# Infor M3 H5 Development Guide

Version 10.4.1
Published October 2025

## Publication Information

Release: Infor M3 H5 10.4.1

Publication date: < October 20, 2025 >

# Contents

# About this guide

The M3 H5 Development Guide provides conceptual and how-to information for developing, deploying, and adding scripts to MForms. It also contains information about the public APIs and script examples that the user can use as a guide in scripting.

This guide is for developers who want to create and deploy scripts for M3 H5.

## Related Documents

You can find the documents in the product documentation section of the Infor Xtreme Support portal, as described in the following section.

*M3 UI Adapter Installation Guide for LifeCycle Manager 9.x*

*M3 UI Adapter Installation Guide for LifeCycle Manager 10.x*

*M3 Core Administration Guide - Windows*

*Infor M3 H5 User and Administration Guide*

## Contacting Infor

If you have questions about Infor products, go to the Infor Support portal at support.infor.com.

If we update this document after the product release, we will post the new version on this Web site. We recommend that you check this Web site periodically for updated documentation.

If you have comments about Infor documentation, contact documentation@infor.com.

# Infor Community M3 Development

The Infor Community M3 Development is the users' go-to space for all things related to H5 scripting within the Infor environment. The community was created to provide users with information, support, and a platform to share experiences or questions!

Access the H5 Scripting Infor Community through this link: M3 Development – Infor Global Community

If the link provided does not work, please follow these steps:

1. Log in to the Infor Community website.
2. Navigate to the "Important Links" section and select 'Developer Community'.
3. In the "Other Forums" section, select the 'M3 Development'.
4. Within the Forums section, look for and access the H5 Scripting Discussion that you need.

Chapter 1

# INTRODUCTION

1

## Scripting Overview

You can enhance M3 forms in H5 by connecting one or more scripts to a form. Adding a script grants it access to the form, allowing it to modify and extend its content.

These scripts serve as a form of personalization and are deployed as JavaScript files to a server via an administration tool. They can access public classes exposed by the M3 H5 framework.

Scripts can be written in JavaScript or TypeScript, with TypeScript preferred for its support of typing, compilation, and refactoring. More details on TypeScript development are available in the next chapter.

Additionally, scripts can leverage jQuery — a lightweight, feature-rich JavaScript library that simplifies HTML traversal, event handling, animation, and Ajax with a cross-browser compatible API. However, it is important to note that support for jQuery may evolve over time as the platform and web standards develop, and future updates could impact its usage or availability.

## Resources

Following is a list of resources to develop scripts:

| For information about | URL |
|---|---|
| jQuery API Documentation | https://api.jquery.com/ |
| TypeScript | https://www.typescriptlang.org/docs/ |
| Template and Samples (KB 1909067) | https://inforsaas.service-now.com/kb?id=kb_article_view&sysparm_article=KB1909067 |
| Angular Documentation | https://angular.dev/overview |

## Creating a Script in M3 H5

Scripts can be created using a Web development IDE or an external text editor as these are client-side scripts only. The development setup and script debugging are detailed in the next chapter.

## Contents of a Script file

A script file must follow certain rules to be used by the H5 framework. If the script does not follow these rules, it will not be executed.

- The scripts' class name must match the script file name (excluding the file extension).
- The script must have a public function called Init with a specific signature.

**Important:** If there is a mismatch between the script class and the script file name, the script will not work when it is deployed to the server.

**Sample Script:**

```
class Test {
    private controller: IInstanceController;
    private log: IScriptLog;
    private args: string;

    constructor(scriptArgs: IScriptArgs) {
        this.controller = scriptArgs.controller;
        this.log = scriptArgs.log;
        this.args = scriptArgs.args;
    }

    /**
     * Script initialization function.
     */
    public static Init(args: IScriptArgs): void {
         /* Write code here */
    }
}
```

### Method and Parameters

The following method and parameters describe the content of the script file.

## Init method

The script must have a public function called `Init` with this signature:

```
    /**
     * Script initialization function.
     */
    public static Init(args: IScriptArgs): void {
         /* Write code here */
    }
```

The `scriptArgs` parameter of the `Init` method contains `JSON` object that can be used to access elements or objects from the framework. These parameters are detailed in the following sections.

## Element parameter

The element parameter is the control to which the script is connected. The argument is `null` if the script is not connected to a specific element. The element parameter can be a `TextBox`, `ComboBox`, `CheckBox`, and so on, depending on which element the script is attached to. To access the element:

```
     public static Init(args: IScriptArgs): void {
             const element = args.elem;
     }
```

## Args parameter

The args parameter contains the script arguments string. The value is `null` if no arguments were specified. To access args:

```
     constructor(scriptArgs: IScriptArgs) {
        this.args = scriptArgs.args;
      }
```

## Controller parameter

The controller is the `InstanceController` for the current program. This is used to access the content of an M3 form. To access the controller:

```
     constructor(scriptArgs: IScriptArgs) {
        this.controller = scriptArgs.controller;
      }
```

### Log parameter

The log parameter is an instance of ScriptLog and can be used for logging to the browser console. To access log:

```
    constructor(scriptArgs: IScriptArgs) {
       this.log = scriptArgs.log;
    }
```

## M3 H5 Limitations

The H5 scripts have inherent limitations primarily due to the constraints of the web platform.

Key restrictions include:

- No access to file systems or applications
- Cross domain calls are not allowed by default
- Browser restrictions for window or iframe access

## M3 H5 Script Minification

Before deploying H5 script files in a production environment, the files should be minified to reduce download size, remove code comments, and obfuscate the code. It is imperative to retain and securely store the original source files to ensure ongoing maintainability and to facilitate effective debugging post-deployment. Several open-source script minifiers are available for this purpose.

# Chapter 2
# DEVELOPING SCRIPTS

2

This chapter describes how to set up the development environment, run the Samples project, and add content to an H5 form via script written in TypeScript.

All the code examples from here on are written in TypeScript. The typing files are available for the H5 APIs, jQuery, and other frameworks.

## Development Environment

### Prerequisites

To develop and test scripts, you will need an editor for the source files, and a web server for hosting the files. Tools such as Visual Studio have a source code editor, debugger, and built-in web server, while some only have a source code editor and a support for running command line tools.

The following is a selection of the tools that can be used for developing scripts:

- Visual Studio Professional or Enterprise
    - Requires a license.
    - Minimum version is Visual Studio 2013 with update 4.
- Visual Studio Community
    - Free version of Visual Studio with limited functionality.
- Visual Studio Code + Node.js web server (Highly Recommended)
    - Visual Studio Code is a free, lightweight, and versatile code editor available on Windows, macOS, and Linux
    - When paired with a Node.js web server, it provides a powerful and flexible environment ideal for developing, testing, and deploying client-side scripts
    - This setup supports modern web development workflows and offers extensive extensions and debugging capabilities

# Using Visual Studio

## Configuring the project

1 Open the H5 Script samples solution (**Samples.sln**).

2 Right click **Samples** tree item.

3 Select **Properties** menu item.

4 Select **Web** tab.

5 Select **Start URL** radio button.

6 Set **Start URL**.

Here, the `localScript` parameter points to a single local script. The `scriptCache` parameter disables script caching for development.

a URL templates

(1) `<H5 URL>?localScript=<IIS Express URL>/<Script name>`

(2) `<H5 URL>?scriptCache=false&localScript=<IIS Express URL>/<Script name>`

b Example URLs:

(1) https://m3ce.inforcloudsuite.com/mne/ext/h5xi/?localScript=http://localhost:49482/H5SampleHelloWorld.js

(2) https://m3ce.inforcloudsuite.com/mne/ext/h5xi/?scriptCache=false&localScript=http://localhost:49482/H5SampleHelloWorld.js

7 **Save**.

## Running Samples

1 Open the H5 Script samples solution (**Samples.sln**).

2 Make sure that the project has been properly configured, as described in the previous section.

3 Modify the script name in the Start URL if needed.

4 Debug the project with **F5** or run with **Ctrl+F5**. The M3 H5 application will be opened.

## Debugging

**Note:** When running in Chrome, the same origin policy needs to be disabled.

1 Set a breakpoint in a script file.

2 To start debugging, press **F5**.

3 Log on to H5.

4 Start the M3 program where the script should be tested.

5 Select **Tools** menu item.

6 Select **Scripts…** menu item under Personalize.

7 Type in the edit box the script name and optional arguments.

8 Click **Add**.

9 Click **Save**.

10 Refresh the panel with the Refresh button in the toolbar or press **F5**.

11 The breakpoint should be hit in Visual Studio.

### Adding a new script to the solution

Alternatively, you can use the H5ScriptsProjectTemplate solution that is included in the SDK.

12 In the solution, right-click on the project tree item.

13 Click **Add** > **New Item**.

14 Select **Web** > **TypeScript** File.

15 Enter **Name**.

16 Click **Add**.

## Using VS Code and Node.js

### Opening project in VS Code

1 **Open Folder**

2 Select **H5ScriptSDK/Samples/Samples**

This is the directory that contains the `tsconfig.json` file.

### Running Samples on a Node.js server

1 To use a Node.js server, follow the instructions in the Appendix and make sure they are correctly installed before trying to run the samples.

2 Double-click InstallWebServer.cmd and wait for the installation to complete.

3 Double-click StartWebServer.cmd after the web server installation. Open this file in a text editor to modify the port number.

4 Open a browser and navigate to http://localhost:8080 to verify that the server is up.

If the default port (8080) is available, the web server will be started. If the server does not start, check that the port is available. If the port is not available, try to start the server on another port by editing StartWebServer.cmd in a text editor. If there are other issues with the installation, check that NPM and Node.js are installed according to the information in the Appendix.

5 Navigate the following URL:

```
<H5 URL>?scriptCache=false&localScript=<Node.js server URL>/<Script name>
```

- Example URL: https://m3ce.inforcloudsuite.com/mne/ext/h5xi/?localScript=http://localhost:8080/H5SampleHelloWorld.js
- To verify that the script is accessible, navigate to `<Node.js server URL>/<Script name>.`

  Example:

  http://localhost:8080/H5SampleHelloWorld.js

6 Log on to H5.

7 Start the M3 program where the script should be tested.

8 Select **Tools** menu item.

9 Select **Scripts…** menu item under Personalize.

10 Enter name of script in the Script field and optionally any arguments in the Argument field

11 Click **Add**.

12 Click **Save**.

13 Refresh the panel with the Refresh button in the toolbar or press **F5**.

## Adding a new script to the project

1 **File** > **New File**

2 Enter the file name with a .ts extension (e.g., H5Test.ts).

3 **Press Ctrl + S to save the file.** If automatic transpilation is configured, this will generate the corresponding JavaScript file (e.g., H5Test.js).

- If automatic transpilation is not set up, manually run the TypeScript compiler by executing the following command in the terminal:

  ```
  tsc H5Test.ts
  ```

  This will transpile the TypeScript file into JavaScript.

4 To manually build all configured TypeScript files, press Ctrl + Shift + B to run the build task. Debugging in browsers Edge and Chrome provide robust developer tools, which can be opened by pressing Ctrl + Shift + I.
Additionally, TypeScript can be debugged in both browsers using debugger map files.

# MForms Panel Layout

The form that shows the content of an M3 program is located between the command bar and the status bar. This is the form that can have its content modified by scripts. The layout of the contents of a form is displayed based on the computed positioning of each element passed from an M3 program.

## Adding Content to a Form

The form elements can be accessed through the `ContentElement`, which can be retrieved from the `InstanceController`. To add content to the panel, call the `ContentElement.AddElement` function.

## Element layout

The standard content is positioned on the form using the `Top`, `Left`, `Width`, and `Height` properties of the `PositionElement` class. When you add custom elements to a form, the elements should always be positioned using these properties.

```
private addButton(): void {
        const buttonElement = new ButtonElement();
        buttonElement.Name = "showUserDetails";
        buttonElement.Value = "Show User Details";
        buttonElement.Position = new PositionElement();
        buttonElement.Position.Top = 2;
        buttonElement.Position.Left = 1;
        buttonElement.Position.Width = 5;

        const contentElement = this.controller.GetContentElement();
        const button = contentElement.AddElement(buttonElement);

        button.click(() => {
            this.showDetails();
        });
    }
```

## Events

Because M3 H5 is a web-based application, script developers can utilize the default events of the browser for the elements. For button elements, the commonly used event is the `click` event, which can be used to perform additional validation and to properly handle the data to be submitted or passed.

Events must be properly managed. This includes unsubscribing from certain events after use to prevent memory leaks or multiple invocations of the same script across different instances. Additionally, data used within events must be passed correctly to avoid the use of anonymous functions, which can also lead to memory leaks.

**Example:**

```
private addButton(): void {
    const button = this.controller.GetContentElement().AddElement({
        Name: "showUserDetails",
        Value: "Show User Details",
        Position: { Top: 2, Left: 1, Width: 5 }
    });

    button.click(() => this.showDetails());
}
```

# Chapter 3
# DEPLOYING SCRIPTS

3

## Deploying a Customized Script in M3 H5

When a script has been developed and tested, it must be uploaded to M3 H5 to be available to other users.

To upload an H5 script:

1 Go to **Administration Tools**.

2 Go to **Data files** tab.

3 Select **H5 Script** as **file type** in dropdown.

4 Click **Import** and locate script file.

**Note:** For scripts developed in TypeScript, always upload the transpiled JavaScript (.js) file to the environment—not the TypeScript (.ts) file.

- **Important:** Before minifying or transpiling, securely back up the original .ts files to ensure you can maintain and debug the source code later.
- Keep the original TypeScript files in a version-controlled repository or dedicated backup location.
- This practice helps prevent loss of the readable source and facilitates future updates or troubleshooting.

Chapter 4

# ADDING SCRIPTS TO A FORM

4

## Adding a script to a form

Scripts can be added to an M3 H5 form as a personalization. A script personalization can be created by one user and then deployed to many users using global or role personalization.

The scripts dialog can be used to connect scripts to elements on a form.

In the form:

1 Select **Tools** > **Personalization** > **Scripts**

2 Select a **Target** element from the list. The selected element will be connected and passed as an argument to the script. Select the blank option if there is no target element.

3 Input the script name without the file extension. This name must exactly match the uploaded script's name.

4 Specify the script **Arguments**. This is optional.

5 Click **Add**.

6 Click **Save**.

7 **Refresh** the form for the script to take effect.

**Note:** If no other steps are taken, the script becomes active only for the user who created the personalization. Scripts can be associated with roles or assigned to run globally.

## Adding scripts as a shortcut

Scripts can also be connected to shortcuts.

In the form:

1 Open the **Shortcuts** dialog.

  a Expand **Toolbox** at the right side of the panel.

  b Click the **Shortcuts** icon.

2. Expand the **Advanced** area.
3. Specify the **Name** and **Value** of the shortcut. These are the texts that will be displayed in the toolbox.
4. Select the **Script name** (Ex. H5SampleHelloWorld) from the list.
5. Specify the script **Arguments**. This is optional.
6. Click **Add**.
7. Click **Save**.

# Chapter 5
# PUBLIC APIs

5

## API Overview

There are many public classes, methods, and properties in M3 H5 but only a number of these are allowed for use by scripts. Many classes are public for technical reasons but should only be used internally by the M3 H5 framework.

In scripts, use only public classes that are documented. This helps minimize the risk of scripts breaking when updated versions of M3 H5 are installed.

This chapter provides the methods and properties of classes that are allowed to be used from scripts. There are code examples on how to use them in the TypeScript syntax. For more information, please see the TypeScript Handbook. The following examples are not complete and are mainly used to show the syntax.

## InstanceController

Each instance of an active M3 program includes one controller object. The controller provides access to the content of a panel through the `RenderEngine` property and makes it possible to trigger the function keys and list options from script code.

### Properties

#### *Cache*: SessionCache

Cache items in the MForms session. See SessionCache.

### *ParentWindow:* `JQuery`

The current panel

### *RenderEngine*: `RenderEngine`

Holds controls in a panel. See RenderEngine.

### *Requesting*: `InstanceEvent`

This is a cancelable event that is raised when H5 is about to make a server request. It can be used to validate the panel before a request is started and cancel the event if the validation fails. See Request Event Arguments.

### *Requested*: `InstanceEvent`

This event is raised just before each call to the server and can be used to unsubscribe to events on a panel and clean up other resources. See Request Event Arguments.

### *RequestCompleted*: `InstanceEvent`

This event is raised when a server request is done. See Request Event Arguments.

### *Response*: `ResponseElement`

The response for the M3 programs.

## Functions

### ExportToExcel()

Displays the Export to Excel dialog.

**Returns:** `void`

**Example:**

```
this.controller.ExportToExcel();
```

## ExportToGoogleSheets()

Displays the Export to Google Sheets dialog.

**Returns:** `void`

**Example:**

```
this.controller.ExportToGoogleSheets();
```

## GetContent()

Gets the content panel.

**Returns:** `JQuery` – A content panel

**Example:**

```
let myContent = this.controller.GetContent();
```

## GetContentElement()

Gets the content element.

**Returns:** `ContentElement`

**Example:**

```
let myContentElement = this.controller.GetContentElement();
```

## GetElement()

Gets a named child element on the panel.

**Returns:** `JQuery` – The element

**Example:**

```
let myElement = this.controller.GetElement();
```

## GetGrid()

Gets the current datagrid.

**Returns:** A datagrid or null if there is no grid on the current panel

**Example:**

```
let myGrid = this.controller.GetGrid();
```

## GetInstanceId()

Gets the ID for the current program instance.

**Returns:** `string` - An instance id

**Example:**

```
let myIid = this.controller.GetInstanceId();
```

## GetMode()

Gets the current panel mode.

**Returns:** `string`

The panel mode on a detail panel will be one of:

"1" – Create

"2" – Change

"3" – Copy

"4" – Delete

"5" – Display

"" – The panel mode will be blank on list panels and in other cases when the detail panel modes do not apply

**Example:**

```
let myMode = this.controller.GetMode();
```

## GetPanelName()

Gets the current panel name.

**Returns:** `string`

**Example:**

```
let myPanelName = this.controller.GetPanelName();
```

## GetProgramName()

Gets the current program name.

**Returns:** `string`

**Example:**

```
let myProgramName = this.controller.GetProgramName();
```

## GetSortingOrder()

Gets the current sorting order.

**Returns:** `string`

**Example:**

```
let mySortingOrder = this.controller.GetSortingOrder();
```

## GetValue()

Gets the value of a field on the current panel.

**Parameter:**

*`name`*`: string` – Name of the element

**Returns:** `string` – A value or null

**Example:**

```
let myValue = this.controller.GetValue("elemName");
```

## GetView()

Gets the current view.

**Returns:** `string` – A view or null if no view exists

**Example:**

```
let myView = this.controller.GetView();
```

## HideBusyIndicator()

Hides busy indicator.

**Returns:** `boolean`

**Example:**

```
this.controller.HideBusyIndicator();
```

## ListOption()

Performs a list option request.

**Parameter:**

*`option`*`: string` – The option to execute, 1-99

**Returns:** `void`

**Example:**

```
this.controller.ListOption("5");
```

## OpenFieldHelp()

Opens the field help of a given field id or field help id.

**Parameters:**

*`id`*`: string` – The element id or field help id

**Returns:** `void`

**Example:**

```
this.controller.OpenFieldHelp("ITNO");
```

## PageDown()

Performs a page down request.

**Returns:** `void`

**Example:**

```
this.controller.PageDown();
```

## PressKey()

Performs a key request.

Parameter:

*`key`*`: string` – The key to press, F1-F24 or ENTER

**Returns:** `void`

**Example:**

```
this.controller.PressKey("F5");
```

## SetValue()

Sets the value of a field on the current panel.

**Parameters:**

*`name`*`: string` – The element name

*`val`*`: any` – The element value to set

**Returns:** `void`

**Example:**

```
this.controller.SetValue("elemName", "elemValue");
```

## ShowBusyIndicator()

Displays busy indicator.

**Returns:** `boolean`

**Example:**

```
this.controller.ShowBusyIndicator();
```

## ShowMessage()

Shows a message in the status bar or in a message box depending on the settings for the current user.

**Parameter:**

*`message`*`: string` – The message to show

**Returns:** `void`

**Example:**

```
this.controller.ShowMessage("This is a validation message");
```

## ShowMessageInStatusBar()

Shows a message in the status bar.

**Parameter**:

*`message`*`: string` – The message to show

**Returns:** `void`

**Example:**

```
this.controller.ShowMessageInStatusBar("This is a validation message");
```

# Request Event Arguments

These are the event arguments for the `Requesting`, `Requested` and `RequestCompleted` events. For an example on how to cancel a request, see the Script Examples chapter.

## Properties

### *RequestEventArgs*

Provides data for a request event.

```
interface RequestEventArgs {
    controller: IInstanceController;
    commandType: string;
    commandValue: string;
}
```

### *CancelRequestEventArgs*

Provides data for a cancelable request.

```
interface CancelRequestEventArgs extends RequestEventArgs {
    cancel: boolean;
}
```

## RenderEngine

The RenderEngine class generates the controls on a panel and can be used to update or add content to a panel from scripts.

### Properties

#### *Content*: ContentElement

Provides access to the content panel of an M3 panel.  See ContentElement.

## Functions

### ShowMessage()

Shows a message in the status bar or in a dialog depending on the user settings.

**Parameter**:

*`msg`*`: string` – The message to be displayed

**Returns:** `void`

**Example:**

```
this.controller.ShowMessage("The order number must be entered.");
```

## ListControl

`ListControl` is a wrapper object for the list that provides access to the actual `ListView` control and other data related to the list.

### Properties

#### *`ListView`*

Sub-class for the actual control of the list. See `ListView` API.

### Functions

#### Columns()

Provides the column name(s) of the current displayed list

**Returns:** `string[]` – An array of column name(s) from the list. Returns an empty array if no columns or list are available.

## GetColumnIndexByName()

Provides the zero-based index of a specified column name

**Parameter**:

*`colName`*`: string` – The four-character name of the column

**Returns:** `number`

**Example:**

```
let myIndex = ListControl.GetColumnIndexByName("ITNO");
```

## GetPositionFieldValue()

Gets the value of the positioning field of a specified column, which is the field box located directly on the list headers.

**Parameter**:

*`colName`*`: string` – The name of the column field

**Returns:** `string` – This will return an empty string if the column is not found, or the column does not have a position field.

**Example:**

```
let myPosFieldVal = ListControl.GetPositionFieldValue("MMITNO");
```

## Headers()

Returns the original text for each column header(s)

Column headers can be modified using the Label personalization therefore the returned value(s) using this method may be different from the actual displayed header text(s).

**Returns:** `string []` – An array containing the text for each column header(s). This will return an empty array if there is no column or if there is no list available.

**Example:**

```
let myHeaders = ListControl.Headers();
```

## GetListColumnData()

Returns the column data of the datagrid object for a given InstanceController.

**Parameter**:

*`columnName`*`: string` – The name of the column.

*`controller`*`: InstanceController` – The instance controller object.

**Returns**: `any` – An object containing the information about the column.

**Example**:

```
const controller = scriptArgs.controller;
const grid = ListControl.GetListColumnData("ITNO", controller);
```

## RenderDataGrid()

A method to allow adding a datagrid to the form panel.

**Parameter**:

*`list`*`: IList` – A JQuery object representation of the list element to be rendered on the panel.

*`columns`*: any[] – An array of objects defining the information about each column.

*`rows`*: any[] – An array of object defining each row data.

*`options`*: any – An object containing optional parameters that will help in rendering the datagrid.

**Returns:** `JQuery<HTMLElement>` – A JQuery object representation of the rendered datagrid element.

**Example:**

```
const $list: IList = $("<div>", {
    "class": "inforDataGrid",
    "id": "crs020FGrid",
    "width": "auto",
    "height": "500px"
});
$list.Position = new PositionElement();
$list.Position.Width = "97%";
$list.Position.Top = "250";
$list.Position.Left = "10";

const columns = [{
    id: "C1",
    name: "Column One Header",
    field: "rowField1"
}, {
    id: "C2",
    name: "Column Two Header",
    field: "rowField2"
}];
const rows = [{
    rowField1: 1,
    rowField2: "one"
}, {
    rowField1: 2,
    rowField2: "two"
}];
const options = {
    forceFitColumns: true,
    autoHeight: true
}

const myDatagrid = ListControl.RenderDataGrid($list, columns, rows, options);
```

# ListView

## Functions

### GetDatagrid()

Returns the active datagrid object for a given InstanceController.

**Parameter**:

*`controller`*`: InstanceController` – The instance controller object.

**Returns**: `IActiveGrid` – A representation of the datagrid object.

**Example**:

```
const controller = scriptArgs.controller;
const grid = ListControl.ListView.GetDatagrid(controller);
```

### GetValueByColumnIndex()

Collects all the selected item(s) then gets the values relative to the specified column index

**Parameter**:

*`colIdx`*`: number` - The column index from which the value will be retrieved

**Returns**: `string[]` – An array of value(s) of the provided column index of the selected item(s). This will return an empty array if there is no selected item or if there is no list available. `null` will be returned if column index is not found.

**Example**:

```
//Get values of the first column for the selected list rows
let result = ListControl.ListView.GetValueByColumnIndex(0);
```

## GetValueByColumnName()

Collects all the selected item(s) then gets the values relative to the specified column name

**Parameter**:

*colName*: `string` - The column name from which the value will be retrieved

**Returns**: `string[]` – An array of value(s) of the provided column name of the selected item(s). This will return an empty array if there is no selected item or if there is no list available. `null` will be returned if column index is not found.

**Example**:

```
//Get values of the ITNO column for the selected list rows
let result = ListControl.ListView.GetValueByColumnName("ITNO");
```

## SelectedItem()

Gets the selected rows in the list

**Returns**: `number[]` – A sorted array of zero-based row number(s) of the selected item(s). This will return an empty array if there is no selected item or if there is no list available.

Example:

```
let selected = ListControl.ListView.SelectedItem();
```

## SetValueByColumnIndex()

Sets the value of a column of an editable list corresponding to the index.

The value will be set to the first item of the list if there is no selected item. If there are selected items, the value will be set to the first selected item.

**Parameters**:

*colIdx*: `number` – The zero-based column index to which the value will be set

*value*: `string` – The value to set

**Returns**: `void`

**Example**:

```
ListControl.ListView.SetValueByColumnIndex(0, "newValue");
```

## SetValueByColumnName()

Sets the value of a column of an editable list corresponding to the name.

The value will be set to the first item of the list if there is no selected item. If there are selected items, the value will be set to the first selected item.

**Parameters**:

*`colName`*`: string` – The column name to which the value will be set

*`value`*`: string` – The value to set

**Returns**: `void`

**Example**:

```
ListControl.ListView.SetValueByColumnIndex(0, "newValue");
```

# Active Grid

`IActiveGrid` is a representation of a datagrid object.

## Properties

### *onSelectedRowsChanged*

A property holding the method that will handle the event when the selected rows are changed.

## Functions

### getColumns()

**Returns**: The columns of the active datagrid.

### setColumns()

Sets the columns of the active datagrid.

**Parameter**:

*`columns`*`: any[]` – The array of columns to be set in the datagrid.

## setColumnsFormat()

Sets column format/s by column name/s and column type/s.

**Parameter**:

*columnData: IColumnFormat[]* – The array of column format data that the user wishes to change the format

*IColumnFormat* object parameters:

*name: string* – The column full name

*valueMap?: IValueMap[] | ComboBoxItem[]* – Used for Dropdown column types

*dateFormat?: string* – Used for Date column types

*isEditable?: Boolean* – To set Input or Checkbox column types to editable *(Default: true)*

*columnType: string* – Column Types accepted strings are: "READONLY", "TEXT", "INTEGER", "INPUT", "DROPDOWN", "DATE", and "CHECKBOX" (*case sensitive).*

**Returns**: void

**Example**:

```
let fields: IColumnFormat[] = [
  { name: "LMTS", columnType: "INPUT" },
  { name: "MMECMA", columnType: "READONLY" }
];

const activeGrid = ListControl.ListView.GetDatagrid(this.controller);
activeGrid.setColumnsFormat(fields);
```

## hideColumns()

Hides column/s by column name/s.

**Parameter**:

*columnNames*: `string[]` – The array of column names the user wishes to hide

**Returns**: `void`

**Example**:

```
const activeGrid = ListControl.ListView.GetDatagrid(this.controller);
activeGrid.hideColumns(["MMITNO","MMITDS"]);
```

## getData()

Retrieves the row data from the active datagrid.

**Returns**: An array of rows, where each row represents a data entry in the active datagrid.

### setData()

Sets the data of the active datagrid.

**Parameter**:

*`data`*`: any[]` – The array of data to be set in the datagrid

### setSelectedRows()

Sets the selected rows of the active datagrid

**Parameter**:

*`rows`*`: any[]` – The array of selected rows

### getSelectedGridRows()

**Returns**: The selected row of the datagrid

## Functions for Web Components

This section demonstrates functions designed specifically for the web component version of the list/grid, enabling column formatting and interaction within a modern web interface.

### getCellElement()

Gets the HTML element of the cell in the datagrid. *(for web components only).*

**Parameter**:

*`row: number`* – The row index of the cell

*`columnId: string`* – The column id of the cell

**Returns**: `HTMLElement`

### getPosFieldElement()

Gets the HTML element of the positioning field in the datagrid. *(for web components only).*

**Parameter**:

*`posFieldId: string`* – The positioning field Id

**Returns**: `HTMLElement`

### getRowElement()

Gets the HTML element of the row in the datagrid. *(for web components only).*

**Parameter**:

*`row: number`* – The row index of the cell

**Returns**: `HTMLElement`

# ScriptLog

`ScriptLog` is used to get a log object for logging to the browser console.

## Functions

### Error()

Logs a message as an error.

**Parameters:**

*`message`*`: string` – The message to be logged

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.Error("errorMessage");
```

### Warning()

Logs a message as a warning that might warn of potential problems.

**Parameters:**

*`message`*`: string` – The message to be logged

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.Warning("warningMessage");
```

## Info()

Logs a message as an information at the log files.

**Parameters:**

*`message`*`: string` – The message to be logged

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.Info("infoMessage");
```

## Debug()

Logs debug messages that are of main use for developers.

**Parameters:**

*`message`*`: string` – The message to be logged

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.Debug("debugMessage");
```

## Trace()

Logs messages that are very detailed and are intended only for development.

**Parameters:**

*`message`*`: string` – The message to be logged

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.Trace("traceMessage");
```

## SetDefault()

The `SetDefault` method makes the log level the same as `Info()`.

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
```

```
log.SetDefault();
```

### SetDebug()

The `SetDebug` method makes the log level the same as `Debug()`.

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.SetDebug();
```

### SetTrace()

The `SetTrace` method makes the log level the same as `Trace()`.

**Returns:** `void`

**Example:**

```
let log = scriptArgs.log;
log.SetTrace();
```

## Script Utililty

The `ScriptUtil` class contains utility methods scripts can use.

### Functions

### AddEventHandler()

Attaches an event handler to an element.

You can also include a namespace for the event handler that you want to attach by adding the `(.)` character plus the namespace that you want to the *`eventType`* parameter. The namespace can be used to remove just the event handler that you attached when you use the `ScriptUtil.RemoveEventHandler` method.

**Parameters**:

*`element`*`: jQuery` – The element to which the event handler will be attached

*eventType*: `string` – The event type to which the handler will respond. Examples of event type are `"click"`, `"mousedown "`, `"mousemove"`, etc. The `eventType` parameter can also include the namespace for the event handler. Examples are `"click.myClick"` and `"mouseDown.myMousedown"`.

*callback*: `function(e: Event)` – The callback method to execute when the event occurs. The callback method receives the `Event` object as a parameter.

*paramData* *(optional)*: `any` – Holds parameters to be passed the callback method. This will become the `paramData` property of the `Event` object of the *callback* parameter.

**Returns**: `void`

**Example**:

```
let eventParam = { field1: "myField1", field2: "myField2" };
ScriptUtil.AddEventHandler(button, "click", (event) => {
     //Do something
});
ScriptUtil.AddEventHandler(myButton, "click.myClickEvent", function(event){
     //Do something
}, eventParam);
```

## DoEnterpriseSearch()

Sets value in the search input and execute the search functionality.

This method can be used to execute a search query in the list of an M3 panel.

**Parameter:**

*query*: `string` – The string to search for

*controller* *(optional)*: `InstanceController` – The instance controller; uses the active controller by default

**Returns:** `void`

**Examples:**

```
ScriptUtil.DoEnterpriseSearch("ITNO:0001");
```

## FindChild()

This will find an element in the panel.

**Parameters:**

*parent*: `jQuery` – The parent node of the element to find

*elementName*: `string` – The name of the element to find

**Returns:** `jQuery`

**Example:**

```
ScriptUtil.FindChild(parent, "myChild");
```

## GetFieldValue()

Gets the value of a specified field.

**Parameter:**

*`fieldName`*`: string` – The name of the field whose value will be retrieved

*`controller`* *(optional)*`: InstanceController` – The instance controller; uses the active controller by default

**Returns:** `string` – The value of the field, or `null` if the field cannot be found.

**Example:**

```
ScriptUtil.GetFieldValue(myTextbox.attr("id"));
```

## GetUserContext()

Gets user context data that contains login information of the user.

**Parameter:**

*`contextProp`* *(optional)*`: string` – The name of the context property to get

**Returns:** `string` – The value of the specified user context property. This will return `null` if the property is not found. If no parameter is passed, the whole user context data object will be returned.

**Example:**

```
let myCompany = ScriptUtil.GetUserContext("CurrentCompany"); //"136"
let myContext = ScriptUtil.GetUserContext();
this.log.Info(myCompany === myContext.CurrentCompany); //true
```

## Launch()

Launches a program, file, or a URL.

The default handler of the operation is used to find the program for the files or URLs.

**Parameter:**

*`task`*`: string` – A program name, file name or URL to execute

*`title`*`: string` – Title of the tab

**Returns:** `void`

**Example:**

```
let uri = https://docs.infor.com/',
    title = 'InforDocs';
ScriptUtil.Launch(uri, title);
```

## LoadScript()

Loads an external script file from the script repository.

The script file must contain a class with the same name as the file.

**Parameters:**

*`URL`*`: script` – The location of the script file to be loaded

*`callback`*`: function(data: string)` – A callback method to execute after the script has been loaded. The callback method receives the string content of the loaded script file.

**Returns:** `void`

**Example:**

```
ScriptUtil.LoadScript("scripts/SampleExternalScript.js", data => { });
```

## OpenMenu()

Displays a menu located in the menu bar. The menu name should be provided in lowercase.

**Parameter:**

*`menuName`*`: string` – The name of the menu to open

*`controller`* *(optional)*`: InstanceController` – The instance controller; uses the active controller by default

**Returns:** `void`

**Example:**

```
ScriptUtil.OpenMenu("action");
```

## RemoveEventHandler()

A utility method for removing event handlers that are attached to an element.

If you attached the event handler using `ScriptUtil.AddEventHandler` method and included a namespace for the event handler, you will be able to remove only that specific handler.

**Parameters:**

*`element`*`: jQuery` – The element from which the event handler will be removed

*`eventType`*`: string` – The event type for which the handler will be removed. Examples of event type are `click`, `mousedown`, `mousemove`, etc. The `eventType` parameter can also use a

namespace associated to the event handler by adding a (.) character plus the namespace of the handler. Examples are `click.myClick` and `mouseDown.myMousedown`. This will allow you to remove only the event type under that namespace.

**Returns:** `void`

**Example:**

```
ScriptUtil.RemoveEventHandler(buttonElement, "click");
ScriptUtil.RemoveEventHandler(buttonElement. "click.myClickEvent");
```

## SetFieldValue()

Sets the value of a specified field.

**Parameters:**

*`fieldName`*`: string` – The name of the field whose value will be set

*`value`*`: string` – The value to set. This accepts `true` and `false` for a checkbox field

*`controller`* *(optional)*`: InstanceController` – The instance controller; uses the active controller by default

**Returns:** `void`

**Example:**

```
let field = ScriptUtil.GetFieldValue(myTextbox.attr("id")).toUpperCase();
ScriptUtil.SetFieldValue(field, "myValue");
```

## UnloadScript()

A method to remove other external script that might be added by the script itself. These could be external scripts that are accessed via a URL.

**Parameters:**

*`url`*`: string` – The URL of the script that needs to be removed.

**Returns:** `void`

**Example:**

```
ScriptUtil.UnloadScript("scriptName");
```

# SessionCache

The `SessionCache` is a helper class for scripts to cache items in the MForms session. Make sure to use unique key names to avoid conflicts with other scripts when you add content to the cache.

## Functions

### Add()

Adds value to the cache.

If a value with the same key exists, it will be overwritten.

**Parameters:**

*`key`*`: string` – The key of the value to add

*`value`*`: any` – The value to add

**Returns:** `void`

**Example:**

```
SessionCache.Add("progName", "MMS001");
```

### ContainsKey()

Checks if a key exists in the session cache

**Parameters:**

*`key`*`: string` – The key to check

**Returns:** `boolean - true` if the key exists, `false` otherwise.

**Example:**

```
if(SessionCache.ContainsKey("myKey")) {
     //Do something
}
```

### Get()

Gets a value from the session cache

**Parameter:**

*`key`*`: string` – The key of the value to get

**Returns:** `any` – The cached value. This will return null if key is not found.

**Example:**

```
let myCachedProgram = SessionCache.Get("myProgram");
```

### Remove()

Removes value from the session cache

**Parameter:**

*`key`*`: string` – The key of the value to remove

**Returns:** `boolean` – `true` if the key existed, `false` otherwise.

**Example:**

```
if(SessionCache.Remove("myKey") {
     this.log.Info("myKey was removed");
}
```

## InstanceCache

The `InstanceCache` is a helper class for scripts to cache items for a specific MForms instance. Make sure to use unique key names to avoid conflicts with other scripts when adding content to the cache.

### Functions

### Add()

Adds value to the instance cache.

If a value with the same key exists, it will be overwritten.

**Parameters:**

*`controller`*`: InstanceController` – The controller for the instance

*`key`*`: string` – The key of the value to add

*`value`*`: any` – The value to add

**Returns:** `void`

**Example:**

```
InstanceCache.Add(this.controller, "myKey", true);
```

### ContainsKey()

Checks if a key exists in the instance cache.

**Parameters:**

*controller*: `InstanceController` – The controller for the instance

*key*: `string` – The key to check

**Returns:** `boolean - true` if the key exists, `false` otherwise.

**Example:**

```
const key = this.scriptName;

if (InstanceCache.ContainsKey(this.controller, key)) {
     // The key exists in cache
}
```

## Get()

Gets a value from the instance cache.

**Parameter:**

*controller*: `InstanceController` – The controller for the instance

*key*: `string` – The key of the value to get

**Returns:** `any` – The cached object. This will return null if key is not found.

**Example:**

```
let myCachedValue = InstanceCache.Get(this.controller, "myKey");
```

## Remove()

Removes a value from the instance cache.

**Parameter:**

*controller*: `InstanceController` – The controller for the instance

*key*: `string` – The key of the value to remove

**Returns:** `boolean` – `true` if the key existed, `false` otherwise.

**Example:**

```
if(InstanceCache.Remove(this.controller, "myKey") {
     this.log.Info("myKey was removed");
}
```

# ConfirmDialog

The `ConfirmDialog` class contains a method for showing the kinds of dialogs that conform to the design system of M3 H5.

## Functions

### ShowMessageDialog()

The `ShowMessageDialog` takes in a JSON object that can be used to define what type of Message Dialog to be displayed.

**Parameters:**

*options*: `Object` – A JSON object that contains the options that is available for configuring the Message Dialog to be shown.

*options.dialogType (optional)*: `string` – Determines what type of dialog to be displayed. It can be "Question", "Information", "Warning" or "Error". Default value would be "Information". "Warning" would be used if it is not a valid type.

*options.header*: `string` – The dialog header.

*options.message*: `string` – More detailed message to be displayed in the dialog.

*options.id*: `string` – Only required for Warning and Question dialogs.

*options.withCancelButton (optional)*: `boolean` – Optional property that flags if a cancel button should be displayed or not.

*options.isCancelDefault (optional)*: `boolean` – Optional property that flags if the cancel button is the default button or not.

**Returns:** `void`

**Example 1:**

```
let headerMsg = "Test Header";
let msg = "This is a Sample Message Dialog";
let options = {
     dialogType: "Information",
     header: headerMsg,
     message: msg,
     id: "testScript"
};

ConfirmDialog.ShowMessageDialog(options);
```

**Example 2:**

```
//Retrieving the user response in a Question dialog
ConfirmDialog.ShowMessageDialog({
     header: "My Question",
     message: "Are you?",
     dialogType: "Question",
     closed: (ret) => {
          //If user selects Ok, ret.ok is True and ret.cancel is False
          this.log.Info("Ok: " + ret.ok + " Cancel: " + ret.cancel);
     }
});
```

## ContentElement

The `ContentElement` object provides access to the content panel of an M3 panel.

### Properties

#### *`Children:`* `any[]`

List of panel elements, e.g, `[ButtonElement, DatePickerElement, TextBoxElement, RadioGroupElement,..]`

#### *`ContentPanel:`* `JQuery`

The panel that contains the elements inside an M3 panel

#### *`Host:`* `JQuery`

The panel instance parent of an M3 panel.

### Functions

#### Add()

Adds an element to the content panel.

The exact width and position of the element in pixels must be specified to properly add an element using this method.

**Parameters:**

*`element`*`: any` – The HTML element to be added

**Returns:** `void`

**Example:**

```
let $list = $("<div>",{"class":"inforDataGrid"});
let contentElement = this.controller.GetContentElement();

contentElement.Add($list);
```

## AddElement()

Creates an HTML element based on the provided element data object and adds it to the content panel.

The element data can be an instance of the following: `ButtonElement`, `CheckBoxElement`, `ComboBoxElement`, `DatePickerElement`, `TextBoxElement`, `RadioGroupElement`, etc. This is an alternative to `ContentElement.Add()` method.

*`DatePickerElement` and `RadioGroupElement`* are also available.

**Parameters:**

*`elementData`*`: any` – The data object regarding the element to be created

**Returns:** `any –` An HTML element. The created and added element, `null` if no element is created.

**Example:**

```
let buttonElement = new ButtonElement();
let contentElement = this.controller.GetContentElement();

contentElement.AddElement(buttonElement);
```

## CreateElement()

Creates an element based on the provided data.

The element data can be an instance of the following: `ButtonElement`, `CheckBoxElement`, `ComboBoxElement`, `DatePickerElement`, `TextBoxElement`, `RadioGroupElement`, etc.

*`DatePickerElement` and `RadioGroupElement`* are also available.

**Parameters:**

*`elementData`*`: any` – The data object regarding the element to be created

**Returns:** `JQuery –` The element based on the provided Object data, `null` if no element is created.

**Example:**

```
let buttonElement = new ButtonElement();
let contentElement = this.controller.GetContentElement();

contentElement.CreateElement(buttonElement);
```

## GetContentBody()

Retrieves the `contentBody` element of the panel.

**Returns:** `JQuery –` An HTML element; `null` if not found.

**Example:**

```
let contentBody = this.controller.GetContentBody();
```

## GetElement()

Gets an element in an M3 panel.

**Parameters:**

*`elementName`*`: string` – The target element's name

**Returns:** `JQuery –` An HTML element. The element corresponding to the provided element name, `null` if no element is found.

**Example:**

```
let contentElement = this.controller.GetContentElement();
let $field = contentElement.GetElement("ITNO");
```

## GetPrevContainer()

Finds the immediate element container to the left of a given element container of a field.

Element containers are those HTML elements with `elementContainer` class.

**Parameters:**

*`elementContainer`*`: JQuery` – The element container of a field, the one with "elementContainer" class.

**Returns:** `JQuery –` An HTML element. The immediate element container to the left of the given element container.

**Example:**

```
let $host = this.controller.ParentWindow;
let contentElement = this.controller.GetContentElement();
let $firstLabel = $host.find(".someLabelClass").eq(0);
let $elem =
contentElement.GetPrevContainer($firstLabel.closest(".elementContainer"));
```

### OnUnload()

Provides a way to add callback method to execute when the content panel unloads its content.

**Parameters:**

*`callback`*`: function()` – The function to execute when the content panel unloads.

**Returns:** `void`

**Example:**

```
let content = this.controller.GetContentElement();
content.OnUnload(function(){
     //Code here
});
```

### RemoveScriptComponents()

Provides a way to remove all added components by the script.

**Returns:** `void`

**Example:**

```
let content = this.controller.GetContentElement();
content.RemoveScriptComponents();
```

### Unload()

Triggers the callback method(s) added using `ContentElement.OnUnload().`

This method does not remove the contents added to the content panel.

**Returns:** `void`

**Example:**

```
let content = this.controller.GetContentElement();
content.Unload();
```

## H5 Control Utility

This is a helper class to access and create control not possible to be created using ContentElement.

### Properties

*H5Dialog*: `any`

## H5Dialog

This is a helper class to allow creation modal dialog control that is not possible to be created using ContentElement.

### Functions

#### CreateDialogElement()

Initializer to create a custom dialog element. See H5SampleCustomDialog.ts.

**Parameters:**

*`content`*`: HTMLElement` – The HTML Element to be displayed as content for the dialog.

*`dialogOptions`*`: any` – An object containing more information and parameters about the dialog to be created.

**Returns:** `void`

## MFormsAutomation

This is a helper class to create an automation XML and its equivalent URI, which can be used to launch an M3 program in H5. The MForms Automation chapter discusses the feature in detail.

### Functions

#### addStep()

Adds a step to the automation sequence.

**Parameters:**

*action*: `string` – The command for the automation step. The available values are: Run, Key, ListOption, Set. Use the `ActionType` enum to specify the value.

*parameter*: `string` – The command value.

*expected*: `string (optional)` – The expected value.

**Returns:** `void`

**Example:**

```
const auto = new MFormsAutomation();
auto.addStep(ActionType.Run, "MNS150");
auto.addStep(ActionType.Key, "ENTER");
```

## addField()

Adds a field to the current step.

**Parameters:**

*name*: `string` – The name of the field.

*value*: `string` – The value for the field.

**Returns:** `void`

**Example:**

```
auto.addField("W1USID", ScriptUtil.GetUserContext("USID"));
```

## setFocus()

Sets the focus on the element.

**Parameters:**

*name*: `string` – The name of the field.

**Returns:** `void`

**Example:**

```
auto.setFocus("WWQTTP");
```

## toEncodedURI()

Converts the automation to a URI string.

**Returns:** `string`

**Example:**

```
auto.addStep(ActionType.Run, "MNS150");
```

```
const uri = auto.toEncodedUri();
```

# MIService

This is a helper class for calling M3 MI programs and reading the response. See the MIService chapter for more information.

## Functions

### createMiRequest()

Creates well-formed MI Request data.

**Parameter**:

*program*: `string` – The name of the program to be requested. This is detailed in the MIService chapter.

*transaction*: `string` – The type of transaction. This is detailed in the MIService chapter.

*record*: `any` – The input data to the transaction in JSON format. This is detailed in the MIService chapter.

*maxReturnedRecords*: `number` – (Optional) Indicates the maximum number of records to return. This is detailed in the MIService chapter.

**Returns:** `MIRequest` – The MI request data that can be used in executing the request.

**Example:**

```
const myRequest = MIService.createMiRequest(‘MNS150MI’, ‘GetUserData’, {
USID: "myID" }, 33);

MIService.executeRequest(myRequest).then((response: IMIResponse)=>{
     //Read results here
}).catch((response: IMIResponse)=>{
     //Handle errors here
});
```

### executeRequest() and executeRequestV2()

Performs an MI transaction using the request object.

**Parameter**:

*request*: `MIRequest` – Contains input and information about the request. It contains the parameters for program, transaction, record, or output fields. This is detailed in the MIService chapter.

**Returns:** `Promise –` A promise that will resolve an `MIResponse`. If the promise is rejected, the response will contain the error information.

**Note:** The `executeRequestV2()` function uses version 2, offering a more robust and reliable implementation, especially with enhanced transaction parameter support. The `executeRequest()` function will now be using version 2 endpoint starting October 2025.

By default, the `MIResponse` of the version 2 API does not include metadata. To receive metadata, you must explicitly add the `includeMetadata` parameter to your request. For more details, please refer to the M3 Foundation API guide.

**Example:**

```
const myRequest = new MIRequest();

Version 1
MIService.executeRequest(myRequest).then((response: IMIResponse)=>{
     //Read results here
}).catch((response: IMIResponse)=>{
     //Handle errors here
});

Version 2
MIService.executeRequestV2(myRequest).then((response: IMIResponse)=>{
     //Read results here
}).catch((response: IMIResponse)=>{
     //Handle errors here
});
```

## execute() and executeV2()

Performs an MI transaction.

**Parameter**:

*program*: `string` – The program involved in the request.

*transaction*: `string` – The method to be used.

*record*: `any (optional)` – An object that contains records of the data involved.

*outputfields*: `string array (optional)` – Corresponding output fields.

*timeout*: `number (optional)` – How long would it take for the timeout.

**Returns:** `Promise –` A promise that will resolve an `MIResponse`. If the promise is rejected, the response will contain the error information.

**Note**: The `executeV2()` function uses version 2, offering a more robust and reliable implementation, especially with enhanced transaction parameter support. The `executeRequest()` function will now be using version 2 endpoint starting October 2025.

By default, the `MIResponse` of the version 2 API does not include metadata. To receive metadata, you must explicitly add the `includeMetadata` parameter to your request. For more details, please refer to the M3 Foundation API guide.

**Example:**

```
const program = "MNS150MI";
const transaction = "GetUserData";
const record = { USID: this.usid };
const outputFields = ["USID", "CONO", "DIVI", "DTFM"];

Version 1
MIService.execute(program, transaction, record, outputfields).then((response:
IMIResponse)=>{
     //Read results here
}).catch((response: IMIResponse)=>{
     //Handle errors here
});

Version 2
MIService.executeV2(program, transaction, record,
outputfields).then((response: IMIResponse)=>{
     //Read results here
}).catch((response: IMIResponse)=>{
     //Handle errors here
});
```

## IonApiService

This is a helper class for calling ION APIs. See the API Gateway chapter for more information.

### Functions

#### execute()

Performs an ION API request.

**Parameter**:

*request*: `IonApiRequest` – Contains input and information about the request. It contains the parameters for `program`, `transaction`, `record` or `output fields`. This is detailed in the ION API chapter.

**Returns:** `Promise` – A promise that will resolve an `IonApiResponse`.

**Example:**

**Method: "GET"**

*url:* Contains the whole URL of the API transaction.

*record:* Contains the key value pair parameters of the transaction, this will be processed and will be converted to become the query string parameters of the URL.

```
const request: IonApiRequest = {
     url: "/M3/m3api-rest/execute/MNS150MI/GetUserData/",
     method: "GET",
     record: {
          USID: ScriptUtil.GetUserContext("USID")
     }
}
IonApiService.Current.execute(request).then((response: IonApiResponse) => {
     //Read response here
}).catch((response: IonApiResponse) => {
     //Handle errors here
});
```

**Method: "POST"**

*url*: Contains the whole url of the API transaction with the query string parameter.

Example of a query string parameter:

`?logicalId=lid%3A%2F%2Finfor.m3.m3`

*record*: Contains the JSON body of the transaction.

```
const request: IonApiRequest = {
     url: "
/IONSERVICES/process/application/v1/workflow/start?logicalId=lid%3A%2F%2Finfo
r.m3.m3",
     method: "POST",
     record: {
            "workflowName": "string",
            "instanceName": "string",
            "inputVariables": [
                 {
                      "name": "string",
                      "dataType": "STRING",
                      "value": "string"
                 }
            ],
            "inputStructures": [
                 {
                      "name": "string",
                      "fields": [
                      {
                               "name": "string",
                               "dataType": "STRING",
                               "value": "string"
                      }
                      ],
                 "subStructures": [
                      null
                      ]
                 }
                 ]
        }
IonApiService.Current.execute(request).then((response: IonApiResponse) => {
     //Read response here
}).catch((response: IonApiResponse) => {
     //Handle errors here
});
```

## getBaseUrl()

Returns the ION API base URL.

**Returns:** `string`

Example:

```
let baseUrl = IonApiService.Current.getBaseUrl();
```

# Chapter 6
# MFORMS AUTOMATION

6

## What is MForms Automation?

MForms automation makes it possible to start M3 programs and perform single automated steps on M3 UI Adapter (MUA) server before returning the result and control to the user. The steps in the automation sequence can set values, press keys, execute lists options and set focus, etc. Automation data can be sent to the MNE server as a small XML document that contains the definition of the automation Automations can be started from both Infor Smart Office and Workplace.

The automation functionality has some limitations and is not a complete replacement for LWS/BCI scripts. It can however be a more lightweight solution for many scenarios.

One important limitation is that the automation will only work if the program starts on the panel it was defined for (A/B etc). If the user has set a different start panel for a program the automation will not work, it will simply stop at the start panel.

## Automation Sequences

An automation sequence consists of the exact same actions that a user would do when running an M3 BE program. The user can basically enter data on a panel and make requests to the server by using enter, function keys and list options. The automation sequence will contain one step for each request, and each step can set values on the panel.

Note that in some cases it might be necessary to first set values for sorting order and panel version and pressing enter before assuming that specific fields are available.

A typical automation could be to start a program, select an item and open an E-panel in change mode. The user would take the following steps to achieve that:

- Start the program (for example MMS001)
- Select sorting order. Changing sorting order in the UI will automatically trigger the ENTER key.
- Enter an item number in the first position field, change the panel sequence and press ENTER.
- Select the first row in the list and choose the change list option.

  The E-panel is displayed.

These user actions must be mapped to steps in the automation sequence. An automation sequence can contain 1-n steps.

## Automation Steps

Each automation step must have a command, a value and 0-n fields. The available commands and values are listed in the table below.

| Command | Values | Description |
|---|---|---|
| RUN | M3 BE program name | Starts an M3 BE program. The short program name is used as value, for example MMS001.<br>The RUN command cannot set field values, fields will be ignored if they are added to the RUN command. |
| KEY | ENTER, F1-F24 | Press a key. The next button is the same as ENTER and the back button is the same as F12. |
| LSTOPT | 1-99, -1 | Performs a list option. The default is to set the option on the first visible list row, unless selected rows are specified using the SELROWS parameter. Note that -1 must be used for option create, 1 is used for option select. |
| AUTOSET | N/A | This command can set field values but do not make a request to the server. The command can be used to enter data on the last panel. |

## Automation Fields

Fields are used to enter data on a panel and for sending other special data such as focus. A field has a name and a value. You can locate the name of a field on an H5 panel in the field help window. To open the field help window, set the focus on a control in the panel and press F1.

**Example:**

Field name = `WWITNO` Field value = `ITEM001`

- Focus
    - It can be set on a specific field by using the special field name FCS.

  - Example: Field name = `FCS` Field value = `WWITNO`
- Selected List Rows
  - When using the list option command, the default is to select the first row in the list. It is possible to select other list rows using the special field name `SELROWS`. The value is a comma-separated list of row names. The name for a row is on the format R<1-based list index>.
  - Example: Field name = `SELROWS` Field value = `R1,R2,R3`

## Automation XML

An automation sequence is described in the form of a small XML document. The XML document can be generated in runtime or created using XML templates.

**Examples:**

- Empty automation XML

```
<?xml version="1.0" encoding="utf-8"?>
<sequence>
     <step command="" value="" />
     <step command="" value="">
          <field name=""></field>
     </step>
     <step command="" value="">
          <field name=""></field>
          <field name=""></field>
          <field name=""></field>
     </step>
</sequence>
```

- Automation XML with values

```
<?xml version="1.0" encoding="utf-8"?>
<sequence>
     <step command="RUN" value="MMS001" />
     <step command="KEY" value="ENTER">
          <field name="WWQTTP">1</field>
     </step>
     <step command="KEY" value="ENTER">
          <field name="W1ITNO">TESTITEM</field>
          <field name="WWPSEQ">E</field>
     </step>
     <step command="LSTOPT" value="2" />
</sequence>
```

## Starting Automations in M3 H5

Automations can be started in M3 H5 using a URI with the host _automation and the automation XML in the data parameter. The automation XML value should be URL encoded.

**URI Format**

```
mforms://_automation/?data=<AUTOMATION XML>
```

**Example:**

```
mforms://_automation?data=%3c%3fxml+version%3d%221.0%22+encoding%3d%22utf-
8%22%3f%3e%3csequence%3e%3cstep+command%3d%22RUN%22+value%3d%22MMS001%22+%2f%
3e%3c%2fsequence%3e
```

## Helper Class

There is a helper class that can be used to generate automation XML from code. From the generated automation XML, it creates a URI to launch an M3 program in H5. See the API chapter and an example of it for reference.

# Chapter 7
# MIService

7

The M3 framework has support for calling M3 MI programs using the REST endpoint on the BE server. The `MIService` is a helper class that can be used to call transactions to M3 MI programs. It builds the request URL from the set of request parameters, executes the request, and parses the MI response values.

Use `MIService` instance to avoid creating objects every time. See the API chapter for the available functions, and the Sample scripts chapter for an example.

## MIRequest

The `executeRequest()` and `executeRequestV2()` accepts an `MIRequest` object with the following properties:

| Properties | Values | Description |
|---|---|---|
| program | M3 BE program name | The BE program to be requested from.<br>Example: `MNS150MI` |
| transaction | Transaction name | Example: `GetUserData` |
| outputFields | String array | The names of the output fields to return from a transaction.<br><br>Use this property to limit the amount of data to transfer from the server. Only specify the names of the fields that will be used since that will approve the performance.<br>Example: `["CONO", "DIVI"]` |
| record | JSON | The input data to the transaction. This property is not required for transactions without mandatory input fields.<br>Example: `{ USID: "myID" }` |

| Properties | Values | Description |
|---|---|---|
| includeMetadata | true/false | Indicates if metadata should be included as part of the reply. Default is false. |
| typedOutput | true/false | Indicates if output should be converted to numbers and dates according to the meta data definition for the MI transaction. This implicitly turns on `includeMetadata` in the options to load the meta data information. Default is false. |
| maxReturnedRecords | number | Indicates the maximum number of records to return. Default is 33. |
| timeout | number | Indicates the timeout value for the HTTP request in milliseconds. Default is 55000. |

**Example:**

```
const myRequest = new MIRequest();
myRequest.program = "MNS150MI";
myRequest.transaction = "GetUserData";
myRequest.outputFields = ["CONO", "DIVI", "DTFM"];
myRequest.record = { USID: "USERFOO" };
myRequest.includeMetadata = true;
myRequest.typedOutput = true;
myRequest.maxReturnedRecords = 10;
myRequest.timeout = 60000;

MIService.Current.executeRequest(myRequest).then(
    (response: IMIResponse) => {
        // Success
    },
    (response: IMIResponse) => {
        // Error
    }
);
```

`MIService` builds this request into the following URL:

```
execute/MNS150MI/GetUserData;metadata=true;maxrecs=10;excludempty=false;co
no=760;divi=AAA;returncols=CONO,DIVI,DTFM&USID=USERFOO
```

# Chapter 8
# DRILLBACK

8

This chapter explains how to use standard Portal drillback links to seamlessly launch M3 bookmarks within the H5 interface, enabling quick navigation and improved workflow efficiency.

## Invoking a Drillback

Drillbacks can be invoked by calling the Infor Portal JavaScript API to fire the drillback message. A script is provided in the examples chapter.

### infor.companyon.client.sendPrepareDrillbackMessage()

**Returns:** `void`

**Example:**

```
let myDrillback = "?LogicalId=lid://infor.m3.1&AccountingEntity=136_AAA…";
infor.companyon.client.sendPrepareDrillbackMessage(myDrillback);
```

## Drillback Parameters

| Name | Description |
|---|---|
| LogicalId | Used to determine which ERP to forward the drillback request to<br>Must start with `lid://` |
| AccountingEntity | If blank or missing, CONO and DIVI must be part of ID1-ID7 |
| ID1-ID7 | Used to provide bookmark key values |
| Title | Title of the tab |

| Name | Description |
|---|---|
| ViewId | Used to provide metadata about an M3 bookmark<br>Key-value-pairs separated by semi colon (;) and the key and value separated by equals (=)<br>Keys are case-insensitive |

| Key | Description |
|---|---|
| Program | |
| TableName | |
| Option | |
| Panel | |
| PanelSequence | Optional |
| KeyNames | The bookmark key names |
| Keys | Consists of bookmark key names and value keys separated by comma (,)<br>Has priority over KeyNames |
| CombinedElementSeparator | Optional<br>The separator character for the AccountingEntity parameter<br>The default value is underscore (_) |
| CombinedElementsAccountingEntity | Optional<br>A comma separated list of key names for the AccountingEntity parameter<br>The default value is CONO,DIVI |

Examples:

- Using title

```
?LogicalId=lid://infor.m3.1&AccountingEntity=136_AAA&ViewId=Program=CRS610
;TableName=OCUSMA;Option=5;Panel=E;KeyNames=OKCONO,OKCUNO&ID1=MANGO&title=
CustomTitle
```

- Using KeyNames
    - Company and division will be automatically retrieved from the `AccountingEntity` parameter
    - Additional bookmark keys are retrieved from ID1-ID7

```
?LogicalId=lid://infor.m3.1&AccountingEntity=136_AAA&ViewId=Program=CRS610
;TableName=OCUSMA;Option=5;Panel=E;KeyNames=OKCONO,OKCUNO&ID1=MANGO
```

- Using Keys
    - The value key can be CONO, DIVI or ID1-ID7

  - CONO is retrieved from the `AccountingEntity`

    ```
    ?LogicalId=lid://infor.m3.1&AccountingEntity=136_AAA&ViewId=Progr
    am=CRS610;TableName=OCUSMA;Option=5;Panel=E;Keys=OKCONO,CONO,OKCU
    NO,ID1&ID1=MANGO
    ```

  - CONO is specified using the ID1-parameter

    ```
    ?LogicalId=lid://infor.m3.1&AccountingEntity=136_AAA&ViewId=Progr
    am=CRS610;TableName=OCUSMA;Option=5;Panel=E;Keys=OKCONO,ID1,OKCUN
    O,ID2&ID1=136&ID2=MANGO
    ```

- Using `CombinedElementsAccountingEntity`

  ```
  ?LogicalId=lid://infor.m3.1&AccountingEntity=136_AAA&ViewId=CombinedElemen
  tSeparator=_;CombinedElementsAccountingEntity=CONO,DIVI;Program=CRS610;Tab
  leName=OCUSMA;Option=5;Panel=E;Keys=OKCONO,ID1,OKCUNO,ID2&ID1=136&ID2=MANG
  O
  ```

# Chapter 9
# API Gateway

9

The API Gateway uses OAuth as authentication mechanism. OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The client then uses the access token to access the protected resources hosted by the resource server. The OAuth access token must be set as a request header. This means that all access to a URL must be done in code to manually update the header before making the request.

## Prerequisites

1 MUA version 10.4, due to Grid server restrictions for OAuth resource.

2 Application API deployed in API Gateway server in that environment.

3 Access to Portal and H5 in that environment.

## IonApiService

The `IonApiService` is a helper class for retrieving the OAuth token and consuming ION APIs from H5 scripts. Use the `IonApiService.Current` instance to avoid creating objects every time. See the API chapter for the available functions.

### OAuth access token

The `IonApiService` handles the retrieval of the token and updates the request authorization header before executing the ION API requests. When the token expires and an API call returns a 401 Unauthorized error, a new token is requested automatically, and the API call is retried once with this refreshed token.

## IonApiRequest

The `execute()` function accepts and `IonApiRequest` object with the following properties:

| **Properties** | **Values** | **Description** |
|---|---|---|
| url | Request URL | If relative URL, ION API base URL will be prepended<br>If absolute URL, it will be executed as it is |
| method | | HTTP method<br>Default: GET |
| responseType *(optional)* | Depends on API; usually JSON or XML | Default: JSON |
| record *(optional)* | JSON | The input data. This property is not required for APIs without mandatory input fields.<br>On execution, values will be parsed and appended to request URL |
| data *(optional)* | Depending on API | Data to be sent |
| cache *(optional)* | true/false | Default: false |
| headers*(optional)* | Array of key-value pairs | Authorization headers will always be added first, followed by these headers |

## IonApiResponse

The `execute()` function resolves an `IonApiResponse` object with the following properties:

| **Properties** | **Values** | **Description** |
|---|---|---|
| data | Response data | Depending on API |
| status *(optional)* | | |
| statusText | | |
| message *(optional)* | | |

# Development Environment

The `H5SampleIonApiService` script shows how to connect to the Portal and M3 APIs. **If the script is uploaded through Admin Tools, skip the following steps and it should run as intended provided that the ION API prerequisites are satisfied**. The following steps are a workaround when working in a development environment with no access to an authorization server, like `localhost`.

## Acquire an OAuth token string

**1** Log on to Portal

**2** Locate the server that H5 is running on within Portal. You will need to check the server and port for H5 and apply it to the template below:

- o Open the browser developer tools (how depends on browser)
- o Use the element inspector and click the top bar in H5. Look for an `iframe` element with a context root called `/mne`

**3** Open a new tab in the same browser and navigate to the Grid SAML Session Provider OAuth resource.

**Template:** `https://{h5server}:{h5port}/grid/rest/security/sessions/oauth`

**4** Copy the OAuth token string from the browser window.

## Set the token as a script argument

Set the copied token as an argument when adding the script in the Tools > Personalize > Scripts dialog. When this OAuth token times out, you must acquire a new token and update the script argument.

# Chapter 10
# BEST PRACTICES

10

This chapter provides suggestions to improve the quality and maintainability of your scripts.

## Limitations and Recommendations

Although there are many accessible functions, it is recommended to use those that are listed in the public API. This ensures that the scripts will not break when H5 is updated.

## Some examples of commonly used properties and functions:

- `InstanceController.ParentWindow`

  Use this property to get the current panel, instead of using CSS class selectors, i.e., `$(".lawsonHost:visible")` and `$(".visible-tab-host")`.

- `ContentElement.AddElement()`

  Consider using this function to add elements and easily align them in the panel row, as opposed to `ContentElement.Add()` which requires the exact position in pixels and `ControlFactory` which is not part of the public API.

- `MIService`

  Use this utility for M3 API requests, instead of the now deprecated `ScriptUtil.ApiRequest()`.

## Proper Logging Utility

Using the log object from the script arguments provides greater control over logging by allowing you to leverage built-in log levels—unlike console.log(), which cannot be turned off.

## TypeScript

TypeScript allows writing safer code. Although optional, **it is best to include the data type**, avoiding `any` as much as possible, when declaring a variable to take advantage of its type-checking. It also allows usage of some ES6 features by transpiling them to ES5-compliant code. It is recommended to utilize these features such as:

1. `const` and `let`

   These allow definition of block-scoped variables, as opposed to the function-scoped `var` declarations. Use these instead of `var` whenever possible. Use `const` to make a variable immutable.

2. `for…of` and `for…in`

   These constructs eliminate potential index bugs in the for loop. Note that `for…of` iterates over the elements of an array, while `for…in` iterates over the keys of an object.

3. Arrow functions (=>)

   The fat arrow captures `this` from the surrounding context as opposed to an ordinary function that tends to lose the meaning of `this` when it is passed around. It is recommended to use this in callback functions.

## Be wary of using ES6/ES2016 Features

As of this writing, some browsers, like Internet Explorer, do not fully support ES6/ES2016 features. Verify that the functions and properties you use are supported in the browsers that you target your scripts to run in. Note that the TypeScript compiler can convert some, not all, code to be ES5-compliant.

## Use relative URLs when calling M3 URLs sharing the same base URL with M3 H5

When using `ScriptUtil.Launch()` or adding personalization shortcuts, use relative URLs when calling H5 SDKs or other M3 URLs sharing the same base URL with M3H5. This is to prevent incorrect URLs when migrating from one environment to another.

# Chapter 11
# SCRIPT EXAMPLES

11

This chapter contains examples of scripts written in TypeScript. The scripts are complete and can be copied from this document to a new TypeScript file. The transpiled JavaScript file can then be deployed to M3 H5.

## Starting Script

### H5SampleScriptTemplate.ts

This is a starter template for H5 script development that provides the essential class structure with logging capabilities and a standardized initialization pattern. Developers use this as a foundation to build their custom scripts by adding their specific logic to the run method.

```
class H5SampleScriptTemplate {
    private controller: IInstanceController;
    private log: IScriptLog;
    private args: string;

    constructor(scriptArgs: IScriptArgs) {
        this.log = scriptArgs.log;
    }

    /**
     * Script initialization function.
     */
    public static Init(args: IScriptArgs): void {
        new H5SampleScriptTemplate(args).run();
    }

    private run(): void {
        this.log.Info("H5SampleScriptTemplate");
        /* Write code here */
    }
}
```

## Sample Scripts

### H5SampleAddElements.ts

This script shows how to add `label, textbox, combobox, checkbox, button, date picker, and radio group` elements to a panel. On button click, it toggles the `checkbox` and prints the `textbox` and `combobox` values on the console.

NOTE: There is a current limitation in creating a vertical radio group.

### H5SampleCancelRequest.ts

This example shows how to cancel a server request and is built for the E-panel in POS015. The script subscribes to the `Requesting` event and when that event is raised it checks the name of the project leader. If the name is invalid, it cancels the request and shows an error message.

### H5SampleCustomColumns.ts

This script adds a column to the grid and populates the new column with dummy data.

### H5SampleCustomDialog.ts

This script creates and displays a custom dialog before proceeding with a request.

### H5SampleDrillback.ts

This script takes a customer ID from the script arguments and launches CRS610/E for this customer using a standard Portal drillback URL. The drillback is invoked using the Infor Portal CE JavaScript API.

### H5SampleExportToExcel.ts

Displays a dialog that gives user the option to export columns to MS Excel or Google Sheets.

## H5SampleHideColumns.ts

The script is designed to be launched in MMS001 and is responsible for hiding the columns with the names "MMITNO" and "MMITDS".

NOTE: Limitations exist on action/s that do not re-render panel, for example, on searching for a query; on edit view cancel; and on personalization/s. Users need to refresh panel after these actions to hide columns.

## H5SampleImageFromList.ts

This script shows an image when a list row is selected, it gets the text in the first column to build a URL to an image. The image is loaded and displayed next to the list. The base URL to the images is retrieved from the script arguments. For the script to work properly, there must be images in the folder of the URL whose names (not including the file extension) match the values of the first column in the list.

## H5SampleIonApiService.ts

This script retrieves the user GUID and email from the IFS (Portal APIs are deprecated last 10/30/2024) and M3 ION APIs. You may need to set the OAuth token manually to run the script in your development environment. Please see the ION API chapter for more details.

## H5SampleMFormsAutomation.ts

This example retrieves the user ID from the user context, creates an automation that starts MNS150, and opens the user in change mode.

## H5SampleMIService.ts

This script executes M3 API calls to retrieve user data using the MIService utility.

**Caution:**

Some minifiers report an error on the occurrence of Promise.catch as `catch` is a reserved word. To work around this, you can pass the `catch` function as the second parameter to the `then` function instead.

## H5SampleOpenFieldHelp.ts

This example displays through a dialog box the field help of the corresponding element argument attached to it. The script adds a `click event` to the attached `element` which opens the field help of the element based on the argument supplied. When there is no argument, the script will use the element name where it is attached.

## H5SampleOptionButton.ts

Adds one or more option buttons that execute a list option when clicked.

## H5SamplePreviewListHeader.ts

This example is a program-specific script loaded from the MNEAI element of an H5 panel. This script searches for two specific elements that contain the headers and the row content to be displayed in a table. When this script is loaded in CRS020/F, a table with one row of data is displayed.

## H5SampleRegexValidator.ts

This script validates the content of the configured fields on a panel and shows a validation message for the first validation that fails. Validation runs on Enter or Next, and on failure, the request will be cancelled, and the user will not be able to continue until the validation errors have been addressed.

## H5SampleRequestTracer.ts

This script logs the command type and command value for the `Requesting`, `Requested` and `RequestCompleted` events on a panel.  It uses the `InstanceCache` to ensure that the script is attached only once for a program instance.

## H5SampleShowOnMap.ts

This is a program-specific script for OIS002/F. It adds a custom button element in the panel that opens a location in Google Maps depending on the latitude, longitude, and zoom values specified in the text fields.

## H5SampleShowXML.ts

This example displays the raw `XML` content of an `MForm` from the `View Definitions` file. The script retrieves the `XML` file and converts text symbols to `HTML` codes to properly display the text symbols in a Message Dialog.

## H5SampleUserDetails.ts

This script adds a custom `ButtonElement` on the panel. When the button is clicked, a message dialog shows M3-related data about the logged-on user.

# Chapter 12
# APPENDIX

12

## Node.js

Node.js should be installed to be able to use the web server included in the SDK samples in developing scripts. You can skip this section if you have a working Node.js installation or if you do not want to use the web server.

### Install Node.js

Download and install Node.js from http://nodejs.org/

When the installation is complete you can follow the steps in the next two sections to verify that installation's work. Note that the instructions are for Microsoft Windows operating systems only. Refer to the Node.js documentation for other operating systems.

### Verify the Node Package Manager

Follow these steps to verify that the Node package manager works.

1 Verify that the following folder exists and create it manually if not.

- `C:\Users\<userid>\AppData\Roaming\npm`
- You need to show Hidden items in Windows Explorer to be able to see the AppData folder.

2 Open a Windows Command Prompt window.

3 Run the following command.

- `npm -version`

4 Verify that a version number is printed, such as 2.15.8.

- If the command fails, verify that the `npm` directory has been created. See previous step and create it if necessary.
- When the directory has been created, retry the `npm -version` command.

- On some operating systems, the command might be complete even if the npm folder is missing. In these cases, the installation of the node packages will fail. This can be solved by manually creating the npm folder.
- On some operating systems, you might have to restart the computer after adding the npm folder.

## Verify the Node Executable

Follow these steps to verify that Node executable works.

1 Open a Windows Command Prompt window.

2 Run the following command.

- node -v

3 Verify that a version number is printed, such as v4.4.7.

- If the command fails, it could be that the node directory is not on the Windows path.
- Add the following directory to the Windows System Path.
  - `C:\Program Files\nodejs`
- Close the command Window, start a new one and test `node -v` again.
  - The command should succeed this time.
- To apply the new Windows path, the computer might have to be restarted.

# TypeScript

Follow these steps to install the TypeScript compiler. If you choose to use Visual Studio, you can skip this part as the IDE already includes a compiler plugin.

1 Open a Windows Command Prompt window.

2 Run the following command:

- `npm install -g typescript@2.2`

3 Verify installation. A version number should be printed when you run the following command:

- `npm typescript -version`

**Note:** Minimum typescript version is 2.2

# IDS Web Component Datagrid Scan Tool

This program is designed for H5 Script developers to identify potential issues in H5 New UI, which utilizes the new `IDS Web Component Datagrid`. Because this `datagrid` encapsulates its elements within a `ShadowRoot`, the tool provides a comprehensive list of jQuery codes that may no longer function correctly. Specifically, direct selection of cells, rows, and positioning fields will not work as expected in the H5 New UI when using the `IDS Web Component Datagrid`.

Running the application:

1. Unzip the `H5ScriptSDK_10.4.1_2025xxxx.zip`
2. On `H5ScriptSDK_10.4.1_2025xxxx\Samples`, run `InstallWebServer.cmd`
3. On `H5ScriptSDK_10.4.1_2025xxxx\Samples\scantool` run `StartScanTool.cmd`
4. Open chrome and enter `localhost:8008`.

Using the application:

1. On H5A, open Administration Tools - H5 Administration - Data Files
2. Set Change File Type to H5 Script
3. Select all and click export.
4. Open the scan tool.
5. Click on the folder icon.
6. Select the exported H5 Script zip file.
7. View the results and analyze if the codes are safe (does not access element inside the datagrid).