PSWriteHTML Unofficial Manual

Release 1.41.1

PSWriteHTML created by Przemysław Kłys - Evotecl t

Table of Contents

List of Figures

Chapter 1: What is PSWriteHTML?

Table of Contents

1.1 About this manual

This manual is designed to help you quickly understand the capabilities of PSWriteHTML and how to implement it in your PowerShell scripts. Each chapter provides practical examples, best practices, and detailed explanations of the module's features.

Nico Zanferrari (https://github.com/nicozanf) assembled it, after reading all the available documentations - mainly from the author Przemysław Kłys (EvotecIT) - see the last chapter for a list of sources, including video tutorials. Surely AI made a lot of work, but most of the results were manually refined and carefully reviewed. Nowadays it is maintained as an open-source project on GitHub at https://github.com/nicozanf/PSWriteHTML-doc . The resulting web manual is automatically generated from the repository on https://nicozanf.github.io/PSWriteHTML-doc , along with the pdf and epub versions.

Contributions, feedback, and suggestions are always welcome! Just open an issue or submit a pull request on the GitHub repository. If you find this manual useful, please consider starring the repository to show your support.

1.2 PSWriteHTML Overview

PSWriteHTML is an open-source PowerShell module created by Przemysław Kłys (Evotec) designed to automate the creation of rich, interactive, and modern HTML documents, reports, dashboards, and emails without requiring manual HTML, CSS, or JavaScript authoring.

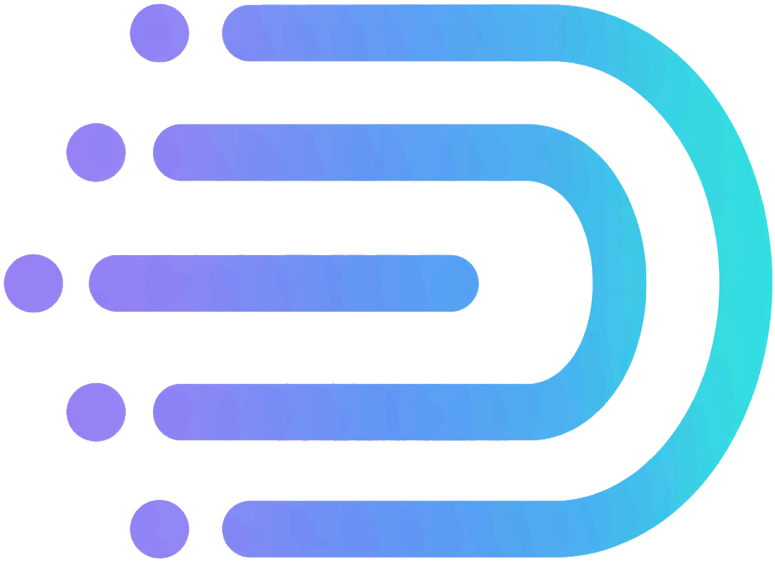

Figure 1.1. *PSWriteHTML makes report generation in PowerShell simple, flexible, and visually engaging.*

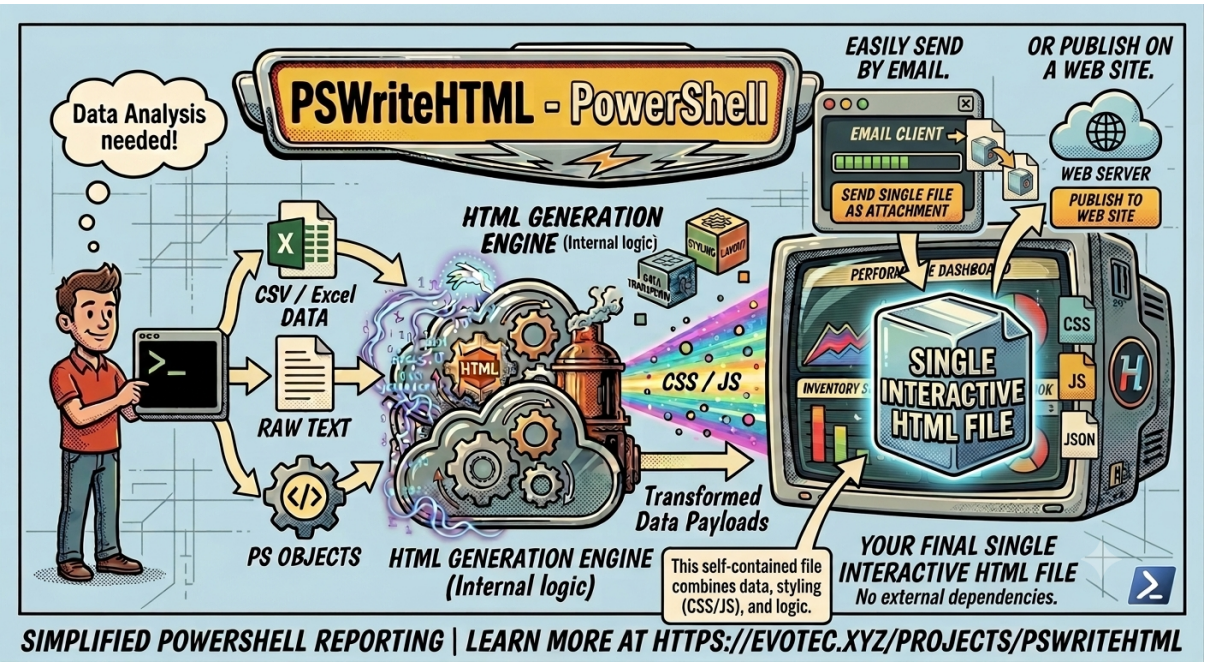

Figure 1.2. The PSWriteHTML rendering engine transforms PowerShell data into HTML reports.

1.3 Why PSWriteHTML?

Generating IT reports, audit logs, system health dashboards, or automated client emails often requires presenting complex datasets clearly. PSWriteHTML bridges the gap between raw data collection in PowerShell and professional presentation.

1.3.1 Key Benefits

- **Zero Web Development Required:** Write pure PowerShell using a declarative Domain-Specific Language (DSL).
- **Interactive Data Representation:** Out-of-the-box support for search, pagination, sorting, and export (PDF, CSV, Excel) powered by DataTables.
- **Rich Visualization:** Integrate interactive charts (bar, line, pie, donut, gauge) powered by ApexCharts and *Chart.js*.
- **Flexible Layout Systems:** Organize content using tabs, accordions, multi-column panels, grids, and collapsible sections.
- **Network & Flow Diagrams:** Render visual topology, organizational charts, and relational graphs using Vis Network.
- **Email & Document Ready:** Generate inline CSS emails optimized for Microsoft Outlook or standalone single-file HTML reports.

Native PowerShell includes the `ConvertTo-Html` cmdlet, but its output is static, unstyled, and difficult to format. PSWriteHTML replaces plain text tables with dynamic dashboards featuring interactive tables, real-time charts, tabbed navigation, collapsible sections, custom styling, and diagramming capabilities.

1.4 Common Use Cases

1. **Active Directory Auditing:** Reporting on inactive accounts, password expiration dates, and group membership changes.
2. **Infrastructure Monitoring:** Daily health-check summaries for VMware, Hyper-V, Azure resources, or network devices.
3. **Office 365 & Exchange Reports:** Displaying mailbox usage, license allocation, and security compliance scores.
4. **Automated HTML Emails:** Sending formatted, inline-styled alert notifications via SMTP.

Tip Because PSWriteHTML bundles required JavaScript and CSS libraries into self-contained HTML files (or references CDN sources), the output reports can easily be hosted on static web servers, AWS S3, or sent directly as email attachments. See Chapter 10: Advanced Features, Tips, and Troubleshooting

---

1.5 Quick Start Example

Here is a minimal working example that collects services data from your computer and generates an interactive report saved to disk:

```
Get-Service -ErrorAction SilentlyContinue | Out-HtmlView
```

This is an example of the power of PSWriteHTML: with a single command you can transform some data into a beautiful html page, with pagination, sorting, advanced searching capabilities, and even export in Excel/CSV/PDF !

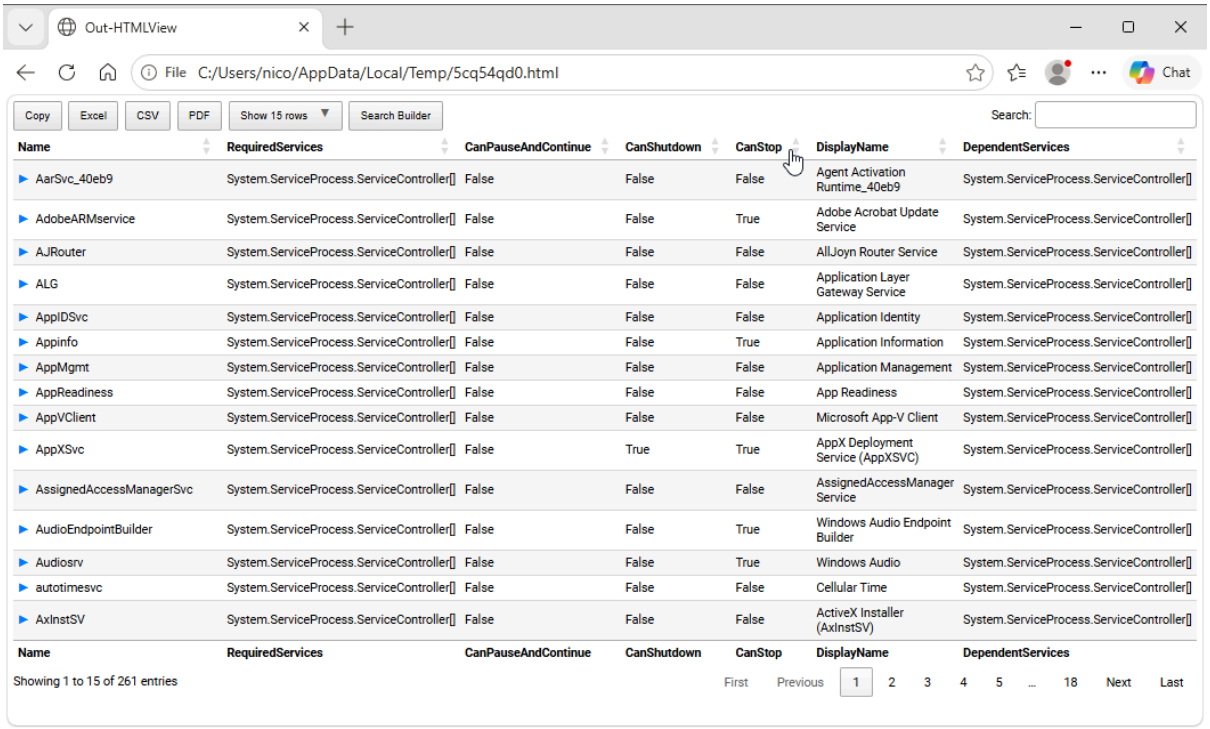

Figure 1.3. An interactive HTML table generated with the `Out-HtmlView` shortcut.

But this simple *Out-HtmlView* cmdlet in fact is a shortcut for the underlying *New-HTML* command, which allows you to further customize the report (for example selecting the columns included):

```
Import-Module PSWriteHTML # need PSWriteHTML already installed

# Gather services while suppressing permission errors
$Services = Get-Service -ErrorAction SilentlyContinue |
    Select-Object -Property Name, DisplayName, Status, StartType

# Generate HTML Report
New-HTML -Title "Service Status" -Show {
            New-HTMLTable -DataTable $Services
}
```

1.6 Core Architecture & DSL Concept

1.6.1 Component Hierarchy

The general layout structure follows a logical container model:

```
New-HTML
├── New-HTMLTab (Tab 1)
│   └── New-HTMLSection
│       └── New-HTMLPanel
│           └── New-HTMLTable / New-HTMLChart
└── New-HTMLTab (Tab 2)
    └── New-HTMLSection
        └── New-HTMLPanel
            └── New-HTMLDiagram
```

1.7 How PSWriteHTML Works: The Rendering Engine

PSWriteHTML acts as an abstraction layer over modern front-end web technologies. When you execute a `New-HTML` script block:

1. **AST & DSL Evaluation:** The module processes your nested PowerShell commands (`New-HTMLTab`, `New-HTMLSection`, `New-HTMLTable`, `New-HTMLChart`) and translates your PowerShell objects into structured JSON and HTML markup.
2. **Library Dependency Resolution:** The engine automatically attaches and configures industry-standard web libraries behind the scenes:
    - **DataTables:** For dynamic searching, column filtering, pagination, and multi-format data exports (CSV, Excel, PDF).
    - **ApexCharts / Chart.js:** For rendering interactive line, bar, pie, radial, and donut charts.
    - **FontAwesome:** For vector icons across cards, buttons, and navigation tabs.
    - **Bootstrap & Custom Layout CSS:** For responsive grids, modern flexbox containers, and card styling.

Next Chapter: Chapter 2: Installation and Module Setup

Chapter 2: Installation, Dependencies, and Setup

Table of Contents

2.1 Installation Overview

Before using **PSWriteHTML** to construct interactive reports, dashboards, or emails, you must properly set up the PowerShell environment and ensure all core and optional module dependencies are available.

PSWriteHTML is distributed via the PowerShell Gallery and supports cross-platform execution on Windows, macOS, and Linux.

2.2 Prerequisites & Requirements

PSWriteHTML works across major PowerShell editions and operating systems:

- **PowerShell 5.1** (Windows PowerShell included with Windows 10/11 & Windows Server 2016+)
- **PowerShell 7.x+** (PowerShell Core — Windows, macOS, Linux)
- **Execution Policy:** Must permit running locally installed scripts (e.g., `RemoteSigned` or `Unrestricted`).

Note PowerShell 7.x is recommended for optimal performance, especially when processing large datasets or generating complex multi-tab reports with heavy DataTables integration.

This manual was written for version 1.41.0 and later of PSWriteHTML. For older versions, refer to the archived documentation on the GitHub repository.

2.3 Module Dependencies

PSWriteHTML is designed as an ecosystem parent/child module. Depending on the visual compo-

nents you use (charts, diagrams, color palettes), PSWriteHTML automatically utilizes or suggests complementary modules developed by Evotec:

2.3.1 Primary Dependencies

Module Name	Purpose / Functionality
PSSharedGoods	Core helper functions, file handling, and string manipulation.
PSWriteColor	Enhanced console output formatting during report generation.
Connectimo	Optional network visualizer integration for topology diagrams.

Warning When installing PSWriteHTML via `Install-Module`, PowerShell automatically resolves and installs required dependencies like `PSSharedGoods`. If installing manually in an isolated/air-gapped environment, you must copy these dependency modules manually.

2.4 Installation Methods

2.4.1 Method 1: Installation from PowerShell Gallery (Recommended)

The quickest way to install PSWriteHTML for the current user or system-wide is using `Install-Module`.

For the Current User (No Admin Rights Required):

```
Install-Module -Name PSWriteHTML -Scope CurrentUser -Repository PSGallery -Force
```

System-Wide (Requires Administrator Privilege):

```
Install-Module -Name PSWriteHTML -Scope AllUsers -Repository PSGallery -Force
```

2.4.2 Method 2: Offline / Air-Gapped Installation

If your target machine lacks direct internet access (e.g., secure production servers), download the module on an internet-connected machine and transfer it.

1. **Save the module and dependencies to a local folder:**

   ```
   Save-Module -Name PSWriteHTML -Path "C:\OfflineModules" -Repository PSGallery
   ```

2. **Copy the folder** to the target machine's PowerShell module directory:
   - **Windows PowerShell 5.1:** `C:\Program Files\WindowsPowerShell\Modules\`
   - **PowerShell 7 (All Users):** `C:\Program Files\PowerShell\7\Modules\`
   - **PowerShell 7 (Current User):** `$HOME\Documents\PowerShell\Modules\`

3. **Verify the path is listed in environment variables:**

   ```
   $env:PSModulePath -split ';'
   ```

2.4.3 Method 3: Automated Scripting / CI/CD Pipeline

For automated deployment pipelines (GitHub Actions, Azure DevOps, Jenkins), ensure the package provider is updated before installing:

```
# Trust PowerShell Gallery and install silently
Set-PSRepository -Name 'PSGallery' -InstallationPolicy Trusted
Install-Module -Name PSWriteHTML -Scope CurrentUser -Force -AllowClobber
```

2.5 Verifying the Installation

After completing installation, verify that the module is imported correctly and inspect available cmdlets.

1. **Check Installed Version:**

```
Get-InstalledModule -Name PSWriteHTML
```

2. **Import and Inspect Available Cmdlets:**

```
Import-Module PSWriteHTML -Verbose
Get-Command -Module PSWriteHTML | Select-Object -First 10
```

Expected output includes core functions like ``New-HTML``, ``New-HTMLTab``, ``New-HTMLTable``, and ``New-HTMLChart``.

Figure 2.1. PowerShell modules available after installing PSWriteHTML and its dependencies.

Successful setup allows instant execution of DSL-based layout commands.

If you should face any errors with the PowerShell execution policy that prevents script execution, you can normally resolve it by running the following command:

```
Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
```

2.6 Updating & Uninstalling

2.6.1 Updating to the Latest Version

To update PSWriteHTML and pull in the latest web libraries (ApexCharts, DataTables, FontAwesome updates):

```
Update-Module -Name PSWriteHTML -Force
```

2.6.2 Uninstalling the Module

To remove PSWriteHTML and clean up the environment:

```
Remove-Module -Name PSWriteHTML -ErrorAction SilentlyContinue
Uninstall-Module -Name PSWriteHTML -AllVersions -Force
```

2.7 Importing the module

As usual with PowerShell modules, before using the module you should import it with the following command:

```
Import-Module PSWriteHTML
```

But this is not mandatory because PowerShell 3.0+ supports module auto-loading: when you execute a command belonging to an installed module, PowerShell automatically imports the module, provided the module is in $env:PSModulePath and you're using a modern version of PowerShell. From now on, all examples in this manual will not explicitly import the module at the beginning of the script.

On the other hand, if you want to use the module in a script, it is recommended to import it explicitly at the beginning of the script, so that:

- makes the script's dependency obvious
- causes a failure immediately if PSWriteHTML cannot be loaded
- makes troubleshooting easier

2.8 Troubleshooting Common Setup Issues

Issue: Script Execution Disabled

- *Error:* `...cannot be loaded because running scripts is disabled on this system.`
- *Solution:* Run `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser`.

Issue: NuGet Provider Prompt

- *Error:* `NuGet provider is required to continue...`
- *Solution:* Force NuGet provider installation with `Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Force`.

Issue: Dependency Conflict

- *Error:* Multiple versions of `PSSharedGoods` loaded.
- *Solution:* Remove older module versions using `Uninstall-Module PSSharedGoods -AllVersions` and re-import PSWriteHTML.

Next Chapter: Chapter 3: Basic HTML Structure and Document Flow

Chapter 3: Basic HTML Structure and Document Flow

Table of Contents

3.1 Structure Overview

Building reports with **PSWriteHTML** relies on a clear, hierarchical layout engine based on PowerShell script blocks (`{ ... }`). Understanding how components wrap inside one another allows you to build everything from a quick single-page summary table to a complex, multi-tab executive dashboard.

This chapter details the fundamental document containers, structural scope rules, page configuration options, and the general flow of a PSWriteHTML script.

3.2 The Component Hierarchy Model

PSWriteHTML enforces a top-down structural sequence. Each element acts as a visual container for the elements beneath it.

```
Document Container    ──>  New-HTML
    └── Tab Container  ──>      ├── New-HTMLTab (Page / Tab Level)
        └── Section    ──>      │       └── New-HTMLSection (Row / Banner Level)
            └── Panel  ──>      │               └── New-HTMLPanel (Card / Content
Container)
                └── Content ──> │                       ├── New-HTMLTable
                                │                       ├── New-HTMLChart
                                │                       └── New-HTMLText
```

3.2.1 Container Levels Explained

1. **Root Container (New-HTML):** The mandatory parent wrapper. Configures global settings, file output path, HTML header metadata, themes, and CSS imports.
2. **Tabs (New-HTMLTab):** Primary navigation containers. Each tab generates a top-level tab button to switch between independent views within the same HTML file.

3. **Sections (New-HTMLSection):** Horizontal grouping rows within a tab. Sections organize content vertically and can feature section header titles.
4. **Panels (New-HTMLPanel):** Content cards inside a section. Panels hold visual widgets such as tables, charts, diagrams, or free-form text blocks, arranging them into structured columns or grid blocks.

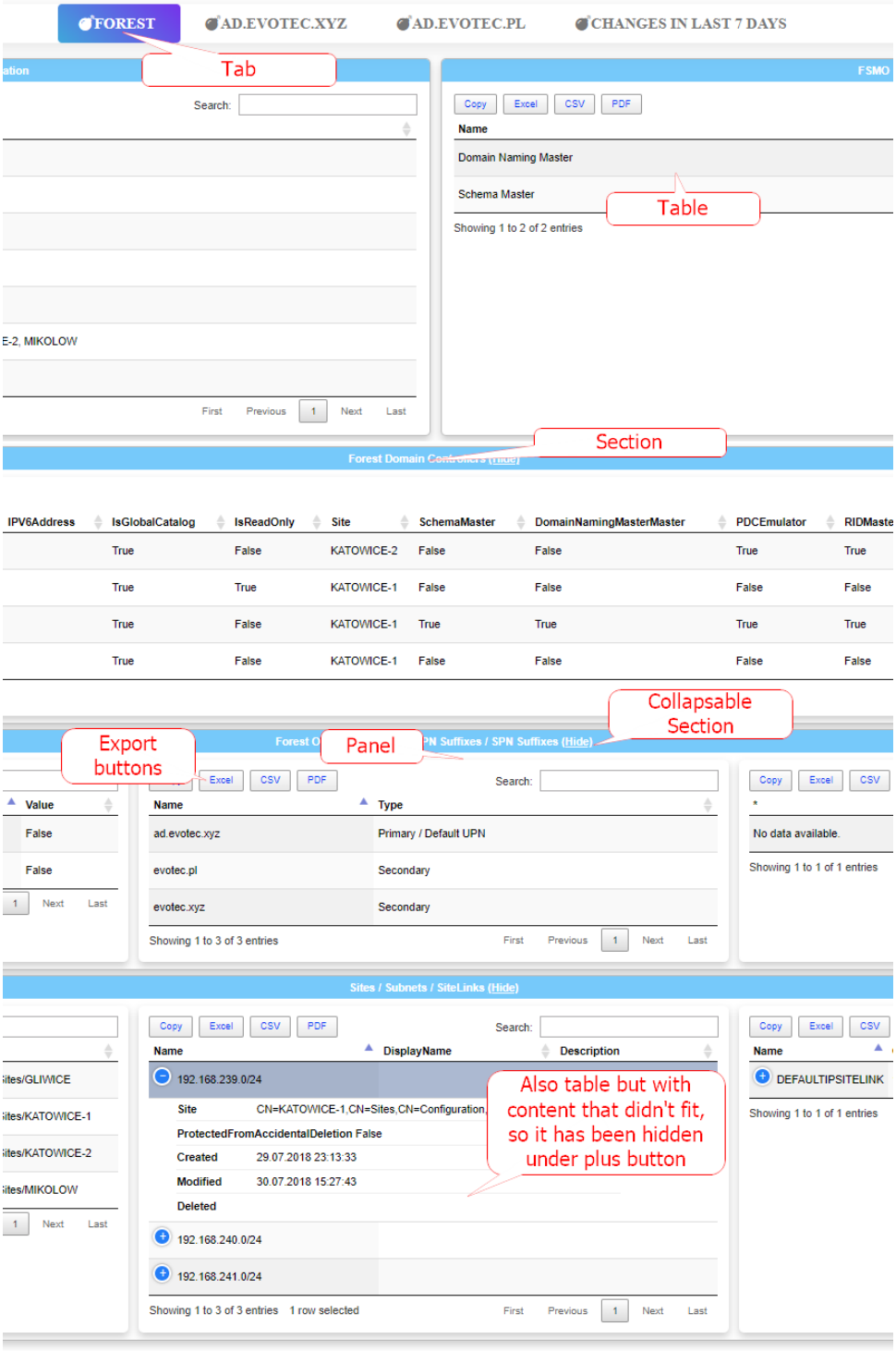

Figure 3.1. A tabbed dashboard built from nested PSWriteHTML layout commands.

A tabbed dashboard created using simple nested script blocks in PowerShell.

3.3 Core Syntax: Script Block Nesting

PSWriteHTML utilizes a nested **PowerShell DSL (Domain-Specific Language)** structure. Instead of building HTML tags manually, you nest PowerShell script blocks (`{ ... }`) within parent components.

3.3.1 Basic Syntax Pattern

```
New-HTML -FilePath "C:\Reports\Dashboard.html" -Title "My First Report" {
    New-HTMLTab -Name "Overview" {
        New-HTMLSection -HeaderText "System Overview" {
            New-HTMLPanel {
                New-HTMLText -Text "Welcome to the automated system dashboard."
            }
        }
    }
}
```

Note The curly braces `{ ... }` define the scope of each container. Indenting nested blocks makes complex reports significantly easier to maintain and troubleshoot.

3.4 Configuring Document-Level Properties (`New-HTML`)

The New-HTML cmdlet controls global document behaviors, including output file path, title, theme, and asset resolution. It is the only mandatory cmdlet in a PSWriteHTML script, except Out-HtmlView.

Hint You can see the official reference pages for all the PSWriteHTML commands with a complete list of parameters and their descriptions on the online GitHub repository.

Key parameters include:

3.4.1 Output Controls

- **-FilePath:** Specifies the absolute or relative target path for the saved `.html` file. If `-FilePath` is omitted, the module generates a temporary file in the system's temp directory.
- **-Show:** Switch parameter that automatically launches the default web browser upon file creation. You can also use `-ShowHTML` which is an alias for the same functionality.
- **-Title:** Sets the HTML `<title>` tag displayed in browser tabs.

3.4.2 HTML generation Options

- **-Online:** Loads web assets (Bootstrap, DataTables, FontAwesome) from Internet via public CDN URLs. [default]
- **-Offline:** Embeds all static script assets in the HTML file itself (ideal for air-gapped environment reporting).

3.5 Simple Example of PSWriteHTML usage

Here is a first example of its usage:

```
New-HTML -Title "My First Report" -FilePath "C:\Temp\Dashboard.html" -Show {
    New-HTMLTab -Name "Overview" {
        New-HTMLSection -HeaderText "System Overview" {
            New-HTMLPanel {
                New-HTMLText -Text "Welcome to the automated system dashboard."
            }
        }
    }
}
```

It produces a simple data table, as shown below:

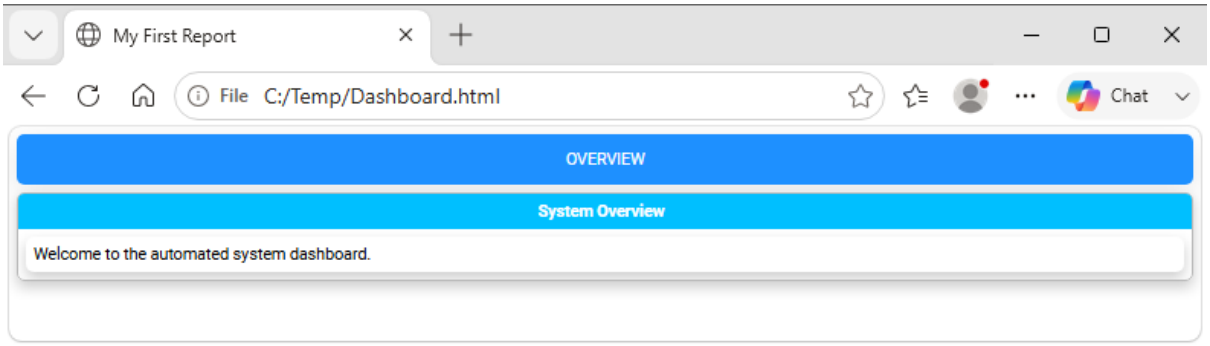

Figure 3.2. A first PSWriteHTML report rendered from a PowerShell script.

3.6 Building Your First Complete Report

The following complete script demonstrates how to query Windows services, disk drives, and system uptime, then render them into a formatted multi-panel single-tab report.

```
Import-Module PSWriteHTML

# 1. Collect Infrastructure Data
$Services = Get-Service | Where-Object Status -eq 'Running' | Select-Object -First
5 -Property Name, DisplayName, Status
$Disks    = Get-CimInstance -ClassName Win32_LogicalDisk | Select-Object DeviceID,
@{N='Free Space (GB)'; E={[math]::Round($_.FreeSpace/1GB, 2)}}, @{N='Size (GB)';
E={[math]::Round($_.Size/1GB, 2)}}

# 2. Render Document Structure
New-HTML -Title "Local Node Assessment" -FilePath
"$env:USERPROFILE\Desktop\LocalNode.html" -Show {

    New-HTMLTab -Name "System Status" {

        # Section 1: Running Services
        New-HTMLSection -HeaderText "Core Running Services" {
            New-HTMLPanel {
                New-HTMLTable -DataTable $Services
            }
        }
```

```
        # Section 2: Storage Allocation
        New-HTMLSection -HeaderText "Storage Overview" {
            New-HTMLPanel {
                New-HTMLTable -DataTable $Disks
            }
        }
    }
}
```

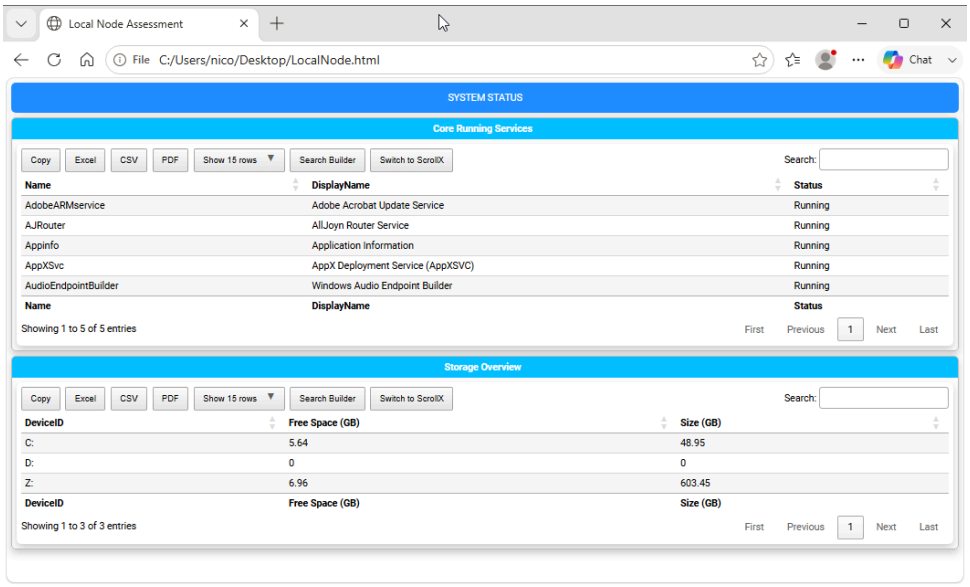

Figure 3.3. A local-node assessment report with service, disk, and uptime panels.

Resulting table presentation created by nesting `New-HTMLTable` within `New-HTMLPanel` containers.

3.7 Deep Dive: Online vs. Offline Resource Embedding

One of the most critical architecture features of PSWriteHTML is how it handles web assets (JavaScript libraries, CSS stylesheets, and font glyphs). Understanding this mechanism ensures your reports function reliably across both internet-connected and air-gapped enterprise environments.

3.7.1 Online Mode (Default)

When generating a document without extra asset flags (or explicitly using `-Online`):

- **Mechanism:** The generated HTML file includes lightweight link tags pointing to high-speed public Content Delivery Networks (CDNs) for DataTables, FontAwesome, ApexCharts, and jQuery.
- **File Size:** Output *.html* files remain tiny—often under **50 KB**—because no asset binaries are stored inside the document.
- **Best Used For:** Internal dashboards hosted on corporate web servers where endpoint devices have open internet access.

3.7.2 Offline Mode (100% Self-Contained Output)

When building reports for secure, isolated, or air-gapped networks, add the `-Offline` switch parameter to `New-HTML`:

```
New-HTML -Title "Air-Gapped Infrastructure Report" -FilePath
"C:\Reports\Status.html" -Offline {
    ...
}
```

When `-Offline` is specified, PSWriteHTML changes how it builds the document:

1. **Local File Extraction:** Instead of referencing external CDN URLs, the engine reads the bundled JavaScript libraries, CSS stylesheets, and web font files directly from the installed PSWriteHTML module directory on your build server.
2. **Base64 Encoding & Inlining:** The engine reads the raw CSS, JS, and font binaries, converts them into Base64 data strings, and embeds them **directly inline** within `<style>` and `<script>` tags inside the single HTML output document.
3. **Zero External Dependencies:** No secondary asset folders, subdirectories, or auxiliary *.css*/*.js* files are created alongside your output file.

3.7.3 Key Architectural Takeaway

Because `-Offline` mode creates a **single, 100% self-contained HTML file**, you can copy, relocate, or host that single *.html* file anywhere:

- You can move it to an isolated IIS server without needing to copy accompanying asset subfolders.
- You can attach it to an email, open it off a USB drive, or view it on an isolated machine with no network connection.
- All table sorting, search filtering, dynamic charts, and vector icons will function completely offline.

Note Because all JavaScript engines and vector fonts are embedded into the file, an `-Offline` report file size typically starts around **2 MB to 5 MB**. For high-density reporting, this trade-off guarantees total portability across secure subnets.

3.8 Common Document Flow Mistakes

1. **Orphaned Content Cmdlets:** Placing a `New-HTMLTable` or `New-HTMLText` directly inside `New-HTML` without a surrounding `New-HTMLTab` or `New-HTMLSection` can cause formatting misalignment or missing element styles.
2. **Unclosed Braces:** Omitting a closing script block brace `}` breaks PowerShell syntax parsing. Always verify brace matching when adding deep nesting levels.
3. **Mixing Data Collection inside DSL Blocks:** For performance and readability, gather all data into variables **before** calling `New-HTML`. Executing heavy PowerShell cmdlets (like `Get-ADUser`) directly inside nested script blocks slows down document construction and complicates debugging.

Next Chapter: Chapter 4: Working with DataTables and Data Grids

Chapter 4: Working with DataTables and Data Grids

Table of Contents

4.1 DataTables Overview

The core strength of **PSWriteHTML** lies in its integration with DataTables. Instead of rendering static, unstyled HTML tables, the `New-HTMLTable` cmdlet converts standard PowerShell objects into fully interactive data grids complete with real-time searching, sorting, pagination, column visibility toggles, and direct data export options (PDF, Excel, CSV, Copy).

This chapter explains how to configure tables, apply conditional cell formatting, customize column properties, and optimize data grid performance.

4.2 Basic Table Generation (`New-HTMLTable`)

To create a table, use the New-HTMLTable cmdlet, passing a collection of PowerShell objects to the `-DataTable` parameter within a script block.

Here is a simple example of generating a table from the output of `Get-Service`:

```
$Services = Get-Service | Select-Object Name, DisplayName, Status, StartType |
Select-Object -First 15

New-HTML -Title "Service Overview" -FilePath "Services.html" -Show {
    New-HTMLTab -Name "Services" {
        New-HTMLSection -HeaderText "Local System Services" {
            New-HTMLPanel {
                New-HTMLTable -DataTable $Services
            }
        }
    }
}
```

It produces a simple interactive table, as shown below:

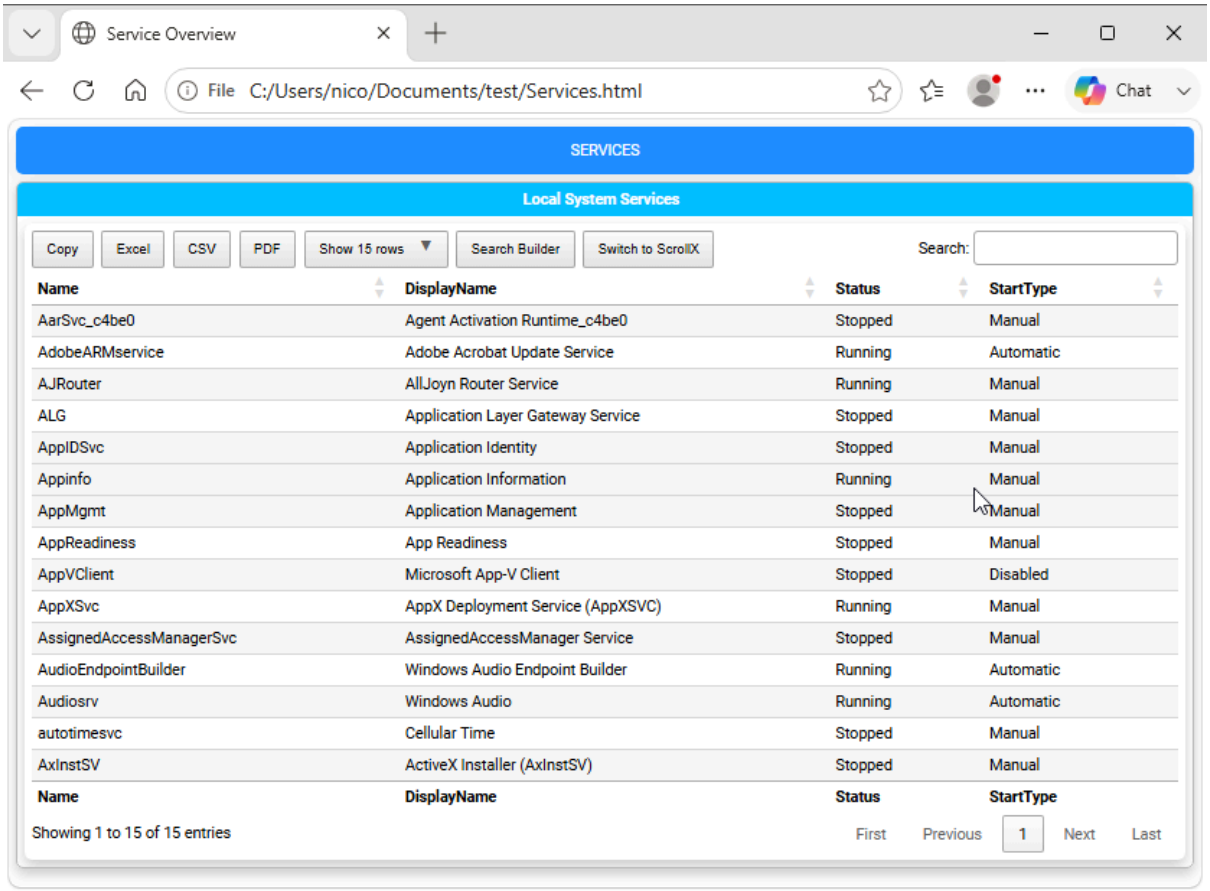

Figure 4.1. A first interactive DataTable with sorting, searching, and paging controls.

4.3 Interactivity Controls & Buttons

You can fine-tune table controls using parameters on New-HTMLTable :

4.3.1 Filtering and Pagination

- **-Filtering** Adds column-based search input fields at the header or footer of the table [default: Not enabled].
- **-Paging $true:** Enables pagination controls.
- **-PageLength 25:** Sets the default number of rows displayed per page (e.g., 10, 25, 50, 100).
- **-SearchState $true:** Preserves search keywords and pagination states in browser session storage across page reloads.

> **Warning** The `-Filtering` parameter is a switch, so (if ever needed) it must be specified without a value. Using `-Filtering $true` will not enable filtering and may cause unexpected behavior.

4.3.2 Export Buttons & Feature Toggles

PSWriteHTML allows users to download or copy table content directly from the generated HTML interface using the `-Buttons` parameter:

```
New-HTMLTable -DataTable $Services -Buttons 'copyHtml5', 'excelHtml5', 'csvHtml5',
'pdfHtml5', 'print'
```

Common Button Types:

- **copyHtml5:** Copies current table contents to the clipboard.
- **excelHtml5:** Downloads table data as a native `.xlsx` spreadsheet.
- **csvHtml5:** Exports raw data as a comma-separated values file.
- **pdfHtml5:** Generates a downloadable PDF document from the current table view.
- **print:** Opens a printer-friendly view of the table in a new browser tab.
- **pageLength:** Adds a dropdown menu allowing users to select how many rows to display per page.
- **searchPanes:** Adds a sidebar panel for multi-column filtering with checkboxes.
- **searchBuilder:** Adds a panel for building complex search queries across multiple columns.
- **columnVisibility:** Adds a dropdown menu allowing users to dynamically hide or show specific table columns.
- **toggleView:** Switches between standard table view and card-based layout for mobile-friendly displays.

By default, PSWriteHTML includes many buttons in the generated table interface (see screenshot above). You can customize which buttons appear, and even their order, by specifying a subset in the `-Buttons` parameter.

4.4 Sorting

You can control the initial sort order of a table by specifying the `-DefaultSortColumn` and `-DefaultSortOrder` parameters on `New-HTMLTable`. For example:

```
New-HTMLTable -DataTable $Services -DefaultSortColumn 'Name' -DefaultSortOrder
'Ascending'
```

4.5 Column Formatting & Customization (`New-TableContent` and `New-TableHeader`)

To customize individual columns, use nested New-TableContent and New-TableHeader script blocks. Use `New-TableContent` for cell formatting and `New-TableHeader` for displayed header names and header styling.

```
New-HTMLTable -DataTable $Services {
      New-TableContent -ColumnName 'Status' -Alignment center
      New-TableHeader -Names 'StartType' -Title 'Startup Type'
}
```

4.5.1 New-TableContent Parameters

- **-ColumnName:** Exact property name from the incoming PowerShell object.
- **-Alignment:** Text alignment: `left`, `center`, or `right`.

4.5.2 New-TableHeader Parameters

- **-Names:** Column name or names whose table headers should be customized.
- **-Title:** Custom display title for the selected table header.
- **-Alignment:** Text alignment: `left`, `center`, or `right`.

4.6 Conditional Formatting & Cell Highlighting (`New-TableCondition`)

Visual indicators are crucial for system health reports. PSWriteHTML allows conditional styling based on exact string values, numbers, or comparison operators using New-TableCondition.

4.6.1 Coloring Cells Based on Status

The following example highlights services based on whether they are running or stopped:

```
New-HTMLTable -DataTable $Services {

    # Highlight "Running" status in light green
      New-TableCondition -Name 'Status' -ComparisonType string -Operator eq -Value
'Running' -BackgroundColor '#d4edda' -Color '#155724'

    # Highlight "Stopped" status in light red with bold text
      New-TableCondition -Name 'Status' -ComparisonType string -Operator eq -Value
'Stopped' -BackgroundColor '#f8d7da' -Color '#721c24' -FontWeight bold
}
```

4.6.2 Comparison Operators

In addition to exact string matching, comparison logic supports numeric thresholds:

- **-Operator 'eq'** (Equal)
- **-Operator 'ne'** (Not Equal)
- **-Operator 'gt'** / **-Operator 'ge'** (Greater Than / Greater Than or Equal)
- **-Operator 'lt'** / **-Operator 'le'** (Less Than / Less Than or Equal)
- **-Operator 'contains'** (Substring match)

4.6.3 Numeric Range Example

```
$Disks = Get-CimInstance Win32_LogicalDisk | Select-Object DeviceID,
@{N='FreeSpaceGB'; E={[math]::Round($_.FreeSpace/1GB, 2)}}

New-HTMLTable -DataTable $Disks {
    # Highlight drives with less than 15 GB free space in warning red
       New-TableCondition -Name 'FreeSpaceGB' -ComparisonType number -Operator lt
-Value 15 -BackgroundColor '#ff4d4d' -Color '#ffffff'
}
```

4.7 Complete Example: Interactive Disk & Service Dashboard

Here is a full, ready-to-run script combining pagination, export buttons, column formatting, and

conditional color highlighting:

```
    # 1. Prepare Data
    $Services = Get-Service | Select-Object Name, DisplayName, Status, StartType |
Select-Object -First 20

    # 2. Build Document
    New-HTML -Title "Infrastructure Audit" -FilePath
"$env:USERPROFILE\Desktop\AuditReport.html" -Show {
        New-HTMLTab -Name "System Audit" {
            New-HTMLSection -HeaderText "Active Services State" {
                New-HTMLPanel {
                    New-HTMLTable -DataTable $Services -Buttons 'excelHtml5',
'pdfHtml5', 'columnVisibility' -PagingLength 10 {

                        # Center header and adjust title
                        New-TableContent -ColumnName 'Status' -Alignment center

                        # Formatting Rules
                        New-TableCondition -Name 'Status' -ComparisonType string
-Operator eq -Value 'Running' -BackgroundColor '#C8E6C9' -Color '#2E7D32'
                        New-TableCondition -Name 'Status' -ComparisonType string
-Operator eq -Value 'Stopped' -BackgroundColor '#FFCDD2' -Color '#C62828'
                    }
                }
            }
        }
    }
```

It produces a complete colored interactive table, as shown below:

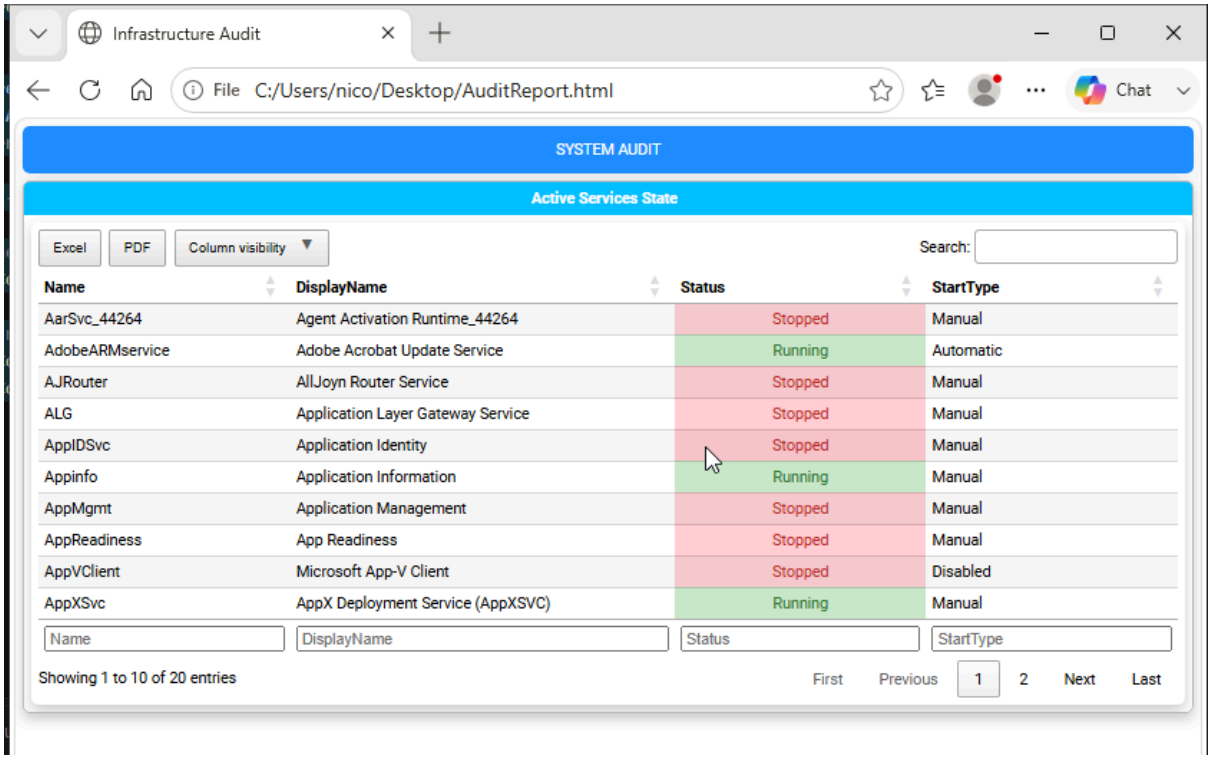

Figure 4.2. A complete DataTable combining export buttons, sorting, and conditional formatting.
Combining table conditions and controls results in clean, executive-ready documentation.

4.8 Performance Tips for Large Datasets

When outputting tables containing tens of thousands of rows (such as Active Directory event logs or large file system scans), consider these optimization guidelines:

1. **Limit Selected Properties:** Pre-filter objects with `Select-Object` in PowerShell before passing them to `-DataTable`.
2. **Disable Unnecessary Search Controls:** Do not use the `-Filtering` switch parameter if it's not needed, as this reduces browser processing overhead on massive DOM trees.
3. **Use Default Page Lengths:** Keep `-PageLength` at 25 or 50 so the browser only renders visible rows into memory.

Next Chapter: Chapter 5: Visualizing Data with Charts and Graphs

Chapter 5: Visualizing Data with Charts and Graphs

Table of Contents

5.1 Charts and Graphs Overview

While data grids excel at granular details, visual summaries are often necessary for executive dashboards and high-level health assessments. **PSWriteHTML** includes native support for interactive charts powered by ApexCharts, enabling you to build dynamic bar, line, pie, donut, area, radar, and gauge charts directly from PowerShell objects.

With the New-HTMLChart cmdlet, you can turn raw numerical metrics into responsive visual representations featuring tooltips, legends, animations, and image export options.

5.2 Chart Types Overview

PSWriteHTML supports multiple chart archetypes depending on the data structure:

Chart Type	Target Cmdlet / Parameter	Best Used For
Bar / Column	`New-ChartBar`	Comparing discrete values across categories (e.g., Disk space by volume, services by state).
Pie / Donut	`New-ChartDonut`	Displaying proportional distribution or percentages (e.g., OS breakdown, memory usage).
Line / Area	`New-ChartLine`	Tracking trends over continuous time series or sequential data points (e.g., CPU load history).
Gauge	`New-ChartRadial`	Representing progress toward a single threshold or target percentage (e.g., SLA compliance score).

5.3 Creating Your First Chart (`New-HTMLChart`, `New-ChartDonut`)

To render a chart, place one or more chart builder commands such as `New-ChartBar`, `New-ChartLine`, or `New-ChartDonut` inside the New-HTMLChart script block. The nested command determines the chart type.

5.3.1 Basic Donut Chart Example

The following example gathers fake account status distributions and displays them in an interactive donut chart:

```
# Prepare data categories, as an array of PSCustomObjects with two properties:
Status and Count
$Data = @(
    [PSCustomObject]@{ Status = 'Active';   Count = 142 }
    [PSCustomObject]@{ Status = 'Disabled'; Count = 18 }
    [PSCustomObject]@{ Status = 'Locked';   Count = 5 }
)

New-HTML -Title "Account Status Overview" -FilePath "AccountChart.html" -Show {
    New-HTMLTab -Name "Identity Summary" {
        New-HTMLSection -HeaderText "User Account States" {
            New-HTMLPanel {
                New-HTMLChart -Title "Account Breakdown" {
                    $Data | ForEach-Object {
                        New-ChartDonut -Name $_.Status -Value $_.Count
                    }
                }
            }
        }
    }
}
```

This is the result:

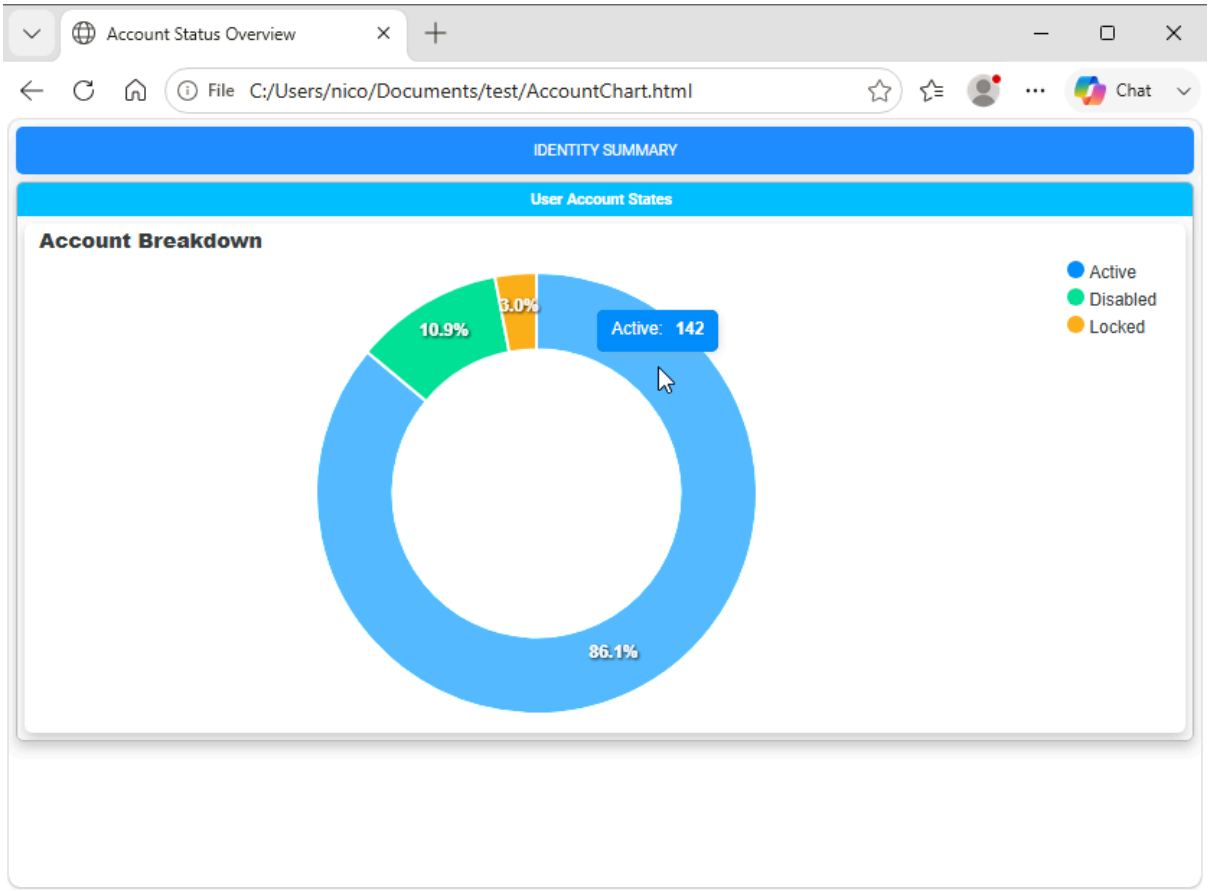

Figure 5.1. A donut chart showing the distribution of account statuses.

5.4 Bar and Column Charts (`New-ChartBar`)

Bar and column charts are ideal for comparing system statistics side by side, such as disk utilization across multiple servers.

5.4.1 Multi-Series Bar Chart Example

```
$DiskData = @(
    [PSCustomObject]@{ Drive = 'C:'; UsedGB = 120; FreeGB = 30 }
    [PSCustomObject]@{ Drive = 'D:'; UsedGB = 450; FreeGB = 50 }
    [PSCustomObject]@{ Drive = 'E:'; UsedGB = 200; FreeGB = 300 }
)

New-HTML -Title "Disk Capacity Dashboard" -FilePath "DiskChart.html" -Show {
    New-HTMLTab -Name "Storage" {
        New-HTMLSection -HeaderText "Drive Capacity Distribution (GB)" {
            New-HTMLPanel {
                New-HTMLChart -Title "Used vs Free Space" -Height 350 {
                    New-ChartBarOptions -Vertical
                    New-ChartLegend -Name 'Used Space (GB)', 'Free Space (GB)' `
                                    -Color '#E53935', '#43A047'

                    foreach ($Disk in $DiskData) {
                        New-ChartBar -Name $Disk.Drive `
```

```
                                        -Value $Disk.UsedGB, $Disk.FreeGB
                    }
                }
            }
        }
    }
}
```

This is the result:

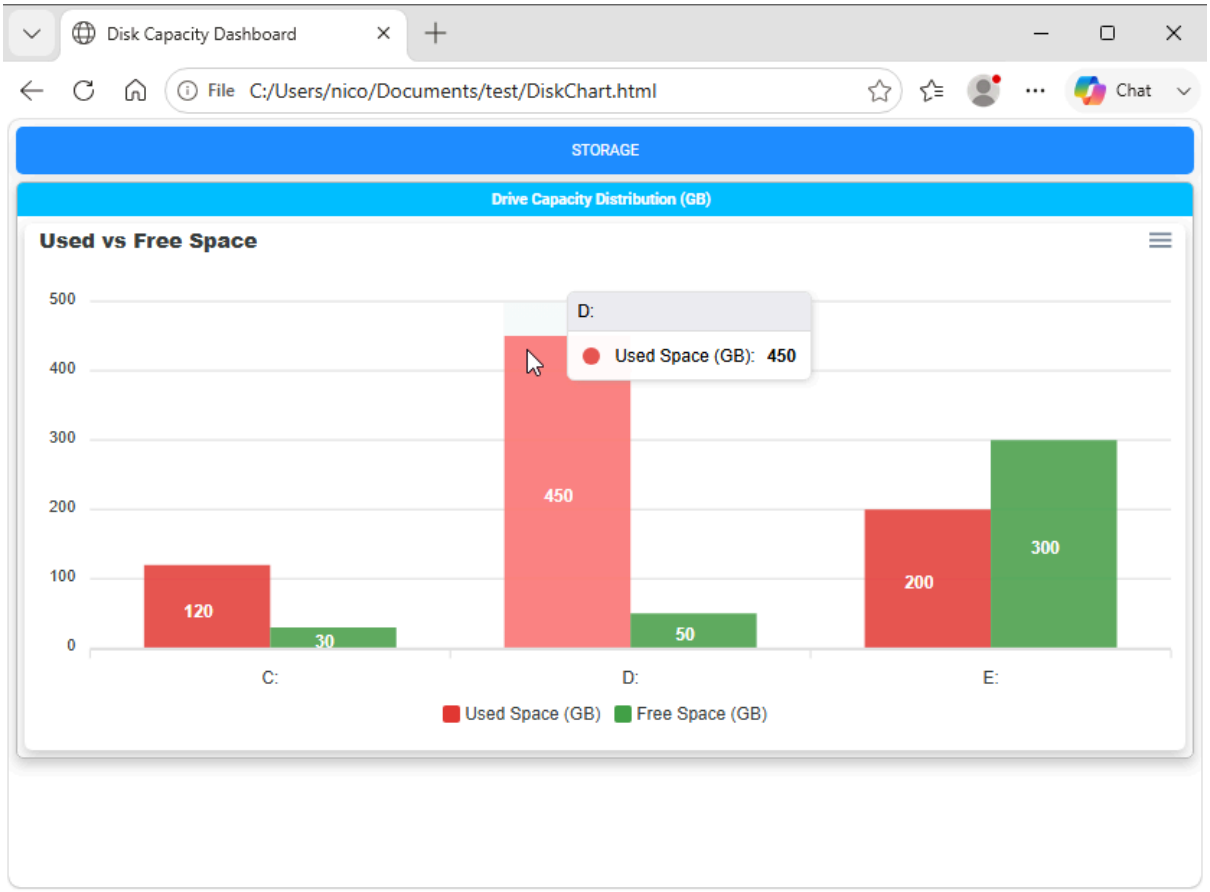

Figure 5.2. A multi-series bar chart comparing values across categories.

5.5 Line Charts (`New-ChartLine`, `New-ChartAxisX`)

Line charts are useful for showing how a value changes across an ordered sequence, such as CPU usage measured over several days. Use New-ChartAxisX to define the categories and New-ChartLine to provide one or more named series with matching values:

```
    $CpuHistory = @(
        [PSCustomObject]@{ Day = 'Monday';    Usage = 42 }
        [PSCustomObject]@{ Day = 'Tuesday';   Usage = 58 }
        [PSCustomObject]@{ Day = 'Wednesday'; Usage = 47 }
    )

New-HTML -Title "CPU Usage Dashboard" -FilePath "CPUChart.html" -Show {
    New-HTMLChart -Title "CPU Usage History" -Height 350 {
        New-ChartAxisX -Names @($CpuHistory | ForEach-Object { $_.Day })
```

```
        New-ChartLine -Name "CPU Usage (%)" `
                      -Value @($CpuHistory | ForEach-Object { $_.Usage }) `
                      -Color '#1976D2' -Curve smooth
    }
}
```

This is the result:

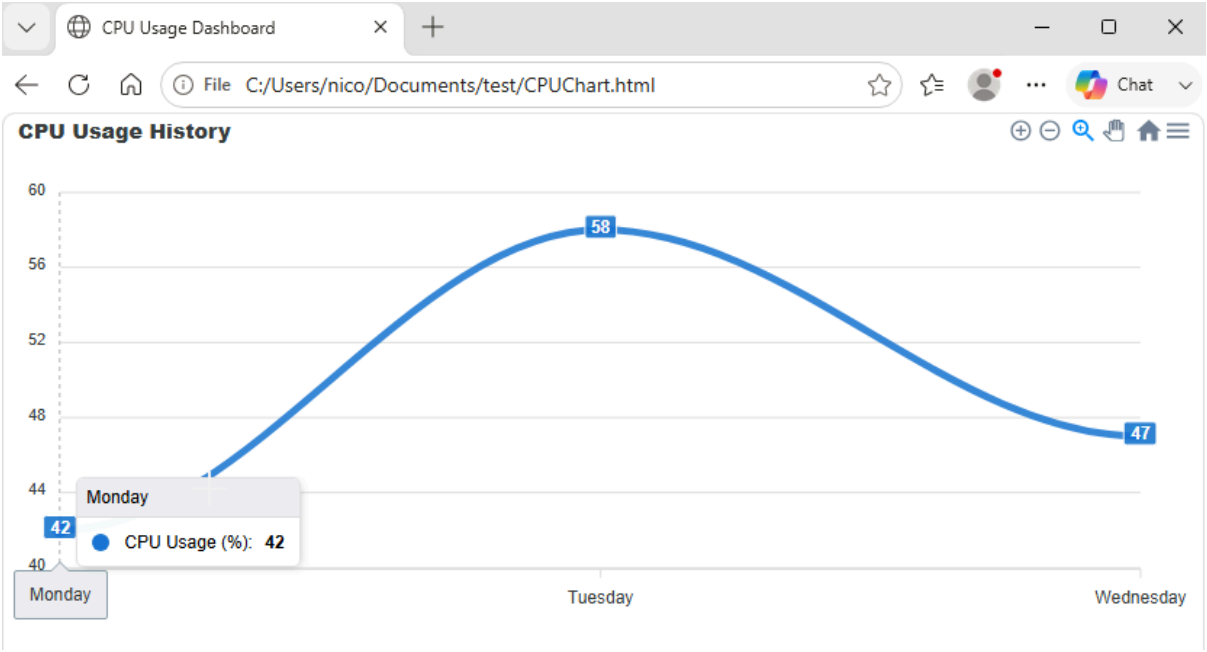

Figure 5.3. A line chart displaying changes across an ordered sequence.

Note that line chart have a built-in tooltip that displays the series name and value when hovering over a data point. You can also customize the line color, width, and style using parameters in *New-ChartLine*. Also, for line charts, a new menu appears in the top-right corner of the chart, allowing users to toggle series visibility, pan/zoom in/out, and export the chart as an image or PDF.

5.6 Radial Gauge Charts (`New-ChartRadial`)

Radial charts display a progress or percentage value against an implicit target of 100, making them suitable for metrics such as SLA compliance. Use New-ChartRadial , pass the metric label to `-Name` and its numeric value to `-Value`:

```
New-HTML -Title "SLA Dashboard" -FilePath "SLA-Chart.html" -Show {
    New-HTMLChart -Title "SLA Compliance" -Height 350 {
        New-ChartRadial -Name "SLA Compliance" -Value 92 -Color '#43A047'
    }
}
```

This example displays an SLA compliance score of 92 percent in a radial gauge.

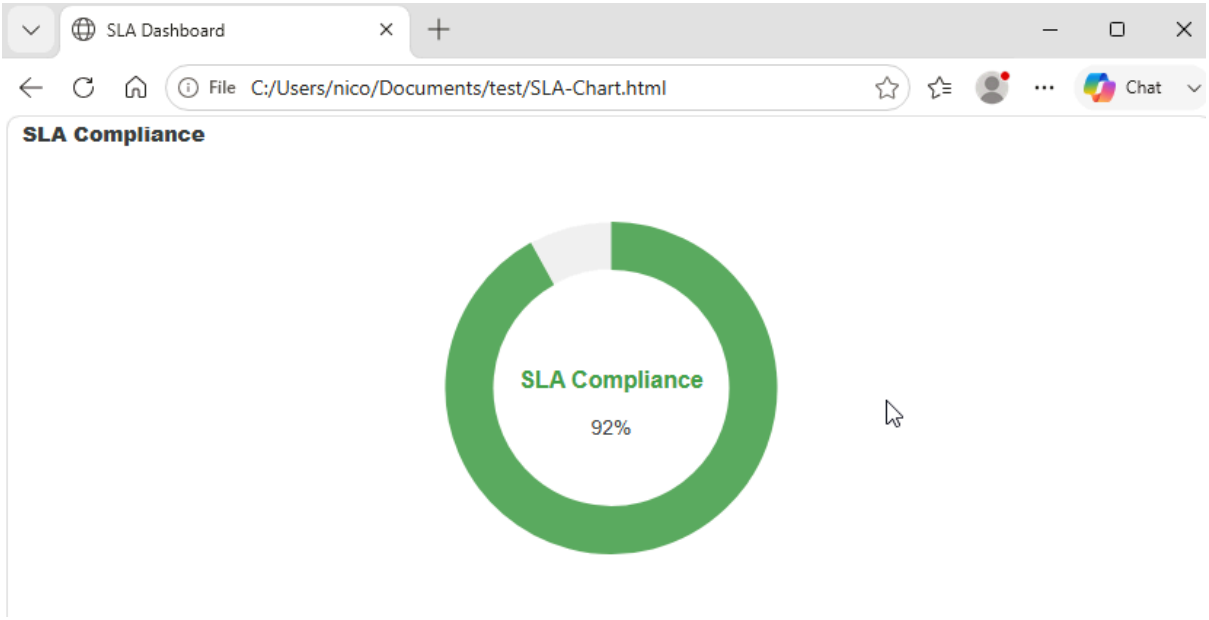

Figure 5.4. A radial gauge chart showing an SLA compliance score.

5.7 Interactive Geographic Maps (`New-HTMLMap`)

For geographic visualizations, New-HTMLMap renders an interactive map using one of PSWriteHTML's built-in regions. Use `-Map` to select `Poland`, `Usa_States`, `World_Countries`, or `European_Union`. You can customize the map with parameters such as `-AnchorName`, `-AreaTitle`, `-PlotTitle`, `-FillColor`, `-StrokeColor`, `-StrokeWidth`, `-ShowAreaLegend`, and `-ShowPlotLegend`. For data-driven area colors or plotted values, use `-MapSettings` to provide the map areas, plots, and legend configuration.

The following example creates a simple interactive world map:

```
New-HTML -Title "World Map" -FilePath "WorldMap.html" -Online -Show {
    New-HTMLSection -HeaderText "World Countries" {
        New-HTMLPanel {
            New-HTMLMap -Map "World_Countries" -AnchorName "WorldMap" -AreaTitle
"Selected Countries" -ShowAreaLegend -FillColor "#DCEAF7" -StrokeColor "#536878"
-StrokeWidth 1 {
                New-MapArea -Area "US" -Value 1 -Tooltip { "United States" }
                New-MapArea -Area "CA" -Value 2 -Tooltip { "Canada" }
                New-MapArea -Area "AU" -Value 3 -Tooltip { "Australia" }
                New-MapArea -Area "IT" -Value 4 -Tooltip { "Italy" }

                New-MapLegendSlice -Type area -Label "USA" -MinimumValue 1
-MaximumValue 1 -FillColor "#E74C3C"
                New-MapLegendSlice -Type area -Label "Canada" -MinimumValue 2
-MaximumValue 2 -FillColor "#3498DB"
                New-MapLegendSlice -Type area -Label "Australia" -MinimumValue 3
-MaximumValue 3 -FillColor "#2ECC71"
                New-MapLegendSlice -Type area -Label "Italy" -MinimumValue 4
-MaximumValue 4 -FillColor "#F1C40F"
            }
        }
    }
}
```

That renders nicely:

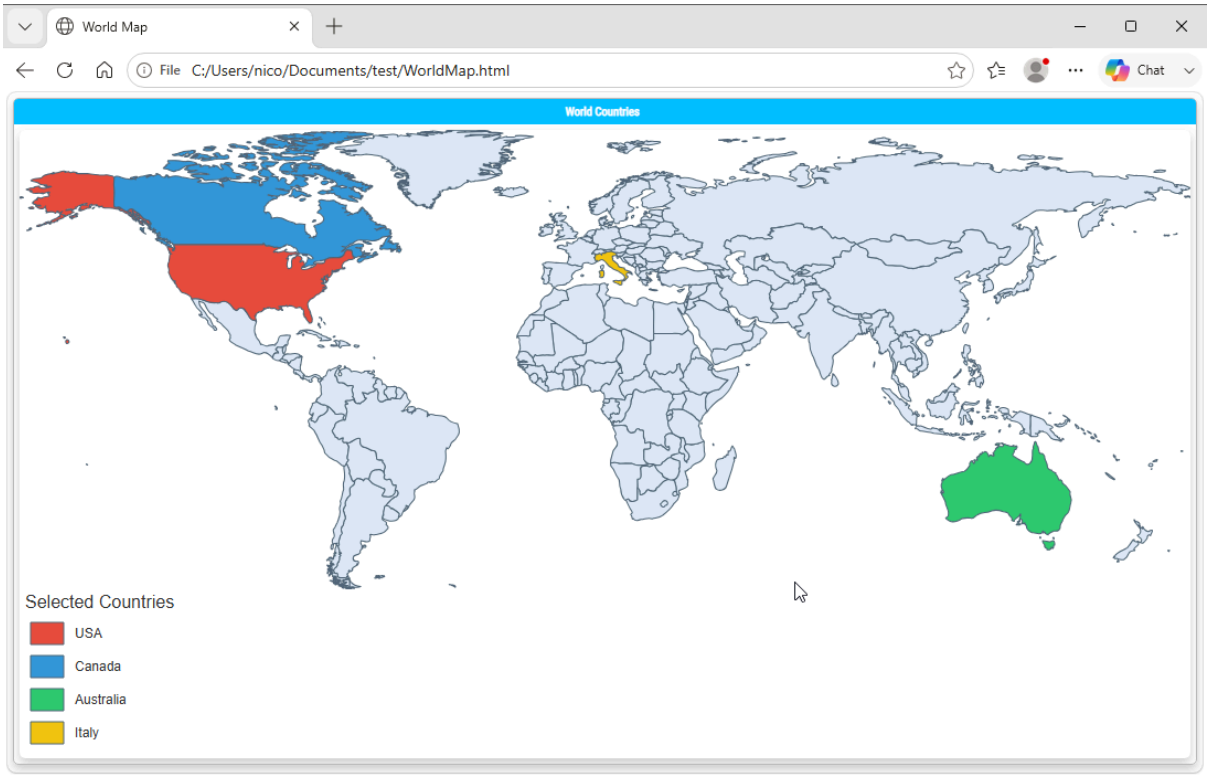

Figure 5.5. An interactive world map rendered with PSWriteHTML.

5.8 Chart Styling and Customization Options

New-HTMLChart and its nested chart builder commands expose extensive options to match corporate branding or dashboard requirements:

5.8.1 Color Schemes & Themes

- **-Color:** Pass a hex color to `New-ChartBar`, `New-ChartPie`, or another chart builder to customize a series or slice. If not specified, a default color will be used.
- **New-ChartTheme -Palette:** Select a pre-built color palette (e.g., `palette1` through `palette10`).

5.8.2 Sizing and Layout

- **-Height:** Set height explicitly in pixels (e.g., `-Height 400`).
- **-Width:** Set width explicitly in pixels (e.g., `-Width 600`). If omitted, the chart will expand to fill its container.
- **New-ChartLegend -LegendPosition:** Control where dataset legends appear (`top`, `bottom`, `left`, `right`).
- **New-ChartSpark:** Creates a minimalist inline chart without axes or legends, ideal for executive metric summary cards.

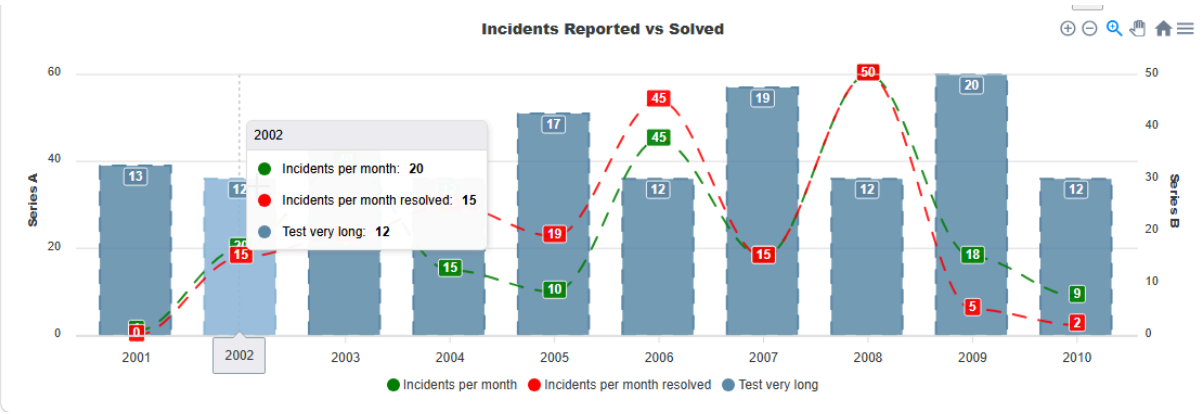

Figure 5.6. A combined chart view with multiple visualizations in one report.

You can also combine multiple charts into a single one.

5.9 Combining Charts and Tables

For comprehensive reporting, you can place charts and DataTables side-by-side or stacked inside the same section to provide both high-level visual summaries and actionable underlying records.

5.9.1 Full Operational Example: Service Distribution Report

```
# 1. Collect Local Service Data
$Services = Get-Service
$GroupedServices = $Services | Group-Object Status | Select-Object
@{N='Status';E={$_.Name}}, Count

# 2. Build Integrated Report
New-HTML -Title "Services Status & Breakdown" -FilePath
"$env:USERPROFILE\Desktop\ServicesReport.html" -Show {
    New-HTMLTab -Name "Service Metrics" {

        # Top Row: Chart Summary
        New-HTMLSection -HeaderText "Status Summary Chart" {
            New-HTMLPanel {
                New-HTMLChart -Title "Running vs Stopped Services" -Height 300 {
                    $GroupedServices | ForEach-Object {
                        $Color = if ($_.Status -eq 'Running') { '#4CAF50' } else {
'#F44336' }
                        New-ChartPie -Name $_.Status -Value $_.Count -Color $Color
                    }
                }
            }
        }

        # Bottom Row: Granular Data Grid
        New-HTMLSection -HeaderText "Detailed Service Records" {
            New-HTMLPanel {
                New-HTMLTable -DataTable ($Services | Select-Object Name,
DisplayName, Status, StartType -First 25) {
                    New-TableCondition -Name 'Status' -ComparisonType string
-Operator eq -Value 'Running' -BackgroundColor '#E8F5E9' -Color '#2E7D32'
                    New-TableCondition -Name 'Status' -ComparisonType string
-Operator eq -Value 'Stopped' -BackgroundColor '#FFEBEE' -Color '#C62828'
                }
```

```
                }
            }
        }
}
```

The result is a dashboard that combines a pie chart for service status with a detailed service status table below it:

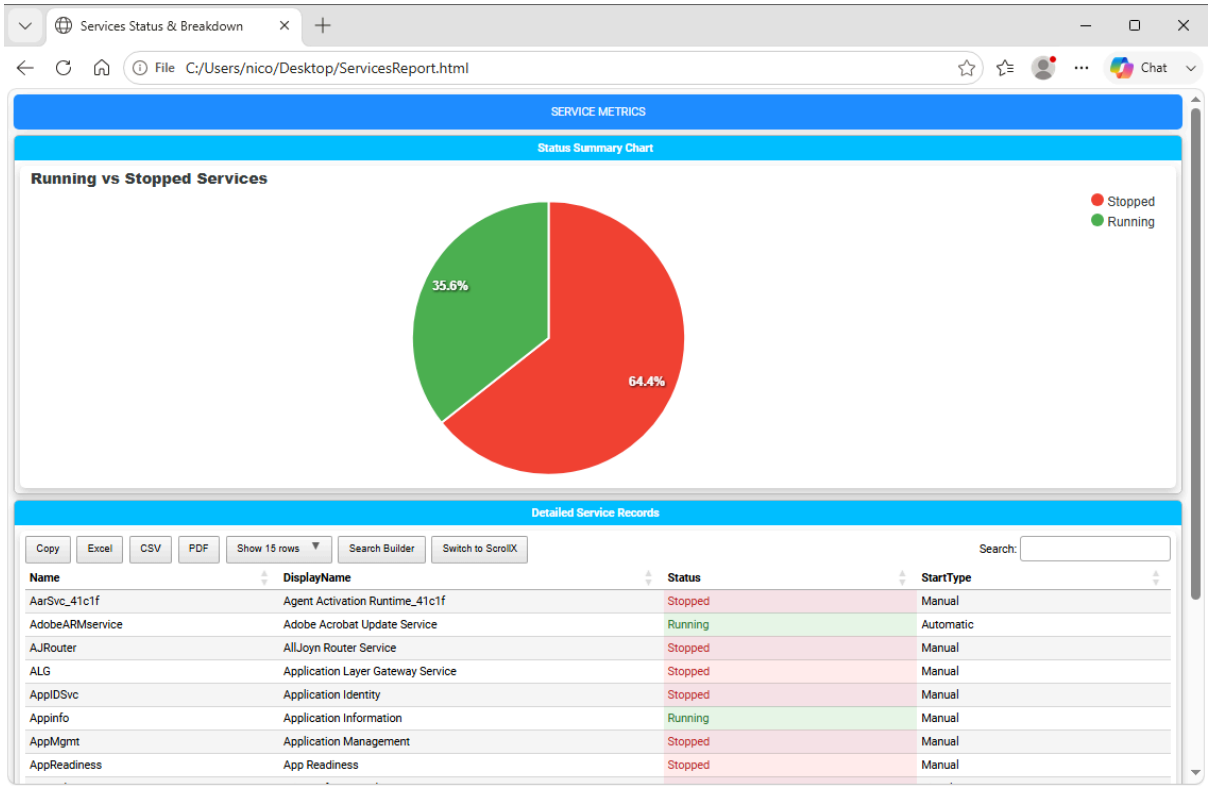

Figure 5.7. A service-status report combining a pie chart with a detailed table.

Note For an interactive example that correlates chart selections with table rows using `-DataTableID`, `-ID`, and `-ColumnID`, see Section 11.3.3 on Chapter 11.

5.10 Chart Best Practices

1. **Avoid Overcrowding Categories:** Pie and donut charts become unreadable with more than 7–8 slices. Group small items into an "Other" category prior to rendering.
2. **Explicit Colors for Statuses:** Consistently assign red/green/yellow hex values for status indicators (e.g., pass/fail or warning/critical states) rather than relying on random theme colors.
3. **Set Fixed Heights:** Specifying `-Height` ensures layout consistency across different desktop monitor resolutions.

Next Chapter: Chapter 6: Organizing Content with Tabs, Sections, and Panels

Chapter 6: Organizing Content with Tabs, Sections, and Panels

Table of Contents

6.1 Report Structure Overview

As reports grow in scope and complexity, structure becomes as vital as the data itself. **PSWriteHTML** provides a modular, responsive layout engine based on a hierarchical grid model.

By strategically combining **Tabs**, **Sections**, **Panels**, and **Columns**, you can transform long, overwhelming operational logs into structured, executive-ready dashboards.

We've already introduced the basic building blocks in Chapter 3, but this chapter delves deeper into advanced layout techniques, best practices for multi-tab navigation, and strategies for creating visually appealing, data-rich reports.

6.2 Layout Architecture Deep-Dive

The layout engine uses a nested container model that flows from top-level navigation down to individual content cards:

```
New-HTML
└── New-HTMLTab (Top Navigation)
    └── New-HTMLSection (Horizontal Row)
        └── New-HTMLPanel (Card Container)
            └── Content Widgets (Tables, Charts, Text, Diagrams)
```

6.2.1 Component Responsibilities

- **New-HTMLTab:** Acts as the primary top-level menu item. Each tab hides or shows full logical sub-pages without reloading the browser.
- **New-HTMLSection:** Functions as a full-width horizontal row or banner within a tab. Sections isolate distinct reporting topics (e.g., "CPU Metrics" vs "Memory Usage").
- **New-HTMLPanel:** Acts as a floating card or column block within a section. Panels wrap specific tables, charts, or text blocks with clean borders and background padding.

6.3 Mastering Tabs (`New-HTMLTab`)

New-HTMLTab allow you to pack dozens of metrics into a single standalone HTML document while keeping the viewport clutter-free.

6.3.1 Tab Customization Options

- **-Name:** Text displayed on the navigation button.
- **-IconSolid / ``-IconRegular`` / ``-IconBrands``:** Adds a FontAwesome icon to the tab title (e.g., `-IconSolid 'server'` or `-IconBrands 'windows'`).
- **-IconColor:** Hex or color text for the FontAwesome icon (e.g., `-IconColor 'Red'` or `-IconColor '#FF0000'`).

6.3.2 Tab Icon Example

```
New-HTML -Title "Infrastructure Status" -FilePath "MultiTab.html" -Show {

    # Tab 1: Server Health with FontAwesome Icon
    New-HTMLTab -Name "Servers" -IconSolid "server" {
        New-HTMLSection -HeaderText "Active Nodes" {
            New-HTMLPanel {
                New-HTMLText -Text "Server details go here."
            }
        }
    }

    # Tab 2: Security Alerts with a Red Warning Badge
    New-HTMLTab -Name "Alerts" -IconSolid "exclamation-triangle" -IconColor "Red"
{
        New-HTMLSection -HeaderText "Unresolved Incident Reports" {
            New-HTMLPanel {
                New-HTMLText -Text "Security incident details go here."
            }
        }
    }
}
```

This is the result:

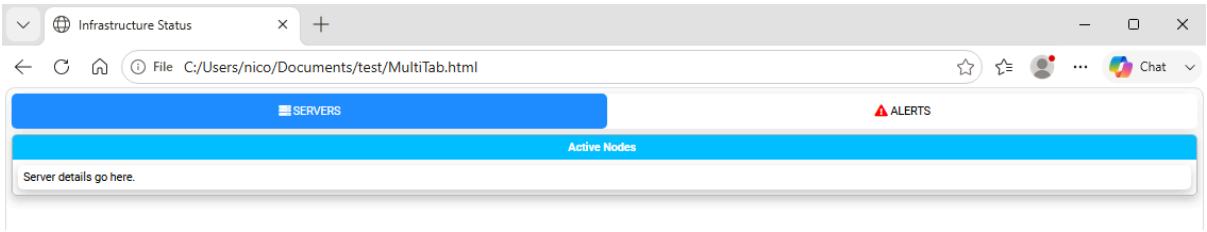

Figure 6.1. A report organized with tabbed navigation.

6.4 Section & Column Grid Layouts (`New-HTMLPanel` and `New-HTMLSection`)

By default, every New-HTMLPanel added inside a New-HTMLSection expands to fill the available

width. When multiple panels are placed within the same section, PSWriteHTML automatically arranges them into multi-column side-by-side card grids.

6.4.1 Creating Multi-Column Layouts

To create a 2-column or 3-column dashboard row, place multiple New-HTMLPanel blocks inside a single New-HTMLSection .

```
New-HTML -Show {
    New-HTMLSection -HeaderText "System Resource Utilization" {

        # Left Column (Panel 1)
        New-HTMLPanel {
            New-HTMLText -Text "CPU Load History Chart"
        }

        # Middle Column (Panel 2)
        New-HTMLPanel {
            New-HTMLText -Text "RAM Consumption Chart"
        }

        # Right Column (Panel 3)
        New-HTMLPanel {
            New-HTMLText -Text "Storage Allocation Summary"
        }
    }
}
```

This is the result:

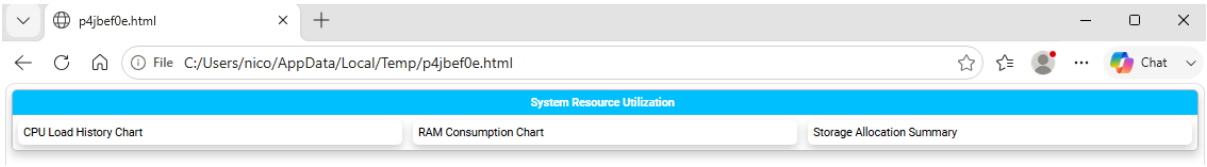

Figure 6.2. Three panels arranged in a responsive multi-column section.

Three panels placed inside a single section, automatically flowing into an aligned multi-column grid.

6.4.2 Collapsible Sections

For detailed logs or secondary diagnostic details that do not need to be visible immediately, make a section collapsible with the switch `-CanCollapse` and start it collapsed with the related switch `-Collapsed`:

```
New-HTMLSection -HeaderText "Advanced Active Directory Diagnostics" -CanCollapse
-Collapsed {
    New-HTMLPanel {
        New-HTMLText -Text "Detailed diagnostic logs and raw output parameters..."
    }
}
```

6.4.3 Panel Styling & Backgrounds

You can highlight critical information by applying custom background shades or text styles directly to panels:

- **-BackgroundColor:** Hex code or CSS color name for the panel background.
- **-HeaderText on New-HTMLSection:** Adds a header above the section, which can also be made collapsible.

6.5 Complete Operational Dashboard Example

The following script combines multi-tab navigation, FontAwesome icons, status badges, multi-column sections, and a collapsible diagnostic section:

```
    # 1. Mock Health Check Data
    $Nodes  = Get-CimInstance Win32_OperatingSystem | Select-Object CSName,
Caption, OSArchitecture, Version
    $Disks  = Get-CimInstance Win32_LogicalDisk | Select-Object DeviceID,
@{N='FreeGB'; E={[math]::Round($_.FreeSpace/1GB, 2)}}

    # 2. Build Dashboard
    New-HTML -Title "Enterprise Health Monitor" -FilePath
"$env:USERPROFILE\Desktop\HealthMonitor.html" -Show {

        # Tab 1: Executive Overview
        New-HTMLTab -Name "Overview" -IconSolid "chart-line" {

            # Top Row: Side-by-Side Cards
            New-HTMLSection -HeaderText "Environment Snapshot" {
                New-HTMLPanel {
                    New-HTMLTable -DataTable $Nodes -DisablePaging -DisableSearch
                }

                New-HTMLPanel {
                    New-HTMLTable -DataTable $Disks -DisablePaging -DisableSearch
                }
            }

            # Bottom Row: Collapsible Raw Logs
            New-HTMLSection -HeaderText "Raw WMI Event Logs" -CanCollapse
-Collapsed {
                New-HTMLPanel {
                    New-HTMLText -Text "Verbose event trace logs captured during
system boot sequence."
                }
            }
        }

        # Tab 2: Security & Compliance
        New-HTMLTab -Name "Compliance" -IconSolid "shield-alt" {
            New-HTMLSection -HeaderText "Audit Trail" {
                New-HTMLPanel {
                    New-HTMLText -Text "Status: OK. All local security policies
match baseline standards."
                }
            }
        }
    }
```

This is the result:

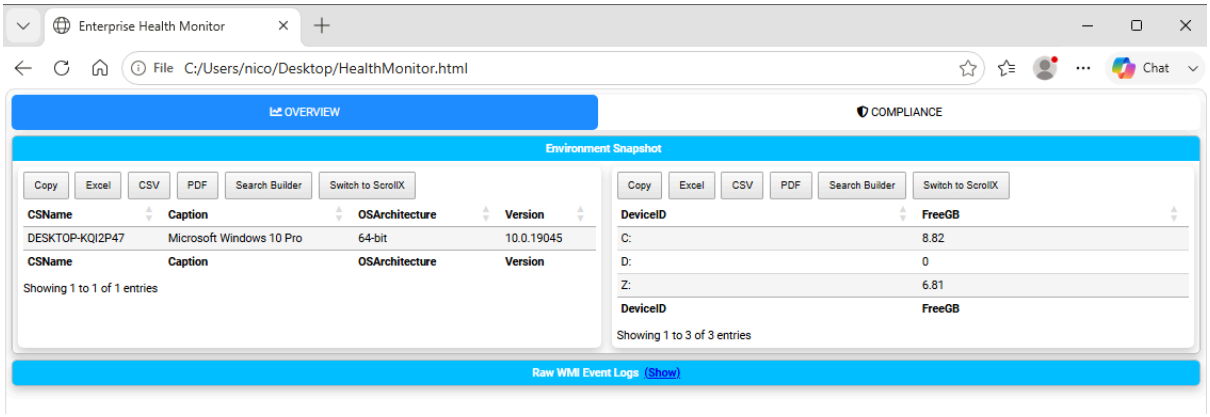

Figure 6.3. A complete operational dashboard with tabs, panels, icons, and collapsible content.

6.6 Layout Design Rules

1. **Keep Related Metrics Grouped:** Use a single `New-HTMLSection` for metrics that need side-by-side visual comparison (e.g., Disk Space and Disk I/O).
2. **Limit Primary Tabs:** Limit top-level tabs to 5–7 items to prevent tab-bar wrapping on lower-resolution screens.

Next Chapter: Chapter 7: Network Diagrams, Flowcharts, and Visualizations

Chapter 7: Network Diagrams, Flowcharts, and Visualizations

Table of Contents

7.1 Complex visualizations overview

Beyond tables and charts, complex IT environments frequently require structural visualization—such as Active Directory trust maps, network topology diagrams, cloud architecture maps, and organizational hierarchies.

PSWriteHTML includes native support for interactive diagramming powered by Vis Network through the `New-HTMLDiagram` cmdlet. You can build nodes, connections, flowcharts, and relational graphs dynamically using pure PowerShell syntax without needing external design tools like Visio or Lucidchart.

7.2 Core Diagram Concepts: Nodes and links (`New-HTMLDiagram`)

Diagrams in PSWriteHTML are constructed using two primary structural primitives inside a New-HTMLDiagram container:

1. **Nodes (New-DiagramNode):** Visual entities (e.g., servers, routers, switches, user accounts, cloud databases).
2. **Links (New-DiagramLink):** Directional or non-directional connection lines connecting two nodes (e.g., network cables, trust relationships, data flows).

Note Links in some outdated diagramming libraries are called "edges" or "connections." In PSWriteHTML, we use the term "links" to align with common networking terminology. `New-DiagramLink` has many aliases: `DiagramLink`, `DiagramEdge`, `DiagramEdges` and `New-DiagramEdge` for backward compatibility.

7.2.1 Basic Diagram Syntax

```
New-HTML -Title "Basic Topology Map" -FilePath "Topology.html" -Show {
```

```
    New-HTMLTab -Name "Network Map" {
        New-HTMLSection -HeaderText "Local Network Topology" {
            New-HTMLPanel {
                New-HTMLDiagram -Diagram {

                    # Define Nodes
                    New-DiagramNode -Id 1 -Label "Core Router" -Shape "box"
                    New-DiagramNode -Id 2 -Label "Switch-01"   -Shape "box"
                    New-DiagramNode -Id 3 -Label "Server-DB"   -Shape "ellipse"

                    # Define Connections (Links)
                    New-DiagramLink -From 1 -To 2 -Label "10Gbps Trunk"
                    New-DiagramLink -From 2 -To 3 -Label "1Gbps Link"
                } -Height "500px"
            }
        }
    }
}
```

This is the result, a simple network diagram with three nodes and two links, fully interactive with zooming, panning, and node dragging:

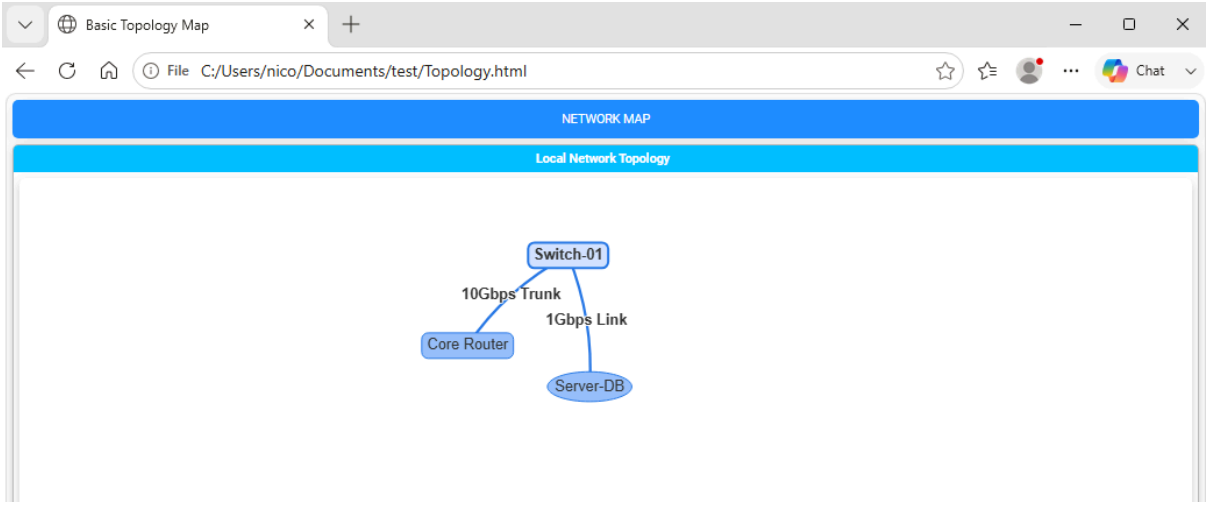

Figure 7.1. A basic interactive diagram containing nodes and links.

7.3 Customizing Nodes (`New-DiagramNode`)

Nodes are defined with New-DiagramNode . They can be customized with specific shapes, background colors, font styles, and FontAwesome icons to represent specific hardware or service roles.

7.3.1 Node Parameters & Options

- **-Id:** Unique identifier (integer or string) used to reference the node in link definitions. If not set, label will be used as Id.
- **-Label:** Display text shown on or below the node.
- **-Shape:** Geometry of the node (`circle`, `dot`, `diamond`, `ellipse`, `database`, `box`, `square`, `triangle`, `triangleDown`, `text`, `star`, `hexagon`).
- **-ColorBackground:** Hex color code for the internal node fill.
- **-ColorBorder:** Hex color code for the node outline.

- **-IconSolid / -IconBrands:** Embeds a FontAwesome vector icon inside the node. Cannot be used with `-Shape` values like `box` or `ellipse`.

7.4 Customizing Connections (`New-DiagramLink`)

New-DiagramLink control how relationships between nodes are visualized, supporting directional arrows, line styling, color coding, and labels.

7.4.1 Link Customization Parameters

- **-From / -To:** IDs of the source and target nodes.
- **-ArrowsToEnabled:** Switch parameter to enable arrows at the end of the link.
- **-ArrowsToType:** Type of arrowhead to display at the end of the link. Valid values are 'arrow', 'bar', or 'circle'.
- **-Color:** Hex color code for the connection line.
- **-Dashed:** Boolean (`$true`/`$false`) to switch between solid and dashed lines (ideal for backup links or optional paths).
- **-Label:** Text annotation displayed along the link line.

7.4.2 Directional & Dashed Link Example

```
New-DiagramLink -From "Firewall" -To "DMZ-Host" -ArrowsToEnabled -Color "#E53935"
-Label "HTTPS (443)"
New-DiagramLink -From "DMZ-Host" -To "Backup-Srv" -ArrowsToEnabled -Dashed $true
-Color "#757575" -Label "Replication"
```

7.4.3 Node Icon & Color Example

A simple workflow diagram with three nodes and two directional links, fully interactive with customized icons, colors, and labels:

```
New-HTML -Title "Basic Topology Map" -FilePath "Topology.html" -Show {
    New-HTMLTab -Name "Network Map" {
        New-HTMLSection -HeaderText "Local Network Topology" {
            New-HTMLPanel {


                New-HTMLDiagram -EnableFiltering -EnableFilteringButton -Diagram {

                    New-DiagramOptionsInteraction -Hover $true

                    # Define Nodes
                    #New-DiagramNode -Id 1 -Label "Core Router"  -IconSolid
broadcast-tower

                    New-DiagramNode -Id "GW" -Label "Gateway Router"  -IconBrands
hubspot  -IconColor Blue
                    New-DiagramNode -Id "DB" -Label "SQL Cluster"  -Shape database
                    New-DiagramNode -Id "WEB" -Label "Web Server"  -IconSolid
server  -IconColor Blue

                    # Define Connections (Edges)
                    New-DiagramEdge -From "GW" -To "WEB" -ArrowsToEnabled
                    New-DiagramEdge -From "WEB" -To "DB" -Label "1Gbps"
-ArrowsToEnabled
```

```
                } -Height "500px"
            }
        }
    }
}
```

This is the result:

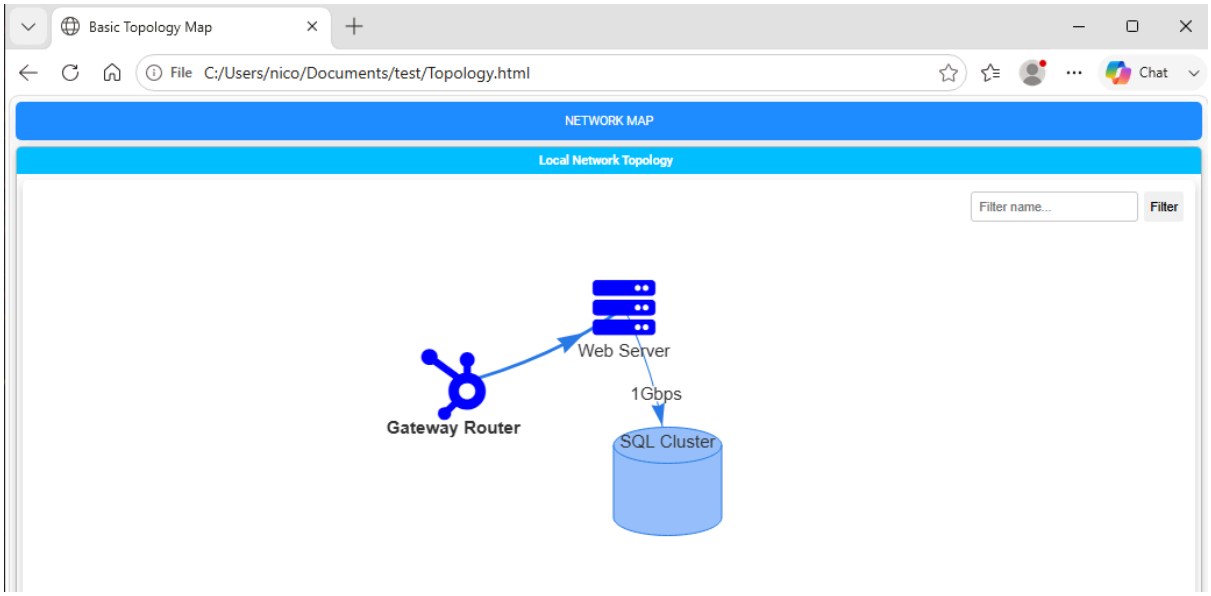

Figure 7.2. A directional network diagram with customized nodes and links.

7.5 Layout Hierarchies & Physics Engines (New-DiagramOptionsLayout)

Vis Network includes physics engines that automatically arrange nodes dynamically or snap them into rigid hierarchical trees.

7.5.1 Hierarchical Layouts (Org Charts & Tree Diagrams)

For structured trees like Active Directory domain trust structures or organizational charts, enable hierarchical positioning using New-DiagramOptionsLayout :

- **-HierarchicalEnabled:** Switch parameter to enable hierarchical layout.
- **-HierarchicalDirection:** The direction of the hierarchical layout. Valid values are: FromUpToDown, FromDownToUp, FromLeftToRight, FromRigthToLeft.

```
New-HTML -Title "AD Topology Map" -FilePath "ADTopology.html" -Show {
    New-HTMLTab -Name "AD Map" {
        New-HTMLSection -HeaderText "Local AD Topology" {
            New-HTMLPanel {
                New-HTMLDiagram -Diagram {

                # Layout Engine Configuration
                New-DiagramOptionsLayout -HierarchicalEnabled $true
-HierarchicalDirection FromUpToDown
```

```
                New-DiagramNode -Id 1 -Level 1 -Label "Forest Root Domain" -Shape
"box"
                New-DiagramNode -Id 2 -Level 2 -Label "Child Domain A"     -Shape
"box"
                New-DiagramNode -Id 3 -Level 2 -Label "Child Domain B"     -Shape
"box"

                New-DiagramLink -From 1 -To 2 -ArrowsToEnabled -ArrowsFromEnabled
-Label "Two-Way Trust"
                New-DiagramLink -From 1 -To 3 -ArrowsToEnabled -ArrowsFromEnabled
-Label "Two-Way Trust"
            } -Height "500px"
            }
        }
    }
}
```

This is the result:

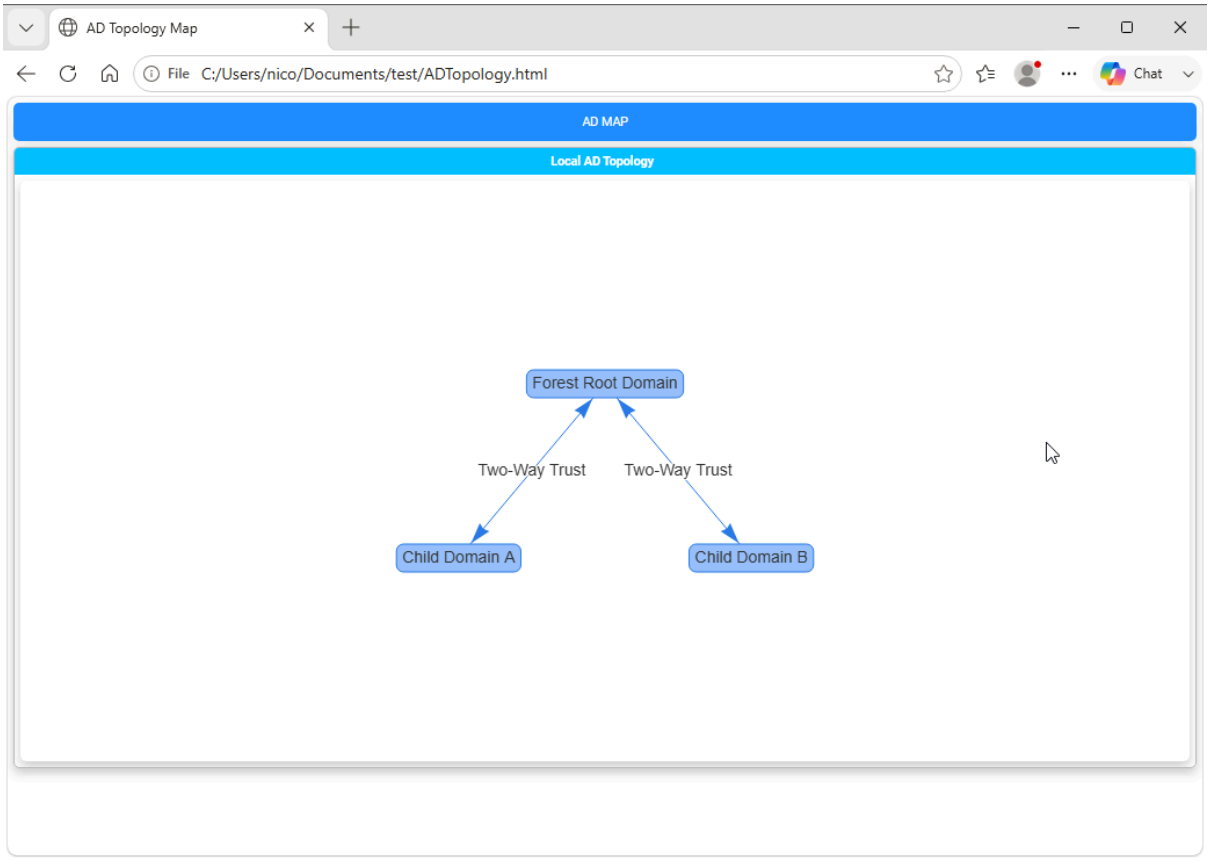

Figure 7.3. An Active Directory topology diagram using a hierarchical layout.

7.6 Complete Practical Example: Infrastructure Network Topology Map

The following script gathers network interface data and constructs a multi-tier network map complete with interactive node dragging, zooming, directional paths, and FontAwesome status indicators:

```
    New-HTML -Title "Data Center Network Diagram" -FilePath
```

```
 "$env:USERPROFILE\Desktop\NetworkMap.html" -Show {
       New-HTMLTab -Name "Topology" -IconSolid "network-wired" {
           New-HTMLSection -HeaderText "Production Infrastructure Mesh" {
               New-HTMLPanel {
                   New-HTMLDiagram -Diagram {

                       # 1. External & Perimeter Layer
                       New-DiagramNode -Id "INET" -Label "Internet / WAN" -Shape
"circle" -ColorBackground "#0288D1"
                       New-DiagramNode -Id "FW01" -Label "Perimeter Firewall"
-Shape "box" -ColorBackground "#D32F2F"

                       # 2. Distribution Layer
                       New-DiagramNode -Id "SW01" -Label "Core Switch 01" -Shape
"box" -ColorBackground "#1976D2"
                       New-DiagramNode -Id "SW02" -Label "Core Switch 02 (HA)"
-Shape "box" -ColorBackground "#1976D2"

                       # 3. Workload Layer
                       New-DiagramNode -Id "APP"  -Label "App Tier Cluster" -Shape
"ellipse" -ColorBackground "#388E3C"
                       New-DiagramNode -Id "DB"   -Label "SQL Availability Group"
-Shape "database" -ColorBackground "#F57C00"

                       # 4. Define Interconnects
                       New-DiagramLink -From "INET" -To "FW01" -ArrowsToEnabled
-Label "Inbound Traffic"
                       New-DiagramLink -From "FW01" -To "SW01" -ArrowsToEnabled
-Color "#D32F2F"
                       New-DiagramLink -From "FW01" -To "SW02" -ArrowsToEnabled
-Color "#D32F2F" -Dashes $true

                       # Core Switch Trunking
                       New-DiagramLink -From "SW01" -To "SW02" -ArrowsToEnabled
-ArrowsFromEnabled -Label "LACP Trunk" -Color "#1976D2"

                       # App & DB Connects
                       New-DiagramLink -From "SW01" -To "APP"  -ArrowsToEnabled
                       New-DiagramLink -From "SW02" -To "APP"  -ArrowsToEnabled
                       New-DiagramLink -From "APP"  -To "DB"   -ArrowsToEnabled
-Label "Port 1433"
                   } -Height "650px"
               }
           }
       }
   }
```

This is the result:

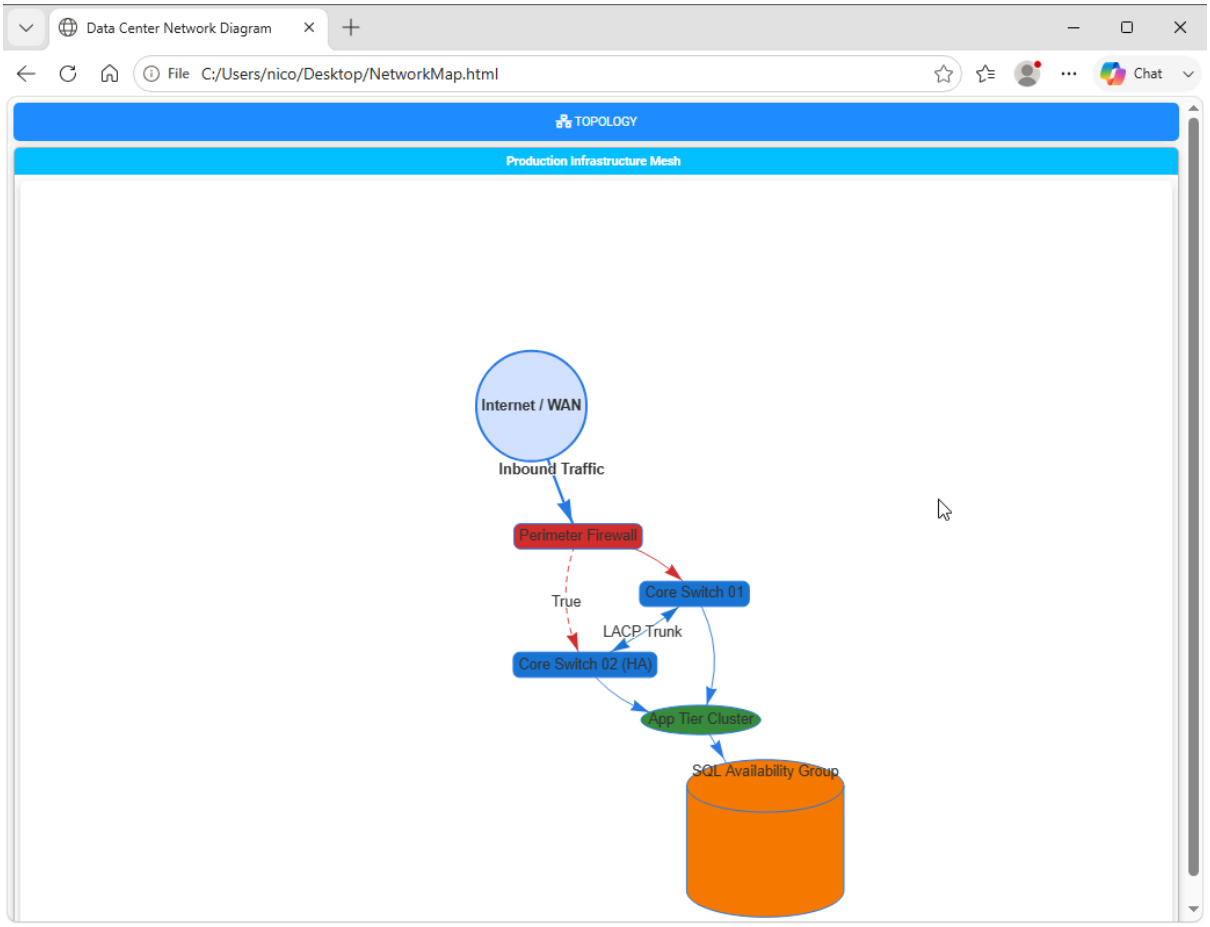

Figure 7.4. An infrastructure network topology map with directional paths and status icons.

7.7 Diagramming Best Practices

1. **Explicit Heights:** Always set a explicit `-Height` parameter (e.g., `"500px"` or `"700px"`) on `New-HTMLDiagram` to prevent canvas collapse inside panels.
2. **Unique Node IDs:** Ensure every node has a strictly unique `-Id` string or integer. Duplicate IDs cause link routing failures in Vis.js.
3. **Use Hierarchy for Trees:** When displaying parent-child structures (such as management org charts or DNS delegation trees), always enable `New-DiagramOptionsLayout -HierarchicalEnabled $true` to prevent nodes from drifting into floating physics clusters.

Next Chapter: Chapter 8: Custom Styling, CSS, and Themes

Chapter 8: Custom Styling, CSS, and Themes

Table of Contents

8.1 Styling Overview

While **PSWriteHTML** ships with clean, modern styling out of the box, enterprise environments often require matching corporate design standards, applying dark modes, or injecting custom typography and custom headers.

This chapter details how to inject custom CSS, apply custom typography, use FontAwesome icons, and override component styling directly using PowerShell DSL parameters.

8.2 Document-Level Theme Configuration

The root New-HTML cmdlet provides the document container for component-level styling and custom CSS. Header colors are configured on sections or through the corresponding style cmdlets.

8.2.1 Built-in Themes & Header Colors

You can alter section colors directly with supported New-HTMLSection parameters:

- **``-HeaderBackGroundColor``:** Sets the background color of a section header (Hex, RGB, or standard CSS color names).
- **``-HeaderTextColor``:** Controls the font color of a section header.
- **``-BackgroundColor``:** Sets the background color of a section (Hex, RGB, or standard CSS color names).

8.2.2 Basic Color Override Example

```
New-HTML -TitleText "Corporate Security Audit" -FilePath "Audit.html" -ShowHTML {

    New-HTMLTab -Name "Overview" {
        New-HTMLSection -HeaderText "Executive Summary" `
```

```
                        -HeaderBackGroundColor "#309135" `
                        -HeaderTextColor "#e01965" -BackgroundColor "lightyellow" {
            New-HTMLPanel {
                New-HTMLText -Color Amazon -Text "All perimeter checks passed
corporate baseline guidelines."
            }
        }
    }
}
```

8.3 Custom CSS Injection (`Add-HTMLStyle`)

For fine-grained visual control, PSWriteHTML uses the Add-HTMLStyle function allows you to add CSS styles to HTML documents in various ways such as inline styles, external stylesheets,and content from files or strings.

8.3.1 Injecting Raw CSS

Pass custom CSS rules to `Add-HTMLStyle -Placement Header -Content` to override fonts, table borders, panel padding, or background gradients across the entire page:

```
# Define custom CSS rules for the entire document
$CustomStyles = @"
    body {
        font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
        background-color: #f4f6f9;
    }
    .card {
        border-radius: 8px !important;
        box-shadow: 0 4px 6px rgba(0,0,0,0.1);
    }
    h1, h2, h3 {
        color: #1a202c;
    }
"@

New-HTML -TitleText "Custom Styled Report" -FilePath "CustomCSS.html" -Show {
    Add-HTMLStyle -Placement Header -Content $CustomStyles

    New-HTMLTab -Name "Dashboard" {
        New-HTMLSection -HeaderText "Custom UI Styling" {
            New-HTMLPanel {
                New-HTMLText -Text "This panel features custom rounded corners and
subtle dropshadows."
            }
        }
    }
}
```

8.3.2 Component-Level Styles

Use `Add-HTMLStyle` **inside** the script block passed to New-HTML to add CSS to the generated document:

```
New-HTML -TitleText "Scoped Styling" -FilePath "ScopedStyle.html" -Show {

    # Inject scoped CSS rule inside HTML head
    Add-HTMLStyle -Placement Header -Content @'
        .custom-highlight {
```

```
            background-color: #fff3cd;
            border-left: 5px solid #ffc107;
            padding: 15px;
            font-weight: bold;
        }
'@

    New-HTMLTab -Name "Audit" {
        New-HTMLSection -HeaderText "Notifications" {
            New-HTMLPanel {
                New-HTMLText -Text "<div class='custom-highlight'>Warning: 2
backup targets are approaching capacity.</div>"
            }
        }
    }
}
```

8.4 Icon Integration & Typography (`New-HTMLFontIcon`)

PSWriteHTML natively integrates FontAwesome icons in supported components, including tabs. Other layout elements can include an icon with `New-HTMLFontIcon` where appropriate. FontAwesome provides icons, not a general-purpose typography provider; custom fonts should be configured with CSS or component font parameters.

8.4.1 Using FontAwesome Icons

Supported components expose dedicated icon parameters. The icon names below link to the corresponding Font Awesome icon gallery so you can browse and preview the available icons before using them:

- **``-IconSolid``:** Solid-style icons (e.g., `"server"`, `"shield-alt"`, `"database"`).
- **``-IconRegular``:** Outline-style icons (e.g., `"file-alt"`, `"clock"`).
- **``-IconBrands``:** Brand logos (e.g., `"windows"`, `"linux"`, `"aws"`, `"github"`).

8.4.2 Inline Icon Example

```
New-HTMLTab -Name "Cloud Infrastructure" -IconBrands "aws" {
    New-HTMLSection -HeaderText "EC2 Instances" {
        New-HTMLPanel {
            New-HTMLFontIcon -IconSolid "server"
            New-HTMLText -Text "Region: us-east-1"
            New-HTMLText -Text "Active instance nodes in availability zone A."
        }
    }
}
```

8.5 Inserting a company logo

To place a company logo at the far left of a tab label or navigation item, add a scoped CSS rule to the document and use the logo URL as the `background-image` of a `::before` pseudo-element. This keeps the logo aligned with the label while preserving any existing icon or text. The complete example below demonstrates this technique with an external image URL.

8.6 Building Dark Mode & Corporate Theme Templates

To enforce consistent corporate branding across all enterprise reports, wrap your standard `New-HTML` configuration into reusable PowerShell helper functions or script execution templates.

8.6.1 Enterprise Dark Theme Template

The following complete script demonstrates how to combine custom CSS, section header colors, and FontAwesome icons into a cohesive dark-mode dashboard:

```
# 1. Define Corporate Dark Palette CSS
$DarkThemeCSS = @"
    body {
        background-color: #121212 !important;
        color: #e0e0e0 !important;
    }
    .card, .panel {
        background-color: #1e1e1e !important;
        border: 1px solid #333333 !important;
        color: #ffffff !important;
    }
    .table {
        color: #e0e0e0 !important;
        background-color: #1e1e1e !important;
    }
    .table-striped tbody tr:nth-of-type(odd) {
        background-color: #252525 !important;
    }
    .tabsWrapper > .tabsSlimmer > div[data-tabs="true"] > div:first-child::before
{
        content: "";
        display: inline-block;
        width: 60px;
        height: 60px;
        margin-right: 8px;
        background-image: url("https://avatars.githubusercontent.com/u
/15376314?s=60&v=4");
        background-size: cover;
        background-position: center;
        vertical-align: middle;
    }
"@

# 2. Collect System Sample Data
$Data = Get-Process | Select-Object -First 10 ID, ProcessName, CPU, WorkingSet

# 3. Generate Dark Mode Report
New-HTML -TitleText "Dark Mode Enterprise Report" `
        -FilePath "$env:USERPROFILE\Desktop\DarkReport.html" -Show {

    Add-HTMLStyle -Placement Header -Content $DarkThemeCSS

    New-HTMLTab -Name "SYSTEM METRICS" -IconSolid "chart-bar" {
        New-HTMLSection -HeaderText "Resource Monitoring" `
                        -HeaderBackGroundColor "#000000" `
                        -HeaderTextColor "#00E676" {
            New-HTMLPanel {
                New-HTMLText -Text "Top Processes"
                New-HTMLTable -DataTable $Data -Filtering
            }
```

```
            }
        }
}
```

This is the result:

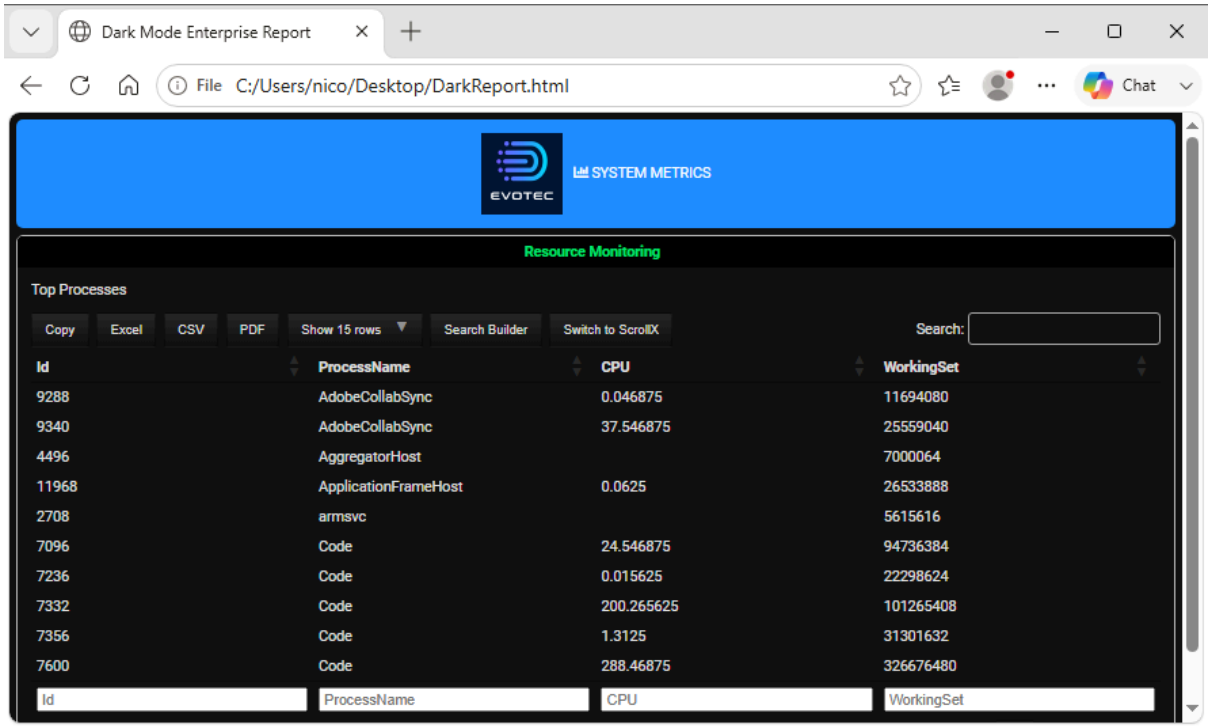

Figure 8.1. A dark-themed PSWriteHTML dashboard with custom colors and styles.

8.7 Styling Best Practices

1. **Use ``!important`` flags for CSS Overrides:** PSWriteHTML loads third-party framework CSS (Bootstrap/DataTables). When writing custom CSS to override default component colors, add `!important` to ensure your rules take precedence.
2. **Prefer Hex Codes over Named Colors:** Use explicit 6-digit hex color codes (e.g., `#1A365D`) rather than named colors (`blue`, `navy`) to guarantee uniform rendering across different web browsers.
3. **Keep Inline HTML Minimal:** Use PowerShell DSL parameters (like `New-HTMLTableCondition` or section `-HeaderBackGroundColor`) whenever possible rather than embedding raw HTML tags directly inside string parameters.

Next Chapter: Chapter 9: Automated Reporting and Email Integration

Chapter 9: Automated Reporting and Email Integration

Table of Contents

9.1 Automation Overview

Beyond producing interactive browser-based dashboards saved as local *.html* files, **PSWriteHTML** includes specialized functionality for automated email reporting. Sending rich HTML emails via PowerShell often presents challenges because major mail clients (especially desktop Microsoft Outlook) use restrictive rendering engines that strips external stylesheets, CSS grids, and JavaScript.

PSWriteHTML solves this by providing dedicated email layout cmdlets (`Email`, `EmailBody`, `EmailLayoutRow`, `EmailLayoutColumn`) that automatically inline CSS styles into raw table-based HTML compatible with legacy email clients.

9.2 HTML Email Architecture vs. Web Dashboards

When building content for email delivery, standard web layouts (like `New-HTML`, JavaScript-based DataTables, and ApexCharts) should not be used directly because email clients block external scripts and dynamic DOM manipulation for security reasons.

9.2.1 Key Email Constraints & Differences

Feature	Web Dashboard (`New-HTML`)	Email Report (`Email`)
Primary Container	`New-HTML`	`Email / EmailBody`
Layout Model	Responsive CSS Flex/Grid Panels	Inlined Table-based Rows & Columns
JavaScript Support	Full (DataTables, ApexCharts)	None (Blocked by mail clients)
CSS Handling	Linked / Embedded Head Styles	Direct Inline CSS (`style="..."`)
Image Delivery	Web URLs or local relative paths	Inline images or CID (Content-ID) MIME attachments

9.3 Building HTML Emails (`Email` / `EmailBody`)

To construct and send an email, use the Email cmdlet with an `-Email` script block. Use EmailBody together with `EmailLayout`, `EmailLayoutRow`, and `EmailLayoutColumn` to build the table-based body. These commands apply email-safe layout and styling directly to the generated HTML.

9.3.1 Basic Email Template Structure

```
# Generate and send an email with an inlined HTML body
Email -To 'infrastructure@example.com' -From 'reports@example.com' `
    -Subject 'Morning backup status' -Server 'smtp.example.com' -Port 587 -SSL
-Email {
    EmailBody {
        EmailLayout {
            EmailLayoutRow {
        EmailLayoutRow {
            EmailLayoutColumn {
                    New-HTMLText -Text "Below is the automated morning backup
status summary."
                }
            }
        EmailLayoutRow {
            EmailLayoutColumn {
                    New-HTMLTable -DataTable $BackupResults -DisablePaging
                }
            }
        EmailLayoutRow {
            EmailLayoutColumn {
}
```

Note EmailLayoutRow and EmailLayoutColumn replace dashboard sections and panels for this purpose. They ensure email clients render the multi-column layout using native HTML `<table>` rows and cells.

9.4 Embedding Images in Email Reports (`New-HTMLImage`)

External image links (e.g., `<img src="https://...">`) may be blocked by default in Microsoft Outlook and Apple Mail until the recipient clicks "Download Images". To ensure logos and status icons display immediately, embed images directly using New-HTMLImage . For a self-contained HTML body, use `New-HTMLImage -Inline`. This embeds the image data in the generated markup and avoids relying on external image URLs. CID inline images are delivery-library-specific. The mail client library must add the image as an inline MIME attachment, assign it a unique Content-ID such as `CompanyLogo`, and include that same identifier in the HTML as `src="cid:CompanyLogo"`. A `cid:` reference by itself is only a pointer; without the matching MIME attachment and Content-ID, the recipient's mail client cannot load the image.

```
Email -Email {
    EmailBody {
        EmailLayout {
            EmailLayoutRow {
                EmailLayoutColumn {
                    New-HTMLImage -Source 'C:\Reports\CompanyLogo.png' -Inline
-AlternativeText 'Company logo' -Width 150
                    New-HTMLText -Text '<h2>Daily Audit Summary</h2>'
                }
```

```
            }
        }
    }
}
```

9.5 Sending the Email Report via SMTP or Microsoft Graph

The `Email` cmdlet can deliver the generated body directly. If another delivery library is required, use its HTML-body option and pass the output from `Email -OutputHTML` to that library.

9.5.1 Method 1: Native `Send-MailMessage` (Legacy SMTP)

```
# The built-in PowerShell cmdlet Send-MailMessage is obsolete.
# Keep this only for existing legacy SMTP scripts.
Send-MailMessage -To 'admin@company.com' -From 'reports@company.com' `
    -Subject "Daily System Health Check - $(Get-Date -Format 'yyyy-MM-dd')" `
    -Body $EmailBody -BodyAsHtml -SmtpServer 'smtp.company.com' -Port 25
```

9.5.2 Method 2: Microsoft Graph API (Modern Cloud Delivery)

For Office 365 environments where basic SMTP authentication is disabled, send the generated HTML email via Microsoft Graph PowerShell SDK:

```
# Requires Microsoft.Graph.Mail module
Import-Module Microsoft.Graph.Mail

$Message = @{
    Subject = "Automated Storage Alert"
    Body = @{
        ContentType = "Html"
        Content     = $EmailBody
    }
    ToRecipients = @(
        @{ EmailAddress = @{ Address = "admin@company.com" } }
    )
}

Send-MgUserMail -UserId "reports@company.com" -Message $Message
```

9.6 Automating Generation via Windows Scheduled Tasks

To run reports automatically on a daily or weekly schedule, wrap your PowerShell data gathering and PSWriteHTML script into an automated Scheduled Task.

9.6.1 Creating the Scheduled Task via PowerShell

The following administrative script registers a daily task that executes an automated PSWriteHTML report at 06:00 AM every morning:

```
# Define Script Path
$ScriptPath = "C:\Automation\Scripts\Generate-DailyReport.ps1"

# Create Action: Execute PowerShell 7 / Windows PowerShell silently
$Action = New-ScheduledTaskAction -Execute "pwsh.exe" -Argument "-ExecutionPolicy
Bypass -NoProfile -File `"$ScriptPath`""
```

```
# Create Trigger: Daily at 06:00 AM
$Trigger = New-ScheduledTaskTrigger -Daily -At "06:00 AM"

# Register Task under SYSTEM or Dedicated Service Account
Register-ScheduledTask -TaskName "PSWriteHTML_DailyReport" `
                       -Action $Action `
                       -Trigger $Trigger `
                       -User "NT AUTHORITY\SYSTEM" `
                       -RunLevel Highest
```

9.7 Complete End-to-End Operational Example

The following complete script collects disk space usage, builds a highlighted inline HTML table, generates an email body, and dispatches an automated alert email if any drive drops below critical thresholds:

```
# 1. Collect Volume Data
$DiskDrives = Get-CimInstance Win32_LogicalDisk |
    Where-Object DriveType -eq 3 |
    Select-Object DeviceID,
                  @{N='SizeGB'; E={[math]::Round($_.Size/1GB, 2)}},
                  @{N='FreeGB'; E={[math]::Round($_.FreeSpace/1GB, 2)}},
                  @{N='FreePercent'; E={[math]::Round(($_.FreeSpace
/$_.Size)*100, 1)}}

# 2. Select drives that require an alert
$LowDiskDrives = $DiskDrives | Where-Object FreePercent -lt 15

if ($LowDiskDrives) {

# 3. Build and send the inlined HTML email body
Email -To 'sysadmin@company.com' -From 'alerts@company.com' `
    -Subject "STORAGE WARNING: $env:COMPUTERNAME Low Disk Space Alert" -Server
'smtp.company.com' -Email {
    EmailBody {

        # Banner Header
        EmailLayout {
            EmailLayoutRow {
                EmailLayoutColumn {
                New-HTMLText -Text "<h2 style='margin:0;'>Critical Storage Status
Alert</h2>"
                New-HTMLText -Text "System evaluation completed on
<b>$env:COMPUTERNAME</b>."
                }
            }
        }

        # Data Grid with Conditional Formatting
        EmailLayoutRow {
            EmailLayoutColumn {
                New-HTMLTable -DataTable $LowDiskDrives -DisablePaging {

                    # Column Formatting
                    New-TableColumnOption -ColumnIndex 0 -Width '20%'
                    New-TableColumnOption -ColumnIndex 3 -Width '20%'

                    # Highlight low disk space in red
                    New-TableCondition -Name 'FreePercent' -ComparisonType number
```

```
 -Operator lt -Value 15 -BackgroundColor '#FFCDD2' -Color '#B71C1C'
                }
            }
        }

        # Footer Section
        EmailLayoutRow {
            EmailLayoutColumn {
                New-HTMLText -Text "<p style='font-size: 11px; color:
#777;'>Automated message generated by PSWriteHTML Reporting Engine.</p>"
            }
        }
    }
}

# 3. Deliver Email via SMTP
$SmtpParams = @{
    To          = 'sysadmin@company.com'
    From        = 'alerts@company.com'
    Subject     = "STORAGE WARNING: $env:COMPUTERNAME Low Disk Space Alert"
    Body        = $EmailContent
    BodyAsHtml = $true
    SmtpServer = 'smtp.company.com'
}

Send-MailMessage @SmtpParams
```

9.8 Email Automation Best Practices

1. **Use ``Email`` for Emails, ``New-HTML`` for Web Pages:** Never use standard `New-HTML` directly inside an email body. Desktop mail clients like Outlook will strip non-inlined CSS and script blocks.
2. **Disable DataTables Interactivity in Emails:** Always set `-Paging $false` and `-Filtering $false` on `New-HTMLTable` inside email blocks, as email clients cannot run the JavaScript engine required for search and paging.
3. **Specify Explicit Widths:** Use explicit table and column percentage widths (e.g., `width="100%"`) to prevent email templates from rendering distorted on mobile email clients.

Next Chapter: Chapter 10: Advanced Features, Tips, and Troubleshooting

Chapter 10: Publishing Live Dashboards on Windows

Table of Contents

10.1 Publishing Overview

As you scale **PSWriteHTML** across enterprise infrastructure, you will likely want to transition from generating static, on-demand *.html* files to publishing live, continuously updated dashboards accessible across your organization.

Because PSWriteHTML generates standalone HTML files bundled with JavaScript and CSS, publishing a live interactive dashboard does not require complex web application frameworks like ASP.NET or Node.js.

The standard architectural pattern consists of a **scheduled PowerShell task** that periodically writes an updated *index.html* file to a directory served by a web server.

10.2 Method 1: Hosting via Windows IIS (Internet Information Services)

Windows IIS provides a robust, built-in option for hosting live PSWriteHTML dashboards internally with Active Directory Windows Authentication support.

1. **Install IIS Web Server:**
   Run PowerShell as Administrator:

```
Install-WindowsFeature -Name Web-Server -IncludeManagementTools
```

2. **Configure Web Root Directory:**
   Create a dedicated folder to host your live report (e.g., `C:\inetpub\wwwroot\Dashboards`).
3. **Configure Scheduled Auto-Refresh Pipeline:**
   Configure your PowerShell generation script to output directly to the IIS web directory and set the document to auto-refresh in the browser using custom header tags:

```
# Target IIS Web Root
$OutputPath = "C:\inetpub\wwwroot\Dashboards\index.html"

# Auto-Refresh Meta Header (Refreshes browser every 300 seconds / 5 minutes)
```

```
$CustomHeader = '<meta http-equiv="refresh" content="300">'

New-HTML -Title "Live Infrastructure Monitor" -FilePath $OutputPath
-CustomJavaScript $CustomHeader {
    New-HTMLTab -Name "Live Metrics" -IconSolid "heartbeat" {
        New-HTMLSection -HeaderText "Last Updated: $(Get-Date -Format
'yyyy-MM-dd HH:mm:ss')" {
            New-HTMLPanel {
                New-HTMLTable -DataTable (Get-Service | Select-Object Name,
DisplayName, Status -First 15)
            }
        }
    }
}
```

4. **Automate Update Frequency:**
   Register a Windows Scheduled Task to execute this generation script every 5 to 15 minutes. Users browsing to `http://your-server-name/Dashboards/` will always view the latest live snapshot.

10.3 Method 2: Publishing to Cloud Static Web Hosting

For cloud-managed environments, you can automatically upload the generated HTML dashboard to a static web container (such as Azure Static Web Apps or AWS S3) using native PowerShell cloud modules:

```
# Generate Report Locally
$LocalReport = "$env:TEMP\index.html"
New-HTML -Title "Cloud Infrastructure Health" -FilePath $LocalReport { ... }

# Upload to Azure Storage Blob ($web Static Website Container)
# Requires Az.Storage module
Set-AzStorageBlobContent -Container '$web' -File $LocalReport -Blob "index.html"
-Context $StorageContext -Force
```

10.4 Method 3: Lightweight Hosting with Pode Framework

If you want to serve live dashboards on a Windows machine without enabling full IIS features, you can use the lightweight open-source **Pode** PowerShell web framework (see https://badgerati.github.io/Pode/latest/) to serve your PSWriteHTML reports on demand:

```
Import-Module Pode

Start-PodeServer {
    Add-PodeEndpoint -Address localhost -Port 8080 -Protocol Http

    # Serve static PSWriteHTML directory
    Use-PodeStatic -Path "C:\Reports\LiveDashboard"
}
```

10.5 Publishing Best Practices

1. **Grant Service Account File Permissions:** Ensure the service account running your scheduled generation script has explicit Modify/Write permissions to `C:\inetpub\wwwroot\`.

2. **Understand Offline Resource Embedding:** When publishing in air-gapped networks, always use `New-HTML -Offline`. PSWriteHTML embeds and base64-encodes all required JavaScript libraries, fonts, and CSS directly inside the single `index.html` file. No secondary assets or subfolders are needed on the web server.
3. **Include "Last Refreshed" Timestamps:** Always render the current timestamp (`Get-Date`) in a section header or badge so users can immediately verify data freshness.

Next Chapter: Chapter 11: Advanced Features, Optimization, and Troubleshooting

Chapter 11: Advanced Features, Layout Customization, and Troubleshooting

Table of Contents

11.1 Advanced features Overview

As you scale **PSWriteHTML** across enterprise environments, building reports evolves beyond basic tables into crafting high-density, visually appealing dashboards.

This chapter covers modern UI components like **InfoCards**, dynamic layout density controls, advanced JavaScript/CSS customization, performance tuning, and diagnostic troubleshooting.

11.2 Modern Dashboard Components & Styling

11.2.1 Key Metrics with `New-HTMLInfoCard`

To display high-level KPIs, operational counters, or status summaries at the top of a dashboard, use New-HTMLInfoCard . InfoCards support icons (emojis, FontAwesome), custom colors, subtitle text, and distinct visual styles:

- **-Styles:** `Standard`, `Compact`, `Fixed`, `Classic`, `NoIcon`.
- **-ShadowIntensity:** `None`, `Subtle`, `Normal`, `Bold`, `ExtraBold`, `Custom` (with `-CustomShadowColor`).
- **-Icon:** Icon to display on the card. Can be an emoji (like ?, ?, ?).
- **-IconSolid, -IconRegular, -IconBrands:** Use FontAwesome icons with different styles

```
New-HTML -Title "Executive Security Summary" -Show {
    New-HTMLTab -Name "Overview" -IconSolid "user-shield" {

        # Section wrapping InfoCards with Comfortable spacing
        New-HTMLSection -HeaderText "Key Security Metrics" -Density Comfortable {
```

```
            New-HTMLInfoCard -Title "Identity Protection" `
                             -Number "Active" `
                             -Subtitle "0 Risky Users Flagged" `
                             -Icon "Shield"  `
                             -IconColor '#0078d4'

            New-HTMLInfoCard -Title "Failed Logins (24h)" `
                             -Number "142" `
                             -Subtitle "Within expected threshold" `
                             -Icon "Error" `
                             -IconColor '#d9534f'

            New-HTMLInfoCard -Title "MFA Enrollment" `
                             -Number "98.4%" `
                             -Subtitle "+1.2% from last week" `
                             -Icon "Report" `
                             -IconColor '#5cb85c'
        }
    }
}
```

This is the result:

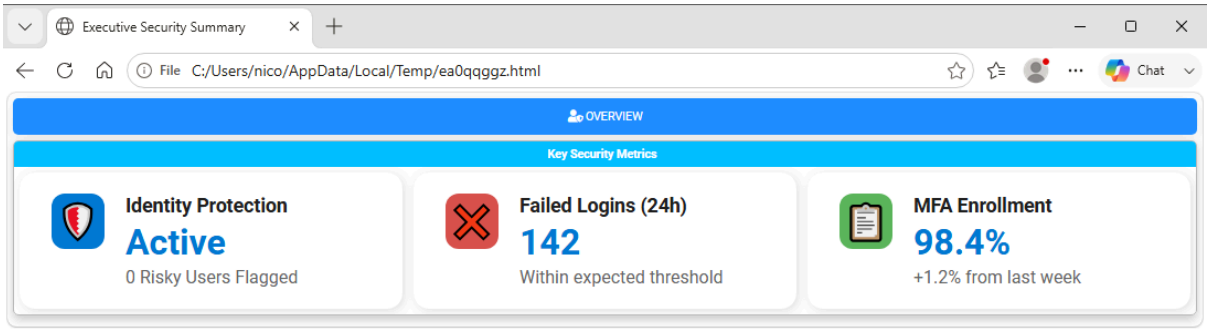

Figure 11.1. Dashboard information cards displaying high-level operational metrics.

11.2.2 Controlling Layout Density with `-Density`

To handle responsive grid layouts without manually computing pixel widths or complex CSS flexbox rules, use the `-Density` parameter on `New-HTMLSection` or `New-HTMLPanel`. This automatically enables responsive wrapping and adjusts card/element margins:

- **``Spacious``:** Generous padding and whitespace; ideal for high-level executive summaries.
- **``Comfortable``:** Balanced spacing suited for standard multi-card layouts.
- **``Compact``:** Reduced margins for viewing dense telemetry datasets.
- **``Dense`` / ``VeryDense``:** Tight alignment maximizing screen real estate for NOC display walls.

```
New-HTMLSection -HeaderText "NOC Operations Grid" -Wrap wrap -Density VeryDense {
    # InfoCards or panels rendered here auto-wrap tightly to fit dense monitor
setups
    foreach ($Metric in $NocMetrics) {
        New-HTMLInfoCard -Title $Metric.Name -Number $Metric.Value -Icon "server"
    }
}
```

11.3 Advanced Scripting Patterns

11.3.1 Dynamic Component Loop Generation

Instead of hardcoding every tab, section, or table, generate PSWriteHTML layout components dynamically using standard PowerShell loops (`foreach`, `for`).

```
$Servers = @('DC01', 'DC02', 'SQL01', 'WEB01')

New-HTML -Title "Multi-Server Assessment" -FilePath "MultiServer.html" {
    foreach ($Server in $Servers) {
        New-HTMLTab -Name $Server -IconSolid "server" {
            New-HTMLSection -HeaderText "Diagnostic Summary for $Server" -Density
Compact {
                New-HTMLInfoCard -Title "Health" -Number "OK" -Icon "check-circle"
                New-HTMLPanel {
                    New-HTMLText -Text "Detailed diagnostic metrics captured for
node: <b>$Server</b>"
                }
            }
        }
    }
}
```

11.3.2 Embedding Raw Client-Side JavaScript

Inject custom JavaScript directly into the document using the `-UseJavaScriptLinks` parameter on New-HTML to handle bespoke interactions:

```
$ScriptBlock = @"
    document.addEventListener('DOMContentLoaded', function() {
        console.log('PSWriteHTML Dashboard Execution Initialized.');
    });
"@

New-HTML -Title "Custom Scripting" -FilePath "Scripting.html" -UseJavaScriptLinks
$ScriptBlock {
    New-HTMLTab -Name "Main" {
        New-HTMLSection -HeaderText "Console Verification" {
            New-HTMLPanel {
                New-HTMLText -Text "Open browser developer console (F12) to
inspect client execution logs."
            }
        }
    }
}
```

11.3.3 Correlating Tables and Charts with Events (`New-ChartEvent`, `New-DiagramEvent`, and `New-TableEvent`)

Interactive charts can be connected to a table so that selecting a chart value searches for and highlights the matching table rows. This is implemented using New-ChartEvent or New-DiagramEvent cmdlets with the `-DataTableID` parameter on `New-HTMLTable` and the `-ID` and `-ColumnID` parameters on the chart or diagram event. The table must be rendered before the chart or diagram for proper correlation.

The parameters used to connect the components are:

- **``-ID``: Identifies the target table when used with `New-DiagramEvent`. It is also an alias for `-TableID` in**
  `New-TableEvent`.
- **``-ColumnID``: Selects the zero-based table column used to match a chart or diagram event. The number follows the property**
  order passed to `-DataTable`, with the first property being column `0`. For example, if the table is built from `Select-Object Name, Status`, then `Name` is column `0` and `Status` is column `1`.
- **``-DataTable``:** Supplies the objects or records that `New-HTMLTable` renders as rows.
- **``-DataTableID``: Assigns a stable identifier to the table. Use the same value in `New-ChartEvent` or `-ID` on a related**
  diagram/table event.
- **``-DataStore``: Selects how `New-HTMLTable` provides its data to the page. The supported values are `HTML` (the**
  default, renders the table data directly as HTML), `JavaScript` (embeds the data in the page as JavaScript and is required for browser-side chart or diagram correlation), and `AjaxJSON` (loads the data from a JSON endpoint for hosted/server-side tables). Use `JavaScript` for complex scenarios with client-side data; `AjaxJSON` requires a `-FilePath` on `New-HTML` and a web server that can serve the generated JSON data.

Give the table a stable identifier with `-DataTableID` and pass the same value to `New-ChartEvent -DataTableID`. The `-ColumnID` parameter is zero-based and identifies the table column used for matching, so it must follow the order of the properties supplied to `-DataTable`. Table and diagram event commands also use `-ID` (an alias for the target table ID) and `-ColumnID`. This event ID is separate from the chart's own `-Id` parameter.

Emit the table before the chart event consumer and use `-DataStore JavaScript` when the chart should correlate with client-side table data. For a diagram, place `New-DiagramEvent -ID $TableID -ColumnID 0` inside `New-HTMLDiagram`; `-ID` identifies the table and `-ColumnID` selects the table column used by the diagram event.

The following example creates a pie chart and a table from the same data. Clicking a pie slice searches the first table column, `Name`, because it is column `0` in the selected property order:

```
$Data = @(
    [PSCustomObject]@{ Name = 'Alpha'; Value = 12 }
    [PSCustomObject]@{ Name = 'Beta';  Value = 18 }
    [PSCustomObject]@{ Name = 'Gamma'; Value = 9 }
)

$TableID = 'ChartTable'

New-HTML -TitleText 'Chart and Table' -FilePath "$PWD\chart-table.html" -Online
-Show {
    New-HTMLTable -DataTable $Data -DataTableID $TableID -DataStore HTML

    New-HTMLChart -Title 'Values by Name' {
        foreach ($Item in $Data) {
            New-ChartPie -Name $Item.Name -Value $Item.Value
        }

        New-ChartEvent -DataTableID $TableID -ColumnID 0
    }
}
```

It produces the following interactive correlation:

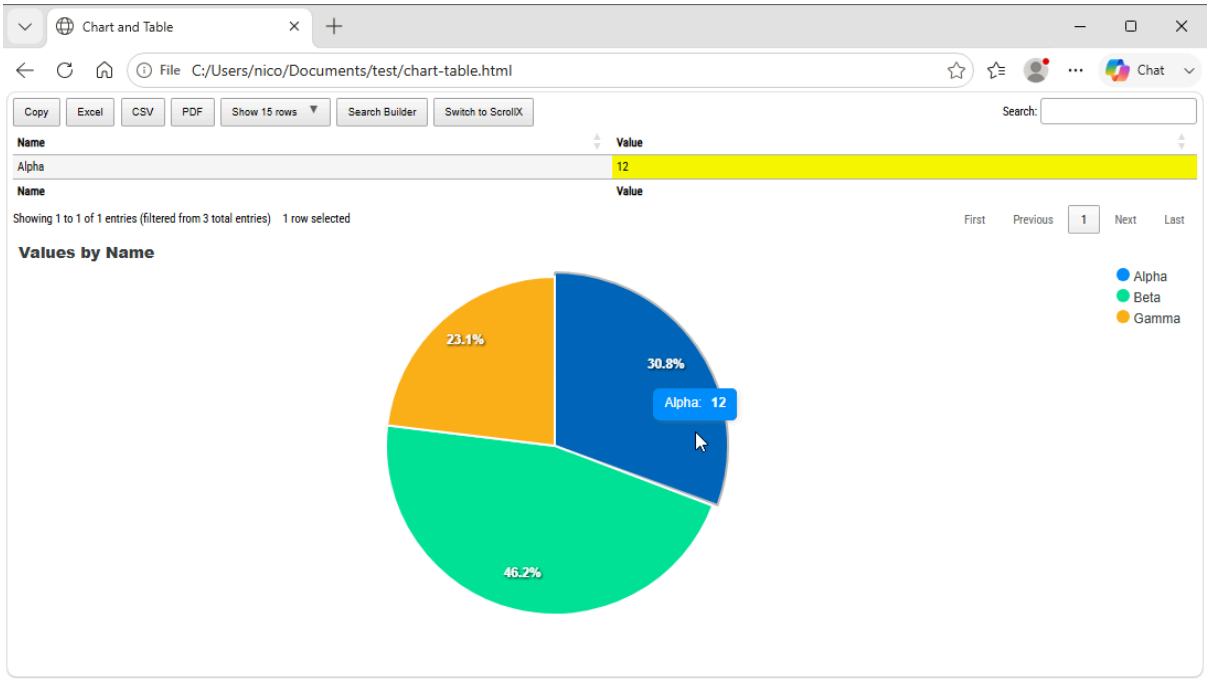

Figure 11.2. A chart and DataTable connected through an interactive correlation event.

You can also use New-TableEvent. It listens for a row selection in one table and filters another table.

Note PSWriteHTML provides built-in events for chart-to-table, diagram-to-table, and table-to-table correlation through **New-ChartEvent**, **New-DiagramEvent**, and **New-TableEvent**. There is no built-in table-to-chart event. If you need to filter a chart based on table row selection, implement a custom JavaScript event handler using the `-UseJavaScriptLinks` parameter on `New-HTML`.

11.4 Performance Optimization for Enterprise Scaling

Generating reports with tens of thousands of records can impact initial creation time or browser responsiveness. Follow these tuning rules:

1. **Filter Data Upfront:** Select required properties *before* passing objects into `New-HTMLTable`. Avoid passing raw, unfiltered WMI/CIM or Active Directory objects directly.

```
# Bad: Passing raw AD objects with 100+ un-needed properties
$Users = Get-ADUser -Filter * -Properties *

# Good: Selecting target properties upfront
$Users = Get-ADUser -Filter * -Properties DisplayName, Mail | Select-Object
SamAccountName, DisplayName, Mail
```

2. **Balance Online vs. Offline Mode:**
   - Use ``**-Online**`` for live internal dashboards where client web endpoints have internet access. Web dependencies (Bootstrap, DataTables, ApexCharts) pull from fast CDNs, keeping output file sizes below 50 KB.
   - Use ``**-Offline**`` when deploying reports into air-gapped corporate subnets. Note that offline asset bundling increases file size.

3. **Limit DataTables Page Lengths:** Keep `-PageLength 25` or `-PageLength 50` on `New-HTMLTable`. Defaulting page lengths to `All` forces client browsers to render thousands of dynamic DOM elements concurrently, causing tab freezing.

11.5 Troubleshooting Common Pitfalls

11.5.1 Issue 1: Empty Tables or Missing InfoCards

- **Symptom:** Page renders, but card sections or tables are completely blank.
- **Root Cause:** Variable supplied to `-DataTable`, `-Data`, or `-Number` is null or unpopulated.
- **Solution:** Add upfront data validation before building HTML containers:

```
if ($null -eq $MyData -or $MyData.Count -eq 0) {
    Write-Warning "Data array empty. Rendering fallback card."
    $MyData = [PSCustomObject]@{ Status = "No Records Available"; Count = 0 }
}
```

11.5.2 Issue 2: IIS 404 or Access Denied During Dashboard Update

- **Symptom:** Scheduled task fails to overwrite `index.html` in `C:\inetpub\wwwroot\`.
- **Root Cause:** The service account executing the PowerShell scheduled task lacks file Modify /Write permissions.
- **Solution:** Grant full folder permissions on the target web root to the execution user account or `NT AUTHORITY\SYSTEM`.

11.5.3 Issue 3: Missing Icons in Air-Gapped Environments

- **Symptom:** FontAwesome icons or InfoCard symbols display as hollow square glyphs on internal servers.
- **Root Cause:** Report was generated with `-Online`, but client browsers cannot reach CDN endpoints.
- **Solution:** Pass `-Offline` to `New-HTML` to bundle static local assets into the target path.

11.6 Diagnostic Checklist

Verify this checklist when diagnosing reporting pipelines:

```
[ ] 1. Is PSWriteHTML up to date on the build host? (Update-Module PSWriteHTML
-Force)
[ ] 2. Does the target IIS folder permit Write actions for the Scheduled Task
user?
[ ] 3. Are layout density settings (-Density) applied to appropriate section
levels?
[ ] 4. Is the browser refresh meta header present for real-time NOC dashboards?
[ ] 5. Are script blocks and curly braces balanced across all nested sections?
[ ] 6. If emailing alerts: Is Email-HTML used instead of New-HTML?
```

Next Chapter: Chapter 12: The Evotec PowerShell Module Suite

Chapter 12: The Evotec PowerShell Module Suite

Table of Contents

12.1 Evotec products Overview

PSWriteHTML is part of an extensive suite of over 80 open-source PowerShell modules created and maintained by **Przemysław Kłys** (Przemyslaw.Klys on PowerShell Gallery).

This chapter categorizes the complete ecosystem of modules into distinct administrative domains—ranging from messaging, office document generation, and Active Directory auditing to security policies, image processing, and system customization.

12.2 Core Frameworks & Helper Libraries

- **PSSharedGoods:** The core foundation library containing hundreds of shared helper functions for string manipulation, file handling, logging, and object processing across all Evotec modules.
- **PSPublishModule:** A project build and publishing framework for preparing, testing, and uploading PowerShell modules to the PowerShell Gallery.
- **PSParseHTML:** An HTML/CSS/JavaScript parser designed to extract, inspect, and analyze web content programmatically.

12.3 Communication & Notification Modules

- **Mailozaurr:** An advanced email engine utilizing MailKit and MimeKit supporting SMTP, POP3, IMAP, Graph API, and OAuth2.
- **PSTeams:** Sends rich webhook notifications to Microsoft Teams (supports Adaptive Cards, Hero Cards, and List Cards).

- **PSDiscord:** A lightweight module for sending structured webhooks and notifications to Discord channels.
- **Connectimo:** A connectivity module for handling network checks and remote endpoint testing.
- **Emailimo:** Helper module for managing inline email body structures and templates.

12.4 Office & Document Generation

- **PSWriteOffice:** Creates and reads Word (`.docx`), Excel (`.xlsx`), PowerPoint (`.pptx`), Markdown, and CSV files natively without Microsoft Office installed. See https://github.com/EvotecIT/PSWriteOffice
- **PSWriteWord / Documentimo:** Dedicated tools for building Microsoft Word documentation and structured reports. (obsolete, functionality merged into PSWriteOffice)
- **PSWriteExcel / Excelimo:** Modules for creating and formatting Excel workbooks without requiring local Office installations. (obsolete, functionality merged into PSWriteOffice)
- **PSWritePDF:** Programmatically creates, edits, merges, splits, and formats PDF documents. (obsolete, functionality merged into PSWriteOffice)
- **MarkdownPrince:** A utility for parsing, converting, and processing Markdown files. (obsolete, functionality merged into PSWriteOffice)

12.5 Active Directory, Security & Auditing

- **Testimo:** An Active Directory health and security audit framework evaluating domains against hundreds of best-practice checks.
- **GPOZaurr:** Group Policy analysis and repair tool designed to audit, troubleshoot, and fix GPO inconsistencies.
- **PSWinReporting / PSWinReportingV2:** Event log viewing, collecting, and security reporting engine focused on Domain Controllers.
- **SecurityPolicy:** A module wrapping `secedit` for managing Windows User Rights Assignments and local security policies.
- **AuditPolicy:** Replaces `auditpol.exe` with a custom wrapper to view and adjust Windows Security Audit policies.
- **PSPasswordExpiryNotifications:** Automates password expiry warning emails to users and managers using customizable templates.
- **PSWinDocumentation (AD, DNS, Exchange, O365, AWS):** Datasets and extraction tools that document infrastructure components into Word, Excel, or SQL databases.
- **AccountTracker:** Tracks non-compliant account placement in Active Directory services and OUs.
- **PSBlackListChecker:** Verifies IP addresses against global DNS blacklists and sends automated alert reports.

12.6 File Transfer, Security & Cryptography

- **Transferetto:** A reusable module/library for FTP, FTPS, SFTP, SCP, FXP, SSH commands, shell access, and SSH tunneling.
- **PSPGP:** Encrypts and decrypts files, folders, and text strings using PGP keys natively.

- **VirusTotalAnalyzer:** Interacts with the VirusTotal API to scan files, hashes, and URLs for threat intelligence.
- **PowerShellManager:** Extracts and recovers deleted or execution-flagged PowerShell scripts straight from Windows Event Logs for malware analysis.

12.7 Graphics, UI & Desktop Customization

- **ImagePlayground:** Image processing engine capable of generating QR codes, barcodes, charts, and applying image filters.
- **PowerBGInfo:** Modern BGInfo replacement that generates dynamic system information desktop background wallpapers.
- **DesktopManager:** Manages, positions, and switches wallpapers across multi-monitor setups.
- **ConsoleMonster:** Terminal UI engine for building rich interactive console applications using Spectre.Console.
- **Statusimo / Dashimo:** Legacy status page and dashboard builders (integrated into modern PSWriteHTML workflows).

12.8 Cloud, Service & Vendor Integrations

- **O365Essentials / GraphEssentials / Graphimo:** Helper modules for managing Microsoft 365, Azure AD, and Intune via Microsoft Graph API.
- **O365Synchronizer:** Cross-tenant synchronization utility for sync-ing personal contacts, users, and guest objects.
- **PowerInfoblox:** Helper module for managing Infoblox IPAM and DNS appliance configurations.
- **PSLansweeper:** Queries Lansweeper asset databases for reporting and inventory tracking.
- **IISParser:** High-performance IIS log parsing module for traffic and diagnostic audits.
- **UnifiStockTracker:** Utility for tracking product stock in the Ubiquiti Unifi online store.
- **PSWordPress:** Interacts with WordPress web instances via REST API endpoints.

12.9 Ecosystem Integration Pattern

Combining these specialized modules yields robust automation solutions. For example, auditing AD security, rendering an HTML report, converting to PDF, and emailing a summary via Microsoft Graph:

```
Import-Module Testimo
Import-Module PSWriteHTML
Import-Module Mailozaurr

# 1. Run Active Directory Health Audit
$AuditData = Invoke-Testimo -RootDomain

# 2. Build Interactive HTML Report
$ReportPath = "$env:TEMP\ADHealthReport.html"
New-HTML -Title "AD Security Audit" -FilePath $ReportPath {
    New-HTMLTab -Name "Audit Details" {
        New-HTMLSection -HeaderText "Testimo Results" {
```

```
            New-HTMLTable -DataTable $AuditData
        }
    }
}

# 3. Generate HTML Email Body
$Body = New-EmailBody {
    New-EmailSection {
        New-EmailPanel {
            New-EmailText -Text "The Active Directory health audit completed
successfully. Attached is the interactive report."
        }
    }
}

# 4. Send via Mailozaurr Graph API
Send-MgEmail -To "admin@domain.com" `
             -Subject "Weekly AD Audit Report" `
             -Body $Body `
             -BodyType HTML `
             -Attachments $ReportPath
```

Next Chapter: Chapter 13: Documentation and Resources

Chapter 13: Documentation and Resources

Table of Contents

13.1 Web pages

There are lots of official WEB pages and and articles available for **PSWriteHTML** on the Evotec site, like:

- **PSWriteHTML project page:** Current official project portal. Installation, releases, API, examples and downloads.
- **PSWriteHTML API Reference:** Current API documentation covering the module's functions /cmdlets. Individual pages are generated from the source documentation.
- **New-HTML API:** Main entry point for constructing an HTML document. https://evotec.pl/projects/pswritehtml/api/new-html

But also official articles are available there, like:

- Advanced HTML reporting using PowerShell (probably one of the best general PSWriteHTML articles). https://evotec.xyz/advanced-html-reporting-using-powershell
- Enhanced Dashboards with PSWriteHTML: Introducing InfoCards and Density Options. https://evotec.xyz/enhanced-dashboards-with-pswritehtml-introducing-infocards-and-density-options
- Seamless HTML Report Creation: Harness the Power of Markdown with PSWriteHTML. https://evotec.xyz/unlocking-seamless-html-report-creation-using-markdown-with-pswritehtml-powershell-module
- Image Manipulation, Image Resize, Image Combine and more with PowerShell. https://evotec.xyz/image-manipulation-image-resize-image-combine-and-more-with-powershell
- Easy way to create diagrams using PowerShell and PSWriteHTML. https://evotec.xyz/easy-way-to-create-diagrams-using-powershell-and-pswritehtml

Do not forget the official GitHub repository on GitHub at https://github.com/EvotecIT/PSWriteHTML, which is is arguably the single most important page.

The repository currently has roughly 2,655 commits and contains:

Docs Examples Public Private Resources Tests Website changelog module manifest/source README and community resources

13.2 Videos

A particularly useful YouTube series was made in 2023 by JackedProgrammer:

- Part 1: HTML Reports made easy https://www.youtube.com/watch?v=gazRo-otfwA
- Part 2: Create Visio Style Diagrams https://www.youtube.com/watch?v=p6SRDWNJdeg
- Part 3: Create Web Calendars https://www.youtube.com/watch?v=RUXnS6Rkzoc
- Part 4: Create Dashboards https://www.youtube.com/watch?v=TBRlvcdlDPA
- Part 5: Charts https://www.youtube.com/watch?v=P7g6rsESauQ
- Part 6: Adding Events https://www.youtube.com/watch?v=cD-LYYbEg9s

- genindex
- modindex
- search