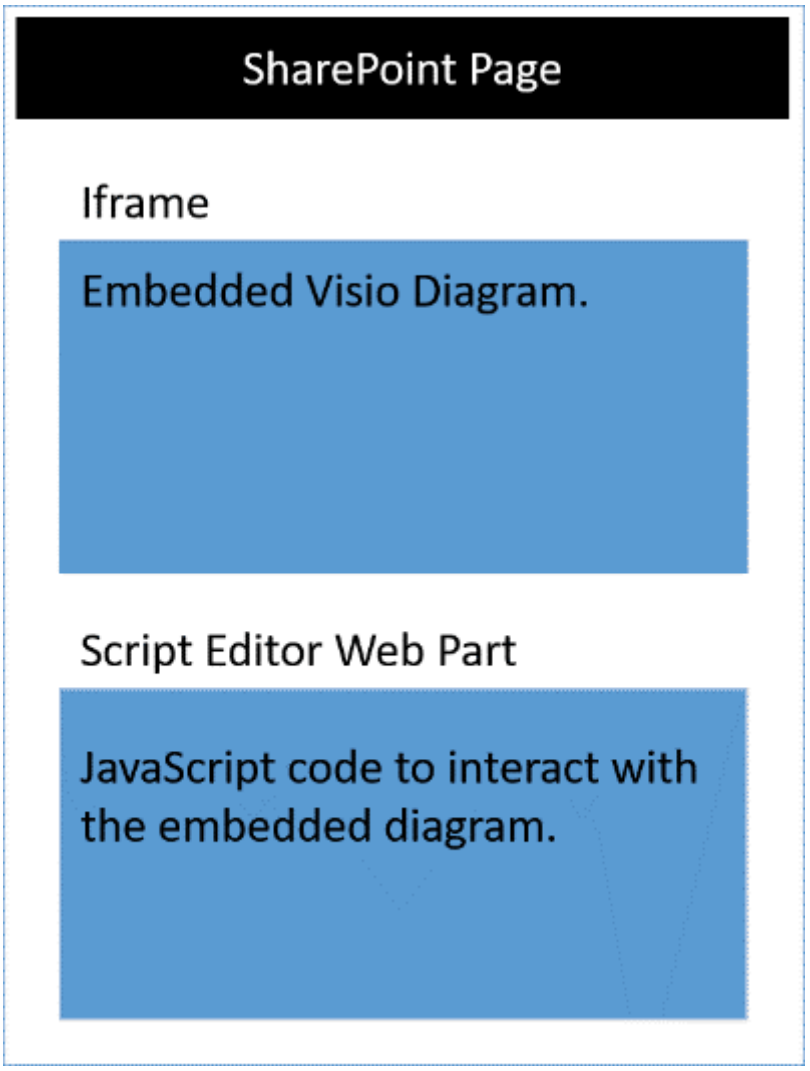
Visio JavaScript API overview

07/18/2025

You can use the Visio JavaScript APIs to embed Visio diagrams in *classic* SharePoint pages in SharePoint Online. (This extensibility feature isn't supported in on-premises SharePoint or SharePoint Framework pages.)

An embedded Visio diagram is stored in a SharePoint document library and displayed on a SharePoint page. To embed a Visio diagram, display it in an HTML `<iframe>` element. Then use Visio JavaScript APIs to programmatically work with the embedded diagram.



You can use the Visio JavaScript APIs to:

- Interact with Visio diagram elements like pages and shapes.
- Create visual markup on the Visio diagram canvas.
- Write custom handlers for mouse events within the drawing.
- Expose diagram data, such as shape text, shape data, and hyperlinks, to your solution.

This article describes how to use the Visio JavaScript APIs with Visio on the web to build solutions for SharePoint Online. It introduces key concepts that are fundamental to using the

APIs, such as `EmbeddedSession`, `RequestContext`, and JavaScript proxy objects, and the `sync()`, `Visio.run()`, and `load()` methods. The code examples show you how to apply these concepts.

EmbeddedSession

The EmbeddedSession object initializes communication between the developer frame and the Visio frame in the browser.

JavaScript

```
const session = new OfficeExtension.EmbeddedSession(url, { id: "embed-
iframe",container: document.getElementById("iframeHost") });
session.init().then(function () {
    window.console.log("Session successfully initialized");
});
```

Visio.run(session, function(context) { batch })

`Visio.run()` executes a batch script that performs actions on the Visio object model. The batch commands include definitions of local JavaScript proxy objects and `sync()` methods that synchronize the state between local and Visio objects and promise resolution. The advantage of batching requests in `Visio.run()` is that when the promise is resolved, any tracked page objects that were allocated during execution are automatically released.

The `run` function takes in session and RequestContext object and returns a promise (typically, just the result of `context.sync()`). You can run the batch operation outside of `Visio.run()`. However, in such a scenario, any page object references need to be manually tracked and managed.

RequestContext

The RequestContext object facilitates requests to the Visio application. Because the developer frame and the Visio web client run in two different iframes, the RequestContext object (context in the next example) is required to get access to Visio and related objects such as pages and shapes from the developer frame.

JavaScript

```
function hideToolbars() {
    Visio.run(session, function(context){
        const app = context.document.application;
        app.showToolbars = false;
```

```
        return context.sync().then(function () {
            window.console.log("Toolbars Hidden");
        });
    }).catch(function(error)
    {
        window.console.log("Error: " + error);
    });
};
```

Proxy objects

The Visio JavaScript objects declared and used in an embedded session are proxy objects for the real objects in a Visio document. All actions taken on proxy objects aren't realized in Visio, and the state of the Visio document isn't realized in the proxy objects until the document state has been synchronized. The document state is synchronized when `context.sync()` is run.

For example, the local JavaScript object getActivePage is declared to reference the selected page. You can use this to queue the setting of its properties and invoke methods. The actions on such objects aren't realized until the `sync()` method is run.

JavaScript

```
const activePage = context.document.getActivePage();
```

sync()

The `sync()` method synchronizes the state between JavaScript proxy objects and real objects in Visio by executing instructions queued on the context and retrieving properties of loaded Office objects for use in your code. This method returns a promise, which is resolved when synchronization is complete.

load()

The `load()` method is used to fill in the proxy objects created in the JavaScript layer. When trying to retrieve an object such as a document, a local proxy object is created first in the JavaScript layer. You can use such an object to queue the setting of its properties and invoke methods. However, for reading object properties or relations, the `load()` and `sync()` methods need to be invoked first. The load() method takes in the properties and relations that need to be loaded when the `sync()` method is called.

The following shows the syntax for the `load()` method.

JavaScript

```
object.load(string: properties); //or object.load(array: properties); //or
object.load({loadOption});
```

1. **properties** is the list of property names to be loaded, specified as comma-delimited strings or array of names. See `.load()` methods under each object for details.

2. **loadOption** specifies an object that describes the selection, expansion, top, and skip options. See object load options for details.

Example: Printing all shapes text in active page

The following example shows you how to print shape text value from an array shapes object. The `Visio.run()` function contains a batch of instructions. As part of this batch, a proxy object is created that references shapes on the active document.

All these commands are queued and run when `context.sync()` is called. The `sync()` method returns a promise that can be used to chain it with other operations.

JavaScript

```
Visio.run(session, function (context) {
    const page = context.document.getActivePage();
    const shapes = page.shapes;
    shapes.load();
    return context.sync().then(function () {
        for(let i=0; i<shapes.items.length;i++) {
            let shape = shapes.items[i];
            window.console.log("Shape Text: " + shape.text );
        }
    });
}).catch(function(error) {
    window.console.log("Error: " + error);
    if (error instanceof OfficeExtension.Error) {
        window.console.log ("Debug info: " + JSON.stringify(error.debugInfo));
    }
});
```

Error messages

Errors are returned using an error object that consists of a code and a message. The following table provides a list of possible error conditions.

Expand table

error.code	error.message
InvalidArgument	The argument is invalid or missing or has an incorrect format.
GeneralException	There was an internal error while processing the request.
NotImplemented	The requested feature isn't implemented.
UnsupportedOperation	The operation being attempted is not supported.
AccessDenied	You cannot perform the requested operation.
ItemNotFound	The requested resource doesn't exist.

Get started

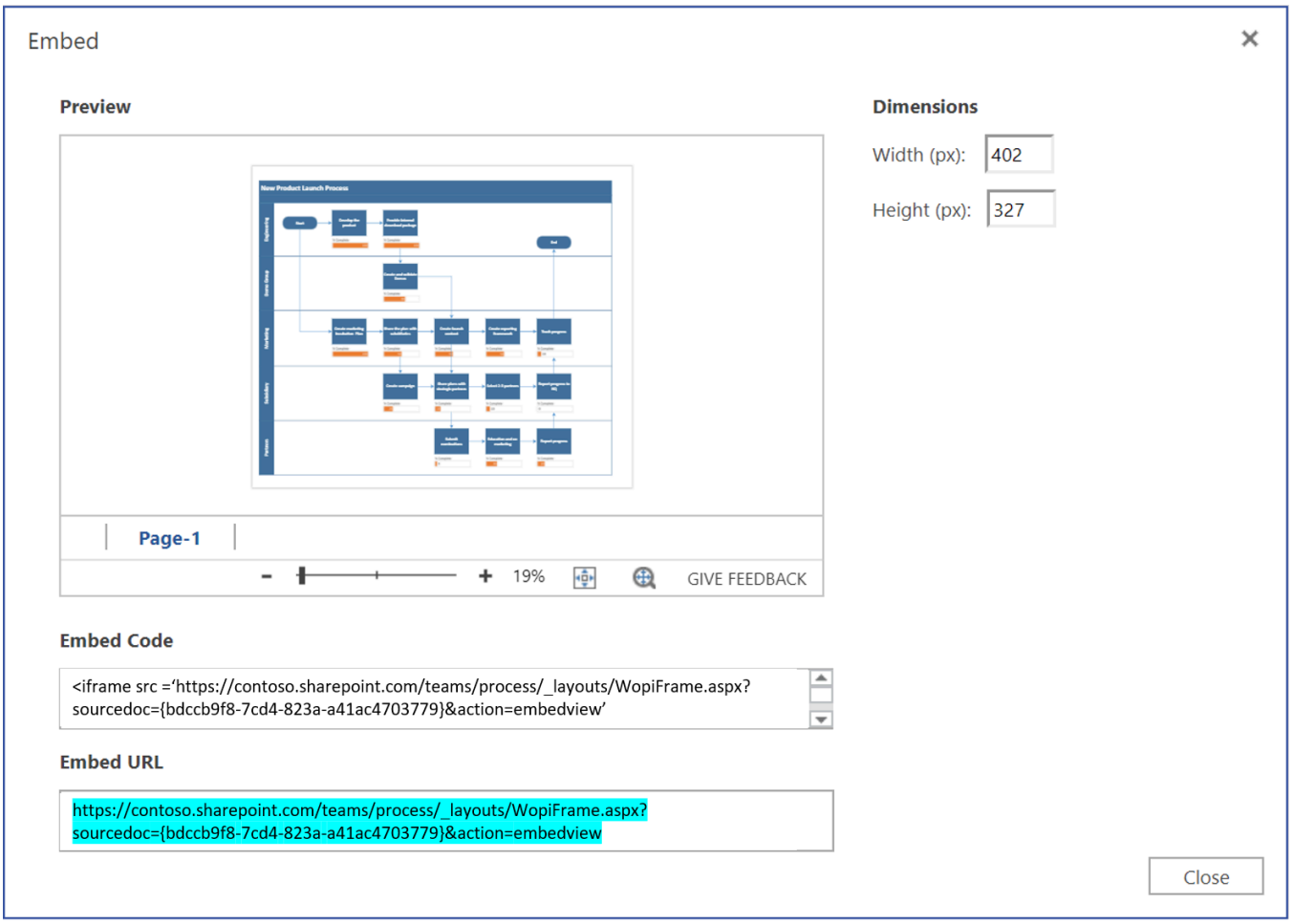
You can use the example in this section to get started. This example shows you how to programmatically display the shape text of the selected shape in a Visio diagram. To begin, create a classic page in SharePoint Online or edit an existing page. Add a script editor web part on the page and copy-paste the following code.

HTML

```
<script src='https://appsforoffice.microsoft.com/embedded/1.0/visio-web-
embedded.js' type='text/javascript'></script>

Enter Visio File Url:<br/>
<script language="javascript">
document.write("<input type='text' id='fileUrl' size='120'/>");
document.write("<input type='button' value='InitEmbeddedFrame'
onclick='initEmbeddedFrame()' />");
document.write("<br />");
document.write("<input type='button' value='SelectedShapeText'
onclick='getSelectedShapeText()' />");
document.write("<textarea id='ResultOutput' style='width:350px;height:60px'>
</textarea>");
document.write("<div id='iframeHost' />");

let session; // Global variable to store the session and pass it afterwards in
Visio.run()
let textArea;
// Loads the Visio application and Initializes communication between developer
frame and Visio online frame
function initEmbeddedFrame() {
    textArea = document.getElementById('ResultOutput');
    let url = document.getElementById('fileUrl').value;
    if (!url) {
        window.alert("File URL should not be empty");
```

```
    }
    // APIs are enabled for EmbedView action only.
    url = url.replace("action=view","action=embedview");
    url = url.replace("action=interactivepreview","action=embedview");
    url = url.replace("action=default","action=embedview");
    url = url.replace("action=edit","action=embedview");

    session = new OfficeExtension.EmbeddedSession(url, { id: "embed-
iframe",container: document.getElementById("iframeHost") });
    return session.init().then(function () {
        // Initialization is successful
        textArea.value  = "Initialization is successful";
    });
}

// Code for getting selected Shape Text using the shapes collection object
function getSelectedShapeText() {
    Visio.run(session, function (context) {
        const page = context.document.getActivePage();
        const shapes = page.shapes;
        shapes.load();
        return context.sync().then(function () {
            textArea.value = "Please select a Shape in the Diagram";
            for(let i=0; i<shapes.items.length;i++) {
                let shape = shapes.items[i];
                if ( shape.select == true) {
                    textArea.value = shape.text;
                    return;
                }
            }
        });
    }).catch(function(error) {
        textArea.value = "Error: ";
        if (error instanceof OfficeExtension.Error) {
            textArea.value += "Debug info: " + JSON.stringify(error.debugInfo);
        }
    });
}
</script>
```

After that, all you need is the URL of a Visio diagram that you want to work with. Upload the Visio diagram to SharePoint Online and open it in Visio on the web. From there, open the Embed dialog and use the Embed URL in the above example.



If you're using Visio on the web in Edit mode, open the Embed dialog by choosing **File** > **Share** > **Embed**. If you're using Visio on the web in View mode, open the Embed dialog by choosing '...' and then **Embed**.

Visio JavaScript API reference

For detailed information about the Visio JavaScript API, see the Visio JavaScript API reference documentation.

visio package

Classes

Expand table

Visio.Application	Represents the Application.
Visio.Comment	Represents the Comment.
Visio.Comment Collection	Represents the CommentCollection for a given Shape.
Visio.Document	Represents the Document class.
Visio.Document View	Represents the DocumentView class.
Visio.Hyperlink	Represents the Hyperlink.
Visio.Hyperlink Collection	Represents the Hyperlink Collection.
Visio.Page	Represents the Page class.
Visio.Page Collection	Represents a collection of Page objects that are part of the document.
Visio.PageView	Represents the PageView class.
Visio.Request Context	The RequestContext object facilitates requests to the Visio application. Since the Office add-in and the Visio application run in two different processes, the request context is required to get access to the Visio object model from the add-in.
Visio.Selection	Represents the Selection in the page.
Visio.Shape	Represents the Shape class.
Visio.Shape Collection	Represents the Shape Collection.
Visio.ShapeData Item	Represents the ShapeDataItem.
Visio.ShapeData ItemCollection	Represents the ShapeDataItemCollection for a given Shape.
Visio.ShapeView	Represents the ShapeView class.

Interfaces

Expand table

Visio.BoundingBox	Represents the BoundingBox of the shape.
Visio.DataRefreshCompleteEventArgs	Provides information about the document that raised the DataRefreshComplete event.
Visio.DocumentErrorEventArgs	Provides information about DocumentError event.
Visio.DocumentLoadCompleteEventArgs	Provides information about the success or failure of the DocumentLoadComplete event.
Visio.Highlight	Represents the highlight data added to the shape.
Visio.Interfaces.ApplicationData	An interface describing the data returned by calling `application.toJSON()`.
Visio.Interfaces.ApplicationLoadOptions	Represents the Application.
Visio.Interfaces.ApplicationUpdateData	An interface for updating data on the Application object, for use in `application.set({ ... })`.
Visio.Interfaces.CollectionLoadOptions	Provides ways to load properties of only a subset of members of a collection.
Visio.Interfaces.CommentCollectionData	An interface describing the data returned by calling `commentCollection.toJSON()`.
Visio.Interfaces.CommentCollectionLoadOptions	Represents the CommentCollection for a given Shape.
Visio.Interfaces.CommentCollectionUpdateData	An interface for updating data on the CommentCollection object, for use in `commentCollection.set({ ... })`.
Visio.Interfaces.CommentData	An interface describing the data returned by calling `comment.toJSON()`.
Visio.Interfaces.CommentLoadOptions	Represents the Comment.
Visio.Interfaces.CommentUpdateData	An interface for updating data on the Comment object, for use in `comment.set({ ... })`.
Visio.Interfaces.DocumentData	An interface describing the data returned by calling `document.toJSON()`.
Visio.Interfaces.DocumentLoadOptions	Represents the Document class.

Visio.Interfaces.DocumentUpdateData	An interface for updating data on the Document object, for use in `document.set({ ... })`.
Visio.Interfaces.DocumentViewData	An interface describing the data returned by calling `documentView.toJSON()`.
Visio.Interfaces.DocumentViewLoadOptions	Represents the DocumentView class.
Visio.Interfaces.DocumentViewUpdateData	An interface for updating data on the DocumentView object, for use in `documentView.set({ ... })`.
Visio.Interfaces.HyperlinkCollectionData	An interface describing the data returned by calling `hyperlinkCollection.toJSON()`.
Visio.Interfaces.HyperlinkCollectionLoadOptions	Represents the Hyperlink Collection.
Visio.Interfaces.HyperlinkCollectionUpdateData	An interface for updating data on the HyperlinkCollection object, for use in `hyperlinkCollection.set({ ... })`.
Visio.Interfaces.HyperlinkData	An interface describing the data returned by calling `hyperlink.toJSON()`.
Visio.Interfaces.HyperlinkLoadOptions	Represents the Hyperlink.
Visio.Interfaces.PageCollectionData	An interface describing the data returned by calling `pageCollection.toJSON()`.
Visio.Interfaces.PageCollectionLoadOptions	Represents a collection of Page objects that are part of the document.
Visio.Interfaces.PageCollectionUpdateData	An interface for updating data on the PageCollection object, for use in `pageCollection.set({ ... })`.
Visio.Interfaces.PageData	An interface describing the data returned by calling `page.toJSON()`.
Visio.Interfaces.PageLoadOptions	Represents the Page class.
Visio.Interfaces.PageUpdateData	An interface for updating data on the Page object, for use in `page.set({ ... })`.
Visio.Interfaces.PageViewData	An interface describing the data returned by calling `pageView.toJSON()`.
Visio.Interfaces.PageViewLoadOptions	Represents the PageView class.
Visio.Interfaces.PageViewUpdateData	An interface for updating data on the PageView object, for use in `pageView.set({ ... })`.
Visio.Interfaces.SelectionData	An interface describing the data returned by calling `selection.toJSON()`.

Visio.Interfaces.ShapeCollectionData	An interface describing the data returned by calling `shapeCollection.toJSON()`.
Visio.Interfaces.ShapeCollectionLoadOptions	Represents the Shape Collection.
Visio.Interfaces.ShapeCollectionUpdateData	An interface for updating data on the ShapeCollection object, for use in `shapeCollection.set({ ... })`.
Visio.Interfaces.ShapeData	An interface describing the data returned by calling `shape.toJSON()`.
Visio.Interfaces.ShapeDataItemCollectionData	An interface describing the data returned by calling `shapeDataItemCollection.toJSON()`.
Visio.Interfaces.ShapeDataItemCollectionLoadOptions	Represents the ShapeDataItemCollection for a given Shape.
Visio.Interfaces.ShapeDataItemCollectionUpdateData	An interface for updating data on the ShapeDataItemCollection object, for use in `shapeDataItemCollection.set({ ... })`.
Visio.Interfaces.ShapeDataItemData	An interface describing the data returned by calling `shapeDataItem.toJSON()`.
Visio.Interfaces.ShapeDataItemLoadOptions	Represents the ShapeDataItem.
Visio.Interfaces.ShapeLoadOptions	Represents the Shape class.
Visio.Interfaces.ShapeUpdateData	An interface for updating data on the Shape object, for use in `shape.set({ ... })`.
Visio.Interfaces.ShapeViewData	An interface describing the data returned by calling `shapeView.toJSON()`.
Visio.Interfaces.ShapeViewLoadOptions	Represents the ShapeView class.
Visio.Interfaces.ShapeViewUpdateData	An interface for updating data on the ShapeView object, for use in `shapeView.set({ ... })`.
Visio.PageLoadCompleteEventArgs	Provides information about the page that raised the PageLoadComplete event.
Visio.PageRenderCompleteEventArgs	Provides information about the page that raised the PageRenderComplete event.
Visio.Position	Represents the Position of the object in the view.
Visio.SelectionChangedEventArgs	Provides information about the shape collection that raised the SelectionChanged event.

Visio.ShapeMouseEnterEvent Args	Provides information about the shape that raised the ShapeMouseEnter event.
Visio.ShapeMouseLeaveEvent Args	Provides information about the shape that raised the ShapeMouseLeave event.
Visio.TaskPaneStateChanged EventArgs	Provides information about the TaskPaneStateChanged event.

Enums

Expand table

Visio.ColumnType	Represents the type of column values.
Visio.Connector Direction	Direction of connector in DataVisualizer diagram.
Visio.CrossFunctional FlowchartOrientation	Represents the orientation of the Cross Functional Flowchart diagram.
Visio.DataSourceType	Represents the type of source for the data connection.
Visio.DataValidation ErrorType	Represents the types of data validation error.
Visio.DataVisualizer DiagramOperationType	Type of the Data Visualizer Diagram operation
Visio.DataVisualizer DiagramResultType	Result of Data Visualizer Diagram operations.
Visio.DataVisualizer DiagramType	DiagramType for Data Visualizer diagrams.
Visio.ErrorCodes	
Visio.EventType	EventType represents the type of the events Host supports.
Visio.LayoutVariant	Represents the type of layout.
Visio.MessageType	MessageType represents the type of message when event is fired from Host.
Visio.OverlayHorizontal Alignment	Represents the Horizontal Alignment of the Overlay relative to the shape.
Visio.OverlayType	Represents the type of the overlay.
Visio.OverlayVertical Alignment	Represents the Vertical Alignment of the Overlay relative to the shape.

Visio.TaskPaneType	TaskPaneType represents the types of the First Party TaskPanes that are supported by Host through APIs. Used in case of Show TaskPane API, TaskPane State Changed, or similar events.
Visio.ToolBarType	Toolbar IDs of the app.

Functions

Expand table

Visio.run(batch)	Executes a batch script that performs actions on the Visio object model, using a new request context. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.
Visio.run(object, batch)	Executes a batch script that performs actions on the Visio object model, using the request context of a previously-created API object.
Visio.run(objects, batch)	Executes a batch script that performs actions on the Visio object model, using the request context of previously-created API objects.
Visio.run(context Object, batch)	Executes a batch script that performs actions on the Visio object model, using the RequestContext of a previously-created object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

Function Details

Visio.run(batch)

Executes a batch script that performs actions on the Visio object model, using a new request context. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

TypeScript

```
export function run<T>(batch: (context: Visio.RequestContext) => Promise<T>):
Promise<T>;
```

Parameters

batch (context: Visio.RequestContext) => Promise<T>

A function that takes in an Visio.RequestContext and returns a promise (typically, just the result of `context.sync()`). The context parameter facilitates requests to the Visio

application. Since the Office add-in and the Visio application run in two different processes, the request context is required to get access to the Visio object model from the add-in.

Returns

Promise<T>

Visio.run(object, batch)

Executes a batch script that performs actions on the Visio object model, using the request context of a previously-created API object.

TypeScript

```
export function run<T>(object: OfficeExtension.ClientObject |
OfficeExtension.EmbeddedSession, batch: (context: Visio.RequestContext) =>
Promise<T>): Promise<T>;
```

Parameters

`object` OfficeExtension.ClientObject | OfficeExtension.EmbeddedSession

A previously-created API object. The batch will use the same request context as the passed-in object, which means that any changes applied to the object will be picked up by `context.sync()`.

`batch` (context: Visio.RequestContext) => Promise<T>

A function that takes in an Visio.RequestContext and returns a promise (typically, just the result of `context.sync()`). When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

Returns

Promise<T>

Visio.run(objects, batch)

Executes a batch script that performs actions on the Visio object model, using the request context of previously-created API objects.

TypeScript

```
export function run<T>(objects: OfficeExtension.ClientObject[], batch:
(context: Visio.RequestContext) => Promise<T>): Promise<T>;
```

Parameters

`objects` OfficeExtension.ClientObject[]

An array of previously-created API objects. The array will be validated to make sure that all of the objects share the same context. The batch will use this shared request context, which means that any changes applied to these objects will be picked up by `context.sync()`.

`batch` (context: Visio.RequestContext) => Promise<T>

A function that takes in a Visio.RequestContext and returns a promise (typically, just the result of `context.sync()`). When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

Returns

Promise<T>

Visio.run(contextObject, batch)

Executes a batch script that performs actions on the Visio object model, using the RequestContext of a previously-created object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

TypeScript

```
export function run<T>(contextObject: OfficeExtension.ClientRequestContext,
batch: (context: Visio.RequestContext) => Promise<T>): Promise<T>;
```

Parameters

`contextObject` OfficeExtension.ClientRequestContext

A previously-created Visio.RequestContext. This context will get re-used by the batch function (instead of having a new context created). This means that the batch will be able to pick up changes made to existing API objects, if those objects were derived from this same context.

`batch` (context: Visio.RequestContext) => Promise<T>

A function that takes in a RequestContext and returns a promise (typically, just the result of `context.sync()`). The context parameter facilitates requests to the Visio application. Since the Office add-in and the Visio application run in two different processes, the RequestContext is required to get access to the Visio object model from the add-in.

Returns

Promise<T>

Word add-ins documentation

With Word add-ins, you can use familiar web technologies such as HTML, CSS, and JavaScript to build a solution that runs in Word across multiple platforms, including on the web, Windows, Mac, and iPad. Learn how to build, test, debug, and publish Word add-ins.

About Word add-ins

OVERVIEW

What are Word add-ins?

QUICKSTART

Build your first Word add-in

Explore APIs with Script Lab

HOW-TO GUIDE

Use the Word JavaScript API to interact with document content and metadata

Test and debug a Word add-in

Deploy and publish a Word add-in

SAMPLE

Import a Word document template with a Word add-in

Manage citations through your Word add-in

Key Office Add-ins concepts

OVERVIEW

Office Add-ins platform overview

CONCEPT

Core concepts for Office Add-ins

Design Office Add-ins

Develop Office Add-ins

Resources

REFERENCE

Ask questions

Request features

Report issues

Office Add-ins additional resources