Web设计器 API 介绍

本文档提供了ar-designer-types.d.ts文件的内容，它可以作为你项目的插件使用。在这里你会发现Web Designer中可用的属性和功能的用法和描述。

```
                    /* ===== designer ===== */
/**
 * Type of **GrapeCity.ActiveReports.WebDesigner** object exported by
**web-designer.js** module.
 */
export type DesignerApi = {
    /**
     * Creates the default **DesignerOptions** object to be passed to
**renderApplication()** function.\
     * This object includes both required and optional Designer
settings.
     *
     * **Example:**
     * ```javascript
     * var designerOptions =
GrapeCity.ActiveReports.WebDesigner.createDesignerOptions();
     * ```
     */
    createDesignerOptions: () => DesignerOptions;
    /**
     * Renders **Web Designer** to **<div>** element with id
**designerElementId** using the specified **DesignerOptions** object.
     *
     * **Example:**
     * ```javascript
     *
GrapeCity.ActiveReports.WebDesigner.renderApplication('designer-id',
designerOptions);
     * ```
     *
     * @param designerElementId string
     * @param designerOptions DesignerOptions object
     */
    renderApplication: (designerElementId: string, designerOptions:
DesignerOptions) => Promise<any>;
    /**
     * Returns focus to Designer. Focus may be lost when plug-in
components are opened/closed.\
     * Returning focus is essential to continue using Designer hotkeys
like Ctrl+Z (undo), Ctrl+Y (redo), etc.
     *
     * **Example:**
     * ```javascript
     * GrapeCity.ActiveReports.WebDesigner.focus();
     * ```
     */
    focus: () => void;
    /**
     * Returns **AboutInfo** object containing information about
application name/version and product title/version.
```

```
     *
     * **Example:**
     * ```javascript
     * var aboutInfo =
GrapeCity.ActiveReports.WebDesigner.getAboutInfo();
     * ```
     */
    getAboutInfo: () => AboutInfo;

    /**
     * Returns **HelpInfo[]** containing title/link pairs of
Designer-related Help pages.
     *
     * **Example:**
     * ```javascript
     * var helpInfos =
GrapeCity.ActiveReports.WebDesigner.getHelpInfos();
     * ```
     */
    getHelpInfos: () => HelpInfo[];
    /**
     * This object includes functions allowing to create/open/save
report, etc.
     */
    api: ReportingApi;
};
export type DesignerOptions = {
    /**
     * Specifies locale used for displaying Designer.
     * If **locale** is not specified explicitly here, the locale
corresponding to the browser preferences is used.
     *
     * **Example:**
     * ```javascript
     * designerOptions.locale = 'ja';
     * ```
     */
    locale?: 'en' | 'ja' | 'zh';
    /**
     * Specifies custom localization data.
     *
     * **Example:**
     * ```javascript
     * designerOptions.localeData = customLocaleData;
     * ```
     */
    localeData?: {
        /** namespace */
        ns: string;

        /** locale */
        lng: string;

        /** localization resources */
        resources: any;
    }[];
    /**
```

```
     * Specifies the default measurement units used in Designer.
     * If **units** are not specified explicitly here, they are
identified depending on **locale**.
     *
     * **Example:**
     * ```javascript
     * designerOptions.units = 'cm';
     * ```
     */
    units?: 'in' | 'cm';
    /**
     * Specifies the list of fonts displayed in font properties
drop-downs all over Designer.
     * If **fonts** are not specified explicitly here, the default list
of fonts is used.
     *
     * **Example:**
     * ```javascript
     * designerOptions.fonts = ['Arial', 'Courier New', 'Times New
Roman'];
     * ```
     */
    fonts?: string[];
    /**
     * It is possible to limit and/or reorder available report items.
     * Specify comma-separated report items keys from this list:
     *
     * ```none
     * BandedList, Barcode, Bullet, Chart, CheckBox, Container,
FormattedText, Image, InputField, Line, List,
     * OverflowPlaceholder, Shape, Sparkline, Subreport, Table,
TableOfContents, Tablix, TextBox
     * ```
     *
     * **Example:**
     * ```javascript
     * designerOptions.reportItems =
'TextBox,CheckBox,Table,Chart,Image';
     * ```
     */
    reportItems?: string;
    /**
     * Customizable report items features are specified here.
     */
    reportItemsFeatures: {
        /** Barcode features */
        barcode: {
            /**
             * Overrides the default symbology used for newly-created
barcodes.\
             * By default new barcodes are created with QR Code
symbology.
             *
             * **Example:**
             * ```javascript
             *
designerOptions.reportItemsFeatures.barcode.defaultSymbology =
```

```
'Code_128_A';
             * ```
             */
            defaultSymbology?: BarcodeSymbology;
            /**
             * Limits the list of barcode symbologies available for
creation.\
             * By default all barcode symbologies supported by
ActiveReports are available.
             *
             * **Example:**
             * ```javascript
             * designerOptions.reportItemsFeatures.barcode.symbologies =
['Code_128_A', 'Code_128_B', 'Code_128_C'];
             * ```
             */
            symbologies?: BarcodeSymbology[];
        };
        /** Table features */
        table: {
            /**
             * Specifies whether **Table Header** needs to be
auto-filled when a field is dropped to **Table Details**.\
             * For example, if **ProductName** field is dropped to
**Details**, **Product Name** value is set to **Header**.\
             * By default this feature is **enabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.reportItemsFeatures.table.autoFillHeader
= false;
             * ```
             */
            autoFillHeader: boolean;
            /**
             * Specifies whether **Table Footer** needs to be
auto-filled when a field is dropped to **Table Details**.\
             * For example, if **ProductName** field is dropped to
**Details**, **=Count([ProductName])** value is set to **Footer**.\
             * By default this feature is **disabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.reportItemsFeatures.table.autoFillFooter
= true;
             * ```
             */
            autoFillFooter: boolean;
            /**
             * Specifies whether vertical merge of cells is enabled
within **Table Header**, **Details** and **Footer**.\
             * By default this feature is **enabled**.
             *
             * **Example:**
             * ```javascript
             *
designerOptions.reportItemsFeatures.table.canMergeCellsVertically =
```

```
false;
             * ```
             */
            canMergeCellsVertically: boolean;
        };
        /** Tablix features */
        tablix: {
            /**
             * Specifies whether **Tablix Corner Cell** needs to be
auto-filled when a field is dropped to **Tablix Row Group Cell**.\
             * For example, if **ProductName** field is dropped to **Row
Group Cell**, **Product Name** value is set to **Corner Cell**.\
             * By default this feature is **enabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.reportItemsFeatures.tablix.autoFillCorner
= false;
             * ```
             */
            autoFillCorner: boolean;
            /**
             * Specifies whether Tablix Wizard is available for
creating/editing Tablix.\
             * By default this feature is **enabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.reportItemsFeatures.tablix.canUseWizard =
false;
             * ```
             */
            canUseWizard: boolean;
        };
    };
    /**
     * When **lockLayout** is enabled, it is only possible to modify
properties of existing report items.\
     * I.e., adding a new report item or deleting of an existing one is
not possible as well as other operations that modify report layout
structure.\
     * By default this feature is **disabled**.
     *
     * **Example:**
     * ```javascript
     * designerOptions.lockLayout = true;
     * ```
     */
    lockLayout: boolean;
    /**
     * When **restoreUnsavedReport** is **enabled**, the last unsaved
report can be restored if browser tab or browser itself gets
accidentally closed.\
     * In case **restoreUnsavedReport** is **disabled**, the
aforementioned functionality is not available.\
     * By default this feature is **enabled**.
     *
```

```
     * **Example:**
     * ```javascript
     * designerOptions.restoreUnsavedReport = false;
     * ```
     */
    restoreUnsavedReport: boolean;
    /**
     * If **reportInfo.id** is specified, the corresponding report will
be opened in Designer when Designer application is rendered.
     *
     * **Example:**
     * ```javascript
     * designerOptions.reportInfo.id = 'MyReport.rdlx';
     * ```
     */
    reportInfo: {
        /** report id */
        id?: string | null;
    };
    /**
     * Specifies the expression syntax used in Designer:
     * 'i11n' - interpolation syntax, 'rdl' - "old" rdl expression
syntax
     * By default the interpolation syntax is used for expressions.
     *
     * **Example:**
     * ```javascript
     * designerOptions.expressionSyntax = 'rdl';
     * ```
     */
    expressionSyntax: 'i11n' | 'rdl';
    /**
     * DO NOT DOCUMENT THIS PROPERTY!\
     * Specifies custom About information - useful for white-label Web
Designer.
     */
    aboutInfo: AboutInfo | {};
    /**
     * DO NOT DOCUMENT THIS PROPERTY!\
     * Specifies custom Help information - useful for white-label Web
Designer.
     */
    helpInfos?: HelpInfo[];
    /**
     * DO NOT DOCUMENT THIS PROPERTY!\
     * Documentation links are specified here.
     */
    documentation: {
        /**
         *  Specifies **Designer** home documentation link.
         */
        home?: string;
        /**
         * Specifies **Designer** report items transformation
documentation link.
         */
        reportItemsTransformation?: string;
```

```
    };
    /**
     * You can plug-in **Report Viewer** component by providing
**openViewer** function to **DesignerOptions** object.\
     * When **openViewer** is passed to **DesignerOptions**, **Preview**
button appears in Designer application bar.
     *
     * **Example:**
     * ```javascript
     * designerOptions.openViewer = function(options) {
     *     // ... create viewer and open report
     * };
     * ```
     *
     * @param options ViewerOptions object
     */
    openViewer?: (options: ViewerOptions) => void;
    /** **File View** settings */
    fileView: {
        /**
         * Specifies whether **File View** tab needs to be shown.\
         * **File View** tab is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.fileView.visible = false;
         * ```
         */
        visible: boolean;
    };
    /** **Save** button settings */
    saveButton: {
        /**
         * Specifies whether **Save** button needs to be shown.\
         * **Save** button is **not visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.saveButton.visible = true;
         * ```
         */
        visible: boolean;
    };
    /** **Save As** button settings */
    saveAsButton: {
        /**
         * Specifies whether **Save As** button needs to be shown.\
         * **Save As** button is **not visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.saveAsButton.visible = true;
         * ```
         */
        visible: boolean;
    };
    /** **Open** button settings */
```

```
    openButton: {
        /**
         * Specifies whether **Open** button needs to be shown.\
         * **Open** button is **not visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.openButton.visible = true;
         * ```
         */
        visible: boolean;
    };
    /** **Insert** tab settings */
    insertTab: {
        /**
         * Specifies whether **Insert** tab needs to be shown in
Designer application bar.\
         * **Tool Box** and **Insert** tab are interchangeable.\
         * **Insert** tab is **not visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.insertTab.visible = true;
         * ```
         */
        visible: boolean;
    };
    /** **Report Explorer** settings */
    reportExplorer: {
        /**
         * Specifies whether **Report Explorer** button needs to be
shown.\
         * **Report Explorer** button is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.reportExplorer.visible = false;
         * ```
         */
        visible: boolean;
    };
    /** **Group Editor** settings */
    groupEditor: {
        /**
         * Specifies whether **Group Editor** button needs to be shown.\
         * **Group Editor** button is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.groupEditor.visible = false;
         * ```
         */
        visible: boolean;
    };
    /** **Tool Box** settings */
    toolBox: {
        /**
```

```
         * Specifies whether left-side menu **Tool Box** needs to be
shown.\
         * **Tool Box** is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.toolBox.visible = false;
         * ```
         */
        visible: boolean;
    };
    /** **Properties** tab settings */
    propertiesTab: {
        /**
         * Specifies whether **Properties** tab needs to be shown.\
         * **Properties** tab is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.propertiesTab.visible = false;
         * ```
         */
        visible: boolean;
        /**
         * Specifies available properties modes.\
         * The default value is **Both**.
         *
         * **Example:**
         * ```javascript
         * designerOptions.propertiesTab.mode = 'Basic';
         * ```
         */
        mode: 'Basic' | 'Advanced' | 'Both';
        /**
         * Relevant only when mode is **Both**.\
         * If **undefined**, the last used properties mode is set.
         *
         * **Example:**
         * ```javascript
         * designerOptions.propertiesTab.defaultMode = 'Advanced';
         * ```
         */
        defaultMode?: 'Basic' | 'Advanced';
    };
    /** **Data** tab settings */
    dataTab: {
        /**
         * Specifies whether **Data** tab needs to be shown.\
         * **Data** tab is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.dataTab.visible = false;
         * ```
         */
        visible: boolean;
        /** **Data Sources** section settings */
```

```
        dataSources: {
            /**
             * Specifies whether **Data Sources** section needs to be
shown.\
             * **Data Sources** section is **visible** by default.
             *
             * **Example:**
             * ```javascript
             * designerOptions.dataTab.dataSources.visible = false;
             * ```
             */
            visible: boolean;
            /**
             * Specifies whether it is possible to modify (including
add/edit/remove) data sources.\
             * By default this feature is **disabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.dataTab.dataSources.canModify = true;
             * ```
             */
            canModify: boolean;
            /** Options for data sources and data source editor */
            options: {
                /** Specifies the list of predefined data providers
available in data source editor.\
                 * By default all the predefined providers are present.
                 *
                 * **Example:**
                 * ```javascript
                 *
designerOptions.dataTab.dataSources.options.predefinedProviders =
['SQL', 'JSON'];
                 * ```
                 */
                predefinedProviders?: ('SQL' | 'OLEDB' | 'ODBC' | 'JSON'
| 'CSV' | 'XML')[],
                /** Specifies the list of OLE DB providers available in
data source editor.\
                 * By default 'Microsoft.Jet.OLEDB.4.0', 'SQLOLEDB.1',
'MSDataShape.1', 'MSDASQL.1' are present.
                 *
                 * **Example:**
                 * ```javascript
                 *
designerOptions.dataTab.dataSources.options.oleDbProviders =
['Microsoft.Jet.OLEDB.4.0', 'SQLOLEDB.1'];
                 * ```
                 */
                oleDbProviders?: string[],
                /** Specifies the list of custom data providers
available in data source editor.\
                 * By default there are no custom data providers
present.
                 *
                 * **Example:**
```

```
                 * ```javascript
                 *
designerOptions.dataTab.dataSources.options.customProviders = [{ key:
'CDP', name: 'Custom Data Provider' }];
                 * ```
                 */
                customProviders?: {
                    /**
                     * key - data provider identifier\
                     * This value is used for
**DataSource.ConnectionProperties.DataProvider** property.
                     */
                    key: string;

                   /**
                     * name - data provider label\
                     * This value is used as a friendly data provider
label in UI.
                     */
                    name: string;
                }[],
            },
        };
        /** **Data Sets** section settings */
        dataSets: {
            /**
             * Specifies whether **Data Sets** section needs to be
shown.\
             * **Data Sets** section is **visible** by default.
             *
             * **Example:**
             * ```javascript
             * designerOptions.dataTab.dataSets.visible = false;
             * ```
             */
            visible: boolean;
            /**
             * Specifies whether it is possible to modify (including
add/edit/remove) data sets.\
             * By default this feature is **disabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.dataTab.dataSets.canModify = true;
             * ```
             */
            canModify: boolean;
        };
        /** **Parameters** section settings */
        parameters: {
            /**
             * Specifies whether **Parameters** section needs to be
shown.\
             * **Parameters** section is visible by default.
             *
             * **Example:**
             * ```javascript
```

```
             * designerOptions.dataTab.parameters.visible = false;
             * ```
             */
            visible: boolean;
            /**
             * Specifies whether it is possible to modify (including
add/edit/remove) report parameters.\
             * By default this feature is **enabled**.
             *
             * **Example:**
             * ```javascript
             * designerOptions.dataTab.parameters.canModify = false;
             * ```
             */
            canModify: boolean;
        };
        /** **Common Values** section settings */
        commonValues: {
            /**
             * Specifies whether **Common Values** section needs to be
shown.\
             * **Common Values** section is **visible** by default.
             *
             * **Example:**
             * ```javascript
             * designerOptions.dataTab.commonValues.visible = false;
             * ```
             */
            visible: boolean;
        };
    };
    /** **Grid Size** editor settings */
    gridSize: {
        /**
         * Specifies whether **Grid Size** editor in **Status Bar**
needs to be shown.\
         * **Grid Size** editor is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.gridSize.visible = false;
         * ```
         */
        visible: boolean;
        /**
         * If **Grid Size** editor is **not visible**, it is possible to
specify **Grid Size** value in **in/cm** (i.e. inches or centimeters).
         *
         * **Example:**
         * ```javascript
         * designerOptions.gridSize.value = '0.5in';
         * ```
         */
        value?: string;
    };
    /** **Show Grid** toggle settings */
    showGrid: {
```

```
        /**
         * Specifies whether **Show Grid** toggle in Status Bar needs to
be shown.\
         * **Show Grid** toggle is **visible** by default.
         *
         * **Example:**
         * ```javascript
         * designerOptions.showGrid.visible = false;
         * ```
         */
        visible: boolean;
        /**
         * If **Show Grid** toggle is **not visible**, it is possible to
specify **Show Grid** value as *true* or *false*.
         *
         * **Example:**
         * ```javascript
         * designerOptions.showGrid.value = false;
         * ```
         */
        value?: boolean;
    };
    disableFocusTimer: boolean;
    /** server-related settings */
    server: {
        /** Specifies the base URL for Designer Server API.\
         * By default its value is 'api'.
         *
         * **Example:**
         * ```javascript
         * designerOptions.server.url = 'api/designer';
         * ```
         */
        url: string;
    };
    /**
     * You can implement custom logic for updating route when edited
report `id` gets updated in Designer.\
     * Report `id` is updated on creating/opening/saving a report -
after that `updateRoute` is called.
     *
     * By default `updateRoute` is `undefined` so route remains
unchanged.
     *
     * **Example:**
     * ```javascript
     * designerOptions.updateRoute = function(options) {
     *     // ... custom logic for updating route
     * };
     * ```
     *
     * @param options UpdateRouteOptions object
     */
    updateRoute?: (options: UpdateRouteOptions) => void;
};
type Report = Record<string, any>;
type ReportTemplate = Record<string, any>;
```

```
export type BarcodeSymbology = 'Ansi39' | 'Ansi39x' | 'BC412' |
'Codabar' | 'Code_11' | 'Code_128_A' | 'Code_128_B' | 'Code_128_C' |
'Code_128auto'
    | 'Code_2_of_5' | 'Code_93' | 'Code25intlv' | 'Code39' | 'Code39x' |
'Code49' | 'Code93x' | 'DataMatrix' | 'EAN_13' | 'EAN_8' | 'EAN128FNC1'
    | 'GS1QRCode' | 'HIBCCode128' | 'HIBCCode39' | 'IATA_2_of_5' |
'IntelligentMail' | 'IntelligentMailPackage' | 'ISBN' | 'ISMN' | 'ISSN'
    | 'ITF14' | 'JapanesePostal' | 'Matrix_2_of_5' | 'MaxiCode' |
'MicroPDF417' | 'MicroQRCode' | 'MSI' | 'Pdf417' | 'Pharmacode' |
'Plessey'
    | 'PostNet' | 'PZN' | 'QRCode' | 'RM4SCC' | 'RSS14' | 'RSS14Stacked'
| 'RSS14StackedOmnidirectional' | 'RSS14Truncated' | 'RSSExpanded'
    | 'RSSExpandedStacked' | 'RSSLimited' | 'SSCC_18' | 'Telepen' |
'UCCEAN128' | 'UPC_A' | 'UPC_E0' | 'UPC_E1';
export type ViewerOptions = {
    /** element id where to render viewer */
    element: string;
    /** locale passed by Designer */
    locale: 'en' | 'ja' | 'zh';
    /** application title passed by Designer */
    applicationTitle: string;
    /** information on the report to-be-previewed */
    reportInfo: {
        /** report id */
        id: string;

        /** Report content in JSON format that can be useful for report
viewers with in-browser rendering. */
        content: Report;

        /** report name */
        name: string;

        /** Specifies whether the report to-be-previewed is an existing
report saved on server side. */
        isTemporary: boolean;
    };
};
export type UpdateRouteOptions = {
    /** report id */
    id: string;
};
export type ReportingApi = {
    /**
     * Indicates whether report has unsaved changes.
     *
     * **Example:**
     * ```javascript
     * var hasUnsavedChanges =
GrapeCity.ActiveReports.WebDesigner.api.isReportDirty();
     * if (hasUnsavedChanges) console.log('Currently edited report has
unsaved changes.');
     * ```
     */
    isReportDirty: () => boolean;
    /**
     * Returns information about the currently edited report.
```

```
     *
     * **Example:**
     * ```javascript
     * var reportInfo =
GrapeCity.ActiveReports.WebDesigner.api.getReportInfo();
     * console.log(`Report "${reportInfo.name}" is currently edited.`);
     * ```
     */
    getReportInfo: () => CurrentReportInfo;
    /**
     * Creates a new report to be edited in Designer using the specified
**CreateReportOptions** object.
     *
     * **Example:**
     * ```javascript
     * GrapeCity.ActiveReports.WebDesigner.api.createReport({
     *     onFinish: () => {
     *         console.log('An empty RDL report is created.');
     *     },
     * });
     * ```
     *
     * @param options CreateReportOptions object
     */
    createReport: (options: CreateReportOptions) => void;
    /**
     * Opens anOpens an existing report to be edited in Designer using
the specified **OpenReportOptions** object.
     *
     * **Example:**
     * ```javascript
     * GrapeCity.ActiveReports.WebDesigner.api.openReport({
     *     reportInfo: {
     *         id: 'MyReport.rdlx',
     *         name: 'MyReport.rdlx',
     *     },
     *     onFinish: () => {
     *         console.log('An existing report "MyReport.rdlx" is
opened.');
     *     },
     * });
     * ```
     *
     * @param options OpenReportOptions object
     */
    openReport: (options: OpenReportOptions) => void;
    /**
     * Saves the report currently edited in Designer using the specified
**SaveReportOptions** object.
     *
     * **Example:**
     * ```javascript
     * GrapeCity.ActiveReports.WebDesigner.api.saveReport({
     *     reportInfo: {
     *         name: 'MyReport.rdlx',
     *     },
     *     onFinish: () => {
```

```
     *         console.log('A new report "MyReport.rdlx" is saved.');
     *     },
     * });
     * ```
     *
     * @param options SaveReportOptions object
     */
    saveReport: (options: SaveReportOptions) => void;
};
export type AboutInfo = {
    /** Application title replaces **ActiveReports Web Designer** in all
the places where it is used by default. */
    applicationTitle: string;
    /** Compact application title is used in places where there is not
enough space for a full title. */
    applicationTitleCompact: string;
    /** application version */
    applicationVersion: string;
    /** product title - can be overwritten */
    productTitle?: string;
    /** product version - can be overwritten */
    productVersion?: string;
};
export type HelpInfo = {
    /** help page title */
    title?: string;
    /** help page URL */
    link: string;
};
//
export type CurrentReportInfo = {
    /**
     * report id
     *
     * If an existing report is edited, **id** is defined.\
     * Otherwise, if a new report is edited, **id** is **null**.
     */
    id: string | null;
    /** report name */
    name: string;
};
export type CreateReportOptions = {
    /** template info - if it is specified for report creation, either
**id** or **content** needs to be defined */
    templateInfo?: {
        /** report template name */
        name: string;

        /** report template id */
        id?: string;

        /** Report template content in JSON format that can be useful in
case you would like to create a non-empty new report. */
        content?: ReportTemplate;
    };
    /** data sets infos */
    dataSets?: {
```

```
        /** data set id */
        id: string;

        /** data set name */
        name: string;
    }[];
    /** callback on starting to create a report */
    onStart?: () => void;
    /** callback on finishing to create a report */
    onFinish?: () => void;
};
export type OpenReportOptions = {
    reportInfo: {
        /** report name */
        name: string;

        /** report id */
        id?: string;

        /** report content in JSON format */
        content?: Report;
    };
    /** callback on starting to open a report */
    onStart?: () => void;
    /** callback on finishing to open a report */
    onFinish?: () => void;
};
export type SaveReportOptions = {
    /** report info */
    reportInfo: {
        /**
         * report id
         *
         * If an existing report is to be overwritten on saving, the
correct **id** should be specified explicitly.
         */
        id?: string;

        /** The correct name needs to be always specified explicitly. */
        name: string;
    };
    /** callback on starting to save a report */
    onStart?: () => void;
    /** callback on finishing to save a report */
    onFinish?: () => void;
```

```
};
```