загруженные расширения.

```
getExtensions(): Extension[];
```

## ExtensionManager

**ExtensionManager** – это класс-менеджер расширений, позволяет зарегистрировать или разрегистрировать расширения в компоненте **PilotWeb3D**.

`ExtensionManager` доступен из пространства имен **PilotWeb3D** через свойство `theExtensionManager`.

Пример:

```
// описываем расширение
class MyExtension extends PilotWeb3D.Extension {
  ...
}
// регистрируем
PilotWeb3D.theExtensionManager.registerExtensionType('myExtension',
MyExtension);
```

### *Методы*

### registerExtensionType()

Метод регистрирует новое расширение в системе. После этого это расширение можно загрузить.

```
registerExtensionType(extensionId: string, extension: typeof Extension) :
boolean;
```

где:
`extensionId` – уникальное имя расширения.

`extension` – тип расширения, унаследованный от `PilotWeb3d.Extension` или `PilotWeb2D.Extension`.

### unregisterExtensionType()

Метод разрегистрирует расширение.

```
unregisterExtensionType(extensionId: string) : boolean;
```

где:
`extensionId`– уникальное имя расширения.

### getExtensionType()

Метод получает тип зарегистрированного расширения.

```
getExtensionType(extensionId: string): typeof Extension
```

где:
`extensionId`– уникальное имя расширения.

# GizmoControl

GizmoControl – контроллер, прикрепляемый к 3D-объекту на сцене и управляющий его положением.

## GizmoAxis

GizmoAxis – базовый класс осей `GizmoControl`.

GizmoTranslationAxis – встроенная реализация оси переноса.
GizmoRotationAxis – встроенная реализация оси вращения.
GizmoScaleAxis – встроенная реализация оси масштабирования.

## GizmoBuilder

GizmoBuilder – вспомогательный класс для построения `GizmoControl` с реализацией по умолчанию.

## GizmoObject

IGizmoObject – базовый интерфейс гизмо объектов.
GizmoObject – встроенная реализация интерфейса `IGizmoObject`. GizmoMaterials – материалы, используемые объектами гизмо.

## GizmoAxis

**GizmoAxis** – абстрактный класс, базовый для всех осей GizmoControl.
Каждая ось описывает свой тип манипуляций над контролом.

Примеры реализаций: GizmoTranslationAxis, GizmoRotationAxis, GizmoScaleAxis.

```
export class GizmoAxis extends THREE.Object3D implements IGizmoObject {
  protected _isHovered: boolean;
  protected _isActive: boolean;
  protected _plane: THREE.Plane;
  protected _axisDir: THREE.Vector3;
  protected _raycaster: THREE.Raycaster;

  protected _worldPositionStart: THREE.Vector3;
  protected _worldQuaternionStart: THREE.Quaternion;
  protected _worldAxisDir: THREE.Vector3;

  protected _startPoint: THREE.Vector3;
  protected _endPoint: THREE.Vector3;

  constructor(axisDir: THREE.Vector3, readonly handle: IGizmoObject, readonly
picker?: IGizmoObject, readonly helper?: IGizmoObject);

  getActive(): boolean;
  setActive(value: boolean): void;
  getHovered(): boolean;
  setHovered(value: boolean): void;
  dispose(): void;
  abstract moveByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera):
THREE.Matrix4;
  protected abstract updateGizmoPlane(camera: THREE.Camera): void;
  protected setStartPt(ndcPos: THREE.Vector2, camera: THREE.Camera): void;
}
```

### *Поля*

### handle: IGizmoObject

Поле хранит геометрию оси, рисуемую на сцене. Если picker не задан, то используется для проверки пересечений. Подробнее: IGizmoObject.

```
readonly handle: IGizmoObject;
```

### picker: IGizmoObject

Поле хранит геометрию оси, используемую для проверки пересечений. Подробнее: IGizmoObject.

```
readonly picker: IGizmoObject;
```

### helper: IGizmoObject

Поле хранит вспомогательную геометрию оси. Подробнее: IGizmoObject.

```
readonly helper: IGizmoObject;
```

### _isHovered: boolean

Поле хранит значение ховера для оси.

```
protected _isHovered: boolean;
```

### _isActive: boolean

Поле хранит значение активности оси.

```
protected _isActive: boolean;
```

### _plane: THREE.Plane

Поле хранит плоскость, используемую для расчётов смещения курсора при манипуляциях с осью. Подробнее: THREE.Plane.

```
protected _plane: THREE.Plane;
```

### _axisDir: THREE.Vector3

Поле хранит вектор направления оси в локальных координатах. Подробнее: THREE.Vector3.

```
protected _axisDir: THREE.Vector3;
```

### _raycaster: THREE.Raycaster

Поле хранит объект THREE.Raycaster, используемый для расчётов смещения при манипуляциях с осью.

```
protected _raycaster: THREE.Raycaster;
```

### _worldPositionStart: THREE.Vector3

Поле хранит значение положения оси в мировых координатах в момент активации оси. При setActive(`true`) в данный вектор сохраняется значение положения оси. Используется для расчета смещения оси относительно начального положения во время манипуляций над осью. Подробнее: THREE.Vector3.

```
protected _worldPositionStart: THREE.Vector3;
```

### _worldQuaternionStart: THREE.Quaternion

Поле хранит значение кватерниона оси в момент активации оси.
При setActive(`true`) в данный кватернион сохраняется значение поворота оси.
Используется для расчета поворота оси относительно начального положения во время манипуляций над осью. Подробнее: THREE.Quaternion.

```
protected _worldQuaternionStart: THREE.Quaternion;
```

## _worldAxisDir: THREE.Vector3

Поле хранит вектор направления оси в мировых координатах в момент активации оси.
При setActive(`true`) в данный вектор сохраняется направление оси в мировых координатах.
Подробнее: THREE.Vector3.

```
protected _worldAxisDir: THREE.Vector3;
```

## _startPoint: THREE.Vector3

Поле хранит начальное положение курсора на плоскости оси при начале манипуляции с осью. Используется для расчета смещения курсора в плоскости оси во время манипуляций с осью. Подробнее: THREE.Vector3.

```
protected _startPoint: THREE.Vector3;
```

## _endPoint: THREE.Vector3

Поле хранит текущее или последнее положение курсора на плоскости оси при манипуляции с осью. Используется для расчета смещения курсора в плоскости оси относительно начальной точки во время манипуляций с осью. Подробнее: THREE.Vector3.

```
protected _endPoint: THREE.Vector3;
```

### *Конструктор*

```
constructor(axisDir: THREE.Vector3,

            handle: IGizmoObject,
            picker?: IGizmoObject,
            helper?: IGizmoObject);
```

где:
`axisDir` – направление оси в локальных координатах. Подробнее: THREE.Vector3.

`handle` – задает handle – геометрию оси, рисуемую на сцене. Подробнее: IGizmoObject.

`picker` – задает picker – геометрию оси, используемую для проверки пересечений. Не обязательный параметр. Подробнее: IGizmoObject.

`helper` – задает helper – вспомогательную геометрию оси. Не обязательный параметр. Подробнее: IGizmoObject.

### *Методы*

## getHovered()

Метод возвращает значение ховера для оси.

```
getHovered(): boolean;
```

Возвращает `true`, если ховер активен.

## setHovered()

Метод устанавливает значение ховера для оси, а так же вызывает методы `setHovered(value)` у объектов handle, picker и helper.

```
setHovered(value: boolean): void;
```

где:
`value` – значение ховера.

## getActive()

Метод возвращает статус активности для оси.

```
getActive(): boolean;
```

Возвращает `true`, если объект активен.

## setActive()

Метод устанавливает статус активности для оси, а так же вызывает методы `setActive(value)` у объектов handle, picker и helper.

```
setActive(value: boolean): void;
```

где:
`value` – значение активности оси. Если `value` равен `true`, то запоминает текущее положение оси в мировых координатах, кватернион поворота и вектор направления оси в мировых координатах. В противном случае сбрасывает значения начального и конечного положения курсора.

## dispose()

Метод освобождает ресурсы, выделенные оси, а также вызывает методы `dispose()` у объектов handle, picker и helper.

```
dispose(): void;
```

## moveByNdcPt()

Абстрактный метод, вычисляет матрицу трансформации объекта, вызванной манипуляциями с осью.

```
abstract moveByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera):
THREE.Matrix4;
```

где:
`ndcPos` – текущее положение курсора в Normalized Device Coordinates (NDC пространство). Подробнее: THREE.Vector2.

`camera` – камера, используемая для отрисовки `GizmoAxis` на сцене. Используется для перевода Normalized Device Coordinates (NDC пространство) в мировые координаты. Подробнее: THREE.Camera.

Возвращает матрицу трансформации объекта. Подробнее: THREE.Matrix4.

## updateGizmoPlane()

Абстрактный метод, обновляет плоскость, используемую для расчётов смещения курсора при манипуляциях с осью.

```
protected abstract updateGizmoPlane(camera: THREE.Camera): void;
```

где:
`camera` – камера, используемая для отрисовки `GizmoAxis` на сцене. Подробнее: THREE.Camera.

## setStartPt()

Метод вычисляет начальное положение курсора на плоскости оси.

```
protected setStartPt(ndcPos: THREE.Vector2, camera: THREE.Camera): void;
```

где:
`ndcPos` – положение курсора в Normalized Device Coordinates (NDC пространство). Подробнее: THREE.Vector2.

`camera` – камера, используемая для отрисовки `GizmoAxis` на сцене. Используется для перевода Normalized Device Coordinates (NDC пространство) в мировые координаты. Подробнее: THREE.Camera.

## GizmoBuilder

**GizmoBuilder** – вспомогательный класс, конструирующий GizmoControl с реализацией по умолчанию.

```
export class GizmoBuilder {
  public static build(camera: THREE.Camera, navAgent: INavigationAgent,
translation: GizmoAxisDir = GizmoAxisDir.XYZ, rotation: GizmoAxisDir =
GizmoAxisDir.XYZ, scale: GizmoAxisDir = GizmoAxisDir.XYZ): GizmoControl;

  public static buildTranslationAxis(axisDir: THREE.Vector3,
    handleBaseMatrial?: THREE.Material, handleHoveredMaterial?:
THREE.Material, handleSelectedMaterial?: THREE.Material,
    pickerBaseMatrial?: THREE.Material, pickerHoveredMaterial?:
THREE.Material, pickerSelectedMaterial?: THREE.Material
  ): GizmoAxis;

  public static buildRotationAxis(axisDir: THREE.Vector3,
    handleBaseMatrial?: THREE.Material, handleHoveredMaterial?:
THREE.Material, handleSelectedMaterial?: THREE.Material,
    pickerBaseMatrial?: THREE.Material, pickerHoveredMaterial?:
THREE.Material, pickerSelectedMaterial?: THREE.Material
  ): GizmoAxis;

  public static buildScaleAxis(axisDir: THREE.Vector3,
    handleBaseMatrial?: THREE.Material, handleHoveredMaterial?:
THREE.Material, handleSelectedMaterial?: THREE.Material,
  ): GizmoAxis;
```

### *Методы*

### build()

Строит GizmoControl с реализацией по умолчанию.

```
public static build(camera: THREE.Camera, navAgent: INavigationAgent,
  translation: GizmoAxisDir,
  rotation: GizmoAxisDir,
  scale: GizmoAxisDir): GizmoControl;
```

где:
`camera` – камера, используемая для отрисовки `GizmoControl` на сцене. Подробнее: THREE.Camera.

`navAgent` – агент навигации. Подробнее: INavigationAgent..

`translation` – перечисление осей переноса, которые необходимо добавить в контроллер. Не обязательный параметр. По умолчанию GizmoAxisDir.**XYZ**.

`rotation` – перечисление осей вращения, которые необходимо добавить в контроллер. Не обязательный параметр. По умолчанию GizmoAxisDir.**XYZ**.

`scale` – перечисление осей масштабирования, которые необходимо добавить в контроллер. Не обязательный параметр. По умолчанию GizmoAxisDir.**XYZ**.

Возвращает объект GizmoControl.

## buildTranslationAxis()

Строит ось переноса с реализацией по умолчанию. Подробнее GizmoTranslationAxis

```
public static buildTranslationAxis(axisDir: THREE.Vector3,
  handleBaseMatrial?: THREE.Material,
  handleHoveredMaterial?: THREE.Material,
  handleSelectedMaterial?: THREE.Material,
  pickerBaseMatrial?: THREE.Material,
  pickerHoveredMaterial?: THREE.Material,
  pickerSelectedMaterial?: THREE.Material
): GizmoAxis
```

где:
`axisDir` – направление оси переноса в локальных координатах. Подробнее: THREE.Vector3.

`navAgent` – агент навигации. Подробнее: INavigationAgent.

`handleBaseMatrial` – базовый материал handle - геометрии. Необязательный параметр. По умолчанию: GizmoMaterials.**gizmoMaterial**.

`handleHoveredMaterial` – материал handle - геометрии, при активном ховере. Не обязательный параметр. По умолчанию: GizmoMaterials.**matYellow**.

`handleSelectedMaterial` – материал handle - геометрии, при активной манипуляции над осью. Не обязательный параметр. По умолчанию: GizmoMaterials.**matYellow**.

`pickerBaseMatrial` – базовый материал picker - геометрии. По умолчанию: GizmoMaterials.**matInvisible**.

`pickerHoveredMaterial` – материал picker - геометрии, при активном ховере. Не обязательный параметр. По умолчанию: GizmoMaterials.**matInvisible**.

`pickerSelectedMaterial` – материал picker - геометрии, при активной манипуляции над осью.GizmoMaterials.**matInvisible**.

Возвращает объект GizmoAxis.

## buildRotationAxis()

Строит ось вращения с реализацией по умолчанию. Подробнее GizmoRotationAxis

```
public static buildRotationAxis(axisDir: THREE.Vector3,
  handleBaseMatrial?: THREE.Material,
  handleHoveredMaterial?: THREE.Material,
```

```
    handleSelectedMaterial?: THREE.Material,
    pickerBaseMatrial?: THREE.Material,
    pickerHoveredMaterial?: THREE.Material,
    pickerSelectedMaterial?: THREE.Material
  ): GizmoAxis
```

где:
`axisDir` – направление оси вращения в локальных координатах. Подробнее: THREE.Vector3.

`navAgent` – агент навигации. Подробнее: INavigationAgent.

`handleBaseMatrial` – базовый материал handle - геометрии. Не обязательный параметр. По умолчанию: GizmoMaterials.**gizmoMaterial**.

`handleHoveredMaterial` – материал handle - геометрии, при активном ховере. Не обязательный параметр. По умолчанию: GizmoMaterials.**matYellow**.

`handleSelectedMaterial` – материал handle - геометрии, при активной манипуляции над осью. Не обязательный параметр. По умолчанию: GizmoMaterials.**matViolet**.

`pickerBaseMatrial` – базовый материал picker - геометрии. По умолчанию: GizmoMaterials**.matInvisible**.

`pickerHoveredMaterial` – материал picker - геометрии, при активном ховере. Необязательный параметр. По умолчанию: GizmoMaterials**.matYellowTransparent**.

`pickerSelectedMaterial` – материал picker - геометрии, при активной манипуляции над осью.GizmoMaterials**.matYellowTransparent**.

Возвращает объект GizmoAxis.

## buildScaleAxis()

Строит ось масштабирования с реализацией по умолчанию. Подробнее GizmoScaleAxis

```
  public static buildScaleAxis(axisDir: THREE.Vector3,
    handleBaseMatrial?: THREE.Material,
    handleHoveredMaterial?: THREE.Material,
    handleSelectedMaterial?: THREE.Material
  ): GizmoAxis
```

где:
`axisDir` – направление оси вращения в локальных координатах. Подробнее: THREE.Vector3.

`navAgent` – агент навигации. Подробнее: INavigationAgent.

`handleBaseMatrial` – базовый материал handle - геометрии. Не обязательный параметр. По умолчанию: GizmoMaterials.**gizmoMaterial**.

`handleHoveredMaterial` – материал handle - геометрии, при активном ховере. Необязательный параметр. По умолчанию: GizmoMaterials.**matYellow**.

`handleSelectedMaterial` – материал handle - геометрии, при активной манипуляции над осью. Необязательный параметр. По умолчанию: GizmoMaterials.**matYellow**.

Возвращает объект GizmoAxis.

## GizmoAxisDir

Направление осей GizmoControl, используемые в GizmoBuilder.

```
export enum GizmoAxisDir {
  // Не строить данный тип осей
  NONE = 0,
  // Строить ось X
  X = 1 << 0,
  // Строить ось Y
  Y = 1 << 1,
  // Строить ось Z
  Z = 1 << 2,
  // Строить ось X и ось Y
  XY = X | Y,
  // Строить ось Y и ось Z
  YZ = Y | Z,
  // Строить ось X и ось Y
  XZ = X | Z,
  // Строить оси X, Y и Z
  XYZ = X | Y | Z
}
```

## GizmoControl

**GizmoControl** – контроллер, прикрепляемый к 3D-объекту на сцене и управляющий его положением.

Прикрепляемый к объекту контроллер может быть размещён на сцене как независимо, так и быть добавлен как дочерний элемент к 3D-объекту. Подробнее: .attachTo().
По умолчанию `GizmoControl` помещается в начало координат локального пространства объекта привязки. Перемещение, вращение и масштабирование осуществляется относительно положения `GizmoControl`. Задать смещение `GizmoControl` относительно объекта привязки можно с помощью метода .updateGizmoOffset().

В контроллер можно добавить собственные реализации осей. Для того, чтобы создать свою ось контроллера, необходимо унаследоваться от класса GizmoAxis.

Экземпляр можно создать самостоятельно, либо можно получить реализацию по умолчанию с помощью GizmoBuilder.

```
export class GizmoControl extends THREE.Object3D {
  constructor(camera: THREE.Camera, navAgent: INavigationAgent);
  attachTo(object: THREE.Object3D, asChild = false): void;
  detach(): void;
  updateGizmoOffset(position?: THREE.Vector3, quaternion?: THREE.Quaternion):
void;
  addAxis(axis: GizmoAxis): void;
  dispose(): void;
}
```

### *Конструктор*

```
  constructor(camera: THREE.Camera, navAgent: INavigationAgent);
```

где:
`camera` - камера, используемая на сцене. Подробнее: THREE.Camera.
`navAgent` - агент навигации. Подробнее: INavigationAgent.
Камеру и агент навигации можно получить из INavigation.

```
//Пример создания:
const camera = PilotWeb3D.ViewerInstance.navigation.getCamera();
const navAgent = PilotWeb3D.ViewerInstance.navigation.getNavigationAgent();
const gizmoControl = new PilotWeb3D.GizmoControl(camera, navAgent);
```

### *Методы*

#### attachTo()

Метод прикрепляет `GizmoControl` к 3D-объекту на сцене. Контрол привязывает своё положение к положению объекта, а также, манипуляции над `GizmoControl` начинают

влиять на положение связанного объекта.
Если `GizmoControl` добавляется как дочерний объект, то по умолчанию `GizmoControl` помещается в точку начала координат в локальном пространстве объекта привязки. Для смещения `GizmoControl` относительно объекта привязки используется метод .updateGizmoOffset().

```
attachTo(object: THREE.Object3D, asChild = false): void;
```

где:
`object` – Объект привязки.
`asChild` – Параметр указывает, добавить ли `GizmoControl` дочерним элементом к объекту. Не обязательный параметр, по умолчанию: `false`.
Если `asChild` равен `true`, то `GizmoControl` не нужно добавлять на сцену вручную. Он будет автоматически размещён на той же сцене что и родительский объект. В противном случае `GizmoControl` нужно вручную добавить на нужную сцену и задать необходимые координаты.

## detach()

Метод открепляет `GizmoControl` от объекта привязки, если привязка существует.

```
detach(): void;
```

## updateGizmoOffset()

Метод устанавливает смещение и вращение `GizmoControl` относительно объекта привязки.

```
  updateGizmoOffset(position?: THREE.Vector3, quaternion?: THREE.Quaternion):
void;
```

где:
`position` – позиция `GizmoControl` в локальных координатах объекта привязки. Не обязательный параметр. Подробнее: THREE.Vector3.
`quaternion` – вращение `GizmoControl` в локальных координатах объекта привязки. Не обязательный параметр. Подробнее: THREE.Quaternion.

## addAxis()

Метод добавляет новую ось для манипуляции в `GizmoControl`.

```
  addAxis(axis: GizmoAxis): void;
```

где:
`axis` – ось контроллера. Подробнее: GizmoAxis.

## dispose()

Метод освобождает выделенные контроллеру ресурсы, разрывает привязку, удаляет `GizmoControl` со сцены.

```
dispose(): void;
```

## GizmoMaterials

**GizmoMaterials** – материалы, используемые для отрисовки гизмо объектов. Подробнее: THREE.Material, THREE.MeshBasicMaterial.

```
export class GizmoMaterials {
  // базовый материал для мешей
  static gizmoMaterial: THREE.MeshBasicMaterial;

  // полностью прозрачный материал
  static matInvisible: THREE.MeshBasicMaterial;

  // красный непрозрачный материал
  static matRed: THREE.MeshBasicMaterial;

  // зелёный непрозрачный материал
  static matGreen: THREE.MeshBasicMaterial;

  // синий непрозрачный материал
  static matBlue: THREE.MeshBasicMaterial;

  // желтый непрозрачный материал
  static matYellow: THREE.MeshBasicMaterial;

  // фиолетовый непрозрачный материал
  static matViolet: THREE.MeshBasicMaterial;

  // желтый полупрозрачный материал
  static matYellowTransparent: THREE.MeshBasicMaterial;
}
```

## GizmoRotationAxis

**GizmoRotationAxis** – встроенная реализация оси вращения для GizmoControl. Подробнее: GizmoAxis.

```
export class GizmoRotationAxis extends GizmoAxis {
  constructor(axisDir: THREE.Vector3, readonly handle: IGizmoObject, readonly
picker?: IGizmoObject, readonly helper?: IGizmoObject) {
    super(axisDir, handle, picker, helper);
    this.name = 'GizmoRotationAxis';
  }

  public moveByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera):
THREE.Matrix4 {
    this.updateGizmoPlane(camera);

    if (!this._startPoint)
      this.setStartPt(ndcPos, camera);

    this._raycaster.setFromCamera(ndcPos, camera);
    const planeIntersect = this._raycaster.ray.intersectPlane(this._plane,
new THREE.Vector3());
    if (!planeIntersect)
      return null;

    const localStartPt =
this._startPoint.clone().sub(this._worldPositionStart);
    this._endPoint.copy(planeIntersect);
    const localEndPt = this._endPoint.clone().sub(this._worldPositionStart);

    let angle = localEndPt.angleTo(localStartPt);

    const crossSign =
Math.sign(localStartPt.clone().cross(localEndPt).dot(this._worldAxisDir));
    angle *= crossSign;

    return new THREE.Matrix4().makeRotationAxis(this._axisDir, angle);
  }

  protected override updateGizmoPlane(camera: THREE.Camera): void {
    this._plane.setFromNormalAndCoplanarPoint(this._worldAxisDir,
this._worldPositionStart);
  }
}
```

## GizmoScaleAxis

**GizmoScaleAxis** – встроенная реализация оси масштабирования для GizmoControl. Подробнее: GizmoAxis.

```
export class GizmoScaleAxis extends GizmoAxis {
  constructor(axisDir: THREE.Vector3, readonly handle: IGizmoObject, readonly
picker?: IGizmoObject, readonly helper?: IGizmoObject) {
    super(axisDir, handle, picker, helper);
    this.name = 'GizmoScaleAxis';
  }

  public moveByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera):
THREE.Matrix4 {
    this.updateGizmoPlane(camera);

    if (!this._startPoint)
      this.setStartPt(ndcPos, camera);

    this._raycaster.setFromCamera(ndcPos, camera);

    const planeIntersect = this._raycaster.ray.intersectPlane(this._plane,
new THREE.Vector3());
    if (!planeIntersect)
      return null;

    this._endPoint.copy(planeIntersect);

    const localStart =
this._startPoint.clone().sub(this._worldPositionStart).dot(this._worldAxisDir
);
    const localEnd =
this._endPoint.clone().sub(this._worldPositionStart).dot(this._worldAxisDir);

    const scaleValue = localEnd / localStart;

    return new THREE.Matrix4().makeScale(
      this._axisDir.x !== 0 ? scaleValue : 1,
      this._axisDir.y !== 0 ? scaleValue : 1,
      this._axisDir.z !== 0 ? scaleValue : 1
    );
  }

  protected override updateGizmoPlane(camera: THREE.Camera): void {
    this._raycaster.setFromCamera({ x: 0, y: 0 }, camera);
    const eye = this._raycaster.ray.direction;

    const alignVector = eye.clone().cross(this._worldAxisDir);

    let dirVector = this._worldAxisDir.clone().cross(alignVector);

    if (dirVector.length() === 0) {
      dirVector = eye;
    }

    this._plane.setFromNormalAndCoplanarPoint(dirVector,
this._worldPositionStart);
  }
}
```

## GizmoTranslationAxis

**GizmoTranslationAxis** – встроенная реализация оси переноса для GizmoControl.
Подробнее: GizmoAxis.

```
export class GizmoTranslationAxis extends GizmoAxis {
  constructor(axisDir: THREE.Vector3, readonly handle: IGizmoObject, readonly
picker?: IGizmoObject, readonly helper?: IGizmoObject) {
    super(axisDir, handle, picker, helper);
    this.name = 'GizmoTranslationAxis';
  }

  moveByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera): THREE.Matrix4 {
    this.updateGizmoPlane(camera);

    if (!this._startPoint)
      this.setStartPt(ndcPos, camera);

    this._raycaster.setFromCamera(ndcPos, camera);

    const planeIntersect = this._raycaster.ray.intersectPlane(this._plane,
new THREE.Vector3());
    if (!planeIntersect)
      return null;

    this._endPoint.copy(planeIntersect);

    const offset = this._endPoint.clone().sub(this._startPoint);

    const offsetValue = offset.dot(this._worldAxisDir);
    offset.copy(this._worldAxisDir).multiplyScalar(offsetValue);

    return new THREE.Matrix4().makeTranslation(offset.x, offset.y, offset.z);
  }

  protected override updateGizmoPlane(camera: THREE.Camera): void {
    this._raycaster.setFromCamera({ x: 0, y: 0 }, camera);
    const eye = this._raycaster.ray.direction;

    const alignVector = eye.clone().cross(this._worldAxisDir);

    let dirVector = this._worldAxisDir.clone().cross(alignVector);

    if (dirVector.length() === 0) {
      dirVector = eye;
    }

    dirVector.normalize();

    this._plane.setFromNormalAndCoplanarPoint(dirVector,
this._worldPositionStart);
  }
}
```

## IGizmoObject

**IGizmoObject** – интерфейс, описывающий поведение объектов гизмо.

```
export interface IGizmoObject extends THREE.Object3D {
  //Метод возвращает статус ховера над IGizmoObject.
  getHovered(): boolean;
  //Метод задает ховер для IGizmoObject.
  setHovered(value: boolean): void;
  //Метод возвращает статус активности манипуляций над IGizmoObject.
  getActive(): boolean;
  //Метод задает статус активности манипуляций над IGizmoObject.
  setActive(value: boolean): void;
  //Метод освобождает выделенные объектом ресурсы.
  dispose(): void;
}
```

## GizmoObject

**GizmoObject** – реализация интерфейса IGizmoObject. При изменении ховера, либо активности объекта, меняет материал геометрий.

```
export class GizmoObject extends THREE.Object3D implements IGizmoObject {
  constructor(protected _meshes: THREE.Mesh[],
    readonly baseMaterial: THREE.Material,
    readonly hoverMaterial: THREE.Material,
    readonly activeMaterial: THREE.Material
  );
  getHovered(): boolean;
  setHovered(value: boolean): void;
  getActive(): boolean;
  setActive(value: boolean): void;
  dispose(): void;

  override raycast(raycaster: THREE.Raycaster, intersects:
THREE.Intersection<THREE.Object3D<THREE.Event>>[]): void;
}
```

### *Поля*

### _meshes: THREE.Mesh[]

Геометрии, используемые для рендера на сцене. Также используются при расчете пересечений с `GizmoObject`.

### baseMaterial: THREE.Material

Материал геометрий, применяемый в отсутствии ховера и при неактивном `GizmoObject`. По умолчанию: GizmoMaterials.**gizmoMaterial**.

```
baseMaterial: THREE.Material;
```

## hoverMaterial: THREE.Material

Материал геометрий, применяемый при ховере над `GizmoObject`. По умолчанию: GizmoMaterials.**matYellowTransparent**.

```
hoverMaterial: THREE.Material;
```

## activeMaterial: THREE.Material

Материал геометрий, применяемый при активной манипуляции над `GizmoObject`. По умолчанию: GizmoMaterials.**matYellow**.

```
activeMaterial: THREE.Material;
```

### *Конструктор*

```
constructor(protected _meshes: THREE.Mesh[],

  readonly baseMaterial: THREE.Material,
  readonly hoverMaterial: THREE.Material,
  readonly activeMaterial: THREE.Material
);
```

где:
`_meshes` – геометрии объекта для отрисовки на сцене.

`baseMaterial` – материал геометрий, применяемый в отсутствии ховера и при неактивном `GizmoObject`. Не обязательный параметр. Подробнее: THREE.Material.

`hoverMaterial` – материал геометрий, применяемый при ховере над `GizmoObject`. Не обязательный параметр. Подробнее: THREE.Material.

`activeMaterial` – материал геометрий, применяемый при активной манипуляции над `GizmoObject`. Не обязательный параметр. Подробнее: THREE.Material.

### *Методы*

## getHovered()

Метод возвращает статус ховера для `GizmoObject`.

```
getHovered(): boolean;
```

Возвращает `true`, если над объектом находится курсор.

## setHovered()

Метод устанавливает значение ховера для `GizmoObject`.

```
setHovered(value: boolean): void;
```

где:
`value` - значение ховера.
Если `value` равно `true`, то материал геометрий изменится на `hoverMaterial`, если `GizmoObject` не активен. В противном случае, материалом геометрий будет или `baseMaterial`, или `activeMaterial`, в зависимости от активности `GizmoObject`.

## getActive()

Метод возвращает статус активности для `GizmoObject`.

```
getActive(): boolean;
```

Возвращает `true`, если объект активен.

## setActive()

Метод устанавливает статус активности для `GizmoObject`.

```
setActive(value: boolean): void;
```

где:
`value` - значение активности.
Если `value` равно `true`, то материал геометрий изменится на `activeMaterial`. В противном случае, материалом геометрий будет или `baseMaterial`, или `hoverMaterial`, в зависимости от значения ховера для `GizmoObject`. Активность `GizmoObject` имеет больший приоритет в установке материала геометрий, чем ховер.

## raycast

Переопределённый метод Object3D.raycast. Проверяет пересечения для всех геометрий `GizmoObject`.

```
override raycast(raycaster: THREE.Raycaster, intersects:
THREE.Intersection<THREE.Object3D<THREE.Event>>[]): void
```

## GuiViewer3D

**GuiViewer3D** – это класс для компонента просмотра 3D-моделей. Он расширяет возможности базового класса Viewer3D и содержит всё, что нужно для отображения и взаимодействия с 3D-моделями, полученными из системы **Pilot-BIM**.

### *Методы*

### getToolbar()

Метод получает объект для работы с панелью инструментов.

```
getToolbar(): ViewerToolbar;
```

## IfcType

**IfcType** – перечисление для типов IFC - элементов.

```
export enum IfcType {
    IfcAbsorbedDoseMeasure = 0,
    IfcAccelerationMeasure = 1,
    IfcActionRequest = 2,
    IfcActionRequestTypeEnum = 3,
    IfcActionSourceTypeEnum = 4,
    IfcActionTypeEnum = 5,
    IfcActor = 6,
    IfcActorRole = 7,
    IfcActorSelect = 8,
    IfcActuator = 9,
    IfcActuatorType = 10,
    IfcActuatorTypeEnum = 11,
    IfcAddress = 12,
    IfcAddressTypeEnum = 13,
    IfcAdvancedBrep = 14,
    IfcAdvancedBrepWithVoids = 15,
    IfcAdvancedFace = 16,
    IfcAirTerminal = 17,
    IfcAirTerminalBox = 18,
    IfcAirTerminalBoxType = 19,
    IfcAirTerminalBoxTypeEnum = 20,
    IfcAirTerminalType = 21,
    IfcAirTerminalTypeEnum = 22,
    IfcAirToAirHeatRecovery = 23,
    IfcAirToAirHeatRecoveryType = 24,
    IfcAirToAirHeatRecoveryTypeEnum = 25,
    IfcAlarm = 26,
    IfcAlarmType = 27,
    IfcAlarmTypeEnum = 28,
    IfcAlignment = 29,
    IfcAlignment2DHorizontal = 30,
    IfcAlignment2DHorizontalSegment = 31,
    IfcAlignment2DSegment = 32,
    IfcAlignment2DVerSegCircularArc = 33,
    IfcAlignment2DVerSegLine = 34,
    IfcAlignment2DVerSegParabolicArc = 35,
    IfcAlignment2DVertical = 36,
    IfcAlignment2DVerticalSegment = 37,
    IfcAlignmentCurve = 38,
    IfcAlignmentTypeEnum = 39,
    IfcAmountOfSubstanceMeasure = 40,
    IfcAnalysisModelTypeEnum = 41,
    IfcAnalysisTheoryTypeEnum = 42,
    IfcAngularVelocityMeasure = 43,
    IfcAnnotation = 44,
    IfcAnnotationFillArea = 45,
    IfcApplication = 46,
    IfcAppliedValue = 47,
    IfcAppliedValueSelect = 48,
    IfcApproval = 49,
    IfcApprovalRelationship = 50,
    IfcArbitraryClosedProfileDef = 51,
    IfcArbitraryOpenProfileDef = 52,
    IfcArbitraryProfileDefWithVoids = 53,
```

```
IfcArcIndex = 54,
IfcAreaDensityMeasure = 55,
IfcAreaMeasure = 56,
IfcArithmeticOperatorEnum = 57,
IfcAssemblyPlaceEnum = 58,
IfcAsset = 59,
IfcAsymmetricIShapeProfileDef = 60,
IfcAudioVisualAppliance = 61,
IfcAudioVisualApplianceType = 62,
IfcAudioVisualApplianceTypeEnum = 63,
IfcAxis1Placement = 64,
IfcAxis2Placement = 65,
IfcAxis2Placement2D = 66,
IfcAxis2Placement3D = 67,
IfcBeam = 68,
IfcBeamStandardCase = 69,
IfcBeamType = 70,
IfcBeamTypeEnum = 71,
IfcBenchmarkEnum = 72,
IfcBendingParameterSelect = 73,
IfcBinary = 74,
IfcBlobTexture = 75,
IfcBlock = 76,
IfcBoiler = 77,
IfcBoilerType = 78,
IfcBoilerTypeEnum = 79,
IfcBoolean = 80,
IfcBooleanClippingResult = 81,
IfcBooleanOperand = 82,
IfcBooleanOperator = 83,
IfcBooleanResult = 84,
IfcBoundaryCondition = 85,
IfcBoundaryCurve = 86,
IfcBoundaryEdgeCondition = 87,
IfcBoundaryFaceCondition = 88,
IfcBoundaryNodeCondition = 89,
IfcBoundaryNodeConditionWarping = 90,
IfcBoundedCurve = 91,
IfcBoundedSurface = 92,
IfcBoundingBox = 93,
IfcBoxAlignment = 94,
IfcBoxedHalfSpace = 95,
IfcBSplineCurve = 96,
IfcBSplineCurveForm = 97,
IfcBSplineCurveWithKnots = 98,
IfcBSplineSurface = 99,
IfcBSplineSurfaceForm = 100,
IfcBSplineSurfaceWithKnots = 101,
IfcBuilding = 102,
IfcBuildingElement = 103,
IfcBuildingElementPart = 104,
IfcBuildingElementPartType = 105,
IfcBuildingElementPartTypeEnum = 106,
IfcBuildingElementProxy = 107,
IfcBuildingElementProxyType = 108,
IfcBuildingElementProxyTypeEnum = 109,
IfcBuildingElementType = 110,
IfcBuildingStorey = 111,
IfcBuildingSystem = 112,
IfcBuildingSystemTypeEnum = 113,
```

```
IfcBurner = 114,
IfcBurnerType = 115,
IfcBurnerTypeEnum = 116,
IfcCableCarrierFitting = 117,
IfcCableCarrierFittingType = 118,
IfcCableCarrierFittingTypeEnum = 119,
IfcCableCarrierSegment = 120,
IfcCableCarrierSegmentType = 121,
IfcCableCarrierSegmentTypeEnum = 122,
IfcCableFitting = 123,
IfcCableFittingType = 124,
IfcCableFittingTypeEnum = 125,
IfcCableSegment = 126,
IfcCableSegmentType = 127,
IfcCableSegmentTypeEnum = 128,
IfcCardinalPointReference = 129,
IfcCartesianPoint = 130,
IfcCartesianPointList = 131,
IfcCartesianPointList2D = 132,
IfcCartesianPointList3D = 133,
IfcCartesianTransformationOperator = 134,
IfcCartesianTransformationOperator2D = 135,
IfcCartesianTransformationOperator2DnonUniform = 136,
IfcCartesianTransformationOperator3D = 137,
IfcCartesianTransformationOperator3DnonUniform = 138,
IfcCenterLineProfileDef = 139,
IfcChangeActionEnum = 140,
IfcChiller = 141,
IfcChillerType = 142,
IfcChillerTypeEnum = 143,
IfcChimney = 144,
IfcChimneyType = 145,
IfcChimneyTypeEnum = 146,
IfcCircle = 147,
IfcCircleHollowProfileDef = 148,
IfcCircleProfileDef = 149,
IfcCircularArcSegment2D = 150,
IfcCivilElement = 151,
IfcCivilElementType = 152,
IfcClassification = 153,
IfcClassificationReference = 154,
IfcClassificationReferenceSelect = 155,
IfcClassificationSelect = 156,
IfcClosedShell = 157,
IfcCoil = 158,
IfcCoilType = 159,
IfcCoilTypeEnum = 160,
IfcColour = 161,
IfcColourOrFactor = 162,
IfcColourRgb = 163,
IfcColourRgbList = 164,
IfcColourSpecification = 165,
IfcColumn = 166,
IfcColumnStandardCase = 167,
IfcColumnType = 168,
IfcColumnTypeEnum = 169,
IfcCommunicationsAppliance = 170,
IfcCommunicationsApplianceType = 171,
IfcCommunicationsApplianceTypeEnum = 172,
IfcComplexNumber = 173,
```

```
IfcComplexProperty = 174,
IfcComplexPropertyTemplate = 175,
IfcComplexPropertyTemplateTypeEnum = 176,
IfcCompositeCurve = 177,
IfcCompositeCurveOnSurface = 178,
IfcCompositeCurveSegment = 179,
IfcCompositeProfileDef = 180,
IfcCompoundPlaneAngleMeasure = 181,
IfcCompressor = 182,
IfcCompressorType = 183,
IfcCompressorTypeEnum = 184,
IfcCondenser = 185,
IfcCondenserType = 186,
IfcCondenserTypeEnum = 187,
IfcConic = 188,
IfcConnectedFaceSet = 189,
IfcConnectionCurveGeometry = 190,
IfcConnectionGeometry = 191,
IfcConnectionPointEccentricity = 192,
IfcConnectionPointGeometry = 193,
IfcConnectionSurfaceGeometry = 194,
IfcConnectionTypeEnum = 195,
IfcConnectionVolumeGeometry = 196,
IfcConstraint = 197,
IfcConstraintEnum = 198,
IfcConstructionEquipmentResource = 199,
IfcConstructionEquipmentResourceType = 200,
IfcConstructionEquipmentResourceTypeEnum = 201,
IfcConstructionMaterialResource = 202,
IfcConstructionMaterialResourceType = 203,
IfcConstructionMaterialResourceTypeEnum = 204,
IfcConstructionProductResource = 205,
IfcConstructionProductResourceType = 206,
IfcConstructionProductResourceTypeEnum = 207,
IfcConstructionResource = 208,
IfcConstructionResourceType = 209,
IfcContext = 210,
IfcContextDependentMeasure = 211,
IfcContextDependentUnit = 212,
IfcControl = 213,
IfcController = 214,
IfcControllerType = 215,
IfcControllerTypeEnum = 216,
IfcConversionBasedUnit = 217,
IfcConversionBasedUnitWithOffset = 218,
IfcCooledBeam = 219,
IfcCooledBeamType = 220,
IfcCooledBeamTypeEnum = 221,
IfcCoolingTower = 222,
IfcCoolingTowerType = 223,
IfcCoolingTowerTypeEnum = 224,
IfcCoordinateOperation = 225,
IfcCoordinateReferenceSystem = 226,
IfcCoordinateReferenceSystemSelect = 227,
IfcCostItem = 228,
IfcCostItemTypeEnum = 229,
IfcCostSchedule = 230,
IfcCostScheduleTypeEnum = 231,
IfcCostValue = 232,
IfcCountMeasure = 233,
```

```
IfcCovering = 234,
IfcCoveringType = 235,
IfcCoveringTypeEnum = 236,
IfcCrewResource = 237,
IfcCrewResourceType = 238,
IfcCrewResourceTypeEnum = 239,
IfcCsgPrimitive3D = 240,
IfcCsgSelect = 241,
IfcCsgSolid = 242,
IfcCShapeProfileDef = 243,
IfcCurrencyRelationship = 244,
IfcCurtainWall = 245,
IfcCurtainWallType = 246,
IfcCurtainWallTypeEnum = 247,
IfcCurvatureMeasure = 248,
IfcCurve = 249,
IfcCurveBoundedPlane = 250,
IfcCurveBoundedSurface = 251,
IfcCurveFontOrScaledCurveFontSelect = 252,
IfcCurveInterpolationEnum = 253,
IfcCurveOnSurface = 254,
IfcCurveOrEdgeCurve = 255,
IfcCurveSegment2D = 256,
IfcCurveStyle = 257,
IfcCurveStyleFont = 258,
IfcCurveStyleFontAndScaling = 259,
IfcCurveStyleFontPattern = 260,
IfcCurveStyleFontSelect = 261,
IfcCylindricalSurface = 262,
IfcDamper = 263,
IfcDamperType = 264,
IfcDamperTypeEnum = 265,
IfcDataOriginEnum = 266,
IfcDate = 267,
IfcDateTime = 268,
IfcDayInMonthNumber = 269,
IfcDayInWeekNumber = 270,
IfcDefinitionSelect = 271,
IfcDerivedMeasureValue = 272,
IfcDerivedProfileDef = 273,
IfcDerivedUnit = 274,
IfcDerivedUnitElement = 275,
IfcDerivedUnitEnum = 276,
IfcDescriptiveMeasure = 277,
IfcDimensionalExponents = 278,
IfcDimensionCount = 279,
IfcDirection = 280,
IfcDirectionSenseEnum = 281,
IfcDiscreteAccessory = 282,
IfcDiscreteAccessoryType = 283,
IfcDiscreteAccessoryTypeEnum = 284,
IfcDistanceExpression = 285,
IfcDistributionChamberElement = 286,
IfcDistributionChamberElementType = 287,
IfcDistributionChamberElementTypeEnum = 288,
IfcDistributionCircuit = 289,
IfcDistributionControlElement = 290,
IfcDistributionControlElementType = 291,
IfcDistributionElement = 292,
IfcDistributionElementType = 293,
```

```
    IfcDistributionFlowElement = 294,
    IfcDistributionFlowElementType = 295,
    IfcDistributionPort = 296,
    IfcDistributionPortTypeEnum = 297,
    IfcDistributionSystem = 298,
    IfcDistributionSystemEnum = 299,
    IfcDocumentConfidentialityEnum = 300,
    IfcDocumentInformation = 301,
    IfcDocumentInformationRelationship = 302,
    IfcDocumentReference = 303,
    IfcDocumentSelect = 304,
    IfcDocumentStatusEnum = 305,
    IfcDoor = 306,
    IfcDoorLiningProperties = 307,
    IfcDoorPanelOperationEnum = 308,
    IfcDoorPanelPositionEnum = 309,
    IfcDoorPanelProperties = 310,
    IfcDoorStandardCase = 311,
    IfcDoorStyle = 312,
    IfcDoorStyleConstructionEnum = 313,
    IfcDoorStyleOperationEnum = 314,
    IfcDoorType = 315,
    IfcDoorTypeEnum = 316,
    IfcDoorTypeOperationEnum = 317,
    IfcDoseEquivalentMeasure = 318,
    IfcDraughtingPreDefinedColour = 319,
    IfcDraughtingPreDefinedCurveFont = 320,
    IfcDuctFitting = 321,
    IfcDuctFittingType = 322,
    IfcDuctFittingTypeEnum = 323,
    IfcDuctSegment = 324,
    IfcDuctSegmentType = 325,
    IfcDuctSegmentTypeEnum = 326,
    IfcDuctSilencer = 327,
    IfcDuctSilencerType = 328,
    IfcDuctSilencerTypeEnum = 329,
    IfcDuration = 330,
    IfcDynamicViscosityMeasure = 331,
    IfcEdge = 332,
    IfcEdgeCurve = 333,
    IfcEdgeLoop = 334,
    IfcElectricAppliance = 335,
    IfcElectricApplianceType = 336,
    IfcElectricApplianceTypeEnum = 337,
    IfcElectricCapacitanceMeasure = 338,
    IfcElectricChargeMeasure = 339,
    IfcElectricConductanceMeasure = 340,
    IfcElectricCurrentMeasure = 341,
    IfcElectricDistributionBoard = 342,
    IfcElectricDistributionBoardType = 343,
    IfcElectricDistributionBoardTypeEnum = 344,
    IfcElectricFlowStorageDevice = 345,
    IfcElectricFlowStorageDeviceType = 346,
    IfcElectricFlowStorageDeviceTypeEnum = 347,
    IfcElectricGenerator = 348,
    IfcElectricGeneratorType = 349,
    IfcElectricGeneratorTypeEnum = 350,
    IfcElectricMotor = 351,
    IfcElectricMotorType = 352,
    IfcElectricMotorTypeEnum = 353,
```

```
IfcElectricResistanceMeasure = 354,
IfcElectricTimeControl = 355,
IfcElectricTimeControlType = 356,
IfcElectricTimeControlTypeEnum = 357,
IfcElectricVoltageMeasure = 358,
IfcElement = 359,
IfcElementarySurface = 360,
IfcElementAssembly = 361,
IfcElementAssemblyType = 362,
IfcElementAssemblyTypeEnum = 363,
IfcElementComponent = 364,
IfcElementComponentType = 365,
IfcElementCompositionEnum = 366,
IfcElementQuantity = 367,
IfcElementType = 368,
IfcEllipse = 369,
IfcEllipseProfileDef = 370,
IfcEnergyConversionDevice = 371,
IfcEnergyConversionDeviceType = 372,
IfcEnergyMeasure = 373,
IfcEngine = 374,
IfcEngineType = 375,
IfcEngineTypeEnum = 376,
IfcEvaporativeCooler = 377,
IfcEvaporativeCoolerType = 378,
IfcEvaporativeCoolerTypeEnum = 379,
IfcEvaporator = 380,
IfcEvaporatorType = 381,
IfcEvaporatorTypeEnum = 382,
IfcEvent = 383,
IfcEventTime = 384,
IfcEventTriggerTypeEnum = 385,
IfcEventType = 386,
IfcEventTypeEnum = 387,
IfcExtendedProperties = 388,
IfcExternalInformation = 389,
IfcExternallyDefinedHatchStyle = 390,
IfcExternallyDefinedSurfaceStyle = 391,
IfcExternallyDefinedTextFont = 392,
IfcExternalReference = 393,
IfcExternalReferenceRelationship = 394,
IfcExternalSpatialElement = 395,
IfcExternalSpatialElementTypeEnum = 396,
IfcExternalSpatialStructureElement = 397,
IfcExtrudedAreaSolid = 398,
IfcExtrudedAreaSolidTapered = 399,
IfcFace = 400,
IfcFaceBasedSurfaceModel = 401,
IfcFaceBound = 402,
IfcFaceOuterBound = 403,
IfcFaceSurface = 404,
IfcFacetedBrep = 405,
IfcFacetedBrepWithVoids = 406,
IfcFailureConnectionCondition = 407,
IfcFan = 408,
IfcFanType = 409,
IfcFanTypeEnum = 410,
IfcFastener = 411,
IfcFastenerType = 412,
IfcFastenerTypeEnum = 413,
```

```
IfcFeatureElement = 414,
IfcFeatureElementAddition = 415,
IfcFeatureElementSubtraction = 416,
IfcFillAreaStyle = 417,
IfcFillAreaStyleHatching = 418,
IfcFillAreaStyleTiles = 419,
IfcFillStyleSelect = 420,
IfcFilter = 421,
IfcFilterType = 422,
IfcFilterTypeEnum = 423,
IfcFireSuppressionTerminal = 424,
IfcFireSuppressionTerminalType = 425,
IfcFireSuppressionTerminalTypeEnum = 426,
IfcFixedReferenceSweptAreaSolid = 427,
IfcFlowController = 428,
IfcFlowControllerType = 429,
IfcFlowDirectionEnum = 430,
IfcFlowFitting = 431,
IfcFlowFittingType = 432,
IfcFlowInstrument = 433,
IfcFlowInstrumentType = 434,
IfcFlowInstrumentTypeEnum = 435,
IfcFlowMeter = 436,
IfcFlowMeterType = 437,
IfcFlowMeterTypeEnum = 438,
IfcFlowMovingDevice = 439,
IfcFlowMovingDeviceType = 440,
IfcFlowSegment = 441,
IfcFlowSegmentType = 442,
IfcFlowStorageDevice = 443,
IfcFlowStorageDeviceType = 444,
IfcFlowTerminal = 445,
IfcFlowTerminalType = 446,
IfcFlowTreatmentDevice = 447,
IfcFlowTreatmentDeviceType = 448,
IfcFontStyle = 449,
IfcFontVariant = 450,
IfcFontWeight = 451,
IfcFooting = 452,
IfcFootingType = 453,
IfcFootingTypeEnum = 454,
IfcForceMeasure = 455,
IfcFrequencyMeasure = 456,
IfcFurnishingElement = 457,
IfcFurnishingElementType = 458,
IfcFurniture = 459,
IfcFurnitureType = 460,
IfcFurnitureTypeEnum = 461,
IfcGeographicElement = 462,
IfcGeographicElementType = 463,
IfcGeographicElementTypeEnum = 464,
IfcGeometricCurveSet = 465,
IfcGeometricProjectionEnum = 466,
IfcGeometricRepresentationContext = 467,
IfcGeometricRepresentationItem = 468,
IfcGeometricRepresentationSubContext = 469,
IfcGeometricSet = 470,
IfcGeometricSetSelect = 471,
IfcGloballyUniqueId = 472,
IfcGlobalOrLocalEnum = 473,
```

```
IfcGrid = 474,
IfcGridAxis = 475,
IfcGridPlacement = 476,
IfcGridPlacementDirectionSelect = 477,
IfcGridTypeEnum = 478,
IfcGroup = 479,
IfcHalfSpaceSolid = 480,
IfcHatchLineDistanceSelect = 481,
IfcHeatExchanger = 482,
IfcHeatExchangerType = 483,
IfcHeatExchangerTypeEnum = 484,
IfcHeatFluxDensityMeasure = 485,
IfcHeatingValueMeasure = 486,
IfcHumidifier = 487,
IfcHumidifierType = 488,
IfcHumidifierTypeEnum = 489,
IfcIdentifier = 490,
IfcIlluminanceMeasure = 491,
IfcImageTexture = 492,
IfcIndexedColourMap = 493,
IfcIndexedPolyCurve = 494,
IfcIndexedPolygonalFace = 495,
IfcIndexedPolygonalFaceWithVoids = 496,
IfcIndexedTextureMap = 497,
IfcIndexedTriangleTextureMap = 498,
IfcInductanceMeasure = 499,
IfcInteger = 500,
IfcIntegerCountRateMeasure = 501,
IfcInterceptor = 502,
IfcInterceptorType = 503,
IfcInterceptorTypeEnum = 504,
IfcInternalOrExternalEnum = 505,
IfcIntersectionCurve = 506,
IfcInventory = 507,
IfcInventoryTypeEnum = 508,
IfcIonConcentrationMeasure = 509,
IfcIrregularTimeSeries = 510,
IfcIrregularTimeSeriesValue = 511,
IfcIShapeProfileDef = 512,
IfcIsothermalMoistureCapacityMeasure = 513,
IfcJunctionBox = 514,
IfcJunctionBoxType = 515,
IfcJunctionBoxTypeEnum = 516,
IfcKinematicViscosityMeasure = 517,
IfcKnotType = 518,
IfcLabel = 519,
IfcLaborResource = 520,
IfcLaborResourceType = 521,
IfcLaborResourceTypeEnum = 522,
IfcLagTime = 523,
IfcLamp = 524,
IfcLampType = 525,
IfcLampTypeEnum = 526,
IfcLanguageId = 527,
IfcLayeredItem = 528,
IfcLayerSetDirectionEnum = 529,
IfcLengthMeasure = 530,
IfcLibraryInformation = 531,
IfcLibraryReference = 532,
IfcLibrarySelect = 533,
```

```
IfcLightDistributionCurveEnum = 534,
IfcLightDistributionData = 535,
IfcLightDistributionDataSourceSelect = 536,
IfcLightEmissionSourceEnum = 537,
IfcLightFixture = 538,
IfcLightFixtureType = 539,
IfcLightFixtureTypeEnum = 540,
IfcLightIntensityDistribution = 541,
IfcLightSource = 542,
IfcLightSourceAmbient = 543,
IfcLightSourceDirectional = 544,
IfcLightSourceGoniometric = 545,
IfcLightSourcePositional = 546,
IfcLightSourceSpot = 547,
IfcLine = 548,
IfcLinearForceMeasure = 549,
IfcLinearMomentMeasure = 550,
IfcLinearPlacement = 551,
IfcLinearPositioningElement = 552,
IfcLinearStiffnessMeasure = 553,
IfcLinearVelocityMeasure = 554,
IfcLineIndex = 555,
IfcLineSegment2D = 556,
IfcLoadGroupTypeEnum = 557,
IfcLocalPlacement = 558,
IfcLogical = 559,
IfcLogicalOperatorEnum = 560,
IfcLoop = 561,
IfcLShapeProfileDef = 562,
IfcLuminousFluxMeasure = 563,
IfcLuminousIntensityDistributionMeasure = 564,
IfcLuminousIntensityMeasure = 565,
IfcMagneticFluxDensityMeasure = 566,
IfcMagneticFluxMeasure = 567,
IfcManifoldSolidBrep = 568,
IfcMapConversion = 569,
IfcMappedItem = 570,
IfcMassDensityMeasure = 571,
IfcMassFlowRateMeasure = 572,
IfcMassMeasure = 573,
IfcMassPerLengthMeasure = 574,
IfcMaterial = 575,
IfcMaterialClassificationRelationship = 576,
IfcMaterialConstituent = 577,
IfcMaterialConstituentSet = 578,
IfcMaterialDefinition = 579,
IfcMaterialDefinitionRepresentation = 580,
IfcMaterialLayer = 581,
IfcMaterialLayerSet = 582,
IfcMaterialLayerSetUsage = 583,
IfcMaterialLayerWithOffsets = 584,
IfcMaterialList = 585,
IfcMaterialProfile = 586,
IfcMaterialProfileSet = 587,
IfcMaterialProfileSetUsage = 588,
IfcMaterialProfileSetUsageTapering = 589,
IfcMaterialProfileWithOffsets = 590,
IfcMaterialProperties = 591,
IfcMaterialRelationship = 592,
IfcMaterialSelect = 593,
```

```
IfcMaterialUsageDefinition = 594,
IfcMeasureValue = 595,
IfcMeasureWithUnit = 596,
IfcMechanicalFastener = 597,
IfcMechanicalFastenerType = 598,
IfcMechanicalFastenerTypeEnum = 599,
IfcMedicalDevice = 600,
IfcMedicalDeviceType = 601,
IfcMedicalDeviceTypeEnum = 602,
IfcMember = 603,
IfcMemberStandardCase = 604,
IfcMemberType = 605,
IfcMemberTypeEnum = 606,
IfcMetric = 607,
IfcMetricValueSelect = 608,
IfcMirroredProfileDef = 609,
IfcModulusOfElasticityMeasure = 610,
IfcModulusOfLinearSubgradeReactionMeasure = 611,
IfcModulusOfRotationalSubgradeReactionMeasure = 612,
IfcModulusOfRotationalSubgradeReactionSelect = 613,
IfcModulusOfSubgradeReactionMeasure = 614,
IfcModulusOfSubgradeReactionSelect = 615,
IfcModulusOfTranslationalSubgradeReactionSelect = 616,
IfcMoistureDiffusivityMeasure = 617,
IfcMolecularWeightMeasure = 618,
IfcMomentOfInertiaMeasure = 619,
IfcMonetaryMeasure = 620,
IfcMonetaryUnit = 621,
IfcMonthInYearNumber = 622,
IfcMotorConnection = 623,
IfcMotorConnectionType = 624,
IfcMotorConnectionTypeEnum = 625,
IfcNamedUnit = 626,
IfcNonNegativeLengthMeasure = 627,
IfcNormalisedRatioMeasure = 628,
IfcNullStyle = 629,
IfcNumericMeasure = 630,
IfcObject = 631,
IfcObjectDefinition = 632,
IfcObjective = 633,
IfcObjectiveEnum = 634,
IfcObjectPlacement = 635,
IfcObjectReferenceSelect = 636,
IfcObjectTypeEnum = 637,
IfcOccupant = 638,
IfcOccupantTypeEnum = 639,
IfcOffsetCurve = 640,
IfcOffsetCurve2D = 641,
IfcOffsetCurve3D = 642,
IfcOffsetCurveByDistances = 643,
IfcOpeningElement = 644,
IfcOpeningElementTypeEnum = 645,
IfcOpeningStandardCase = 646,
IfcOpenShell = 647,
IfcOrganization = 648,
IfcOrganizationRelationship = 649,
IfcOrientationExpression = 650,
IfcOrientedEdge = 651,
IfcOuterBoundaryCurve = 652,
IfcOutlet = 653,
```

```
    IfcOutletType = 654,
    IfcOutletTypeEnum = 655,
    IfcOwnerHistory = 656,
    IfcParameterizedProfileDef = 657,
    IfcParameterValue = 658,
    IfcPath = 659,
    IfcPcurve = 660,
    IfcPerformanceHistory = 661,
    IfcPerformanceHistoryTypeEnum = 662,
    IfcPermeableCoveringOperationEnum = 663,
    IfcPermeableCoveringProperties = 664,
    IfcPermit = 665,
    IfcPermitTypeEnum = 666,
    IfcPerson = 667,
    IfcPersonAndOrganization = 668,
    IfcPHMeasure = 669,
    IfcPhysicalComplexQuantity = 670,
    IfcPhysicalOrVirtualEnum = 671,
    IfcPhysicalQuantity = 672,
    IfcPhysicalSimpleQuantity = 673,
    IfcPile = 674,
    IfcPileConstructionEnum = 675,
    IfcPileType = 676,
    IfcPileTypeEnum = 677,
    IfcPipeFitting = 678,
    IfcPipeFittingType = 679,
    IfcPipeFittingTypeEnum = 680,
    IfcPipeSegment = 681,
    IfcPipeSegmentType = 682,
    IfcPipeSegmentTypeEnum = 683,
    IfcPixelTexture = 684,
    IfcPlacement = 685,
    IfcPlanarBox = 686,
    IfcPlanarExtent = 687,
    IfcPlanarForceMeasure = 688,
    IfcPlane = 689,
    IfcPlaneAngleMeasure = 690,
    IfcPlate = 691,
    IfcPlateStandardCase = 692,
    IfcPlateType = 693,
    IfcPlateTypeEnum = 694,
    IfcPoint = 695,
    IfcPointOnCurve = 696,
    IfcPointOnSurface = 697,
    IfcPointOrVertexPoint = 698,
    IfcPolygonalBoundedHalfSpace = 699,
    IfcPolygonalFaceSet = 700,
    IfcPolyline = 701,
    IfcPolyLoop = 702,
    IfcPort = 703,
    IfcPositioningElement = 704,
    IfcPositiveInteger = 705,
    IfcPositiveLengthMeasure = 706,
    IfcPositivePlaneAngleMeasure = 707,
    IfcPositiveRatioMeasure = 708,
    IfcPostalAddress = 709,
    IfcPowerMeasure = 710,
    IfcPreDefinedColour = 711,
    IfcPreDefinedCurveFont = 712,
    IfcPreDefinedItem = 713,
```

```
IfcPreDefinedProperties = 714,
IfcPreDefinedPropertySet = 715,
IfcPreDefinedTextFont = 716,
IfcPreferredSurfaceCurveRepresentation = 717,
IfcPresentableText = 718,
IfcPresentationItem = 719,
IfcPresentationLayerAssignment = 720,
IfcPresentationLayerWithStyle = 721,
IfcPresentationStyle = 722,
IfcPresentationStyleAssignment = 723,
IfcPresentationStyleSelect = 724,
IfcPressureMeasure = 725,
IfcProcedure = 726,
IfcProcedureType = 727,
IfcProcedureTypeEnum = 728,
IfcProcess = 729,
IfcProcessSelect = 730,
IfcProduct = 731,
IfcProductDefinitionShape = 732,
IfcProductRepresentation = 733,
IfcProductRepresentationSelect = 734,
IfcProductSelect = 735,
IfcProfileDef = 736,
IfcProfileProperties = 737,
IfcProfileTypeEnum = 738,
IfcProject = 739,
IfcProjectedCRS = 740,
IfcProjectedOrTrueLengthEnum = 741,
IfcProjectionElement = 742,
IfcProjectionElementTypeEnum = 743,
IfcProjectLibrary = 744,
IfcProjectOrder = 745,
IfcProjectOrderTypeEnum = 746,
IfcProperty = 747,
IfcPropertyAbstraction = 748,
IfcPropertyBoundedValue = 749,
IfcPropertyDefinition = 750,
IfcPropertyDependencyRelationship = 751,
IfcPropertyEnumeratedValue = 752,
IfcPropertyEnumeration = 753,
IfcPropertyListValue = 754,
IfcPropertyReferenceValue = 755,
IfcPropertySet = 756,
IfcPropertySetDefinition = 757,
IfcPropertySetDefinitionSelect = 758,
IfcPropertySetDefinitionSet = 759,
IfcPropertySetTemplate = 760,
IfcPropertySetTemplateTypeEnum = 761,
IfcPropertySingleValue = 762,
IfcPropertyTableValue = 763,
IfcPropertyTemplate = 764,
IfcPropertyTemplateDefinition = 765,
IfcProtectiveDevice = 766,
IfcProtectiveDeviceTrippingUnit = 767,
IfcProtectiveDeviceTrippingUnitType = 768,
IfcProtectiveDeviceTrippingUnitTypeEnum = 769,
IfcProtectiveDeviceType = 770,
IfcProtectiveDeviceTypeEnum = 771,
IfcProxy = 772,
IfcPump = 773,
```

```
IfcPumpType = 774,
IfcPumpTypeEnum = 775,
IfcQuantityArea = 776,
IfcQuantityCount = 777,
IfcQuantityLength = 778,
IfcQuantitySet = 779,
IfcQuantityTime = 780,
IfcQuantityVolume = 781,
IfcQuantityWeight = 782,
IfcRadioActivityMeasure = 783,
IfcRailing = 784,
IfcRailingType = 785,
IfcRailingTypeEnum = 786,
IfcRamp = 787,
IfcRampFlight = 788,
IfcRampFlightType = 789,
IfcRampFlightTypeEnum = 790,
IfcRampType = 791,
IfcRampTypeEnum = 792,
IfcRatioMeasure = 793,
IfcRationalBSplineCurveWithKnots = 794,
IfcRationalBSplineSurfaceWithKnots = 795,
IfcReal = 796,
IfcRectangleHollowProfileDef = 797,
IfcRectangleProfileDef = 798,
IfcRectangularPyramid = 799,
IfcRectangularTrimmedSurface = 800,
IfcRecurrencePattern = 801,
IfcRecurrenceTypeEnum = 802,
IfcReference = 803,
IfcReferent = 804,
IfcReferentTypeEnum = 805,
IfcReflectanceMethodEnum = 806,
IfcRegularTimeSeries = 807,
IfcReinforcementBarProperties = 808,
IfcReinforcementDefinitionProperties = 809,
IfcReinforcingBar = 810,
IfcReinforcingBarRoleEnum = 811,
IfcReinforcingBarSurfaceEnum = 812,
IfcReinforcingBarType = 813,
IfcReinforcingBarTypeEnum = 814,
IfcReinforcingElement = 815,
IfcReinforcingElementType = 816,
IfcReinforcingMesh = 817,
IfcReinforcingMeshType = 818,
IfcReinforcingMeshTypeEnum = 819,
IfcRelAggregates = 820,
IfcRelAssigns = 821,
IfcRelAssignsToActor = 822,
IfcRelAssignsToControl = 823,
IfcRelAssignsToGroup = 824,
IfcRelAssignsToGroupByFactor = 825,
IfcRelAssignsToProcess = 826,
IfcRelAssignsToProduct = 827,
IfcRelAssignsToResource = 828,
IfcRelAssociates = 829,
IfcRelAssociatesApproval = 830,
IfcRelAssociatesClassification = 831,
IfcRelAssociatesConstraint = 832,
IfcRelAssociatesDocument = 833,
```

```
IfcRelAssociatesLibrary = 834,
IfcRelAssociatesMaterial = 835,
IfcRelationship = 836,
IfcRelConnects = 837,
IfcRelConnectsElements = 838,
IfcRelConnectsPathElements = 839,
IfcRelConnectsPorts = 840,
IfcRelConnectsPortToElement = 841,
IfcRelConnectsStructuralActivity = 842,
IfcRelConnectsStructuralMember = 843,
IfcRelConnectsWithEccentricity = 844,
IfcRelConnectsWithRealizingElements = 845,
IfcRelContainedInSpatialStructure = 846,
IfcRelCoversBldgElements = 847,
IfcRelCoversSpaces = 848,
IfcRelDeclares = 849,
IfcRelDecomposes = 850,
IfcRelDefines = 851,
IfcRelDefinesByObject = 852,
IfcRelDefinesByProperties = 853,
IfcRelDefinesByTemplate = 854,
IfcRelDefinesByType = 855,
IfcRelFillsElement = 856,
IfcRelFlowControlElements = 857,
IfcRelInterferesElements = 858,
IfcRelNests = 859,
IfcRelProjectsElement = 860,
IfcRelReferencedInSpatialStructure = 861,
IfcRelSequence = 862,
IfcRelServicesBuildings = 863,
IfcRelSpaceBoundary = 864,
IfcRelSpaceBoundary1stLevel = 865,
IfcRelSpaceBoundary2ndLevel = 866,
IfcRelVoidsElement = 867,
IfcReparametrisedCompositeCurveSegment = 868,
IfcRepresentation = 869,
IfcRepresentationContext = 870,
IfcRepresentationItem = 871,
IfcRepresentationMap = 872,
IfcResource = 873,
IfcResourceApprovalRelationship = 874,
IfcResourceConstraintRelationship = 875,
IfcResourceLevelRelationship = 876,
IfcResourceObjectSelect = 877,
IfcResourceSelect = 878,
IfcResourceTime = 879,
IfcRevolvedAreaSolid = 880,
IfcRevolvedAreaSolidTapered = 881,
IfcRightCircularCone = 882,
IfcRightCircularCylinder = 883,
IfcRoleEnum = 884,
IfcRoof = 885,
IfcRoofType = 886,
IfcRoofTypeEnum = 887,
IfcRoot = 888,
IfcRotationalFrequencyMeasure = 889,
IfcRotationalMassMeasure = 890,
IfcRotationalStiffnessMeasure = 891,
IfcRotationalStiffnessSelect = 892,
IfcRoundedRectangleProfileDef = 893,
```

```
IfcSanitaryTerminal = 894,
IfcSanitaryTerminalType = 895,
IfcSanitaryTerminalTypeEnum = 896,
IfcSchedulingTime = 897,
IfcSeamCurve = 898,
IfcSectionalAreaIntegralMeasure = 899,
IfcSectionedSolid = 900,
IfcSectionedSolidHorizontal = 901,
IfcSectionedSpine = 902,
IfcSectionModulusMeasure = 903,
IfcSectionProperties = 904,
IfcSectionReinforcementProperties = 905,
IfcSectionTypeEnum = 906,
IfcSegmentIndexSelect = 907,
IfcSensor = 908,
IfcSensorType = 909,
IfcSensorTypeEnum = 910,
IfcSequenceEnum = 911,
IfcShadingDevice = 912,
IfcShadingDeviceType = 913,
IfcShadingDeviceTypeEnum = 914,
IfcShapeAspect = 915,
IfcShapeModel = 916,
IfcShapeRepresentation = 917,
IfcShearModulusMeasure = 918,
IfcShell = 919,
IfcShellBasedSurfaceModel = 920,
IfcSimpleProperty = 921,
IfcSimplePropertyTemplate = 922,
IfcSimplePropertyTemplateTypeEnum = 923,
IfcSimpleValue = 924,
IfcSIPrefix = 925,
IfcSite = 926,
IfcSIUnit = 927,
IfcSIUnitName = 928,
IfcSizeSelect = 929,
IfcSlab = 930,
IfcSlabElementedCase = 931,
IfcSlabStandardCase = 932,
IfcSlabType = 933,
IfcSlabTypeEnum = 934,
IfcSlippageConnectionCondition = 935,
IfcSolarDevice = 936,
IfcSolarDeviceType = 937,
IfcSolarDeviceTypeEnum = 938,
IfcSolidAngleMeasure = 939,
IfcSolidModel = 940,
IfcSolidOrShell = 941,
IfcSoundPowerLevelMeasure = 942,
IfcSoundPowerMeasure = 943,
IfcSoundPressureLevelMeasure = 944,
IfcSoundPressureMeasure = 945,
IfcSpace = 946,
IfcSpaceBoundarySelect = 947,
IfcSpaceHeater = 948,
IfcSpaceHeaterType = 949,
IfcSpaceHeaterTypeEnum = 950,
IfcSpaceType = 951,
IfcSpaceTypeEnum = 952,
IfcSpatialElement = 953,
```

```
IfcSpatialElementType = 954,
IfcSpatialStructureElement = 955,
IfcSpatialStructureElementType = 956,
IfcSpatialZone = 957,
IfcSpatialZoneType = 958,
IfcSpatialZoneTypeEnum = 959,
IfcSpecificHeatCapacityMeasure = 960,
IfcSpecularExponent = 961,
IfcSpecularHighlightSelect = 962,
IfcSpecularRoughness = 963,
IfcSphere = 964,
IfcSphericalSurface = 965,
IfcStackTerminal = 966,
IfcStackTerminalType = 967,
IfcStackTerminalTypeEnum = 968,
IfcStair = 969,
IfcStairFlight = 970,
IfcStairFlightType = 971,
IfcStairFlightTypeEnum = 972,
IfcStairType = 973,
IfcStairTypeEnum = 974,
IfcStateEnum = 975,
IfcStrippedOptional = 976,
IfcStructuralAction = 977,
IfcStructuralActivity = 978,
IfcStructuralActivityAssignmentSelect = 979,
IfcStructuralAnalysisModel = 980,
IfcStructuralConnection = 981,
IfcStructuralConnectionCondition = 982,
IfcStructuralCurveAction = 983,
IfcStructuralCurveActivityTypeEnum = 984,
IfcStructuralCurveConnection = 985,
IfcStructuralCurveMember = 986,
IfcStructuralCurveMemberTypeEnum = 987,
IfcStructuralCurveMemberVarying = 988,
IfcStructuralCurveReaction = 989,
IfcStructuralItem = 990,
IfcStructuralLinearAction = 991,
IfcStructuralLoad = 992,
IfcStructuralLoadCase = 993,
IfcStructuralLoadConfiguration = 994,
IfcStructuralLoadGroup = 995,
IfcStructuralLoadLinearForce = 996,
IfcStructuralLoadOrResult = 997,
IfcStructuralLoadPlanarForce = 998,
IfcStructuralLoadSingleDisplacement = 999,
IfcStructuralLoadSingleDisplacementDistortion = 1000,
IfcStructuralLoadSingleForce = 1001,
IfcStructuralLoadSingleForceWarping = 1002,
IfcStructuralLoadStatic = 1003,
IfcStructuralLoadTemperature = 1004,
IfcStructuralMember = 1005,
IfcStructuralPlanarAction = 1006,
IfcStructuralPointAction = 1007,
IfcStructuralPointConnection = 1008,
IfcStructuralPointReaction = 1009,
IfcStructuralReaction = 1010,
IfcStructuralResultGroup = 1011,
IfcStructuralSurfaceAction = 1012,
IfcStructuralSurfaceActivityTypeEnum = 1013,
```

```
IfcStructuralSurfaceConnection = 1014,
IfcStructuralSurfaceMember = 1015,
IfcStructuralSurfaceMemberTypeEnum = 1016,
IfcStructuralSurfaceMemberVarying = 1017,
IfcStructuralSurfaceReaction = 1018,
IfcStyleAssignmentSelect = 1019,
IfcStyledItem = 1020,
IfcStyledRepresentation = 1021,
IfcStyleModel = 1022,
IfcSubContractResource = 1023,
IfcSubContractResourceType = 1024,
IfcSubContractResourceTypeEnum = 1025,
IfcSubedge = 1026,
IfcSurface = 1027,
IfcSurfaceCurve = 1028,
IfcSurfaceCurveSweptAreaSolid = 1029,
IfcSurfaceFeature = 1030,
IfcSurfaceFeatureTypeEnum = 1031,
IfcSurfaceOfLinearExtrusion = 1032,
IfcSurfaceOfRevolution = 1033,
IfcSurfaceOrFaceSurface = 1034,
IfcSurfaceReinforcementArea = 1035,
IfcSurfaceSide = 1036,
IfcSurfaceStyle = 1037,
IfcSurfaceStyleElementSelect = 1038,
IfcSurfaceStyleLighting = 1039,
IfcSurfaceStyleRefraction = 1040,
IfcSurfaceStyleRendering = 1041,
IfcSurfaceStyleShading = 1042,
IfcSurfaceStyleWithTextures = 1043,
IfcSurfaceTexture = 1044,
IfcSweptAreaSolid = 1045,
IfcSweptDiskSolid = 1046,
IfcSweptDiskSolidPolygonal = 1047,
IfcSweptSurface = 1048,
IfcSwitchingDevice = 1049,
IfcSwitchingDeviceType = 1050,
IfcSwitchingDeviceTypeEnum = 1051,
IfcSystem = 1052,
IfcSystemFurnitureElement = 1053,
IfcSystemFurnitureElementType = 1054,
IfcSystemFurnitureElementTypeEnum = 1055,
IfcTable = 1056,
IfcTableColumn = 1057,
IfcTableRow = 1058,
IfcTank = 1059,
IfcTankType = 1060,
IfcTankTypeEnum = 1061,
IfcTask = 1062,
IfcTaskDurationEnum = 1063,
IfcTaskTime = 1064,
IfcTaskTimeRecurring = 1065,
IfcTaskType = 1066,
IfcTaskTypeEnum = 1067,
IfcTelecomAddress = 1068,
IfcTemperatureGradientMeasure = 1069,
IfcTemperatureRateOfChangeMeasure = 1070,
IfcTendon = 1071,
IfcTendonAnchor = 1072,
IfcTendonAnchorType = 1073,
```

```
IfcTendonAnchorTypeEnum = 1074,
IfcTendonType = 1075,
IfcTendonTypeEnum = 1076,
IfcTessellatedFaceSet = 1077,
IfcTessellatedItem = 1078,
IfcText = 1079,
IfcTextAlignment = 1080,
IfcTextDecoration = 1081,
IfcTextFontName = 1082,
IfcTextFontSelect = 1083,
IfcTextLiteral = 1084,
IfcTextLiteralWithExtent = 1085,
IfcTextPath = 1086,
IfcTextStyle = 1087,
IfcTextStyleFontModel = 1088,
IfcTextStyleForDefinedFont = 1089,
IfcTextStyleTextModel = 1090,
IfcTextTransformation = 1091,
IfcTextureCoordinate = 1092,
IfcTextureCoordinateGenerator = 1093,
IfcTextureMap = 1094,
IfcTextureVertex = 1095,
IfcTextureVertexList = 1096,
IfcThermalAdmittanceMeasure = 1097,
IfcThermalConductivityMeasure = 1098,
IfcThermalExpansionCoefficientMeasure = 1099,
IfcThermalResistanceMeasure = 1100,
IfcThermalTransmittanceMeasure = 1101,
IfcThermodynamicTemperatureMeasure = 1102,
IfcTime = 1103,
IfcTimeMeasure = 1104,
IfcTimeOrRatioSelect = 1105,
IfcTimePeriod = 1106,
IfcTimeSeries = 1107,
IfcTimeSeriesDataTypeEnum = 1108,
IfcTimeSeriesValue = 1109,
IfcTimeStamp = 1110,
IfcTopologicalRepresentationItem = 1111,
IfcTopologyRepresentation = 1112,
IfcToroidalSurface = 1113,
IfcTorqueMeasure = 1114,
IfcTransformer = 1115,
IfcTransformerType = 1116,
IfcTransformerTypeEnum = 1117,
IfcTransitionCode = 1118,
IfcTransitionCurveSegment2D = 1119,
IfcTransitionCurveType = 1120,
IfcTranslationalStiffnessSelect = 1121,
IfcTransportElement = 1122,
IfcTransportElementType = 1123,
IfcTransportElementTypeEnum = 1124,
IfcTrapeziumProfileDef = 1125,
IfcTriangulatedFaceSet = 1126,
IfcTriangulatedIrregularNetwork = 1127,
IfcTrimmedCurve = 1128,
IfcTrimmingPreference = 1129,
IfcTrimmingSelect = 1130,
IfcTShapeProfileDef = 1131,
IfcTubeBundle = 1132,
IfcTubeBundleType = 1133,
```

```
IfcTubeBundleTypeEnum = 1134,
IfcTypeObject = 1135,
IfcTypeProcess = 1136,
IfcTypeProduct = 1137,
IfcTypeResource = 1138,
IfcUnit = 1139,
IfcUnitaryControlElement = 1140,
IfcUnitaryControlElementType = 1141,
IfcUnitaryControlElementTypeEnum = 1142,
IfcUnitaryEquipment = 1143,
IfcUnitaryEquipmentType = 1144,
IfcUnitaryEquipmentTypeEnum = 1145,
IfcUnitAssignment = 1146,
IfcUnitEnum = 1147,
IfcURIReference = 1148,
IfcUShapeProfileDef = 1149,
IfcValue = 1150,
IfcValve = 1151,
IfcValveType = 1152,
IfcValveTypeEnum = 1153,
IfcVaporPermeabilityMeasure = 1154,
IfcVector = 1155,
IfcVectorOrDirection = 1156,
IfcVertex = 1157,
IfcVertexLoop = 1158,
IfcVertexPoint = 1159,
IfcVibrationIsolator = 1160,
IfcVibrationIsolatorType = 1161,
IfcVibrationIsolatorTypeEnum = 1162,
IfcVirtualElement = 1163,
IfcVirtualGridIntersection = 1164,
IfcVoidingFeature = 1165,
IfcVoidingFeatureTypeEnum = 1166,
IfcVolumeMeasure = 1167,
IfcVolumetricFlowRateMeasure = 1168,
IfcWall = 1169,
IfcWallElementedCase = 1170,
IfcWallStandardCase = 1171,
IfcWallType = 1172,
IfcWallTypeEnum = 1173,
IfcWarpingConstantMeasure = 1174,
IfcWarpingMomentMeasure = 1175,
IfcWarpingStiffnessSelect = 1176,
IfcWasteTerminal = 1177,
IfcWasteTerminalType = 1178,
IfcWasteTerminalTypeEnum = 1179,
IfcWindow = 1180,
IfcWindowLiningProperties = 1181,
IfcWindowPanelOperationEnum = 1182,
IfcWindowPanelPositionEnum = 1183,
IfcWindowPanelProperties = 1184,
IfcWindowStandardCase = 1185,
IfcWindowStyle = 1186,
IfcWindowStyleConstructionEnum = 1187,
IfcWindowStyleOperationEnum = 1188,
IfcWindowType = 1189,
IfcWindowTypeEnum = 1190,
IfcWindowTypePartitioningEnum = 1191,
IfcWorkCalendar = 1192,
IfcWorkCalendarTypeEnum = 1193,
```

```
    IfcWorkControl = 1194,
    IfcWorkPlan = 1195,
    IfcWorkPlanTypeEnum = 1196,
    IfcWorkSchedule = 1197,
    IfcWorkScheduleTypeEnum = 1198,
    IfcWorkTime = 1199,
    IfcZone = 1200,
    IfcZShapeProfileDef = 1201,
    UNDEFINED = 1202,
    Other3DModel = 1203
}
```

## Model

**Model** – это класс управления консолидированной моделью, загруженной в компонент.

### *Методы*

#### getAllModelParts()

Метод возвращает все загруженные части консолидированной модели.

```
getAllModelParts(): ModelPart[];
```

#### getModelPart()

Метод возвращает часть консолидированной модели по её идентификатору.

```
getModelPart(id: string): ModelPart;
```

где:
`id` – идентификатор части консолидированной модели.

#### getVisibleModelParts()

Метод возвращает все видимые части консолидированной модели.

```
getVisibleModelParts(): ModelPart[]
```

#### getHiddenModelParts()

Метод возвращает все скрытые части консолидированной модели.

```
getHiddenModelParts(): ModelPart[]
```

#### hideModelPart()

Метод позволяет скрыть часть консолидированной модели.

```
hideModelPart(modelPart: string | ModelPart): void;
```

где:
`modelPart` – идентификатор или экземпляр части модели.

#### showModelPart()

Метод позволяет показать часть консолидированной модели.

```
showModelPart(modelPart: string | ModelPart): void
```

где:
`modelPart` – идентификатор или экземпляр части модели.

## hide()

Метод позволяет скрыть элементы модели.

```
hide(elementIds: string[] | string, modelPart?: string | ModelPart): void
```

где:
`elementIds` – один или несколько идентификаторов элементов модели.

`modelPart` – идентификатор или экземпляр части модели. Необязательный параметр. Если этот параметр не задан, то обрабатываются элементы первой загруженной части модели.

## hideAll()

Метод позволяет скрыть все элементы и части модели.

```
hideAll(): void;
```

## show()

Метод позволяет показать скрытые элементы.

```
show(elementIds: string[] | string, modelPart?: string | ModelPart): void;
```

где:
`elementIds` – один или несколько идентификаторов элементов модели.

`modelPart` – идентификатор или экземпляр части модели. Необязательный параметр. Если этот параметр не задан, то обрабатываются элементы первой загруженной части модели.

## showAll()

Метод позволяет показать все элементы и части модели.

```
showAll(): void;
```

## isolate()

Метод позволяет изолировать элементы одной части модели. Все неизолированные элементы модели будут скрыты.

```
isolate(elementIds: string[], modelPart: string | ModelPart): void;
```

где:
`elementIds` – один или несколько идентификаторов элементов модели.

`modelPart` – идентификатор или экземпляр части модели. Необязательный параметр. Если этот параметр не задан, то обрабатываются элементы первой загруженной части модели.

### isolateMultiple()

Метод позволяет изолировать элементы модели. Все неизолированные элементы модели будут скрыты.

```
isolateMultiple(elements: ModelElementIds[]): void;
```

где:
`elements` – список идентификаторов элементов модели, сгруппированных по части модели. Подробнее: ModelElementIds.

### select()

Метод позволяет выделить элементы.

```
select(elementIds: string[] | string, modelPart: string | ModelPart,
selectionMode: SelectionMode): void
```

где:
`elementIds` – один или несколько идентификаторов элементов модели.

`modelPart` – идентификатор или экземпляр части модели.

`selectionMode` – режим выделения. Подробнее: SelectionMode.

### deselect()

Метод позволяет снять выделение с заданных элементов.

```
deselect(elementIds: string[] | string, modelPart?: string | ModelPart): void
```

где:
`elementIds` – один или несколько идентификаторов элементов модели.

`modelPart` – идентификатор или экземпляр части модели. Необязательный параметр. Если этот параметр не задан, то обрабатываются элементы первой загруженной части модели.

### clearSelection()

Метод позволяет снять выделение со всех элементов модели.

```
clearSelection(): void;
```

### getSelection()

Метод позволяет получить выделенные элементы модели. Подробнее: ModelElementIds.

```
getSelection(): ModelElementIds[];
```

## getHiddenElements()

Метод позволяет получить все скрытые элементы. Подробнее: ModelElementIds.

```
getHiddenElements(): ModelElementIds[]
```

## getIsolatedElements()

Метод позволяет получить все изолированные элементы. Подробнее: ModelElementIds.

```
getIsolatedElements(): ModelElementIds[]
```

## getVisibleElements()

Метод позволяет получить все видимые элементы. Подробнее: ModelElementIds.

```
getVisibleElements(): ModelElementIds[]
```

## setColor()

Метод позволяет задать цвет элементов модели.

```
setColor(elementIds: string[] | string, r: number, g: number, b: number, a:
number, modelPart?: string | ModelPart): void
```

где:
`elementIds` – один или несколько идентификаторов элементов модели.

`modelPart` – идентификатор или экземпляр части модели. Необязательный параметр. Если этот параметр не задан, то обрабатываются элементы первой загруженной части модели.

`r` – красный цвет (0-255).

`g` – зеленый цвет (0-255).

`b` – синий цвет (0-255).

`a` – альфа-канал (0-1).

## clearColors()

Метод позволяет восстановить все цвета для элементов модели.

```
clearColors(model? : string | ModelPart): void;
```

где:
`modelPart` – идентификатор или экземпляр части модели. Необязательный параметр. Если этот параметр не задан, то обрабатывается первая загруженная часть модели.

## setGhostMode()

Метод задает призрачный режим отображения для скрытых объектов. В этом режиме скрытые объекты отображаются на сцене в полупрозрачном, бесцветном виде.

```
setGhostMode(value: boolean): void;
```

где:
`value` – задаёт активность призрачного режима.

## getElementProperties()

Метод позволяет получить свойства элемента.

```
getElementProperties(elementId: string, modelPart?: string | ModelPart,
version?: BigInt): ModelElementPropertySet[]
```

где:
`elementId` – идентификатор элемента.

`modelPart` – идентификатор части модели или экземпляр части модели. Если этот параметр не задан, то обрабатывается первая загруженная часть модели.

`version` – версия модели. Задается в тиках. Если версия не указана, то берутся свойства актуальной версии загруженной части модели.

Возвращает набор данных типа ModelElementPropertySet.

## canDelete()

Метод проверяет, что переданные идентификаторы объектов соответствуют фильтрам объектов для удаления.

```
canDelete(elementIds: ModelElementIds[]): boolean;
```

где:
`elementIds` – идентификаторы объектов для проверки удаления. Подробнее: ModelElementIds.

Возвращает `true`, если каждый из переданных идентификаторов соответствует хотя бы одному фильтру.

## delete()

Метод вызывает DELETE_OBJECTS_EVENT с переданными идентификаторами объектов модели, если каждый идентификатор соответствует хотя бы одному фильтру.

```
delete(elementIds: ModelElementIds[]): boolean;
```

где:
`elementIds` – идентификаторы объектов для удаления. Подробнее: ModelElementIds.

Возвращает `true`, если удалось вызвать DELETE_OBJECTS_EVENT. В противном случае возвращается `false`.

## addDeletionFilter()

Метод добавляет фильтр объектов для удаления.

```
addDeletionFilter(filter: DeleteEventFilter): void;
```

где:
`filter` – фильтр объектов для удаления. Подробнее: DeleteEventFilter.

## removeDeletionFilter()

Метод удаляет фильтр объектов для удаления.

```
removeDeletionFilter(filter: DeleteEventFilter): void;
```

где:
`filter` – фильтр объектов для удаления. Подробнее: DeleteEventFilter.

## DeleteEventFilter

Фильтр объектов для удаления.

```
interface DeleteEventFilter {
  (modelId: string, entityId: string): boolean;
}
```

где:
`modelId` – идентификатор модели.

`entityId` – идентификатор элемента модели.

Возвращает `true` для объектов, которые можно удалить. В противном случае – `false`.

# ModelElement

## ModelElement

**ModelElement** – это интерфейс для получения информации об элементе.

### *Свойства*

#### id

Получает идентификатор элемента.

```
get id(): string;
```

#### modelPartId

Получает идентификатор части модели, к которой относится этот элемент.

```
get modelPartId(): string;
```

#### parent

Получает родительский элемент. Если родитель отсутствует, то вернется значение `undefined`.

```
get parent(): ModelElement | undefined;
```

#### type

Получает тип элемента.

```
get type(): string;
```

#### name

Получает имя элемента.

```
get name(): string;
```

#### children

Получает детей элемента.

```
get children(): ModelElement[];
```

#### hasGeometry

Проверяет наличие геометрии у элемента.

```
get hasGeometry(): boolean;
```

### viewObject

Получает геометрию элемента, если она есть. В противном случае возвращает `undefined`.

```
get viewObject(): ViewObject | undefined;
```

Подробнее: ViewObject.

## ModelElementIds

**ModelElementIds** – этот класс описывает агрегированный список идентификаторов элементов по части модели.

### *Свойства*

### modelPartId

Идентификатор части модели.

```
modelPartId: string;
```

### elementIds

Идентификаторы элементов выборки.

```
elementIds: string[];
```

## ModelElementProperty

**ModelElementProperty** – этот класс описывает свойство элемента модели.

```
class ModelElementProperty {
  name: string;
  unit: number;
  value: ModelElementPropertyValue;
}
```

### *Свойства*

#### name

Имя свойства элемента

```
name: string;
```

#### unit

Единица измерения свойства элемента.

```
unit: number;
```

#### value

Значение свойства элемента. Подробнее: см ModelElementPropertyValue

```
value: ModelElementPropertyValue;
```

## ModelElementPropertySet

**ModelElementPropertySet** – этот класс описывает свойства элемента модели.

```
class ModelElementPropertySet {
  name: string;
  properties: ModelElementProperty[];
  type: IfcType;
}
```

### *Свойства*

#### name

Имя категории свойств элемента

```
name: string;
```

#### properties

Список свойств элемента. Подробнее: см ModelElementProperty

```
properties: ModelElementProperty[];
```

#### type

Тип категории свойств элемента. Подробнее: см IfcType

```
type: IfcType;
```

## ModelElementPropertyValue

**ModelElementPropertyValue** – этот класс описывает значение свойства элемента модели.

```
class ModelElementPropertyValue {
  str_value?: string;
  int_value?: number;
  double_value?: number;
  date_value?: BigInt;
  array_value: Array<string>;
  decimal_value?: number;
  guid_value?: string;
  array_int_value: Array<number>;
  bool_value?: boolean;
  get value(): any;
}
```

**Заполнено будет только одно поле, соответствующее типу значения этого свойства. Например: если значение равно `0.0`, то будет заполнено поле `double_value`.**

## ModelElementTree

**ModelElementTree** – это интерфейс для работы с деревом элементов части консолидированной модлели.

### *Методы*

### enumElementChildren()

Метод позволяет пройтись по всем элементам дерева и выполнить заданную функцию над каждым элементом.

```
enumElementChildren(element: string | ModelElement, callback: (guid: string)
=> void, recursive?: boolean): void;
```

где:
`element` – идентификатор или экземпляр элемента.

`callback` – функция-обработчик элемента.

`recursive` – обрабатывать рекурсивно все ветки элемента или только его детей. Параметр не обязательный.

### getRootElement()

Метод позволяет получить корневой элемент части консолидированной модели.

```
getRootElement(): ModelElement;
```

### getAllElements()

Метод позволяет получить все элементы дерева списком.

```
getAllElements(): ModelElement[]
```

### getElement()

Метод позволяет получить элемент дерева по идентификатору.

```
getElement(id: string) : ModelElement
```

### isViewableElement()

Метод проверяет, может ли элемент быть отрисован на сцене.

```
isViewableElement(element: string | ModelElement): boolean;
```

где:
`element` – идентификатор или экземпляр элемента.

## isDetachedElement()

Метод проверяет, отсоединён элемент от корневого элемента или нет.

```
isDetachedElement(element: string | ModelElement): boolean;
```

где:
`element` – идентификатор или экземпляр элемента.

## getChildLevelNumber()

Метод получает уровень вложенности для элемента.

```
getChildLevelNumber(element: string | ModelElement): number
```

где:
`element` – идентификатор или экземпляр элемента.

## ModelLoadingOptions

**ModelLoadingOptions** - опции загрузки модели в компонент **PilotWeb3d**.

```
export class ModelLoadingOptions {
  guid: string; // уникальный идентификатор модели
  isConsolidatedModel?: boolean; // флаг, позволяющий дозагрузить модели в
уже загруженную сцену
}
```

# ModelPart

**ModelPart** – этот интерфейс позволяет получить информацию о части консолидированной модели.

### *Свойства*

## id

Получает идентификатор части консолидированной модели.

```
get id() : string;
```

## elementTree()

Получает экземпляр дерева элементов части консолидированной модели.

```
get elementTree() : ModelElementTree
```

### *Методы*

## dispose()

Освобождает ресурсы, занятые частью консолидированной модели.

```
dispose() : void;
```

# Navigation

Работа с навигацией осуществляется через объект навигации - INavigation, который можно получить из объекта Viewer3D.

```
const navigation = viewer3D.navigation;
```

## CameraNavigationMode

**CameraNavigationMode** – классификация перемещений, осуществляемых камерой при навигации.

```
export enum CameraNavigationMode {
  // Движение отсутствует
  None = 0,
  // Вращение по круговой орбите
  Rotation = 1 << 0,
  // Линейное движение
  Translation = 1 << 1,
  // Вращение вокруг собственной оси
  Spin = 1 << 2,
  // Приближение или удаление
  Zoom = 1 << 3
}
```

## CameraOrientation

**CameraOrientation** – описание ориентации камеры в мировом пространстве.

```
export type CameraOrientation = {
  viewDir: THREE.Vector3,
  upDir?: THREE.Vector3
};
```

### *Свойства*

### viewDir

Вектор направления взгляда камеры.

```
viewDir: THREE.Vector3;
```

### upDir

Вектор направления верха камеры. Не обязательный параметр. Если не указан, то сохраняется текущее направление верха камеры.

```
upDir: THREE.Vector3;
```

## CameraParameters

**CameraParameters** – описание параметров камеры.

```
export type CameraParameters = {
  /** Camera position */
  position: Point3,
  /** Camera view direction: vector pointing from camera to target */
  eyeDir: Point3,
  /** Field of view: angle in radians */
  angle: number,
  /** (optional) Camera up vector*/
  upDir?: Point3,
  /** (optional) Vector pointing to the center of viewing area.*/
  viewCenter?: Point3
};
```

### position

Позиция камеры. Подробнее: Point3.

```
position: Point3;
```

### eyeDir

Вектор направления взгляда камеры. Подробнее: Point3.

```
eyeDir: Point3;
```

### angle

Поле зрения камеры. Задается в радианах.

```
angle: number
```

### upDir

Вектор направления верха камеры, не обязательный параметр. Подробнее: Point3.

```
upDir?: Point3
```

### viewCenter

Вектор точки взгляда, не обязательный параметр. Подробнее: Point3.
Используется как опорная точка при вращении камеры во время навигации, если не задан PivotPoint.

```
viewCenter?: Point3;
```

## DesktopNavigation

**DesktopNavigation** - Обработчик навигации для десктопной версии приложения.

```
export class DesktopNavigation extends NavigationTool {
  protected _prevMousePos?: THREE.Vector2;
  protected _mouseLftIsDown = false;
  protected _mouseLftIsDownPos?: THREE.Vector2;
  protected _mouseRhtIsDown = false;
  protected _mouseMidIsDown = false;

  protected onMouseEnter(ev: MouseEvent & NavigationEvent): void;
  protected onMouseLeave(ev: MouseEvent & NavigationEvent): void;
  protected onMouseMove(ev: MouseEvent & NavigationEvent): void;
  protected onMouseClick(ev: MouseEvent & NavigationEvent): void;
  protected onMouseDoubleClick(ev: MouseEvent & NavigationEvent): void;
  protected onMouseScroll(ev: WheelEvent & NavigationEvent): void;
  protected onMouseUp(ev: MouseEvent & NavigationEvent): void ;
  protected onKeyDown(ev: KeyboardEvent & NavigationEvent): void;
  protected onKeyUp(ev: KeyboardEvent & NavigationEvent): void;
  protected isAllMouseButtonsUp(): boolean;
}
```

## ICameraControl

**ICameraControl** – интерфейс, позволяющий взаимодействовать с контроллером камеры.

```
export interface ICameraControl {
  getCamera(): THREE.Camera;
  getCameraParameters(): CameraParameters;
  setCameraParameters(iParams: CameraParameters): void;
  setAspectRatio(width: number, heigth: number): boolean;

  rotate(movement: THREE.Vector2, rotationCenter: THREE.Vector3): void;
  translate(startNdcPos: THREE.Vector2, endNdcPos: THREE.Vector2, viewCenter:
THREE.Vector3): void;
  spin(movement: THREE.Vector2): void;
  getCameraOrientation(): CameraOrientation;
  setCameraOrientation(iOrientation: CameraOrientation, isAnimationEnabled?:
boolean): void;
  setViewCenter(viewCenter: THREE.Vector3): void;
  getViewCenter(): THREE.Vector3;
  zoomToPoint(deltaSign: number, point: THREE.Vector3): void;
  zoomToFit(bb: THREE.Box3, iOrientation?: CameraOrientation,
isAnimationEnabled?: boolean): void;

  getNavigationMode(): CameraNavigationMode;
  setNavigationMode(mode: CameraNavigationMode, isEnable: boolean, duration?:
number): void;
}
```

### *Методы*

### getCamera()

Метод позволяет получить камеру.

```
  getCamera(): THREE.Camera;
```

Возвращает объект камеры. Подробнее: THREE.Camera.

### getCameraParameters()

Метод позволяет получить параметры камеры.

```
getCameraParameters(): CameraParameters;
```

Возвращает параметры камеры. Подробнее: CameraParameters.

### setCameraParameters()

Метод позволяет задать параметры камеры.

```
  setCameraParameters(iParams: CameraParameters): void;
```

где:
`iParams` – параметры камеры. Подробнее: CameraParameters.

### setAspectRatio()

Метод позволяет задать соотношение сторон камеры.

```
setAspectRatio(width: number, heigth: number): boolean;
```

где:
`width` – ширина изображения, может быть в любых величинах.
`heigth` – высота изображения, может быть в любых величинах.
Результирующе соотношение сторон равно `width / heigth`.
Возвращает `true`, если новое соотношение сторон отличается от предыдущего..

### rotate()

Метод позволяет повернуть камеру вокруг точки вращения по круговой орбите.

```
rotate(movement: THREE.Vector2, rotationCenter: THREE.Vector3): void;
```

где:
`movement` – смещение в экранных координатах.
`rotationCenter` – точка, относительно которой осуществляется вращение камеры.

### translate()

Метод позволяет переместить камеру по прямолинейной траектории относительно заданной точки.

```
translate(startNdcPos: THREE.Vector2, endNdcPos: THREE.Vector2, viewCenter:
THREE.Vector3): void;
```

где:
`startNdcPos` – начальное положение в Normalized Device Coordinates (NDC пространство).
`endNdcPos` – конечное положение в Normalized Device Coordinates (NDC пространство).
`viewCenter` – точка, относительно которой осуществляется смещение камеры.

### spin()

Метод позволяет повернуть камеру вокруг `Up` вектора камеры.

```
spin(movement: THREE.Vector2): void;
```

где:
`movement` – смещение в экранных координатах.

### getCameraOrientation()

Метод позволяет получить ориентацию камеры в пространстве. Подробнее: CameraOrientation.

```
getCameraOrientation(): CameraOrientation;
```

## setCameraOrientation()

Метод позволяет ориентировать камеру в пространстве, позиция камеры при этом не изменяется.

```
setCameraOrientation(iOrientation: CameraOrientation, isAnimationEnabled?:
boolean): void;
```

где:
`iOrientation` – конечная ориентация камеры. Подробнее: CameraOrientation.
`isAnimationEnabled` – анимация при изменении ориентации, `true` – отключить анимацию, `false` – включить анимацию. По умолчанию анимация включена.

## getViewCenter()

Метод позволяет получить положение точки взгляда. Подробнее: ViewCenter.

```
getViewCenter(): THREE.Vector3;
```

## setViewCenter()

Метод позволяет задать положение точки взгляда. Подробнее: ViewCenter.

```
setViewCenter(viewCenter: THREE.Vector3): void;
```

где:
`viewCenter` – положение точки взгляда в мировом пространстве.

## zoomToPoint()

Метод позволяет приблизить либо отдалить камеру относительно точки.

```
zoomToPoint(deltaSign: number, point: THREE.Vector3): void;
```

где:
`deltaSign` – дистанция приближения.
`point` – точка приближения.

## zoomToFit()

Метод позволяет центрировать камеру относительно ограничивающего объема в нужной ориентации.

```
zoomToFit(boundingBox: THREE.Box3, iOrientation?: CameraOrientation,
isAnimationEnabled?: boolean): void;
```

где:
`boundingBox` – ограничивающий объем, относительно которого центрируется камера. Подробнее: THREE.Box3.
`iOrientation` – конечная ориентация камеры. По умолчанию сохраняется текущая ориентация камеры. Подробнее: CameraOrientation.
`isAnimationEnabled` – анимация при центрировании, `true` – отключить анимацию, `false` – включить анимацию. По умолчанию анимация включена.

## getNavigationMode()

Метод позволяет получить тип навигации, осуществляемой над камерой.

```
getNavigationMode(): CameraNavigationMode;
```

Возвращает тип навигации камеры. Подробнее: CameraNavigationMode.

## setNavigationMode()

Метод позволяет задать тип навигации камеры в пространстве.

```
setNavigationMode(mode: CameraNavigationMode, isEnable: boolean, duration?:
number): void;
```

где:
`mode` – тип навигации камеры. Подробнее: CameraNavigationMode.
`isEnable` – активность.
`duration` – продолжительность навигации в миллисекундах. Параметр применяется только если `isEnable == true`.

Если `isEnable == true`, и определён `duration`, то по истечении задержки `duration` произойдет отключение заданного типа навигации. Эквивалентно вызову `setNavigationMode(mode, false)` после задержки `duration`.

Если во время ожидания происходит вызов `setNavigationMode` с любыми параметрами, то задержка сбрасывается и вызов `setNavigationMode(mode, false)` происходит немедленно, затем применяются новые параметры.

## INavigation

**INavigation** – интерфейс, позволяющий взаимодействовать с навигацией на сцене.

```
export interface INavigation {
  registerNavigation(navigationTool: INavigationTool): void;
  unregisterNavigation(navigationTool: INavigationTool): void;
  setActive(navigationToolName: string, isActive: boolean): void;
  getActiveNavigation(): INavigationTool | null;
  getNavigationAgent(): INavigationAgent;
  getNavigationArea(): DOMRect;
  setCameraParameters(params: CameraParameters): void;
  getCameraParameters(): CameraParameters;
  getCameraControl(): ICameraControl;
  getCamera(): THREE.Camera;
  fitToView(elementIds: string[] | string, modelPart: string | ModelPart,
immediate?: boolean): void;
  setPivotPoint(point: Point3): void;
  getPivotPoint(): Point3;
  resetPivotPoint(): void;
}
```

### *Методы*

### registerNavigation()

Метод позволяет зарегистрировать обработчик событий навигации.

```
registerNavigation(navigationTool: INavigationTool): void;
```

где:\

`navigationTool` – реализация обработчика событий. Подробнее: INavigationTool.

### unregisterNavigation()

Метод позволяет разрегистрировать обработчик событий навигации.

```
unregisterNavigation(navigationTool: INavigationTool): void;
```

где:\

`navigationTool` – реализация обработчика событий. Подробнее: INavigationTool.

### setActive()

Метод позволяет активировать обработчик событий навигации. Активным может быть только один обработчик навигации в каждый момент времени.

```
setActive(navigationToolName: string, isActive: boolean): void;
```

где:
`navigationToolName` – имя обработчика навигации,
`isActive` – активность.

### getActiveNavigation()

Метод позволяет получить текущий обработчик навигации.

```
getActiveNavigation(): INavigationTool | null;
```

Возвращает объект INavigationTool.

### getNavigationAgent()

Метод позволяет получить INavigationAgent - предоставляющий источники событий навигации.

```
getNavigationAgent(): INavigationAgent;
```

Возвращает объект INavigationAgent.

### getNavigationArea()

Метод позволяет получить прямоугольник области навигации.

```
  getNavigationArea(): DOMRect;
```

Возвращает объект DOMRect.

### setCameraParameters()

Метод позволяет установить параметры камеры.

```
setCameraParameters(params: CameraParameters): void;
```

где:\

`params` – параметры камеры. Подробнее: CameraParameters.

### getCameraParameters()

Метод позволяет получить параметры камеры.

```
getCameraParameters(): CameraParameters;
```

Возвращает параметры камеры. Подробнее: CameraParameters.

### getCameraControl()

Метод позволяет получить контроллер камеры.

```
getCameraControl(): ICameraControl;
```

Возвращает контроллер камеры. Подробнее: ICameraControl.

## getCamera()

Метод позволяет получить камеру.

```
getCamera(): THREE.Camera;
```

Возвращает объект камеры. Подробнее: THREE.Camera.

## fitToView()

Метод позволяет спозиционировать заданные элементы в центре экрана.

```
fitToView(elementIds: string[] | string, modelPart: string | ModelPart,
immediate?: boolean): void;
```

где:

`elementIds` – список идентификаторов или один идентификатор элемента сцены,
`modelPart` – идентификатор части консолидированной модели или объект части консолидированной модели,
`immediate` – анимация при центрировании, `true` – отключить анимацию, `false` – включить анимацию. По умолчанию анимация включена.

## setPivotPoint()

Метод позволяет задать положение опорной точки камеры.

```
setPivotPoint(point: Point3): void;
```

где:

`point` – точка в пространстве сцены. Подробнее: Point3.

## getPivotPoint()

Метод позволяет получить положение опорной точки камеры.

```
getPivotPoint(): Point3;
```

Возвращает объект типа Point3.

## resetPivotPoint()

Метод позволяет сбросить положение опорной точки камеры.

```
resetPivotPoint(): void;
```

## INavigationAgent

**INavigationAgent** – интерфейс, позволяющий работать с источниками событий навигации.

```
export interface INavigationAgent  {

  readonly canvasNavigationSource: INavigationEventSource;
  readonly keyboardNavigationSource: INavigationEventSource;

  get isActive(): boolean;
  set isActive(value: boolean);

  getNavigationArea(): DOMRect;
}
```

### *Поля*

### canvasNavigationSource

Источник DOM-событий для навигации с помощью мыши, тачпада и т.д. Подробнее: INavigationEventSource.

```
readonly canvasNavigationSource: INavigationEventSource;
```

Пример подписки на событие mousemove, используются NavigationEventOptions, заданные по умолчанию:

```
navigationAgent.canvasNavigationSource.addEventListener("mousemove",
onMouseMove);
//или
navigationAgent.canvasNavigationSource.addEventListener("mousemove",
onMouseMove, false); //capture: false
//Те же опции, указанные явно:
navigationAgent.canvasNavigationSource.addEventListener("mousemove",
onMouseMove, new NavigationEventOptions(false,
NavigationHandlerPriority.DefaultNavigation, false, undefined));
//Для отписки от события, нужно передавать те же параметры, что и при
подписке:
navigationAgent.canvasNavigationSource.removeEventListener("mousemove",
onMouseMove, new NavigationEventOptions(false,
NavigationHandlerPriority.DefaultNavigation, false, undefined));
```

### keyboardNavigationSource

Источник DOM-событий для навигации с помощью клавиатуры. Подробнее: INavigationEventSource.

```
readonly keyboardNavigationSource: INavigationEventSource;
```

Пример подписки на событие `keyup`, используются NavigationEventOptions, заданные по умолчанию:

```
navigationAgent.keyboardNavigationSource.addEventListener("keyup",
onMouseMove);
//или
navigationAgent.keyboardNavigationSource.addEventListener("keyup", onKeyUp,
{capture: false});
//Те же опции, указанные явно:
navigationAgent.keyboardNavigationSource.addEventListener("keyup", onKeyUp,
new NavigationEventOptions(false,
NavigationHandlerPriority.DefaultNavigation, false, 'desktopNavigation'));
//Для отписки от события, нужно передавать те же параметры, что и при
подписке:
navigationAgent.keyboardNavigationSource.removeEventListener("keyup",
onKeyUp, new NavigationEventOptions(false,
NavigationHandlerPriority.DefaultNavigation, false, 'desktopNavigation'));
```

### *Свойства*

## isActive()

Свойство определяет активность агента навигации.

```
  get isActive(): boolean;
  set isActive(value: boolean);
```

### *Методы*

## getNavigationArea()

Метод позволяет получить прямоугольник текущей рабочей области навигации.

```
getNavigationArea(): DOMRect;
```

Возвращает объект DOMRect.

**В десктопной реализации навигации, при перемещении курсора за пределы рабочей области навигации происходит деактивация источника событий клавиатуры. При возвращении курсора в рабочую область происходит активация источника событий клавиатуры.**
**Таким образом, события клавиатуры не генерируются, если курсор находится за пределами рабочей области.**

## INavigationEventSource

**INavigationEventSource** – интерфейс источника событий навигации.

```
export interface INavigationEventSource {
  // Свойство определяет активность источника событий навигации.
  get isActive(): boolean;
  set isActive(value: boolean);

  addEventListener<T extends keyof NavigationEventSourceEventMap>(type: T,
listener: (this: object, ev: NavigationEventSourceEventMap[T] &
NavigationEvent) => void, options?: boolean | EventListenerOptions |
NavigationEventOptions): void;

  removeEventListener<T extends keyof NavigationEventSourceEventMap>(type: T,
listener: (this: object, ev: NavigationEventSourceEventMap[T] &
NavigationEvent) => void, options?: boolean | EventListenerOptions |
NavigationEventOptions): void;
}
```

## NavigationEventSourceEventMap

**NavigationEventSourceEventMap** – события, генерируемые источником событий навигации.
Расширяет `HTMLElementEventMap` – список встроенных DOM событий.

```
export interface NavigationEventSourceEventMap extends HTMLElementEventMap {
  `active`: ActiveEvent
}
```

где:
`active` – событие возникающее при изменении активности данного источника событий навигации. Подробнее: ActiveEvent.

### ActiveEvent

Представляет событие, которое происходит при изменении активности источника событий навигации.
Расширяет `Event`.

```
export interface ActiveEvent extends Event {
  // Активность источника событий навигации.
  readonly isActive: boolean
}
```

## NavigationEvent

**NavigationEvent** – базовый класс события навигации.

```
export class NavigationEvent {
  // Задает и показывает было ли обработано событие.
```

```
  isHandled?: boolean;
}
```

## NavigationEventOptions

**NavigationEventOptions** – опции подписки на событие навигации.

```
export class NavigationEventOptions implements EventListenerOptions {
  // Перехват события при всплытии (false), иначе при погружении (true). По
умолчанию: false.
  capture: boolean;
  // Приоритет вызова обработчиков: от наибольшего к наименьшему. По
умолчанию: NavigationHandlerPriority.CustomExtensions.
  priority: number | NavigationHandlerPriority;
  // Перехват события, если событие уже было обработано ранее (true), иначе
событие игнорируется (false). По умолчанию false.
  alwaysHandle: boolean;
  // Идентификатор подписчика на событие. Не обязательный параметр.
  navigationTargetName?: string;
}
```

## NavigationHandlerPriority

**NavigationHandlerPriority** – приоритет вызова обработчиков события навигации. Обработчики вызываются в порядке убывания приоритета: от `CustomExtensions` к `DefaultNavigation`.

```
export enum NavigationHandlerPriority {
  DefaultNavigation = 0,
  GizmoHandlers = 20,
  EmbeddedExtensions = 40,
  CustomExtensions = 100
}
```

## INavigationTool

**INavigationTool** – интерфейс, позволяющий взаимодействовать с навигацией на сцене.

Для того, чтобы создать свой обработчик для навигации по сцене, необходимо реализовать интерфейс PilotWeb3D.INavigationTool, либо унаследоваться от класса PilotWeb3D.NavigationTool.

Например, PilotWeb3D.MobileNavigation и PilotWeb3D.DesktopNavigation наследуются от PilotWeb3D.NavigationTool, который является реализацией интерфейса `PilotWeb3D.INavigationTool`.

```
export interface INavigationTool {
  get name(): string;
  init(navAgent: INavigationAgent, cameraControl: ICameraControl,
intersectionChecker: IModelIntersectionChecker): void;
  setActive(isActive: boolean): void;
  getPivotPoint(): THREE.Vector3;
  setPivotPoint(pivotPoint: THREE.Vector3): void;
  setCameraParameters(iParams: CameraParameters): void;
  getCameraParameters(): CameraParameters;
}
```

### *Методы*

#### get name()

Метод позволяет получить имя обработчика навигации.

```
get name(): string;
```

Возвращает имя обработчика навигации.

#### init()

Инициализатор обработчика навигации. Вызывается из INavigation при регистрации обработчика.

```
init(navAgent: INavigationAgent, cameraControl: ICameraControl,
intersectionChecker: IModelIntersectionChecker): void;
```

где:
`navAgent` – агент навигации. Подробнее: INavigationAgent.
`cameraControl` – контроллер камеры. Подробнее: ICameraControl.
`intersectionChecker` – обработчик пересечений на сцене. Подробнее: IModelIntersectionChecker.

#### setActive()

Метод позволяет активировать обработчик событий навигации. Вызывается из INavigation при активации/деактивации обработчика.

```
setActive(isActive: boolean): void;
```

где:
`isActive` – активность.

## getPivotPoint()

Метод позволяет получить положение опорной точки камеры.

```
getPivotPoint(): THREE.Vector3;
```

Возвращает объект типа THREE.Vector3.

## setPivotPoint()

Метод позволяет задать положение опорной точки камеры.
Опорная точка используется при навигации по сцене, вращение камеры осуществляется относительно опорной точки. Если точка не задана, то используется `viewCenter` заданный в `CameraControl`. Подробнее: getViewCenter, ViewCenter.

```
setPivotPoint(pivotPoint: THREE.Vector3): void;
```

где:
`pivotPoint` – точка в мировом пространстве. Подробнее: THREE.Vector3.

## setCameraParameters()

Метод позволяет задать параметры камеры.

```
  setCameraParameters(iParams: CameraParameters): void;
```

где:
`iParams` – параметры камеры. Подробнее: CameraParameters.

## getCameraParameters()

Метод позволяет получить параметры камеры.

```
getCameraParameters(): CameraParameters;
```

Возвращает параметры камеры. Подробнее: CameraParameters.

## MobileNavigation

**MobileNavigation** - Обработчик навигации для мобильной версии приложения.

```
export class MobileNavigation extends NavigationTool {
  protected _initialTap?: THREE.Vector2;
  protected _prevTap?: THREE.Vector2;
  protected _prevPinch?: [THREE.Vector2, THREE.Vector2];

  protected onTouchStart(evt: TouchEvent & NavigationEvent): void;
  protected onTouchEnd(evt: TouchEvent & NavigationEvent): void;
  protected onTouchMove(evt: TouchEvent & NavigationEvent): void;
  protected onSwipe(currentPos: THREE.Vector2): void;
  protected onPinch(iTouchPair: [THREE.Vector2, THREE.Vector2]): void;
  protected getPinchCenter(iTouchPair: [THREE.Vector2, THREE.Vector2]):
THREE.Vector2;
  protected getTouchPoint(touch: Touch): THREE.Vector2;
  protected getTouchPair(curTouches: TouchList): [THREE.Vector2,
THREE.Vector2];
}
```

## NavigationTool

**NavigationTool** - Обработчик навигации, базовый класс.

```
export abstract class NavigationTool implements INavigationTool {
  protected _isActive: boolean;
  protected _cameraControl: ICameraControl;
  protected _intersectionChecker: IModelIntersectionChecker;
  protected _viewCenter: THREE.Vector3;
  protected _pivotPoint: THREE.Vector3;
  protected _navAgent: INavigationAgent;

  public abstract get name(): string;
  public init(navAgent: INavigationAgent, cameraControl: ICameraControl,
intersectionChecker: IModelIntersectionChecker): void;
  public setActive(isActive: boolean): void;
  public getPivotPoint(): THREE.Vector3;
  public setPivotPoint(pivotPoint: THREE.Vector3): void;
  public setCameraParameters(iParams: CameraParameters): void;
  public getCameraParameters(): CameraParameters;

  protected abstract addEvents(): void;
  protected abstract removeEvents(): void;
  protected handleHovered(object: THREE.Object3D): void;
  protected handleClick(object: THREE.Object3D, ctrlKey: boolean): void;
  protected handleDblClick(object: THREE.Object3D): void;
  protected getViewCenter(): THREE.Vector3;
  protected setViewCenter(viewCenter: THREE.Vector3): void;
  protected rotate(movement: THREE.Vector2): void;
  protected translate(prevPos: THREE.Vector2, currPos: THREE.Vector2): void;
  protected spin(movement: THREE.Vector2): void;
  protected zoom(deltaSign: number): void;
  protected resetSelection();
  protected onEventHandled(event: Event & NavigationEvent): void;
}
```

## Point3

**Point3** – точка в 3-х координатах.

```
type Point3 = {
  x: number;
  y: number;
  z: number;
}
```

# PilotWeb3D

`PilotWeb3D` – это пространство имён верхнего уровня для взаимодействия с компонентом **PilotWeb3D**.

### *Методы*

## Initializer()

Метод для инициализации компонента **PilotWeb3D**. Все методы работы с компонентом следует использовать после вызова этого метода.

```
type InitializeSuccessCallback = () => void;
function Initializer(options, callback: InitializeSuccessCallback): void;
```

где:
`options` – содержит настройки для инициализации компонента. `callback` – метод обратного вызова. Вызывается, когда завершится инициализация компонента.

Пример:

```
let options = {};
var myCallback = function() {
   console.log("initialization complete, creating the viewer...");
};
PilotWeb3D.Initializer(options, myCallback);
```

## CreateViewer()

Создает экземпляр компонента для просмотра документов.

```
function CreateViewer(container: HTMLElement) : GuiViewer3D;
```

где:
`container` – HTML элемент, в котором создается компонент.

Пример:

```
let htmlDiv = document.getElementById('pilotViewer');
let viewer = PilotWeb3D.CreateViewer(htmlDiv);
```

# Render

## Отображение моделей

Работа с отображением и взаимодействием с 3D-моделями осуществляется через объект IRenderViewer3d, который можно получить из объекта Viewer3D.

```
const renderView = viewer3D.renderView;
```

## Color

**Color** – класс, описывающий цвет ViewObject.

```
export class Color {
  constructor(public r: number, public g: number, public b: number, public a:
number);

  static fromThreeColor(color: THREE.Color, alpha = 1.0): Color;

  static fromColorRepresentation(representation: THREE.ColorRepresentation):
Color;

  static fromMaterial(material: THREE.Material): Color;

  threeColor(): THREE.Color;

  alpha(): number;

  fromArray(array: ArrayLike<number>, offset = 0): Color {
    this.r = array[offset];
    this.g = array[offset + 1];
    this.b = array[offset + 2];
    this.a = array[offset + 3];
    return this;
  }

  toArray(array: Array<number>, offset = 0): Array<number> {
    array[offset] = this.r;
    array[offset + 1] = this.g;
    array[offset + 2] = this.b;
    array[offset + 3] = this.a;
    return array;
  }

  clone(): Color;
}
```

### *Конструктор*

```
constructor(r: number, g: number, b: number, a: number);
```

где:
`r` – значение интенсивности красной компоненты цвета, от `0.0` до `1.0`.
`g` – значение интенсивности зелёной компоненты цвета, от `0.0` до `1.0`.
`b` – значение интенсивности синей компоненты цвета, от `0.0` до `1.0`.
`a` – альфа канал, значение прозрачности, от `0.0` до `1.0`.

### *Методы*

## fromThreeColor()

Метод возвращает объект `Color` с `r`, `g`, `b` - компонентами цвета, соответствующими `r`, `g`, `b` - компонентам параметра `color` и прозрачностью равной `alpha`.

```
static fromThreeColor(color: THREE.Color, alpha = 1.0): Color;
```

где:
`color` - источник `r`, `g`, `b` компонент цвета. Подробнее: THREE.Color.
`alpha` - значение прозрачности, от `0.0` до `1.0`. По умолчанию: `1.0`.

## fromColorRepresentation()

Метод возвращает объект `Color` с параметрами цвета, соответствующими `THREE.ColorRepresentation` и прозрачностью равной `1.0`.

```
static fromColorRepresentation(representation: THREE.ColorRepresentation):
Color;
```

где:
`THREE.ColorRepresentation` - объект представления цвета в виде `THREE.Color | string | number`. Подробнее: THREE.Color

## fromMaterial()

Метод возвращает объект `Color` с параметрами цвета, соответствующими `material`.

```
static fromMaterial(material: THREE.Material): Color;
```

где:
`material` - материал, по которому вычисляются параметры цвета.
`r`, `g`, `b` - компоненты цвета берутся равнымим `r`, `g`, `b` - компонентам свойства `color`, если материал определяет это свойство (пример: MeshBasicMaterial.color). Если нет, то берутся значения по умолчанию: `1.0`.
`alpha` - значение прозрачности берётся равным значению свойства Material.opacity.

## threeColor()

Метод возвращает объект THREE.Color с параметрами цвета, соответствующими текущему `Color`.

```
threeColor(): THREE.Color;
```

## alpha()

Метод возвращает значение прозрачности.

```
alpha(): number;
```

## clone()

Метод возвращает копию текущего объекта.

```
clone(): Color;
```

## I3DRenderer

**I3DRenderer** – интерфейс, позволяющий взаимодействовать с программой отрисовки. Поодробнее THREE.WebGLRenderer.

```
export interface I3DRenderer {
  clear(color?: boolean, depth?: boolean, stencil?: boolean): void;
  clearDepth(): void;
  render(scene: THREE.Object3D, camera: THREE.Camera): void;
  getSize(target: THREE.Vector2): THREE.Vector2;
  setSize(width: number, height: number, updateStyle?: boolean): void;
  setViewport(x: THREE.Vector4 | number, y?: number, width?: number, height?:
number): void;
  getViewport(target: THREE.Vector4): THREE.Vector4;
  clippingPlanes: THREE.Plane[];
  domElement: HTMLCanvasElement;
}
```

### *Поля*

#### clippingPlanes

Глобальные секущие плоскости. Влияют на все операции отрисовки.

```
  clippingPlanes: THREE.Plane[];
```

#### domElement

Canvas на котором происходит отрисовка.

```
  domElement: HTMLCanvasElement;
```

### *Методы*

#### clear()

Метод очищает цветовой буффер, буффер глубины и буффер шаблона.

```
  clear(color?: boolean, depth?: boolean, stencil?: boolean): void;
```

где:
`color` – `true`, для очистки цветового буффера. По умолчанию: `true`.
`depth` – `true`, для очистки буффера глубины. По умолчанию: `true`.
`stencil` – `true`, для очистки буффера шаблона. По умолчанию: `true`.

#### clearDepth(): void

Метод очищает буффер глубины. Эквивалентно вызову .clear(false, true, false).

```
clearDepth(): void;
```

## render()

Метод отрисовывает THREE.Object3D с помощью камеры.

```
render(scene: THREE.Object3D, camera: THREE.Camera): void;
```

где:
`scene` – объект для отрисовки. Подробнее: THREE.Object3D.
`camera` – камера. Подробнее: THREE.Camera.

## getSize()

Метод возвращает ширину и высоту domElement в пикселях.

```
getSize(target: THREE.Vector2): THREE.Vector2;
```

где:
`target` – результат будет скопирован в этот THREE.Vector2.
Возвращает `target`.

## setSize()

Метод изменяет размер domElement, а также устанавливает область просмотра в соответствии с этим размером.

```
setSize(width: number, height: number, updateStyle?: boolean): void;
```

где:
`width` – ширина окна.
`height` – высота окна.
`updateStyle` – при значении `false` предотвращает любые изменения стиля domElement.

## setViewport()

Метод устанавливает область просмотра для отрисовки: от (x, y) до (x + width, y + height).

```
setViewport(x: THREE.Vector4 | number, y?: number, width?: number, height?:
number): void;
```

где:
`x` – x-координата левого нижнего угла окна, либо 4-компонентный вектор: THREE.Vector4, задающий параметры окна.
`y` – y-координата левого нижнего угла окна.
`width` – ширина окна.
`height` – высота окна.

## getViewport()

Метод возвращает размеры области отрисовки. Результат будет записан в `target` вектор.

```
getViewport(target: THREE.Vector4): THREE.Vector4;
```

где:
`target` – вектор с размерами области отрисовки. Подробнее: THREE.Vector4.
`x` – x-координата левого нижнего угла окна.
`y` – y-координата левого нижнего угла окна.
`z` или `width` – ширина окна.
`w` или `height` – высота окна.

## IntersectionChecker

**IIntersectionChecker** – базовый интерфейс обработки пересечений объектов. Подробнее: ISceneIntersectionChecker и IModelIntersectionChecker.

```
export interface IIntersectionChecker<TOptions extends
IntersectionCheckOptions> {
  getIntersectionPoint(): THREE.Intersection<THREE.Object3D> | undefined;
  getIntersectionByRay(ray: THREE.Ray, camera: THREE.Camera, options?:
TOptions): THREE.Intersection<THREE.Object3D> | undefined;
  getIntersectionIDByRay(ray: THREE.Ray, camera: THREE.Camera, options?:
TOptions): { modelId: string, guid: string } | undefined;
  getIntersectionByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera,
options?: TOptions): THREE.Intersection<THREE.Object3D> | undefined;
  getIntersectionIDByNdcPt(ndcPos: THREE.Vector2, camera: THREE.Camera,
options?: TOptions): { modelId: string, guid: string } | undefined;
  getIntersectionIDByFrustumNdcPt(ndcFrustumBox: THREE.Box3, unProjMatrix:
THREE.Matrix4, isContainsOnly: boolean, options?: TOptions): { modelId:
string; guid: string; }[] | undefined;
}
```

### *Методы*

### getIntersectionPoint()

Метод возвращает последнее рассчитанное пересечение модели.

```
  getIntersectionPoint(): THREE.Intersection<THREE.Object3D> | undefined;
```

Возвращает объект типа `THREE.Intersection`, если пересечение существует. В противном случае возвращает `undefined`.

### getIntersectionByRay()

Метод возвращает ближайшее пересечение объекта модели с лучом.

```
  getIntersectionByRay(ray: THREE.Ray, camera: THREE.Camera, options?:
TOptions): THREE.Intersection<THREE.Object3D> | undefined;
```

где:
`ray` – луч, с которым считаются пересечения. Подробнее: THREE.Ray.

`camera` – камера, используемая в отрисовке. Необходима для проверки пересечений с объектами, не зависящими от глубины кадра: спрайтами, текстовыми метками, точками замечаний и т.д.

`options` – параметры проверки пересечений. Необязательный параметр. Подробнее: SceneCheckOptions, ModelCheckOptions.
Возвращает объект типа `THREE.Intersection`, если пересечение существует. В противном случае возвращает `undefined`.

## getIntersectionIDByRay()

Метод возвращает `modelId` и `entityGuid` ближайшего объекта модели, пересекающегося с лучом.

```
  getIntersectionIDByRay(ray: THREE.Ray, camera: THREE.Camera, options?:
TOptions): { modelId: string, guid: string } | undefined;
```

где:
`ray` – луч, с которым считаются пересечения. Подробнее: THREE.Ray.

`camera` – камера, используемая в отрисовке. Необходима для проверки пересечений с объектами, не зависящими от глубины кадра: спрайтами, текстовыми метками, точками замечаний и т.д.

`options` – параметры проверки пересечений. Необязательный параметр. Подробнее: SceneCheckOptions, ModelCheckOptions.
Возвращает объект `{ modelId: string, guid: string }`, если пересечение существует. В противном случае возвращает `undefined`.

## getIntersectionByNdcPt()

Метод возвращает ближайшее пересечение объекта модели с лучом, выпущенным из точки нахождения камеры в направлении точки в Normalized Device Coordinates (NDC пространство).

```
  getIntersectionByNdcPt(ndcPoint: THREE.Vector2, camera: THREE.Camera,
options?: TOptions): THREE.Intersection<THREE.Object3D> | undefined;
```

где:
`ndcPoint` – 2D координаты точки в Normalized Device Coordinates (NDC пространство), в которую выпускается луч. Подробнее: THREE.Vector2.

`camera` – камера, используемая для определения положения начала луча и для перевода Normalized Device Coordinates (NDC пространство) в мировые координаты. Подробнее: THREE.Camera.

`options` – параметры проверки пересечений. Необязательный параметр. Подробнее: SceneCheckOptions, ModelCheckOptions.
Возвращает объект типа `THREE.Intersection`, если пересечение существует. В противном случае возвращает `undefined`.

## getIntersectionIDByNdcPt()

Метод возвращает `modelId` и `entityGuid` ближайшего объекта модели, пересекающегося с лучом, выпущенным из точки нахождения камеры в направлении точки в Normalized Device Coordinates (NDC пространство).

```
  getIntersectionIDByNdcPt(ndcPoint: THREE.Vector2, camera: THREE.Camera,
options?: TOptions): { modelId: string, guid: string } | undefined;
```

где:
`ndcPoint` – 2D координаты точки, в которую выпускается луч, в Normalized Device Coordinates (NDC пространство). Подробнее: THREE.Vector2.

`camera` – камера, используемая для определения положения начала луча и для перевода Normalized Device Coordinates (NDC пространство) в мировые координаты. Подробнее: THREE.Camera.

`options` – параметры проверки пересечений. Необязательный параметр. Подробнее: SceneCheckOptions, ModelCheckOptions.
Возвращает объект `{ modelId: string, guid: string }`, если пересечение существует. В противном случае возвращает `undefined`.

## getIntersectionIDByFrustumNdcPt()

Метод возвращает список `modelId` и `entityGuid` объектов модели, пересекаемых усечённой пирамидой (Frustum).

```
  getIntersectionIDByFrustumNdcPt(ndcFrustumBox: THREE.Box3, unProjMatrix:
THREE.Matrix4, isContainsOnly: boolean, options?: TOptions): { modelId:
string; guid: string; }[];
```

где:
`ndcFrustumBox` – представление `Frsutum` в Normalized Device Coordinates (NDC пространство). Подробнее: THREE.Box3.

`unProjMatrix` – матрица проекции координат Normalized Device Coordinates (NDC пространство) в мировые координаты. Подробнее: THREE.Matrix4.

`isContainsOnly` – если `true`, то отбрасываются не полностью содержащиеся внутри пирамиды объекты. В противном случае в вывод включаются как содержащиеся внутри пирамиды объекты, так и касающиеся или пересекающиеся с ней.
Если при расчёте учитываются секущие плоскости, то `isContainsOnly` также указывает на то, что объект не должен быть обрезан секущими.

`options` – параметры проверки пересечений, необязательный параметр. Подробнее: SceneCheckOptions, ModelCheckOptions.

Возвращает список объектов `{ modelId: string, guid: string }`, если пересечение существует. В противном случае возвращает пустой массив.

Пример расчёта пересечений с `Frustum`, образованным областью видимости камеры:

```
  const camera = viewer3D.navigation.getCamera();
  const unprojectionMatrix = new THREE.Matrix4();
  const ndcFrustumBox = new THREE.Box3();
  //матрица проекции из Normalized Device Coordinates (NDC пространство) в
мировые координаты:
  unprojectionMatrix.multiplyMatrices(camera.matrixWorld,
camera.projectionMatrixInverse);
```

```
  //левый нижний угол усечённой пирамиды в Normalized device coordinates (NDC
пространство: x=-1, y=-1), ближняя плоскость камеры (nearPlane): z = -1:
  ndcFrustumBox.min = new THREE.Vector3(-1, -1, -1);
  //правый верхний угол усечённой пирамиды в Normalized device coordinates
(NDC пространство: x=1, y=1), дальняя плоскость камеры (farPlane): z = 1:
  ndcFrustumBox.max = new THREE.Vector3(1, 1, 1);

  //Находим все объекты, содержащиеся внутри и/или пересекающие усечённую
пирамиду, без учета секущих плоскостей.
  const intersections =
intersectionChecker.getIntersectionIDByFrustumNdcPt(ndcFrustumBox,
unprojectionMatrix, false, false);
```

Более сложный пример использования: BoxSelectionExtension.

## ISceneIntersectionChecker

**ISceneIntersectionChecker** – интерфейс обработки пересечений объектов на отдельной сцене. Расширяет `IIntersectionChecker`.

```
  export interface ISceneIntersectionChecker extends
IIntersectionChecker<SceneCheckOptions> {
    //Возвращает ограничивающий объём сцены.
    get boundingBox(): THREE.Box3;
  }
```

### SceneCheckOptions

**SceneCheckOptions** – опции проверки пересечений для `ISceneIntersectionChecker`.

```
  export interface SceneCheckOptions {
    //Определяет, учитываются ли секущие плоскости при расчёте пересечений на
данной сцене.
    //Если учитываются, то пересечения с объектами за пределами секущего
объёма отбрасываются.
    filterByClipping?: boolean;
  }
```

## IModelIntersectionChecker

**IModelIntersectionChecker** – интерфейс обработки пересечений всех объектов модели. Расширяет `IIntersectionChecker`.

```
  export interface IModelIntersectionChecker extends
IIntersectionChecker<ModelCheckOptions> {
    // Возвращает ограничивающий объём всей модели.
    get boundingBox(): THREE.Box3;
    // Возвращает центр ограничивающего объёма модели.
    get modelCenter(): THREE.Vector3;
  }
```

### ModelCheckOptions

**ModelCheckOptions** – опции проверки пересечений для `IModelIntersectionChecker`.

```
export interface ModelCheckOptions extends SceneCheckOptions {
  // Наименования сцен, которые участвуют в проверке пересечений.
  //  Если не задано, то пересечения проверяются для всех сцен.
  sceneNames?: string[],
}
```

## IRenderOperationContext

**IRenderOperationContext** – интерфейс, представляющий контекст операции рендера.

```
export interface IRenderOperationContext {
  isRedrawRequested: boolean;
  isForcedExecution: boolean;

  get renderer(): I3DRenderer;
  get camera(): THREE.Camera;
  get settings(): RenderViewSettings;
  get isNavigation(): boolean;
  get isSuspensionRequested(): boolean;
  get remainedTime(): DOMHighResTimeStamp;
  get elapsedTime(): DOMHighResTimeStamp;
  get lastFrameTimestamp(): DOMHighResTimeStamp;
  get lastRenderCycleTimestamp(): DOMHighResTimeStamp;
  get isElapsed(): boolean;
  get userData(): Map<string, object>;
}
```

### *Поля*

#### isRedrawRequested

Задает принудительную перерисовку сцен.

```
isRedrawRequested: boolean;
```

#### isForcedExecution

Снимает ограничение на время исполнения операций рендера.
Если `true`, операции исполняются принудительно, время исполнения не ограничено.

```
isForcedExecution: boolean;
```

### *Свойства*

#### get renderer()

Возвращает объект I3DRenderer, используемый для отрисовки сцен.

```
  get renderer(): I3DRenderer;
```

Возвращает объект I3DRenderer.

#### get camera()

Возвращает камеру, используемую для отрисовки сцен.

```
get camera(): THREE.Camera;
```

Возвращает объект THREE.Camera.

### get settings()

Возвращает текущие настройки рендера.

```
get settings(): RenderViewSettings;
```

Возвращает настройки рендера.

### get isNavigation()

Показывает производится ли в данный момент навигация по сцене.

```
get isNavigation(): boolean;
```

Возвращает `true`, если навигация активна. В противном случае возвращает `false`.

### get isSuspensionRequested()

Показывает что необходимо приостановить текущую операцию рендера и вернуть управление планировщику.

```
get isSuspensionRequested(): boolean;
```

Возвращает `true`, если требуется приостановить операцию. В противном случае возвращает `false`.
Если isForcedExecution равен `true`, то isSuspensionRequested всегда возвращает `false`.

### get remainedTime()

Сообщает сколько выделенного времени осталось на выполнение операции.

```
get remainedTime(): DOMHighResTimeStamp;
```

Возвращает DOMHighResTimeStamp - количество оставшегося времени на выполнение.

### get elapsedTime()

Сообщает сколько времени затрачено на выполнение операций в текущей итерации.

```
get remainedTime(): DOMHighResTimeStamp;
```

Возвращает DOMHighResTimeStamp - затраченное время текущей итерации.

## get lastFrameTimestamp()

Хранит время отрисовки последнего кадра.

```
get lastFrameTimestamp(): DOMHighResTimeStamp;
```

Возвращает DOMHighResTimeStamp - время последней отрисовки.

## get lastRenderCycleTimestamp()

Хранит время завершения последнего цикла рендера.

```
get lastRenderCycleTimestamp(): DOMHighResTimeStamp;
```

Возвращает DOMHighResTimeStamp - время последнего цикла рендера.

## get isElapsed()

Показывает закончилось ли время текущей итерации, выделенное на выполнение операций рендера.

```
get isElapsed(): boolean;
```

Возвращает `true`, если выделенное время кончилось. В противном случае возвращает `false`.
В случае, если isElapsed равен `true`, а isForcedExecution равен `false`, то isSuspensionRequested вернёт `true`.

## get userData()

Данные, совместно используемые операциями в одном цикле рендера.

```
get userData(): Map<string, object>;
```

Возвращает словарь `Map<string, object>`.

## IRenderViewer3D

**IRenderViewer3D** – интерфейс для работы с отрисовкой сцен и объектами на сцене.

```
export interface IRenderViewer3D {
  getIntersectionChecker(): IModelIntersectionChecker;
  updateCurrentCanvas(): Promise<void>;
  placeObjectOnScene(iObj: THREE.Object3D, sceneID?: string): Promise<void>;
  removeObjectFromScene(iObj: THREE.Object3D): Promise<void>;
  setClipping(planes: THREE.Plane[], sceneID?: string): void;
  setActiveClipPlaneIndices(indices: number[]): void;
  getScenes(): IUserScene[];
  addScene(name: string, isClippable: boolean): IUserScene;
  removeScene(scene: IUserScene): void;
}
```

### *Методы*

### getIntersectionChecker()

Предоставляет IModelIntersectionChecker – интерфейс обработки пересечений для всех сцен.

```
getIntersectionChecker(): IModelIntersectionChecker;
```

Возвращает интерфейс обработки пересечений. Подробнее: IModelIntersectionChecker.

### updateCurrentCanvas()

Метод вызывает полную перерисовку сцен.
Если параметр `force` установлен в `true`, то отрисовка кадра произойдёт после принудительного обновления сцен.

```
updateCurrentCanvas(force?: boolean): Promise<void>;
```

где:
`force` – флаг, указывающий на принудительное обновление сцен. По умолчанию – `false`.

### placeObjectOnScene()

Метод помещает объект на определённую сцену.

```
 placeObjectOnScene(iObj: THREE.Object3D, sceneID?: string): Promise<void>;
```

где:
`iObj` – объект, который нужно поместить на сцену.
`sceneID` – имя сцены. Необязательный параметр. По умолчанию – `MainScene`.

### removeObjectFromScene()

Метод удаляет объект со сцены.

```
removeObjectFromScene(iObj: THREE.Object3D): Promise<void>;
```

где:
`iObj` – объект, который нужно удалить.

## setClipping()

Метод задает секущие плоскости для визулизации.

```
setClipping(planes: THREE.Plane[]): void;
```

где:
`planes` – список плоскостей сечения. Подробнее: THREE.Plane.

## setActiveClipPlaneIndices()

Метод задает активные плоскости сечения. Сечения активных плоскостей окрашены в голубой цвет.

```
setActiveClipPlaneIndices(indices: number[]): void;
```

где:
`indices` – список индексов активных плоскостей сечения. Каждый индекс указывает на плоскость в списке, переданном в setClipping().

## getScenes()

Метод возвращает список всех используемых сцен.

```
getScenes(): IUserScene[];
```

Возвращает список сцен. Подробнее: IUserScene.

## addScene()

Метод добавляет новую сцену для отрисовки.

```
addScene(name: string, isClippable: boolean): IUserScene;
```

где:
`name` – идентификатор новой сцены.

`isClippable` – параметр, указывающий, влияют ли секущие плоскости на отрисовку этой сцены. Если `false`, то секущие плоскости не применяются к объектам на этой сцене и проверка пересечений с объектами на этой сцене также не учитывает секущие плоскости. Возвращает добавленную сцену. Подробнее: IUserScene.

## removeScene()

Метод удаляет сцену из списка используемых для отрисовки сцен.

```
removeScene(scene: IUserScene): void;
```

где:
`scene` – сцена для удаления. Подробнее: IUserScene.

## IUserScene

**IUserScene** – интерфейс, позволяющий взаимодействовать со сценой. Расширяет THREE.Scene

```
export interface IUserScene extends THREE.Scene {
  readonly name: string;
  get needsUpdate(): boolean;
  get needsRedraw(): boolean;
  get intersectionChecker(): ISceneIntersectionChecker;
  set clippingEnable(value: boolean);
  get clippingEnable(): boolean;

  addRange(objects: THREE.Object3D[]): void;
  updateRange(objects: TPair<THREE.Object3D, UpdateType>[]): void;
  removeRange(objects: THREE.Object3D[]): void;
  has(obj: THREE.Object3D): boolean;
  traverse(callback: (object: THREE.Object3D) => void): void;
  setClipping(planes: THREE.Plane[]): void;
  manageScene(context?: IRenderOperationContext): boolean;
  render(context: IRenderOperationContext): void;
}
```

### *Поля*

#### readonly name

Имя сцены.

```
readonly name: string;
```

### *Свойства*

#### get needsUpdate()

Показывает, нужно ли обновить сцену.

```
 get needsUpdate(): boolean;
```

Возвращает `true`, если нужно обновить сцену, в противном случае – `false`.

#### get needsRedraw()

Показывает, нужно ли перерисовать сцену.

```
  get needsRedraw(): boolean;
```

Возвращает `true`, если нужно перерисовать сцену, в противном случае – `false`.

## get intersectionChecker()

Предоставляет ISceneIntersectionChecker – интерфейс обработки пересечений для данной сцены.

```
get intersectionChecker(): ISceneIntersectionChecker | null;
```

Если проверка пересечений на сцене поддерживается, то возвращается интерфейс обработки пересечений. В противном случае возвращается `null`. Подробнее: ISceneIntersectionChecker.

## get clippingEnable()

Показывает, влияют ли секущие плоскости на отрисовку и проверку пересечений на данной сцене.

```
get clippingEnable(): boolean;
```

Возвращает `true`, если секущие плоскости влияют на отрисовку и проверку пересечений на данной сцене.

## set clippingEnable()

Включает или выключает влияние секущих плоскостей на отрисовку и проверку пересечений на данной сцене.

```
set clippingEnable(value: boolean);
```

где:
`value` – параметр. Если `value` равен `true`, то объекты на сцене обрезаются секущими плоскостями, а также при проверке пересечений на данной сцене не учитываются отсечённые объекты.

### *Методы*

## addRange()

Метод добавляет список объектов на сцену.

```
addRange(objects: THREE.Object3D[]): void;
```

где:
`objects` – список объектов для добавления на сцену. Подробнее: THREE.Object3D.

## updateRange()

Метод обновляет объекты на сцене.

```
updateRange(objects: TPair<THREE.Object3D, UpdateType>[]): void;
```

где:
`objects` – список объектов для обновления. Каждый элемент списка является парой из самого объекта и соответствующего ему типа обновления. Подробнее: THREE.Object3D, UpdateType.

### removeRange()

Метод удаляет список объектов со сцены.

```
removeRange(objects: THREE.Object3D[]): void;
```

где:
`objects` – список объектов для удаления. Подробнее: THREE.Object3D.

### has()

Метод показывает, добавлен ли объект на сцену.

```
has(obj: THREE.Object3D): boolean;
```

где:
`objects` – проверяемый объект. Подробнее: THREE.Object3D.
Возвращает `true`, если объект добавлен на сцену. В противном случае – `false`.

### traverse()

Метод осуществляет перебор объектов на сцене.

```
traverse(callback: (object: THREE.Object3D) => void): void;
```

где:
`callback` – функция, вызываемая для всех объектов на сцене.

### setClipping()

Метод задает плоскости сечения для данной сцены.

```
setClipping(planes: THREE.Plane[]): void;
```

где:
`planes` – список плоскостей сечения. Подробнее: THREE.Plane.

### manageScene()

Метод передает управление сцене для выполнения внутренних операций: обработки изменений на сцене, оптимизации отрисовки, оптимизации проверки пересечений и т.д.

```
manageScene(context?: IRenderOperationContext): boolean;
```

где:
`context` – контекст операции рендера. Подробнее: IRenderOperationContext. Возвращает `true`, если все запланированные операции на сцене были выполнены. Возвращает `false`, если требуется повторная передача управления.

## render()

Метод выполняет отрисовку сцены в данном контексте.

```
render(context?: IRenderOperationContext): void;
```

где:
`context` – контекст операции рендера. Подробнее: IRenderOperationContext.

## LabelSprite

**LabelSprite** – класс, описывающий 2D текст на сцене. Расширяет THREE.Sprite.

```
export class LabelSprite extends THREE.Sprite {
   constructor(parameters: LabelSpriteParameters);

  get sizeAttenuation(): boolean;
  set sizeAttenuation(value: boolean);

  get text(): string;
  set text(value: string);

  get fontFace(): string | FontFace;
  set fontFace(value: string | FontFace);

  get fontSize(): number;
  set fontSize(value: number);

  get borderThickness(): number;
  set borderThickness(value: number);

  get borderColor(): Color;
  set borderColor(value: Color);

  get borderRadius(): number;
  set borderRadius(value: number);

  get backgroundColor(): Color;
  set backgroundColor(value: Color);

  get textColor(): Color;
  set textColor(value: Color);

  get textPadding(): THREE.Vector4Tuple;
  set textPadding(value: THREE.Vector4Tuple);

  dispose(): void;
}
```

### *Конструктор*

```
  constructor(parameters: LabelSpriteParameters);
```

где:
`parameters`– параметры текстовой метки. Подробнее: LabelSpriteParameters.

### *Свойства*

#### sizeAttenuation: boolean

Задает или возвращает значение флага, показывающего влияние перспективы на текстовую метку - уменьшается ли размер метки с глубиной кадра. Влияет на отображение только с перспективной камерой.

```
get sizeAttenuation(): boolean;
set sizeAttenuation(value: boolean);
```

По умолчанию: `false` - размер метки не зависит от перспективы.

### text: string

Задает или возвращает строку отображаемого текста.

```
get text(): string;
set text(value: string);
```

По умолчанию пустая строка.

### fontFace: string | FontFace

Задает или возвращает параметры шрифта.

```
get fontFace(): string | FontFace;
set fontFace(value: string | FontFace);
```

где:
`fontFace` – строка задающая font-family, или объект FontFace описывающий параметры шрифта.
По умолчанию: `Roboto, sans-serif`.

### fontSize: number

Задает или возвращает размер отображаемого текста в пикселях.

```
get fontSize(): number;
set fontSize(value: number);
```

По умолчанию: `12`.

### borderThickness: number

Задает или возвращает толщину рамки метки в пикселях.

```
get borderThickness(): number;
set borderThickness(value: number);
```

По умолчанию: `2`.

### borderColor: Color

Задает или возвращает цвет рамки метки.

```
get borderColor(): Color;
set borderColor(value: Color);
```

По умолчанию: `Color(0, 0, 0, 1.0)`. Подробнее: Color.

### borderRadius: number

Задает или возвращает радиус скругления рамки метки в пикселях.

```
get borderRadius(): number;
set borderRadius(value: number);
```

По умолчанию: `1`.

### backgroundColor: Color

Задает или возвращает цвет фона метки.

```
get backgroundColor(): Color;
set backgroundColor(value: Color);
```

По умолчанию: `Color(1.0, 1.0, 1.0, 1.0)`. Подробнее: Color.

### textColor: Color

Задает или возвращает цвет отображаемого текста.

```
get textColor(): Color;
set textColor(value: Color);
```

По умолчанию: `Color(0, 0, 0, 1.0)`. Подробнее: Color.

### textPadding: THREE.Vector4Tuple

Задает или возвращает отступ текста от границ метки. Отступ задается в пикселях по порядку: слева, сверху, справа, снизу.

```
get textPadding(): THREE.Vector4Tuple;
set textPadding(value: THREE.Vector4Tuple);
```

По умолчанию: `[0, 0, 0, 0]`.

#### *Методы*

### dispose()

Метод освобождает ресурсы, выделенные `LabelSprite`.

```
dispose(): void;
```

### *LabelSpriteParameters*

Параметры текстовой метки. Задают соответствующие свойства `LabelSprite`.

```
export interface LabelSpriteParameters {
  text?: string | undefined;
  sizeAttenuation?: boolean | undefined;
  fontFace?: string | FontFace | undefined;
  fontSize?: number | undefined;
  borderColor?: Color | undefined;
  backgroundColor?: Color | undefined;
  borderThickness?: number | undefined;
  borderRadius?: number | undefined;
  textColor?: Color | undefined;
  textPadding?: THREE.Vector4Tuple | undefined;
}
```

## MeshLine

**MeshLine** – класс описывающий линии для отрисовки в виде полигональной сетки. Расширяет THREE.Mesh.

```
export class MeshLine extends THREE.Mesh {
  override material: MeshLineMaterial;
  override geometry: MeshLineGeometry;

 constructor(geometry?: MeshLineGeometry, material?: MeshLineMaterial);
}
```

### *Конструктор*

```
  constructor(geometry?: MeshLineGeometry, material?: MeshLineMaterial);
```

где:
`geometry` – геометрия, опциональный параметр. Если не задано, используется пустая геометрия.
`material` – материал линий, опциональный параметр. Если не задано, используется материал по умолчанию.

### *Поля*

#### material: MeshLineMaterial

Материал линий. Подробнее MeshLineMaterial.

```
  material: MeshLineMaterial;
```

#### geometry: MeshLineGeometry

Геометрия линий. Подробнее MeshLineGeometry.

```
  geometry: MeshLineGeometry;
```

## MeshLineMaterial

**MeshLineMaterial** – материал для отрисовки линий в виде полигональной сетки.

```
export class MeshLineMaterial extends THREE.ShaderMaterial {
  constructor(parameters: MeshLineMaterialParameters);

  get linewidth(): number;
  set linewidth(value: number);
```

```
  get opacity(): number;
  set opacity(value: number);

  get worldUnits(): boolean;
  set worldUnits(value: boolean);

  get dashed(): boolean;
  set dashed(value: boolean);

  get resolution(): THREE.Vector2;

  color: THREE.Color;

  dashScale: number;

  dashSize: number;

  dashOffset: number;

  gapSize: number;
}
```

### *Конструктор*

```
  constructor(parameters: MeshLineMaterialParameters);
```

где:
`parameters` – параметры материала. Подробнее MeshLineMaterialParameters.

### *Свойства*

## worldUnits: boolean

Определяет в каком пространстве задана ширина линий. Если `true`, то ширина линий задана в мировых координтах и на линию будет влиять перспектива - размер линии будет уменьшаться с глубиной кадра. В противном случае, размер линии считается в пикселях и остаетя неизменным, переспектива не оказывает влияния на ширину линиии.

```
  get worldUnits(): boolean;
  set worldUnits(value: boolean);
```

По умолчанию: `true`.

## dashed: boolean

Определяет является ли линия пунктирной.
Для отрисовки пунктирной линии также необходимо вычислить расстояния между точками в `MeshLineGeometry`. Подробнее: MeshLineGeometry.computeLineDistances.

```
  get dashed(): boolean;
  set dashed(value: boolean);
```

По умолчанию: `false`.

### resolution: THREE.Vector2

Задает и получает размеры области отрисовки. Нужно для корректной отрисовки линий в `worldUnits = false` режиме. Данное свойство обновляется автоматически перед отрисовкой.

```
get resolution(): THREE.Vector2;
set resolution(value: THREE.Vector2);
```

Подробнее: THREE.Vector2.

#### *Поля*

### color: THREE.Color

Определяет значение цвета линий. Общий цвет для всех линий применяется если в материале свойство `vertexColors` установлено в `false` (значение по умолчанию).

```
color: THREE.Color;
```

По умолчанию: `new THREE.Color(0xffffff)`. Подробнее: Color.

### dashScale: boolean

Определяет коэффициент-делитель для пунктира. Длина пунктирного отрезка (длина штриха + длина пустого промежутка) делится на данный коэффициент.
Например значение `dashScale = 2` приведёт к уменьшению длины пунктирного отрезка в два раза и соответственному увеличению частоты штрихов в линии, тоже в два раза.

```
dashScale: number;
```

По умолчанию: `1`.

### dashSize: boolean

Определяет длину штриха в пунктирном отрезке.

```
dashSize: number;
```

По умолчанию: `1`.

### dashOffset: boolean

Определяет смещение штриха в пунктирном отрезке.

```
  dashOffset: number;
```

По умолчанию: `0`.

### gapSize: boolean

Определяет длину пустого промежутка в пунктирном отрезке.

```
  gapSize: number;
```

По умолчанию: `1`.

#### *MeshLineMaterialParameters*

Параметры `MeshLineMaterial`. Задают соответствующие свойства MeshLineMaterial.

```
export interface MeshLineMaterialParameters extends
THREE.ShaderMaterialParameters {
  color?: THREE.ColorRepresentation | undefined;
  dashed?: boolean | undefined;
  dashScale?: number | undefined;
  dashSize?: number | undefined;
  dashOffset?: number | undefined;
  gapSize?: number | undefined;
  linewidth?: number | undefined;
  resolution?: THREE.Vector2 | undefined;
  worldUnits?: boolean | undefined;
}
```

## MeshLineGeometry

Геометрия, используемая для отрисовки линий в виде полигональной сетки. Линии задаются указанием точек.

```
export class MeshLineGeometry extends THREE.InstancedBufferGeometry {

  constructor();

  setPoints(points: THREE.Vector3[]): this;

  setPositions(array: ArrayLike<number>): this;

  setColors(array: ArrayLike<number>): this;

  computeLineDistances(): this;

  updatePoint(index: number, point: THREE.Vector3): void;

  updateColor(index: number, color: Color): void;
}
```

*Методы*

## setPoints()

Метод задает точки линии.

```
setPoints(points: THREE.Vector3[]): this;
```

где:
`points` – точки линии. Подробнее: THREE.Vector3.

При вызове этого метода происходит перестроение атрибутов геометрии, что является ресурсозатратной операцией. Поэтому, для изменения координат точек, если количество точек не изменяется, следует использовать метод updatePoint.
Если нужно отрисовать пунктирную линию, то следует пересчитать расстояния между точками с помощью computeLineDistances.

## setPositions()

Метод задает точки линии из координат переданных в виде массива.

```
setPositions(array: ArrayLike<number>): this;
```

где:
`array` – массив с координатами точек. Координаты расположены последовательно, по 3 элемента на точку: `[ x0, y0, z0, x1, y1, z1, ... ]`.

При вызове этого метода происходит перестроение атрибутов геометрии, что является ресурсозатратной операцией. Поэтому, для изменения координат точек, если количество точек не изменяется, следует использовать метод updatePoint.
Если нужно отрисовать пунктирную линию, то следует пересчитать расстояния между точками с помощью computeLineDistances.

## setColors()

Метод задает цвета отдельных участков линии. Цвета действуют в окрестностях точек к которым применяются, то есть все отрезки между точками будут поделены пополам и каждая половина будет окрашена в цвет ближайшей примыкающей точки. Если цвета не заданы, то для всей линии используется цвет материала: MeshLineMaterial.color.
Цвета применяются, только если в `MeshLineMaterial` свойство `vertexColors` установлено в `true`.

```
setColors(array: ArrayLike<number>): this;
```

где:
`array` – массив со значенями цветов точек. Значения расположены последовательно, по 4 элемента на точку: `[ r0, g0, b0, a0, r1, g1, b1, a1, ... ]`.

При вызове этого метода происходит перестроение атрибутов геометрии, что является ресурсозатратной операцией. Поэтому, для изменения цветов точек, если количество точек не изменяется, следует использовать метод updateColor.

## computeLineDistances()

Метод вычисляет расстояния между точками и сохраняет значения в атрибуты геометрии. Необходимо только для отрисовки пунктирных линий.

```
computeLineDistances(): this;
```

## updatePoint()

Метод обновляет координаты точки, без перестроения всей геометрии.

```
updatePoint(index: number, point: THREE.Vector3): void;
```

где:
`index` – порядковый номер точки.
`point` – обновлённые координаты точки. Подробнее: THREE.Vector3.

Если нужно отрисовать пунктирную линию, то следует пересчитать расстояния между точками с помощью computeLineDistances.

## updateColor()

Метод обновляет цвет точки, без перестроения всей геометрии.

```
updateColor(index: number, color: Color): void;
```

`index` – порядковый номер точки.
`color` – обновлённый цвет точки. Подробнее: Color.

## UpdateType

**UpdateType** – флаговое перечисление, описывающее типы изменений ViewObject.

```
export enum UpdateType {
  /**No changes */
  None = 0,
  /**Object has been removed from the scene */
  Remove = 1 << 0,
  /**Object has been added to the scene */
  Add = 1 << 1,
  /**Object has been significantly changed. Re-insert required */
  Hard = 1 << 2,
  /**Child objects has been changed*/
  Children = 1 << 3,
  /**Object geometry has been changed */
  Geometry = 1 << 4,
  /**Object position has been changed */
  Position = 1 << 5,
  /**Object material has been changed */
  Material = 1 << 6,
  /**Object color has been changed */
  Color = 1 << 7,
  /**Object selection status has been changed */
  Selection = 1 << 8,
  /**Object visibility has been changed */
  Visibility = 1 << 9,
}
```

## ViewObject

**ViewObject** – абстрактный класс, описывающий объект на сцене. Расширяет THREE.Object3D.

```
export abstract class ViewObject extends THREE.Object3D {
  readonly entityGuid: string;
  readonly modelGuid: string;

  constructor(entityGuid?: string, modelGuid?: string, color?: Color);

  abstract get mesh(): THREE.Mesh | null;
  abstract get edges(): THREE.LineSegments | null;
  setVisible(value: boolean): void;
  isVisible(): boolean;
  setHidden(value: boolean): void;
  isHidden(): boolean;
  setSelected(value: boolean): void;
  isSelected(): boolean;
  setHovered(value: boolean): void;
  isHovered(): boolean;
  setColor(color: Color): void;
  getColor(): Color;
  resetColor(): void;
  getOriginalColor(): Color;
  isGhosted(): boolean;
  setGhosted(value: boolean): void;
  dispose(): void;

  getBoundingBox(): THREE.Box3;
  raycast(iRaycaster: THREE.Raycaster, oIntersects: THREE.Intersection[]):
void;

  protected setHoveredForObject(value: boolean): void;
  protected setSelectedForObject(value: boolean): void;
  protected setHiddenForObject(value: boolean): void;
  protected setVisibleForObject(value: boolean): void;
  protected setColorForObject(color: Color): void;
  protected setGhostModeForObject(value: boolean): void;
  protected resetColorForObject(): void;

  protected riseOnUpdated(updateType?: UpdateType, object?: THREE.Object3D):
void;
}
```

### *Конструктор*

```
  constructor(entityGuid?: string, modelGuid?: string, color?: Color);
```

где:

`entityGuid`– идентификатор объекта модели, которому соответствует `ViewObject`. Опциональный параметр. Если не задан, то объекту присваивается нулевой guid. Подробнее: ModelElement.id.

`modelGuid` – идентификатор части модели, к которой относится объект модели. Опциональный параметр. Если не задан, то объекту присваивается нулевой guid. Подробнее: ModelElement.modelPartId.

`color` – начальный цвет объекта. Опциональный параметр. Если не задан, то объекту присваивается цвет по умолчанию – `new Color(1, 1, 1, 1)`. Подробнее: Color.

### *Поля*

## entityGuid

Идентификатор элемента модели, соответствующего этому `ViewObject`. Подробнее: ModelElement.id.

```
readonly entityGuid: string;
```

Значение по умолчанию: `00000000-0000-0000-0000-000000000000`.

## modelGuid

Идентификатор части модели, к которой относится элемент модели, соответствующий этому `ViewObject`. Подробнее: ModelElement.modelPartId.

```
readonly entityGuid: string;
```

Значение по умолчанию: `00000000-0000-0000-0000-000000000000`.

### *Свойства*

## get mesh()

Геометрическое представление `ViewObject` в виде `THREE.Mesh`. Подробнее: THREE.Mesh. Используется для отрисовки объекта при выделении (Hover/Select) и при расчёте пересечений с `Frustum`.

```
abstract get mesh(): THREE.Mesh | null;
```

## get edges()

Геометрическое представление `ViewObject` в виде `THREE.LineSegments`. Подробнее: THREE.LineSegments.
Используется для отрисовки объекта при выделении (Hover/Select) и при расчёте пересечений с `Frustum`.

```
abstract get edges(): THREE.LineSegments | null;
```

*Методы*

## setVisible()

Метод управляет видимостью объекта на сцене.

```
setVisible(value: boolean): void;
```

где:

`value` – видимость объекта.

## isVisible()

Метод проверяет видимость объекта на сцене.

```
isVisible(): boolean;
```

Возвращает `true`, если объект виден. В противном случае возвращает `false`.

## setHidden()

Метод управляет скрытием объекта со сцены. Скрытие объектов используется только для ускорения отрисовки сцены и не влияет на проверку пересечений.

```
setHidden(value: boolean): void;
```

где:

`value` – скрыт ли объект.

## isHidden()

Метод проверяет видимость объекта на сцене.

```
isHidden(): boolean;
```

Вовзращает `true`, если объект скрыт. В противном случае возвращает `false`.

## setSelected()

Метод управляет селектированием объекта.

```
setSelected(value: boolean): void;
```

где:

`value` – выбран ли объект.

### isSelected()

Метод проверяет селектирование объекта.

```
isSelected(): boolean;
```

Возвращает `true`, если объект выбран. В противном случае возвращает `false`.

### setHovered()

Метод управляет ховером объекта.

```
setHovered(value: boolean): void;
```

где:

`value` – активность ховера над объектом.

### isHovered()

Метод проверяет активность ховера над объектом.

```
isHovered(): boolean;
```

Возвращает `true`, если ховер активен. В противном случае возвращает `false`.

### setColor()

Метод задаёт цвет объекта.

```
setColor(color: Color): void;
```

где:

`color` – цвет объекта. Подробнее: Color.

### getColor()

Метод возвращает текущий цвет объекта. Подробнее: Color.

```
getColor(): Color;
```

### resetColor()

Сбрасывает цвет объекта на изначальный.

```
resetColor(): void;
```

### getOriginalColor()

Метод возвращает изначальный цвет объекта.

```
getOriginalColor(): Color;
```

Возвращает объект типа Color.

## isGhosted()

Метод проверяет, находится ли объект в призрачном режиме отрисовки.

```
isGhosted(): boolean;
```

## setGhosted()

Метод задает призрачный режим отрисовки объекта – объект рисуется бесцветным и полупрозрачным.

```
setGhosted(value: boolean): void;
```

где:

`value` – задаёт активность призрачного режима.

## dispose()

Метод освобождает ресурсы, выделенные `ViewObject`.

```
dispose(): void;
```

## getBoundingBox()

Метод возвращает граничный объём `ViewObject`.
Реализация по умолчания возвращает пустой `THREE.Box3`. Метод доступен для переопределения.

```
getBoundingBox(): THREE.Box3;
```

Возвращает объект типа THREE.Box3.

## raycast()

Метод вычисляет пересечение `ViewObject` с лучом.
Для расчета пересечений на сцене должен быть переопределён в подклассах.
Метод пустой по умолчанию, доступен для переопределения. Подробнее: THREE.Object3D.raycast

```
raycast(iRaycaster: THREE.Raycaster, oIntersects: THREE.Intersection[]): void;
```

## protected setHoveredForObject()

Метод описывает поведение объекта при ховере.
Метод пустой по умолчанию, доступен для переопределения.

```
protected setHoveredForObject(value: boolean): void;
```

где:

`value` – активность ховера над объектом.

## protected setSelectedForObject()

Метод описывает поведение объекта при селекте.
Метод пустой по умолчанию, доступен для переопределения.

```
protected setSelectedForObject(value: boolean): void;
```

где:

`value` – значение селекта над объектом.

## protected setHiddenForObject()

Метод описывает поведение объекта при скрытии.
Реализация по умолчанию работает только для объектов на основной сцене `MainScene`.
Метод доступен для переопределения.

```
protected setHiddenForObject(value: boolean): void;
```

где:

`value` – видимость объекта на сцене.

## protected setVisibleForObject()

Метод описывает поведение объекта при изменении видимости.
Реализация по умолчанию использует свойство Object3D.visibility. Метод доступен для переопределения.

```
protected setVisibleForObject(value: boolean): void;
```

где:

`value` – видимость объекта на сцене.

## protected setColorForObject()

Метод описывает поведение объекта при изменении цвета.
Метод пустой по умолчанию, доступен для переопределения.

```
protected setColorForObject(color: Color): void;
```

где:

`color` – цвет объекта. Подробнее: Color.

## protected setGhostModeForObject()

Метод описывает поведение объекта в призрачном режиме.
Метод пустой по умолчанию, доступен для переопределения.

```
protected setGhostModeForObject(value: boolean): void;
```

где:

`value` – задаёт активность призрачного режима.

## protected resetColorForObject()

Метод описывает поведение объекта при сбрасывании цвета объекта на изначальный. Реализация по умолчанию использует setColorForObject(originalColor). Метод доступен для переопределения.

```
protected resetColorForObject(): void;
```

## protected riseOnUpdated()

Метод сообщает подписчикам об изменении объекта. Подробнее: EventDispatcher.

```
  protected riseOnUpdated(updateType?: UpdateType, object?: THREE.Object3D):
void;
```

где:
`updateType` – тип обновления. Подробнее: UpdateType.
`object` – объект, источник обновления. По умолчанию – текущий объект.

```
// При изменении дочерних объектов, которые не являются `ViewObject`
// для оповещения об их изменениях можно использовать `riseOnUpdated`
родительского объекта.

// При изменении видимости текущего объекта:
this.visible = false;
// Оповещаем об изменениях:
this.riseOnUpdated(UpdateType.Visibility);
// Эквивалентно вызову:
this.dispatchEvent({ type: 'update', updateType: UpdateType.Visibility });

// При изменении видимости дочернего объекта:
childMesh.visibility = false;
```

```
// Оповещаем об изменениях:
this.riseOnUpdated(UpdateType.Visibility, childMesh);
// Эквивалентно вызову:
childMesh.dispatchEvent({ type: 'update', updateType: UpdateType.Visibility
});
```

# SelectionMode

## Режим выбора элементов

```
enum SelectionMode {
  // Добавить новые элементы в уже выбранные
  Append = 0,
  // Заменить выбранные элементы новыми
  Replace = 1
}
```

# User Interface

## ButtonBuilder

**ButtonBuilder** – это класс строителя элемента управления **Кнопка**.

### *Методы*

### withCaption()

Метод задает подпись для кнопки.

```
withCaption(caption: string) : ButtonBuilder
```

где:
`caption` – подпись кнопки.
`return ButtonBuilder` – возвращает класс строителя кнопки.

### withIcon()

Метод задает иконку для кнопки.

```
withIcon(icon: string): ButtonBuilder
```

где:
`icon` – имя класса `CSS` с иконкой.
`return ButtonBuilder` – возвращает класс строителя кнопки.

### withClickAction()

Метод задает колбэк для обработки события нажатия на кнопку.

```
withClickAction(action: EventListener) : ButtonBuilder;
```

где:
`action` – обработчик события нажатия кнопки.
`return ButtonBuilder` – возвращает класс строителя кнопки.

**Метод еще не доработан и может быть изменен в следующих версиях.**

### withIsChecked()

Метод задает стиль для нажатой кнопки.

```
withIsChecked(value: boolean): ButtonBuilder;
```

где:
`value` – обработчик события нажатия кнопки.
`return ButtonBuilder` – возвращает класс строителя кнопки.

## Control

**Control** – это базовый класс для описания элемента управления.

### *Методы*

### addClass()

Добавляет пользовательский `CSS` стиль к элементу управления.

```
addClass(cssClass: string): void;
```

где:
`cssClass` – имя класса `CSS` стиля.

### removeClass()

Удаляет пользовательский `CSS` стиль из элемента управления.

```
removeClass(cssClass: string): void;
```

где:
`cssClass` – имя класса `CSS` стиля.

# Toolbar

**Toolbar** – это базовый класс управления панелью инструментов компонента. Класс расширяет возможности класса `Control`.

### *Методы*

## addControl()

Метод добавляет кнопку в панель инструментов.

```
addControl(control: Control): void;
```

где:
`control` – экземпляр элемента управления.

## removeControl()

Метод добавляет кнопку в панель инструментов.

```
removeControl(id: string): void;
```

где:
`id` – идентификатор элемента управления.

## addClass()

Метод добавляет пользовательский `CSS` стиль к элементу управления.

```
addClass(cssClass: string): void;
```

где:
`cssClass` – имя класса `CSS` стиля.

## removeClass()

Метод удаляет пользовательский `CSS` стиль из элемента управления.

```
removeClass(cssClass: string): void;
```

где:
`cssClass` – имя класса `CSS` стиля.

## ToolbarBuilder

**ToolbarBuilder** – это класс создателя панели инструментов.

### *Методы*

### addButton()

Метод добавляет кнопку в панель инструментов.

```
addButton(id: string) : ButtonBuilder;
```

где:

`id` - идентификатор нового элемента управления.

`ButtonBuilder` - конструктор для кнопки.

### removeItem()

Удаляет элемент управления из панели инструментов.

```
removeItem(id: string): void;
```

где:
`id` - идентификатор элемента управления.

## ViewerToolbar

**ViewerToolbar** – это класс управления панелью инструментов компонента. Он расширяет возможности класса `Toolbar`.

# Viewer3D

**Viewer3D** – это базовый класс для всех видов компонентов работы с BIM-моделями.

Этот класс содержит всё необходимое для отображения и взаимодействия с моделями, полученными из системы **Pilot-BIM**.

### *Свойства*

## container

HTML элемент, в котором создан компонент просмотра 3D-моделей.

```
container: HTMLElement;
```

## extensionsLoader

Тип работы с расширениями. Подробнее: ExtensionLoader.

```
extensionsLoader: ExtensionLoader;
```

## events

Свойство для управления событиями компонента.

```
get events(): IEventsDispatcher;
```

## model

Свойство для получения консолидированной модели.

```
get model(): Model;
```

## navigation

Свойство для получения объекта управления навигацией по модели.

```
get navigation(): INavigation;
```

Подробнее: INavigation

### *Методы*

## start()

Метод инициализирует внутренние механизмы компонента.

```
start(): Promise<number>;
```

## finish()

Метод деинициализирует внутренние механизмы компонента.

```
finish(): void;
```

## loadModelPart()

Метод для управления загрузкой частей модели.

```
loadModelPart(data: ArrayBuffer | string, options: ModelLoadingOptions,
onSuccessCallback: SuccessCallback, onErrorCallback: ErrorCallback): void;
```

где:
`data` – массив байт модели или ссылка на модель, `options` – опции для загрузки части модели. Подробнее: ModelLoadingOptions, `onSuccessCallback` – метод для обратного вызова в случае успешной загрузки части модели, `onErrorCallback` – метод для обратного вызова в случае неудачи загрузки части модели.

## unloadModelPart()

Метод для выгрузки части модели.

```
unloadModelPart(modelPart: string | ModelPart): void;
```

где:
`modelPart` – идентификатор части модели или экземпляр части модели.

## setVirtualOrigin()

Метод задаёт положение виртуального начала координат. Координаты объектов модели и положение камеры пересчитываются относительно нового начала координат.
При изменении ВНК возникает событие `VIRTUAL_ORIGIN_CHANGED`. Подробнее: Events3D.

```
setVirtualOrigin(point: Point3): Promise<void>;
```

где:
`point` – положение ВНК. Подробнее: Point3.

## getVirtualOrigin()

Метод возвращает положение виртуального начала координат. Подробнее: Point3.

```
getVirtualOrigin(): Point3;
```

## makeScreenshot()

Метод позволяет сделать снимок сцены.

```
makeScreenshot(mimeType?: string, quality?: number): Promise<Blob>;
```

где:\

`mimeType` – необязательный параметр. Задает тип изображения (image/png, image/jpg и т.д.). `quality` – качество снимка.

## getConfiguration()

Метод получает конфигурацию вьювера. Подробнее: Viewer3DConfiguration.

```
getConfiguration(): Viewer3DConfiguration;
```

# API Reference 2D

# Configuration

## Viewer2DConfiguration

**Viewer2DConfiguration** - класс описывающий настройки компонента **PilotWeb2D**. Этот класс наследуется от базового класса настроек ViewerConfiguration

```
class Viewer2DConfiguration extends ViewerConfiguration {
  settings?: ViewerSettings;
}
```

### *Свойства*

### settings

```
settings?: ViewerSettings;
```

Необязательное поле для задания настроек отображения просмотрщика 2D документов. Подробнее: ViewerSettings

## ViewerConfiguration

**ViewerConfiguration** - Базовый класс описывающий настройки компонентов **PilotWeb2D** и **PilotWeb3D**.

```
class ViewerConfiguration {
  appearance: ViewerSettings = {
    [BaseSettingsNames.TOOLBAR] : {
      direction: ToolbarDirection.TOP_FLUENT,
      content: ToolbarContentAlignment.CENTER
    } as ToolbarStyle
  };
}
```

### *Свойства*

### appearance

```
appearance: ViewerSettings;
```

Свойство для изменения внешнего вида просмотрщика.

### direction

```
direction: string;
```

Управляет положение панели инструментов.

```
export enum ToolbarDirection {
  TOP_FIXED = 'ascn-toolbar-fixed-top',
  TOP_FLUENT = 'ascn-toolbar-top',
  BOTTOM_FIXED = 'ascn-toolbar-fixed-bottom',
  BOTTOM_FLUENT = 'ascn-toolbar-bottom',
}
```

### content

```
content: string;
```

Управляет расположением кнопок на панели инструментов.

```
export enum ToolbarContentAlignment {
  CENTER = 'ascn-toolbar-content-center',
  START = 'ascn-toolbar-content-start',
  END = 'ascn-toolbar-content-end'
}
```

## ViewerSettings

**ViewerSettings** - класс описывающий настройки просмотрщика компонента **PilotWeb2D**.

```
type ViewerSettings = Record<string, any>;
```

# ViewerToolbar

**ViewerToolbar** – это класс для взаимодействия с панелью инструментов.

### *Методы*

## addControl()

```
addControl(control: Control): void
```

Добавить контрол на панель инструментов. Подробнее: Control

## addControl()

```
removeControl(id: string): void
```

Удалить контрол из панели инструментов.

# PilotWeb2D

**PilotWeb2D** – это пространство имён верхнего уровня для взаимодействия с компонентом **PilotWeb2D**.

### *Методы*

## Initializer()

Метод для инициализации компонента **PilotWeb2D**. Все методы работы с компонентом следует использовать после вызова этого метода.

```
type InitializeSuccessCallback = () => void;
function Initializer(options, callback: InitializeSuccessCallback): void;
```

где:
`options` – содержит настройки для инициализации компонента. `callback` – метод обратного вызова. Вызывается, когда завершится инициализация компонента.

Пример:

```
let options = {};
var myCallback = function() {
   console.log("initialization complete, creating the viewer...");
};
PilotWeb2D.Initializer(options, myCallback);
```

## CreateViewer()

Создает экземпляр компонента для просмотра документов.

```
function CreateViewer(container: HTMLElement) : GuiViewer2D;
```

где:
`container` – HTML элемент, в котором создается компонент.

Пример:

```
let htmlDiv = document.getElementById('pilotViewer');
let viewer = PilotWeb2D.CreateViewer(htmlDiv);
```

# GuiViewer2D

**GuiViewer2D** – это класс для компонента просмотра 2D документов. Он расширяет возможности базового класса Viewer2D и содержит всё, что нужно для отображения и взаимодействия с 2D документами `xps`.

### *Методы*

## getToolbar()

```
getToolbar(): ViewerToolbar;
```

Метод получает объект для работы с панелью инструментов. Подробнее: ViewerToolbar

# Viewer2D

**Viewer2D** – это базовый класс для работы с документами компонента **PilotWeb2D**.

Этот класс содержит всё необходимое для отображения и взаимодействия с документами, полученными из системы **Pilot**.

### *Свойства*

## container

HTML элемент, в котором создан компонент просмотра 3D моделей.

```
container: HTMLElement;
```

## extensionsLoader

Тип работы с расширениями. Подробнее: ExtensionLoader.

```
extensionsLoader: ExtensionLoader;
```

## events

Свойство для управления событиями компонента.

```
get events(): EventsDispatcher;
```

### *Методы*

## start()

Метод инициализирует внутренние механизмы компонента.

```
 start(): Promise<number>;
```

## finish()

Метод деинициализирует внутренние механизмы компонента.

```
finish(): void;
```

## loadDocument()

Метод загружает документ в компонент.

```
loadDocument(data: ArrayBuffer | string, options: DocumentLoadingOptions,
onSuccessCallback: SuccessCallback, onErrorCallback: ErrorCallback): void
```

где:
`data` - массив байт документа или ссылка на документ.
`options` - опции для загрузки документа (подробнее: DocumentLoadingOptions).
`onSuccessCallback` - метод для обратного вызова в случае успешной загрузки документа.
`onErrorCallback` - метод для обратного вызова в случае неудачи загрузки документа.

## unloadDocument()

Выгружает документ из компонента.

```
unloadDocument(): void
```

## getConfiguration()

Получает текущие настройки просмотрщика. Подробнее: ViewerConfiguration.

```
getConfiguration(): ViewerConfiguration
```

# Events

## *Системные события*

### Общие события

```
  class CoreEventTypes {
    // Событие изменения размера вьювера.
    static VIEWER_RESIZE_EVENT: string;
    // Событие нажатия левой клавиши мыши.
    static VIEWER_MOUSE_DOWN_EVENT: string;
    // Событие перемещения мыши.
    static VIEWER_MOUSE_MOVE_EVENT: string;
    // Событие отпускания левой клавиши мыши.
    static VIEWER_MOUSE_UP_EVENT: string;
    // Событие изменения настройки.
    static SETTING_CHANGED_EVENT: string;
    // Событие восстановления настройки в значение по умолчанию.
    static SETTING_RESET_EVENT: string;
    // Имя события загрузки расширения. Вызывается после загрузки расширения.
    static EXTENSION_LOADED;
    // Имя события выгрузки расширения. Вызывается непосредственно перед
выгрузкой расширения.
    static EXTENSION_UNLOADING;
    // Имя события выгрузки расширения. Вызывается после выгрузки расширения.
    static EXTENSION_UNLOADED;
  }
```

### Классы событий для 2D

```
// Класс события загрузки или выгрузки расширения.
class ExtensionEvent extends Event {
  extensionName: string;
}
```

# Extensions

## Extension

**Extension** – это базовый класс описания расширения.

### *Свойства*

#### _viewer

```
protected _viewer: Viewer2D;
```

### *Методы*

#### load()

Метод вызвается, когда расширение было загружено.

```
load() : boolean | Promise<boolean>;
```

#### unload()

Метод вызвается, когда расширение было выгружено.

```
unload(): boolean;
```

#### activate()

Активировать работу модуля расширения.

```
activate() : boolean;
```

Возвращает `true`, если активация прошла успешно.

#### deactivate()

Деактивировать работу плагина

```
deactivate(): boolean;
```

Возвращает `true`, если деактивация прошла успешно.

#### getName()

Метод вызвается, когда расширение пытается получить имя расширения.

```
getName(): string;
```

## onToolbarCreated()

Метод вызывается, когда панель инструментов построилась, и расширение имеет возможность добавить/изменить/удалить элементы управления.

```
onToolbarCreated(builder: ToolbarBuilder): void;
```

где:
`builder` – построитель панели инструментов.

## onMouseDown()

Метод вызывается, когда произошло событие нажатия левой клавиши мыши.

```
onMouseDown(event: MouseEvent): void;
```

где:
`event` – событие мыши.

## onMouseMove()

Метод вызывается, когда произошло событие перемещения мыши.

```
onMouseMove(event: MouseEvent): void;
```

где:
`event` – событие мыши.

## onMouseUp()

Метод вызывается, когда произошло событие отпускания левой клавиши мыши.

```
onMouseUp(event: MouseEvent): void;
```

где:
`event` – событие мыши.

## ExtensionLoader

`ExtensionLoader` – это класс, предназначенный для загрузки и инициализации расширений, которые были зарегистрированы в компонентах. Перед тем, как загрузить расширение, его необходимо зарегистрировать с помощью `ExtensionManager`.

Пример:

```
// описываем расширение
class MyExtension extends PilotWeb2D.Extension {
  ...
}
// регистрируем
PilotWeb2D.theExtensionManager.registerExtensionType('myExtension',
MyExtension);

// загружаем в компонент
let viewer = PilotWeb2D.CreateViewer(div);
viewer.extensionLoader.loadExtension('myExtension');
```

### *Методы*

### loadExtension()

Метод загружает зарегистрированное расширение в компонент.

```
loadExtension(extensionId: string): Promise<Extension>;
```

где:
`extensionId` – уникальное имя расширения.

### unloadExtension()

Метод выгружает расширение.

```
unloadExtension(extensionId: string) : Promise<boolean>;
```

где:
`extensionId` – уникальное имя расширения.

### getExtensions()

Метод получает все загруженные расширения.

```
getExtensions(): Extension[] ;
```

## ExtensionManager

**ExtensionManager** – это класс-менеджер расширений, позволяет зарегистрировать или разрегистрировать расширения в компоненте **PilotWeb2D**.

`ExtensionManager` доступен из пространства имен **PilotWeb2D** через свойство `theExtensionManager`.

Пример:

```
// описываем расширение
class MyExtension extends PilotWeb2D.Extension {
  ...
}
// регистрируем
PilotWeb2D.theExtensionManager.registerExtensionType('myExtension',
MyExtension);
```

### *Методы*

### registerExtensionType()

Метод регистрирует новое расширение в системе. После этого это расширение можно загрузить.

```
registerExtensionType(extensionId: string, extension: typeof Extension) :
boolean;
```

где:
`extensionId` – уникальное имя расширения.
`extension` – тип расширения унаследованный от `PilotWeb2D.Extension`

### unregisterExtensionType()

Метод разрегистрирует расширение.

```
unregisterExtensionType(extensionId: string) : boolean;
```

где:
`extensionId`– уникальное имя расширения.

### getExtensionType()

Метод получает тип зарегистрированного расширения.

```
getExtensionType(extensionId: string): typeof Extension
```

где:
`extensionId`– уникальное имя расширения.

## IAnnotationLayer

**IAnnotationLayer** – интерфейс управления замечаниями на странице документа.

```
export interface IAnnotationLayer {
  get pageDiv(): HTMLDivElement;
  get div(): HTMLDivElement | null;

  addOverlay(overlay: HTMLElement): void;
  removeOverlay(overlay: HTMLElement): void;
  getOverlays(): HTMLElement[];
}
```

### *Свойства*

#### pageDiv

Содержит корневой HTML-элемент страницы.

```
get pageDiv(): HTMLDivElement;
```

#### div

Содержит корневой HTML-элемент слоя замечаний.

```
get div(): HTMLDivElement | null;
```

### *Методы*

#### addOverlay

Добавляет HTML-элемент на слой замечаний.

```
addOverlay(overlay: HTMLElement): void;
```

где:
`overlay` – HTML-элемент.

#### removeOverlay

Удаляет HTML-элемент из слоя замечаний.

```
removeOverlay(overlay: HTMLElement): void;
```

где:
`overlay` – HTML-элемент.

## getOverlays

Получает все добавленные на слой замечаний HTML-элементы.

```
getOverlays(): HTMLElement[];
```

# IDocument

**IDocument** – это интерфейс для доступа к элементам управления документа.

```
export interface IDocument {
  increaseScale(scaleFactor?: number): void;
  decreaseScale(scaleFactor?: number): void;
  fit(): void;

  getPageAsync(pageNumber: number): Promise<IDocumentPage>;
  getPageByTarget(element: HTMLElement): IDocumentPage | undefined;

  scrollPageIntoViewAsync(pageNumber: number): Promise<void>;
}
```

### *Методы*

## increaseScale

Увиличивает масштаб отображения документа.

```
increaseScale(scaleFactor?: number): void;
```

где:
`scaleFactor` – коэффициент масштаба. Коэффициент 1 = 100%.

## decreaseScale

Уменьшает масштаб отображения документа.

```
decreaseScale(scaleFactor?: number): void;
```

где:
`scaleFactor` – коэффициент масштаба. Коэффициент 1 = 100%.

## fit

Выравнивает документ по ширине просмотрщика.

```
fit(): void;
```

## getPageAsync

Получает страницу по номеру. Подробнее: IDocumentPage

```
getPageAsync(pageNumber: number): Promise<IDocumentPage>;
```

где:
`pageNumber` – номер запрашиваемой страницы.

## getPageByTarget

Получает страницу по заданному HTML-элементу. Подробнее: IDocumentPage

```
getPageByTarget(element: HTMLElement): IDocumentPage | undefined;
```

где:
`element` – HTML-элемент, который находится внутри страницы.

## scrollPageIntoViewAsync

Проскролировать до указанной страницы.

```
scrollPageIntoViewAsync(pageNumber: number): Promise<void>;
```

где:
`pageNumber` – номер страницы.

# IDocumentPage

**IDocumentPage** - интерфейс управления страницей документа.

```
export interface IDocumentPage {
  get div(): HTMLDivElement;
  get canvas(): HTMLCanvasElement | SVGSVGElement;
  get pageNumber(): number;
  get viewport(): DOMRect;
  get annotationLayer(): IAnnotationLayer | null;
  get height() : number;
  get width() : number;
  get scale(): number;

  update(params?: IPageUpdateParams): void;
  dispose();
}
```

### *Свойства*

## div

Содержит корневой HTML-элемент страницы.

```
get div(): HTMLDivElement;
```

## canvas

Содержит HTML-элемент с содержимым страницы.

```
get canvas(): HTMLCanvasElement | SVGSVGElement;
```

## pageNumber

Содержит номер страницы.

```
get pageNumber(): number;
```

## viewport

Содержит габаритные размеры страницы.

```
get viewport(): DOMRect;
```

## annotationLayer

Содержит объект для взаимодействия со слоем замечаний. Подробнее: IAnnotationLayer.

```
get annotationLayer(): IAnnotationLayer | null;
```

### height

Содержит высоту страницы в пикселях.

```
get height() : number;
```

### width

Содержит ширину страницы в пикселях.

```
get width() : number;
```

### scale

Содержит текущий масштаб страницы.

```
get scale() : number;
```

#### *Методы*

### update

Обновляет страницу с заданными параметрами. Подробнее: IPageUpdateParams.

```
update(params?: IPageUpdateParams): void;
```

### dispose

Очищает ресурсы страницы.

```
dispose(): void;
```

## IPageUpdateParams

**IPageUpdateParams** – параметры обновления страницы документа.

```
export interface IPageUpdateParams {
  scale?: number; // масштаб
}
```

## DocumentLoadingOptions

**DocumentLoadingOptions** – опции загрузки документа в компонент **PilotWeb2D**.

```
export class DocumentLoadingOptions {
  documentId?: string; // уникальный идентификатор документа
}
```

## Point2

**Point2** – точка в 2-х координатах.

```
class Point2 {
  x: number;
  y: number;

  constructor(x: number, y: number);
}
```

# Common API Reference

## Button

**Button** – это класс для элемента управления **Кнопка**. Класс расширяет возможности базового класса `Control`. Подробнее: Control.

### *Свойства*

#### caption

Задаёт или получает имя кнопки.

```
caption: string;
```

#### clickAction

Слушает нажатия на кнопку.

```
clickAction: EventListener;
```

**Метод еще не доработан и может быть изменен в следующих версиях.**

### *Методы*

#### addClass()

Метод добавляет пользовательский `CSS` стиль к элементу управления.

```
addClass(cssClass: string): void;
```

где:
`cssClass` – имя класса `CSS` стиля.

#### removeClass()

Метод удаляет пользовательский `CSS` стиль из элемента управления.

```
removeClass(cssClass: string): void;
```

где:
`cssClass` - имя класса `CSS` стиля.

#### setIsChecked()

Метод позволяет установить стиль выбора для кнопки.

```
setIsChecked(value: boolean): void;
```

где:
`value` - селектирована кнопка или нет.

### setState()

Метод позволяет установить состояние кнопки.

```
setState(state: Button.State): boolean;
```

где:
`state` - состояние кнопки. Подробнее: Button.State.

### setText()

Метод позволяет установить текст кнопки.

```
setText(text: string): void;
```

где:
`text` - текст, который будет отображать кнопка.

### getState()

Метод возвращает состояние кнопки.

```
getState(): Button.State;
```

### setIcon()

Метод позволяет установить иконку для кнопки.

```
setIcon(iconClassName: string): void;
```

где:
`iconClassName` - имя класса `CSS` стиля, где содержится иконка.

### setFromSvgTemlate()

Метод позволяет установить иконку кнопки в виде svg.

```
setFromSvgTemlate(template: string): void
```

где:
`template` - svg элемент иконки в виде строки.

## Control

**Control** – базовый класс для создания ui элементов.

### *Свойства*

#### container

```
container: HTMLElement;
```

Получает HTML елемент контрола.

### *Методы*

#### addClass()

Добавляет класс к элементу.

```
addClass(cssClass: string): void
```

где:
`cssClass` – имя класса.

#### removeClass()

Удаляет класс у элемента.

```
removeClass(cssClass: string): void
```

где:
`cssClass` – имя класса.

#### getId()

Получить идентификатор контрола.

```
getId(): string
```

#### setToolTip()

Задать подсказку для контрола

```
setToolTip(tooltipText: string): void
```

где:
`tooltipText` – текст для всплывающей подсказки.

## setText()

Задает текст для элемента управления. Используется в наследниках.

```
setText(text: string): void {}
```

где:
`text` – текст.

## setState()

Устанавливает состояние элемента управления. Используется в наследниках.

```
setState(state: ControlState.State): void {}
```

где:
`state` – состояние элемента управления. Подробнее: ControlState.

## ControlState

**ControlState.State** - возможные состояния элементов управления.

```
export namespace ControlState {
  export enum State {
    ACTIVE = 0, // Активна.
    INACTIVE = 1, // Не активна.
    DISABLED = 2 // Отключена.
  }
}
```

## Dialog

**Dialog** – класс для создания диалоговых окон

## ElementClass

**ElementClass** – интерфейс, добавляющий названия классов к заголовку, контенту и панели к которой прикреплен диалог.

```
export interface ElementClass {
  //Свойство, которое устанавливает класс панели к которой прикреплен диалог.
  panelClassName?: string;
  //Свойство, которое устанавливает класс контента диалога.
  contentClassName?: string;
  //Свойство, которое устанавливает класс заголовока диалога.
  headerClassName?: string;
}
```

### *Конструктор*

```
  constructor(id: string, panelToAttach: HTMLElement);
```

где:
`id`– идентификатор диалога.
`panelToAttach` – элемент на странице к которому будет прикреплено диалоговое окно.

### *Свойства*

#### dialog

Получает созданный элемент диалога.

```
get dialog(): HTMLElement;
```

#### dialogContent

Получает созданный элемент контента внутри диалога.

```
get dialogContent(): HTMLElement;
```

#### resizable

Получает флаг, который показывает возможность изменения размеров окна.

```
get resizable(): boolean;
```

*Методы*

## setDialogContent()

Устанавливает содержимое диалога и возвращает класс Dialog.

```
setDialogContent(value: HTMLElement): Dialog
```

где:
`value` - HTML представление элемента.

## setHeader()

Устанавливает заголовок диалога и возвращает класс Dialog.

```
setHeader(value: HTMLElement): Dialog
```

где:
`value` - HTML представление элемента.

## setCaption()

Устанавливает текст заголовка диалога и возвращает класс Dialog.

```
setCaption(value: string) : Dialog
```

где:
`value` - текст заголовка.

## setSvgIcon()

Метод позволяет установить иконку заголовка диалога в виде svg и возвращает класс Dialog.

```
setSvgIcon(template: string): Dialog
```

где:
`template` - svg элемент иконки в виде строки.

## setFooter()

Устанавливает нижний колонтитул диалога и возвращает класс Dialog.

```
setFooter(value: HTMLElement): Dialog
```

где:
`value` - HTML представление элемента.

## setDialogElementClassNames()

Устанавливает css классы содержимого в диалоговом окне и возвращает класс Dialog.

```
setDialogElementClassNames(value: ElementClass): Dialog
```

где:
`value` - объект с названиями классов для элементов внутри диалога.

## setResizable()

Устанавливает возможность изменения ширины и высоты диалога, возвращает класс Dialog.

```
setResizable(value: boolean): Dialog
```

где:
`value` - флаг для назначения включения и выключения возможности изменения размеров окна.

## setWindowOptions()

Устанавливает настройки для измнения положения и ширины/высоты диалога, возвращает класс Dialog. Без этой опции диалог не будет сохранять свои изменённые свойства. Подробнее: IWindowStateOptions.

```
setWindowOptions(value: IWindowStateOptions): Dialog
```

где:
`value` - настройки для работы измнения положения и ширины/высоты диалога.

## setDraggable()

Устанавливает возможность изменения положения диалога, возвращает класс Dialog.

```
setDraggable(value: boolean): Dialog
```

где:
`value` - флаг для назначения включения/выключения возможности изменения положения.

## setCloseble(value: boolean): Dialog

Устанавливает возможность закрытия диалога, возвращает класс Dialog.

```
setCloseble(value: boolean): Dialog
```

где:
`value` - флаг для назначения включения/выключения возможности закрытия диалога.

## openDialog()

Создаёт компонет диалога на panelToAttach и возвращает его как HTMLElement.

```
openDialog(): HTMLElement
```

## destroyDialog()

Уничтожает компонет диалога.

```
destroyDialog(): void
```

## isDialogShown()

Возвращает состояние диалога, закрыт/открыт.

```
isDialogShown(): boolean
```

## subscribe()

Подписка на состояние диалога, закрыт/открыт.

```
subscribe(fn: (state: boolean) => void): void
```

где:
`fn` - функция callback для обработки события открытия/закрытия окна. `state` - состояние открыт или закрыт диалог.

## IconsSet

**IconsSet** – класс-хранилище иконок.

### Список доступных иконок:

```
 export namespace Viewer3DIcons {
  export const VIEWER_SETTINGS_ICON: string;
  export const VIEWER_MODEL_BROWSER_ICON: string;
  export const VIEWER_FULL_SCREEN_ICON: string;
  export const VIEWER_COLLAPSE_ICON: string;
  export const VIEWER_ELEMENT_PROPERTIES_ICON: string;
  export const VIEWER_ADD_CLIPPING_PLANE_ICON: string;
  export const VIEWER_CLIPPING_FLIP_ICON: string;
  export const VIEWER_DELETE_CLIPPING_PLANE_ICON: string;
  export const VIEWER_CLIPPING_CUBE_ICON: string;
  export const VIEWER_DISABLED_DELETE_CLIPPING_PLANE_ICON: string;
  export const VIEWER_DISABLED_CLIPPING_FLIP_ICON: string;
  export const VIEWER_ADD_REMARK_ICON: string;
}

export namespace ViewerGeneralIcons {
  export const ZOOM_IN: string;
  export const ZOOM_OUT: string;
  export const CLOSE: string;
  export const STRETCH_WINDOW: string;
  export const EXPAND_TREE: string;
  export const COLLAPSE_TREE: string;
  export const ARROW_DROP_DOWN: string;
  export const ARROW_DROP_RIGHT: string;
  export const CIRCLE_ICON: string;
}
```

### Пример использования:

Использовать как html элемент, а именно через свойство innerHTML.

```
  element.innerHTML = PilotWeb3D.Viewer3DIcons.VIEWER_SETTINGS_ICON
```

### Пользовательское объявление иконки:

Чтобы добавить иконку:

1. Сохранить svg представление иконки в виде строки.
2. Вставить иконку с помощью innerHTML.

```
export class IconExample {
  private readonly EXAMPLE_ICON  = `
        <?xml version="1.0" encoding="UTF-8"?>
        <svg version="1.0" viewBox="0 0 24 24"
xmlns="http://www.w3.org/2000/svg">...</svg>`

  hereAddIcon(): void {
    ...
    const element = document.getElementById('myDiv');
```

```
    element.innerHTML = this.EXAMPLE_ICON;
    ...
  }
}
```

## ISettings

**ISettings** - интерфейс управления настройками через вьювер.

```
export interface ISettings {
  changeSetting<T>(name: string, value: T, notify?: boolean, providedData?:
any): void;
  getSettingValue<T>(name: string): T;
}
```

### *Методы*

### changeSetting()

Метод изменяющий настройку вьювера

```
changeSetting<T>(name: string, value: T, notify?: boolean, providedData?:
any): void;
```

где:

`name` – имя настройки.
`value` – значение настройки.
`notify` – флаг для работы _eventDispatcher, если ничего не передано или true, то выбросит событие для подписчиков.
`providedData` – дополнительные данные, если необходимо получить подписчикам _eventDispatcher.

### getSettingValue()

Метод возвращает настройку вьювера

```
getSettingValue<T>(name: string): T;
```

где:

`name` – имя настройки.

## Localization

**Localization** – это класс для управления локализацией компонентов. Компоненты **PilotWeb3D** и **PilotWeb2D** используют библиотеку i18next для поддержки локализации. На данный момент поддерживаются 2 языка: `ru` и `en`.

### *Методы*

### initialize()

Инициализировать локализацию компонента.

```
static initialize(options: any): Promise<void>
```

где:
`options`: - опции для указания, какой язык использовать.

Пример:

```
const options = {
  language: "ru"
}
```

### translate()

Получить переведенный текст на заданном языке.

```
static translate(stringToTrans: string): string;
```

где:
`stringToTrans`: - ключ, указанный в `translation.json`. Подробнее: i18next documentation.

### setLanguage()

Изменить текущий язык для компонентов.

```
static setLanguage(language: string): Promise<void>;
```

где:
`language`: - новый язык. Например: `ru` или `en`.

### extendLocalization()

Добавить дополнительную локализацию.

```
static extendLocalization(locales: any): boolean;
```

где:
`locales`: - дополнительная локализация для модуля расширения.

Пример использования:

Файл `/assets/locales/en/translation.json`

```
{
  "Extension": {
    "title": "Properties"
  }
}
```

Файл `ExtensionLocales.js`

```
import json_en from '/assets/locales/en/translation.json';
import json_ru from '/assets/locales/ru/translation.json';

export const locales = {
  en: json_en,
  ru: json_ru
};
```

Файл `Extension.js`

```
import { locales } from './ExtensionLocales';
export class Extension extends PilotWeb3D.Extension {

  constructor(viewer: PilotWeb3D.Viewer3D, options?: object) {
    super(viewer, options);

    // Добавляем локализацию для нашего расширения
    PilotWeb3D.Localization.extendLocalization(locales);

    // ...

    // Получить локализованную строку
    const title = PilotWeb3D.Localization.translate('Extension.title');
  }
}
```

# Toolbar

**Toolbar** – Класс, описывающий панель инструментов.

### *Методы*

## changeToolbarPosition()

Меняет позицию тулбара с текущей на предоставленную.

```
changeToolbarPosition(direction: string): void;
```

## changeToolbarContentAlignment()

Меняет порядок отображения контента в тулбаре.

```
changeToolbarContentAlignment(content: ToolbarContentAlignment): void;
```

где:
`content` – порядок отображения контента внутри тулбара. Подробнее: ToolbarContentAlignment.

## addControl()

Добавляет элемент в тулбар.

```
addControl(control: Control, index: number = 0): void;
```

где:
`control` – объект типа Control. Подробнее: Control.
`index` – индекс.

## addDropdown()

Добавляет выпадающий список в тулбар.

```
addDropdown(id: string, tooltip: string, positionIndex: number = 0,
selectedIndex: number, dropdownEmitter: Control, controls: Control [],
    itemClicked: ({ index, event}: {
    index: number,
    event: Event
  }) => void): void;
```

где:
`id` – идентификатор элемента.
`tooltip` – всплывающая подсказка на элементе.
`positionIndex` – позиция элемента в тулбаре.
`selectedIndex` – выбранный в выпадающем меню пункт, если ничего не выбрано

передавать `-1`.
`dropdownEmitter` – элемент-триггер, по клику на который откроется выпадающее меню.
`controls` – пункты меню.
`itemClicked` – колбек функция-обработчик клика по пункту меню.
`index` - номер пункта меню по которому был произведён клик.
`event` - событие клика.

## removeControl()

Удаляет элемент из тулбара.

```
removeControl(id: string): void
```

где:
`id` – идентификатор элемента.

## ViewerConfiguration

**ViewerConfiguration** - Базовый класс, описывающий настройки компонентов **PilotWeb2D** и **PilotWeb3D**.

```
class ViewerConfiguration {
  appearance: ViewerSettings = {
    [BaseSettingsNames.TOOLBAR] : {
      direction: ToolbarDirection.TOP_FLUENT,
      content: ToolbarContentAlignment.CENTER
    } as ToolbarStyle,
    [BaseSettingsNames.THEME] : SettingsTheme.LIGHT_THEME;
  };
}
```

### *Свойства*

#### appearance

```
appearance: ViewerSettings;
```

Параметры внешнего вида вьювера. Подробнее: ViewerSettings.

#### direction

```
direction: string;
```

Управляет положением панели инструментов. Подробнее: ToolbarDirection.

#### content

```
content: string;
```

Управляет расположением кнопок на панели инструментов. Подробнее: ToolbarContentAlignment.

#### theme

```
theme: string;
```

Управляет темой вьювера и его компонентов. Подробнее: SettingsTheme.

#### settingsPrefix

```
settingsPrefix: string;
```

Задает префикс для настроек, хранящихся в браузере клиента.

*Методы*

### mergeConfigurationAndSettings()

Метод объединяет переданную конфигурацию и настройки, заменяя значения из конфигурации значениями из настроек.

```
public mergeConfigurationAndSettings(configuration: ViewerSettings, settings:
ISettings): void;
```

где:

`configuration` – конфигурация. Подробнее: ViewerSettings.
`settings` – значение настройки. Подробнее: ISettings.

### createConfiguration()

Метод объединяет объединяет свойства из переданных параметров в объект origin. Origin подменяется свойствами из configuration, если configuration не был передан или был передан пустой объект, то configuration присваивается origin по ссылке.

```
public createConfiguration(configuration: ViewerSettings, origin:
ViewerSettings): void;
```

где:

`configuration` – конфигурация. Подробнее: ViewerSettings.
`origin` – значение настройки. Подробнее: ViewerSettings.

### changeTheme()

Метод меняет тему вьювера.

```
public changeTheme(newTheme: string): void;
```

где:

`newTheme` – значение новой темы из SettingsTheme. Подробнее: SettingsTheme.

## ToolbarDirection

**ToolbarDirection** - возможные позиции панели инструментов.

```
export enum ToolbarDirection {
  TOP_FIXED = 'ascn-toolbar-fixed-top', // Фиксированный сверху
  TOP_FLUENT = 'ascn-toolbar-top', // Не фиксированный сверху
  BOTTOM_FIXED = 'ascn-toolbar-fixed-bottom', // Фиксированный снизу
  BOTTOM_FLUENT = 'ascn-toolbar-bottom', // Не фиксированный снизу
}
```

## ToolbarContentAlignment

**ToolbarContentAlignment** - возможные позиции содержимого в панели инструментов.

```
export enum ToolbarContentAlignment {
  CENTER = 'ascn-toolbar-content-center', // По центру
  START = 'ascn-toolbar-content-start', // Прижато к левому краю
  END = 'ascn-toolbar-content-end' // Прижато к правому краю
}
```

## SettingsTheme

**SettingsTheme** - возможные варианты темы вьювера.

```
export enum SettingsTheme {
  LIGHT_THEME = 'ascn-light', // Тёмная тема
  DARK_THEME = 'ascn-dark', // Светлая тема
}
```

## ViewerSettings

**ViewerSettings** - настройки, ключами свойств которого являются строки, а значениями свойств является любой тип.

```
type ViewerSettings = Record<string, any>;
```

## WindowStater

**WindowStater** – класс для хранения состояния диалогового окна.

## IWindowStyle

**IWindowStyle** – интерфейс, хранящий свойства стилей диалогового окна.

```
export interface IWindowStyle {
  // Свойство, хранящее ширину окна
  width: string,
  // Свойство, хранящее высоту окна
  height: string,
  // Свойство, хранящее координату по горизонтали относительно левого края
окна
  left: string,
  // Свойство, хранящее координату по горизонтали относительно правого края
окна
  right: string,
  // Свойство, хранящее координату по вертикали
  top: string
}
```

## IWindowStateOptions

**IWindowStateOptions** – интерфейс, хранящий свойства для работы ресайза и перетаскивания диалогового окна.

```
export interface IWindowStateOptions {
  // Ключ по которому будут сохранены настройки для восстановления при
последующем открытии окна
  saveKey: string;
  // Свойство, которое обозначает надо ли устанавливать окну сохранённые
настройки размера при последующем открытии
  restoreWindowSize: boolean;
  // Свойство, которое обозначает надо ли устанавливать окну сохранённые
настройки позиции при последующем открытии
  restoreWindowPosition: boolean;
}
```

### *Конструктор*

```
  constructor(savedStateOptions: IWindowStateOptions, containerToRestore:
HTMLElement);
```

где:

`savedStateOptions`– объект для хранения свойств работы с окном.

`containerToRestore` – HTML элемент с которым происходит сохранение/восстановление настроек (css стилей). Подробнее: IWindowStyle.

### *Свойства*

## windowStylesState

Получает сохранённые стили окна, если был установлен saveKey.

```
get windowStylesState(): IWindowStyle;
```

## windowOptionsState

Получает настройки, установленные окну.

```
get windowOptionsState(): IWindowStateOptions;
```

### *Методы*

## restore()

Восстанавливает положение окна и его ширину/высоту исходя из IWindowStyle.

```
restore(): void
```

## saveWindowState()

Сохраняет стили окна в localStorage, если был передан saveKey из IWindowStateOptions.

```
saveWindowState(): void
```

## ExtensionLoader

`ExtensionLoader` – класс, отвечающий за загрузку и инициализацию зарегистрированных (в компоненте) расширений.

### *Методы*

### loadExtension()

Метод загружает и инициализирует зарегистрированное расширение.

```
loadExtension(extensionId: string): Promise<Extension>;
```

где:
`extensionId` – уникальное имя расширения.

### unloadExtension()

Метод выгружает расширение.

```
unloadExtension(extensionId: string) : Promise<boolean>;
```

где:
`extensionId` – уникальное имя расширения.

### getExtensions()

Метод получает все загруженные расширения.

```
getExtensions(): Extension[] ;
```

## IEventsDispatcher

**IEventsDispatcher** - интерфейс управления подписками на события.

```
export interface IEventsDispatcher {
  addEventListener(event: string, listener: EventListener, options?: object): void;
  removeEventListener(event: string, listener: EventListener): void;
  hasEventListener(event: string, listener: EventListener): boolean;
  dispatchEvent(event: string | Event): void;
  dispatchEventAsync(event: string | Event): void;
  clearListeners(): void;
}
```

### *Методы*

### addEventListener()

Подписаться на событие

```
addEventListener(event: string, listener: EventListener, options?: object): void;
```

где:
`event` – имя события.
`listener` – обработчик события.
`options` – дополнительные параметры подписки. Необязательный параметр.

### removeEventListener()

Отписаться от события

```
removeEventListener(event: string, listener: EventListener): void;
```

где:
`event` – имя события.
`listener` – обработчик события.

### hasEventListener()

Проверить подписан ли обработчик на событие.

```
hasEventListener(event: string, listener: EventListener): boolean;
```

где:
`event` – имя события.
`listener` – обработчик события.

Возвращает результат проверики.

## dispatchEvent()

Возбудить событие.

```
dispatchEvent(event: string | Event): void;
```

где:
`event` – имя события или объект `Event`.

## dispatchEventAsync()

Возбудить событие. Асинхронный метод.

```
dispatchEventAsync(event: string | Event): void;
```

где:
`event` – имя события или объект `Event`.

## clearListeners()

Удалить всех подписчиков.

```
clearListeners(): void;
```

## ComboButton

**ComboButton** – класс, управляющий кнопкой с выпадающим списком.

### *Свойства*

### subMenu

Возвращает объект SubMenu. Подробнее: SubMenu.

```
get subMenu(): SubMenu;
```

### *Методы*

### onClick()

Функция для обработчика клика на кнопку раскрывающегося меню.

```
onClick = function(event: PointerEvent): void;
```

где:
`event` – событие клика на кнопку.

### setText()

Метод позволяет установить текст кнопки.

```
setText(text: string): void;
```

где:
`text` - текст, который будет отображать кнопка.

### setFromSvgTemlate()

Метод позволяет установить иконку кнопки в виде svg

```
setFromSvgTemlate(template: string): void
```

где:
`template` - svg элемент иконки в виде строки.

## SubMenu

**SubMenu** – класс, создающий и управляющий выпадающим меню.

### *Свойства*

```
readonly controls: IControl[];
```

Массив элементов меню. Подробнее: IControl.

### selectedIndex : number

Индекс выбранного элемента управления.

```
get selectedIndex(): number;
set selectedIndex(value: number);
```

По умолчанию: `-1`.

### *Методы*

### addControl()

Функция добавляет элемент управления.

```
addControl(control: IControl, index?: number): void;
```

где:
`control` – добавляемый элемент управления.
`index` – позиция в списке куда добавляется элемент, по умолчанию в конец.

### removeControl()

Функция удаляет элемент управления из списка.

```
removeControl(index?: number): void;
```

где:
`index` – позиция в списке откуда удаляется элемент, по умолчанию с конца.

### clearList()

Метод очищает список и массив controls.

```
clearList(): void;
```

## changeControl()

Метод меняет указанный в списке элемент управления.

```
changeControl(control: IControl, index: number): void;
```

где:
`index` – позиция элемента, который нужно заменить.
`control` – новый элемент управления.

## getControlsCount()

Метод возвращает количество элементов управления в списке.

```
getControlsCount(): number;
```

# IControl

**IControl** - интерфейс, описывающий базовый элемент управления.

```
export interface IControl {
  container: HTMLElement;
  getId(): string;
  setToolTip(tooltipText: string): void;
  setText(text: string): void;
}
```

### *Свойства*

## container

```
container: HTMLElement;
```

DOM представление компонента.

### *Методы*

## getId()

Возвращает идентификатор элемента.

```
getId(): void;
```

## setToolTip()

Устанавливает подсказку при наведении.

```
setToolTip(tooltipText: string): void;
```

где:
`tooltipText` - текст, который будет отображаться при наведении.

## setText()

Устанавливает текст элемента.

```
setText(text: string): void;
```

где:
`text` - текст, который будет отображаться в элементе.

## Select

**Select** – класс для создания списка выбора с полем.

## ISelectItem

**ISelectItem** – интерфейс, хранящий описание элемента списка.

```
export interface ISelectItem {
  // Свойство, хранящее текст элемента списка, который будет отображаться
  text: string;
  // Свойство, хранящее значение элемента списка, которое необходимо для
работы списка
  value: string;
}
```

### *Свойства*

### select : HTMLElement

HTML представление списка выбора.

```
get select(): HTMLElement;
```

### disabled : boolean

Состояние для отключения списка.

```
get disabled(): boolean;
set disabled(value: boolean);
```

### selectedIndex : number

Индекс выбранного элемента в списке.

```
get selectedIndex(): number;
set selectedIndex(value: number);
```

### placeholder : string

Текст-подсказка поля выбора.

```
get placeholder(): string;
set placeholder(value: string);
```

### label : HTMLElement

HTML представление текста-подсказки.

```
get label(): HTMLElement;
```

### previousSelectedIndex : number

Индекс предыдущего выбранного элемента в списке.

```
get previousSelectedIndex(): HTMLElement;
```

#### *Методы*

### onChange()

Подписка на событие изменения выбранного в списке значения.

```
onChange({ index: number, value: string }): void
```

где:
`index` - индекс выбранного значения.
`value` - значение элемента списка.

### update()

Метод обновляет элементы списка.

```
update(array: ISelectItem [], selectedIndex: number): void
```

где:
`array` - массив новых элементов. Подробнее: ISelectItem.
`selectedIndex` - элемент, выбранный в новом списке.

## Checkbox

**Checkbox** – класс для создания элемента управления состоянием, флажок, который можно снимать или устанавливать.

### *Свойства*

### checked : boolean

Состояние для снятия/установки галочки.

```
get checked(): boolean;
set checked(value: boolean);
```

### disabled : boolean

Состояние для отключения элемента управления.

```
get disabled(): boolean;
set disabled(value: boolean);
```

### label : HTMLElement

HTML представление текста-подсказки.

```
get label(): HTMLElement;
```

### *Методы*

### onChange()

Подписка на событие изменения значения.

```
onChange(value: boolean): void {}
```

где:
`value` - значение элемента, установлена галочка или нет.

### createElement()

Создает и возвращает HTML представление элемента элемента управления.

```
createElement(): Element
```

## ExtensionActivationController

**ExtensionActivationController** – класс, управляющий активностью расширения.

### *Методы*

### getActive()

Метод возвращает имя активного расширения.

```
getActive(): string;
```

### activate()

Метод активирует новое расширение. Активное расширение принимает значение `name` и инициирует асинхронное событие, в котором поле `extensionName` равно переданному `name`.

```
activate(name: string): void;
```

где:
`name` – имя нового расширения.

### deactivate()

Метод деактивирует расширение. Активное расширение принимает значение `null` и инициирует асинхронное событие, в котором поле `extensionName` равно `null`.

```
deactivate(): void;
```

# Базовые расширения 3D

## Базовые расширения 3D

Компоненты работы с BIM-моделями и документами поддерживают расширение функциональности с помощью подключаемых модулей расширений. Система **PilotComponentKit** включает в себя набор модулей расширений, которые при необходимости могут быть подключены. Каждый модуль расширения в этом списке вносит новые возможности в компоненты системы. Если вам нужно добавить специализированные функции Вы можете создавать свои собственные расширения.

## BoxSelection

**BoxSelection** – расширение, которое позволяет селектировать элементы с помощью рамки. Учитывает секущие плоскости на сцене. Объекты за пределами секущего объема не выделяются.

Для того чтобы выделить объекты рамкой: зажмите и удерживайте клавишу `Shift`, затем зажмите и удерживайте левую кнопку мыши - появится рамка выделения. После появления рамки можно больше не зажимать клавишу `Shift`.

Выделение рамкой слева-направо выделяет только полностью содержащиеся внутри рамки объекты. Объекты, обрезанные секущими плоскостями не выделяются. Рамка рисуется голубым цветом.

Выделение рамкой справа-налево выделяет все объекты касающиеся рамки и содержащиеся внутри. Выделяются даже те объекты, которые были обрезаны секущими плоскостями. Рамка рисуется зелёным цветом.

Расширение имеет имя `PilotWeb3D.BoxSelection`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/BoxSelection3D
/BoxSelection.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.BoxSelection");
```

### *Методы*

### activate()

Активировать расширение. Расширение подписывается на события клавиатуры и мыши.

```
activate(): void;
```

## deactivate()

Деактивировать расширение. Расширение отписывается от событий клавиатуры и мыши.

```
deactivate(): void;
```

## ClippingPlaneExtension

**ClippingPlaneExtension** – расширение, которое позволяет задать секущие плоскости на сцене.

Расширение имеет имя **`PilotWeb3D.ClippingPlane`**.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/ClippingPlane3
D/ClippingPlane.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.ClippingPlane");
```

### *Методы*

### activate()

Метод активирует расширение.

```
activate(): void;
```

### deactivate()

Метод деактивирует расширение.

```
deactivate(): void;
```

### addPlanes()

Метод добавляет плоскости сечения к уже существующим.

```
addPlanes(planes: ClippingPlaneDescription[]): void;
```

где:
`planes` – список описаний плоскостей сечения. Подробнее: ClippingPlaneDescription.

### setPlanes()

Метод задаёт плоскости сечения на основной сцене, уже существующие плоскости на сцене удаляются.

```
public setPlanes(planes: ClippingPlaneDescription[]): void;
```

где:
`planes` – список описаний плоскостей сечения. Подробнее: ClippingPlaneDescription.

## getPlanes()

Метод возвращает описания плоскостей сечения на сцене. Подробнее: ClippingPlaneDescription.

```
public getPlanes(planeIDs?: string[]): ClippingPlaneDescription[];
```

где:
`planeIDs` – список идентификаторов плоскостей сечения. Не обязательный параметр. Если ничего не определено, то возвращается описание всех плоскостей сечения.

## removePlanes()

Метод удаляет плоскости сечения.

```
public removePlanes(planeIDs?: string[]): void;
```

где:
`planeIDs` – список идентификаторов плоскостей сечения. Не обязательный параметр. Если ничего не определено, то удаляются все плоскости сечения.

## ClippingPlaneExtension.ClippingPlaneDescription

Описание плоскости сечения.

```
export type ClippingPlaneDescription = {
    normal: Point3,
    origin: Point3,
    guid?: string };
```

## normal

Нормаль плоскости сечения.

```
normal: Point3
```

где:
`normal` – координаты вектора нормали в мировом пространстве. Подробнее: Point3.

## origin

Точка, принадлежащая плоскости сечения. Также в эту точку помещается `ClippingPlaneViewObject` - вспомогательный визуальный объект для отображения плоскости.

```
origin: Point3
```

где:
`origin` – координаты точки в мировом пространстве. Подробнее: Point3.

## guid

Идентификатор плоскости сечения. Необязательный параметр. Может использоваться для выборочного удаления плоскостей сечения.

```
guid?: string
```

## DeleteButtonExtension

**DeleteButtonExtension** – расширение для удаления выделенных на сцене объектов. По нажатию на кнопку `Удалить выбранные элементы` в панели инструментов или по нажатию клавиши `Delete` на клавиатуре расширение вызовет delete метод модели со списком выделенных объектов.

Непосредственным удалением объектов занимаются владельцы данных объектов. Для этого им необходимо подписаться на событие удаления DELETE_OBJECTS_EVENT. Также владелец объекта должен передать в модель фильтр для собственных удаляемых объектов - addDeletionFilter.

При выделении объектов на сцене расширение проверяет, что все выбранные объекты соответствуют фильтрам удаления объектов с помощью метода canDelete . Если соответствуют, то удаление разрешено – можно удалить объекты нажатием кнопки либо нажатием клавиши `Delete`. В противном случае кнопка будет заблокированной, а нажатие клавиши `Delete` проигнорируется.

Расширение имеет имя `PilotWeb3D.DeleteButton`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/DeleteButton/D
eleteButton.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.DeleteButton");
```

### *Методы*

### activate()

Активировать расширение.

```
activate(): void;
```

### deactivate()

Деактивировать расширение.

```
deactivate(): void;
```

## FullScreenExtension

**FullScreenExtension** – расширение, которое позволяет перейти в полноэкранный режим. Также расширение добавляет кнопку на панель инструментов для управления полноэкранным режимом.

Используйте метод `activate()` для перехода в полноэкранный режим.
Расширение имеет имя `PilotWeb3D.FullScreen`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/FullScreen3D/F
ullScreen.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.FullScreen");
```

### *Методы*

### activate()

Перейти в полноэкранный режим.

```
activate(): void;
```

### deactivate()

Выход из полноэкранного режима.

```
deactivate(): void;
```

### getMode()

Получить режим отображения.

```
getMode(): FullScreenExtension.FullScreenMode;
```

Перечисление FullScreenExtension.FullScreenMode.

```
FullScreenExtension.FullScreenMode.NORMAL
```

Нормальный режим.

```
FullScreenExtension.FullScreenMode.FULLSCREEN
```

Полноэкранный режим

## MeasurementToolsExtension

**MeasurementToolsExtension** – расширение, которое предоставляет различные инструменты измерений.

Расширение имеет имя `PilotWeb3D.MeasurementTools`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/MeasurementTo
ls3D/MeasurementTools.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.MeasurementTools");
```

### *Методы*

### activate()

Активировать расширение

```
activate(): void;
```

### deactivate()

Деактивировать расширение

```
deactivate(): void;
```

## ModelsBrowserExtension

**ModelsBrowserExtension** – расширение, которое позволяет посмотреть дерево элементов загруженных моделей. Также расширение добавляет кнопку на панель интсрументов.

Используйте метод `activate()` для того, чтобы показать дерево элементов.
Расширение имеет имя `PilotWeb3D.ModelsBrowser`.

Пример подключения в `html`:

```
<link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/extensions/ModelsBrowser
3D/ModelsBrowser.css">
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/ModelsBrowser3
D/ModelsBrowser.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.ModelsBrowser");
```

### *Методы*

### activate()

Показать дерево элементов

```
activate(): void;
```

### deactivate()

Скрыть дерево элементов

```
deactivate(): void;
```

## RemarksExtension

**RemarksExtension** – расширение, которое позволяет задать точки замечаний на сцене. Замечания прикрепляются к графическим объектам на сцене, изменяя свое положение при изменении положения целевого объекта. Замечания рисуются на отдельном слое. Каждая точка замечания имеет статус – опционально отображаемую текстуру.

Расширение имеет имя `PilotWeb3D.Remarks`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/Remarks3D/Rema
rks.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
await viewer.extensionsLoader.loadExtension("PilotWeb3D.Remarks");
```

### *Свойства*

#### get remarkManager(): RemarkManager

Возвращает менеджер точек замечаний. Подробнее: RemarkManager.

```
public get remarkManager(): RemarkManager;
```

### *Методы*

#### activate()

Метод включает расширение.

```
activate(): void;
```

#### deactivate()

Метод выключает расширение.

```
deactivate(): void;
```

## RemarkManager

**RemarkManager** – менеджер замечаний, который предоставляет методы API для работы с точками замечаний.

```
export class RemarkManager {

  readonly events: PilotWeb3D.IEventsDispatcher;
  readonly remarkSceneName = 'RemarkViewObjectScene';

  public get placeRemarkOnClick(): boolean;
  public set placeRemarkOnClick(value: boolean);

  public get selectedRemarks(): RemarkViewObject[];

  public setActive(value: boolean): void;

  public getRemark(remarkId: string): RemarkViewObject | undefined;

  public addRemark(remarkParameters: RemarkObjectParameters,
statusParameters?: RemarkStatusParameters): RemarkViewObject;

  public removeRemarks(remarkIds?: string[]): void;

  public select(remarkId: string | string[]): void;

  public deselect(remarkId: string | string[]): void;

  public setRemarkStatus(remarkId: string, statusParameters:
RemarkStatusParameters): void;

  public setRemarksVisibility(visibiliity: boolean, remarkIds?: string[]):
void;

  public setRemarksLayerVisibility(visibiliity: boolean): void;

}
```

### *Поля*

### remarkSceneName : string

Наименование слоя замечаний.

```
readonly remarkSceneName = 'RemarkViewObjectScene';
```

### events : IEventsDispatcher

Диспетчер событий замечаний. Подробнее: IEventsDispatcher.
Список типов событий замечаний: RemarkEventMap.

```
  readonly events: PilotWeb3D.IEventsDispatcher;
```

### *Свойства*

## placeRemarkOnClick : boolean

Включает или выключает режим размещения точек замечаний по клику на сцене. Если `true`, то клик по объекту на сцене приведёт к добавлению точки замечания для данного объекта в месте клика. После добавления точки замечания, либо при клике в пустую область, режим сбрасывается, и свойство становится `false`.
При смене режима размещения точек возникает событие remarkPlacingModeChanged.

```
get placeRemarkOnClick(): boolean;
set placeRemarkOnClick(value: boolean);
```

По умолчанию: `false`.

## selectedRemarks : RemarkViewObject[]

Возвращает выбранные объекты замечаний либо пустой массив, если ни одно замечание не выбрано.
Подробнее: RemarkViewObject.

```
get selectedRemarks(): RemarkViewObject[];
```

По умолчанию: `[]`.

### *Методы*

## setActive()

Метод активирует или деактивирует менеджер точек замечаний.

```
setActive(value: boolean): void;
```

## getRemark()

Метод возвращает объект замечания с указанным ID либо `undefined`, если объект с указанным ID не найден.

```
public getRemark(remarkId: string): RemarkViewObject | undefined;
```

где:
`remarkId` – идентификатор объекта замечания.

## addRemark()

Метод добавляет точку замечания на слой замечаний.

```
public addRemark(remarkParameters: RemarkObjectParameters, statusParameters?:
RemarkStatusParameters): RemarkViewObject | null;
```

где:
`remarkParameters` – параметры точки замечания. Подробнее: RemarkObjectParameters.

`statusParameters` – параметры статуса замечания. Опциональный параметр. Подробнее: RemarkStatusParameters.

Возвращает добавленный на сцену объект замечания или `null`, если добавить точку не удалось. Подробнее: RemarkViewObject.

### removeRemarks()

Метод удаляет точки замечаний и освобождает ресурсы, выделенные для удаляемых точек.

```
public removeRemarks(remarkIds?: string[]): void;
```

где:
`remarkIds` – идентификаторы точек замечаний для удаления. Опциональный параметр. Если не задан, то удаляются все добавленные на сцену точки замечаний.

### select()

Метод выделяет замечания на документе.

```
select(remarkId: string | string[]): void;
```

где:
`remarkId` – идентификатор замечания или массив идентификаторов для выделения.

### deselect()

Метод снимает выделение замечаний на документе.

```
deselect(remarkId: string | string[]): void;
```

где:
`remarkId` – идентификатор замечания или массив идентификаторов для снятия выделения.

### clearSelection()

Метод снимает выделение с текущего выбранного замечания.

```
clearSelection(): void;
```

### setRemarkStatus()

Метод задает параметры статуса точки замечания.

```
public setRemarkStatus(remarkId: string, statusParameters:
RemarkStatusParameters): void;
```

где:
`remarkId` – идентификатор точки замечания для обновления статуса.

`statusParameters` – параметры статуса замечания. Подробнее: RemarkStatusParameters.

### setRemarksLayerVisibility()

Метод задает видимость слоя точек замечаний.

```
public setRemarksLayerVisibility(visibiliity: boolean): void;
```

где:

`visibiliity` – параметр видимости слоя замечаний. Если `true`, то слой замечаний отрисовывается в процессе рендера. В противном случае слой замечаний не рисуется, и объекты замечаний на сцене не показываются.

## RemarkEventMap

События замечаний.

```
interface RemarkEventMap {
  'remarkPlacingModeChanged' : Event;
  'remarkClicked' : PilotWeb3D.ClickedEvent;
}
```

### remarkPlacingModeChanged

```
  'remarkPlacingModeChanged' : Event;
```

Событие возникает при изменении свойства placeRemarkOnClick.

### remarkClicked

```
  'remarkClicked' : PilotWeb3D.ClickedEvent;
```

Событие возникает при клике по точке замечания. Подробнее: Events3D.

## RemarkViewObject

Графический объект, представляющий точку замечания. Добавляется на слой замечаний. Расширяет ViewObject.

```
export class RemarkViewObject extends PilotWeb3D.ViewObject {
  constructor(remarkParamters?: RemarkObjectParameters, statusParameters?:
RemarkStatusParameters);

  get remarkParameters(): RemarkObjectParameters;

  get statusParameters(): RemarkStatusParameters;

  updateRemark(parameters: RemarkObjectParameters): void;

  updateStatus(parameters: RemarkStatusParameters): void;
}
```

### *Конструктор*

```
  constructor(remarkParamters?: RemarkObjectParameters, statusParameters?:
RemarkStatusParameters);
```

где:

`remarkParamters` – параметры точки замечания, опциональный параметр. Если не заданы, то создается точка замечания со значениями по умолчанию. Подробнее: RemarkObjectParameters.

`statusParameters` – параметры статуса замечания, опциональный параметр. Подробнее: RemarkStatusParameters.

### *Свойства*

## remarkParameters() : RemarkObjectParameters

Возвращает текущие параметры замечания.

```
  get remarkParameters(): RemarkObjectParameters;
```

Подробнее: RemarkObjectParameters.

## statusParameters() : RemarkStatusParameters

Возвращает текущие параметры статуса замечания.

```
  get statusParameters(): RemarkStatusParameters;
```

Подробнее: RemarkStatusParameters.

### *Методы*

## updateRemark()

Метод обновляет параметры точки замечания. Внутри `parameters` можно определять только изменившиеся параметры точки замечания. Например, для изменения размера точки достаточно передать `{ markSize : newSize }`. При этом позиция точки, иконка замечания и другие параметры останутся неизменными. Подробнее: RemarkObjectParameters.

```
updateRemark(parameters: RemarkObjectParameters): void;
```

### updateStatus()

Метод обновляет параметры статуса точки замечания. Внутри `parameters` можно определять только изменившиеся параметры статуса. Например, для изменения размера статуса достаточно передать `{ statusSize : newSize }`. При этом смещение статуса, иконка статуса и другие параметры останутся неизменными. Подробнее: RemarkStatusParameters.

```
updateStatus(parameters: RemarkStatusParameters): void;
```

## RemarkObjectParameters

Параметры точки замечания.

```
export interface RemarkObjectParameters {
  remarkGuid?: string,
  targetModelGuid?: string,
  targetEntityGuid?: string,
  position?: Point3,
  relativePosition?: Point3,
  markSize?: { x: number, y: number },
  svgIcon?: string
}
```

### remarkGuid : string

Идентификатор точки замечания. Опциональный параметр. Если не задан, уникальный идентификатор генерируется автоматически.

### targetModelGuid : string

Идентификатор части модели, которой принадлежит целевой объект.
Необязательный параметр, используется совместно с `targetEntityGuid`. Если не указан, то поиск целевого объекта выполняется по всем частям модели.
Подробнее: ModelElement.modelPartId.

### targetEntityGuid : string

Идентификатор элемента модели, геометрия которого используется как целевой объект для привязки замечания.
Подробнее: ModelElement.id.

### position : Point3

Координаты точки замечания в мировом пространстве. Опциональный параметр. Если координаты не заданы, но задан целевой объект (targetEntityGuid) и относительное положение точки замечания (relativePosition), то абсолютное положение точки замечания рассчитывается, исходя из этих параметров. В противном случае используется значение по умолчанию: `{ x: 0, y: 0, z: 0 }`. Подробнее: Point3.

### relativePosition : Point3

Координаты точки замечания относительно целевого объекта. Опциональный параметр. Относительные координаты применяются только в том случае, если задан целевой объект (targetEntityGuid) и не заданы абсолютные координаты (position). В случае, если заданы и целевой объект, и абсолютные координаты, то относительные координаты будут рассчитаны, исходя из этих параметров. Подробнее: Point3.

### markSize : {x: number, y: number}

Размеры точки замечания. Опциональный параметр. Если не задан, то используются значения по умолчанию: `{ x: 25, y: 25 }`.

### svgIcon : string

Иконка точки замечания, строковое описание `svg`. Необязательный параметр.

## RemarkStatusParameters

Параметры статуса точки замечания.

```
export interface RemarkStatusParameters {
  visible?: boolean,
  mapColor?: PilotWeb3D.Color,
  statusSize?: { x: number, y: number },
  statusOffset?: { x: number, y: number },
  svgIcon?: string
}
```

### visible : boolean

Определяет видимость статуса замечания на сцене. Опциональный параметр. Если не задан, то используется значение по умолчанию: `false`.

### mapColor : PilotWeb3D.Color

Определяет цвет статуса замечания. Опциональный параметр. При отрисовке цвет текстуры замечания умножается на этот цвет. Если не задан, то используется значение по умолчанию: `new PilotWeb3D.Color(1, 1, 1, 1)`.
Подробнее: Color.

### statusSize : { x: number, y: number }

Определяет размеры текстуры статуса замечания в пикселях. Опциональный параметр. Если не задан, то используется значение по умолчанию: `{ x: 30, y: 30 }`.

### statusOffset : { x: number, y: number }

Определяет смещение текстуры статуса замечания относительно точки замечания. Опциональный параметр. Указывается в пикселях. Если не задан, то используется значение по умолчанию: `{ x: 25, y: 25 }`.

### svgIcon : string

Иконка статуса замечания, строковое описание `svg`. Необязательный параметр.

## ViewerSettingsExtension

**ViewerSettingsExtension** – расширение, которое позволяет посмотреть настройки просмотрщика 3D моделей. Также расширение добавляет кнопку на панель интсрументов.

Используйте метод `activate()` для того, чтобы показать диалог настроек просмотрщика 3D моделей.
Расширение имеет имя `PilotWeb3D.ViewerSettings`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/ViewerSettings
3D/ViewerSettings.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.ViewerSettings");
```

### *Методы*

### activate()

Показать диалог настроек просмотрщика 3D моделей

```
activate(): void;
```

### deactivate()

Скрыть диалог настроек просмотрщика 3D моделей

```
deactivate(): void;
```

## WasdNavigationExtension

**WasdNavigationExtension** – расширение, которое позволяет навигироваться по сцене с помощью клавиатуры, либо с помощью методов API.
Клавиши клавиатуры: `W` - вперед, `A` - влево, `S` - назад, `D` - вправо, `Q` - вниз, `E` - вверх, `Shift` - ускорение.

Расширение имеет имя `PilotWeb3D.WasdNavigation`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/WasdNavigation
3D/WasdNavigation.min.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.WasdNavigation");
```

### *Поля*

#### standardSpeed

Скорость перемещения камеры по умолчанию.

```
standardSpeed: number = 5;
```

#### increasedSpeed

Скорость перемещения камеры при ускоренном движении.

```
increasedSpeed: number = 30;
```

#### timeAcceleratingToSpeedLimit

Время разгона камеры с standardSpeed до increasedSpeed.

```
timeAcceleratingToSpeedLimit: number = 500;
```

#### speedGoingThroughObstacles

Скорость перемещения камеры сквозь препятствия.

```
speedGoingThroughObstacles: number = 0.5;
```

### speedGoingThroughObstaclesIncreased

Скорость перемещения камеры сквозь препятствия при ускоренном движении.

```
speedGoingThroughObstaclesIncreased: number = 1;
```

### distanceToObstacleWhereStartToSlowDown

Расстояние до препятствия, начиная с которого камера начинает замедляться.

```
distanceToObstacleWhereStartToSlowDown: number = 500;
```

#### *Методы*

### activate()

Включить подписку на события клавиатуры.

```
activate(): void;
```

### deactivate()

Выключить подписку на события клавиатуры.

```
deactivate(): void;
```

### setImpulseDirection()

Метод задаёт направление движения камеры.

```
setImpulseDirection(dir: Direction, add: boolean): void;
```

где:
`dir` – Направление движения отсносительно камеры. Подробнее: Direction.
`add` – `true` для добавления, `false` для вычитания.

### getImpulseDirection()

Метод возвращает направление движения камеры.

```
getImpulseDirection(): Direction;
```

Возвращает перечисление Direction.

### setIncreasedImpulse()

Метод активирует ускоренное движение.

```
setIncreasedImpulse(isIncreased: boolean): void;
```

где:
`isIncreased` – `true`, для ускоренного движения.

Перечисление WasdNavigationExtension.Direction

Направления относительно камеры.

```
export enum Direction {
  None = 0,
  Forward = 1 << 1,
  Left = 1 << 2,
  Backward = 1 << 3,
  Right = 1 << 4,
  Down = 1 << 5,
  Up = 1 << 6,
}
```

# Базовые расширения 2D

## Базовые расширения 2D

Компонент работы с документами поддерживает расширение функциональности с помощью подключаемых модулей расширений. Система **PilotCloud** включает в себя набор модулей расширений, которые при необходимости могут быть подключены. Каждый модуль расширения в этом списке вносит новые возможности в компоненты системы. Если вам нужно добавить специализированные функции Вы можете создавать свои собственные расширения.

## RemarksExtension

**RemarksExtension** – расширение, которое позволяет задать точки замечаний на документе. Замечания рисуются на отдельном слое. Также, каждая точка замечания имеет статус - опционально отображемую `svg` иконку.

Расширение имеет имя `PilotWeb2D.Remarks`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/Remarks2D/Rema
rks.min.js"></script>
```

Пример подключения в `javascript`:

```
const htmlDiv = document.getElementById('pilotViewer')
const configuration = new PilotWeb2D.Viewer2DConfiguration();
const viewer = PilotWeb2D.CreateViewer(htmlDiv, configuration);
await viewer.start();
// загружаем расширение и активируем его
const remarksExtension = await
viewer.extensionsLoader.loadExtension("PilotWeb2D.Remarks");
remarksExtension.activate();
```

### *Свойства*

### get RemarksManager(): RemarksManager

Возвращает менеджер точек замечаний. Подробнее: RemarksManager.

```
get remarkManager(): RemarksManager;
```

### *Методы*

### activate()

Метод включает расширение.

```
activate(): boolean;
```

### deactivate()

Метод выключает расширение.

```
deactivate(): boolean;
```

### getClickPage()

Метод получает страницу, по которой был произведен клик мыши. Подробнее: IDocumentPage.

```
protected getClickPage(event: MouseEvent): PilotWeb2D.IDocumentPage |
undefined
```

### getClickPoint()

Метод получает координаты точки без учета масштаба страницы. Подробнее: Point2. Чтобы получить точки в масштабе страницы необходимо координаты точки умножить на масштаб страницы.

```
protected getClickPoint(event: MouseEvent, offsetX = 0, offsetY = 0):
PilotWeb2D.Point2
```

## RemarksManager

**RemarksManager** – менеджер замечаний, который предоставляет методы API для работы с точками замечаний.

```
class RemarksManager {
  setActive(value: boolean): void;
  getRemark(remarkId: string): Remark | undefined;
  addRemark(remarkParams: RemarkParameters, status?: RemarkStatus):
Promise<Remark>;
  removeRemarks(ids: string[]): boolean;
  select(remarkId: string): void;
  deselect(remarkId: string): void;
  setStatus(remarkIs: string, status: RemarkStatus): boolean;
  setRemarksVisibility(visibility: boolean, remarkIds?: string[]): void;
}
```

#### *Методы*

### setActive()

Метод активирует или деактивирует менеджер точек замечаний.

```
setActive(value: boolean): void;
```

## getRemark()

Метод возвращает объект замечания с указанным идентификатором либо `undefined`, если объект не найден. Подробнее Remark

```
getRemark(remarkId: string): Remark | undefined;
```

где:
`remarkId` – идентификатор замечания.

## addRemark()

Метод добавляет точку замечания на слой замечаний.

```
addRemark(remarkParams: RemarkParameters, status?: RemarkStatus):
Promise<Remark>;
```

где:
`remarkParams` – параметры точки замечания. Подробнее: RemarkParameters.
`status` – параметры статуса замечания, опциональный параметр. Подробнее: RemarkStatus.
Возвращает добавленный на документ объект замечания. Подробнее: Remark.

## removeRemarks()

Метод удаляет точки замечаний.

```
removeRemarks(ids: string[]): boolean;
```

где:
`ids` – идентификаторы точек замечаний для удаления, опциональный параметр. Если не задан, то удаляются все добавленные на сцену точки замечаний.

## select()

Метод управляет селектированием точек замечаний. Выбранное замечание может быть только одно.
При вызове метода `select` выбирается замечание, идентификатор которого был передан как аргумент, а предыдущий выбор сбрасывается. Если замечание с нужным идентификатором не найдено, либо `remarkId` неопределён, то ничего не происходит.

```
select(remarkId: string): void;
```

где:
`remarkId` – идентификатор точки замечания для выбора.

## deselect()

Метод снимает селект с заданого замечания.

```
deselect(remarkId: string): void;
```

где:
`remarkId` – идентификатор точки замечания.

### clearSelection()

Метод снимает селект с текущего выбранного замечания.

```
clearSelection(): void;
```

### setStatus()

Метод задает параметры статуса точки замечания.

```
setStatus(remarkId: string, status: RemarkStatus): boolean
```

где:
`remarkId` – идентификатор точки замечания для обновления статуса.
`status` – параметры статуса замечания. Подробнее: RemarkStatus.

### setRemarksVisibility()

Метод задает видимость точек замечаний.

```
setRemarksVisibility(visibility: boolean, remarkIds: string[]): void
```

где:
`visibiliity` – параметр видимости замечаний. `remarkIds` – идентификаторы замечаний.

## Remark

Объект представляющий точку замечания, добавляется на слой замечаний.

```
class Remark {
  id: string;
  pageNumber: number;
  container: RemarkHtmlContainer;
  positionX: number;
  positionY: number;
  type?: string;
}
```

### id : string

Поле содержит идентификатор замечания

### pageNumber: number

Поле содержит номер страницы, на которой размещено замечание.

### container: RemarkHtmlContainer

Поле содержит объект с описанием HTML-элементов к замечанию. Подробнее: RemarkHtmlContainer.

### positionX: number

Поле содержит позицию по оси x на документе.

### positionY: number

Поле содержит позицию по оси y на документе.

### type: string

Поле содержит дополнительные параметры для идентификации замечания.

## RemarkHtmlContainer

Объект представляющий описание HTML-элементов к замечанию.

```
class RemarkHtmlContainer {
  containerHtmlElement: HTMLElement;
  remarkHtmlElement: HTMLElement;
  statusHtmlElement: HTMLElement;
}
```

### containerHtmlElement: HTMLElement

Поле содержит HTML-контейнер для всего замечания.

### remarkHtmlElement: HTMLElement

Поле содержит HTML-контейнер для точки замечания.

### statusHtmlElement: HTMLElement

Поле содержит HTML-контейнер для статуса замечания.

## RemarkParameters

Параметры точки замечания.

```
interface RemarkParameters {
  id: string;
  positionX: number;
  positionY: number;
  htmlElement?: HTMLElement | string;
  pageNumber?: number;
  containerClass?: string;
  mark?: string;
  type?: string;
}
```

### id: string

Идентификатор замечания.

### positionX: number

Позиция по оси x на документе.

### positionY: number

Позиция по оси y на документе.

### htmlElement: HTMLElement | string

Описание замечания в виде HTML-элементов или строковое описание `svg`. Параметр не обязательный. Если этот параметр не задан, то замечание будет отрисовываться стилем по умолчанию.

### pageNumber: number

Номер страницы, на которой необходимо разместить замечание. Параметр не обязательный. Если этот параметр не задан, то замечание разместится на первой странице.

### containerClass: string

Имя `css` класса, который будет применен к общему контейнеру замечания. Параметр не обязательный.

### mark: string

Метка на замечании. Параметр не обязательный.

### type: string

Дополнительные параметры описания замечания. Параметр не обязательный.

## RemarkStatus

Параметры статуса точки замечания.

```
interface RemarkStatus {
  visible?: boolean;
  statusOffsetX?: number;
  statusOffsetY?: number;
  htmlElement?: HTMLElement | string;
}
```

### visible: boolean

Определяет видимость статуса замечания, опциональный параметр. Если не задан, то используется значение по умолчанию: `false`.

### statusOffsetX: number

Смещение статуса по оси x от центра точки замечания. Параметр не обязательный.

### statusOffsetY: number

Смещение статуса по оси y от центра точки замечания. Параметр не обязательный.

### htmlElement: HTMLElement | string

Описание замечания в виде HTML-элементов или строковое описание `svg`. Параметр не обязательный.

## ZoomExtension

**ZoomExtension** – расширение, которое добавляет элементы управления масштабом документа в панель инструменотов компонента **PIlotWeb2D**.

Расширение имеет имя `PilotWeb2D.Zoom`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/extensions/Zoom2D/Zoom.mi
n.js"></script>
```

Пример подключения в `javascript`:

```
const htmlDiv = document.getElementById('pilotViewer')
const configuration = new PilotWeb2D.Viewer2DConfiguration();
const viewer = PilotWeb2D.CreateViewer(htmlDiv, configuration);
await viewer.start();
// загружаем расширение. Расширение активируется автоматически
const extension = await
viewer.extensionsLoader.loadExtension("PilotWeb2D.Zoom");
```

# Примеры расширений 3D

## CameraChangeExtension

**CameraChangeExtension** – пример расширения, которое обрабатывает события перемещения камеры. Расширение имеет имя `CameraChangeExtensionSample`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/CameraChangeExten
sion/CameraChangeExtension.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("CameraChangeExtensionSample");
```

## ElementPropertiesExtension

**ElementPropertiesExtension** – пример расширения, которое позволяет просматривать свойства выделенного BIM-элемента. Расширение имеет имя `ElementPropertiesExtensionSample`.

Пример подключения в `html`:

```
<link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/samples/ElementPropertie
sExtension/ElementPropertiesExtension.css">
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/ElementProperties
Extension/ElementPropertiesExtension.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("ElementPropertiesExtensionSample");
```

## RemarksUIExtension

**RemarksUIExtension** – пример расширения, которое использует API расширения RemarksExtension для добавления и редактирования точек замечаний на сцене. Расширение имеет имя `RemarksUIExtensionSample`.

Пример подключения в `html`:

```
<link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/samples/RemarksUIExtensi
on/RemarksUIExtension.css">
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/RemarksUIExtensio
n/RemarksUIExtension.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
await viewer.extensionsLoader.loadExtension("RemarksUIExtensionSample");
```

## SceneObserverExtension

**SceneObserverExtension** – пример расширения, которое позволяет просматривать дерево графических объектов на сцене. Также расширение добавляет кнопку на панель интсрументов. Расширение имеет имя `PilotWeb3D.SceneObserver`.

Пример подключения в `html`:

```
<link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/samples/SceneObserverExt
ension/SceneObserverExtension.css">
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/SceneObserverExte
nsion/SceneObserverExtension.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("PilotWeb3D.SceneObserver");
```

### *Методы*

### activate()

Показать дерево графических объектов.

```
activate(): void;
```

### deactivate()

Скрыть дерево графических объектов.

```
deactivate(): void;
```

## SelectionExtension

**SelectionExtension** – пример расширения, которое показывает возможности работы различного `API` компонента **PilotWeb3D**. Расширение имеет имя `Selection3DExtension`.

Пример подключения в `html`:

```
<link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/samples/SelectionExtensi
on/SelectionExtension.css">
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/SelectionExtensio
n/SelectionExtension.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("Selection3DExtension");
```

## SetPivotPositionExtension

**SetPivotPositionExtension** – пример расширения, которое показывает возможности работы с опорной точкой камеры. Расширение имеет имя `SetPivotPositionExtensionSample`.

Пример подключения в `html`:

```
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/SetPivotPositionE
xtension/SetPivotPositionExtension.js"></script>
```

Пример подключения в `javascript`:

```
var htmlDiv = document.getElementById('pilotViewer')
viewer = PilotWeb3D.CreateViewer(htmlDiv);
await viewer.start();
viewer.extensionsLoader.loadExtension("SetPivotPositionExtensionSample");
```

# Примеры расширений 2D

## RemarksUIExtension

**RemarksUIExtension** – пример расширения, которое использует API расширения RemarksExtension для добавления и редактирования точек замечаний на сцене. Расширение имеет имя `RemarksUIExtensionSample`.

Пример подключения в `html`:

```
<link rel="stylesheet"
href="https://pilotcloud.ascon.net/components/24.2.0/samples/RemarksUIExtensi
on2D/RemarksUIExtension.css">
<script
src="https://pilotcloud.ascon.net/components/24.2.0/samples/RemarksUIExtensio
n2D/RemarksUIExtension.js"></script>
```

Пример подключения в `javascript`:

```
const htmlDiv = document.getElementById('pilotViewer');
const configuration = new PilotWeb2D.Viewer2DConfiguration();
const viewer = PilotWeb2D.CreateViewer(htmlDiv, configuration);
await viewer.start();
await viewer.extensionsLoader.loadExtension("Remarks2D.UIExtensionSample");
```