Reports WebData Reference

Use WebData sources in reports.

Reports WebData (Reference) onfiguration objects facilitate data exchange between applications using JSON, XML, HTML, or text formats, enabling integration with web services and RESTful APIs. Web Data objects provide:

- RESTful API integration
- Multiple data format support
- Authentication handling
- Dynamic tag binding
- Request/response management
- File save capabilities

WebData enables both consuming external APIs and exposing solution data to other systems.

Configuration Properties
Creating WebData Objects
Authentication Methods
Request Methods
WebData Editor
Response Handling
Headers Configuration
File Operations
Padding Options
Error Handling
Common Use Cases
Best Practices Checklist
Troubleshooting

Configuration Properties

Property	Description	Required
Name	Unique WebData identifier	Yes
Encoding	Data format (JSON/XML/HTML/Text)	Yes
DefaultURL	Default API endpoint	No
Authorization	Authentication method	No
Headers	Request/response metadata	No
SaveFileName	Local file path for saving	No
Padding	Whitespace handling	No
EditSecurity	Permission groups	No
Length	Data size in bytes	Auto
Description	Documentation text	No

Creating WebData Objects

1. Navigate to **Reports** **WebData**
2. Click **Add** button
3. Configure minimum requirements:
    - **Name**: Unique identifier
    - **Encoding**: Select format
4. Optional: Set DefaultURL and Authorization
5. Click **OK**

Authentication Methods

No Auth

No credentials sent:

```
Authorization: NoAuth
```

Bearer Token

Token-based authentication:

```
Authorization: Bearer <token>
Headers: {"Authorization": "Bearer {{Tag.APIToken}}"}
```

Basic Auth

Username/password authentication:

```
Authorization: Basic
Username: {{Tag.Username}}
Password: {{Tag.Password}}
```

Custom

Define custom authentication:

```
Headers: {
  "X-API-Key": "{{Tag.APIKey}}",
  "X-Client-ID": "{{Tag.ClientID}}"
}
```

Request Methods

GET Requests

Execute data retrieval:

```
// Async GET request
string response = await @Report.WebData.MyAPI.GetRequestAsync();

// Store in tag
@Tag.APIResponse = response;

// Access via DocumentClient
string data = @Report.WebData.MyAPI.DocumentClient;
```

POST Requests

Send data to API:

```
// Execute POST
await @Report.WebData.MyAPI.PostRequestAsync();

// Response stored in specified tag
string result = @Tag.PostResult;
```

In POST requests, the response from the server is not returned directly by the method call. Instead, the response binding is configured declaratively in **Reports / WebData**, using the **Headers** column. There, you map the built-in ResponseTag property to any Text-type tag in your solution using the standard tag reference syntax, you can see the example in the table below. Once that mapping is in place, executing the request with await @Report.WebData.MyAPI.PostRequestAsync() will automatically store the server's response body into the linked tag.

Key	TagName
ResponseTag	MyResponseTag



ResponseTag is a default name to receive the response.

MyResponseTag is any Text-type tag in your solution.

WebData Editor

Define request structure and bind tags:

JSON Format

```
{
  "timestamp": "{{Info.Date}}",
  "values": {
    "temperature": {{Tag.TankFarm/Tank1/Temp}},
    "pressure": {{Tag.TankFarm/Tank1/Pressure}},
    "status": "{{Tag.Equipment/Status}}"
  }
}
```

XML Format

```
<data>
  <timestamp>{{Info.Date}}</timestamp>
  <temperature>{{Tag.TankFarm/Tank1/Temp}}</temperature>
  <pressure>{{Tag.TankFarm/Tank1/Pressure}}</pressure>
</data>
```

Response Handling

Store Response Methods

Method 1: Direct Assignment

```
@Tag.JSONResponse = await @Report.WebData.API.GetRequestAsync();
```

Method 2: DocumentClient Property

```
await @Report.WebData.API.GetRequestAsync();
string data = @Report.WebData.API.DocumentClient;
```

Method 3: Tag Binding in Editor Map response fields to tags in WebData Editor

---

Headers Configuration

Common headers:

```
{
  "Content-Type": "application/json",
  "Accept": "application/json",
  "ResponseTag": "Tag.APIResult",
  "X-Request-ID": "{{Tag.RequestID}}"
}
```

File Operations

Save Response to File

```
// Configure SaveFileName
@Report.WebData.API.SaveFileName = @"C:\Data\response.json";

// Execute and save
await @Report.WebData.API.GetRequestAsync();
@Report.WebData.API.Save();
```

Dynamic File Names

```
C:\Data\API_{{Tag.Date}}_{{Tag.Time}}.json
```

Padding Options

Option	Effect	Use Case
Compact	No extra whitespace	Minimal payload
PadRight	Space after values	Fixed-width format
PadLeft	Space before values	Numeric alignment

Error Handling

```
try
{
    string response = await @Report.WebData.API.GetRequestAsync();

    if (!string.IsNullOrEmpty(response))
    {
        @Tag.Status = "Success";
        ProcessResponse(response);
    }
}
catch (Exception ex)
{
    @Tag.ErrorMessage = ex.Message;
    @Tag.Status = "Failed";
}
```

Common Use Cases

Weather API Integration

```
@Report.WebData.Weather.DefaultURL =
    "https://api.weather.com/v1/location/{{Tag.City}}";

string weather = await @Report.WebData.Weather.GetRequestAsync();
```

IoT Data Upload

```
// Configure POST body with sensor data
await @Report.WebData.IoT.PostRequestAsync();
```

Database Sync

```
// Get external data
string data = await @Report.WebData.External.GetRequestAsync();

// Parse and update local database
UpdateDatabase(data);
```

Best Practices Checklist

- **Validate URLs** - Check endpoints before deployment
- **Secure credentials** - Use tags for sensitive data
- **Handle timeouts** - Implement retry logic
- **Parse responses** - Validate JSON/XML structure
- **Log errors** - Track failed requests
- **Test thoroughly** - Verify all scenarios
- **Document APIs** - Clear descriptions

Troubleshooting

Connection failed:

- Verify URL accessibility
- Check network/firewall
- Confirm authentication
- Review proxy settings

Invalid response:

- Check encoding format
- Validate JSON/XML syntax
- Verify content-type header
- Review API documentation

Authentication errors:

- Confirm credentials
- Check token expiration
- Verify header format
- Review API permissions

Performance issues:

- Implement caching
- Reduce request frequency
- Optimize payload size
- Use async methods

In this section...
