ExagoBI Technical Guide

Version 2021.2

Table of Contents

Technical Specifications 5
High Availability 7
Updating Recommendations 11
Updating to the Latest Version (Potentially Breaking Changes) 13
Configuring IIS for Exago 30
Installing Exago on Windows 34
Installing Exago on Linux 36
Installing Exago on Azure 43
Installing Exago on Docker 48
Configuring Remote Report Execution Host for Azure App Service Deployments 49
Installing Optional Features 50
Manual Application Installation 50
Windows Registry Changes 56
Installing the Scheduler Service 60
Storage Management: Introduction 60
Storage Management: Database Schema 66
Storage Management: Getting Started 70
Storage Management: Transitioning from Legacy Storage Methods 72
Moving Files Between Storage Management Databases 85
Storage Management: Custom Implementation 89
Storage Management: Utility (v2021.1+) 96
Scheduler Service Configuration 103
Time Zone Calculation Enhancements in v2019.1 105
Remote Execution 107
Execution Caching 107
Scheduler Queue 109
User Identification 112
Setting up a State Server 115
Deploying to Production 118
Transition from HTTP to HTTPS 120
Cross-Origin Embedding of Exago 121
Security Checklist 122
Multi-Tenant Environment Integration 124
HTML Tag Sanitization 126
Enable FIPS Compliant Encryption on Windows 126
About the Administration Console 128
Storage Management 130
Main Settings 133
Culture Settings 134
Feature/UI Settings 135
Report-Level SQL Objects 141
Programmable Object Settings 142
Filter Settings 143
Database Settings 144
Incremental Loading 148
Scheduler Settings 149
Other Settings 152
Admin Console Password Encryption 154
Data Sources 155
Data Source Drivers 159

CData Drivers 160
Data Objects 163
Multiple Data Models 178
Automatic Database Discovery 179
Join Configuration 182
Special Cartesian Processing 183
Parameters 184
Roles 187
Custom Functions 191
Custom Filter Functions 195
Custom Aggregate Functions 199
Custom Options 203
Hidden Flags 205
Configuration Management and Propagation 206
Configuration File Options and Optimizations 208
Modifying Configs in a Production Environment 213
Using the Administration Console and Exago API 213
Config File XML & API Setting Reference (General Nodes) 215
Config File XML Reference (All Nodes but General) 229
Monitoring: Setup 242
Monitoring: System Overview 244
Monitoring Database Schema 245
Styling Exago 247
Styling the Home Page 248
Customizing Getting Started Content 249
Custom Context Sensitive Help 251
Themes 254
Create a Custom ExpressView Theme (v2021.1+) 256
Adding Fonts to Exago 258
Keyboard Shortcuts 258
Multi-Language Support 260
An Overview of Exago Extensions 261
Introduction to Action Events 263
Global Action Events 264
Actionable UI Elements 267
ClientInfo 268
Introduction to Server Events 270
List of Server Events 272
Global Event: OnAfterLoadReportsList 273
Global Event: OnAfterReportExportSuccess 273
Global Event: OnConfigLoadEnd 274
Global Event: OnConfigLoadStart 274
Global Event: OnDataCombined 274
Global Event: OnDataFieldsRetrieved 275
Global Event: OnExceptionThrown 275
Global Event: OnExecuteSqlStatementConstructed 275
Global Event: OnExportCsvCell 276
Global Event: OnFilterSqlStatementConstructed 276
Global Event: OnGetUserPreferences 276
Global Event: OnLoadReportParameters 277
Global Event: OnOkFiltersDialog 277
Global Event: OnOkParametersDialog 277

Global Event: OnParameterSqlStatementConstructed 278
Global Event: OnRenameFolderEnd 278
Global Event: OnRenameFolderStart 278
Global Event: OnReportExecuteEnd 278
Global Event: OnReportExecuteInit 279
Global Event: OnReportExecuteStart 279
Global Event: OnReportFailValidation 280
Global Event: OnReportSaveStart 280
Global Event: OnScheduledReportComplete 280
Global Event: OnScheduledReportExecuteStart 281
Global Event: OnScheduledReportExecuteSuccess 281
Global Event: OnSetUserPreferences 281
Global Event: OnWebServiceExecuteEnd 282
Introduction to the .NET API 282
REST – Introduction 282
Using JSON 286
List of REST Endpoints 288
REST – Batch 290
REST – Sessions 292
REST – Data Sources 298
REST – Data Objects 300
REST – Data Joins 305
REST – Folders 308
REST – Storage Management 311
REST – Functions 312
REST – Parameters 315
REST – Role Security 316
REST – SchedulesV2 323
REST – Server Events 332
REST – Config Settings 333
Constants and Enumerators 341
Executing Reports with the API 349
Excel: Grouping on Separate Worksheets 352
JavaScript API 353
JavaScript API: Filters and Parameters 362
JavaScript API: Report and Dashboard Execution Timing 366
Application and Performance Logging 366
Version, Connections, and Permissions 371
Troubleshooting Report Output 374
No Data Qualified Options 375
Accessing the Log File 377
Submitting a Debug Package 378
WebReportsDebug.xml 379
See Full Error Details 379
Error Message Glossary 379

Technical Specifications

Operating Systems

> **IMPORTANT**
>
> When using Content Security Policy (CSP) rules in the web server configuration, additional response headers are required for compatibility with Exago. See Security Checklist for more information.

Windows

- Windows Server 2012+ / Windows 10
- Internet Information Services (IIS) 7+
- .NET Framework 4.6.1+, version 4.7.2 recommended

Linux

Supported/Recommended Distributions

- Ubuntu 14+

Additional Distributions

- Red Hat Enterprise Linux 7
- SUSE Linux Enterprise Server 12
- CentOS 7
- Fedora 21-25
- Debian 8-9

> **WARNING**
>
> Exago is incompatible with SELinux.

Requirements

Exago is not compatible with mono versions installed from system repositories. It is highly recommended that only official mono-project repositories are used or that the Exago installer install mono.

Mono

> **WARNING**
>
> These are the only versions of Mono that are compatible with Exago. In addition, the Exago installer will install a patched version of System.Web.dll. This patch can also be installed manually.

- mono 5.10.1.20, 6.8.0.105 or 6.8.0.123 (*v2020.1+*).
- *Optional*: mono-basic, which provides support for VB.NET

Apache 2.4+

- mod-mono

NGINX

- mono-fastcgi-server4

The Exago installer can automatically download and install supported versions of mono, mod-mono or mono-fastcgi-server4.

Data Sources

Fully Supported

  - Microsoft SQL Server 2000+
  - PostgreSQL 7.1+
  - Oracle Version 9i+
  - MySQL 5.0+
  - Amazon Aurora
  - IBM DB2
  - IBM Informix
  - Redshift
  - Snowflake

- SQLite v2021.1.8+

Partially Supported

- MariaDB
- ElasticSearch*
- MongoDB*
- Microsoft OLAPâ€

* Limited support available, see CData Drivers: Known Limitations.
â€ Limited support available, see Data Sources: OLAP and MDX Queries.

Database APIs

- ODBC
- .NET Assembly Methods

Files

- XML
- Excel

Data Object Types

- Database Tables
- Database Views
- Stored Procedures
- Database Functions
- Parameterized SQL Statements
- .NET Assembly Methods

Browsers

Exago BI targets the most current stable versions of the following web browsers.

Desktop

Desktop browsers support the full user interface and Admin Console.

- Mozilla Firefox
- Internet Explorer 11
- Microsoft Edge
- Google Chrome
- Apple Safari (MacOS)

Mobile

Mobile browsers support viewing report output in the embedded Report Viewer and Dashboard Viewer.

- Google Chrome
- Apple Safari

Storage

Reports

- **Storage Management** (*v2020.1+*) â€" see the Storage Management: Introduction article for more information.

Schedules

- **File System**
- **Scheduler Queue** (custom)
  - .NET

Configuration

- **File System**
- **Azure Blob Storage**

Application Integration

AJAX enabled â€" .NET API and REST Web Service API for communication between Exago and host application

- **Server-side**
  - .NET API
  - REST API
- **Client-side**
  - JavaScript
  - IFrame

Extensibility

- **Custom Functions**

    - .NET
    - JavaScript (server-side)
- **Server Events**
    - .NET
    - JavaScript (server-side)
- **Action Events**
    - .NET (server-side) + JavaScript (client-side)
    - JavaScript (server- & client-side) [*Windows only*]

> **TIP**
>
> See **An Overview of Exago Extensions** for more information.

High Availability

This article will highlight recommendations for designing a highly available Exago environment. By creating multiple areas of redundancy, single points of failure can be eliminated. A system which is adaptable to component failures is highly available. This article will focus on how to make Exago highly available and is not an exhaustive discussion on the concept of high availability.

It is assumed that the host application integrating Exago will be made highly available on its own. Any reference to a host app in this document refers to the embedding host application.

Components

A highly available system will require a series of architecture choices that support redundancy. This includes implementation of:

- Exago's REST Web Service API
- Exago's Scheduler Service for Remote Execution and the Scheduler Queue if scheduling reports
- A Web Farm
- A State Server
- Use of highly available network or cloud storage for temporary files
- Use of databases or other distributed storage mechanism for content storage, Scheduler Queue storage and session storage

REST Web Service API

Since all high availability architectures are distributed, utilizing the **REST Web Service API** separates the host application from Exago. Clients looking to build a highly available system should use the REST Web Service API instead of the the .NET API, since the .NET API requires the host app and Exago reside on the same server.

When deployed in a Web Farm, each instance of the REST Web Service API will be duplicated for each instance of the Web Application.

REST Web Service API Resources

- REST Web Service API
- Admin Support Lab – .NET and REST APIs
- Admin Support Lab – Batch REST API

Scheduler Service and Remote Execution

The **Scheduler Service** lays the foundation for moving the execution of reports off of the Exago Web Application server and on to redundant execution servers. As the name suggests, Scheduler Services handle scheduled report execution but they also support distributed report execution.

Remote Execution servers are simply Exago Scheduler Services configured to process all report execution activity from the application immediately. A highly available implementation should include multiple Remote Execution servers, eliminating single points of failure.

With Remote Execution configured, when a user runs a report from the the application either by exporting it or clicking the run button, the first available scheduler configured for remote execution will execute that report. If a report is scheduled with a recurrence pattern, it will be run on schedulers configured for scheduled report executions.

Scheduler Service Resources

- Installing the Scheduler Service
- Scheduler Service Configuration
- Scheduler Settings
- Admin Support Lab â€" Scheduler Configuration
- Remote Execution

Scheduler Queue

The **Scheduler Queue** is an extensibility feature that builds upon out-of-the-box functionality of the Scheduler Service. The Scheduler Queue provides additional behavior that contributes to a high availability environment:

- **Centralized work control** â€" without the Scheduler Queue, the Exago Web Application will assign jobs to Scheduler Services in a round-robin fashion. Once the Web Application sends a job off to a Scheduler Service, that Service "owns" the job from then on. When a **Scheduler Queue** is implemented, the Queue owns the jobs and delegates work as required to the individual Scheduler Services.
- **Handling of Scheduler Service outages** â€" the Scheduler Queue is aware of the online status of the Scheduler Services. Should one go offline for any reason, the queue can reassign work to the others. Work for the individual Scheduler Services is managed by the Scheduler Queue instead of each isolated Scheduler Service.

The Scheduler Queue is replicated for each Exago Web Application and Scheduler Service. A database can store the scheduled job information and the Scheduler Queue instances access the shared information from that database.

The Scheduler Queue will only handle scheduled jobs, and is not utilized for remote execution jobs. Therefore, it is not necessary to implement the Queue when using the Scheduler Service exclusively for Remote Execution.

Scheduler Queue Resources

- Scheduler Queue

Web Farm

A **Web Farm** is a group of two or more servers supporting multiple instances of the application, managing user data transparently between them. While not an Exago-specific technology, a Web Farm increases availability by decreasing the likelihood of an outage due to Web Application and REST Web Service API servers being unavailable.

Exago web farms require a load balancer, the use of a state server and shared temporary file storage. Sharing the Exago configuration file is also recommended.

Web Farm Resources

- Set Up Exago in a Web Farm

State Server

In a standalone environment an Exago server will track its own user sessions. In a web farm, a user's requests may be handled by several different Exago servers over the lifetime of the session. A centralized **State Server** keeps track of user sessions so that many Exago instances can share the session information among them. The State Server should use a database for storing user session data.

State Server Resources

- Setting up a State Server
- Set Up Exago in a Web Farm

Temporary File Storage

Over the course of the report execution cycle, several **temporary files** are generated. As mentioned previously, in a web farm requests from a single user may be handled by several Exago servers. Therefore, all of the Exago servers in the farm need access to these temporary files.

Exago supports network and cloud storage of temporary files. This could be a mapped network drive, a symbolic file system link or a cloud solution such as Amazon S3 or Azure file storage.

Cloud storage, although it requires an Internet connection, typically comes with high availability from the service provider.

Temporary File Storage Resources

- Amazon S3 File Storage
- Microsoft Azure File Storage
- Admin Console > General > Main Settings

Use of Databases

Databases manufacturers support high availability. As each database is different, Exago doesn't make any specific recommendations on *how* to make them highly available. Consult with your database administrator or the documentation from your database provider for more information.

At a minimum, Exago suggests using databases for the following components:

- **Content Storage** â€" reports, themes, folders and templates created by users should be stored in a database. Exago's *Folder Management* and current *Storage Management* solutions provide this facility. Beginning with *v2020.1+*, Storage Management is the only supported content storage mechanism.
- **Scheduler Queue** â€" the Scheduler Queue can be implemented to use a database for job storage. Exago's Production Example of the Queue uses a database for this purpose.
- **State Server** â€" a state server is an integral part of a web farm. Storing sessions in a highly available database prevents loss of user sessions unexpectedly.

Database Resources

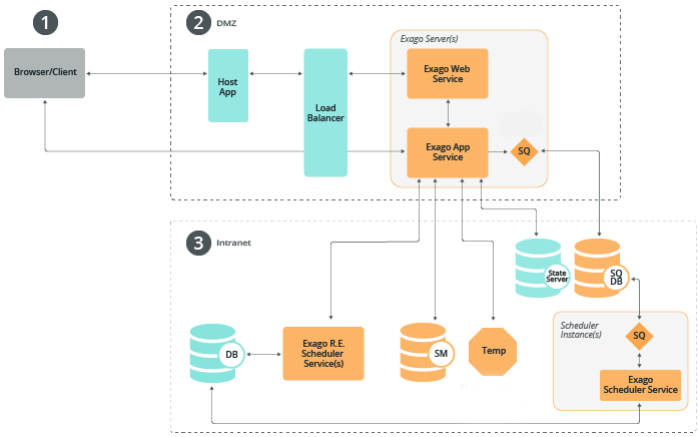
- Storage Management: Introduction
- Scheduler Queue
- Setting up a State Server

Architecture

Consider a basic highly available architecture depicted in the figure below. For clarity, redundant services such as multiple Exago servers, Remote Execution servers and Scheduler Services are only shown once.

The figure is broken up into three zones. While the zones are co-located in the diagram, in actual implementation they need not be. It may be advantageous to separate the different components physically and/or geographically if practical. They are:

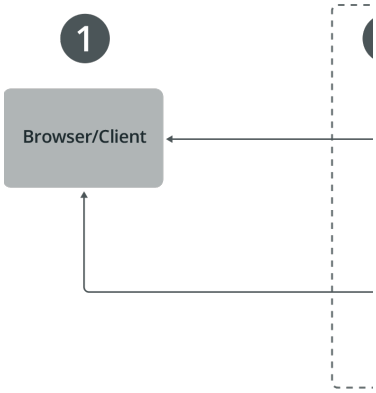
1. User's browser client
2. DMZ
3. Intranet



Basic highly available implementation

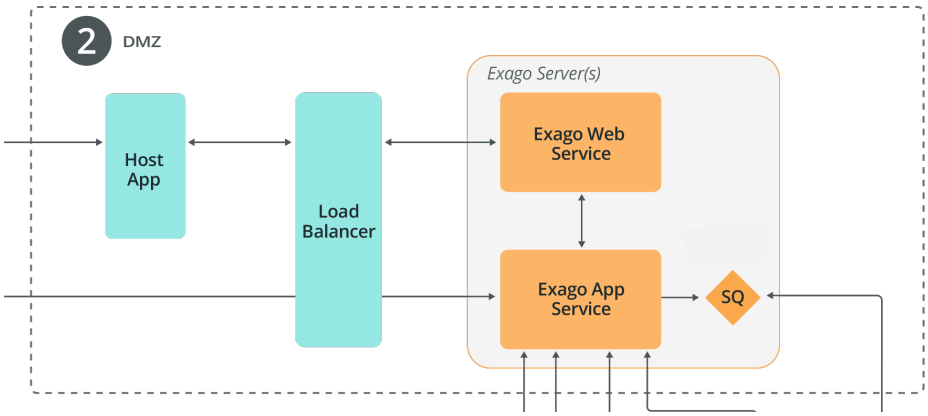
Each zone will be considered individually and then the pieces will be assembled together.

1. Browser Client



The browser communicates with both the host app and Exago (though a load balancer in the DMZ) via standard HTTP ports.

2. DMZ



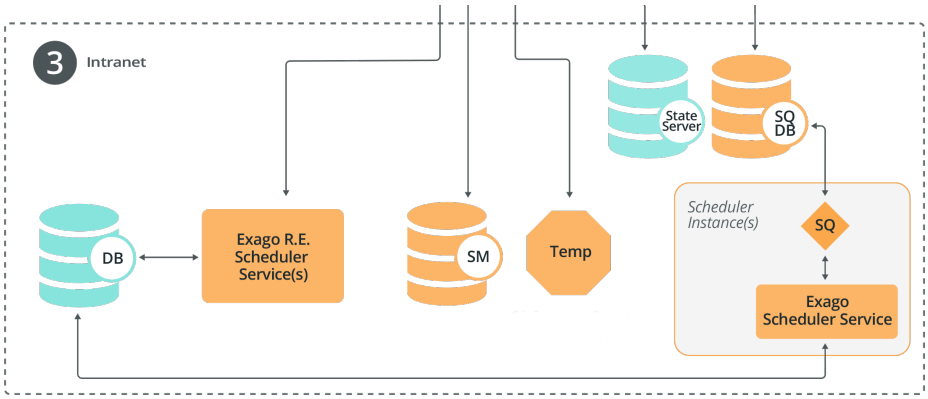
The **Demilitarized Zone (DMZ)** contains the servers that are exposed to the Internet for access by end user web browsers. It sits between the end user's browser and servers in the Intranet zone. This affords a layer of security without sacrificing high availability.

End user browsers communicate with the **Host App** and the **Load Balancer** via standard HTTP ports.

The job of the load balancer is to divide work between multiple **Exago Servers** in a **Web Farm**. Requests coming from users and the host application reach the load balancer where they are forwarded on to an available Exago server in the farm. User session state is managed between the different servers by a State Server in the Intranet zone.

Each Exago Server contains an instance of the REST Web Service API, the Exago Web Application and an implementation of the Scheduler Queue (if using the Scheduler Service for executing reports on a scheduled basis) increasing the redundancy of each of these components.

Intranet



The **Intranet** zone contains the servers and databases accessed by the Exago Web Application, the Scheduler Services (via the Scheduler Queue) and perhaps the host app. Servers in the Intranet zone are not directly accessible to the end user's browser. In the diagram above from top-to-bottom, left-to-right:

- **State Server** â€" maintains end user browser sessions in a database so that all instances of Exago in the web farm can access session state. The state server communicates with all of the instances of the Exago Web Application in the DMZ.
- **SQ DB** â€" short for **Scheduler Queue Database**, this is where the job information is centrally stored by the Scheduler Queue. Each instance of the Scheduler Queue, either in the DMZ or Intranet communicate with this database.
- **SQ** â€" short for **Scheduler Queue**. It is connected to each Scheduler Service and to each instance of the Web Application. Its function is to liaise between the Web Application server, Scheduler Services and SQ DB.
- **DB** â€" short for **Database**. Also known as the **Business Intelligence Database**. This is the data source that Exago queries to build reports. Both the Scheduler Services and the Remote Execution services will query this database when generating reports. If Remote Execution servers are not used, then the Web Application servers will communicate with the database directly instead (not pictured in the diagram).
- **Exago R.E. Scheduler Services** â€" are the Scheduler Services designated as **Remote Execution** servers. Whenever a user executes a report in the Web Application or through the REST Web Service API it will be executed by one of these servers.
- **SM** â€" short for **Storage Management**. This is the content storage for all user created content such as reports, themes, folders and templates.
- **Temp** â€" this is the temporary file storage hosted either by a cloud storage provider or a redundant network storage system
- **Exago Scheduler Service** â€" one or more Scheduler Services

Bringing the Pieces Together

By making each component in the system redundant and distributed, single points of failure are eliminated. A failure of one or more components doesn't necessarily take the entire system down. Instead, the work is divided among the remaining parts of the system until failed parts are restored.

For each redundant component in the system, the system is available in a worst-case scenario even if all but one of those components fails. As an example, in an environment with *n* number of Exago servers, there can be *n-1* failures before a total outage occurs. Of course in this situation, *all* requests would be forwarded to a single server which may not be able to handle that level of activity.

> **IMPORTANT**
>
> As there can only be one load balancer, it remains a potential failure point. Generally load balancers are simple and very reliable.

The examples below walk through the flow of information into the different zones:

Example 1

Consider a user interacting with the host application and Exago. When the user runs a report, at a very high level the following sequence of actions takes place:

1. Requests are sent from the user's web browser to the host application and to the Exago server web farm via standard HTTP ports.
   - The host app will make requests to the Exago server web farm via the REST Web Service API as well. This also occurs over standard HTTP ports.
2. The **load balancer** in front of the Exago Servers distributes the requests from the user's web browser and host app to the available instances of the Exago Web Application and the REST Web Service API. Since this is a web farm, the requests originating from one user's browser don't necessarily end up at the same server â€" the load balancer determines where they go.
3. Each **Exago server** checks with the **State Server** in the Intranet to track the state of the user's session. Since each Exago server references the same state server, it doesn't matter which Exago server handles a particular request. The work of the load balancer is transparent to the end user because of the state server.
4. The **Exago server** then relays the execution of the report to one of its configured **Remote Execution** servers. As there should be several remote execution instances, work will be assigned in a round-robin fashion to the next one.
5. The **Remote Execution** server queries the **Business Intelligence Database** to build the report. **Temporary files** generated as part of the report execution are written to the **Temporary storage location**, making them available to the other servers.
6. The completed report is handed back to the calling Exago Web Application Server in the DMZ. The Web Application Server sends the completed report to the end user's browser.

On the next interaction from the user, this process starts all over again from step 1. Since requests are handled by the load balancer each time a request is sent from a user, every single request could be handled by a different Exago Web Application server. The state of each user session is handled by the state server, and temporary files are stored in a location accessible to all Exago servers.

Example 2

Taking Example 1 a bit further, consider a user scheduling a report to run a payroll report on the first of every month.

1. Requests are sent from the user's web browser to the host application and directly to the Exago server web farm in the DMZ via standard HTTP ports.
    - The host app will make requests to the Exago server web farm via the REST Web Service API as well. This also occurs over standard HTTP ports.
2. The **load balancer** in front of the Exago Servers distributes the requests from the user's web browser and host app to the available instances of the Exago Web Application and REST Web Service API. Since this is a web farm, the requests originating from one user's browser don't necessarily end up at the same server â€" the load balancer determines where they go.
3. Each **Exago server** checks with the **State Server** in the Intranet to track the state of the user's session. Since each Exago server references the same state server, it doesn't matter which Exago server handles a particular request. The work of the load balancer is transparent to the end user.
4. The **Exago server** makes a call to the **Scheduler Queue** to schedule the new job. The Scheduler Queue stores the job in its database. From here on out, the Scheduler Queue is responsible for the management and execution of the job.
5. **Scheduler Queue** responds to the **Exago server** indicating the scheduling of the job was successful, message is displayed to the user.

From this point, the interaction is completed. From the Scheduler Service's point of view:

1. At the scheduled time, the **Scheduler Queue** will send the job off to one of the available **Scheduler Services** for execution.
2. The **Scheduler Service** queries the **Business Intelligence Database** to build the report. **Temporary files** generated as part of the report execution are written to the **Temporary storage location**, making them available to the other servers.
3. The Scheduler Service handles the report disposition (either by e-mailing or saving it to a repository).

Conclusions

In summary, Exago-specific points of high availability are:

- Multiple Exago servers, each containing an instance of the Web Application, the REST Web Service API and a Scheduler Queue (if using Scheduler Services) should be setup in a web farm
- Temporary files and the Exago configuration file should be shared between instances with either a networked drive or cloud storage
- Use highly available databases wherever possible for:
    - Exago's Scheduler Queue
    - State Server shared between the multiple Exago servers
    - Storage or Folder Management for content storage

Additional Resources

- Admin Support Lab â€" High Availability (video)

Updating Recommendations

Updating can be done very quickly. Due to the fact that users rely on the insights Exago produces, it is important to establish a meticulous process to assure their experience is fluid and uninterrupted. Below is the process we recommend when updating.

Read the Release Notes

The Updating to the Latest Version article lists breaking changes that will highlight any actions beyond the normal updating process that are required.

For all releases, the full change log is listed below the installer download on the Support Center.

Three-Environment Updating

We recommend using at least three environments when updating, consisting of **Development**, **Quality Assurance** (QA), and **Production**. In the Development environment, a developer would first update Exago on their local machine or a testing server. After testing the new features and verifying issue fixes, they would update a QA environment for more complete testing, likely implementing a **User Acceptance Testing** (UAT) server. Once the QA team has signed off on testing (more details below) Exago can be upgraded on the production servers.

> **TIP**
>
> In supporting multiple environments, it is recommended to externalize environment-specific configurations to config files like web.config for easy re-deployment without code changes.

Steps for Updating Exago

1. Download the Exago installer from our Support Site.
2. Backup the environment
3. Run the installer
    1. Update the Web Application
    2. Update the Web Service API
    3. Update the Scheduler Service
4. Update Storage Management
5. Update extensibility assemblies
6. Update CData driver license
7. Perform recommended testing
8. Update to production

2. Backup

Critical files and any files that may have been manually or programmatically changed should be backed up prior to updating. These files include

but are not limited to:

Scheduler Service

- eWebReportsScheduler.xml
- eWebReportsScheduler.exe.config
- Contents of the working directory

Exago Web Application

- web.config
- appSettings.config

Web Service API

- appSettings.config
- Config/WebReportsApi.xml

Monitoring Service

- Monitoring.exe.config
- Monitoring.sqlite

Customized Files and Reports

Any custom language files, images or CSS files being kept directly within the Exago installation should be backed up as well.Â Any files added to bin directories should be safe during updates but may still need to be updated.

Report definitions should be backed up. Reports created in the previous version will be updated with changes for the new version and may no longer work in the previous version.

3.1. Run the installer to update the Web Application

It is safe practice to shut down associated application pools and web sites (IIS), or similar in Apache, as well as associated Scheduler and Monitoring Services in order to relinquish handles on incorporated DLLs before running the installer. There is no added benefit in updating while the service is running.

> **NOTE**
>
> The update can be installed over the existing version.

If you are using a custom Application Theme, make sure to recreate it in order for it to pick up the latest Exago icons and CSS. In *v2020.1.3+*, use **Admin Console** > **General** > **Feature/UI Settings** > **Update All Application Themes** to quickly update all available themes.

3.2. Run the installer to update the Web Service API

If your installation of Exago utilizes the Web Service API, please update this as well.

> **IMPORTANT**
>
> If you are using the REST Web Service API, make sure to add the REST key back to appSettings.config (or restore from your backup in step 2) as it will be overwritten.

3.3 Run the installer to update the Scheduler Service

Ensure that all the settings in eWebReportsScheduler.xml are correct.

If applicable, copy any custom language files into the Language folder of each **Scheduler Service** installation.

4. Update Storage Management v2020.1+

If the new version of Exago has been installed to a different directory than the previous, and the configuration file will be copied from one install to the other, the **Admin Console** > **Storage Management** > **Assembly Location** will need to reference the new location.

Between versions, some changes may be made to the Storage Management database schema. A new **Admin Console** > **Storage Management**Â >**Update Reports** button will update report content records with the new schema as necessary. This button may also be “clicked programmatically” with the .NET API:

```
Api api = new Api("appPath");
String updateResults = (new WebReports.Api.ReportMgmt.UpdateReportsUtility(api.PageInfo)).RewriteAllReports();
```

If upgrading from a previous version of Exago without Storage Management, or there are new element themes, import them to the Storage Management database with one of the transitioning utilities or the **Load Themes** button in the Admin Console.

For more information about the returned string, refer to Storage Management.

5. Update Extensibility Assemblies

1. Review the Updating to the Latest Version article for any breaking changes that might be encountered in your assembly. Change code as necessary.
2. Update references:
    1. If your installation uses the .NET API, update the references to these Exago libraries within host application and perform a Rebuild:
        1. WebReportsApi.dll
        2. ExagoPuppeteerRasterizer.dll
    2. If your installation uses a .NET Assembly as a data source, update the reference to WebReportsAsmi.dll within the application launching Exago and perform a Rebuild.
    3. If your installation uses a .NET Assembly for any Extensibility Features (Folder Management, Server Events, Custom Functions, Custom Filter Functions, Custom Aggregate Functions): Update the reference to WebReportsApi.dll within the application launching Exago and perform a Rebuild. Update the appSettings.config and web.config files in Exago if they are utilized by Extensibility.
3. Ensure that all of the extensibility features in Exago are referencing the correct version of your .NET assembly.

Exago will throw an error as soon as the application is entered if assemblies aren't re-compiled.

6. Update CData Driver

Clients who are utilizing a CData driver to connect to certain data sources and have updated the **driver** from one major version to another (e.g. from 2018 to 2019) will need to have the license key re-generated. The Exago provided license key is valid only for the major CData driver version that applied when it was issued.

7. Recommended Testing

While our QA team does extensive testing and runs automated tests on every released build, we still recommend that your team performs some basic checks as due diligence. Particular areas of focus we suggest include:

- **API integration:**
    - Verify Exago launches successfully and the appropriate permissions, with a focus on data and multi-tenancy, are respected.
- **Features that utilize extensibility of Exago, such as:**
    - Report Folder Management
    - Server & Action Events
    - Custom Functions
    - Custom Filter Functions
    - Custom Aggregate Functions
- **Key reports and Dashboards:**
    - This can be as simple as running a number of "standard" reports that all users may access or involve exporting reports to excel, pdf, or csv output and comparing the results with a stored master from a previous Exago version.

It is also recommended to ensure that each instantiated Role's functionality is still in place, and that Exago isÂ run with the browser's Developer Tools in order to ensure no JavaScript errors are happening behind the scenes.

8. Updating to Production

Moving to production follows the same process as updating a UAT or staging environment, but involves a few additional steps:

- Only update production during scheduled downtime, as updating Exago or changing the Config XML file in any way will invalidate current active user sessions.
- Only bring the encrypted .xml.enc files into the production environment.
- Ensure that the **Admin Console** is removed from the environment.

A full security checklist can be found here.

Updating to the Latest Version (Potentially Breaking Changes)

When updating Exago, consult this guide for potential breaking changes. Note that when updating from more than one version behind the current one, information from that section and all sections above it will apply.

> **NOTE**
>
> Please also review the **Updating Recommendations** article to ensure a smooth updating process.

If the version is not listed, then a direct update path to the latest version is not supported. Please file a support request for further assistance.

> **NOTE**
>
> To ensure a smooth upgrade when overwriting a previous installation, ensure that the web server (IIS/Apache/NGINX), the **Monitoring Service**, and any **Scheduling Services** are turned off before running the installer.

User Interface, API and Breaking Changes

Updating from v2021.1 or earlier

This section applies to any maintenance release version of *v2021.1*.

Today() and Now() Date Functions Use the End-User Locale

Prior to *v2021.2.0*, the return value for these functions was based off the server's time. Now, the return value is based on the end-users locale, set by the Culture Settings in the user session. For more information, see Culture Settings and Date Functions.

There are some additional effects of this change:

- The **Admin Console** > **General Settings** > **Culture Settings** > **Server Time Zone Offset** no longer accepts null as a value.
- A newÂ hidden flag named **UseExternalTimeZoneConverter** when *True* takes the place of setting **Server Time Zone Offset** to null. This should only be set to *True* to support legacy use of the External Interface, otherwise it should be left as as *False*. The External Interface is a deprecated extensibility feature. However, those using the External Interface to do time zone conversion need to adjust the **Server Time Zone Offset** and setting **UseExternalTimeZoneConverter** *True*.

Updated Snowflake CDATA driver

The Snowflake CDATA driver has been updated from version 20.0.7768.0 to version 21.0.7887.0. Clients using the Snowflake CDATA driver need to contact their Customer Success Manager to have a new license key generated.

New dbconfigs.json and dbconfigs.overrides.json Schema for the FormulaDictionary

The schema for the **FormulaDictionary** section of the dbconfigs.json and dbconfigs.overrides.json files has changed to allow for multiple properties to be included for some of the function translations. Environments that have included customization for the SQL translations with the dbconfigs.overrides.json should review the **Writing SQL Translations** section of the **Database Settings** and Managing the dbconfigs.json File articles for more information.

Change to Column Alias Handling in SQL Generation

During SQL generation, Exago assigns column aliases in the SELECT clause. In *v2021.1.0,* the SQL generation engine uses those column aliases in the ORDER BY clause. This results in more compact and neat SQL generated.

> **NOTE**
>
> The column aliases set as part of the Column Metadata are different from the column aliases that the SQL generation engine uses.

Example of SQL Generated Prior to *v2021.1.0*

```
SELECT DISTINCT (CASE WHEN ((Northwind.[CATEGORIES].[CATEGORYID] = 3) OR (Northwind.[CATEGORIES].[CATEGORYID] = 5)) THEN 1 ELSE 0 END) AS c0 FROM Northwind.[C
```

Example of SQL Generated After *v2021.1.0*

Note in this example, that the column alias **c0** for the SELECT clause is then referenced in the ORDER BY clause, instead of repeating the entire SELECT clause.

```
SELECT DISTINCT (CASE WHEN ((Northwind.[CATEGORIES].[CATEGORYID] = 3) OR (Northwind.[CATEGORIES].[CATEGORYID] = 5)) THEN 1 ELSE 0 END) AS c0 FROM Northwind.[C
```

Custom Function Category Names Must Be Translated

When creating Custom Functions or Custom Aggregate Functions with a custom category name via the Admin Console, the category name must be specified by a language file ID. Put another way, names for custom category names must exist in the language files, and using a static text string in the **Specify Category Name** field is not acceptable. Any environment utilizing a custom category name should immediately translate the category name and update the configuration to use the language file ID instead. Leaving static text strings for a custom category name can cause parts of the application to stop working.

For more information, see Multi-Language Support, Custom Functions andÂ Custom Aggregate Functions.

Updating from v2021.1.16 or earlier

New Hidden Flag to Stop Export if File Size is Exceeded

When a report is exported, either through the user interface, or with a GetExecute() method, if the size of the file exceeds value of the new hidden flag <general><maxexportfilesizebytes></general> number of bytes, report execution is cancelled and a message is displayed to the end user. The default value is *0*, disabling this limit. Any positive integer value is is accepted, including *0*.

Updating from v2021.1.15 or earlier

New Hidden Flag to Stop Report Execution if Database Row Limit is Exceeded

When the **Admin Console** > **General** > **Database Settings** > **Database Row Limit** is exceeded, and the new hidden flag <errorondbrowlimitexceeded> is *True*, report execution is cancelled and an error message is displayed to the user. For scheduled reports, the execution abends and an error message e-mail is sent to the administrator (if so configured). The default value is *False*. For more information, see the Hidden Flags, Database Settings and Scheduler Service Configuration articles.

Updating from v2021.1.14 or earlier

/rest/SchedulesV2 objects updated

A new ParentJobId field has been added to the **Schedule** and **ScheduleListItem** objects in the /rest/SchedulesV2/ endpoint.

Updating from v2021.1.13 or earlier

Optional successCallback Available for Some JavaScript API Functions

The **EditReport**, **NewReport**, **ScheduleReportManager** and **ScheduleReportWizard** JavaScript API functions now support an optional successCallback function. For more information, review the JavaScript API article.

Updating from v2021.1.10 or earlier

RedShift CDATA Driver

The Redshift CDATA driver has been updated from version 20.0.7654.0 to version 21.0.7872.0. Clients using the Redshift CDATA driver need to contact their Customer Success Manager to have a new license key generated.

New Language File Element

Added new language file element DashboardVizChartTypeLbl to translate the **Chart Type** label in the Dashboard Designer

JavaScript API ExecuteReport() Function

The ExecuteReport() function in the JavaScript API now supports calling reports by the report ID. To accomplish this, an additional boolean argument has been added to the function call.

Updating from v2021.1.8 or earlier

New Licenses Directory

A new /Licenses directory will be installed with the Web Application installer. This directory contains copies of license agreements for all third party components included with Exago.

Completed Scheduler Service Jobs Now Automatically Flush

Prior to this version, the Scheduler Service's <flush_time> setting was set to *-1*. This disabled automatic flushing of completed jobs. With this version, this setting now defaults to *24*, so completed jobs flush automatically every 24 hours.

Updating from v2021.1.7 or earlier

New Max Chained Report Collation Executions Configuration Setting

A new configuration setting, found in **Admin Console** > **General** > **Other Settings** > **Max Chained Report Collection Executions** places a limit on the number of individual report executions caused when executing a collated chained report. For more information how this setting works, review Chained Reports and Other Settings.

This new setting is available in the .NET API General object, the REST Web Service API /rest/settings endpoint and as an XML configuration file node.

Native Support for SQLite as a Reportable Data Source Type

SQLite Data Sources may now be added to the application just as any other supported data source can with an included ADO.NET driver.

Updating from v2021.1.6 or earlier

New maxRasterizationsBeforeReset Application Setting

The Web Application's appSettings.config and the Scheduler Service's WebReportsScheduler.exe.config files now include a new setting called maxRasterizationsBeforeReset. This setting tells Exago when to dispose of Puppeteer chart rendering pages. After this number of executions on the page, the process will be killed and a new one created for further rendering. See Application Settings for more information.

Updating from v2021.1.5 or earlier

Abended Scheduled Jobs now Flushed Automatically

Prior to this version, scheduled jobs that *Abended* would not be cleared when flushing the Scheduler Service job files. After updating, Abended jobs will be available for flushing after the Scheduler Service's <flush_time> timeout lapses.

Schema Access Type Now Honored for Vertical Tables

Prior to this version, Exago always queries the Data Source for the schema of a Vertical Table, regardless of the setting of the **Schema Access Type** dropdown for the Data Object. After updating, if the **Schema Access Type** is *Metadata*, the schema will be read from the configuration file. This has the potential to break reports if they are referencing columns not included in the object's metadata.

After updating, verify that all column metadata exists for all columns of a vertical table.

Updating from v2021.1.4 or earlier

Report Viewer Interactive Dock Change

The interactive dock in the Report Viewer now defaults to the right side of the screen when opened. This setting can be reverted back to the previous left-hand position with the **Admin Console** > **General** > **Feature/UI Settings** > **Interactive report viewer default dock placement** setting.

Updating from v2021.1.1 or earlier

Scheduler Service Configuration File Change

The <default_job_timeout> setting in the Scheduler Service Configuration File is now deprecated. Use <max_job_execution_minutes> exclusively to limit schedule execution.

A new setting, <smtp_timeout> is now available to set the SMTP server connection timeout in place of <default_job_timeout>.

Updating from v2021.1.0 or earlier

This section applies when upgrading from any maintenance release version of *v2021.1*.

/rest/Schedules Replaced

The /rest/Schedules endpoints are now deprecated.

Clients using the REST Web Service API to create, modify or delete scheduled reports should upgrade to the new /rest/SchedulesV2 endpoints as soon as possible. The legacy /rest/Schedules endpoints will be removed in a future version of Exago.

Allow Editing ExpressView with Live Data now defaults to *True*

The default value for **Admin Console** > **General** > **Feature/UI Settings** > **Allow Editing ExpressView with Live Data** has been changed from *False* to *True*.

Updating from v2020.1 or earlier

This section applies when upgrading from any minor version of *v2020.1* (e.g. *v2020.1.0*, *v2020.1.5*, etc…)

Storage Management Changes

New content.associated_reports Column

A new associated_reports column has been added to the Storage Management content table, containing a comma-separated list of content_ids for each report that is associated with a particular content record.

Reports become associated with others when they are components in a composite report such as Chained Report or Dashboard, or if an Advanced Report contains links to other reports (a.k.a. drilldowns).

The **Update Reports** function in the Admin Console reads the contents of a Storage Management database, parses the report file contents and generates the comma separated lists of **associated_reports** where applicable.

Clicking the **Update Reports** button in the Admin Console is a required part of the upgrade process for all clients when updating to *v2021.1*.

Access Flags Active

The **access flags** built in to the **Storage Management** system have been made active. In addition, the access flags themselves have been changed from their original definition in the *v2020.1* release.

Review the Storage Management: Introduction and Storage Management: Database Schema articles for a definition of the new access flags and their locations in the bitmap.

New **Default Access Permission** controls in the **Admin Console** allow an administrator to set the default permissions when new content is created. This value can be changed with the .NET or REST Web Service API or by changing the configuration file.

As some of the access flags bits have changed meaning, some features in the application may inadvertently become unavailable or vice versa. The following SQL statement may be run to upgrade all reports in any existing database to keep full access prior to the upgrade:

```
UPDATE content_access SET access_flags = 2047 WHERE access_flags = 511
```

Review the table below for a comparison of the flag bit definitions:

v2020.1	v2021.1
reserved for future use	reserved for future use
reserved for future use	reserved for future use
reserved for future use	reserved for future use
reserved for future use	reserved for future use
reserved for future use	reserved for future use
reserved for future use	CanMove
reserved for future use	CanSchedule
CanView	CanView
CanDownload	removed â€" reserved for future use
CanCopy	CanCopy
CanExecute	removed â€" reserved for future use
CanDelete	CanDelete
CanShare	CanShare
CanRename	CanRename

CanSave	removed â€" reserved for future use
CanEdit	CanEdit

The ContentPermission enum type is also affected by this change.

-o LoadReportsToDb Command Line Switch Added

A new -o command line switch has been added to the **LoadReportsToDb** Storage Management transitioning utility. This allows a user to specify a value for the database driver's CommandTimeout property. Review LoadReportsToDb in the Storage Management: Transitioning from Legacy Storage Methods article.

ExpressView Changes

ExpressView Themes

With the changes to the the **ExpressView Designer**, all new themes have been created. Legacy ExpressView themes are no longer supported.

The new ExpressView themes will need to be loaded into the Storage Management database. For existing installations, this can be done in the **Admin Console** > **Storage Management** by clicking on the **Load Themes** button. Skipping this step will result in there being no themes available in the ExpressView Designer, even if themes were available before updating.

Creating custom ExpressView themes is possible, by manually editing the JSON theme content. The schema has changed, but is documented in the Create a Custom ExpressView Theme article. It is not possible to create an ExpressView theme from the designer.

To remove the legacy themes from the Storage Management database (not required), execute the following SQL statements:

```
DELETE FROM content_access WHERE content_id IN (SELECT content_id FROM content WHERE content_attribute = â€˜ExpressViewâ€™);
```

and then:

```
DELETE FROM content WHERE content_attribute = â€˜ExpressViewâ€™
```

Formatting

In addition to the theme changes noted above, the content formatting controls (e.g. bold, italic, underline, alignment, etc…) have been re-worked.

Opening an ExpressView in the new designer will reset all formatting to the system defaults, where any customization can be reapplied as desired with the tools available. Some styling options have been changed to a report-wide setting (e.g. font type) or are no longer available (e.g. font size)

Tutorial Mode and Hints Removed

The **Tutorial Mode** and **ExpressView Hints** have been removed from the application, and their Admin Console and configuration file nodes have been eliminated.

Maximum of 7 ExpressView Groups

The ExpressView Designer will now enforce a limit of seven **group columns** on a single report for performance. The 7th group will need to be demoted to a detail field before it can be replaced with a new one.

ExpressViews created in previous versions with more than 7 groups will still be editable and executable in *v2021.1*, but no more groups may be added until the number of groups is less than 7.

Default Filter Execution Window Change

The default value for the **Admin Console** > **General** > **Filter Settings** > **Default Filter Execution Window** has changed from *Standard* to *Simple Without Operator*.

In environments where no value had been previously set for this setting, the *Simple Without Operator* filter execution window will be displayed for all Advanced Reports, Dashboards and Chained Reports with prompting filters after this update.

This setting can be overridden on the report-level (if **Admin Console** > **General** > **Filter Settings** > **Allow User to change Filter Window** is *True*) by changing the **Filter Execution Window** setting in the Report Designer's **Advanced** > **Report Options** dialog, or on an application-wide level by changing the aforementioned Admin Console setting.

Learn more in the Filter Settings article.

Advanced Report Changes

Advanced Report Wizard Removed

The **Advanced Report Wizard** has been removed from the application. Creating a new Advanced Report from the main menu or Getting Started page will directly start the Advanced Report Designer and prompt the user to add Data Objects (formerly Categories) to the report to begin report building.

The **helpKeys** related to the wizard for creating Custom Context-Sensitive Help have also been removed from the application.

Toolbar Changes

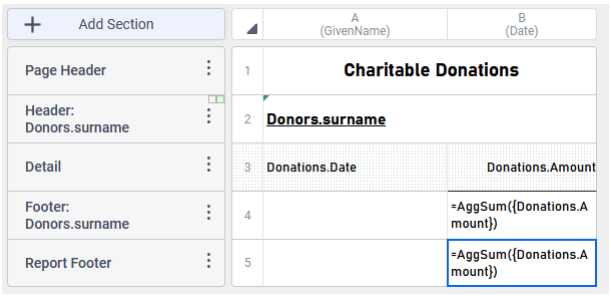
AutoSum Σ Icon Now Hidden by Default

The **AutoSum** Σ

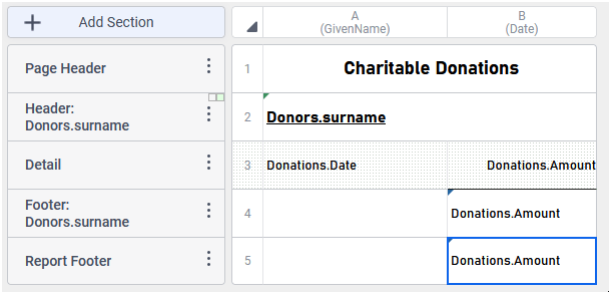
icon from the Advanced Report Designer's toolbar is now disabled by default. A new configuration setting, **Admin Console** > **General** >

Feature/UI Settings > **Show Auto Sum Button** allows the button to be enabled on an application-level or session-level basis. The default value is *False*.

Users are encouraged to use aggregate functions in place of the AutoSum feature. For example, these two reports produce the same output:



Using the AggSum aggregate function to compute the sum of the Donations.Amount data field on the Group and Report level



Using the AutoSum feature (toolbar icon not shown) to compute the sum of the Donations.Amount data field on the Group and Report level

The **AutoSum** Σ

icon will be deprecated in a future version.

Actionable User Interface Item Changes

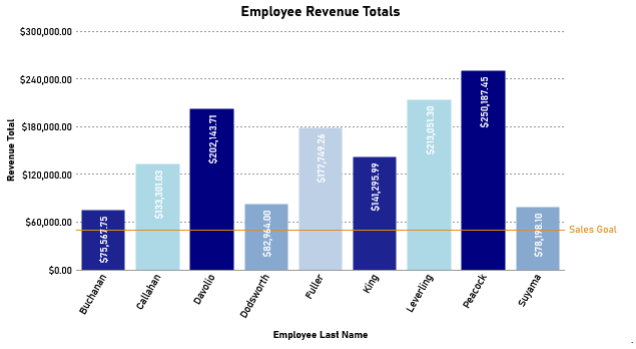
Some of the Actionable User Interface Items on the toolbar have either been removed or replaced. Review the Actionable UI Elements article to see all of the changes. Action Events that use these items may break if they reference elements with new names or elements that are no longer available.

Default Theme Colors Changed

The colors of the *Default* theme have been changed. This new theme is WCAG AA Compliant and color blind accessible.

Single Series Chart Theme Change

New themes have been introduced for charts. In addition, theme color now corresponds with a series, instead of a value. Bar, Stacked Bar, Column, Stacked Column, Pareto, Spark Column, Scatter, Zoom Scatter and Bubble Charts are affected by this change.



A single series column chart *pre-v2021.1*

ETL Removed

The **ETL** feature has been removed.

Review Configuration Changes at the bottom of this article for notes about Scheduler Service and Web Application configuration file nodes that have been removed.

The folder with the ETL Transforms may be safely removed. This code sample can be used to remove the ETL_TRANSFORMS folder and any transforms in it on a Microsoft SQL Server Storage Management database. Run the statements one at a time.

> **NOTE**
>
> Change both instances of *ETL_TRANSFORMS* in the WHERE causes below to the name of the actual ETL Transforms directory.

This can be found in the **Admin Console** > **General** > **ETL Settings** > **ETL Transforms Folder** setting.

Change the names of the **content** and **content_access** tables in the statements below to the actual names of those tables in the database, where applicable.

```
-- Add a flag column
ALTER TABLE content ADD ETL_FLAG INT
```

— Set the flag for content in the specified folder
UPDATE content SET ETL_FLAG=1 FROM content WHERE content.content_id IN (SELECT content_id FROM content_access WHERE parent_id = (SELECT content_id FROM content WHERE name = 'ETL_TRANSFORMS'))

— List the items we are going to delete
SELECT * FROM content WHERE ETL_FLAG = 1

— Delete all the content access records for content in the folder
DELETE FROM content_access WHERE parent_id = (SELECT content_id FROM content WHERE name = 'ETL_TRANSFORMS')

— Delete the content records for those items flagged which don't have another access record
DELETE FROM content FROM content INNER JOIN content_access ON content.content_id = content_access.content_id WHERE ETL_FLAG = 1 AND content_access.content_id IS NULL

— and report those items we did not delete
SELECT * FROM content WHERE ETL_FLAG=1

The transform folder can now be removed from the database.

Updating from v2020.1.22 or earlier

New Language File Element

Added new language file element DashboardVizChartTypeLbl to translate the **Chart Type** label in the Dashboard Designer

JavaScript API ExecuteReport() Function

The ExecuteReport() function in the JavaScript API now supports calling reports by the report ID. To accomplish this, an additional boolean argument has been added to the function call.

Updating from v2020.1.15 or earlier

Schema Access Type Now Honored for Vertical Tables

Prior to this version, Exago always queries the Data Source for the schema of a Vertical Table, regardless of the setting of the **Schema Access Type** dropdown for the Data Object. After updating, if the **Schema Access Type** is *Metadata*, the schema will be read from the configuration file. This has the potential to break reports if they are referencing columns not included in the object's metadata.

After updating, verify that all column metadata exists for all columns of a vertical table.

Updating from v2020.1.13 or earlier

Scheduler Service Configuration File Change

The <default_job_timeout> setting in the Scheduler Service Configuration File is now deprecated. Use <max_job_execution_minutes> to limit schedule execution.

A new setting, <smtp_timeout> is now available to set the SMTP server connection timeout in place of <default_job_timeout>.

Updating from v2020.1.12 or earlier

/rest/Schedules Replaced

The /rest/Schedules endpoints are now deprecated.

Clients using the REST Web Service API to create, modify or delete scheduled reports should upgrade to the new /rest/SchedulesV2 endpoints as soon as possible. The legacy /rest/Schedules endpoints will be removed in a future version of Exago.

Updating from v2020.1.1 or earlier

- The **MongoDB CDATA** driver has been updated from version 19.0.7202.0 to version 20.0.7524.0. Clients using the MongoDB CDATA driver need to contact their Customer Success Manager to have a new license key generated.
- A bug fix to the /rest/Schedules/ endpoint now includes the full report path when making standard and BATCH calls on Linux. This change only affects Linux, as the full report path was always returned on Windows. For example:

  ```
  "ReportName": "Parameter Report"
  ```

  Linux payload prior to change

  ```
  "ReportName": "My ReportsAdvanced ReportsParametersParameter Report"
  ```

  Linux payload after the change

Updating from v2019.2 or earlier

This section applies when upgrading from any minor version of *v2019.2* (e.g. *v2019.2.0*, *v2019.2.11*, etc…)

Storage Management

Exago's **Storage Management** system replaces the legacy file system, cloud storage, folder management and web service (SOAP) storage methods with a relational database. All reports, templates, folders and themes are stored in this database. Out-of-the-box, Exago provides a SQLite database. Microsoft SQL Server, MySQL, Oracle and PostgreSQL are also supported.

As of *v2020.1*, the legacy storage methods are deprecated and have been replaced with the Storage Management system. All clients upgrading to *v2020.1* will need to change their content storage strategy to the Storage Management system. Exago has built-in several handy utilities for this transitioning process, as well as an interface for implementing a customized Storage Management system.

More information about Storage Management and transitioning a legacy storage system to it can be found in these articles:

- Storage Management: Introduction
- Storage Management: Getting Started
- Storage Management: Transitioning from Legacy Storage Methods
- Storage Management: Custom Implementation

User Identification and Permissions

When creating an Exago user session with one of the APIs, in addition to setting the @ and @ system parameters, there are four new Storage Management Identity keys that should also be set.

Details about these keys and how to set them can be found in the Storage Management: Introduction and Navigating the Application articles.

When transitioning to Storage Management, a new permissions model is available. Learn more in Storage Management: Introduction and Storage Management: Getting Started.

User Interface Changes

The Storage Management system is virtually transparent to the end user, with two exceptions:

1. A new **Owner** icon will appear in the Report Tree if a user owns that content. Content ownership is a new concept in the Storage Management system. More information can be found in the Storage Management: Introduction link above.
2. If the Owner of content moves it to a different folder, a new dialog will appear asking the user to copy the permissions of the new folder to the content or to keep the permissions of the content.

More information can be found in the Navigating the Application article.

Caching Settings Moved

Several settings for caching in the applications have been moved in the **Admin Console** and configuration file.

- **General** > **Other Settings** > **Enable Report List Caching** has moved to **Storage Management** > **Enable Report List Caching**
- **General** > **Other Settings** > **Enable Report XML Caching** has moved to **Storage Management** > **Enable Report XML Caching**
- **General** > **Other Settings** > **Report XML Caching Timeout (s)** has moved to **Storage Management** > **Enable Report XML Caching Timeout**

New Caching Setting

The Theme List is now cached with the Storage Management system. A new **Storage Management** > **Theme List Caching Timeout** setting determines when the theme list cache expires.

Puppeteer Rendering Libraries Enabled by Default

The **Puppeteer** visualization rendering libraries were introduced into Exago *v2019.1.5* but were not enabled by default. Now in *v2020.1*, they have been enabled for all installations by default in place of the legacy **wkHtmltoImage** libraries.

Clients using Docker containers or Azure App Services may need to flip the `useWkHtmlToImage` flag back to *True* in the Application Settings for both the Web Application and the Scheduler Service as Puppeteer is not compatible with these types of environments yet.

Clients using the .NET API to integrate Exago need to add a reference to `ExagoPuppeteerRasterizer.dll` located the Exago Web Application's `bin` directory to their application.

Linux Installer No Longer Includes MySQL ADO.NET Driver

The Exago Linux Installer will no longer install a MySQL ADO.NET driver at the time of installation. Instead, clients wishing to use a MySQL data source for either reporting or for Storage Management will need to provide their own.

Exago has provided wrappers around two popular MySQL data drivers that clients may choose to install on their own.

- Devart dotconnect free edition
- MySQL ADO.NET

Contact your Customer Success Manager or the Exago Support Team for assistance. Install the driver by extracting the contents of the download and then running either `installMySql.sh` or `installDevartMySql.sh` as root. Provide the Exago installation path to the installer script. For example:

```
sudo ./installDevartMySql.sh /opt/Exago
```

Once installed, update the **Admin Console** > **General** > **Database Settings** to reflect the new data provider.

System.Data.OracleClient Driver No Longer Supported on Linux

The System.Data.OracleClient driver, which also required the external Oracle Command Line Interface to be installed, is longer supported on Linux.

The Oracle.ManagedDataAccess.Client driver can be used instead. Exago has provided a wrapper that clients may use to install Oracle.ManagedDataAccess.Client on their own.

Contact your Customer Success Manager or the Exago Support Team for assistance. Install the driver by extracting the contents of the download and then running installOracle.sh as root. Provide the Exago installation path to the installer script. For example:

```
sudo ./installOracle.sh /opt/Exago
```

Once installed, update the **Admin Console** > **General** > **Database Settings** to reflect the new data provider.

Several Action Events Now Support Promises

Synchronous calls to Utilities.AjaxRequest are no longer supported. As a workaround, the Action Events below have been upgraded to allow returning Promises, allowing asynchronous calls instead:

- OnRightClickReport
- OnRenameReport
- OnEditReport
- OnDeleteReport
- OnDuplicateReport
- OnExecuteReport
- OnSelectReport
- OnDoubleClickReport

Action Event Code that makes synchronous calls to Utilities.AjaxRequest such as this example:

```
// Do some stuff
const value = Utilities.AjaxRequest(Utilities.AjaxRequest("web.method.name", data);
// Do some other stuff
```

Will need to be changed to make the call asynchronous:

```
// Do some stuff
Utilities.AjaxRequestAsync("web.method.name", data).then((value) => { /* Do some other stuff */ });
//Do some other stuff
```

or

```
// Do some stuff
const value = await Utilities.AjaxRequestAsync("web.method.name", data);
//Do some other stuff
```

> **NOTE**
>
> If a success callback is passed to Utilities.AjaxRequest as in the example below, this already behaves asynchronously so no change is needed.
>
> ```
> Utilities.AjaxRequest(Utilities.AjaxRequest("web.method.name", data, () => console.log("It worked!"));
> ```

For more information, review the Global Action Events article.

Updating from v2019.2.40 or earlier

Added new language file element DashboardVizChartTypeLbl to translate the **Chart Type** label in the Dashboard Designer

Updating from v2019.2.34 or earlier

Schema Access Type Now Honored for Vertical Tables

Prior to this version, Exago always queries the Data Source for the schema of a Vertical Table, regardless of the setting of the **Schema Access Type** dropdown for the Data Object. After updating, if the **Schema Access Type** is *Metadata*, the schema will be read from the configuration file. This has the potential to break reports if they are referencing columns not included in the object's metadata.

After updating, verify that all column metadata exists for all columns of a vertical table.

Updating from v2019.2.30 or earlier

Scheduler Service Configuration File Change

The <default_job_timeout> setting in the Scheduler Service Configuration File is now deprecated. Use <max_job_execution_minutes> to limit schedule execution.

A new setting, <smtp_timeout> is now available to set the SMTP server connection timeout in place of <default_job_timeout>.

Updating from v2019.2.28 or earlier

/rest/Schedules Replaced

The /rest/Schedules endpoints are now deprecated.

Clients using the REST Web Service API to create, modify or delete scheduled reports should upgrade to the new /rest/SchedulesV2 endpoints as soon as possible. The legacy /rest/Schedules endpoints will be removed in a future version of Exago.

Updating from v2019.2.17 or earlier

- Cloned data objects may now have their own **folder** and **description** fields. Accordingly, there are some minor changes to the Admin Console, XML configuration file, and .NET API handling of data object clones.

Updating from v2019.2.15 or earlier

- The **MongoDB CDATA** driver has been updated from version 19.0.7202.0 to version 20.0.7524.0. Clients using the MongoDB CDATA driver need to contact their Customer Success Manager to have a new license key generated.
- A bug fix to the /rest/Schedules/ endpoint now includes the full report path when making standard and BATCH calls on Linux. This change only affects Linux, as the full report path was always returned on Windows. For example:

  ```
  "ReportName": "Parameter Report"
  ```

  Linux payload prior to change

  ```
  "ReportName": "My ReportsAdvanced ReportsParametersParameter Report"
  ```

  Linux payload after the change

Updating from v2019.2.11 or earlier

The **Admin Console** > **General** > **Culture Settings** > **Time Zone Name** will now be used for all time related features, including DateTime functions such as Now(). **Admin Console** > **General** > **Culture Settings** > **Server Time Zone Offset** only applies if **Time Zone Name** is not set.

Updating from v2019.2.0 or earlier

Linux Installer Flags Added

The **Linux Installer** has had several new command line flags added. Any previous headless install scripts will start prompting for missing formation and will require editing to account for the new parameters.

In summary, the new parameters added are:

- -N will inhibit the installer from configuring systemd services for the Web Application and the Scheduler Service. Otherwise, they will be installed and configured by default. Replaces the -R parameter.
- -u is used to specify an operating system user account under which the systemd services and FastCGI listeners (if applicable) will run.
- -D is used to tell the installer to use default values when installing.
- -f is used to specify names for the FastCGI listener services (NGINX only).
- -p is used to specify the port number for the FastCGI listener services (NGINX only).

Read the Installing Exago on Linux article to review the changes made to the installer and the new options.

api.saveData() is deprecated

The .NET api.saveData() method has been deprecated and replaced by two new methods, SaveConfigToApi() and SaveConfigToFile().

SaveConfigToApi takes the place of a SaveData(false) call and **SaveConfigToFile** takes the place of a SaveData(true) call.

Review the .NET API General Reference article for more information.

//

Updating from v2019.1 or earlier

This section applies when upgrading from any minor version of *v2019.1* (e.g. *v2019.1.0*, *v2019.1.11*, etc…)

Settings Removed From Admin Console

The following settings have been removed from the Admin Console, but do remain available in the XML Config Files in case they are needed. Most of these settings effectively retire the legacy Express Report Designer. Advanced Reports should be used instead.

- **General** > **Feature/UI Settings** > **Allow Creation/Editing of Express Reports**
- **General** > **Feature/UI Settings** > **Use Sample Data for Dashboard Visualization Design**
- all of the **General** > **Feature/UI Settings** > **Express Report Designer Settings**
- **Roles** > **Allow Creation/Editing of Express Reports**
- **Roles** > **Show Styling Toolbar**
- **Roles** > **Show Themes**
- **Roles** > **Show Grouping**
- **Roles** > **Show Formula Button**

Record Level Aggregation Enabled by Default Changed to True

In *v2019.1.8* of the application, the **Record Level Aggregation Enabled by Default** setting was added to the system configuration. This setting configures how Exago handles one-to-many relationships when an Aggregate Function is called without the recordLevel argument. This flag has

been changed to *True* by default in *v2019.2*.

Our documentation on Aggregate Functions describes how the recordLevel argument works and how this change may impact your reports.

Dashboard Visualization Tile Update

Visualization tiles built directly on a Dashboard in previous versions are based on ExpressView. In *v2019.2*, they are based on Advanced Reports. The Dashboard Designer will update the visualizations when a legacy Dashboard is saved for the first time in the *v2019.2+* Dashboard Designer, but this is a one-way update.

To retain the ExpressView appearance and functionality, export the tiles as a standalone ExpressView and replace the tile with an Existing Report tile containing the newly created standalone ExpressView. It is very highly recommended that all legacy Dashboards be backed up before upgrading to the *v2019.2*.

> **NOTE**
>
> If **General** > **Feature/UI Settings** > **Save on Report Execution** in the configuration is *True*, clicking Run in the Dashboard Designer will also save the Dashboard and update the visualizations.

Updating from v2019.1.28 and earlier

The **MongoDB CDATA** driver has been updated.

Updating from v2019.1.25 and earlier

The **Admin Console** > **General** > **Culture Settings** > **Time Zone Name** will now be used for all time related features, including DateTime functions such as Now(). **Admin Console** > **General** > **Culture Settings** > **Server Time Zone Offset** only applies if **Time Zone Name** is not set.

Updating from v2019.1.14 and earlier

WebReports.Api.Reports.Report.GetExecute Methods Will Use Remote Execution by Default

The GetExecute() methods [GetExecuteHtml(), GetExecuteCsv(), GetExecuteData(), GetExecuteJson()] are now able to use Remote Execution. If a Remote Execution host is configured, calls to these methods will use the Remote Execution servers for report generation by default.

To run a report locally, disable Remote Execution for the session via the API.

Updating from v2019.1.4 and earlier

When updating from *v2019.1.4* and earlier, the following topics may require your attention.

.NET Requirements

A minimum of .NET Framework version 4.6.1 is required, with version 4.7.2 recommended. .NET Standard 2.0 is required.

Mono

Exago supports the following versions of Mono:

- 4.2.2.30
- 5.10.1.20

> **WARNING**
>
> These are the only versions of Mono that are compatible with Exago. In addition, the Exago installer will install a patched version of System.Web.dll. This patch can also be installed manually.

Using Puppeteer with .NET API Integration

v2019.1.5 contains a new set of libraries for image rendering called **Puppeteer**. These are turned off by default, and can be enabled by setting the **useWkHtmlToImage** flag in Application Settings to *false*.

If you are using a .NET API integration, in order to use Puppeteer, the following files must be copied to the host application's bin directory:

```
js.es5.resourcejs.es6.resource
bin/ExagoPuppeteerRasterizer.dll
bin/Microsoft.AspNetCore.WebUtilities.dll
bin/Microsoft.Extensions.DependencyInjection.Abstractions.dll
bin/Microsoft.Extensions.Logging.Abstractions.dll
bin/Microsoft.Extensions.Logging.dll
bin/Microsoft.Extensions.Options.dll
bin/Microsoft.Extensions.Primitives.dll
bin/Microsoft.Net.Http.Headers.dll
bin/PuppeteerSharp.dll
```

If your Exago web server is running Linux, the following X11 libraries are required to be installed in order to use Puppeteer:

```
gconf-service libasound2 libatk1.0-0 libc6 libcairo2 libcups2 libdbus-1-3 libexpat1 libfontconfig1 libgcc1 libgconf-2-4 libgdk-pixbuf2.0-0 libglib2.0-0 libgtk-3-0 libnspr4 libpango-1.0-0 lib
```

In a CentOS environment, the following libraries are required to be installed in order to use Puppeteer:

```
pango.x86_64 libXcomposite.x86_64 libXcursor.x86_64 libXdamage.x86_64 libXext.x86_64 libXi.x86_64 libXtst.x86_64 cups-libs.x86_64 libXScrnSaver.x86_64 libXrandr.x86_64 GCon
```

Manual Extraction of chromium-win64.zip Required with ZIP Scheduler Service Installation

When installing the **Exago Scheduler Service** using the Windows ZIP Archive the file chromium-win64.zip must be manually extracted. Extract it to a chromium folder inside of the Scheduler Service's installation directory (e.g. C:ExagoExagoSchedulerchromium). This extraction process happens automatically with the Windows Installer but needs to be done manually when installing from ZIP.

Updating from v2019.1.3 and earlier

When updating from *v2019.1.3* and earlier, the following topics may require your attention.

.NET Framework Version Requirement Raised to 4.6.1

v2019.1.4 contains an upgraded version of the Syncfusion Excel exporter library that requires .NET Framework 4.6.1 or above.

In order to continue to export to Excel via the user interface or API Action, web servers running Exago must have .NET Framework 4.6.1 or above.

If you are using .NET API integration, in order to continue to export to Excel via GetExecute() API methods, web servers running the host application must have .NET Framework 4.6.1 or above. The host application does not need to target this version; it must simply be available on the server. Additionally, the System.ValueTuple.dll binary must be present in the host application's bin directory.

Updating from v2019.1.1 and earlier

When updating from *v2019.1.1* and earlier, the following topics may require your attention.

New method for managing the dbconfigs.json file

The **dbconfigs.json** file contains database specific syntax related to formulas, data types, primary and foreign key discovery, datetime casting, and row limiting and range selection. This file is overwritten after each new installation of Exago and, as such, any changes made to this file will be lost.

In order to prevent the loss of such information after each update, a *dbconfigs.overrides.json* file may be created and populated with these changes. For more information on the **dbconfigs.json** and **dbconfigs.overrides.json** files, please see the **Managing the dbconfigs JSON file** article.

Updating from v2018.2 and earlier

When updating from *v2018.2* and earlier, the following topics may require your attention.

- **Updated Linux Scheduler Installer**
- **Unique IDs and aliases are now required for Data Objects**
- **Time zone enhancements and changes to scheduling**
- **Changes to Entity creation via the .NET API**
- **Recommended configuration architectural changes**
- **New warnings regarding Metadata in the Admin Console**
- **Changes to Active Role setting in the Admin Console**
- **Changes to Null Tenant Parameters**
- **Corrected spelling of REST Folder.Propagate property**
- **Report Management Caching & Database Aggregation enabled by default**

Updated Linux Scheduler Installer

The Linux Scheduler installer has been updated in order to allow systemd compatibility and customization of service names. With these enhancements come a few changes in behavior for Scheduler installation including the replacement of start, stop, restart and status checks with a single new script named exago-scheduler.sh by default. The behavior of the Linux installer will not change for overwriting existing installs; however, any new installation of the Scheduler will function according to these changes. For more information please see the Installing Exago on Linux article.

Unique IDs and aliases are now required for Data Objects

Data Objects are now required to have unique IDs and aliases (names). These IDs will have automatically been assigned if the Data Objects were created using the Automatic Database Discovery tool. However, Data Objects created manually or via the API (as Entities) will need to be manually assigned IDs and aliases.

When upgrading to *v2019.1*, it is imperative that each Data Object is assigned a unique ID and alias if they are not already set. Any new Data Objects created without the use of the Automatic Database Discovery tool must now be assigned unique IDs. If they are not assigned a distinct ID value, A warning message will appear preventing these Data Objects from being created.

Furthermore, any existing API code that dynamically creates new Data Objects should be examined to ensure that unique ID and alias values are being assigned.

Time zone enhancements and changes to scheduling

Enhancements have been made to how Exago considers daylight saving time and other non-linear scheduling issues related to geopolitical time zones. In order to implement these enhancements, the **Client Time Zone Name** setting has been added where the client's geopolitical IANA time

zone may be specified.

Furthermore, changes have been made to scheduling calculations with the inclusion of the Noda Time library within Exago, allowing the **Scheduler** tool and **Scheduler Queue** to accurately consider DST and other time zone related issues.

For detailed information on these enhancements please see the **Time Zone Calculation Enhancements in v2019.1** article.

Upon updating, it is recommended that the **Next Execute Date** of existing Schedules is reviewed for accuracy. If **Client Time Zone Name** is being utilized, existing schedules may need to be manually edited or recreated in order to reflect these changes.

Furthermore, certain DateTime values may have been assigned different meanings in the Scheduler Job XML following the implementation of Noda Time. Depending on the modifications made to your implementation of the **Scheduler Queue**, changes may have to be made to its code:

- If you have not implemented the **Scheduler Queue**, then your *v2019.1* upgrading process will not be affected.
- If your implementation of the **Scheduler Queue** utilizes the QueueAPI object from WebReports.Api.Scheduler, then there should be no required for the **Scheduler Queue** to function properly after upgrading to *v2019.1*.
- If your implementation of the **Scheduler Queue** has been modified to directly parse the Job XML, then some DateTime values may have been assigned a different meaning and will need to be adjusted for. It is recommended that you contact our Services team at for help in determining which values will need to be adjusted.

Changes to Entity creation via the .NET API

In the .NET WebReportsAPI class, the *NewEntity()* function now requires the string parameter *entityName.* This parameter represents the name (alias) of the data object, functionally replacing the Entity.Name argument, and must be unique. The Entity.Name argument, however, is still writable.

Ex.

```
Entity etlEnt = api.Entities.NewEntity("New Entity");
```

Furthermore, the *Add()* function of the *Entities* class requires the new Boolean parameter *addEntityData*, which is set to false by default. In order to add the Entity Data to the local collection, its value must be set to true.

Ex. `api.Entities.Add(etlEnt, true);`

Each instance of the *NewEntity()* and *Add()* functions in use in current .NET API code, must be modified to reflect these changes.

Recommended configuration architectural changes

With *v2019.1*, major architectural changes have been made to the configuration with the intention of optimizing the system. The system has been modified allowing a portion of the configuration to be extracted into a static configuration, which will be loaded into static memory and made available to all sessions. The contents of this parent configuration extend its specified attributes to the child (dynamic) configuration, which will contain the remainder of the configuration information and will be loaded at each user session.

> **CAUTION**
>
> If using a static configuration with one or more scheduler or remote execution services, all static configuration .xml files must be added to the **Config** subdirectory of all scheduler instances.

These optimizations will considerably decrease configuration processing time, as well as decrease overhead in storing and executing scheduled reports, and are strongly recommended. However, these changes are entirely optional and do not need to be implemented. The directive for each configuration file will be defaulted to Dynamic status, requiring no additional set up if these changes are not desired.

New warnings regarding Metadata in the Admin Console

The use of full metadata across all data objects is strongly recommended in *v2019.1* as it will reduce the frequency in which the databases are queried whenever metadata is required. As such, new warning icons will appear in the **Admin Console** next to data objects that do not have metadata properly implemented.

Changes to Active Role settings in the Admin Console

The setting in the configuration indicating the currently active role has changed. The **Active Role** setting under the **Main Settings** of the **Admin Console** retains the Id of the role to be made active upon session start. The active role may now be changed manually using the dropdown.

Changes to Null Tenant Parameters

Following changes made to nullable parameters in *v2019.1*, in order to disable tenancy on a tenant column, the parameter value for each tenant column now must be set to â€œ{disabletenancy}.â€ Previously, to ignore tenancy, this parameter value could be left as an empty string; however, in *v2019.1*, any empty tenant parameters currently in use within the system will be identified as empty string tenant filters.

In order to continue disabling tenancy, the value must be set to â€œ{disabletenancy}â€ across all instances of tenant parameter values set to empty strings. For more information, please see the **Multi-Tenant Environment Integration**Â article.

Corrected spelling of REST Folder.Propagate property

In previous versions of Exago BI, the REST Folder.Propagate property was incorrectly spelled as “Propogate.” This spelling has been corrected and as such will affect any REST API code using the previous spelling of the Propagate property. Any instance of the incorrect spelling of the Propagate property should be corrected after updated.

Report Management Caching & Database Aggregation enabled by default

The report management interface caches (**GetReportList**, **GetReportXml**, and **GetTheme**/**ExistTheme**/**GetThemeList**) and **Aggregate and Group in Database** setting have been enabled by default in newer versions of Exago in order to ensure that clients are reaping the performance

benefits associated with these options without having to explicitly activate them.

> **NOTE**
>
> The **userId** and **companyId** parameters are used in the generation of the unique key created for the report management interface caches of user sessions. In order to take full advantage of report management caching, it is recommended that these parameters are properly set up. For more information, please see the **User Identification**Â article.

Updating from v2018.2.5 and earlier

When updating from *v2018.2.5* and earlier, the following topics may require your attention:

- **New Required Field Added for Custom Functions**
- **LoadAssemblyInExternalDomain Moved to AppSettings**
- **SQLUtils.dll Moved to WebReportsApi.dll**

New Required Field Added for Custom Functions

The **Return Type** field has been added to custom function objects. This field has been added to support the conversion of filter formulas to SQL so that they may be evaluated in the database using the **Convert Filter Formulas to SQL** flag in the **Admin Console**.

> **NOTE**
>
> For more information, please see the **Security Precautions for Database Formulas**Â article.

This field is required for all custom functions, and, upon updating, all custom functions will need to be assigned a return value of either *String*, *Integer*, *Decimal*, or *Date*. Return values for all custom functions already existing before the upgrade will default to *String*.

LoadAssemblyInExternalDomain Moved to AppSettings

The <**loadassemblyinexternaldomain**> tag has been moved to the appsettings.config file and has been defaulted to true. This change will rarely affect Exago instances; however, it is important to take note of its new location.

SQLUtils.dll Moved to WebReportsApi.dll

The SQLUtils.dll file has been removed from the base Exago install and has been built into the WebReportsApi.dll file. In order to continue to use its methods, WebReports.Api.SqlUtils will need to be referenced.

Updating from v2018.2.1 and earlier

New Parameters for Conditional Drop Pin Formatting

The *@* and *@* parameters within the Conditional Drop Pin Formatting option of the Google Maps wizard are no longer valid. These parameters have been replaced with the serializing parameters *@metric_value_<index>@*, which automatically receive the values of each map metric in the order of their creation (e.g., *@*, *@*, and so on).

Any reports containing an interactive map with Conditional Drop Pin Formatting that uses either the *@* or *@* will need to be recreated using the new parameters or an error will be thrown upon execution of the report.

Updating from v2018.1 and earlier

When updating from *v2018.1* and earlier, the following topics may require your attention:

- **New uses of User Preferences**
- **New Configuration Settings**
- **Disable the old Monitoring Service**
- **Backup settings files**
- **'Name' Property of Filter Objects Deprecated**

New uses of User Preferences

The new **Tutorial Mode** feature relies upon User Preferences. Depending on your User Preference Storage Method you may or may not need to take action:

- *None*: No action needed but **Tutorial Mode** will be disabled.
- *Local Storage*: No action will be required.
- *External Interface or Server Events*: No action required if you are following the sample code provided by Exago. If you have made customizations to the *GetUserPreferences()* or *SetUserPreferences()* method, we recommend reviewing that code to make sure it properly handles the new values being passed.

New Configuration Settings

With *v2018.2* there are new configuration settings for Exago. Below weâ€™ve listed those that are enabled by default and may warrant consideration or action as you upgrade:

<Showtipsexpressview> and *<Showtutorialexpressview>*:

These settings enable the new **Tutorial Mode** feature. They are True by default but will have no impact if your User Preference Storage Method is

set to *None*. Be sure to set values for the Parameter userId or disable these features.

See the section **Configuration Changes** below for a complete list.

Disable the old Monitoring Service

The installer will install the new Monitoring Service as a separate service instead of overwriting the existing one. If installing over an existing installation, the installer will detect and halt any older services running against the monitoring executable.

In order to prevent older services from restarting on system startup, you should remove or disable them. Otherwise there can be multiple services running against the same installation, which will cause duplicate entries to be written to the monitoring database.

This command line command can be used to delete specified monitoring services:

```
sc delete ExagoMonitoringService.full_version_number
```

```
sc delete ExagoMonitoringService.v2017.2.1.117
```

Back up settings files

Some settings files may be overwritten when updating. Make sure to back up the following files before running the installation:

- <WEBAPP>/Config/Other/dbconfigs.json
- <WEBAPP>/Config/Other/cdataconfig.json
- <WEBAPP>/Config/Other/tagwhitelist.json
- <SCHEDULER>/eWebReportsScheduler.exe.config
- <WEBSERVICE>/appSettings.config

Application settings files for the web application and monitoring service will not be overwritten.

'Name' Property of Filter Objects Deprecated

The Name property of Filter Objects has been deprecated. This property has been obsoleted by the property FilterText. This change may affect any references to Filter Objects made within the .NET API. For more information please see .NET API General Reference – Sorts and Filters.

Configuration File Changes

The following section details the changes made to the configuration XML files.

v2021.2.0

Added to <webreports>:

- <UseExternalTimeZoneConverter>
- <general>
  - <errorondbrowlimitexceeded>

v2021.1.17

Added to <webreports>

- <general>
  - <maxexportfilesizebytes>

v2021.1.16

Added to <webreports>:

- <general>
  - <errorondbrowlimitexceeded?

v2021.1.8

Added to <webreports>:

- <general>
  - <chainedreportmaxcollationexecutions>
  - <sqliteprovider>
  - <sqliteprovidertableschema>
  - <sqliteproviderviewschema>
  - <sqliteproviderfunctionchema>
  - <sqliteproviderprocedurechema>

v2021.1.2

Added to the **Scheduler Service** Configuration file:

- <smtp_timeout>

Deprecated from the **Scheduler Service** Configuration file:

- <default_job_timeout>

v2021.1.0

Added to <**webreports**>:

- <entity>
  - <canreexecuteindb> â€" set to *False* to prevent the Dashboard Designer from executing filters in the data source. See **Interactive Filtering in Database** in Data Objects for more information.
- <storagemgmt>
  - <default_access_flags> â€" the definition of the <default_access_flags> has changed in this version. Review the Storage Management section of the Config File XML Reference article for more information.
- <general>
  - <dashboardautomaticallyrefreshtiles> â€" set to *True* to allow tiles in the Dashboard Designer to refresh for all changes made to them. This also disables the Refresh Reminder feature. See Feature/UI Settings for more information.
  - <showautosum> â€" show (*true*) or hide (*false*) the **AutoSum** Σ icon on the toolbar in the Advanced Report Designer.

Removed from **<webreports>:**

- <etl>
- <general>
  - <etlreportfolder>
  - <showtipsexpressview>
  - <showtutorialexpressview>

Removed from the **Scheduler Service** Configuration file:

- <enable_etl_bulk_insert>
- <etl_bulk_insert_local_path>
- <etl_bulk_insert_database_path>
- <use_etl_bulk_insert_csv_format>
- <etl_bulk_insert_field_terminator>
- <etl_bulk_insert_debug>

v2020.1.14

Added to the **Scheduler Service** Configuration file:

- <smtp_timeout>

Deprecated from the **Scheduler Service** Configuration file:

- <default_job_timeout>

v2020.1.2

Added to the **Scheduler Service** Configuration file:

- <enable_etl_bulk_insert>
- <etl_bulk_insert_local_path>
- <etl_bulk_insert_database_path>
- <use_etl_bulk_insert_csv_format>
- <etl_bulk_insert_field_terminator>
- <etl_bulk_insert_debug>

v2020.1.0

Added to <**webreports**>:

- <general><sqlgenerationlevel></general> â€" Set the level of in-database formula grouping used by Exago. This value should be *1* unless otherwise directed by Exago Support. More information can be found in the Hidden Flags article.
- <storagemgmt> â€" defines a Storage Management configuration
  - <assembly>
  - <class>
  - <table_prefix>
  - <report_list_cache_timeout>
  - <report_xml_cache_timeout>
  - <theme_list_cache_enabled>
  - <report_xml_cache_enabled>
  - <default_inherit_flag>
  - <default_party_type_id>
  - <default_access_flags>
  - <option>
    - <DbType>
    - <DbProvider>
    - <ConnectionString>
    - <ReportListCacheKey>
  - <identity>
    - userId
    - classId
    - companyId
    - ownerId

v2019.2.31

Added to the **Scheduler Service** Configuration file:

- <smtp_timeout>

Deprecated from the **Scheduler Service** Configuration file:

- <default_job_timeout>

v2019.2.18

Added to <**webreports**>:

- <entity>
  - <inherit_category>
  - <inherit_description>

v2019.2.0

A number of Express Reports settings were removed from the Admin Console and now exist only as hidden flags.

v2019.1.0

Added to <**webreports**>

- <**config**> Specify directive information for different config types.
  - <**lifespan**> How long, in seconds, the configuration should be retained.
  - <**type**> Whether the configuration file is dynamic or static.
  - <**parent**> The name of the parent configuration.
- <**etl**> ETL Job object information
  - <**name**>
  - <**id**>
  - <**datasource_id**>
  - <**schema**>
  - <**object_name**>
  - <**report_name**>
  - <**job_id**>
  - <**schedule_info**>
  - <**enabled**>

Added to <**general**>

- <**activeroleid**> Set or view the currently active Role.
- <**showsqlwindow**> If True, users will be able to preview the SQL that will be sent to the database when executing Advacned Reports.
- <**clienttimezonename**> Set the client's default geopolitical time zone.
- <**useexternaltimezoneconverter**>
- <**mintilewidth**> Set the minimum tile width during automatic resizing.
- <**mindesktopwidth**> Set the minimum desktop width during automatic resizing.
- <**reportxmlcacheenabled**>
- <**reportxmlcachetimeout**>
- <**etlreportpath**> Report path where the ETL Reports Folder should be stored, defaulted to standard report path.
- <**etlreportfolder**> Folder within the specified report path where ETL reports will be stored.
- <**licensekey**> Licensing information to enable purchasable features.

Added to <**column_metadata**>

- <**col_dateformat**> Date field formatting (e.g., dd-MM-yyyy). For use with vertical table transformations only.

v2018.2.6

Added to <**general**>

- <**evaluateformulasindatabase**> If True, Exago will convert formula filters to SQL to be evaluated in the database.

Added to <**function**>

- <**filter_return_type**> Required field specifying the return type of a function. Can be set to *String*, *Integer*, *Decimal*, or *Date*.

Moved to <**appsettings.config**>

- <**loadassemblyinexternaldomain**>

Moved to <**webreportsapi.dll**>

- **SQLUtils.dll**

> **NOTE**
>
> SQLUtils.dll has been removed from the base install. To continue using its methods, it must now be referenced from WebReports.Api.SqlUtils

v2018.2.0

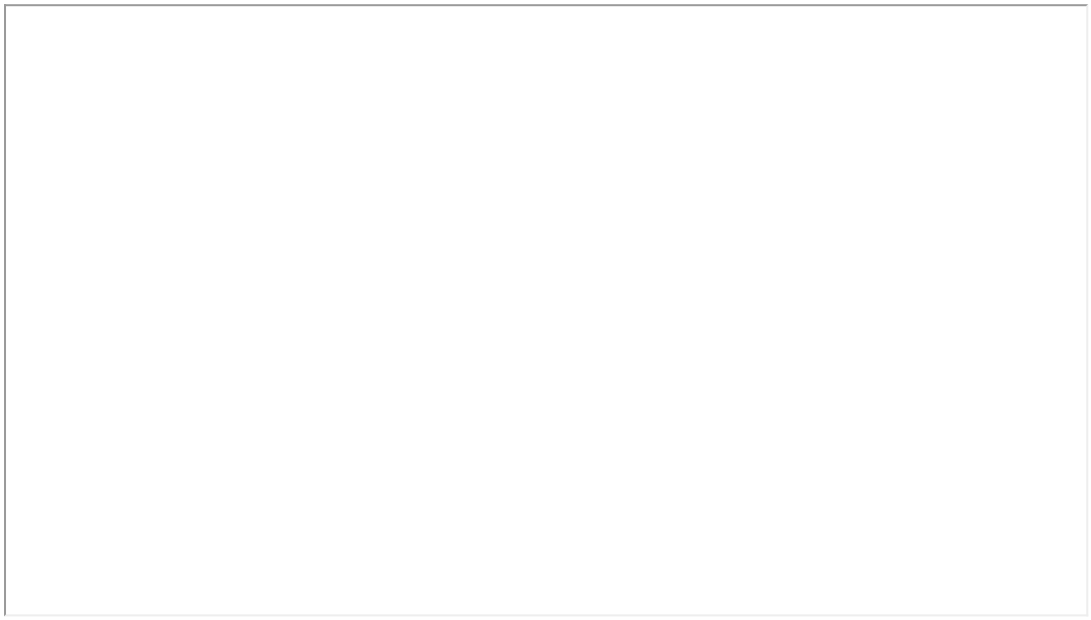
Added to <**general**>

- <**schedulershowreplyto**> Allows a user to specify a â€œReply Toâ€ address when creating Scheduled Reports.
- <**showtipsexpressview**> Enable or Disable â€œTipsâ€ in the ExpressView designer.
- <**showtutorialexpressview**> Enable or Disable the tutorial as a new user enters the ExpressView designer.
- <**reportlistcacheenabled**> If True, Exago will cache the report list returned by Folder Managementâ€™s method GetReportList() to reduce the number of calls being made.
- <**filterdropdownobjecttenancy**>
- <**evaluateformulasindatabase**> (*HIDDEN FLAG*) Currently hard coded to false pending future enhancement.

Configuring IIS for Exago

This guide covers the IIS configuration details necessary in order to install Exago. First-time users are highly encouraged to use this guide during their installation process. Following these steps in order will reduce the amount of troubleshooting necessary to get Exago running.

1. Install Prerequisites
2. Install Exago
3. Set the Application Pool
4. Set Folder Paths and Permissions
5. Check that the Web Site is Running
6. Open the Admin Console

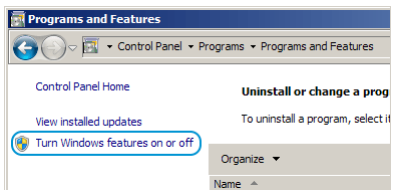


1. Install Prerequisites

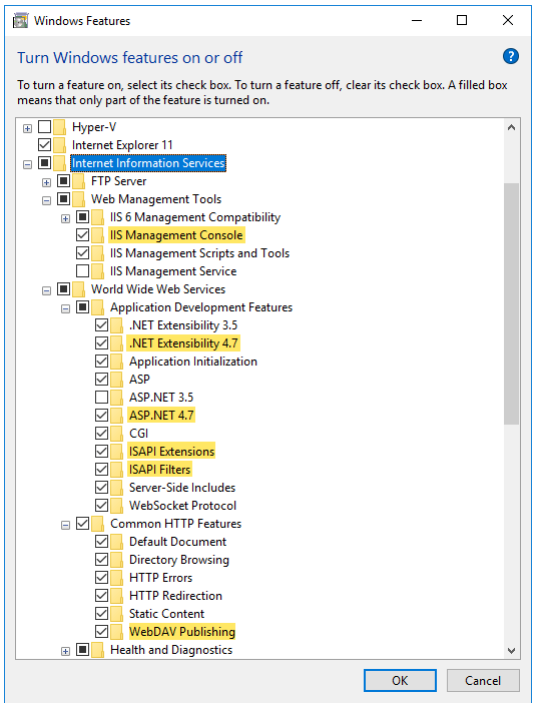
Exago requires the following WindowsÂ Features to be installed:

- Internet Information Services
  - Web Management Tools
    - IIS Management Console
  - World Wide Web Services
    - Application Development Features
      - .NET Extensibility
      - ASP.NET
      - ISAPI Extensions
      - ISAPI Filters
    - Common HTTP Features
      - *Optional:* WebDAV Publishing

Before installing Exago, please ensure that these features are present on your system. To verify, access the Windows Features panel via **Control Panel** > **Programs** > **Programs and Features**, and clickÂ **Turn Windows features on or off**.



In the Windows Features dialog, expand **Internet Information Services**Â and ensure that the prerequisite features are selected. If any are not installed, check the relevant boxes, press OK, then restart the server.



These features are necessaryÂ in order for the proper Application Pools to be available.

2. Install Exago

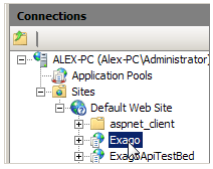
At this point, Exago can be installed. For a detailed walkthrough, see **Installing Exago on Windows**.

3. Set the Application Pool

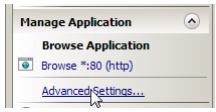
Next, verify that Exago is using a valid application pool.

Open IIS viaÂ **Control PanelÂ** >Â **Administrative Tools** >Â **Internet Information Services (IIS) Manager**.

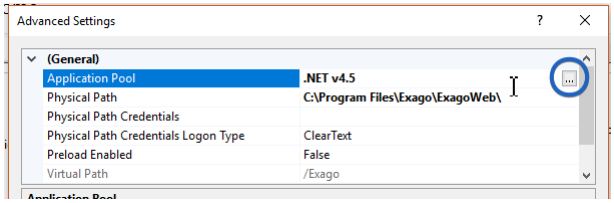
In the left-most Connections pane, select the Exago application.



In the right-most Actions pane, click onÂ **Advanced Settings…**

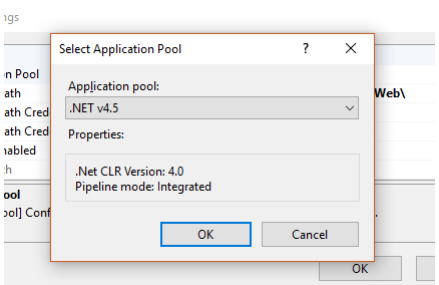


In thisÂ menu, click on the **[…]** button to the right of theÂ **Application Pool**.



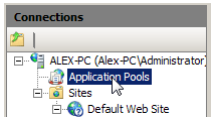
In the **Select Application Pool** menu,Â determine which of the app poolsÂ has the properties:

- .Net Framework Version: 4.5
- Pipeline mode: Integrated

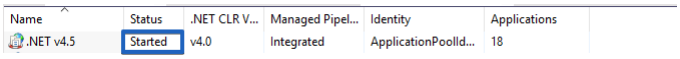


Select this app pool, then press **OK** to close the App Pool menu, and then **OK** to close the Settings menu.

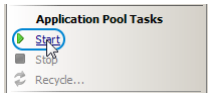
Next,Â ensure thatÂ the app pool is running. In the left-most Connections pane, select **Application Pools**.



Check the **Status** column for your selected app pool.



If it does not say ***Started***, select the app pool and in the right-most Actions pane, press **Start**.

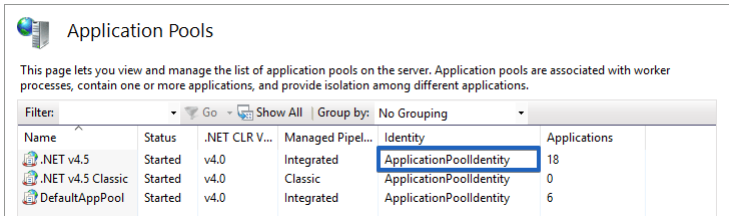


4. Set Folder Paths and Permissions

Exago uses a Temp folder to store working data. Create folders in locations suitable for your environment.

The **Temp** folder may contain sensitive report and database information. It should be in a secure location, inaccessible to web users.

Several Exago folders require you to set additional permissions for the application pool user. First, determine the user:Â Open IIS to theÂ Application Pools pane, and look at the app pool which is running Exago. The Identity property indicates the application user:



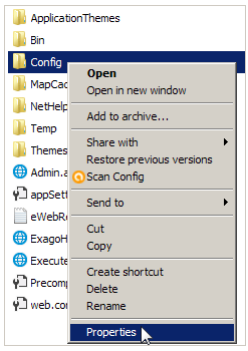
By default, this is set to *ApplicationPoolIdentity*. This corresponds with the built-in user *IIS_IUSRS*. If this is a different user, then set permissions for that user account instead.

The following folders require the app pool user to have read/write permissions:

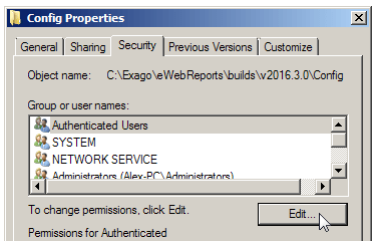
- Config
- Temp
- MapCache
- ApplicationThemes (*v2020.1.3+*)

> **NOTE**
>
> This process may differ slightly depending on the version of Windows.

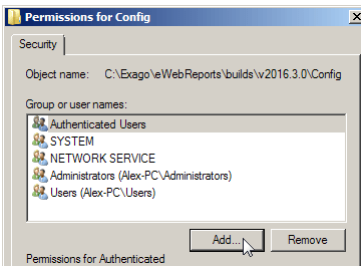
First, right-click on the folder and selectÂ **Properties**.



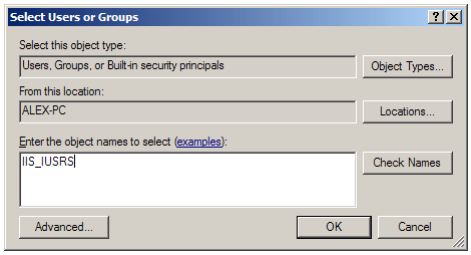
OpenÂ the **Security** tab. Then click the **Edit…** button.



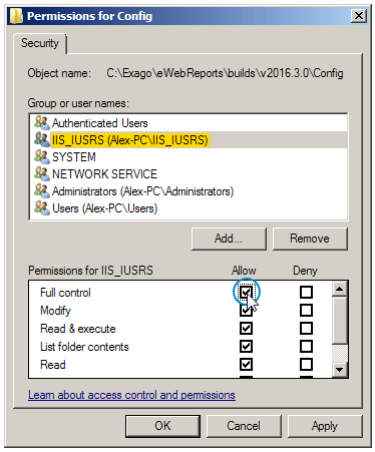
If the application pool user is not available to select, you need to add it. Press the **Add…** button.



Then enter the username in the dialog box, and press OK (the default app pool user isÂ *IIS_IUSRS*).



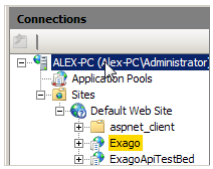
With the user selected, check the box for Allow **Full control**. Then press OK.



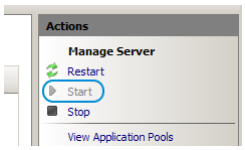
Repeat this process for every folder listed above.

5. Check that the Web Site is Running

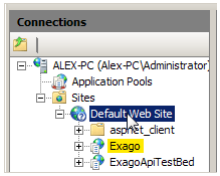
Before startingÂ Exago, ensure that the Web Site is running. OpenÂ IIS, and in the left-most Connections pane,Â locate and select the web serverÂ under which Exago was installed



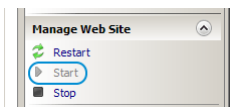
VerifyÂ in the right-mostÂ Actions pane that the **Start** button is greyed out, and the **Restart** and **Stop** buttons are available. If the **Start** button is not greyed out, press it to start the web server.



Next, in the left-most Connections pane,Â locate and select the web site under which Exago was installed.



Verify in the right-mostÂ Actions pane that the **Start** button is greyed out, and the **Restart** and **Stop** buttons are available. If the **Start** button is not greyed out, press it to start the web site.



6. Open the Admin Console

You're almost done! To verify that Exago was installed correctly, open a web browser and navigate to http://[YourServerAddress]/[YourExagoVirtualPath]/Admin.aspx. If you see the **Exago Administration Console**, then your installation wasÂ successful.

In the **Admin Console** > **General Settings**, set your Temp to the folder you previously created. Then click **Okay**.

Setup the **Storage Management** for storing reports, templates and themes.

- Storage Management: Introduction
- Storage Management: Getting Started
- Storage Management: Admin Console

Finally, navigate to http://[YourServerAddress]/[YourExagoVirtualPath]/ExagoHome.aspx. If you can see the Exago Web Application user interface, depending on version the Report Tree should be:

- empty except for a **My Reports** and **Public** folder for versions *v2020.1+*

Congratulations! You've completedÂ your first Exago installation.

Additional Notes

We highly recommend Setting up a State Server to handle Exago sessions.

If you are experiencing problems that aren't detailed in this guide, please file a support ticket.

Installing Exago on Windows

Exago runs on the web server application Microsoft Internet Information Services (IIS). The following sections walk through the installation process forÂ Windows based systems.

This accompanying video is from the Technical Training Series. To see the other videos in this series, start with the Introduction to the Technical Training Series video, or visit the Admin Video Training Series category page.

Prerequisites

- See Considerations When Sizing an Exago System to ensure that you have sufficient hardware to run Exago.
- See Technical Specifications to ensure you have the minimum software requirements to run Exago.
- See Configuring IIS for Exago to ensure that IIS is installed and configured correctly.

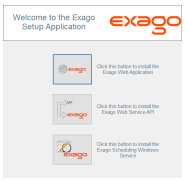
Installation

> **IMPORTANT**
>
> If you are upgrading an existing Exago installation, please ensure that the file *eWebReportsManifest.txt* is present in the install directory. Otherwise, the installer will overwrite any custom config or styling you've applied.

Download the Exago installer from our Downloads page.

Run the installer as an administrator. The installation menu will appear with three downloadable applications. Click the top icon to install the **Exago Web Application**, as pictured below.

> **NOTE**
>
> If Windows Defender prevents this installation, it may be bypassed by clicking "More Info" and then selecting "Run Anyway."



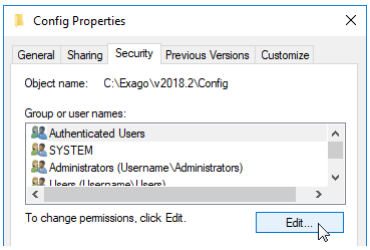
Follow the steps in the wizard to install Exago. You may optionally choose to install the Scheduler and the Web Service from this wizard.

Even though the installer has finished, Exago will most likely not function at this point. You must continue with some additional configuration.

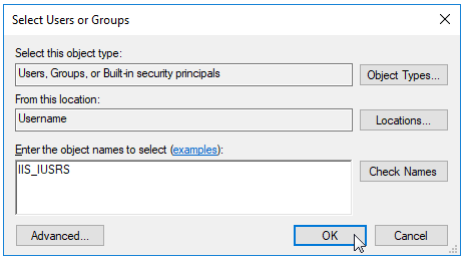
Create the Directory Structure

After the installation is complete, configure Exago using the following steps. (See Configuring IIS for Exago for a more detailed walkthrough.)

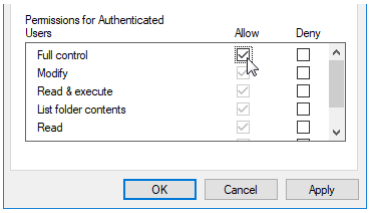
1. Set permissions for the **Config** folder:
    1. Right click on the folder named “Config” and click **Properties**.



    2. In the Security tab click “Edit” then “Add.” Enter the IIS application pool user (default **IIS_IUSRS**).



    3. In the “Permissions for Config” window select the user that was just created and select “Full Control” permissions.



2. Repeat this process for the **ApplicationThemes**, **MapCache** and **Drivers** folders (the latter is only required if a CData Driver is being utilized.)
3. Create a folder for storing temporary data. By default this is a sub-folder of Exago called ‘**Temp**‘. However it is recommended to not use the install path’s temp folder in production environments. Give the Temp folder full control privileges for the IIS application pool user.
4. Point your browser to the **Administration Console**. By default this is **http://<YourServer>/Exago/Admin.aspx**.
    - For *v2020.1+*, setup the **Storage Management** system for storing reports, templates and themes.
        - Upgrade Install â€” If upgrading from a previous version of Exago, review these articles:
            - Storage Management: Introduction
            - Storage Management: Getting Started
            - Storage Management: Transitioning from Legacy Storage Methods
        - New Install â€” If this is a new installation, review these articles:
            - Storage Management: Introduction
            - Storage Management: Getting Started
            - Storage Management: Admin Console
    - For *pre-v2020.1*, in the **General** > **Main Settings** section, specify the location of the **Report** Folder in the ‘**Report Path**‘ setting. Verify the connection is successful by clicking the ⚡ icon.
    - For all versions, in the **General** > **Main Settings** section, specify the location of the **Temp** Folder in the ‘**Temp Path**‘ setting.

What’s Next

Point your browser to the **Admin Console** to verify that your installation was successful. By default this is **http://<YourServer>/Exago/admin.aspx**

If you encounter problems at any point, please see Installation Troubleshooting for some potential solutions. If you cannot resolve your problem, please file a Support Ticket.

At this point you will need to set up your data sources in order to use Exago. See Administration Console Setup to get started.

If you would like to set up Google Maps, GeoCharts, and/or any downloadable Application Themes, please see Installing Optional Features for more information.

Resources

- System Requirements â€" Baseline hardware requirements.
- Configuring IIS for Exago â€" Necessary configuration details for IIS.
- Installing the Scheduler Service â€" Scheduler config info.
- Install and Configure the Web Service â€" Web service config info.
- Installation Troubleshooting â€" Common install problems & their solutions.
- Administration Console Setup â€" Initial data sources setup guide.
- Installing Optional Features â€" How to set up Google Maps, GeoCharts, and Application Themes.

Installing Exago on Linux

The following article walks through the installation process forÂ Linux systems. Before beginning installation, refer to the Technical Specifications article for a list of supported distributions and a list of minimum requirements.

> **IMPORTANT**
>
> The Linux Installer was updated in *v2019.2.1* of the application. This document covers several versions of the installer. Use the **Viewing content for dropdown** to see only the content relevant to a specific application version. Any previous headless install scripts will start prompting for missing formation and will require editing to account for the new parameters.

This article is divided into three sections: Apache, NGINX and Common Install Information. Only follow the instructions in the section that corresponds with the web server in your environment, and review Common Install Information for all installations.

Apache

NGINX

Installation with Apache

Apache must be installed prior to installing Exago.

The Exago Linux Installer can be used to install the Exago Web Application, Web Service API, and Scheduler Service. It can also install mono and mod-mono. Use the following steps to install Exago on Linux with Apache.

> **NOTE**
>
> Apache or NGINX must be installed prior to installing Exago. If changing from one or the other, verify the dependencies (mod-mono or mono-fastcgi-server4) are installed.

1. Navigate to the **Downloads** page, select a build, and then use the Linux Download option.
2. Decompress the download:

```
tar zxvf ExagoInstaller_vX.X.X.tgz
```

3. Change to the newly created Installer directory:

```
cd Installer
```

4. Run installExago.sh as root:

```
sudo ./installExago.sh
```

5. The installer can be run in guided or silent mode. The Linux distribution and Apache version will be detected automatically.

Silent Installation with Parameters

Usage:

```
[-d <install path>] [-m <TRUE|FALSE>] [-i <WEBAPP|WEBAPI|SCHEDULER>] [-y] [-h]
```

Parameter	Description
-d <Install Path>	Directory to install Exago in Default is /opt/Exago
-m <TRUE|FALSE>	Whether or not to install Mono framework with Exago
-i <WEBAPP|WEBAPI|SCHEDULER>	Which component(s) to install • *WEBAPP* â€" Web Application • *WEBAPI* â€" REST Web Service API • *SCHEDULER* â€" Scheduler Service
-a <Web App URL Alias>	Default is /Exago
-s <Web Service URL Alias>	Default is /ExagoWebApi
-r <Scheduler Service Name>	Name to use for the Scheduler Service's systemd service. Default is *exago-scheduler*.
-R	Configure systemd services As of *v2019.2.1*, Exago will automatically install systemd services unless the -N flag is passed to inhibit their installation. This option is not available in Exago versions *pre-v2019.2.1*.
-N	As of *v2019.2.1*, do NOT configure systemd services This option is not available in Exago versions *pre-v2019.2.1*.

-u <userId>	As of *v2019.2.1*, user ID to create in the operating system that the Exago systemd services and FastCGI listeners will run under. A group and home directory with the same name will be created. This option is not available in Exago versions *pre-v2019.2.1*.
-w <APACHE|NGINX>	Available in *v2019.1.15+* Select the web server to configure, if both are available.
-D	As of *v2019.2.1*, select all default values for installation options. Individual options can be overridden by including the parameter on the command line. This option is not available in Exago versions *pre-v2019.2.1*. See Default Values below for further details
-y	Do not prompt for final verification before installing
-h	Show this help screen

Examples:

1. Install the Exago Web Application and Web Service API into /opt/Exago

```
./installExago.sh -d /opt/Exago -m TRUE -i WEBAPP -i WEBAPI
```

2. Full installation using all default values without prompting before starting the install

```
./installExago.sh -D -y
```

3. Install the Scheduler Service only into /opt/ExagoRemote using defaults for everything else

```
./installExago.sh -d /opt/ExagoRemote -i SCHEDULER -D
```

Guided Installation v2019.2.1+

```
##############################################
 ___ ___ ___  ___   ___ _   ___
 / _ \ / / _ \| _ \ / _ \ | / _ \
| (_) > < | (_) |   / | (_) | | (_) |
 \___/_/\_\ \___/|_|_\ \___/|_|\___/
                 \__/|
    Linux Installer |___/
##############################################
```

The Linux Installer will ask a series of questions to setup the environment. Suggested default values for each question are enclosed in [brackets]. To choose the default value, simply press the Enter key on the keyboard. Default values are explained in their own section below.

1. Select base install path. Default value is /opt/Exago.
2. Would you like to install the Mono runtime if not already installed? Default is *yes*.
3. Would you like to install the Exago Web App? Default is *yes*.
4. Would you like to install the Exago Web Service API? Default is *no*.
5. Would you like to install the Exago Scheduling and Remote Execution Service? Default is *no*.
6. Multiple web servers appear to be present. Please choose which web server you would like to use. This question will only appear if both Apache and NGINX are installed on the web server. You must choose 1 for Apache or 2 for NGINX.
7. Enter a name for the Exago Scheduling Service. Default is *exago-scheduler*.
8. Select WebServer Alias for Exago Web App. Default is /Exago
9. Select WebServer Alias for Exago WebService API. Default is /ExagoWebApi
10. Should systemd services be configured? Default is *yes*.
11. Please enter a userId to configure for use with Exago. This user account is used to run the Exago services under (Scheduler Service and Monitoring Service). Default is *exago*.
12. A summary of installation parameters will appear and you will be asked if ready to install.

Continue to Start and Test Install below.

Start and Test Install

Restart Apache.

Point your browser to the **Admin Console** to verify that your installation was successful and to create a default configuration file. By default this is http://<YourServer>/Exago/Admin.aspx

If you encounter problems at any point, please file a Support Ticket.

Continue to Folder Configuration below to continue with the installation.

Folder Configuration

Configure storage locations for the **configuration**, **report storage** (either *Storage Management* for v2020.1.0+ or a Report Path *pre-v2020.1.0*) and **temporary file storage.**

Config

The **Config** sub-folder of the Exago installation has read and write permissions set by default.

Report Storage

For versions *v2020.1+*, setup the **Storage Management** system for storing reports, templates and themes.

Upgrade Install

If upgrading from a previous version of Exago, review these articles:

- Storage Management: Introduction
- Storage Management: Getting Started
- Storage Management: Transitioning from Legacy Storage Methods

New Install

If this is a new installation, review these articles:

- Storage Management: Introduction
- Storage Management: Getting Started
- Storage Management: Admin Console

Setting up the **Storage Management** system in general is as follows:

1. Create and setup the Storage Management database. This can be accomplished using either one of the following methods:
   - *Recommended*: use the **Admin Console** to initialize the database
   - Manually creating database and the tables based on the Storage Management: Database Schema
2. Configure Exago to connect to the Storage Management database

Temporary File Storage

The recommended path for the **Temp** folder is /opt/Exago/Temp.

1. In the **Admin Console**, specify the location of the temp Folder in **General** > **Main Settings** > **Temp Path**
2. Set the **MonitoringService** folder's read and write permissions for the Apache user, (of if not using the default Apache user the user ID specified with -u) to **775**, and set the default ownership to **<apache user>**:**root**.

> **NOTE**
>
> If using a custom Application Theme, the **ApplicationThemes** folder will also need the same permissions.

Installation is now completed. If the Scheduler Service or Monitoring Service is required, continue to the Scheduler and Monitoring Services section at the end of this article.

If all components are installed, the Installer directory may be deleted.

Installation with NGINX

NGINX must be installed prior to installing Exago. NGINX proxies incoming and outgoing requests to a running instance of Exago using a fastcgi module that is installed during the installation process. **mono-fastcgi-server4** is a prerequisite for Exago to run on NGINX.

The Exago Linux Installer can be used to install the Exago Web Application, Web Service API, and Scheduler Service. It can also install mono and mono-fastcgi-server4. Use the following steps to install Exago on Linux with NGINX.

> **NOTE**
>
> Apache or NGINX must be installed prior to installing Exago. If changing from one or the other, verify the dependencies (mod-mono or mono-fastcgi-server4) are installed.

1. Navigate to the **Downloads** page, select a build, and then use the Linux Download option.
2. Decompress the download:

   ```
   tar zxvf ExagoInstaller_vX.X.X.X.tgz
   ```

3. Change to the newly created Installer directory:

   ```
   cd Installer
   ```

4. Run installExago.sh as root:

   ```
   sudo ./installExago.sh
   ```

5. The installer can be run in guided or silent mode. The Linux distribution will be detected automatically.

Silent Installation with Parameters

Usage:

```
[-d <install path>] [-m <TRUE|FALSE>] [-i <WEBAPP|WEBAPI|SCHEDULER>] [-y] [-h]
```

Parameter	Description
-d <Install Path>	Directory to install Exago in Default is /opt/Exago
-m <TRUE|FALSE>	Whether or not to install Mono framework in Exago
	Which component(s) to install

-i <WEBAPP|WEBAPI|SCHEDULER>	• *WEBAPP* â€" Web Application • *WEBAPI* â€" REST Web Service API • *SCHEDULER* â€" Scheduler Service
-a <Web App URL Alias>	Default is /Exago
-s <Web Service URL Alias>	Default is /ExagoWebApi
-r <Scheduler Service Name>	Name to use for the Scheduler Service's systemd service. Default is *exago-scheduler*.
-f <Fastcgi Service Name>	As of *v2019.2.1*, name to use for the FastCGI services used by NGINX. This option is not available in Exago versions *pre-v2019.2.1*.
-p WEBAPP:<port>|WEBAPI:<port>	As of *v2019.2.1*, port numbers to use for the FastCGI listeners for the Web Application or Web Service API, respectively. This option is not available in Exago versions *pre-v2019.2.1*.
-R	Configure systemd services As of *v2019.2.1*, Exago will automatically install systemd services unless the -N flag is passed to inhibit their installation. This option is not available in Exago versions *pre-v2019.2.1*.
-N	As of *v2019.2.1*, do NOT configure systemd services This option is not available in Exago versions *pre-v2019.2.1*.
-u <UserID>	As of *v2019.2.21*, User ID to create in the operating system that the Exago systemd services and FastCGI listeners will run under. A group and home directory with the same name will be created. This option is not available in Exago versions *pre-v2019.2.1*.
-w <APACHE|NGINX>	Available in *v2019.1.15+* Select the web server to configure, if both are available.
-D	As of *v2019.2.1*, select all default values for installation options. Individual options can be overridden by including the parameter on the command line. See Default Values below for further details This option is not available in Exago versions *pre-v2019.2.1*.
-y	Do not prompt for final verification before installing
-h	Show this help screen

Examples:

1. Install the Exago Web Application and Web Service API into /opt/Exago

```
./installExago.sh -d /opt/Exago -m TRUE -i WEBAPP -i WEBAPI
```

2. Full installation using all default values without prompting before starting the install

```
./installExago.sh -D -y
```

3. Install the Scheduler Service only into /opt/ExagoRemote using defaults for everything else

```
./installExago.sh -d /opt/ExagoRemote -i SCHEDULER -D
```

Guided Installation v2019.2.1+

```
##############################################
 ___ ___ ___ ___  ___  ___
/ _ \ / / _ \ | / _ \ / _ \
| __/ > < | (_) | | (_) | | (_) |
 \___|/_/\_\___/|_|\___/ \___/
                 _/ |
    Linux Installer |__/
##############################################
```

The Linux Installer will ask a series of questions to setup your environment. Suggested default values for each question are enclosed in [brackets]. To choose the default value, simply press the Enter key on the keyboard. Default values are explained in their own section below.

1. Select base install path. Default value is /opt/Exago.
2. Would you like to install the Mono runtime if not already installed? Default is *yes*.
3. Would you like to install the Exago Web App? Default is *yes*.
4. Would you like to install the Exago Web Service API? Default is *no*.
5. Would you like to install the Exago Scheduling and Remote Execution Service? Default is *no*.
6. Multiple web servers appear to be present. Please choose which web server you would like to use. This question will only appear if both Apache and NGINX are installed on the web server. You must choose 1 for Apache or 2 for NGINX.
7. Enter a name for the Exago Scheduling Service. Default is *exago-scheduler*.
8. Select WebServer Alias for Exago Web App. Default is /Exago
9. Select WebServer Alias for Exago WebService API. Default is /ExagoWebApi
10. Enter the base fastcgi service name. Default is *exago-fcgi*
11. Enter the port for the web app fastcgi listner. Default is *9000*
12. Enter the port for the web service fastcgi listener. Default is *9001*.
13. Should systemd services be configured? Default is *yes*.
14. Please enter a userId to configure for use with Exago. This user account is used to run the Exago services under (Scheduler Service and Monitoring Service). Default is *exago*.
15. A summary of installation parameters will appear and you will be asked if ready to install.

Continue to Start and Test Install below.

Start and Test Install

Configure NGINX

The required configuration is created in a separate site file located in the NGINX configuration directory.

For Ubuntu systems this is typically /etc/nginx/sites-available/exago. For other systems this may be /etc/nginx/conf.d/exago. The exact location of the file will depend on your system and NGINX installation. Consult with the operating system and NGINX official product documentation for more information.

The site file is not enabled by default.

Example

```
#Exago conf file for fastcgi-mono-server4
server {
 listen 80;
 listen [::]:80;
 server_name _;
 root /var/www;

 location /Exago/ {
  include /etc/nginx/fastcgi_params;
  root /opt/Exago;
  access_log /var/log/nginx/exago.log;
  fastcgi_param SERVER_NAME $host;
  fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
  fastcgi_param PATH_INFO "";
  fastcgi_pass 127.0.0.1:9000;
 }

 location /ExagoWebApi/ {
        include /etc/nginx/fastcgi_params;
        root /opt/Exago/WebServiceApi;
        access_log /var/log/nginx/exago_ws.log;
        fastcgi_param SERVER_NAME $host;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO "";
        fastcgi_pass 127.0.0.1:9001;
 }
}
```

For example on Ubuntu, to enable the site file, link it to /etc/nginx/sites-enabled:

```
sudo ln /etc/nginx/sites-available/exago /etc/nginx/sites-enabled/exago
```

Or include the configuration in another running site configuration file. This step will depend on your specific environment. Consult your operating system and NGINX official documentation.

> **CAUTION**
>
> Make sure that the default port does not conflict with another running site. If it does, you will see a warning when reloading Nginx: nginx: [warn] conflicting server name "Exago" on 0.0.0.0:80

Then reload NGINX to refresh the configuration:

```
sudo nginx -s reload
```

> **NOTE**
>
> This article references <WebApp>/, <WebSvc>/ as a placeholder for the application's install location and the Web Service API's install location, respectively. The default install locations are /opt/Exago/ and /opt/Exago/WebServiceApi; however, these directories may be changed during installation.

Start Listeners

The installer will create two new scripts which can be used either standalone or along with a service manager such as **systemd**.

> **NOTE**
>
> The **exago-fcgi** portion of each service name can be configured from the defaults below with the -f parameter in silent installation or by providing an answer to the **Enter the base fastcgi service name** question during guided installation.

- <WebApp>/exago-fcgi-webapp.sh
- <WebSvc>/exago-fcgi-webservice.sh

When not using a service manager, each of these scripts can be run with either the **start**, **stop** and **restart** directives. For example,

<WebApp>/bin/exago-fcgi-webapp.sh start will start the Web Application's FastCGI listener. <WebSvc>/bin/exago-fcgi-webservice.sh restart will restart the Web Service API's FastCGI listener.

When using systemd, the commands take the following form:

- service exago-fcgi-webapp start
- service exago-fcgi-webservice stop

Content Storage Configuration

The **Config** sub-folder of the Exago installation has read and write permissions set by default.

For versions *v2020.1+*, setup the **Storage Management** system for storing reports, templates and themes.

Upgrade Install

If upgrading from a previous version of Exago, review these articles:

- Storage Management: Introduction
- Storage Management: Transitioning from Legacy Storage Methods
- Storage Management: Getting Started

New Install

If this is a new installation, review these articles:

- Storage Management: Introduction
- Storage Management: Getting Started
- Storage Management: Admin Console

Default UMASK for files written by Exago is 027.

The recommended path for the **Temp** folder is /opt/Exago/Temp.

1. In the **Admin Console**, specify the location of the temp Folder in **General** > **Main Settings** > **Temp Path**

> **NOTE**
>
> If using a custom Application Theme, the **ApplicationThemes** folder will also need the same permissions.

Installation of Exago is now completed. If the Scheduler Service or Monitoring Service is required, continue to the Scheduler and Monitoring Services section at the end of this article.

If all components are installed, the Installer directory may be deleted.

Common Install Information

Scheduler and Monitoring Services

Scheduler Service

A script named exago-scheduler.sh by default is used to manage the Scheduler Service. The name is customizable with the installer's -r parameter.

Installed as a systemd Service

When utilizing a systemd service, the Scheduler Service should start automatically when the system boots, and can be managed with the service command in the following ways:

- service exago-scheduler start
- service exago-scheduler stop
- service exago-scheduler status

When a second scheduler is installed on the same server, the default service name will increment: exago-scheduler2, exago-scheduler3, and so on.

Not Installed as a systemd Service

When the Scheduler Service is not installed as a systemd service (without the -R parameter), it is not possible to know if another service with the same name exists on the same server. When a service with the same name is installed, only one can be used at a time. This is due to the fact that lock files used by the service are based on the service's name, and will prevent the second process from starting. In this scenario, the -r flag should be used and additional schedulers be given different names.

To start the Scheduler Service, execute the exago-scheduler.sh script as the owner of the installation directory (either **root** *pre-v2019.2.21* or the user name specified with the -u parameter, **exago** by default in *v2019.2.21+*). For example:

In *v2019.2.21+*, assuming the Scheduler Service is installed in /opt/Exago/Scheduler/:

```
su exago
/opt/Exago/Scheduler/exago-scheduler.sh start
exit
```

See **Scheduler Configuration** and **Setting Up Monitoring** to configure the respective services correctly. If necessary, configure the Scheduler and Monitoring services to run at startup.

Start Monitoring Service

> **NOTE**
>
> This article references <WebApp>/ as a placeholder for the application's install location. The default install location is /opt/Exago/; however, this directory may be changed during installation.

If the Web Application was installed, the Monitoring Service is also installed, but is disabled by default. This script created during installation needs to be started manually or configured to run at startup:

```
<WebApp>/MonitoringService/startService.sh
```

Default Values

The -D parameter was added to the Linux installer in *v2019.2.1*. This parameter will select the default values for each parameter unless already specified on the command line. These values are:

- defined in the **exagoinstall.properties** file in the Exago installation directory if Exago has already been installed, or
- if Exago has never been installed or is being installed in a new directory for the first time, defined below:
  - **Exago Installation Path**: /opt/Exago
  - **Components to install**: *Web Application*, *Monitoring Service*, *Web Service API*, *Scheduler Service*
  - **Install Mono runtime**: *Yes*
  - **User ID:** *exago*
  - **Scheduler information**
    - **Scheduler install path**: /opt/Exago/Scheduler
    - **Scheduler service name**: *exago-scheduler*
    - **Scheduler user ID**: *exago*
    - **Configure systemd service**: *Yes*
  - **Web Server information**
    - **Selected Web Server**: *Apache* (will use Apache if both NGINX and Apache are available; will use NGINX if only NGINX is available)
    - **Web Application install path**: /opt/Exago
    - **Web App alias**: /Exago
    - **Web App user ID**: *exago* for NGINX. For Apache, the default is system dependent and will be read from either apache.conf or httpd.conf depending on which one is in use.
    - **(NGINX only) NGINX conf file**: /etc/nginx/conf.d/exago.conf
    - **(Apache only) Apache conf file**: /etc/httpd/conf.d/exago.conf
    - **Web Service install path**: /opt/Exago/WebServiceApi
    - **Web Service alias**: /ExagoWebApi
    - **Web Service user ID**: *apache* for Apache, *exago* for NGINX
    - **(NGINX only) Use system for service(s)**: *Yes*
    - **(NGINX only) Web App Service Name/Port**: *exago-fcgi-webapp / 9000*
    - **(NGINX only) Web API Service Name/Port**: *exago-fcgi-webservice / 9001*

> **TIP**
>
> For a completely headless installation with all of the default values, pass the -D and -y parameters. -D will not start the installation on it's own, only select the default values for each parameter.

Once Exago is installed, the settings chosen during installation become the defaults and are written in to the aforementioned **exagoinstall.properties** file in the Exago installation directory.

Upgrading Mono v2019.2.29+ v2020.1.13+ v2021.1.1

The **upgradeMono** script included with the installer can be used to upgrade from one supported version of mono to another. The script will remove the installed version and then install the new version in its place, along with the apache mod_mono module and fastcgi module as applicable.

If the system does not meet the minimum requirements for any of the components, it will upgrade only the installed components but will not install anything additional.

1. Download, uncompress the Linux installer and change to the Installer directory as described in steps 1â€“3 of Installation with Apache or Installation with NGINX above.
2. Call the script as root. The command line parameters described below are optional. If no parameters are provided, the script will prompt for answers to certain questions.

Usage:

```
sudo ./upgradeMono.sh [-f] [-s] [-y] [-a] [-h]
```

Parameter	Description
-f	Force a reinstall, even if the version currently installed is at the recommended version
-s	Skip the reinstall if the version currently installed is already the recommended version
-y	Don't prompt before starting the upgrade, silently install
-a	Upgrade everything despite what is currently installed
-h	Show command line parameter help screen

```
**************************************
Upgrade summary:
Upgrade Mono:        TRUE
Upgrade Mod Mono:    TRUE
Upgrade Mono FastCGI: FALSE
**************************************
Would you like to continue with this upgrade? (Y or N) [Y]
```

An example summary for a system with Apache. Pressing Y or Enter on the keyboard will start the upgrade process

Dependencies

The Exago installer handles the installation and configuration of all of its own dependencies. Installing and configuring dependencies ahead of time is not required, and in some cases could lead to unexpected issues with the Exago deployment. This list is provided as a reference only. In most cases, Exago utilizes the most up to date versions available through the operating system's package manager (i.e. apt, yum, zypper).

The Exago installer will display error messages if certain dependencies are not, cannot be or are not correctly installed.

> **NOTE**
>
> Not all of the packages listed below are installed for every operating system and situation.

- **Installer requirements** â€" for downloading files, adding repositories and installing GPG keys
  - curl
  - unzip
  - wget
  - yum-utils â€" only required on distributions that utilize the yum package manager
- **Base .NET runtime** â€" requirements for all Exago components
  - mono-4.0-service
  - mono-complete
- **Datetime function support**
  - mono-vbnc â€" used on Ubuntu and Debian only
  - mono-basic â€" used on Red Hat Enterprise Linux, CentOS, Fedora and SUSE Linux Enterprise Server
- **Export support**
  - Any font package such as liberation fonts or croscore fonts (chromium installed fonts will also work)
- **Apache web server support**
  - apache2-mod_mono (libapache2-mod-mono)
  - mono-apache-server
- **Puppeteer** â€" support for headless chart rendering in exported files
  - alsa-lib
  - atk
  - chromium
  - cups-libs
  - gconf2
  - gtk3
  - libasound2
  - libatk-bridge2.0-0
  - libgtk-3-0
  - libnss3
  - libx11-xcb1
  - libxcomposite
  - libxcursor
  - libxdamage
  - libxi
  - libxscrnsaver
  - libxrandr
  - libxtst6
  - libxss1
  - nss
  - pango
  - xorg-x11-fonts-75dpi
  - xorg-x11-fonts-100dpi
  - xorg-x11-fonts-cyrillic
  - xorg-x11-fonts-Type1
  - xorg-x11-fonts-misc
  - xorg-x11-utils
- **NGINX web server support**
  - mono-xsp4
  - xsp
  - mono-fastcgi-server4

Installing Exago on Azure

MicrosoftÂ AzureÂ is a cloud infrastructure for hosting files, databases, virtual machines, and web applications. Exago supports various forms of integration with Azure.

- **Virtual Machine** â€" Exago can be installed on a Windows virtual machine on Azure.
- **File Storage** â€" Exago data can be stored and accessed from Azure storage containers.

- **App Service** â€" Although not recommended for production environments, the Exago Web Application, REST Web Service API and .NET API host apps can be installed as Azure App Services. An Azure Virtual Machine is also required for remote report execution when utilizing an Azure App Service.

These methods can be implemented independent of each other. However, Web App integration and VM integration are usually redundant with each other, and most Web App solutions should also implement Azure file storage. This guide will walk through how to set up each of these solutions.

Which solution should I use?

Azure virtual machines are the preferred implementation method for production environments.

> **NOTE**
>
> App Services should only be used for testing and demonstrationâ€"not in production.

File Storage

Exago can be integrated with Azure cloud storage for storage and live access to reports, templates, config, and other data files.

The following Azure resources are:

- Required
  - Storage account
  - Blob storage
- Optional
  - Files storage

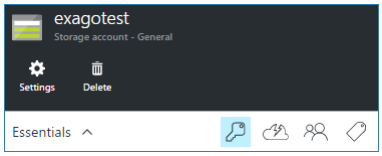
> **IMPORTANT**
>
> The **Web Application**, **Scheduler Service**, **Web Service API** and any cloud storage mechanisms must share at least one security protocol in common otherwise they will not be able to communicate with each other. For more information, refer to the Application Settings article.

Blob storage is a "flat" file system, which stores every file at the root level. To make use of this system, Exago emulates a directory structure using file names. Standard and premium blockblobstorage accounts are supported with Exago.

File storage is a directory-based system. Files are placed into directories, which can have sub-directories.

For *pre-v2020.1*, reports can be stored in File storage or Blob storage. Config files, templates, themes, and temp files must be stored in blob storage. For *v2020.1+*, only config files and temp files are stored locally. Reports, templates and themes are stored in a Storage Management database.

In your Azure Dashboard, begin by creating a new Storage account or navigating to an existing one.



This section is divided into three parts: **Config file** storage, **Reports** storage, and **Temporary files** storage (which includes themes and templates). If you are implementing Exago as a scalable app, you must set all three to static locations.

Config File

An Exago installation contains a configuration file, usually called **WebReports.xml**, which tells the application where to store Reports and Temp files.

First, in the Storage account, navigate to **Access Keys**. Record the two connection strings.

Azure Connection String

An **Azure Connection String** is a formatted string which contains your Azure account name and a unique alphanumeric key. It is used to give applications access to your storage account. The string uses the following format:

```
DefaultEndpointsProtocol=https;AccountName=acctName;AccountKey=encryptedalphanumerickey;
```

Next, there are two places which you need to specify the location of the configuration file:

1. The **appSettings.config** file in the web app install directory.
2. If you're using the .NET API, a parameter in the **API constructor method**.

appSettings.config

Exago BI contains a file called **appSettings.config** in the root folder of the install directory. This file is used for custom app settings which are automatically imported into **Web.config** during runtime.

This file may also be used to specify other important environmental conditions, such as security protocols and input sanitation rules. More information about all available settings can be found in the Application Settings article.

NOTE

Do not edit Web.config file. It is automatically generated by Exago, and any changes will be overridden.

To set the config file location, add the following key to the appSettings.config file:

IMPORTANT

The connection strings used for configuration file and temporary file storage must be unique.

```
<add key="ExagoConfigPath" value="pathtype=azure;credentials='Azure Connection String';storagekey=config"/>
```

- *credentials*: Your Azure Credentials Connection String.
- *storagekey*: The prefix of a blob container used to store the config file.

CAUTION

When both the REST Web Service API and cloud config storage are being used, the ExagoConfigPath node must be included in both the REST Web Service API and Web Application **appSettings.config** files.

API

.NET API host apps cannot access the **appSettings.config** file. Instead, use one of the following two methods to specify a config file location:

1. Place the **config key** within the host application's web.config or app.config.
2. Or pass the connection string in the API constructor method:

```
Api api = new Api("/exago/virtual/path", "configFn.xml", "pathtype=azure;credentials='Azure Connection String';storagekey=config");
```

- *storagekey*: The prefix of the blob container used to store the config file.
- *configFn*: The name of the config file.

CAUTION

When both the REST Web Service API and cloud config storage are being used, the ExagoConfigPath node must be included in both the REST Web Service API and Web Application **appSettings.config** files.

Reports Storage

For versions *v2020.1+*, setup the **Storage Management** system for storing reports, templates and themes.

Upgrade Install

If upgrading from a previous version of Exago, review these articles:

- Storage Management: Introduction
- Storage Management: Transitioning from Legacy Storage Methods
- Storage Management: Getting Started

New Install

If this is a new installation, review these articles:

- Storage Management: Introduction
- Storage Management: Getting Started
- Storage Management: Admin Console

Temporary Files Storage

Azure allows an App Service to be scaled up to multiple instances on separate servers. When implementing this configuration, take the following safeguards in order to prevent loss of user data.

Each instance of Exago BI has its own local temp directory, whose path can (optionally) be specified in the **Temp Path** setting in the config file (defaults to %INSTALLDIR%Temp).

In a scalable configuration, initial user calls will reach one instance, storing temp files on that server, but subsequent calls may reach a separate instance, which will not have those temp files in its local directory. There are two solutions to resolve this issue:

- Temp Cloud Service
- Azure Affinity Cookie

Temp Cloud Service

This is Exago's built-in solution for handling multiple instances. Specifying a **Temp Cloud Service** causes each instance to push its temporary files to a shared Blob container. Then if a subsequent user call reaches a separate instance, that instance will pull the relevant files from the Blob to its local temp directory. Temp files may only be stored in blob storage.

To set up an Azure storage resource for temporary files:

1. Create a container for temporary files with the Azure Storage Explorer. The default container name is wrtemp but any container name, except for those used for configuration storage may be used by providing a storagekey in the connection string in step 2.
2. Set the **Admin Console** > **General** > **Main Settings** > **Temp Cloud Service** to a formatted connection string.

> **IMPORTANT**
>
> The connection strings used for configuration file and temporary file storage must be unique.

The connection string uses the following format:

```
pathtype=azure;credentials='Azure Connection String';
```

to save temporary files to the default blob container named wrtemp or

```
pathtype=azure;credentials='Azure Connection String';storagekey=tempFileStorageKey
```

to save temporary files to a blob container named tempFileStorageKey.

Examples:

```
type=azure;credentials='DefaultEndpointsProtocol=https;AccountName=acctName;AccountKey=encryptedalphanumeric';
```

Store temporary files in the default wrtemp blob container

```
type=azure;credentials='DefaultEndpointsProtocol=https;AccountName=acctName;AccountKey=encryptedalphanumeric';storagekey=ourExagoTempFiles;
```

Store temporary files in a blob container named ourExagoTempFiles

Azure Affinity Cookie

Azure supports setting Affinity Cookies, which track which server instance each user is connected to and cause all calls within the session to reach the same instance.

In your app service, navigate to **Application Settings**. Set ARR Affinity to **On**.



Virtual Machine

Exago can be hosted on a Windows-based Azure Virtual Machine. Installing Exago on a VM differs only marginally from installing it on a local machine. Therefore, this guide will not go into depth on this method. For a full installation and setup guide, see the **Exago Installation Guide**.

You can interact with a VM using either a remote desktop application or a command shell application.

Remote Desktop

1. Using Remote Desktop Connection or another remote desktop application, view your VM as a desktop environment.
2. Use a Web Browser to download the Exago Installer from our **support site**.
3. Run the installer as Administrator and install Exago.
4. Configure Exago (see **Install and Configure**).

Command Shell

1. On a local machine, use the above steps 2-4 to create a temporary Exago installation.
2. Remote into your VM using Windows PowerShell or another command shell application.
3. Copy the Exago directory to a directory on your VM.
4. Configure Windows and IIS (see **Manual Application Installation**).

After configuring IIS, open up the Exago port using a Windows Firewall inbound exception rule. You can then access Exago through the VM's IP address. Set up DNS and data security as desired.

Azure App Services

> **CAUTION**
>
> Deploying the Exago Web Application in an Azure App Service also requires a separate Azure Virtual Machine and the Exago Scheduler Service for handling chart and map rendering. Consider using an Azure Virtual Machine for the entire deployment instead. See Configuring Remote Report Execution Host for Azure App Service Deployments for more information.

The following Azure resources are required:

- App Service plan
- App Service

> **NOTE**
>
> You may require a Storage Account to run Exago as a scalable app. See **File Storage** for details.

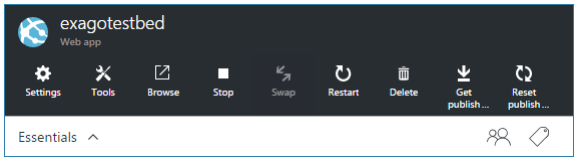
This section is divided into three parts:

- Hosting Exago BI in an App Service
- Using Exago Web Service API in an App Service
- Using the .NET API with Azure

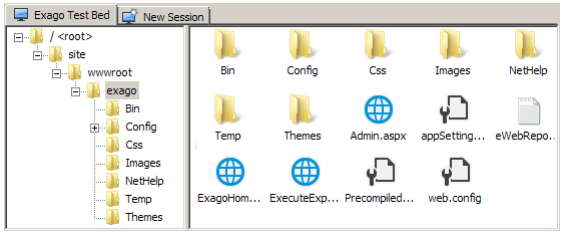
Hosting Exago BI in an App Service

> **NOTE**
>
> This walk-through requires a local Exago installation. See the **Exago Installation Guide** for details.

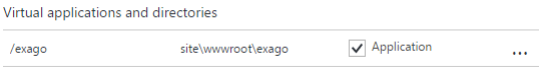
1. In your Azure Dashboard, create a new App Service container or navigate to an existing one.



2. In the App Service, navigate to **Deployment Credentials**. Add a username and password. This will allow you to FTP into your app service to transfer files.
3. Next, you'll need an FTP application. Open a connection using the deployment credentials created in step 2. Copy your local Exago BI web app installation directory to the app service container.



4. In the App Service, navigate to **Application Settings**. Set the following:
   - .NET Framework version: *v4.0* or Later
   - Managed Pipeline Version: *Integrated*
5. Under **Virtual applications and directories** create a virtual directory path to the installation directory and check the *Application* check box.



6. Click **Save** to save your settings.
7. Configure the Remote Report Execution Host as a Virtual Machine.
8. Test your installation and by navigating to the *WebAppUrlvirtual path to Exago App ServiceAdmin.aspx* page (Admin Console).

If you will use Azure Blob Storage to store the config file, follow the **File Storage** instructions before setting the base config. We also recommend setting up a state server either with SQL Server or another database when running Exago in an Azure App Service. Review the Setting Up a State Server article or the Exago Support Team for assistance.

Using Exago Web Service API in an App Service

> **NOTE**
>
> This walkthrough requires either a local Exago Web Service API installation, or a downloadable ZIP file containing the WebServiceApi directory. See the **Installing and Configuring the Web Service** article for details.

1. Copy the contents of the local Exago installation's WebServiceApi folder to the app service container.
2. Under **Virtual applications and directories** create a virtual directory path to the WebServiceApi directory and check the *Application* check box.
3. Click **Save** to save your settings.
4. In Azure, edit the **WebServiceApiConfigWebReportsApi.xml** file as follows:

```
<apppath>virtual path to Exago App Service</apppath>
<webreportsbaseurl>virtual path to Exago App Service</webreportsbaseurl>
```

> **NOTE**
>
> Include a trailing slash for both the apppath and webreportsbaseurl elements.

5. In the **WebServiceApiappSettings.config** file, add a new key to enable the REST service.

```
<appSettings><add key="ExagoREST" value="TRUE" /></appSettings>
```

The result should be the following:

```
<?xml version="1.0"?>
<appSettings>
  <add key="ExagoREST" value="True" />
</appSettings>
```

6. To prevent front end access without authentication outside of the API, in the **Admin Console**, navigate to **General** > **Main Settings** > set **Allow direct access to Exago (bypassing API)** to *False*. Click **OK**. Test your installation by navigating to the *WebAppUrlExagoApi.aspx* page. An 'Unauthorized access. Please contact your administrator' error message should appear.
7. To test that the path was set correctly, navigate to *WebAppUrlvirtual_path_to_Exago_App_ServiceApi.aspx*. A test file listing avaiable supported operations in the API should appear.
8. To prevent unauthorized**Admin Console** access to prevent users from inadvertently reaching it, delete *WebAppUrlExagoAdmin.aspx* and *WebAppUrlExagoBinadmin.aspx.cdcab7d2.compiled*.

> **TIP**
>
> Make a backup of the files before deletion in case **Admin Console** may be needed later.

9. Test that **Admin Console** is no longer accessible by navigating to *WebAppUrlExagoAdmin.aspx*. A 404 error page should appear.
10. Test that REST is enabled by utilizing an advanced REST client such as Advanced REST Client. Make a POST request to *WebAppUrlvirtual_path_to_Exago_App_Servicerestrestsessions*.

Header Name	Header Value
Accept	application/json
Content-Type	application/json
Authorization	The REST authorization key. Review the Introduction to REST article for more information.

Copy the **AppUrl** received in the response and append it to the end of the browser's URL *WebAppUrlvirtual path to Exago App Service* The Exago front end should load, proving that the application can only be reached through the API.

11. Configure the Remote Report Execution Host as a Virtual Machine.

Using the .NET API with Azure

Exago .NET API based host applications must be compiled locally before being uploaded to the Azure app container. References to the WebReports.dll library should be updated manually, and the program recompiled, when upgrading to a new Exago version.

Set your API constructor to use the previously set Exago virtual path:

```
Api api = new Api(@"/exago/virtual/path");
```

> **IMPORTANT**
>
> .NET host apps can only access virtual paths (and not URL paths). Therefore they must be located in the same App Service container as the Exago web app.

Exago utilizes the virtual directory passed in the first argument of this constructor to locate the required .dll libraries to load Exago. In cases where the virtual directory is not specified within the API constructor (e.g., if Exago is installed on the root of the .NET host application), then a reference to the Microsoft.WindowsAzure.Storage.dll library will need to be added manually to the host application.

Installing Exago on Docker

Requirements

This article requires that Docker already be installed and configured on the host. Consult Docker's official documentation for assistance with installation.

Supported Operating Systems

Exago currently can be installed in Docker images with the following operating systems:

- Ubuntu
- CentOS

Review Exago's Technical Specifications article for supported versions of Linux with Exago. Other distributions may be used, contact Exago Support for assistance. Exago does not recommend using Windows in Docker containers.

Best Practices and Recommendations

As each use case is different, there are several variables that contribute to decisions in the deployment of Exago in Docker containers. However, at a minimum:

- System administrators should carefully analyze their Docker resource constraints, and consider limiting access to memory and CPU time. As a best practice, do not allow Docker containers to consume all memory or CPU time on the host. System requirements for a Docker container are similar to those of a Linux virtual machine, therefore the Docker host should be setup with similar specifications. Review Considerations When Sizing an Exago System.
- Use **Docker volumes** to persist data such as configuration files, temp files, logs, application settings and scheduled jobs. This way, critical files are available to all containers or between container starts and stops. In some situations, such as when the Docker host is based in the cloud, volumes may not be an available option. Review the File Storage section of the Installing Exago on Azure article and the Amazon S3

File Storage article for cloud storage solutions in this case.
- Consider implementing the Scheduler Queue. The Scheduler Queue moves the Scheduler Service's job files off of the container to the queue's database. In addition, the Scheduler Queue's design to work with services that go on and offline fits more closely with Docker's deployment paradigm than isolated Scheduler Services.
- Consider the recommendations of the High Availability article
- Review and implement the Security Checklist

Procedure

A general procedure for installing Exago on Docker is as follows:

1. Download the Exago Linux Installer
2. *Optional:* Create an Exago configuration file that the image file created in step 5 will use
3. *Optional:* Create Docker volumes for persistent storage of temp files, config files, etc…
4. Write a Dockerfile to create a Docker image of the Exago installation
5. Build the image and launch containers

1. Download the Exago Linux Installer

From the Exago Downloads Page, download the latest version of the Exago Linux Installer. Be sure to review the Updating to the Latest Version and Updating Recommendations articles for important information.

2. Create an Exago Configuration File

This step is optional. With the Admin Console of an existing Exago installation on the same operating system and Exago version, a file can be created with a familiar graphical user interface. Copying this configuration file into the image can be automated, so once the container is launched the system is up and running.

Scheduler Service configuration files can also be prepared ahead of time, then copied into the image.

3. Create Docker volumes for persistent storage

This step is optional. Depending on implementation, it may be desirable to persist storage of temporary files, configuration files, Storage Management databases or others.

For example, a cluster of Docker containers working in a web farm arrangement will need to share temporary files, even if some of the containers go offline. Additional containers will require access to the same share as they are brought up.

Review Exago's Installation and Configuration documentation section and Docker's documentation for information about the intricacies for each environment.

4. Write a Dockerfile

Consult Docker's documentation and Exago Support for full details. Since every deployment is unique, one approach may be:

1. Begin with an operating system (e.g. `FROM ubuntu`)
2. Install a web server with the OS package manager
3. Download or copy the Linux installer to the image
4. Install Exago with mono from the command line
5. Copy the Exago configuration file created in step 2 to the image
6. Set permissions on directories
7. Set Applications Settings if required
8. Start the web server

5. Build the Image from the Dockerfile

Call `docker build` on the command line.

Once built, launch Docker containers from the new image as necessary.

Configuring Remote Report Execution Host for Azure App Service Deployments

When Exago is deployed as an Azure App Service, some additional configuration is necessary. At a high level, these configuration steps are:

1. The Exago **Scheduler Service** must be installed and properly configured on an Azure Virtual Machine. Exago *v2019.1.5+* must be installed for both the Azure App Service and the Virtual Machine, and the version numbers must be the same.
2. On the Azure App Service installation, enable the Scheduler and Remote Execution.

Detailed information about each of these requirements follows.

1. Azure Virtual Machine

Exago's **Puppeteer** image rendering libraries were introduced in *v2019.1.5* of the application. At a minimum, *v2019.1.5* must be installed on the Azure Virtual Machine. The version number of the Scheduler Service must match the version number of the Web Application.

Refer to the articles in the Recommended References section below for complete details on the installation and configuration of the **Scheduler Service** and how to install Exago on a Virtual Machine in Azure.

Recommended References

- Installing Exago on Azure

- Updating to the Latest Version
- Installing the Scheduler Service
- Configuring the Scheduler Service

2. Settings

The **Scheduler Service** should already be properly installed and configured on the Azure Virtual Machine.

On the Exago installation running as an Azure App Service, there are four parameters in the **Admin Console** which direct it to schedule and execute reports on the Virtual Machine instead of in the App Service:

- **General** > **Scheduler Settings** > **Enable Report Scheduling** must be set to *True*. By default, the value is *False*.
- **General** > **Scheduler Settings** > **Enable Remote Report Execution** must be set to *True*. By default, the value is *False*.
- **General** > **Scheduler Settings** > **Schedule Remoting Host** must be set to the virtual machine Scheduler Service's protocol, host name/IP address and port. (e.g. http://a.b.c.d:2000)
- **General** > **Scheduler Settings** > **Remote Execution Remoting Host** must be set the same as **Schedule Remoting Host** above.

Recommended References

- Admin Console Scheduler Settings

Installing Optional Features

Several features require additional configuration before they can be used.

Google Maps

> **IMPORTANT**
>
> Exago's Google Maps wizard uses the Google Maps API, which requires a paid license for commercial use. You must acquire a Google Maps API license in order to enable this feature.

There are additional steps needed in order to enable the new Google Maps wizard introduced in *v2016.3*.

1. Download and install the **Google Maps Polygon Database** for free from our support site.
2. Extract the polygon database into the **MapCache** directory.
3. If Remote Execution is in use, the polygon database must also be extracted into a **mapCache** folder inside of each Scheduler Service's /bin directory.
4. Give the web server full permissions to read the mapCache directory.
   For Windows:
   1. Right click on the folder and click **Properties**.
   2. In the security tab click **Edit** then **Add**.
   3. Enter *iis_iusrs* then click **OK**.
   4. In the **Permissions for Config** window, select the *IIS_IUSRS* that was just created and check **Allow Full Control**. Click **OK**.

   For Linux:

   1. chown $apacheUserID MapCache
   2. chmod -R u+wr MapCache

Exago's Google Maps wizard uses the Google Maps API, which requires a paid license for commercial use. You must acquire a Google Maps API license in order to enable this feature. Please see the the **Maps** and **Places** > **Geocoding** sections of the Google Maps Pricing Table for details. You must obtain a **Google Maps Javascript API** key and a **Google Maps Geocoding API** key, either as part of the same license, or separate.

> **NOTE**
>
> Licensing fees are subject to change by Google.

> **NOTE**
>
> For help with API keys, see Google API Console Help.

To install the license file:

1. Toggle **General** > **Feature/UI Settings** > **Advanced Report Designer Settings** > **Show Google Maps Wizard** in the **Admin Console** to *True*.
2. Place your unlimited license key or Google Maps JavaScript API key in the **Admin Console**: **General** > **Feature/UI Settings** > **Advanced Report Designer Settings** > **Google Map Key (unlimited or JS API restricted)**.
3. If you used a Google Maps JavaScript API limited key in the previous setting, place your Google Maps Geocoding API key in **General** > **Feature/UI Settings** > **Advanced Report Designer Settings** > **Google Map Key (optional Geocode API restricted)**.

Manual Application Installation

If the host application is deployed on site it may prove convenient and advantageous to integrate the installation of Exago into the hostâ€™s installer. This section will detail how to integrate the installation. To accomplish this task there must be an existing installation of the Exago Web Application, Exago Web Service API and Exago Scheduler Service from which to copy files and directories.

This section will show how to integrate the installation of:

- Web Application and Exago Web Service API
- Scheduler Service

Web Application and Web Service API Installer Integration

Summary

The installer integration of the Exago Web Application and/or the Exago Web Service API has four steps:

1. Copy the Exago and/or the Exago Web Service API files to installation folders.
2. Create IIS Virtual Directory to point to Web Application/Web Service API.
3. Configure IIS as required for the Web Application/Web Service API setup.
4. *Optional:* Modify the system registry.

> **NOTE**
>
> This article references <WebApp>/, <WebSvc>/ and <Sched>/as a placeholder for the Web Application, Web Service API and Scheduler Service's install location respectively. The default install location is C:Program FilesExagoExagoWeb, C:Program FilesExagoExagoWebApi or C:Program FilesExagoExagoScheduler; however, these directories can be changed during installation.

1. Directory Structure

The directory structure should be preserved as follows:

Web Application

- <WebApp>/
  - /ApplicationThemes
  - /Bin
  - /Config
  - /Drivers
  - /Fonts
  - /Licenses (*v2019.2.39+*, *v2020.1.21+*, *v2021.1.9+*)
  - /MapCache
  - /Monitoring
  - /NetHelp
  - /Temp
  - /Themes

REST Web Service API

- <WebSvc>/
  - /Bin
  - /Config
  - /Drivers

File Installation

The host application installer should create a copy of all the files that are initially created by the Web Application and/or Web Service API installer.

For *v2019.5+*, extract the Puppeteer library files from chromium-win64.zip into the bin directory for both the Web Application and the REST Web Service API (if applicable). This file can be found in the Webbin directory of the ZIP installation package.

Configuration

The following configuration files are not part of the initial Exago/Exago Web Service API installation. Including the configuration files with the installation will help to minimize manual configuration. The files are stored in the following directory tree:

Web Application

- /Config/
  - xml and/or WebReports.xml.enc

Web Service API

- /Config/
  - xml

3. IIS Configuration

> **IMPORTANT**
>
> Verify that the Virtual Directory does not exist before attempting to create the new one.

The following is a C# code sample of how to create a new IIS installation of Exago/Exago Web Service API, using

Microsoft.Web.Administration.dll. The code requires the following input:

- **siteName** â€" Name of the IIS Web Site where it will be installed. (e.g. *Default Web Site*)
- **vDirName** â€" Name of Virtual Directory for the installation (e.g. *Exago* or *ExagoApi*)
- **physicalPath** â€" Physical installation path. (e.g. *C:Program FilesExagoExago*)

```
public new void CreateVDir(string siteName, string vDirName, string physicalPath)
{
try
{
ServerManager iisManager = new ServerManager();
string virtDirName = @"/" + vDirName;

// Check if Application/Virtual Directory exists
if (iisManager.Sites[siteName].Applications[virtDirName] != null)
{
iisManager.Sites[siteName].Applications[virtDirName].VirtualDirectories[@"/"].PhysicalPath =
 physicalPath;
}
// Create new Application/Virtual Directory
else
{
iisManager.Sites[siteName].Applications.Add(virtDirName, physicalPath);

Microsoft.Web.Administration.Application app =
iisManager.Sites[siteName].Applications[virtDirName];

app.ApplicationPoolName = "DefaultAppPool";
}

// Commit changes to the webserver
iisManager.CommitChanges();
}
catch
{
throw;
}
}
```

4. Modify the Windows Registry

See Windows Registry below.

Scheduler Service Installer Integration

Summary

The installer integration of the Scheduler Service has six steps:

1. Check to see if the Scheduler Service is running as a Windows Service. If yes, stop the service.
2. Copy the Scheduler Service files to installation folders.
3. *Optional:* Modify the system registry.
4. Modify the security settings on the Scheduler Service installation directory.
5. Create a new Windows Service for the Scheduler Service.
6. Enable the Scheduler Service's Windows Service.

1. Check Windows Services

Before running the installation, the Windows Services should be checked to see if the Scheduler Service is currently installed and/or running as a service. If the Exago Scheduler is currently installed and/or running as a service it should be stopped.

The following C# code provides an example of how to stop the Scheduler Service if it is running.

```
ServiceState serviceSt = WindowsServiceInstaller.GetServiceStatus(â€œExagoSchedulerâ€);

// check to see if the Exago Scheduler service exists
if (serviceSt != ServiceState.NotFound && serviceSt != ServiceState.Unknown)
{
CreateServiceDelegate stDel = new CreateServiceDelegate(WindowsServiceInstaller.StopService);
stDel(â€œExagoSchedulerâ€);

for (int ProgCtr = 0; ProgCtr <= 120; ProgCtr++)
{
Thread.Sleep(1000);
serviceSt = WindowsServiceInstaller.GetServiceStatus(â€œExagoSchedulerâ€);

if (serviceSt == ServiceState.Stop)
break;

if (InvokeRequired)
Invoke(new Change(OnChange), ProgCtr);

(sender as BackgroundWorker).ReportProgress(ProgCtr);
}
}
```

2. File Installation

The host installer should then create a copy of all the files that are initially created by the Exago Scheduler Installer.

For *v2019.5+*, extract the Puppeteer library files from chromium-win64.zip into the Scheduler Service's directory. This file can be found in the Scheduler directory of the ZIP installation package.

> **NOTE**
>
> Overwrite the file ExagoScheduler.xml with a version configured for the host application.

3. Modify the Windows Registry

See Windows Registry below.

4. Directory Security Settings

Changes to the security settings of the installation directory are required to enable Windows to run it as a Windows Service.

The following C# code provides an example of how to make the necessary security changes. It requires the following input.

- **dirName** â€“ Physical installation path of Scheduler Service (e.g. *C:Program FilesExagoExagoScheduler*)

```
private void SetDirSecurity(string dirName)
{
try
{
if (dirName == null)
return;

if (!Directory.Exists(dirName))
return;

DirectoryInfo dirInfo = new DirectoryInfo(dirName);

// get a DirectorySecurity object that represents the current
// security settings
DirectorySecurity dirSecurity = dirInfo.GetAccessControl();

// Add the FileSystemAccessRule to the security settings
dirSecurity.AddAccessRule(new FileSystemAccessRule("LOCAL SERVICE",
FileSystemRights.FullControl, AccessControlType.Allow));
dirSecurity.AddAccessRule(new FileSystemAccessRule("LOCAL SERVICE",
  FileSystemRights.FullControl,
InheritanceFlags.ContainerInherit | InheritanceFlags.ObjectInherit,
PropagationFlags.InheritOnly, AccessControlType.Allow));

// Set the new access settings
try
{
dirInfo.SetAccessControl(dirSecurity);
}
catch (Exception ex)
{
MessageBox.Show(ex.Message);
}
}
catch (Exception ex)
{
MessageBox.Show(this,"Unable to set privileges on install directory: " + dirName +
 ".  Please set 'LOCAL SERVICE' privileges.nnException: " + ex.Message, "Error");
}
}
```

5 & 6. Windows Service Creation and Startup

Before installing the Scheduler Service as a new Service, verify that it is not installed and/or running. If the Exago Scheduler is not installed, install the software and make sure it is running.

The following C# code provides an example of how to make this check.

```
serviceSt = WindowsServiceInstaller.GetServiceStatus(â€œExagoSchedulerâ€);
// is Exago Scheduler already installed as a service
if (serviceSt == ServiceState.NotFound || serviceSt == ServiceState.Unknown)
{
// install Exago as a new Windows Service
WindowsServiceInstaller.Install(â€œExagoSchedulerâ€,â€œExagoSchedulerâ€,
filePath + â€œExagoScheduler.exeâ€);


for (int timeCtr = 0; timeCtr <= 120; timeCtr++)
{
serviceSt = WindowsServiceInstaller.GetServiceStatus(â€œExagoSchedulerâ€);

if (serviceSt == ServiceState.Stop)
{
break;
}

if (InvokeRequired)
Invoke(new Change(OnChange), timeCtr);

(sender as BackgroundWorker).ReportProgress(timeCtr);
}

RegistryKey key = Registry.LocalMachine.OpenSubKey("SYSTEMCurrentControlSet\Services\" +
 â€œExagoSchedulerâ€, true);

if (key != null)
{
key.SetValue("Description", "Exago Scheduler Windows Service");
}
}
// found service already installed, check to see if it is running
else
{
// if the service is not running, attempt to start it
if (this.initialStatus != ServiceState.Stop)
{
CreateServiceDelegate stDel = new CreateServiceDelegate(WindowsServiceInstaller.StartService);
stDel(â€œExagoSchedulerâ€);

for (int timeCtr = 0; timeCtr <= 120; timeCtr++)
{
serviceSt = WindowsServiceInstaller.GetServiceStatus(â€œExagoSchedulerâ€);

if (serviceSt == ServiceState.Starting || serviceSt == ServiceState.Run)
{
break;
}

if (InvokeRequired)
Invoke(new Change(OnChange), timeCtr);

(sender as BackgroundWorker).ReportProgress(timeCtr);
}
}
}
```

Windows Registry

Windows Registry keys will be written by the Exago Installer when installing the Web Application, the Web Service API and the Scheduler Service. The changes to the registry are documented in the article on Windows Registry Changes.

Example of Registry

The following C# code provides an example of how to add items to the registry. It requires the following arguments:

- **application** â€” Set to *Exago*, *ExagoApi* or *ExagoScheduler*.
- **path** â€” Set to the installation path where the component is installed.
- **website** â€” Set to the IIS Web Site where Exago is installed. Leave blank for the Exago Scheduler.
- **vdir** â€” Set to the virtual directory that Exago is set up as. Leave blank for the Exago Scheduler.

```
public static void AddRegistryKey(string application, string path, string webSite, string vdir)
{
try
{
 string ExagoRegKey = application;
 if (application != â€œExagoSchedulerâ€)
{
 vdir = vdir.Replace(@â€â€, @â€/â€);
  ExagoRegKey += @â€â€ + webSite + @â€/â€ + vdir;
 }

RegistryKey registryKey = Registry.LocalMachine.OpenSubKey(REGISTRY_KEY_ROOT +
  ExagoRegKey, true);
if (registryKey == null)
{
registryKey = Registry.LocalMachine.CreateSubKey(REGISTRY_KEY_ROOT
+ ExagoRegKey);
if (registryKey == null)
throw (new Exception("Error creating RegistryKey"));
else
registryKey.SetValue("CreateDate",
 System.DateTime.Now.ToString(CultureInfo.InvariantCulture));
}
using (registryKey)
{
registryKey.SetValue("DisplayName", ExagoRegKey);
registryKey.SetValue("UpdateDate",
System.DateTime.Now.ToString(CultureInfo.InvariantCulture));
registryKey.SetValue("Location", path);
registryKey.SetValue("Version",
System.Reflection.Assembly.GetExecutingAssembly().GetName().Version);
}

return;
}
catch
{
throw;
}
}
```

Windows Registry Changes

Windows Registry keys will be written by the Exago Installer when installing the Web Application, the Web Service API and the Scheduler Service.

Web Application

The Web Application will add four new keys to the registry. Each key and its contents are described in the tables below.

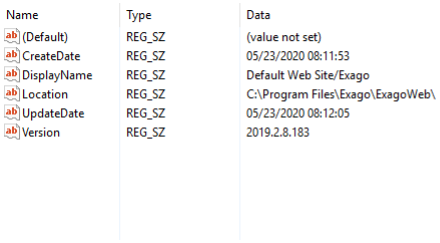
> **NOTE**
>
> This section references <web site name>/<virtual directory name> as a placeholder for the IIS web site and virtual directory configured during the installation process. The default values are *Default Web Site* and *Exago* respectively.

1. **HKEY_LOCAL_MACHINE\SOFTWARE\Exago\ExagoWeb\<web site name>/<virtual directory name>**
   This key is used by the installer to determine the installed version number and the location of the installation. It will be updated with each subsequent installation of Exago into the same web site and virtual directory.

Table A

Name	Type	Description Example
CreateDate	REG_SZ	Timestamp of the first Exago Web Application installation. e.g. 05/19/2020 15:30:53
DisplayName	REG_SZ	Installation Web Site followed by the Virtual Directory Name e.g. Default Web Site/Exago
Location	REG_SZ	Physical location of the installation e.g. C:Program Files\Exago\ExagoWeb
UpdateDate	REG_SZ	Initially set to the installation date. Should be updated whenever Exago is reinstalled. e.g. 05/23/2020 08:12:05
Version	REG_SZ	The version of Exago installed. This value can be found by pressing â€¯Ctrl + Shift+ Vâ€™ in the Web Application. e.g. 2019.2.8.183

Example:



Name	Type	Data
(Default)	REG_SZ	(value not set)
CreateDate	REG_SZ	05/23/2020 08:11:53
DisplayName	REG_SZ	Default Web Site/Exago
Location	REG_SZ	C:\Program Files\Exago\ExagoWeb\
UpdateDate	REG_SZ	05/23/2020 08:12:05
Version	REG_SZ	2019.2.8.183

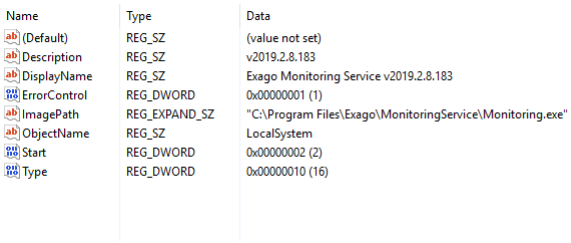
2. **HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\ExagoMonitoringService.version**
   This key represents the Windows Service that manages the Exago Monitoring Service, which is installed automatically with the Web Application.

> **NOTE**
>
> This section references version as a placeholder for the version of the service installed. For example, HKEY_LOCAL_MACHINE\SOFTWARE\Exago\ExagoMonitoringService.2019.2.183 As this key has the Exago version in it's name, on subsequent update installations of Exago, new keys will be written each time instead of updating this current one.

Table B

Name	Type	Description Example
Description	REG_SZ	The description text to display in the Windows Service Control Manager. Typically, the version of Exago installed. e.g. 2019.2.8.183
Display Name	REG_SZ	Name to display in the Windows Service Control Manager e.g. Exago Monitoring Service v2019.2.183
ErrorControl	REG_DWORD	Severity of the error if this service fails to start during startup e.g. 0x1
ImagePath	REG_EXPAND_SZ	Fully qualified path to the Monitoring Service's executable file e.g. "C:Program Files\Exago\MonitoringService\Monitoring.exe"
ObjectName	REG_SZ	The name of the account under which the service executes e.g. LocalSystem
Start	REG_DWORD	Defines when to start the service e.g. 0x2
Type	REG_DWORD	Type of service e.g. 0x10

Example:



Name	Type	Data
(Default)	REG_SZ	(value not set)
Description	REG_SZ	v2019.2.8.183
DisplayName	REG_SZ	Exago Monitoring Service v2019.2.8.183
ErrorControl	REG_DWORD	0x00000001 (1)
ImagePath	REG_EXPAND_SZ	"C:\Program Files\Exago\MonitoringService\Monitoring.exe"
ObjectName	REG_SZ	LocalSystem
Start	REG_DWORD	0x00000002 (2)
Type	REG_DWORD	0x00000010 (16)

3. **HKEY_LOCAL_MACHINE\SOFTWARE\Exago\ExagoMonitoringService.version**
   This key is used by the Exago installer to determine the installed version number and the location of the installation.

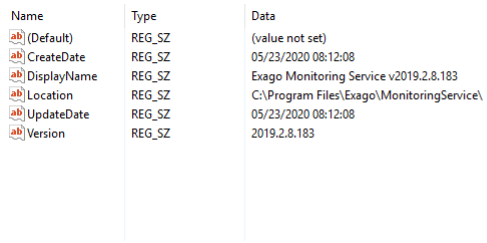
> **NOTE**
>
> This section references version as a placeholder for the version of the service installed. For example, HKEY_LOCAL_MACHINE\SOFTWARE\Exago\ExagoMonitoringService.2019.2.183 As this key has the Exago version in it's name, on subsequent update installations of Exago, new keys will be written each time instead of updating this current one.

Table C

Name	Type	Description Example
Description	REG_SZ	Timestamp of the first Exago Web Application installation. e.g. 05/19/2020 15:30:53
DisplayName	REG_SZ	Name to display in the Windows Service Manager e.g. Exago Monitoring Service v2019.2.183
Location	REG_SZ	Physical location of the installation e.g. C:Program Files\Exago\MonitoringService
UpdateVersion	REG_SZ	Initially set to the installation date. Should be updated whenever Exago is reinstalled. e.g. 05/23/2020 08:12:05

Version	REG_SZ	The version of Exago installed. This value should match that of the Web Application. e.g. 2019.2.8.183

Example:



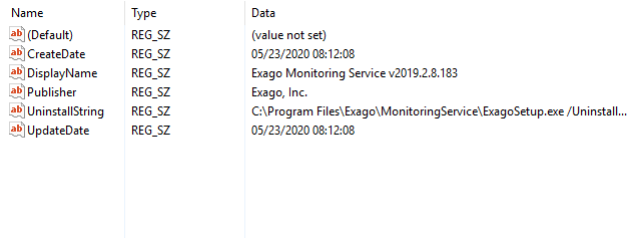
4. **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\ExagoMonitoringService.version**
   This key is used in the Windows Add/Remove Programs Control Panel to allow uninstallation of the Monitoring Service from there.

> **NOTE**
>
> This section references version as a placeholder for the version of the service installed. For example, HKEY_LOCAL_MACHINE\SOFTWARE\Exago\ExagoMonitoringService.2019.2.183 As this key has the Exago version in it's name, on subsequent update installations of Exago, new keys will be written each time instead of updating this current one.

Table D

Name	Type	Description Example
CreateDate	REG_SZ	Timestamp of the first installation of this particular version of the Monitoring Service. e.g. 05/19/2020 15:30:53
DisplayName	REG_SZ	Name to display in the Windows Add/Remove Programs Control Panel e.g. Exago Monitoring Service v2019.2.8.183
Publisher	REG_SZ	Exago, Inc.
UninstallString	REG_SZ	The command line string to start the uninstallation process e.g. C:Program Files\Exago\MonitoringService\ExagoSetup.exe /UninstallMonitoring ExagoMonitoringService.v2019.2.8.183
UpdateDate	REG_SZ	Initially set to the installation date. Should be updated whenever this particular version of the Monitoring Service is reinstalled. e.g. 05/23/2020 08:12:05

Example:



Scheduler Service

The Scheduler Service installation adds three new keys to the registry. Each key and its contents are described in the tables below.

> **NOTE**
>
> This section references <svc_name> as a placeholder for the name of the service configured during the installation process. The default value is *ExagoScheduler*.

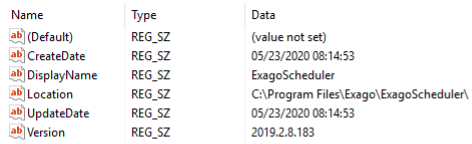
1. **HKEY_LOCAL_MACHINE\SOFTWARE\Exago\eWebReportsScheduler_<svc_name>**
   This key is used by the installer to determine the installed version number and the location of the installation. It will be updated with each subsequent installation of Exago with the same service name.

Table E

Name	Type	Description Example
CreateDate	REG_SZ	Timestamp of the first Scheduler Service installation. e.g. 05/19/2020 15:30:53
DisplayName	REG_SZ	Friendly display name of the Scheduler Serivce e.g. ExagoScheduler
Location	REG_SZ	Physical location of the installation e.g. C:Program FilesExagoExagoScheduler
UpdateDate	REG_SZ	Initially set to the installation date. Should be updated whenever the SCheduler Service is reinstalled.

		e.g. 05/23/2020 08:12:05
Version	REG_SZ	The version of the Scheduler Service installed. e.g. 2019.2.8.183

Example:



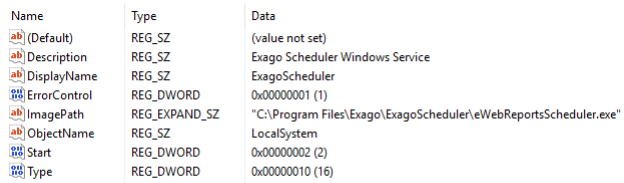
2. **HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Service\seWebReportsScheduler_<svc_name>**

This key represents the Windows Service that manages the Scheduler Service, which is installed automatically with the Web Application.

Table F

Name	Type	Description Example
Description	REG_SZ	The description text to display in the Windows Service Control Manager. e.g. Exago Scheduler Windows Service
Display Name	REG_SZ	Name to display in the Windows Service Control Manager. This is the service name that was provided at the time of installation. e.g. ExagoScheduler
ErrorControl	REG_DWORD	Severity of the error if this service fails to start during startup e.g. 0x1
ImagePath	REG_EXPAND_SZ	Fully qualified path to the Scheduler Service's executable file e.g. "C:Program FilesExagoExagoSchedulereWebReportsScheduler.exe"
ObjectName	REG_SZ	The name of the account under which the service executes e.g. LocalSystem
Start	REG_DWORD	Defines when to start the service e.g. 0x2
Type	REG_DWORD	Type of service e.g. 0x10

Example:



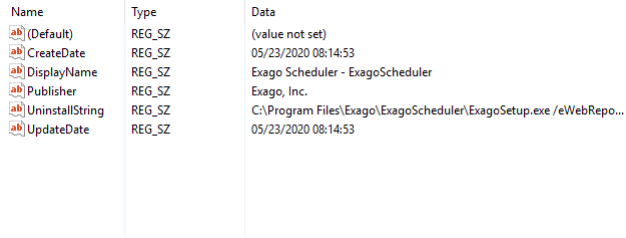
3. **HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\eWebReportsScheduler_<svc_name>**

This key is used in the Windows Add/Remove Program Control Panel to allow uninstallation of the Monitoring Service from there.

Table G

Name	Type	Description Example
CreateDate	REG_SZ	Timestamp of the first installation of this particular version of the Scheduler service. e.g. 05/19/2020 15:30:53
DisplayName	REG_SZ	Name to display in the Windows Add/Remove Programs Control Panel. Typically the words "Exago Scheduler – " followed by the name of the service set during installation. e.g. Exago Scheduler - ExagoScheduler
Publisher	REG_SZ	Exago, Inc.
UninstallString	REG_SZ	The command line string to start the uninstallation process e.g. C:Program FilesExagoExagoSchedulerExagoSetup.exe /eWebReportsScheduler_ExagoScheduler
UpdateDate	REG_SZ	Initially set to the installation date. Should be updated whenever this particular version of the Scheduler Service is reinstalled. e.g. 05/23/2020 08:12:05

Example:



REST Web Service API

The REST Web Service API adds one registry key.

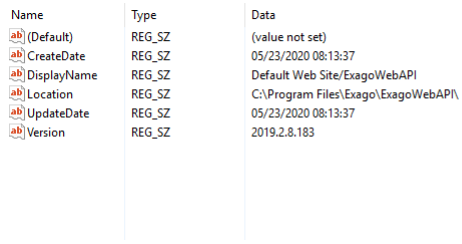
> **NOTE**
>
> This section references <web site name>/<virtual directory name> as a placeholder for the IIS web site and virtual directory configured during the installation process. The default values are *Default Web Site* and *ExagoWebAPI* respectively.

1. **HKEY_LOCAL_MACHINE\SOFTWARE\Exago\ExagoWebAPI<web site name>/<virtual directory name>**
   This key is used by the installer to determine the installed version number and the location of the installation. It will be updated with each subsequent installation of Exago into the same web site and virtual directory.

Table H

Name	Type	Description Example
CreateDate	REG_SZ	Timestamp of the first Exago Web Service API installation. e.g. 05/19/2020 15:30:53
DisplayName	REG_SZ	Installation Web Site followed by the Virtual Directory Name e.g. Default Web Site/ExagoWebAPI
Location	REG_SZ	Physical location of the installation e.g. C:Program Files\ExagoExago\WebAPI
UpdateDate	REG_SZ	Initially set to the installation date. Should be updated whenever Exago is reinstalled. e.g. 05/23/2020 08:12:05
Version	REG_SZ	The version of Web Service API installed. e.g. 2019.2.8.183

Example:



Installing the Scheduler Service

> **IMPORTANT**
>
> The version and build number of the **Scheduler Service** must match that of the **Web Application**. Due to this requirements, Scheduler Service's must run on the same operating system as their corresponding Web Application servers.

You may have different installations of Exago with different versions/builds on separate servers. The Scheduler Service installation wizard allows you to install multiple Schedulers to maintain corresponding version/builds with the Web Application.

Use the following steps to install the Scheduler Service:

1. Download the latest Windows or Linux installers.
2. Make sure your antivirus is software disabled and run the installation wizard as an Administrator.

For Windows

1. Click the **Scheduler** button.
2. Click **Next** to bring up the â€˜Select Installation Locationâ€™ menu.
3. Specify if you want to create a new service or if you want to update an existing one.
4. To create a new service set a name and location.
5. Select to who the Exago Scheduler Windows Service will be installed. By default, â€œEveryoneâ€ is selected. Click **Next**.
6. Confirm your location selections by clicking **Next**
7. Monitor the installation and click **Finish** when it is complete.

> **NOTE**
>
> As of *v2016.2.12*, schedulers take system resources into account when assigning remote execution jobs. This requires that the services have read access to the system registry. This can be done by adding the services to the **Performance Monitor Users** group. See How to Read Performance Counters Without Administrator Privileges (MSDN) for details.

For Linux

If you elect to install the Scheduler Service, the installer will create the sub directory "**Scheduler**" in your previously specified install path.

Follow the instructions in the Installing Exago on Linux article to install the Service. To configure the Service, continue to the Scheduler Configuration article.

Storage Management: Introduction

Exago's **Storage Management** system replaces the legacy *file system*, *cloud storage*, *folder management* and *web service (SOAP)* storage methods with a relational database. All reports, templates, folders and themes are stored in this database. Microsoft SQL Server, MySQL, Oracle and PostgreSQL are supported out-of-the-box. Exago also supports a SQLite database for development, testing and demonstration purposes only.

As of *v2020.1,* all legacy storage methods are deprecated and are replaced with the Storage Management system. Storage Management enables a flexible folder and report permissions model and creates a future development platform for features such as collaboration, annotation, version control and more.

Table of Contents

1. Content Storage
    1. Default Folders
    2. Supported Database Types
2. Permissions
    1. Identity Keys
        1. User Id
        2. Company Id
        3. Class Id
        4. Owner Id
    2. Access Flags
    3. Inheritance
    4. Query Tool
    5. Roles
3. Report Management
    1. Creating a New Report or Folder
    2. Moving a Report or Folder to a Different Folder
    3. Deleting a Folder
    4. Deleting a Report
4. Caching
5. Security Considerations
6. Transitioning to Storage Management
7. Customization
8. Getting Started
9. Additional Resources

Content Storage

All reports, folders, report template files, report themes, chart themes, and GeoChart themes are stored in a relational database. This content is stored in a table with some metadata for easy access (e.g. name, description, exports) and general object management (e.g. updated by, saved date).

Report definitions and themes are stored as plain-text in the database. Template files are stored as binary data.

Default Folders

Two new folders will be available for each user in the system:

- **My Reports**: visible to all users, but content saved in it is only available to the current user.
- **Public**: visible to all users and any content saved in it is available to all other users of the system. This provides a basic way of sharing report content with other users.

Review Permissions below to learn more about access controls with Storage Management.

If these folders already exist in the environment before upgrading, new ones will not be created. These folders can optionally be deleted if not desired.

Supported Database Types

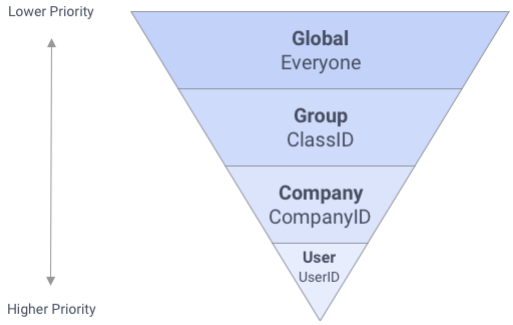
Exago's Storage Management implementation support the following database types for content storage:

- Microsoft SQL Server
- MySQL
- ORACLE
- PostgreSQL
- SQLite

> **IMPORTANT**
>
> Exago recommends using the included SQLite database for development, testing and demonstration purposes only. Use MSSQL, MySQL, Oracle or PostgreSQL in a production environment.

Permissions

Content permissions are based on pools of users called **parties**. The most specific is an individual user, the least is "everyone". In between are various levels of control based on identity key values. For example, company level access can be granted based on the value of the companyId key. These keys are discussed in more detail in the Identity Keys section below and in Admin Support Lab â€" Storage Management Permissioning.

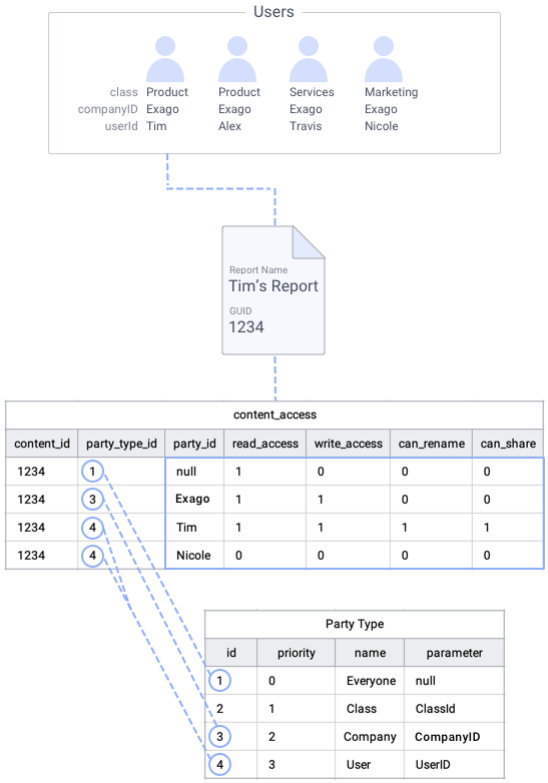


Identity keys work in a hierarchical structure that moves from less specific to more specific

When a user attempts to access content (e.g. a report), all of the access records that match the provided keys are sorted according to their priority. The first record from that set (the highest priority) provides the effective access to the content. That access may allow execution, editing, viewing, downloading or no access at all. If there are two content access records that apply to the same user (e.g. there is a match on their **companyId** and on theirÂ **userId** values) for a content item, the record with higher priority (**userId**) will apply over the other.

In the figure below, everyone except for user “Nicole” will be able to see “Tim’s Report” in their Report Tree. Only user “Tim” can read, write, rename or share “Tim’s Report”. Anyone in the same “Exago” company (“Travis”, “Tim” and “Alex”) can also write to it as well. Any user (except “Nicole” who is explicitly restricted from it), including those outside of “Exago”, can see the report.

This illustration provides a simplistic way of visualizing Storage Management access. In the actual implementation, access control flags such as “read_access” are bit flags in a single column. More information about these access flags follows.



For simplicity, each access flag is shown in it’s own column in this figure. In practice, access flags are expressed as a bitmap in a single access_flags column. Review the Storage Management: Database Schema article for more information.

Identity Keys

A new set of variables called **identity keys** are used to identify ownership and access rights to content in the Storage Management system. These keys are similar in purpose to Exago’s system parameters @ and @ but are specific to the Storage Management mechanism.

Each time an Exago session is created via the API, values for both the system parameters and any Storage Management identity keys should be set. Values for the keys may be set with .NET API calls and through a new REST Web Service API endpoint. Values are case-sensitive, and should be unique across organizations.

Setting the identity keys is not mandatory. However, the Storage Management system will still process access rules with null as the value for any unset key. Therefore, it is strongly recommended all identity keys are given values when accessing the application.

Four identity keys are implemented out of the box: **User Id**, **Company Id**, **Class Id** and **Owner Id**. Additional identity keys may be added by editing the configuration file and making appropriate changes to the Party Type table in the database.

For example, consider an employee named “Astro Boy” who works in the “Report Building” department of the company “Exago, Inc.” in Kingston, NY.

User Id

A unique identifier for a specific user of the application. For example, “Astro Boy” or “aboy”.

Company Id

A unique identifier for the organization or company that the User Id belongs to. This key has second priority in the hierarchy above the User Id. For example, “Exago, Inc.” or “Kingston”.

Class Id

A unique identifier for the user’s class. This key has third priority in the hierarchy above the User Id. For example, “administrator” or “report-builder”.

Owner Id

When a report or folder is created, the owner of that content is assigned to the value of the creator’s **Owner Id**. For example,Â “Astro Boy” or “aboy”.

When the Owner Id column of a content item matches the value of the **Owner Id** identity key, full permission (all access flags are set to *true*) is granted to that content item automatically.

In most cases **Owner Id** will have the same value as the **User Id** key, so that users have ownership of (and therefore full access rights to) content they create. This value doesn’t necessarily need to be an individual user. A company or class can also own content, and so those values are also valid.

> **IMPORTANT**
>
> If the **Owner Id** is not set when content is created, Exago will use the value for **User Id** in its place.

Owners of content will see the **Owner**

icon next to the content in their **Report Tree**.

Access Flags

Access Flags assigned to each content item enable or restrict functionality in the application.

Access Flags v2021.1+

- **Can Edit** â€” whether or not content may be edited/saved in its respective Report Designer. If *false*, content cannot be opened in a Report Designer, child folders cannot be added, the **Upload** option is not available, content may not be moved to the folder and content cannot be moved out of a folder. The **Run/Stop** button in the ExpressView Designer will be hidden and locked in the **Run** mode.
- **Can Rename** â€” whether or not content may be renamed. If *false*, the ability to change a content item’s name is unavailable, even though it may be moved to a different location (if **Can Edit** and **Can Move** permit that change).
- **Can Share** â€” this flag is for future functionality and is not implemented yet
- **Can Delete** â€” whether or not content may be deleted. When *false*, delete options are disabled for folders and reports. When *true*, folders must be empty for all users to be deleted.
- **Can Copy** â€” whether or not content may be duplicated or downloaded. When false, the **Duplicate** and **Download** options are not available. Setting this flag *false* will also prevent ExpressViews from being exported to Advanced Reports, and will disable the **Save Changes as New Report** option in the **Advanced Report Viewer**.
- **Can View** â€” whether or not the content is visible in the **Report Tree** or not. When false, the content item is hidden from view. This is useful for hiding components of **Chained Reports** or **Dashboards** from end users.
- **Can Schedule** â€” whether or not content may be scheduled. When false, the **Schedule Report** and **Email Report** options are not available. Has no effect on existing schedules, the **Schedule Wizard** or **Schedule Manager**.
- **Can Move** â€” whether or not content may be moved to a different location. In order to move a report, both the target and origin folders must have **Can Edit** set to *true*.

> **NOTE**
>
> Content that has the **Can Rename**, **Can Move**, **Can Delete** and **Can Edit** permissions all set to *false* will be considered read-only and marked with the read-onlyÂ
>
> icon. Folders and reports need the **Can View** permission to appear in the Report Tree.

When content is created, access permissions are either inherited from their parent; or a single content access record is written based on the parent’s default party type and access flag settings. If the parent’s access flags are all disabled (or missing) then the new content will use the **Default Access Flags** from the system configuration.

> **CAUTION**
>
> The Report Tree should contain no more than 1,000 items in it for best user experience.

Inheritance

A new concept for permissions management has been introduced with Storage Management, called **inheritance**. Functionally, inheritance allows content to â€œemulateâ€ roles without actually needing to use them. This is meant to provide a migration path away from roles for clients migrating from filesystem or cloud storage, who want to preserve filesystem behavior. However, clients migrating from folder management may want to turn inheritance off, as this behavior is not present in that system.

If a folder's **inherit flag** is *true*, when new content is saved into the folder that content will be assigned all of the same permissions as the folder. All of the content access records assigned to the parent folder will be copied to the new content item.

If a folder's **inherit flag** is *false*, new content in a folder will not inherit the permissions from the parent. Instead of all of the content access records being copied from parent to the new content, a new single content access record will be written with the default access flags and the default party type.

Examples of Inheritance

Example 1 â€" Default Folders

The new Default Folders **Public** and **My Reports** have the inherit_flag set to *false*.

- **My Reports**
  - **Inherit Flag**: *false*
  - **Default Party Type Id**: *4* (user)
- **Public**
  - **Inherit Flag**: *false*
  - **Default Party Type Id**: *1* (everyone)

When new content is created in the **My Reports** folder since the **inherit flag** is false, any content access records on the **My Reports** folder will not be copied to reports saved there. Instead, reports saved to the **My Reports** folder will have a single *user* level content access record written. The value of the **User Id** identity key will be saved into this access record, there by granting access to the content only to the current user.

When new content is created in the **Public** folder since the **inherit flag** is false, any content access records on the **Public** folder will not be copied to reports saved there. Instead, reports saved to the **Public** folder will have a single *everyone* level content access record written. *Everyone* access records apply to all users of the system, and therefore the access record will set access for all users.

If either folder's **inherit flag** were *true*, then any content access record on the respective folder would be copied to reports saved in them. This could for example, grant access to another user of a report that should only be visible to a single user in the **My Reports** folder.

Example 2 â€" A Departmental Folder

Consider a fictional folder for the "Sales Department" in a company. A simplified version of the content access records are shown in Table A to demonstrate how they work.

- **Sales Department**
  - **Inherit Flag**: *true*

Table A â€" Content Access Records for Sales Department folder

Content	Party Type	Party Value	Access Flags	Parent
Sales Department	2 (company)	Sales Dept	508	root
Sales Department	4 (user)	Mike B	511	root

When a new report, for example, *Quarterly Report* is saved to the **Sales Department** folder, all of the content access records in Table A would be copied on to the new report. Therefore, the resulting content access table looks like Table B below:

Table B â€" Content Access Records for Sales Department folder and Quarterly Report report

Content	Party Type	Party Value	Access Flags	Parent
Sales Department	2 (company)	Sales Dept	508	root
Sales Department	4 (user)	Mike B	511	root
Quarterly Report	2 (company)	Sales Dept	508	Sales Department
Quarterly Report	4 (user)	Mike B	511	Sales Department

Query Tool

Exago provides a database query tool called the **Storage Management Utility** that can read from the Storage Management database and display the content access records, access flags and inheritance on content in the system.

In *v2021.1+*, the utility is a separate download on the Support Site Downloads page. Documentation for this version can be found in Storage Management: Utility (v2021.1+).

Roles

The access controls built into Roles essentially sit on top of the Storage Management access flags. This means that Roles can be made more restrictive than the Storage Management mechanism but not less.

If a folder is marked read-only with a Role, a user will not be able to edit, save, delete or rename content in the folder regardless of the access flags or ownership.

Exago recommends choosing one permission model: either Roles or the Storage Management permissions system and not use both at the same

time.

Report Management

Content paths are determined by the access records, thus content can be in different locations for different parties.

Content names in the Storage Management system are case-sensitive.

If the parent's access flags are all disabled, then the new content will use the **Default Access Flags** from the system configuration.

Creating a New Report or Folder

Creating new content requires it be saved into a folder. Folders may be saved in the top-most level called the root or may be children of other folders. Reports may only be saved into folders where the **Can Save** flag is true for that party.

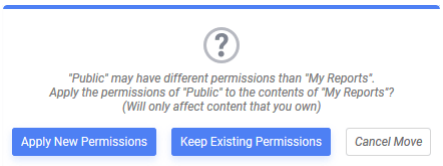
Depending on system configuration, new content can either **inherit** permissions from their parent folder or **default permissions** can be set by the system administrator on a folder-by-folder basis.

Moving a Report or Folder To a Different Folder

Moving a content item is essentially the same as creating it in the new location. Content will pick up permissions based on that container's settings. The existing access records for the content item will be removed, and new records will be written based on the new parent's settings. Review Creating a New Report or Folder above.

If someone **other than the owner** of a content item moves it from one folder to another, the item is moved only for that user. A new user-specific access record is added for the content, with the new location.

If the **owner** of the content item moves it, it is moved for all users. The owner's permissions are "sticky" for that item. The owner will retain the same level of permission they had before the item was moved. If the **owner** moves a folder to one where the permissions are different than its current location, they will be prompted how to handle the permissions after the move:



Folder Move confirmation prompt

These changes only affect content that the current user owns:

- **Apply New Permissions** â€" apply the access permissions of the destination folder to content you own.
  Example: if moving *My Reports* to *Public*, any user of the system will gain access to the content in *My Reports* you own. The permissions don't change for content other users own.
- **Keep Existing Permissions** â€" do not apply the access permissions of the destination folder, retain the permissions the content currently has.
  Example: if moving *My Reports* to *Public*, only the current user will have access to the items in *My Reports*.
- **Cancel Move** â€" do not move the folder and leave it in its current location

Deleting a Folder

Folders may only be deleted if they are empty.

If someone **other than the owner** deletes a folder, the folder is only deleted for that particular user. This is accomplished by adding a Content Access record that restricts that user's access to the item.

If the **owner** deletes a folder, it must be first empty for all users of the application. The folder will then by marked deleted for all users.

In both scenarios, the content item remains in the Storage Management database but becomes invisible to certain or all users.

Deleting a Report

If someone **other than the owner** of a report deletes it, the item is deleted only for that user. Essentially, it becomes invisible for that user while still remaining available for others.

If the **owner** of the report deletes it, it is removed for all users.

In both scenarios, the content item remains in the Storage Management database but becomes invisible to certain or all users.

Caching

The Storage Management system can cache the **Report Tree**, the **Report Definition XML** and the **Theme List**. Exago recommends that all three be enabled to reduce the load on the Storage Management database.

Security Considerations

Information about Database Permissions required for Exago to access the Storage Management database can be found in the Storage Management: Database Schema article.

Information about controlling access to content can be found in the Permissions section of this article.

Transitioning to Storage Management

Exago has developed several tools for the transition of content from legacy storage mechanisms to Storage Management. The transitioning

process and tools are documented in the Storage Management: Transitioning from Legacy Storage Methods article and in Admin Support Lab â€" Storage Management Migration.

To migrate from one Storage Management database to another, use the **ImportExportStorageMgmt** command line utility, documented in the Moving Files Between Storage Management Databases article.

Getting Started

To get up and running quickly with Storage Management, review the Storage Management: Getting Started article.

Customization

In addition to Exago's out-of-the-box implementation clients may write their own implementation of the Storage Management assembly.

Full details are outlined in the Storage Management: Customization article and in Admin Support Lab â€" Storage Management Customization.

Additional Resources

Support Lab Videos

- Admin Support Lab â€" Storage Management Overview
- Admin Support Lab â€" Storage Management Migration
- Admin Support Lab â€" Storage Management Permissioning
- Admin Support Lab â€" Storage Management Customization

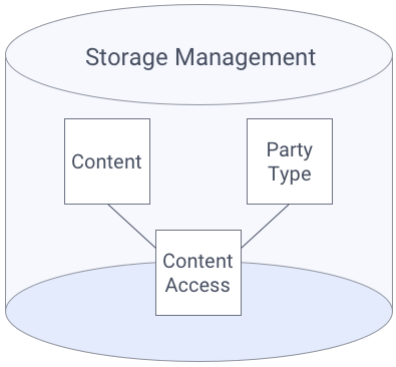
Articles

- Storage Management: Transitioning from Legacy Storage Mechanisms
- Storage Management: Database Schema
- Storage Management: Admin Console
- Storage Management: Custom Implementation
- REST â€" Storage Management
- User Identification

Storage Management: Database Schema

Â

Exago's Storage Management implementation defines several tables: **Content**, **Party Type**, **Content Access**, **Config_File** and **StorageMeta**. At a very high level, the basic relationship between these tables is depicted by the following diagram:



Storage Management table relationship

Database Requirements

Permissions

Exago must be able to **INSERT**, **UPDATE**, **SELECT** and **DELETE** on the Storage Management database. If Exago is used to initialize the database (either through the Admin Console or one of the transitioning utilities), Exago must also be able to **CREATE TABLE**.

MySQL

If using MySQL for the Storage Management database, increase MySQL's max_allowed_packet parameter to greater than the largest report or template file before using the transitioning utilities.

Symbols Used in this Article

ðŸ"‘ indicates a **primary key**
ðŸ·ï¸ indicates a **foreign key**

> **NOTE**
>
> The current Storage Management data schema version is *1.1*.

Content Table

Each report, folder, theme or template is represented by a row in the **Content** table. A GUID is used as a unique identifier throughout the database and application to reference the content.

Table A â€" Storage Management **Content** Table Schema

Column	Description
ðŸ"‘ content_id	ID which uniquely identifies this content item
content_type	The type of content this item represents, using the wrContentType enumeration (e.g. folder, report, theme, template)
report_type	The type of report this row represents, using the wrReportType enumeration. If the **content_type** is not *0* (i.e. this row does not represent a report), this column is ignored.
content_attribute	The component of the application that this item is associated with. Used when **content_type** is *2* to associate chart types with their respective themes. Will be one of the following values: • Chart • CrossTab • Express • ExpressView • ExpressView_v2 • Map â€" theme for GeoChart If the **content_type** is not *2* (i.e. this row does not represent a theme), this column is null and is ignored.
name	The name of the content item.
description	The description of the content item. For reports, this is the **Description** field provided when the report is saved.
text_content	Plain text contents of the content item. Report definitions and themes are stored in this column.
bit_content	Binary contents of the content item. Template files are stored in this column.
deleted_flag	Indicates if the content owner has deleted this item. When an item is deleted by the owner, it is no longer visible to any user. *1* represents *True* (item has been deleted), *0* represents *False* (item has not been deleted).
created_date	UTC timestamp when this content item was created.
created_by	The value of the **User Id** identity key that created this content item.
modified_date	UTC timestamp when this content was last saved.
modified_by	The value of the **User Id** identity key that last saved this content item.
owner_id	The value of the **Owner Id** identity key that owns this content. If the session's **Owner Id** identity key matches this column's value, that party will be given full permissions to the content item.
exports_allowed	A bitmap indicating which export types the content item allows, where *1* indicates an export type is permitted and *0* indicates an export type is not permitted. This column is used for informational, sorting and filtering purposes only. These flags have no affect on the export types allowed for a report, which is still controlled by the report itself. When content is saved, this column will be updated to match the export type saved in the report definition. From MSB to LSB: *Excel*, *CSV*, *RTF*, *PDF*, *HTML*. Examples: • 00001 = 1 = HTML only. No file exports. • 00011 = 3 = HTML and PDF only • 11001 = 25 = HTML (report viewer), Excel Workbook and CSV exports • 11111 = 31 = All file types allowed
inherit_flag	*True* if all of this content item's content access records should be copied to new child content when it's created. If *False*, a new content access record will be written for new child content with the **default_party_type_id** and **default_access_flags.** If *null,* the value in the system configuration will be used.
ðŸ·ï¸ default_party_type_id	If **inherit_flag** is *False*, a content access record with this party type and **default_access_flags** will be written for new content that is a child of this item. If the new content is a folder, this value will always be copied from the parent folder whether or not **inherit_flag** is true.
default_access_flags	If **inherit_flag** is *False*, a content access record with the **default_party_type_id** and these access flags will be written for new content that is a child of this item. For more information, see the access_flags column in Table E below. If the new content is a folder, this value will always be copied from the parent folder whether or not **inherit_flag** is true. These flags will be implemented in *v2020.2* of the application.
extended_attributes	This field is reserved for use by Exago clients, for storing metadata about content as they see fit in their custom implementation.
default_export_type	This column is used for informational, sorting and filtering purposes only. This flag have no affect on the default export type allowed for a report, which is still controlled by the report itself. When content is saved, this column will be updated to match the default export type saved in the report definition. The report's **Default Export Type** setting, represented by the ExportFlag enumeration.
	This column is used for informational, sorting and filtering purposes only. This flag have no affect on the report

report_tree_shortcut_action	tree shortcut for a report, which is still controlled by the report itself. When content is saved, this column will be updated to match the report tree shortcut saved in the report definition. The report's **Report Tree Shortcut** setting, using the TreeShortcut enumeration.
use_cache_execution	This column is used for informational, sorting and filtering purposes only. This flag have no affect on enabling or disabling Execution Caching for a report, which is still controlled by the report itself. When content is saved, this column will be updated to match the execution cache settings saved in the report definition. If this content item is a report (**content_type** = *0*), this column is *True* if Execution Caching is enabled for this report, or *False* if Execution Caching is disabled.
is_cache_valid	This column is for future functionality and is not implemented yet.
associated_reports	A comma separated list of **content_id**s for each report that is associated with this one. Reports become associated with others when they are components in a **Composite Report** such as **Chained Report** or **Dashboard**, or if an **Advanced Report** contains linked reports. • Chained Reports: Each component report in a chain is included. • Dashboards: Each Existing Report tile is included. • Advanced Report with linked report: a report is included for each cell that is linked. For example, if there are 4 cells that link to the same report, the same **content_id** will appear in this list 4 times.

Party Type Table

The **Party Type** defines groups of users. A party could be an individual user or all users, or anywhere in between.

Table B â€" Storage Management **Party Type** Table Schema

Column	Description
ðŸ"‘ party_type_id	Unique key identifying this party type.
priority	Defines the priority level of the access. A larger number increases priority of this party type. See Priority Levels below for additional information.
name	Name of this **party_type**.
parameter	The name of the Storage Management identity key that will be compared to see if the user has access to this content. Out-of-the-box these values are: • *classId* • *companyId* • *userId* but any identity key name defined in the configuration may be used.
description	A long format user friendly description of this party type.

Priority Levels

The party type priorities determine in which scope the associated content access record applies. They are calledÂ *priorities* because higher values override lower ones.

For example if there are two content access records that apply to the same user for a content item, the record with the numerically greater **value** will apply over the other.

For more information, review the Permissions section of the Storage Management: Introduction article.

Table C â€" Party Type Table **priority** Values

Value	Name	Description
1	Everyone	Everyone has access to this item.
2	Class	Users based on the their role in the organization. Exago will compare the value of the *classId* key against **content_access.party_id** for all records associated with the **content_id** to determine if the record applies.
3	Company	Users based on their role in the company. Exago will compare the value of the *companyId* key against **content_access.party_id** for all records associated with the **content_id** to determine if the record applies.
4	User	Individual user access. Exago will compare the value of the *userId* key against **content_access.party_id** for all records associated with the **content_id** to determine if the record applies.

Default Party Type Values

When one of the Transition Utilities is used, or the **Initialize Database** button is clicked in the **Admin Console**, Exago will create the Storage Management database tables and load the following default party types into it:

Table D â€" Default Party Type Values

party_type_id	priority	name	parameter	description
1	0	Everyone	Â	All users belong to this party
2	1	Class	classId	Users based on their role in the organization, e.g. an admin or user
3	2	Company	companyId	Users that belong to a specific company
4	3	User	userId	Individual users

Content Access Table

The relationship between **content** and **party_type** is defined in the **content_access** table. Each content access record defines permissions available for each content item. The content access table also establishes a parent-child relationship between content items.

Table E â€" Storage Management **Content Access** Table Schema

Column	Description
ðŸ·ï¸ content_id	Foreign key referencing the **content_id** in the **content** table that this record relates to.
ðŸ·ï¸ party_type_id	Foreign key referencing the **party_type_id** in the **party_type** table, establishing the party for which this access record applies.
party_id	The value of the Storage Management identity key specified by **party_type_id** that this content access record applies to. Sets the identity of the user group for which this access record applies. Example with default party types: Table F â€" Examples **If party_type_id is…** / **And party_id is…** / **These users are affected by this content access record…** 1 / n/a / All users 2 / report-builder / Any user whose *classId* key value is also report-builder. 3 / Exago, Inc. / Any user whose *companyId* key value is also *Exago, Inc.*, regardless of their *classId* key. 4 / Astro Boy / Only the user whose *userId* key value is *Astro Boy*, regardless of their *companyId* or *classId* keys.
sort_order	Content is sorted first by the value in this column, then by name (case in-sensitive). Larger values appear at the top of the sorted list). For example: **Name** / **sort_order** Nick / 99 emma / 99 Tim / 0 alex / 0 Bailey / 0 the sorted list would be: emma, Nick, alex, Bailey, Tim
access_flags	A bitmap indicating what access is permitted by this record, where *1* indicates a user in the party can complete the action and *0* indicates that they cannot. **CanView**, **CanSave** and **CanEdit** are currently implemented. The other flags are not yet active and will be implemented in a future release. All flags should be set as content is created to insure expected behavior when they are made active. From MSB to LSB: • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **CanView** â€" the content item is visible in the Report Tree • **CanDownload** â€" if **Show Report Upload/Download Options** in the Admin Console is *True*, this value determines if the content can be downloaded. Has no effect if **Show Report Upload/Download Options** is *False*, or if the content is set as read-only by a Role. • **CanCopy** â€" the content can be duplicated • **CanExecute** â€" the report can be run • **CanDelete** â€" the content can be deleted • **CanShare** â€" for future functionality and is not implemented yet • **CanRename** â€" the content can be renamed • **CanSave** â€" the report can be saved • **CanEdit** â€" the report can be opened in the Report Designer Examples: • 1111111111111111 = 65535 = All permissions available • 0000000100100000 = 288 = Report appears in the report tree and can be run, but no other permissions. • 0000000100100011 = 291 = Report may only be run, edited and saved and will appear in the report tree. • 0000000111100000 = 480 = Report appears in the report tree, and can be downloaded, duplicated and run. Otherwise no other permissions. **TIP** This is the same behavior as the **Read Only** option in Roles.
	A bitmap indicating what access is permitted by this record, where *1* indicates a user in the party can complete the action and *0* indicates that they cannot. From MSB to LSB: • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use • **Reserved** â€" this bit is reserved for future use

access_flags v2021.1+	• **CanMove** â€" the item can be moved to a different location • **CanSchedule** â€" the item can be scheduled or e-mailed • **CanView** â€" the content item is visible in the Report Tree • **Reserved** â€" this bit is reserved for future use • **CanCopy** â€" the item may be duplicated or downloaded • **Reserved** â€" this bit is reserved for future use • **CanDelete** â€" the item can be deleted • **CanShare** â€" for future functionality and is not implemented yet • **CanRename** â€" the content can be renamed • **Reserved** â€" this bit is reserved for future use • **CanEdit** â€" the item can be opened in a designer Examples: • 1111111111111111 = 65535 = All permissions available • 0000010100000001 = 1281 = item may be edited in the designer, moved to a different folder and seen in the report tree only • 0000000100000000 = 256 = item is visible in the Report Tree, but no new content may be saved and it may not open in the Designer. **NOTE** This is the same behavior as marking items as **Read Only** with a Role. The **Read Only** icon will appear for items that do not have the **Can Rename**, **Can Move**, **Can Delete** and **Can Edit** permissions.
ðŸ·ï¸ parent_id	Foreign key referencing the **content_id** in the **content** table for the parent of this item. For a report, this would be the folder the report is stored in. For a folder, this would be the parent folder. If *00000000-0000-0000-0000-000000000000*, this item is in the root of the hierarchy (it has no parent).
child_inherits	This column is for future functionality and is not implemented yet.

StorageMeta

Metadata pertaining to the Storage Management system is stored in this table as name-value pairs. It is used internally by Exago's Storage Management implementation.

Table G â€" Storage Management **StorageMeta** Table Schema

Column	Data Type	Description
ðŸ"‘ name	string	The name of the name-value pair.
value	string	The value of the name-value pair.

The **StorageMeta** table will be loaded with some initial name-value pairs when the Storage Management database is initialized. They are:

Table H â€" **StorageMeta** Key-Value Pairs

name	value Data Type	value Description
CREATED	string	A timestamp representing the instant the database was initialized (created). For example: 2020-01-23T10:35:22
SCHEMA_VERSION	string	The Storage Management data schema version currently employed by this implementation. For example: 1.1

Schema JSON

The schema defined above is also detailed in the following JSON file. This file is located in <WebApp>\Config\Other\StorageMgmtSchema.json. It is read by the transitioning utilities and the Web Application when initializing a new database.

In a custom Storage Management implementation, it is possible to add tables, columns and have Exago's tools create them when executed.

Storage Management: Getting Started

Recommended Reading

- Storage Management: Introduction
- Storage Management: Database Schema
- Storage Management: Admin Console

Microsoft SQL Server, MySQL, Oracle and PostgreSQL databases are supported out-of-the-box. Exago also supports a SQLite database for development, testing and demonstration. To use a different database type, consider writing a custom implementation of the Storage Management library.

IMPORTANT

Exago recommends using the included SQLite database for development, testing and demonstration purposes only. Use MSSQL, MySQL, Oracle or PostgreSQL in a production environment.

Installing Exago for the First Time

In order to install Exago for the first time, it is necessary to configure the report storage system. Prior to *v2020.1* there was a requirement to set the **Report Path**. Now with Storage Management, the following steps are required to get the report storage mechanism up and running:

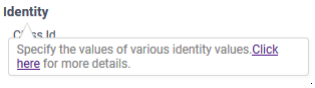
1. Point your browser to the **Administration Console**. By default this is **http://<YourServer>/Exago/Admin.aspx**. Double-click **Storage Management**.

2. Configure and create the database.
    1. Decide which database type to use: either *Microsoft SQL Server*, *MySQL*, *Oracle*, *PostgreSQL* or *SQLite. SQLite* is recommended for testing and evaluation purposes but not for production. SQLite databases can be upgraded to one of the other types later on.
    2. Make the selection in the **Database Type** dropdown.
    3. Provide the **Database Provider** and **Database Connection** string. Review the Database Section of the Storage Management Admin Console article for assistance with these settings.
    4. *Optional:* provide a prefix or schema for the database tables. Review the Storage Management: Database Schema article, or hover your mouse over **Tables** and click the 'click here' link for assistance with these settings.

    Hover over Tables to get additional help with these settings

    5. Initialize the database. This can be done automatically by Exago, or manually by a database administrator. In *v2021.2.0*, the SQLite database comes pre-initialzed, so this step may be skipped if using that database.
        - **Automatically** *(recommended)*: click the **Prepare Database** button. When prompted, click **Okay** to confirm you want to initialize the database. Exago will create the tables, schema and load default values into the database specified in steps 2â€“4 above.
        - **Manually** *(optional)*: click the **Prepare Database (SQL)** button. Exago will display SQL statements that can be copied and pasted into a database management system by a database administrator. Execute the statements before returning to Exago.
    6. Load the default themes into the database by clicking the **Load Themes** button.
3. Set the values for the **Identity Keys**.

    Hover over Identity to get additional help with these settings

    - **While in a testing or development environment where Exago may be accessed directly**, the values specified here will be used whenever content in Exago is created. This is typically sufficient for testing and demonstration purposes if you are not experimenting with the Storage Management permissions system. All users will have the same access to all of the content in this scenario. Controlling access to content can be managed with Roles instead of the Storage Management permissions system.
    - **When generating Exago sessions through one of the APIs**, the values specified here will be used if the identity keys are not set with the API at the time of session creation. It is *strongly recommended* that the identity keys be set appropriately as each session is created with one of the APIs. The User Identification article includes instructions and code samples how to do this with both the REST Web Service API and .NET API.

    Review the Identity Keys section of the Storage Management: Introduction article or hover your mouse over **Identity** and click the 'click here' link for assistance with these settings for more information about these values.

4. Set the **Default Settings** to determine how Exago will handle setting permissions information on newly created root content. Review the Default Settings section of the Storage Management Admin Console article for more information about each of these settings.
    - **To utilize new Storage Management inheritance**, set the default settings as follows. Learn more about inheritance in the Storage Management: Introduction article. You will use the permission system built in to Storage Management to manage access to content. This will require making changes to the Storage Management database, either directly with SQL or with the Storage Management Utility.
        - **Inherit Flag**: *True*
        - **Access Flags**: *Full Access (65535)*
    - **To emulate legacy file system storage**, refer to File System below.
    - **To emulate legacy Folder Management storage**, refer to Exago's Folder Management Example below.
5. Set the **Cache Settings** as desired. The default values are fine in most circumstances. Review the Cache Settings section of the Storage Management Admin Console article for more information.

Upgrading from a SQLite Database to a Production Database

Use the **ImportExportStorageMgmt** command line utility, available in *v2020.1.17+* and *v2021.1.5+,* to transition one SM database to another.

Review the Transitioning Between Storage Management Databases article for full details.

Upgrading from a pre-v2020.1 Version of Exago

When upgrading from *pre-v2020.1*, consult the Storage Management: Transitioning from Legacy Storage Methods article. This guide describes a general storage upgrade procedure and then lays out the steps for updating file system, cloud and Folder Management. Be sure to include the Themes directory when making the transition. Otherwise, no chart, map or report themes will be available after the upgrade.

Emulating a Legacy Storage Mechanism

In certain circumstances, it may be desirable to emulate the behavior of the legacy storage mechanism rather then changeover to the new permissions model provided by Storage Management. In these cases, setting the **Admin Console** > **Storage Management** > **Default Settings** as described below will result in similar behavior.

File System and Cloud Storage

In a file system, no content access permission model was available. Set the default settings as follows. You will use Roles to manage who has access to content in the system.

- **Inherit Flag**: *False*
- **Party Type**: *Everyone*
- **Access Flags**: *Full Access (65535)*

The Exago provided transitioning tools **FileSystemToJson** or **LoadReportsToDb -D** will use *system* as the Owner ID for all content loaded from a legacy file system. Therefore, we recommend setting the **Owner Id** to *system* in the **Admin Console** when emulating a file system storage system.

Exago's Folder Management Example

Using Exago's Folder Management sample code provided a content permission system that did not include **inheritance**. Therefore, set the default settings as follows. You will use the permission system built in to Storage Management to manage access to content. This will require making changes to the Storage Management database, either directly with SQL or with the Storage Management Utility.

- **Inherit Flag**: *False*
- **Party Type**: *UserId*
- **Access Flags**: *Full Access (65535)*

Storage Management: Transitioning from Legacy Storage Methods

Â

Exago's **Storage Management** system replaces the legacy *file system*, *cloud storage*, *folder management* and *web service (SOAP)* storage methods with a relational database. All reports, templates, folders and themes are stored in this database. Microsoft SQL Server, MySQL, Oracle and PostgreSQL are supported out-of-the-box. Exago also supports a SQLite database for development, testing and demonstration.

> **IMPORTANT**
>
> Exago recommends using the included SQLite database for development, testing and demonstration purposes only. Use MSSQL, MySQL, Oracle or PostgreSQL in a production environment.

Review the Storage Management: Database Schema article to gain an understanding of the structure and implementation of this database. The Admin Support Lab â€" Storage Management Migration includes a narrative walk through and demonstration of the migration process.

Table of Contents

1. General Procedure
    1. Transitioning Utilities
        1. CloudToJson
            1. Command Line Structure
            2. Command Line Parameters
            3. Common Usage Examples
            4. Manually Editing the Intermediate JSON File
            5. Access Flags
            6. Troubleshooting
        2. DatabaseToJson
            1. Command Line Structure
            2. Command Line Parameters
            3. Manually Editing the Intermediate JSON File
            4. Access Flags
            5. Troubleshooting
        3. FileSystemToJson
            1. Command Line Structure
            2. Command Line Parameters
            3. Comon Usage Examples
            4. Manually Editing the Intermediate JSON File
            5. Access Flags
            6. Troubleshooting
        4. LoadReportsToDb
            1. Command Line Structure
            2. Command Line Parameters
            3. Common Usage Examples
            4. Duplicate ID Warning
            5. Troubleshooting
2. Upgrading from File System or Cloud Storage
3. Upgrading from Folder Management
4. Upgrading from Web Service (SOAP) Storage
5. Intermediate JSON File
    1. Structure
        1. Content JSON
        2. Metadata JSON
        3. ReportAccessItem JSON

General Procedure

To upgrade from legacy file system, cloud storage or Exago's reference example of Folder Management to relational **Storage Management** the general procedure is as follows. Clients implementing custom Folder Management or SOAP Web Services should contact their Customer Success Manager for further information.

1. Identify the legacy storage method in use (either *file system*, *cloud storage*, *folder management* or *SOAP web service*).
2. Make a backup of the contents of the legacy storage method.
3. Read the contents of the legacy storage method into an intermediate JSON file with one of the Exago provided transitioning utilities. This

intermediate JSON file saves a copy of the legacy storage method's contents, and allows manual editing of it if desired before import.
4. *Not typical:* Make any changes to the intermediate JSON file to suit the destination database environment or access permissions required.
5. Initialize the destination Storage Management database.
6. Load the contents of the intermediate JSON file into the destination database with one of the Exago provided transitioning utilities.
7. Configure the Exago Web Application to connect to the Storage Management database. Information for this and the following steps can be found in the sections at the end of this article: Upgrading from File System or Cloud Storage, Upgrading from Folder Management or Upgrading from Web Service (SOAP) Storage.
8. Update any API integration code to set the identity keys each time a new session is created.

Transitioning Utilities

Exago provides several utilities for completing the above steps, described in **Table A** below.

These utilities can be found in <WebApp>bin, where <WebApp> represents the Web Application's installation directory.

Table A â€" Storage Management Transition Utilities

Step	Utility	Legacy Method(s)	Function/Purpose
2	**CloudToJson**	cloud storage	Reads an existing *cloud storage* implementation and translates it to an intermediate JSON file.
2	**DatabaseToJson**	folder management	Reads an existing database storage implementation and translates it to an intermediate JSON file. Typically used by legacy *folder management* environments implementing the Exago example.
2	**FileSystemToJson**	file system	Reads an existing *file system* implementation and translates it to an intermediate JSON file.
4, 5	**LoadReportsToDb**	all	This utility can change its function depending on command line arguments to either: • *Initialize* the destination Storage Management database with the schema. • *Load* the contents of an intermediate JSON file into the destination database, or load the contents from a file system or cloud stroage repository directly into the destination database. • *Read* the contents of the Storage Management database and display it in a format similar to the tree console command.

Regardless of legacy storage implementation, the transition utilities are non-destructive. The legacy content is only *read* during this process. It is prudent to make a backup of the legacy content store before any system upgrade.

They will run on Windows and Linux. On Windows, they should be run with Administrator privelages.

> **NOTE**
>
> When running the transition utilities on Linux, make sure the execute permission bit is set on the utility's .exe file.

For each of the utilities below, optional command line parameters are enclosed in { braces }. If a parameter has a defined set of possible values, they are enclosed in [brackets]. Default values are **bold**. The utilities process the command line parameters in the order they are entered, and may be combined into multiple actions with a single call.

- CloudToJson
- DatabaseToJson
- FileSystemToJson
- LoadReportsToDb

CloudToJson

CloudToJson is used to read one or more directories on a cloud storage repository and generate an intermediate JSON file describing the contents of that repository. If required, modifications can be made to this JSON file before loading content into the Storage Management database. JSON file will be created in <WebApp>binCloudToJson.json.

> **TIP**
>
> If customizing the JSON file is not necessary, LoadReportsToDb can read from a cloud repository directly without first having to create an intermediate JSON file.

Command Line Structure

```
CloudToJson.exe <cloudspec>
```

Command Line Parameters

Table B â€" **CloudToJson** Command Line Parameter Definitions

Parameter	Description/Usage
<cloudspec>	The string defining the connection to the cloud repository. Typically, this is Exago's **Report Path**.

Common Usage Examples

Create JSON file in default directory: Read the contents of a file system report repository. JSON file will be created in

<WebApp>binCloudToJson.json.

```
CloudToJson.exe "pathtype=s3;region='myRegion';accesskey='myAccessKey';secretkey='mySecretKey';bucketname='myBucketName'"
```

Manually Editing the Intermediate JSON File

In some circumstances, customizing the intermediate JSON file may be desirable or needed to properly set permissions on the content in the destination database. The contents of the JSON file and an explanation of each element can be found in the Intermediate JSON File section of this article.

Access Flags

In a cloud storage mechanism, all users have access to all content without restriction. Access controls are determined by use of Roles. **CloudToJson** will accordingly grant all access to all content.

By default **CloudToJson** will set:

- **AccessFlags** to *111111111111111* granting all access permissions.
- **ReportAccessList** to be empty.

Review Intermediate JSON File below and the Storage Management: Introduction article to learn about each of these access flags.

Troubleshooting

A log file named CloudToJson.log is saved to **CloudToJson**'s installation directory, typically <WebApp>bin where <WebApp> represents the Web Application's installation directory. Send this file to the Exago Support Dept when asking for assistance in troubleshooting this utility.

DatabaseToJson

DatabaseToJson is used to read the report content and access permissions information from a source database to an intermediate JSON file. It is designed for transitioning implementations of Exago's folder management example to Storage Management. It can also be used to transition from Exago's included SQLite database to a production database after testing is completed.

Access permissions from the source database will be transferred to an intermediate JSON file for importing into the Storage Management database. Orphaned reports will be imported into an *Orphaned Reports* folder.

> **NOTE**
>
> An *Orphaned Report* is one not attached to any folder (including the root); or attached to a folder that cannot be traced from the root. This typically happens if there is a missing record in the legacy folder management database's access table. Likewise, folders not traceable to the root will not be imported into the Storage Management database.
>
> Orphaned reports will be visible to all users with all access flags set to true.

Command Line Structure

DatabaseToJson needs to know how to connect to and read the data from the legacy database. This information is conveyed to **DatabaseToJson** in one of two ways:

- If no command line parameter is specified, **DatabaseToJson** will read the information from the <applicationSettings> <DatabaseToJson.Properties.Settings> nodes of the **DatabaseToJson.exe.config** file. These values must be configured before running **DatabaseToJson** the first time. Review **Table D** below for information on each setting.
- An optional command line parameter can be specified to choose a predefined configuration from a DatabaseToJson.config.json file in **DatabaseToJson**'s installation directory. The DatabaseToJson.config.json file may describe one or more database configurations to read from. It must be created before running **DatabaseToJson** the first time. The contents of the file are described in **Table D** below.

```
DatabaseToJson.exe {<configuration>}
```

Command Line Parameters

Table C â€" **DatabaseToJson** Command Line Parameter Definitions

Parameter	Description/Usage
<configuration>	The name of the configuration from the optional DatabaseToJson.config.json file to use. If no configuration parameter is provided, **DatabaseToJson** will read the connection data from DatabaseToJson.exe.config instead.

A sample of the DatabaseToJson.config.json file is below. This file defines two configurations, one named *default* and the other *sqlite*.

```
{
  "default": {
  "DbType": "SqlServer",
  "DbProvider": "System.Data.SqlClient",
  "ConnectionString": "Server=mydbserver;Database=thedatabase;uid=dblogin;pwd=dbpassword",
  "ContentSql": "SELECT id, name, CASE WHEN leaf_flag=1 THEN 0 WHEN leaf_flag=0 THEN 1 WHEN type=1 THEN 3 WHEN type=2 THEN 2 ELSE NULL END AS content_type, Rep
  "AccessSql": "SELECT ReportId AS id, ParentId AS parent_id, PartyType+1 AS party_type_id, CASE WHEN PartyType=0 THEN NULL ELSE PartyId END AS party_id, Sortorder AS sort_
"OutputFile":"DatabaseToJson.json"
 },
"sqlite": {
 "DbType": "SQLite",
 "DbProvider": "SQLite",
 "ConnectionString": "Data Source=C:\Program Files\Exago\ExagoWeb\Config\StorageManagement.sqlite",
 "ContentSql": "SELECT content_id as id, name, content_type, report_type, deleted_flag, created_date, created_by, modified_date, modified_by, owner_id, content_attribute, bit_content
 "AccessSql": "SELECT content_id AS id, parent_id, party_type_id, party_id, sort_order, access_flags & 0x0001 as edit_access, access_flags & 0x0002 as save_access, access_flags
 "OutputFile":"SQLiteToJson.json"
 }
}
```


Table D â€" **DatabaseToJson.config.json** Definition

Name	Type	Description
ConnectionString	string	Connection string to connect to the legacy storage source database.
DbType	string	The type of source database. Must be one of the following: • *Sqlite* • *Mysql* • *SqlServer* â€" Microsoft SQL Server • *Oracle* • *PostgreSQL*
DbProvider	string	Driver to use for connection to the source database **TIP** Use the applicable driver name from the **Admin Console** > **General** > **Database Settings**.
contentSQL	string	SQL statement to read content from the source database. Modify the names of the applicable columns in the source database. Review the Storage Management: Database Schema article for information.
accessSQL	string	SQL statement to read access rights from the source database. Modify the names of the applicable columns in the SQL statement to match those in the source database. Review the Storage Management: Database Schema article for information.
outputFile	string	The fully qualified file name of the intermediate JSON file that will be output. If there is no path, the file will be saved in the directory where DatabaseToJson is run from.

Access Flags

Review Intermediate JSON File below and the Storage Management: Introduction article to learn about each of these access flags.

Manually Editing the Intermediate JSON File

In some circumstances, customizing the intermediate JSON file may be desirable or needed to properly set permissions on the content in the destination database. The contents of the JSON file and an explanation of each element can be found in the Intermediate JSON File section of this article.

Troubleshooting

A log file named DatabaseToJson.log is saved to **DatabasetoJson**'s installation directory, typically <WebApp>bin where <WebApp> represents the Web Application's installation directory. Send this file to the Exago Support Dept when asking for assistance in troubleshooting this utility.

FileSystemToJson

FileSystemToJson is used to read one or more directories on a local file system and generate an intermediate JSON file describing the contents of those directories. If required, modifications can be made to this JSON file before loading content into the Storage Management database.

> **TIP**
>
> If customizing the JSON file is not necessary, LoadReportsToDb can read from a file system repository directly without first having to create an intermediate JSON file.

Command Line Structure

If the last parameter is not an existing directory it is assumed to be the path to the output file.

```
FileSystemToJson.exe <directory> {<directory> ..} {outputfile}
```

Command Line Parameters

Table E â€" **FileSystemToJson** Command Line Parameter Definitions

Parameter	Description/Usage
<directory>	At least one directory must be specified for FileSystemToJson to read. Typically, this is Exago's **Report Path** and the Themes directory. Additional directories may be specified by separating them with a space.
<outputfile>	An optional path and file name to give the JSON file the utility will generate. If not specified here, the file will be named FileSystem.json in the directory **FileSystemToJson** is in.

Common Usage Examples

Create JSON file in default directory: Read the contents of a file system report repository located at C:ExagoReports and any themes located in C:ExagoThemes. JSON file will be created in <WebApp>binFileSystem.json.

```
FileSystemToJson.exe C:ExagoReports C:ExagoThemes
```

Sample output:

```
12:00:39 INFO - Export Starts
12:00:40 INFO - from:'C:ExagoReports'
12:00:40 INFO - to:c:ExagobinFileSystem.json
12:00:40 INFO - Loading from C:ExagoReports
12:00:40 INFO - Loading from C:ExagoThemes
```

Create JSON file in specified directory: Read the contents of a file system report repository located in C:ExagoReports, any templates at C:ExagoTemplates and any themes located in C:ExagoThemes. JSON file will be created in D:MyContent.json.

```
FileSystemToJson.exe C:ExagoReports C:ExagoTemplates C:ExagoThemes D:MyContent.json
```

Sample output:

```
12:02:39 INFO - Export Starts
12:02:39 INFO - from:'C:ExagoReports'
12:02:39 INFO - to:D:MyContent.json
12:02:39 INFO - Loading from C:ExagoReports
12:02:39 INFO - Loading from C:ExagoTemplates
12:02:39 INFO - Loading from C:ExagoThemes
```

Manually Editing the Intermediate JSON File

In some circumstances, customizing the intermediate JSON file may be desirable or needed to properly set permissions on the content in the destination database. The contents of the JSON file and an explanation of each element can be found in the Intermediate JSON File section of this article.

Access Flags

In a file system storage mechanism, all users have access to all content without restriction. Access controls are determined by use of Roles. **FileSystemToJson** will accordingly grant all access to all content.

By default **FileSystemToJson** will set:

- **AccessFlags** to *111111111111111* granting all access permissions.
- **ReportAccessList** to be empty.

Review Intermediate JSON File below and the Storage Management: Introduction article to learn about each of these access flags.

Troubleshooting

A log file named FileSystemToJson.log is saved to **FileSystemToJson**'s installation directory, typically <WebApp>bin where <WebApp> represents the Web Application's installation directory. Send this file to the Exago Support Dept when asking for assistance in troubleshooting this utility.

LoadReportsToDb

LoadReportsToDb is used for three different operations: *initialize*, *load* and *read*.

1. *Initialize* sets up the Storage Management database with the schema and default party types. It should be the first operation run on a new database.
2. *Load* interprets the contents of either an intermediate JSON file, a file system directory or cloud storage repository and loads their contents into the Storage Management database.
3. *Read* verifies aÂ *Load* operation by reading the contents of the Storage Management database and displaying it in a hierarchical directory structure. Can also be called at any time to review the database's content and default party types.

Command Line Structure

NOTE

The command line parameters need to be called in the order they are documented below.

```
LoadReportsToDb.exe {-t [Sqlite|Mysql|SqlServer|Oracle|Postgresql]} -c <connection string> {-d <driver name|Sqlite>} {-p <table-prefix>} {-o <timeout_in_seconds> } {-I} {-L <jsonFile>}
```

Command Line Parameters

Table F â€" **LoadReportsToDb** Command Line Parameter Definitions

Parameter	Description/Usage
-t <**Sqlite** | Mysql | SqlServer | Oracle | Postgresql>	Storage Management database type Choose one of the following: • ***Sqlite*** (default) • *Mysql* • *SqlServer* â€" Microsoft SQL Server • *Oracle* • *PostgreSQL*
-c <connection_string>	Connection string to the Storage Management database. For SQLite, only the path and file name of the database file are required. When connecting to a PostgreSQL Storage Management database, include `Unicode=true` in the connection string. This parameter is required for all use of **LoadReportsToDb**
-d <driver_name | **Sqlite**>	Driver to use for connection to the Storage Management database Defaults to *SQLite* for SQLite (-t Sqlite). If not using SQLite, must be explicitly specified. **TIP** Use the applicable driver name from the **Admin Console** > **General** > **Database Settings**.
-p <prefix_or_schema>	An optional prefix for all Storage Management database tables. Review the Storage Management: Database Schema article for information. This option can be used to add a table name prefix or a schema if it exists. **Examples**: 1. `-p test_` will result in tables named *test_content*, *test_party_type*, *test_content_access* and *test_storage_meta* 2. `-p dbo.` will result in tables named *dbo.content*, *dbo.party_type*, *dbo.content_access* and *dbo.storage_meta*
-o <timeout_in_seconds> v2020.1.12+	An optional integer value to be passed along to the database driver's `CommandTimeout` property. It is up to the driver what to do with this value. It is important to note that each individual driver may implement the handling of this value differently. Therefore, its consequences may vary. The value must be greater than 0.
-I	Initialize operation Create the tables, load the default party types and setup the base configuration in the Storage Management database. Initialize will create two new folders in the file structure: **Public** with *Everyone* permissions and **My Reports** with *User* permissions. If **-t** is *Sqlite*, and the SQLite file does not exist at the connection string specified by **-c**, the file will be created.
-R	Read operation Read the "file structure" of the Storage Management database, and display it in a hierarchical format, similar to the operating system command `tree`. Useful for verifying the contents of the database after a JSON load (-L), directory load (-D) or web load (-W) operation.
-L <path_to_json_file>	Load operation (JSON) Read an intermediate JSON file created by **DatabaseToJson** or **FileSystemtoJson** and load its contents into the Storage Management database. Reports with duplicate IDs will be assigned new IDs and a warning will be displayed to that affect.
	Load operation (Directory) Read the report and other contents of the specified *directory* and load it into the Storage Management database. Bypasses the manual creation of the intermediate

-D <directory>	JSON file and reads the directory directly into the database. Do not include the trailing slash at the end of the directory path. **TIP** If converting from File System storage, specify the **Report Path** here to directly load reports into the database from the legacy file system. The Themes directory can also be read with the -D parameter. Each directory to be imported must be prefixed with it's own -D. Reports with duplicate IDs will be assigned new IDs and a warning will be displayed to that affect.
-W <cloudspec>	Load operation (Cloud/Web) Read the report and other contents of the cloud storage repository referenced by *cloudspec* and load it into the Storage Management Database. Bypasses the manual creation of the intermediate JSON file and reads the contents directly into the database. **TIP** If converting from Cloud storage, specify the **Report Path** here to directly load reports into the database from the cloud repository. Reports with duplicate IDs will be assigned new IDs and a warning will be displayed to that effect.

Common Usage Examples

Upgrading to Storage Management in a test environment: Initialize a SQLite database located at D:StorageMgmt.sqlite, read the contents of a file system report repository located at C:ExagoReports and any themes located in C:ExagoThemes then load it into the SQLite database.

```
LoadReportsToDb.exe -c D:StorageMgmt.sqlite -I -D C:ExagoReports -D C:ExagoThemes
```

Sample output:

```
20:31:07 INFO - LoadReportsToDb -c D:StorageMgmt.sqlite -D C:ExagoReports C:ExagoThemes
20:31:07 INFO - We are in 32bit mode
20:31:07 INFO - Loading from C:ExagoReports
20:31:16 WARN - Duplicate Id found, e8382e37-53ae-4bcc-a804-11cbf448a9be, for Report 'General ReportsDashboard', new id will be created
20:31:34 INFO - Load Completed:Folder:17 Report:159 Template:1
20:31:34 INFO - Execution Completed
```

Upgrading to Storage Management from a cloud storage repository: Initialize an Oracle database with default table names, read the contents of an Amazon S3 repository and load it into the database.

```
LoadReportsToDb.exe -t Oracle -d Oracle.DataAccess.Client -c "Data Source=(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=myOracleServer)(PORT=152
-W "pathtype=s3;region='myRegion';accesskey='myAccessKey';secretkey='mySecretKey';bucketname='myBucketName'"
```

Upgrading to Storage Management with a MySQL database and different table names: Initialize a MySQL database with table **prefix** *dukane*; read the contents of a file system report repository located at C:ExagoReports and any themes located in C:ExagoThemes then load it into the MySQL database.

```
LoadReportsToDb.exe -t Mysql -d Devart.Data.Mysql -c "Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;" -p dukane_ -I -D C:ExagoReports -D
```

Sample output:

```
11:05:29 INFO - We are in 32bit mode
11:05:29 INFO - Found Schema File at 'c:ExagoConfigOtherStorageMgmtSchema.json'
11:05:29 WARN - [ContentDatabase.DbQuery.GetFactory] Getting DbProviderFactory for 'Devart.Data.Mysql'
11:05:31 INFO - Loading from C:ExagoReports
11:05:32 WARN - [ContentDatabase.DbQuery.GetFactory] Getting DbProviderFactory for 'Devart.Data.Mysql'
11:05:43 WARN - Duplicate Id found, e8382e37-53ae-4bcc-a804-11cbf448a9be, for Report 'General ReportsDashboard', new id will be created
11:06:11 WARN - Duplicate Id found, fdfd25ea-8ea2-4205-a891-830ab8e449d5, for Report 'ReportsSales Figures', new id will be created
11:06:23 INFO - Load Completed:Folder:17 Report:159 Template:1
11:06:23 INFO - Loading from C:ExagoThemes
11:06:23 WARN - [ContentDatabase.DbQuery.GetFactory] Getting DbProviderFactory for 'Devart.Data.Mysql'
11:06:36 INFO - Load Completed:Theme:57
11:06:36 INFO - Execution Completed
```

Load an intermediate JSON file into an existing Storage Management database: Load an intermediate JSON file created by **FileSystemToJson** located at <WebApp>binFileSystem.json to a Microsoft SQL Server database that has already been initialized.

```
LoadReportsToDb -t SqlServer -d System.Data.SqlClient -c "Server=myServerAddress;Database=myDataBase;User Id=myUsername;Password=myPassword;" -L FileSystem.json
```

Verify the contents of an intermediate JSON file has been loaded correctly into a MySQL database:

```
LoadReportsToDb -t Mysql -d Devart.Data.Mysql -c "Server=myServerAddress;Database=myDataBase;Uid=myUsername;Pwd=myPassword;" -R
```

Â Sample output:

> **NOTE**
>
> Standard Reports are the legacy name for Advanced Reports. CrossTabs are also considered Advanced Reports.

```
11:48:05 INFO - LoadReportsToDb -c D:StorageMgmt.sqlite -R
11:48:05 INFO - We are in 32bit mode
11:48:05 INFO - Report Contents of Database

Advanced Reports (Inherit:False DefaultPartyType:Everyone)
|--Advanced Report (Blank) (Standard, Everyone)
|--Advanced Report (Blank) (Standard, User)
|--Advanced Report 1 (Standard, Everyone)
|--Advanced Report 1 (Standard, User)
|--Exports (Inherit:True DefaultPartyType:User)
| |--onlyExportToAllExceptPDF (Advanced, User)
| |--onlyExportToCSV (Standard, User)
| |--onlyExportToPDF (Standard, User)
| |--onlyExportToRTF (Standard, User)
| |--onlyExportToXLS (Standard, User)
Chained Reports (Inherit:False DefaultPartyType:Everyone)
|--Chained Report With Prompt (Chained, Everyone)
|--For Exports (Inherit:True DefaultPartyType:User)
| |--exportTypeAllExceptPDF (Chained, User)
| |--exportTypeCSVOnly (Chained, User)
| |--exportTypePDFOnly (Chained, User)
| |--exportTypeRTFOnly (Chained, User)
| |--exportTypeXLSOnly (Chained, User)
Chart Tests (Inherit:True DefaultPartyType:User)
|--Charting (Inherit:False DefaultPartyType:Everyone)
| |--4+ Charts (Standard, Everyone)
| |--Bar Chart (Standard, Everyone)
| |--Bubble Chart (Standard, Everyone)
| |--Revenue by Employee by Month (Standard, Everyone)
| |--YTD (Standard, Everyone)
|--Geocharts (Inherit:False DefaultPartyType:Everyone)
| |--Geocharts Report (Standard, Everyone)
| |--Geocharts Report No Map (Standard, Everyone)
|--Google Map (Inherit:False DefaultPartyType:Everyone)
| |--Google Map Report 1 (Standard, Everyone)
| |--Google Map Report with Custom Aggregates (Standard, Everyone)
Charting (Inherit:False DefaultPartyType:Everyone)
|--4+ Charts (Standard, Everyone)
|--Bar Chart (Standard, Everyone)
|--Bubble Chart (Standard, Everyone)
|--Charting Report (Standard, Everyone)
|--Column Chart (Standard, Everyone)
|--Line Chart (Standard, Everyone)
|--MinimumGauge (Standard, Everyone)
|--MultiAxis Chart (Standard, Everyone)
|--Multi-Series Pie Chart (Standard, Everyone)
|--Pseudo Crosstab Chart (Standard, Everyone)
|--Revenue by Employee by Month (Standard, Everyone)
|--YTD (Standard, Everyone)
```

Duplicate ID Warning

If during the loading operation duplicate report IDs are found, the **LoadReportsToDb** will display a warning similar to the one below:

```
20:31:16 WARN - Duplicate Id found, e8382e37-53ae-4bcc-a804-11cbf448a9be, for Report 'General ReportsDashboard', new id will be created
```

While not an indication of failure,Â duplicate IDs could cause **Scheduler Services** to lose track of the report. This would mean the next time a report is saved, the change will not be pushed to all of the Scheduler Services, or the wrong report will be pushed to them.

To remedy this:

1. Search for and delete the duplicate GUIDs from each Scheduler Service's working directory.
2. Re-schedule the report in the application.

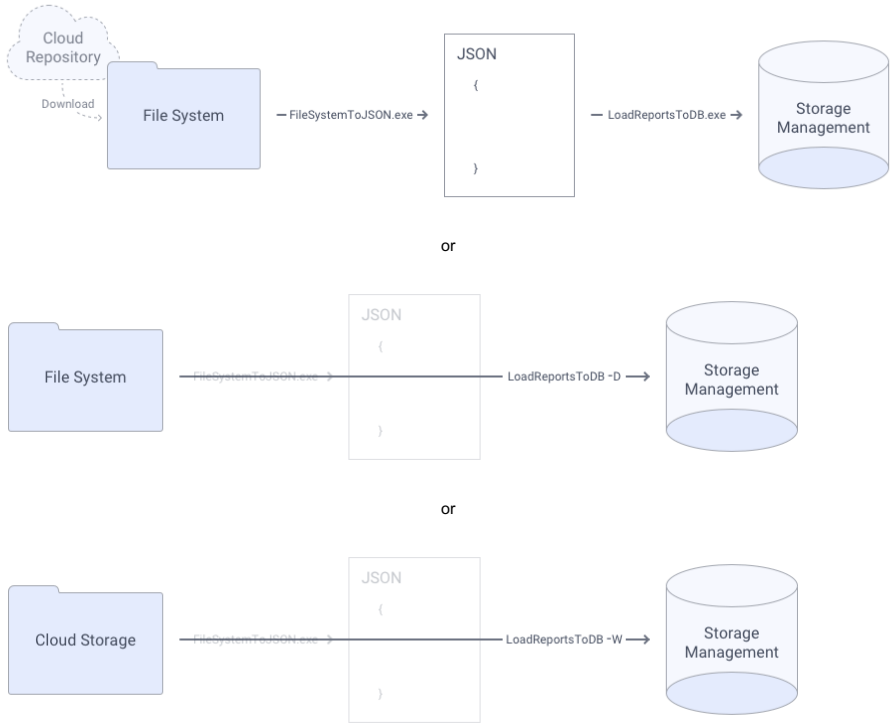
The location of the working directory can be found in the Scheduler Service's configuration file. More information can be found the Scheduler Service Configuration article.

> **NOTE**
>
> Re-scheduling is only required when warnings about duplicated IDs appear, not for every report schedule in a system.

Troubleshooting

A log file named LoadReportsToDb.log is saved to **LoadReportToDb**'s installation directory, typically <WebApp>bin where <WebApp> represents the Web Application's installation directory. Send this file to the Exago Support Dept when asking for assistance in troubleshooting this utility.

Upgrading from File System or Cloud Storage



This section will describe a generic step-by-step process to upgrade an existing legacy file system or cloud storage implementation to Exago's Storage Management system.

Access rights will be imported with global (everyone) access.

1. Identify the location of the Reports, Themes and Templates.
2. Backup the contents of the legacy storage mechanism.
3. Create the destination Storage Management database. User permissions can be found in the Storage Management: Database Schema article. If using SQLite, skip this step.
4. Run **LoadReportsToDb** with the applicable command line parameters as described above. To load report and chart themes, be sure to include the path to the themes directories.
5. Verify the load operation completed correctly by running LoadReportsToDb

   -R

   (along with any database specific parameters as applicable). The output will show the directory structure stored in the database.
6. Navigate to **Storage Management** within the **Admin Console**. Refer to the Storage Management article in the Admin Console section for an explanation of what each of these settings is and what values to provide.
   1. Set the **Database Type**, **Database Provider** and **Database Connection**.
   2. Set the **Tables** names.
   3. Click the **Check Database Settings** button to make sure Exago can connect to the Storage Management database and that the schema is correct.
   4. Set the values of the four **Identity** keys if desired. Since **FileSystemToJson** or **LoadReportsToDb -D** will use system for the owner ID during import, we suggest using system as the value for **Owner ID** in the Admin Console for file system replacements.
   5. Set the **Default Settings** for access rights if desired. To emulate the previous file system behavior, review the suggestions in the Emulating a Legacy Storage Mechanism section of the Storage Management: Getting Started article on how to set these values.
   6. Set the **Cache Settings** if desired.
7. Update any API integration code to account for the Storage Management identity keys as the application is entered.
8. Load the Exago Web Application to verify that it loads without errors and that:
   - users are seeing the correct folders and reports
   - users are *not* seeing folders and reports that they should not have access to
   - themes are available to the ExpressView Designer, Advanced Report Designer, CrossTab Designer, Charts and GeoCharts
   - template files are available to their corresponding reports

Transition to Storage Management is now complete.

Upgrading from Folder Management

This section will describe a generic step-by-step process to upgrade an existing folder management implementation using Exago's folder management example to the new Storage Management system.

Clients should contact their Customer Success Manager or open a Services Ticket for guidance about transitioning a custom folder management implementation to Storage Management.

Access rights will be imported to match the folder management implementation.

1. Backup the existing folder management database.
2. Create or edit the <WebApp>binDatabaseToJson.config.json file with the parameters to match the source database schema.
3. Run **DatabaseToJson** to generate an intermediate JSON file.
4. Create the destination Storage Management database. User access permissions can be found in the Storage Management: Database Schema article. If using SQLite, skip this step.
5. Run **LoadReportsToDb** with the applicable command line parameters as described above. To load report and chart themes, be sure to include the path to the themes directories.
6. Verify the load operation completed correctly by running LoadReportsToDb

   -R

   (along with any database specific parameters as applicable). The output will show the directory structure stored in the database.
7. Configure Exago to use the Storage Management database with the **Admin Console**. Navigate to **Storage Management** within the Admin Console. Refer to the Storage Management article in the Admin Console section for an explanation of what each of these settings is and what values to provide.
   1. Set the **Database Type**, **Database Provider** and **Database Connection**.
   2. Set the **Tables** names.
   3. Click the **Check Database Settings** button to make sure Exago can connect to the Storage Management database and that the schema is correct.
   4. Set the values of the four **Identity** keys if desired.
   5. Set the **Default Settings** for access rights if desired. To emulate the previous Folder Management behavior, review the suggestions in the Emulating a Legacy Storage Mechanism section of the Storage Management: Getting Started article on how to set these values.
   6. Set the **Cache Settings** if desired.
8. Update any API integration code to account for the Storage Management identity keys as the application is entered.
9. Load the Exago Web Application to verify that it loads without errors and that:
   - users are seeing the correct folders and reports
   - users are *not* seeing folders and reports that they should not have access to
   - themes are available to the ExpressView Designer, Advanced Report Designer, CrossTab Designer, Charts and GeoCharts
   - template files are available to their corresponding reports

Transition to Storage Management is now complete.

Upgrading from Web Service (SOAP) Storage

Clients who have implemented a SOAP Web Service for content storage are encouraged to reach out to their Customer Success Manager or to open a Services Ticket for guidance.

Piecemeal Transition

To transition a small number of reports (e.g. one at a time), use the **Download** and **Upload** capability in the Report Tree of the Web Application user interface.

1. Enable **Admin Console** > **General** > **Feature/UI Settings** > **Show Report Upload/Download Options** in both the legacy version and the *v2020.1+* version.
2. In the legacy installation, right-click the report and choose **Download** to save the report definition to the local ocmputer.
3. In the *v2020.1+* installation, right-click the desired destination folder and chooseÂ **Upload** to upload the report definition to the application.

Intermediate JSON File

TheÂ **CloudToJson**, **FileSystemToJson** and **DatabaseToJson** utilities create an intermediate JSON file defining the contents of their respective source storage mechanism. A sample JSON file generated by **FileSystemToJson** defining a single folder named “My Folder” containing a single report named “test” is shown below. A Microsoft Excel template file is also included.

Editing of this file may be necessary to customize permissions or other parameters before loading into the database. The file’s location is determined by the utility used. Review the sections above for details.

A definition of the structure of the file is detailed below in the Structure section.


```
[
 {
  "OwnerId": null,
  "CreatedBy": null,
  "ModifiedBy": null,
  "CreatedDate": "2020-02-21T11:18:36.0473673-05:00",
  "ModifiedDate": "2020-02-21T11:19:20.3949126-05:00",
  "IsDeleted": false,
  "Metadata": {
   "SortOrder": null,
   "ContentXML": null,
   "ContentData": null,
   "AccessFlags": 65535,
   "ExtendedAttributes": null,
   "FullName": "My Folder",
   "ContentId": "c242735e-34d5-40ab-901c-9849d1790998",
   "ParentId": "00000000-0000-0000-0000-000000000000",
   "IsFolder": true,
   "IsReport": false,
   "ReportType": 0,
   "ContentType": 1,
   "Name": "My Folder",
   "Attribute": null,
```

```
    "Folder": "",
    "Description": null,
    "Path": "C:\ExagoReports\Reports\My Folder",
    "AllowedOutputTypes": 0,
    "ClientData": null,
    "ExtensionData": null,
    "AllowOutputHtml": false,
    "AllowOutputPdf": false,
    "AllowOutputCsv": false,
    "AllowOutputExcel": false,
    "AllowOutputRtf": false,
    "CanRename": true,
    "CanShare": true,
    "IsReadOnly": false
   },
   "ReportAccessList": [],
   "Children": [
    {
     "OwnerId": null,
     "CreatedBy": null,
     "ModifiedBy": null,
     "CreatedDate": "2020-02-21T11:17:07.8474847-05:00",
     "ModifiedDate": "2019-10-08T14:52:07.8892432-04:00",
     "IsDeleted": false,
     "Metadata": {
      "SortOrder": null,
      "ContentXML": "==REPORT XML DEFINITION TEXT APPEARS HERE==",
      "ContentData": null,
      "AccessFlags": 65535,
      "ExtendedAttributes": null,
      "FullName": "My Folder\test",
      "ContentId": "784b1167-c0f9-42f8-8045-a2b690cf9d9a",
      "ParentId": "c242735e-34d5-40ab-901c-9849d1790998",
      "IsFolder": false,
      "IsReport": true,
      "ReportType": 0,
      "ContentType": 0,
      "Name": "test",
      "Attribute": null,
      "Folder": "My Folder",
      "Description": "",
      "Path": "C:\ExagoReports\Reports\My Folder\test.wr",
      "AllowedOutputTypes": 31,
      "ClientData": null,
      "ExtensionData": null,
      "AllowOutputHtml": true,
      "AllowOutputPdf": true,
      "AllowOutputCsv": true,
      "AllowOutputExcel": true,
      "AllowOutputRtf": true,
      "CanRename": true,
      "CanShare": true,
      "IsReadOnly": false
     },
     "ReportAccessList": [],
     "Children": [],
     "OriginalId": null
    }
   ],
   "OriginalId": null
  },
  {
   "OwnerId": null,
   "CreatedBy": null,
   "ModifiedBy": null,
   "CreatedDate": "2020-02-20T13:20:31.5527279-05:00",
   "ModifiedDate": "2020-02-20T13:20:31.5527279-05:00",
   "IsDeleted": false,
   "Metadata": {
    "SortOrder": null,
    "ContentXML": null,
    "ContentData": "==BINARY TEMPLATE FILE CONTENT APPEARS HERE==",
    "AccessFlags": 65535,
    "ExtendedAttributes": null,
    "FullName": "MyTemplate8.xlsx",
    "ContentId": "7d2f0719-c12e-41d9-9804-c4e18e7b65d6",
    "ParentId": "00000000-0000-0000-0000-000000000000",
    "IsFolder": false,
    "IsReport": false,
    "ReportType": 0,
    "ContentType": 3
```

```
  "ContentType": 3,
  "Name": "MyTemplate8.xlsx",
  "Attribute": null,
  "Folder": "",
  "Description": null,
  "Path": "C:\ExagoReports\Reports\MyTemplate8.xlsx",
  "AllowedOutputTypes": 0,
  "ClientData": null,
  "ExtensionData": null,
  "AllowOutputHtml": false,
  "AllowOutputPdf": false,
  "AllowOutputCsv": false,
  "AllowOutputExcel": false,
  "AllowOutputRtf": false,
  "CanRename": true,
  "CanShare": true,
  "IsReadOnly": false
 },
 "ReportAccessList": [],
 "Children": [],
 "OriginalId": null
 }
]
```


Structure

The intermediate JSON file contains an array of **Content** objects.

Content JSON

Table G â€" Content JSON Structure

Name	Type	Description
OwnerID	string	The user ID of the user who is the owner of this content item. The value of this column is read from a folder management database. For file system and cloud imports, this value will be null.
CreatedBy	string	The user ID of the user who created this content item. The value of this column is read from a folder management database. For file system and cloud imports, this value will be null.
ModifiedBy	string	The user ID of the user who last saved modified this content item. The value of this column is read from a folder management database. For file system and cloud imports, this value will be null.
CreatedDate	string	Timestamp indicating the date, time and time zone offset when **CreatedBy** created this content item.
ModifiedDate	string	Timestamp indicating the date, time and time zone offset when **ModifiedBy** modified this content item.
IsDeleted	boolean	Indicates whether or not this item is to be marked as deleted for all users. For file system and cloud imports, this value will be always be false.
MetaData	MetaData	A JSON object that represents the metadata for this content item. See **Table H** below.
ReportAccessList	array of ReportAccessItem objects	An array of **ReportAccessItem** JSON objects which set the accessibility of this content item. See **Table I** below.
Children	array of Content objects	If this content item is a container for other items (i.e. it is a folder), this is an array of the **Content** JSON items that are contained in the folder.
OriginalId	String	The ID of this content item as referred to by the legacy storage system. For file system and cloud imports, this value will be null.

MetaData JSON

The contents of this JSON object closely mimic the fields in the Storage Management database's Content and Content Access tables.

Table H â€" MetaData JSON Structure

Name	Type	Description
SortOrder	string	This field is for future use and is not yet implemented. Value will always be null.
ContentXML	string	If this content item is a report or a theme, then its XML content will be stored as text in this column.
ContentData	string	If this content item is a template file or other binary object, an ASCII representation of the file contents will be stored in this field.
AccessFlags	integer	If **ReportAccessList** is empty, a bitmap indicating which access permissions are allowed for the content item, where *1* indicates the action is permitted and *0* indicates the action is not permitted. If **ReportAccessList** is not empty, then the AccessFlags of each **ReportAccessItem** object will determine permissions. See Storage Management: Database Schema, content_access Table, access_flags Column for more information and examples.
ExtendedAttributes	string	This field is reserved for use by Exago clients, for storing metadata about content as they see fit in their custom implementation. Value will always be null.

FullName	string	The name of the content item, including it's path from the root. For example, for a report named *test* in a folder named *My Folder*, this field would be "*My Folder\test*"
ContentId	string	A GUID representing this content item in the destination Storage Management database.
ParentId	string	If **ReportAccessList** is empty, a GUID representing the ContentId field of this item's parent. For example, for a report this would be the GUID of the folder containing it. If *00000000-0000-0000-0000-000000000000*, this item is in the root. If **ReportAccessList** is not empty and this content item is a report, this field is ignored.
IsFolder	boolean	Indicates if this content item represents a folder. *True* if it is a folder, *False* if it is not.
IsReport	boolean	Indicates if this content item is a report. *True* if it is a report, *False* if it is not.
ReportType	wrReportType	If **IsReport** is *True*, this value represents the type of report this item is.
ContentType	integer	The type of content this item represents. See Storage Management: Database Schema, content Table, content_type Column for more information and examples.
Name	string	The name of this content item. For example, for a report named *test* in a contained folder named *My Folder*, this column would be "*test*"
Attribute	string	If this content item represents a chart or report theme, this column describes the type of chart or report that it applies to. For example, *Map* or *CrossTab*.
Folder	string	The name of the folder this content item is contained in. For example, for a report named *test* in a contained folder named *My Folder*, this column would be "*My Folder*"
Description	string	The description text assigned to a report when it is saved.
Path	string	The absolute file system location pointing to this content item prior to import. For example, for a report named *test* contained in a folder named *My Folder*, this column would be "*C:\ExagoReports\My Folder\test.wr*"
AllowedOutputTypes	integer	A bitmap indicating which export types are allowed for the content item, where *1* indicates an export type is permitted and *0* indicates an export type is not permitted. See Storage Management: Database Schema, content Table, exports_allowed Column for more information.
ClientData	String	Content of the UDF metadata string from a custom folder management implementation.
ExtensionData	String	This field is for future functionality and is not implemented yet.
AllowOutputHtml	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AllowedOutputTypes**.
AllowOutpoutPdf	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AllowedOutputTypes**.
AllowOutputCsv	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AllowedOutputTypes**.
AllowOutputExcel	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AllowedOutputTypes**.
AllowOutputRtf	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AllowedOutputTypes**.
CanRename	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AccessFlags**.
CanShare	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AccessFlags**.
IsReadOnly	boolean	Deprecated. This column will not be read during the import process. Instead, set access permissions with the **AccessFlags**.

ReportAccessItem JSON

LoadReportsToDb will generate Content Access table entries based on these ReportAccessItems.

Table I â€" ReportAccessItem JSON Structure

Name	Type	Description
Parent	string	Id of the content item that this item appears as a child to.
PartyTypeId	integer	Refer to Storage Management: Database Schema Table D.
PartyId	string	The value of the corresponding identity key that this **ReportAccessItem** controls. For example, if **PartyTypeId** is *4*, this is the value of UserId for which the **AccessFlags** will apply.
SortOrder	string	Content is sorted first by the value in this column, then by name (case in-sensitive).
AccessFlags	integer	A bitmap indicating which access permissions are allowed for the content item, where *1* indicates the action is permitted and *0* indicates the action is not permitted. See Storage Management: Database Schema, content_access Table,

		access_flags Column for more information and examples.

Moving Files Between Storage Management Databases

Exago recommends using a three-environment development, quality assurance and production paradigm. In this structure, it is often desirable to maintain unique Storage Management databases for each environment. Use the **ImportExportStorageMgmt** command line utility, available in *v2020.1.17+* and *v2021.1.5+*, to transition one SM database to another.

Unlike the DatabaseToJson legacy transitioning utility, **ImportExportStorageMgmt** creates a JSON file **for each** content item in the database named with its content_id. These files can then be easily backed up or version controlled.

As its name implies, **ImportExportStorageMgmt** supports two different operations:

> **IMPORTANT**
>
> **ImportExportStorageMgmt** requires that the primary keys from Exago's factory implementation of Storage Management are in the database. If the primary keys have been changed, an alternate method of transitioning must be used.

- **Export** â€" read the contents of an existing Storage Management database and output a JSON file for each content item. Filters can be applied to include or exclude certain folders as desired. All columns, including custom columns will be exported.
- **Import** â€" read the contents of a directory of JSON files (created during an Export) and load each one into a target database. If an item already exists in the target, update it. Imports are completed in two steps: first for content table records, then for content access table records.

> **NOTE**
>
> The target database must be initialized before **ImportExportStorageMgmt** can import into it. Use the Admin Console's **Prepare Database** button; execute the statements returned with the **Show Prepare Data** button; or call the **LoadReportsToDb** utility with the -I parameter to accomplish this.

Configuration File

Before handling any data, **ImportExportStorageMgmt** must know how to access the source and target databases, where to store the JSON files and any filtering options desired. These settings are contained in a JSON **configuration** file.

There may be more than one configuration file created ahead of time, as ImportExportStorageMgmt allows specifying a configuration file at runtime.

File Structure

A sample of a configuration file is below. Tables A and B below the sample define the different components.

```
{
  "SourceDb":
  {
    "DbType":"SQLite",
    "DbProvider":"SQLite",
    "ConnectionString":"DataSource=D:\ImportExport.sqlite",
    "TablePrefix":"dev_"
  },
   "TargetDb":
  {
    "DbType":"MySQL",
    "DbProvider":"devart.data.mysql",
    "ConnectionString":"Unicode=true; User ID=xxxxxx;Password=xxx;Host=xxx;Database=filesystem",
    "TablePrefix":"prod_"
  },
  "JsonDirectory":"c:\exago\export\Json",
  "BlockList" : [
    "Public",
    "Hidden"
  ],
  "AllowList" : null,
  "AutoMatch" : [
    "Public",
    "My Reports"
  ],
  "FieldSpecification": {
 "created_date" : "datetime",
 "modified_date" : "datetime",
 "bit_content" : "bytearray"
 }
}
```

Table A â€" **ImportExportStorageMgmt** Configuration File JSON Definition

Name	Type	Description
SourceDb	Database Definition	Source database from which to copy content during an Export operation.
TargetDb	Database Definition	Destination database where content is copied to during an import operation.

JSONDirectory	string	File system directory where the JSON files are exported (saved) to, and imported (read) from.
BlockList	array of string	List of root folders (those at the top level of the Report Tree) to exclude from an export operation. During export, content access records that refer to folders on the BlockList will not be written if that content only exists in a blocked folder. To block no folders, set to null. **TIP** To import/export the entire Report Tree, leave both **BlockList** and **AllowList** as null.
AllowList	array of string	List of folders to include in the export operation. During export, content access records will be written only if content exists in one of the AllowList folders and the AllowList is not empty. To allow all folders, set to null. **TIP** To import/export the entire Report Tree, leave both **BlockList** and **AllowList** as null.
AutoMatch	array of string	To match a root folder by *name* instead of *content ID*, provide the name of that folder in this list. The first folder in the target database with the same name in the source database will be assumed to be the same folder.
FieldSpecification	JSON object with key/value pairs	Use to identify the names of *datetime* or *bytearray* columns in the database. Use the column name as the key and the data type, either *datetime* or *bytearray* as the value. Special handling for fields is performed when a fieldâ€™s name matches one specified in the list.

Table B â€" **Database Definition** JSON Definition

Name	Type	Description
ConnectionString	string	Connection string to connect to the database
DbType	string	The type of source database. Must be one of the following: • *Sqlite* • *Mysql* • *SqlServer* â€" Microsoft SQL Server • *Oracle* • *PostgreSQL*
DbProvider	string	Driver to use for connection to the source database **TIP** Use the applicable driver name from the **Admin Console** > **General** > **Database Settings**.
TablePrefix	string	Table prefix in the database, if applicable. If no prefix is needed, set to null.

Command Line

NOTE

This article references <WebApp>/ as a placeholder for the Web Application's install location. The default install location is C:Program FilesExagoExagoWeb (/opt/Exago/ on Linux), however, these directories can be changed during installation.

By default **ImportExportStorageMgmt** is located in <WebApp>/bin. Configuration files will be saved to the current directory.

Structure

Optional command line parameters are enclosed in { braces }. The utility processes the command line parameters in the order they are entered, and may be combined into multiple actions with a single call. When combing actions, they must appear in the order presented below:

```
ImportExportStorageMgmt {-h} -f <configFile> {-E} {-I}
```

Parameters

Table C â€" **ImportExportStorageMgmt** Command Line Parameter Definitions

Parameter	Description/Usage
-f <configFile>	The name and path of the configuration file to use. If the file is not found, the utility will create a default configuration file at this location.

-E	Perform an export operation from the <configFile>'s SourceDb
-I	Perform an import operation to the <configFile>'s TargetDb
-h	Display help message with the command line structure

Common Usage Examples

Create a new configuration file: So long as the file name following the -f does not exist, the utility will create a sample configuration file with that name. This example will create a new configuration file named Dev-to-Prod.json in **ImportExportStorageMgmt**'s installation directory.

```
ImportExportStorageMgmt.exe -f Dev-to-Prod.json
```

Sample Output

```
11:12:04 INFO - ExportStorageMgmt -f Dev-to-Prod.json
11:12:04 ERROR - Specified configuration file not found - I'll try and create one for you
11:12:04 INFO - Created Config File at Dev-to-Prod.json
```

Transfer a single database from one to another: The configuration defines a single source and a single target database.

```
ImportExportStorageMgmt.exe -f Dev-to-Prod.json -E -I
```

Export multiple databases, import to a single database: Define the separate source databases in different configuration files. They must reference the same **JSONDirectory**. Define the target database in the last configuration file. Perform an export operation on all databases, but only a single import on the last one. Since the import operation reads all of the contents of the **JSONDirectory** the contents of all previously exported databases will be imported.

```
ImportExportStorageMgmt.exe -f Dev-to-Prod.json -E -f QA-to-Prod.json -E -I
```

Sample Output

```
14:27:41 INFO - ExportStorageMgmt -f Dev-to-Prod.json -E -f QA-to-Prod.json -E -I
14:27:41 INFO - Starting Export
14:27:43 INFO - Processing Complete, write 1534 records
14:27:43 INFO - Exported 1136 records of type 0
14:27:43 INFO - Exported  251 records of type 1
14:27:43 INFO - Exported  132 records of type 2
14:27:43 INFO - Exported   15 records of type 3
14:27:43 INFO - Starting Export
14:27:45 INFO - Processing Complete, write 860 records
14:27:45 INFO - Exported  616 records of type 0
14:27:45 INFO - Exported  124 records of type 1
14:27:45 INFO - Exported  100 records of type 2
14:27:45 INFO - Exported   20 records of type 3
14:27:45 INFO - Starting Import
14:29:03 INFO - AutoMatch for Public 7da64126-c96c-471d-a3c1-5f6060b2a812=>afe754e9-4439-459a-90ab-b6981ebf3825
14:30:10 INFO - AutoMatch for Public e62df5d8-8228-4839-a87d-2000179f9ff0=>afe754e9-4439-459a-90ab-b6981ebf3825
14:30:18 INFO - AutoMatch for My Reports f3d0dad0-9a5d-49d5-99b3-6c2de8285237=>843e95d0-74c0-414b-b4eb-5a1dfdd96bb7
14:30:19 INFO - AutoMatch for My Reports f6aea47e-b24b-46ca-83ac-98a443c6c3c9=>843e95d0-74c0-414b-b4eb-5a1dfdd96bb7
14:30:23 INFO - AutoMatch for My Reports fcf2b618-034c-417d-9688-6326b38d9632=>843e95d0-74c0-414b-b4eb-5a1dfdd96bb7
14:30:25 INFO - Completed Loading Content Records: Inserted 2389 content records, and updated 0 content records
14:33:03 INFO - Completed Loading Access Records: Inserted 2708 access records, and updated 0 access records
14:33:03 INFO - No Orphan Records Located
```

Troubleshooting

The ImportExportStorageMgmt.log log file is saved to **ImportExportStorageMgmt**'s installation directory. Send this file to the Exago Support Team when asking for assistance in troubleshooting this utility. The log file contains additional DEBUG lines with SQL statements not included in the console.

A sample log is below, without the DEBUG lines:

```
08:42:27 INFO - ExportStorageMgmt -f MySMConfigFile.json -E -I
08:42:27 INFO - Starting Export
08:42:29 INFO - Processing Complete, write 851 records
08:42:29 INFO - Exported  608 records of type 0
08:42:29 INFO - Exported  123 records of type 1
08:42:29 INFO - Exported  100 records of type 2
08:42:29 INFO - Exported   20 records of type 3
08:42:29 INFO - Starting Import
08:43:27 INFO - AutoMatch for My Reports f3d0dad0-9a5d-49d5-99b3-6c2de8285237=>f4fdd6d2-1c32-4bf9-bfbb-f13a1ecdc10b
08:43:30 INFO - Completed Loading Content Records: Inserted 850 content records, and updated 0 content records
08:44:21 INFO - Completed Loading Access Records: Inserted 859 access records, and updated 0 access records
08:44:21 INFO - No Orphan Records Located
```

Log File Contents

The **type** in each Exported ___ records of type line represents one of the Content Types from the wrContentType enumeration.

AutoMatch lines appear when a folder name in the AutoMatch list is matched, with the content ID of the folder in the source database and then the content ID of the folder in the target database.

Completed lines appear when each step of the import operation finishes. Inserts for new items are counted along with items that are updated.

REMINDER

Deleting content from the system doesn't remove records from the database. Instead, either an update to the content or content access records will be made depending on who deleted the item. See Storage Management: Introduction for more information.

JSON Files

The individual JSON files in the **JSONDirectory** represent a single record in the database's content table. Each file is name with the record's **content_id**.

The file consists of two JSON components: one named **Content** and the other **Access** that correlate with the item's content record and content_access records. Refer to Storage Management: Database Schema for information about each field.

NOTE

The **Access** object has two additional properties not in the database: *parentname* and *contentname*. These are created by and for the use of the utility only.

Sample Content Item JSON File

```
{
  "Content": {
    "content_id": "bcbac226-ded2-46ed-bc8b-62648c5ad0f1",
    "content_type": 0,
    "report_type": 2,
    "content_attribute": null,
    "name": "Executive Dashboard",
    "description": null,
    "text_content": "(the contents of a report WR file appears here)",
    "bit_content": null,
    "deleted_flag": false,
    "created_date": "2016-10-17T11:24:15",
    "created_by": "98871680-c7bc-421a-846a-cdcefaada47b",
    "modified_date": "2017-08-23T12:06:44",
    "modified_by": "98871680-c7bc-421a-846a-cdcefaada47b",
    "owner_id": "98871680-c7bc-421a-846a-cdcefaada47b",
    "exports_allowed": 0,
    "inherit_flag": false,
    "default_party_type_id": null,
    "extended_attributes": null,
    "default_access_flags": null,
    "default_export_type": null,
    "report_tree_shortcut_action": null,
    "use_cache_execution": null,
    "is_cache_valid": null,
    "associated_reports": null
  },
  "Access": [
    {
      "content_id": "bcbac226-ded2-46ed-bc8b-62648c5ad0f1",
      "parent_id": "b6d3f26a-0a58-4c7b-8616-9f670d43f9f7",
      "party_type_id": 1,
      "party_id": null,
      "sort_order": 0,
      "access_flags": 480,
      "child_inherits": false,
      "parentname": "Dashboards",
      "contentname": "Executive Dashboard"
    },
    {
      "content_id": "bcbac226-ded2-46ed-bc8b-62648c5ad0f1",
      "parent_id": "b6d3f26a-0a58-4c7b-8616-9f670d43f9f7",
      "party_type_id": 2,
      "party_id": "1",
      "sort_order": 0,
      "access_flags": 503,
      "child_inherits": false,
      "parentname": "Dashboards",
      "contentname": "Executive Dashboard"
    },
    {
      "content_id": "bcbac226-ded2-46ed-bc8b-62648c5ad0f1",
      "parent_id": "b6d3f26a-0a58-4c7b-8616-9f670d43f9f7",
      "party_type_id": 4,
      "party_id": "98871680-c7bc-421a-846a-cdcefaada47b",
      "sort_order": 0,
      "access_flags": 511,
      "child_inherits": false,
      "parentname": "Dashboards",
      "contentname": "Executive Dashboard"
    }
  ]
}
```


Storage Management: Custom Implementation

Recommended Reading

The following articles provide background on Exago's Storage Management system. It is highly recommended to review them before writing a custom implementation of Storage Management.

- Storage Management: Introduction
- Storage Management: Database Schema

Introduction

Clients may write their own implementation of the Storage Management assembly to replace Exago's out-of-the-box implementation.

In *v2021.1+*, the **IStorageManagement** interface methods have been virtualized, so clients need only implement the methods that will operate differently than Exago's out-of-the-box behavior. Refer to the basic example at the end of this article.

Clients looking to write their own Storage Management assembly may download Exago's Storage Management implementation code as a Visual Studio project from the Downloads page on Support Site. Clients are also encouraged to open a Services Ticket in our Support Center for assistance if desired.

SessionInfo

The **SessionInfo object** is not available in the Storage Management system. However, there is a dictionary of options passed around the Storage Management system that stores pertinent information for the database, but additional key-value pairs can be added as needed. This dictionary may be accessed with the .NET API, REST Web Service API and within the Storage Management assembly itself.

Example

This example modifies the constructor of Exago's Storage Management example.

```
public StorageMgmtDatabase(StorageMgmtConfig config) {

 //set the value of the companyId identity key to Exago, Inc.
 config.SetIdentity("companyId", "Exago, Inc.");

 //set a Custom Option named userProfile
 config.SetOption("userProfile", "support_Administrator");
}
...
```

Implementation

> **NOTE**
>
> This article references <WebApp>/, <WebSvc>/ and <Sched>/as a placeholder for the Web Application, Web Service API and Scheduler Service's install location respectively. The default install location is C:Program FilesExagoExagoWeb (/opt/Exago/ on Linux), C:Program FilesExagoExagoWebApi (/opt/Exago/WebServiceApi/ on Linux) or C:Program FilesExagoExagoScheduler (/opt/Exago/Scheduler/ on Linux); however, these directories can be changed during installation.

Once developed, move the finished library into a directory accessible to the Exago Web Application e.g. <WebApp>/bin, Scheduler Services e.g. <Sched>/bin (if applicable) and REST Web Service API e.g. <WebSvc>/bin (if applicable). Then set the **Admin Console** > **Storage Management** > **Assembly Location** and **Class Name** settings accordingly.

> **TIP**
>
> Name the DLL file something other than WebReports.ContentDatabase.dll as not to overwrite the Exago provided library. This way, if troubleshooting is ever needed in the future, it is easy to change back to the factory provided implementation.

ContentMetadata Class

The **ContentMetadata** class contains metadata about reports and other content. This data is saved to the Storage Management database, and is serializable as JSON.

Namespace: WebReports.Api.Common

Properties

A **ContentMetaData** object has the following properties. Properties that appear as hyperlinks below require constants or enums which are documented in the Constants and Enumerators article.

Table A â€" ContentMetaData Class Properties

Name	Type	Writable	Description
AccessFlags	ContentPermission	yes	Integer representation of bitmap of the access flags bitmap
Age	TimeSpan	no	The age of the object, used internally
AllowExportCsv	boolean	yes	Whether this report object can be exported to CSV.
AllowExportExcel	boolean	yes	Whether this report object can be exported as an Excel workbook.
AllowExportHtml	boolean	yes	Whether this report object can be run in a Report Viewer
AllowExportPdf	boolean	yes	Whether this report object can be exported as a PDF file.
AllowExportRtf	boolean	yes	Whether this report object can be exported as an RTF file.
Attribute	string	yes	If this content is a theme, then this is theme type (i.e. Chart, CrossTab, Express, ExpressView, ExpressView_v2, Map)
CacheKey	string	no	Returns the content ID of this object.
CanRename	boolean	yes	This field is not implemented and is ignored.
CanShare	boolean	yes	This field is not implemented and is ignored.
ClientData	string	yes	Content of the UDF metadata string from a custom folder management implementation.
ContentData	array of byte	yes	Holds the binary content of templates

ContentId	GUID	yes	The GUID which uniquely identifies this content item throughout the application
ContentType	wrContentType	yes	The type of content that this content item is
ContentXML	string	yes	Holds the text string for reports and themes
Description	string	yes	User supplied report description field
ExportFlags	integer	yes	Bitmap indicating the various permitted export types. From MSB to LSB: Excel, CSV, RTF, PDF, HTML. This property is for informational purposes only. Changing it will not affect the report's output types. Output types are only controlled by the report.
ExtendedAttributes	string	yes	This field is reserved for use by Exago clients, for storing metadata about content as they see fit in their custom implementation.
ExtensionData	string	yes	This field is for future functionality and is not implemented yet.
Folder	string	yes	Relative folder path of this content item
FullName	string	yes	Folder path plus the report name **Example:** My Reports\Sales Report
IsFolder	boolean	yes	Indicates if this content item is a folder as opposed to a report
IsOwner	boolean	yes	Indicates if the current user is the owner of the content item
IsReadOnly	boolean	yes	Determines if this item is considered to be read-only.
IsReport	boolean	yes	Indicates if this content item is a report. *True* if it is a report, *False* is it is not a report.
Name	string	yes	Report name without the folder or file extension
ParentId	GUID	yes	The GUID that uniquely identifies the parent (folder) of this content item
ReportType	wrReportType	yes	If the **ContentType** property is *wrContentType.Report*, this property indicates the report type it is
Timestamp	DateTime	yes	A DateTime representing the moment when this object was created.
DefaultExportType	ExportFlag	yes	The report's **Default Export Type** setting, represented by the ExportFlag enumeration.
ReportTreeShortcutAction	TreeShortcut	yes	The report's **Report Tree Shortcut** setting, using the TreeShortcut enumeration.
UseCacheExecution	boolean	yes	If this content item is a report, this column is *True* if Execution Caching is enabled for this report, or *False* if Execution Caching is disabled.
IsCacheValid	boolean	yes	This column is for future functionality and is not implemented yet.
AssociatedReports	List of GUID	yes	A list of **content_id**s as GUIDs for each report that is associated with this one. Reports become associated with others when they are components in a **Composite Report** such as **Chained Report** or **Dashboard**, or if an **Advanced Report** contains linked reports. • Chained Reports: Each component report in a chain is included. • Dashboards: Each Existing Report tile is included. • Advanced Report with linked report: a report is included for each cell that is linked. For example, if there are 4 cells that link to the same report, the same **content_id** will appear in this list 4 times.
SourceId	GUID	yes	The content ID that this **ContentMetadata** object is based on. For newly created content or when saving a report with the same name this field will be Guid.Empty. For duplicated reports or when saving an existing report with a new name, this field will be the content ID of the original report. When exporting an ExpressView to an Advanced Report, this field will be the content ID of the ExpressView. Based on the value of this field, it can be determined if an item is being saved for the first time, being duplicated in the user interface, or saved with either a new or existing location.

Constructors

public ContentMetadata()

Description	Instantiate a new **ContentMetadata** object. **ContentMetadata.AccessFlags** will be set to full permissions.

public ContentMetadata(ContentMetadata other)

	Copy constructor: Instantiate a new **ContentMetadata** object and copy the properties of the passed object other to this one. The following properties are copied: • ContentId • ParentId • ReportType

Description	• ContentType • Name • Folder • Description • IsReadOnly • ExportFlags • TimeStamp • ContentXML • ContentData • AccessFlags • ClientData • ExtendedAttributes • SortOrder • IsOwner • ReportTreeShortcutAction • DefaultExportType
Arguments	• other: the **ContentMetadata** object to copy into this new one

Methods

string GetUdfJson()

Description	Serializes this content object as JSON. Includes metadata about the item. Properties included in the JSON: • ContentId • ContentType • ReportType • AccessFlags • ClientData
Returns	A JSON encoded **String** of this object for use in the client

static ContentMetadata GetMetadataFromJson(string jsonStr)

Description	Return a **ContentMetadata** object from the supplied string, assumed to be a JSON serialized form of a **ContentMetadata** object. This method is static.
Arguments	• jsonStr: a JSON serialized form of a **ContentMetadata** object
Returns	A **ContentMetadata** object

void SetFullNamesFromFullName(string fullName)

Description	Set the report name (Name property) and folder (Folder property) from the specified FullName property string.
Arguments	• fullName: a string representing the **FullName** property of a content item.

string ToString()

Description	Return a string representation of the **ContentMetadata** object

Returns	A string in the following format:
	`ReportMetadata Id: {contentId} Name: {name} Folder: {folder} Type: {report or content type}`
	Items in {braces} will be replaced by the object's properties. Examples:
	`ReportMetadata Id: 34952c9c-9b0c-869a-c850-3b623ba4005b Name: Current Product Listing Folder: General Reports Type: ExpressView`
	`ReportMetadata Id: 141f017a-e8c2-4a9a-baec-90826bab4d77 Name: SA Timesheet (blank).pdf Folder: NULL Type: Template`

IStorageManagement Interface

The IStorageManagement interface defines all of the necessary methods that will be called from Exago BI to read and write content to the Storage Mechanism. All of the permissions logic, if applicable, can be built into the methods of this class.

WebReports.Api.Report.ReportProperties objects can be used for reading information about reports directly from their XML definition, without having to first load the report with ReportObjectFactory.LoadFromRepository(string name).

Namespace: WebReports.Api.ReportMgmt

```
public interface IStorageManagement
{
 wrContentType? GetContentType(Guid contentId)

 IEnumerable<ContentMetadata> GetContentList(wrContentType contentType, bool includeContent = false);

 ContentMetadata GetContent(ContentMetadata metadata);

 string GetCacheKey();

 void RenameContent(ContentMetadata metadata, string newName, ContentMetadata parentMetadata);

 void DeleteContent(ContentMetadata metadata);

 void SaveContent(ContentMetadata metadata);

 StorageMgmtResult ValidateContentName(string reportName, string udfData);

 void AddFolder(ContentMetadata metadata);

 void MoveFolder(ContentMetadata folder, ContentMetadata target, bool updatePermissions);

 IEnumerable<string> GetTemplateList();

 byte[] GetTemplate(string templateName);

 void SaveTemplate(string templateName, byte[] templateData)

 IEnumerable<ContentMetadata> GetThemeList(string themeType);

 bool ExistTheme(string themeType, string themeName);

 string GetTheme(string themeType, string themeName);

 void SaveTheme(string themeType, string themeName, string themeData);

 void ClearReportListCache();

 string TestConnection();

 string InitDb(string jsonSchemaFilePath, bool showSQL);
}
```

Methods

wrContentType? GetContentType(Guid contentId);

Description	Return the content type (e.g. *report*, *folder*, *theme*, *template*) of the object whose ID is **contentId** **NOTE** This call ignores the user's access permissions.
Arguments	• contentId: a GUID, the content ID in the database of the content whose type to retrieve
Returns	a value from the wrContentType enumeration representing the type of content (e.g. folder, report, theme, template) to return

IEnumerable<ContentMetadata> GetContentList(wrContentType contentType, bool includeContent)

Description	Return a collection of content items of the specified type as ContentMetadata objects. This method is called whenever a list of content is needed, such as by the Report Tree, in the Linked Report Wizard, the Name tab of the Report Wizard, the Duplicate Report dialog, the Role Folders dialog in the Admin Console.
Arguments	• contentType: a value from the wrContentType enumeration representing the type of content (e.g. folder, report, theme, template) to return • includeContent: boolean value indicating whether or not the content item's full content (either the text or binary content) will be returned in the ContentMetadata object. Set to *true* to return content, *false* to return metadata only. Default value is *false*.
Returns	A collection of ContentMetadata objects

ContentMetadata GetContent(ContentMetadata metadata)

	Return the metadata and content (text or binary) for the specified content item.

Description	This method is called whenever content needs to be retrieved from the Storage Management database. This includes opening a report to edit or execute, reading in template files or themes during report design or execution. This method is also called after clicking **Finish** in the Report Wizard to load the newly created report into the Designer, and when clicking on items in the Report Tree to load details about the item (e.g. export types).
Arguments	• metadata: a **ContentMetadata** object representing the content item to retrieve.
Returns	A **ContentMetadata** object that represents the content requested

string GetCacheKey()

Description	Return the report list cache key. This is generally user specific and is constructed from containing all the identity keys referenced in the party type table. Exago will compare the returned cache keys. If the cache key is different, the cache should be considered expired. This method is called frequently to check if the cache has expired for a particular user, including the start of the session, starting the Report Designers or Report Wizards, and stepping through the Report Wizard.
Returns	A string with the cache key.

void RenameContent(ContentMetadata metadata, string newName, ContentMetadata parentMetadata)

Description	Renames or moves a content item. Move an item by specifying a different parent item. The content indicated by **metadata** has its name changed to **newName**. This method is called when moving or rename reports and when renaming folders. Folders are moved with the void MoveFolder method.
Arguments	• metadata: the **ContentMetadata** object to be renamed or moved. • newname: string with the new name for the content item • parentMetadata: a **ContentMetadata** object representing the parent of metadata. To move a content item to a new folder, specify a different parent object here. To keep the item in the same folder, specify the content item's existing parent here.

void DeleteContent(ContentMetadata metadata)

Description	Delete the content in the **metadata** object. This method is called anytime a content item is deleted, after the user interface requests confirmation from the user the item should be deleted.
Arguments	• metadata: the **ContentMetadata** object to be deleted.

void SaveContent(ContentMetadata metadata)

Description	Save the content in the **metadata** object. This may be a new or updated content. This method is called when writing changes to content into the database. This could be a report, or a theme. Folders are added with the separate void AddFolder(ContentMetadata metadata) method.
Arguments	• metadata: the **ContentMetadata** object to be saved.

StorageMgmtResult ValidateContentName(string reportName, string udfData)

Description	Used to validate that a report name is legitimate and allowed in the database. The method should permit only valid report names to be saved in the database. In Exago's implementation of the interface, this method simply returns null as report name validation is handled elsewhere in the application.
Arguments	• reportName: the report name to be validated • udfData: additional UDF report data
Returns	A **StorageMgmtResult** object that can be queried for a result code.

void AddFolder(ContentMetadata metadata)

Description	Add a new folder. The new folder's parent is set in the **metadata** object.
Arguments	• metadata: a **ContentMetadata** object representing the folder to be added.

void MoveFolder(ContentMetadata folder, ContentMetadata target, bool updatePermissions)

Description	Move a folder into a new location, and optionally copy the permissions from the destination folder to content in the one being moved. This method is called whenever a folder is moved, by dragging it around the Report Tree or when choosing Move Folder to Root from the right-click menu.
Arguments	• folder: a **ContentMetadata** object representing the folder to be moved. • target: a **ContentMetadata** object representing the destination where **folder** will be moved to. • updatePermissions: *true* if the permissions of **target** should be copied to content in **folder** or *false* if the permissions should not be copied. The permissions will only be changed for content that the current user is the owner of.

IEnumerable<string> GetTemplateList()

Description	Return a collection of template names. This method is called when opening the **Template** dialog on the Report Designer's Settings menu and after after selecting a new Template file to upload with the **Upload** button.
Arguments	none
Returns	A collection of strings with names of the templates in the database.

byte[] GetTemplate(string templateName)

Description	Get the binary conrents of a Microsoft Word, Microsoft Excel or PDF template file. This method is called when selecting a template from the dropdown list in the **Template** dialog, immediately after uploading a new template and after saving a report that includes a template.
Arguments	• templateName: the name from the Content table of the template file to read.
Returns	A byte array of the binary contents of the template file.

void SaveTemplate(string templateName, byte[] templateData)

Description	Save the binary contents of a Microsoft Word, Microsoft Excel or PDF template file to the database. This method is called to Upload a new template to the database from the Templates dialog in the Report Designer.
Arguments	• templateName: the name from the Content table of the template file to be saved. • templateData: a byte array of the binary contents of the template file.

IEnumerable<ContentMetadata> GetThemeList(string themeType)

Description	Retrieve a collection of theme names that match the specified theme type as **ContentMetadata** objects. This method will be called from the Report Designers when they are opened, and before calling void SaveTheme(string themeType, string themeName, string themeData)
Arguments	• themeType: the type of theme to retrieve (e.g. *ExpressView*, *Chart*, *Map*, *CrossTab*, *Express*)
Returns	An enumerable collection of **ContentMetadata** objects representing all of the available themes of the requested type.

bool ExistTheme(string themeType, string themeName)

Description	Check to see if a theme exists in the database. This method is called when saving a theme to see if it already exists.
Arguments	• themeType: the type of the theme to check for. • themeName: the name of the theme to check for.
Returns	*true* if the theme exists in the database, *false* if it does not

string GetTheme(string themeType, string themeName)

Description	Get the content of the theme. This method will be called by a Report Designer to retrieve the theme data to apply it to the report.
Arguments	• themeType: the type of the theme to read. • themeName: the name of the theme to read.
Returns	A string with the text content (XML, JSON or CSV) of the theme.

void SaveTheme(string themeType, string themeName, string themeData)

Description	Save new contents into a theme. If the theme exists, it will be overwritten. This method is called when a theme is saved from one of the Report Designers.
Arguments	• themeType: the type of theme that will be saved • themeName: the name of theme that will be saved • themeData: the XML, JSON or CSV content of the theme that will be saved
Example	`this.SaveTheme("Chart", "Crimson Mountain", "#F0433A,#C9283E,#820333,#540032,#2E112D")`

void ClearReportListCache()

Description	Clear the report list cache. This method should be called when saving or updating reports to make sure the Report Tree is up to date with the content changes.

string TestConnection()

Description	Check the connection to the Storage Management database specified in the configuration file is valid. This method is called when clicking the **Check Database Settings** button in the **Admin Console**.
Returns	A string containing any error message, otherwise null if the connection is successful.

string InitDb(string jsonSchemaFilePath, bool showSQL)

Description	Initialize a Storage Management database with the schema definition found in the provided JSON file. This method is called when clicking the **Initialize Database** or **Show Schema SQL** buttons in the **Admin Console**. More information can be found in the Storage Management: Database Schema article. **TIP** Default location is <WebApp>ConfigOtherStorageMgmtSchema.json where <WebApp> is the installation directory of the Web Application.
Arguments	• jsonSchemaFilePath: full path to the schema definition JSON file. • showSQL: whether or not database intializing SQL statements should be returned
Returns	If showSQL is *true*, this method will return SQL statements that can be used to initialize a Storage Management database. If showSQL is *false*, this method will return *null*.

Virtualized Methods v2021.1+

Below is a basic example of a class that overrides only the **SaveContent** and **AddFolder** methods.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using WebReports.Api.Common;
using WebReports.Api.ReportMgmt;
using WebReports.ContentDatabase;

namespace V_versions
{
 public class V_storage : StorageMgmtDatabase
 {
  private static readonly Logger logger = Log.GetLogger();

  /**
   * Provide a constructor
   */
  public V_storage(StorageMgmtConfig config) : base(config)
  {
  }

  /**
   * Override the SaveContent method
   * <param name="metadata">the ContentMetadata object to be saved</param>
  */
  public override void SaveContent(ContentMetadata metadata)
  {
   //Custom implementation of the method goes here
  }

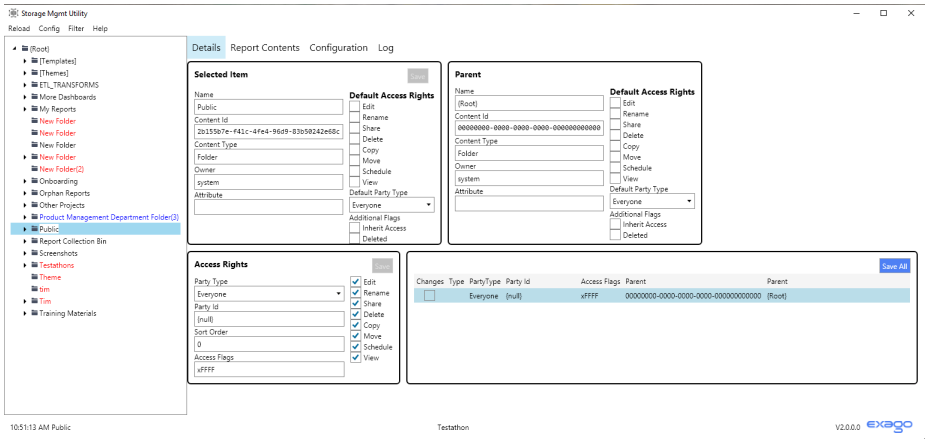
  /**
   * Override the AddFolder method
   * <param name="metadata">a ContentMetadata object representing the folder to be added</param>
  */
  public override void AddFolder(ContentMetadata metadata)
  {
   //Custom implementation of the method does here
  }
 }
}
```

Then, based on this example in the **Admin Console**, set the following:

- **Assembly Location** to the full path to the DLL produced by the project
- **Class Name** â€" V_versions.V_storage

Storage Management: Utility (v2021.1+)

Exago provides the **Storage Management Utility** (**SMU**) to view and manage the content access records and permissions on content items at a low level. Based on .NET Core 3.1, it will run on supported versions of Windows, Mac and Linux. It is only available in English language. The SMU can be downloaded from the Downloads page on our Support Site.



The main SMU window

Recommended Reading

The following articles on the Exago Support Site provide background information for the Storage Management system:

- Storage Management: Introduction
- Storage Management: Database Schema
- Storage Management: Admin Console

Installation

There are no pre-requisties for using the Storage Management Utility. The download includes all drivers and runtimes necessary. The operating system must be compatible with .NET Core 3.1+.

1. From the Downloads page on the Support Center, download the correct version for the operating system in use.
2. Extract the file to an easy to remember location.

> **TIP**
>
> Create a new directory in the Exago installation directory, e.g. C:Program FilesExagoSMU or /opt/Exago/SMU

3. Change directories to the location of the extracted files. Execute **StorageMgmtUtility.exe**.

Configuration

Initial Setup

When first launched, the **Storage Management Utility** must be configured to connect to the Storage Management database.

1. On the menu bar, click on **Config** to open the **Options** dialog.
2. In the list of profiles on the left, click either *New*profile to load the default profile.
3. In the **Database Connection** section on the right, enter in the details to connect to the Storage Management database. These values can be found in the Exago **Admin Console** > **Storage Management** section.
   - **Name** â€" enter a friendly name to reference this profile.
   - **Type** â€" select the type of database either *Microsoft SQL Server*, *MySQL*, *SQLite*, *Oracle* or *PostgreSQL*
   - **Provider** â€" enter the database provider (driver) to use to connect to the database. Choosing a **Type** above will suggest a provider to use automatically
   - **Connection** â€" enter the connection string to connect to the Storage Management database
   - **Prefix** â€" enter the table prefix, if applicable
4. Click **Test** to test the connection to the database server. If successful, the utility will respond with the number of records in the content, content access and party type tables.
5. Click **Ok** to choose the profile and close the Options dialog.

The configuration file is saved to a local application data directory that will vary with operating system. Use the **ConfigDir** menu bar item to retrieve the location of the configuration file directory.

The utility will also save the last 10 saves to this configuration file for version control, backup and recovery purposes. Files will be saved.

Subsequent starts of the Storage Management Utility will load the last selected profile automatically.

Managing Profiles

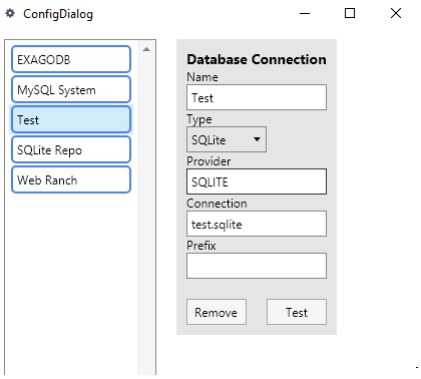
Any number of database profiles may be saved in the configuration and recalled at a later time.

Creating a New Profile

1. On the menu bar, click on **Config** to open the **Config** dialog.
2. Click **New** to create a new profile.
3. Resume from Step 3 in the Initial Setup section above.

Editing a Profile

1. On the menu bar, click on **Config** to open the **Config** dialog.
2. Select a profile from the list by clicking on it. The currently active profile always appears at the top of the list.



3. Make changes as desired.
4. Click **Ok** to save the changes to the profile.

Deleting a Profile

1. On the menu bar, click on **Config** to open the **Config** dialog.
2. Select a profile from the list by clicking on it.
3. Click **Remove**.

User Interface

Menu Bar

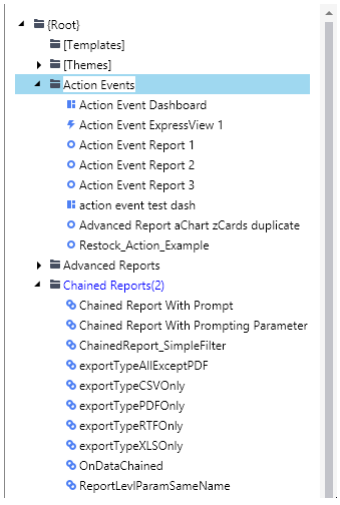


The Title and Menu bars at the top of the window

The **Menu Bar** contains four clickable options to control the application:

- **Reload** â€" connect to the Storage Management database and refresh the **Content Tree** with its content. Any unsaved changes are discarded.
- **Config** â€" open the **Options** dialog to create, edit or view configuration profiles
- **Filter** â€" filter the items in the **Content Tree** by name, content ID, content type, report type, deleted or empty status. More information is in the Filtering the Content Tree section below.
- **ConfigDir** â€" open a dialog that displays the location of the connection configuration file, with an option to the open the directory

Content Tree



Content tree in the SMU showing several folders, reports and some deleted items

The left side of the SMU window is the **Content Tree**. The Content Tree lists all of the content items in the database's **content** table in alphabetical order. The tree is refreshed when the **Reload** item is clicked on the menu bar.

Content in the tree is identified by icons:

- â€" this item is a folder
- â€" this item is an **Advanced Report** or a **CrossTab**
- â€" this item is a **Chained Report**
- â€" this item is a **Dashboard**
- â€" this item is an **Express Report**
- â€" this item is an **ExpressView**
- â€" this item is a theme

- â€" this item is a document template

Items that appear with red text have been deleted by the content owner (the deleted flag is *true*).



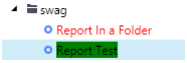
The folder **My Report1** has been deleted by its owner

Items that appear with blue text followed by a number indicate that there are multiple content access rtecords for that item.



The Advanced Report **Standard Report** has 2 content access records

Items that appear with green background text indicate unsaved changes to one or more content access records. See 3. Access Rights and 4. Other Access for more information.

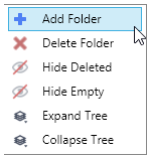


The Advanced Report **Report Test** has an unsaved content access record change

The top of of the **Content Tree** will always have a folder name {Root}, which is the parent of all other content items in the system. Clicking on any item in the **Content Tree** except for the {Root} will load the details for that content item into the **Detail Pane** on the right side of the window.

Themes and template files will appear in “pseudo folders”, named [Themes] and [Templates] respectively. These folders do not actually exist, they only appear in the Storage Management Utility for organization purposes. Themes and templates are actually stored in the root directory in the Storage Management database.

Right-clicking on items in the content tree opens a context-menu with additional options:



Content Tree context-menu

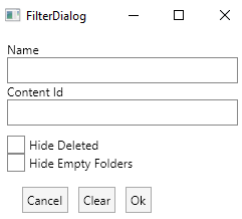
- **Add Folder** â€" add a new child sub folder to this folder

> **CAUTION**
>
> New folders are created with only a single content record, regardless of the Inherit Flag on its parent or other configuration. Setting content access records for new folders manually is required when creating folders with the utility.

- **Delete Folder** â€" expunge the folder from the database, folder must be empty before it can be removed.
- **Hide Deleted/Show Deleted** â€" hide all content that is marked deleted from the content tree, or show it again if its already hidden
- **Hide Empty/Show Empty** â€" hide empty folders, or show them again if already hidden
- **Expand Tree** â€" expand this folder and all its child folders
- **Collapse Tree** â€" collapse this folder and all its child folders

Filtering the Content Tree

To filter the items displayed in the **Content Tree:**



Filter Criteria dialog

1. Click **Filter** in the Title and Menu Bar to open the **Filter Criteria** dialog.
2. Enter one or more criteria to filter by:
   - **Name** â€" the name of the item as it appears in the Exago BI application
   - **Content Id** â€" the GUID that uniquely identifies this content item in the system
   - **Hide Deleted** â€" check this checkbox to hide any content items that have been marked deleted by the content owner (i.e. the item’s deleted flag is true). Items individually deleted by non-owners will still appear as content access records.
   - **Hide Empty Folders** â€" check this checkbox to hide folders that appear empty in the Content Tree. A folder is considered empty if it contains no content or if other filter criteria hides any content.
3. Click the **Ok** button to apply the filters to the content tree, **Clear** to remove all filter criteria selections or click **Cancel** to close the dialog without changing the filters. The dialog will close.

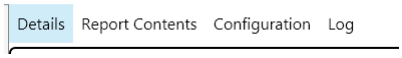
Clearing Filters

To clear filters and show the entire Content Tree again:

1. Click **Filter** in the Title and Menu Bar to open the **Filter Criteria** dialog.
2. Click the **Clear** button. The filters will be removed and the dialog will close.

Details Pane

The right side of the SMU window is the **Detail Pane**. The Detail Pane allows viewing and editing of the access records for content items. It is divided into four tabs: **Details**, **Report Contents**, **Configuration** and **Log**.

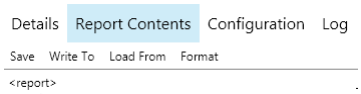


Details, **Report Contents**, **Configuration** and **Log** tabs at the top of the Detail Pane

Report Contents tab

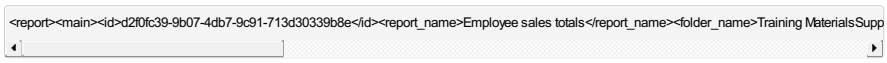
The **Report Contents** tab displays the XML report definition content when a report is selected in the Content Tree. It is disabled if any other content type is selected.

With an XML report definition displayed, the four buttons along the top of the tab become active:



Report Contents buttons

- **Save** â€" save any changes made to the XML back to the Storage Management database
- **Write To** â€" save the displayed XML content to a file on the local computer
- **Load From** â€" read an XML report definition from a file on the local computer into the Storage Management Utility
- **Format** â€" refresh the line breaks and indentation in the XML file to make it more readable. For example:



A portion of an Advanced Report's XML definition before **Format**

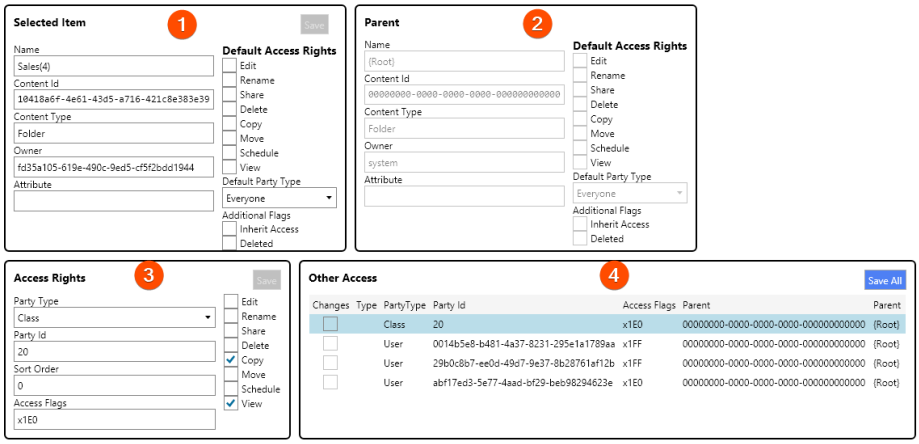
```
<report>
 <main>
  <id>d2f0fc39-9b07-4db7-9c91-713d30339b8e</id>
  <report_name>Employee sales totals</report_name>
  <folder_name>Training MaterialsSupport LabsLinked Reports â€" April 2020Employees example</folder_name>
  <folder_id>00000000-0000-0000-0000-000000000000</folder_id>
  <version>1</version>
  <type>Standard</type>
  <sql_stmt />
  <show_execute_form>False</show_execute_form>
  <filter_execution_window>Default</filter_execution_window>
```

The same report's XML after **Format**

> **NOTE**
>
> Remember to click **Save** after loading content with the **Load From** function to save the content to the database.

Details tab

The **Details** tab displays content item information. It is further divided into four panels:



Anatomy of the SMU Detail Pane Details tab

1. **Selected Item** â€" displays the content metadata for the selected item, loads the content access records into the 4. Other Access panel
2. **Parent** â€" displays the content metadata for the selected item's parent item (i.e. the folder this content item is stored in) based on the currently selected **4. Other Access**
3. **Access Rights** â€" works in conjunction with the 4. Other Access panel to display the access rights for the currently selected access record
4. **Other Access** â€" displays all of the content access records on the selected content item

1. Selected Item

The **Selected Item** panel displays metadata saved in the Storage Management system's content table for the currently selected item in the **Content Tree**.

- **Name** (read-only) â€" the name of the item as it appears in the Exago BI application
- **Content Id** (read-only) â€" the GUID that uniquely identifies this content item in the system
- **Content Type** (read-only) â€" the type of content this item is, either *Folder*, *Report*, *Theme* or *Template*
- **Report Type** (read-only) â€" only displayed when **Content Type** is *Report*. Displays the type of report this content item is. Either *Chained*, *Dashboard*, *Express*, *ExpressView*or *Advanced*

> **NOTE**
>
> CrossTabs are also considered Advanced Reports.

- **Owner** (read-only) â€" the value of the **Owner Id** identity key when this content item was created
- **Attribute** (read-only) â€" only displayed when **Content Type** is *Theme*. Displays the component in the application the theme applies to. Either *ExpressView*, *ExpressView_v2*, *Express Report*, *CrossTab*, *Chart* or *Map*
- **Default Access Rights** (editable) â€" only applicable for folders. Displays the value for each of the Access Flags that will be applied to new content created in the folder.
- **Default Party Type** (editable) â€" only applicabler for folders. Displays the default party type that will be applied to new content created in the folder. Can be changed by selecting a different value from the dropdown.
- **Additional Flags** (editable) â€" only applicable for folders and reports. Displays or sets the values of certain flags on the content item:
  - **Inherit Access** â€" only applicable for folders. Displays the value of the folder's **inherit flag** which determines if content access records are copied from the folder to new content created in the folder. Can be changed by checking/unchecking the checkbox.
  - **Deleted** â€" Displays the value of the item's **deleted flag** which indicates the item was deleted by the content owner and should not be visible to any user of the system. Can be changed by checking/unchecking the checkbox.
- **Description** (read-only) â€" only displayed for Reports. Displays the description field for a report
- **Created By** (read-only) â€" only displayed for Reports. The value of the **User Id** identity key when this report was created
- **Created Date** (read-only) â€" only displayed for Reports. The UTC timestamp at the instant when this report was created
- **Modified By** (read-only) â€" only displayed for Reports. The value of the **User Id** identity key when this report was last saved
- **Modified Date** (read-only) â€" only displayed for Reports. The UTC timestamp at the instant when this report was last saved

If the **Access Flags**, **Default Party Type** or **Additional Flags** are changed, the word **Changes** will appear at the top of the panel. Changes can be saved by clicking on the **Save** button in the top right corner. Doing so will write the changes to the Storage Management database immediately.

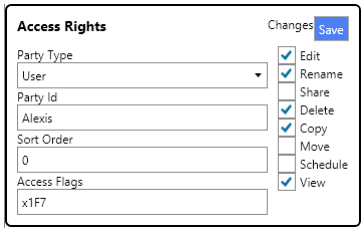
2. Parent

The **Parent** panel displays metadata saved in the Storage Management system's content table for the currently selected item's parent in the **Content Tree**.

All controls in this panel are read-only. The parent of a content item can be changed by right-clicking on a content access record in the 4. Other Access panel

- **Name** â€" the name of the item as it appears in the Exago BI application, except when the parent is the root folder, when {Root} is displayed instead.
- **Content Id** â€" the GUID that uniquely identifies this content item in the system
- **Content Type** â€" the type of content this item is, either *Folder*, *Report*, *Theme* or *Template*
- **Owner** â€" the value of the **Owner Id** identity key when this content item was created
- **Default Access Rights** â€" only applicable for folders. Displays the value for each of the Access Flags that will be applied to new content created in the folder.
- **Default Party Type** â€" only applicable for folders. Displays the default party type that will be applied to new content created in the folder. Can be changed by selecting a different value from the dropdown.
- **Additional Flags** â€" only applicable for folders and reports. Displays or sets the values of certain flags on the content item:
  - **Inherit Access** â€" only applicable for folders. Displays the value of the folder's **inherit flag** which determines if content access records are copied from the folder to new content created in that folder.
  - **Deleted** â€" Displays the value of the item's **deleted flag** which indicates the item was deleted by the content owner and should not be visible to any user of the system.

3. Access Rights



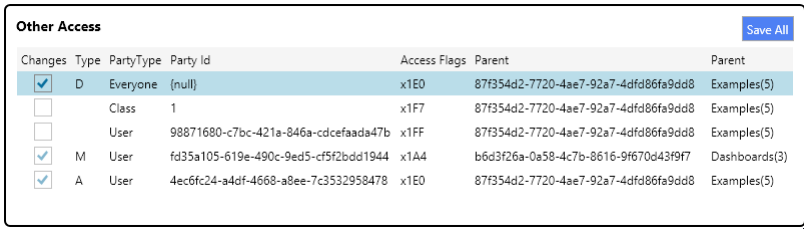
Access Rights panel in Detail Pane

The **Access Rights** panel displays the **party type**, **party id**, **sort order** and **access flags** of the currently selected item in the 4. Other Access panel.

- **Party Type** â€" displays the access record's party type. Can be changed by selecting a different value from the dropdown.
- **Party Id** â€" displays the access record's party id. This is the value of the **Party Type** thay this record applies to. Can be changed by entering a new value into the textbox.
- **Sort Order** â€" displays the access record's sort order value. Can be changed by entering a new value into the textbox.
- **Access Flags** â€" displays the access flags as a hexadecimal value. Can be changed by altering the nine checkboxes.
- **Checkboxes** â€" set or unset the various access flags.

Whenever changes are made, the word **Changes** will appear at the top of the panel, and the item will have a green background in the content tree. Changes can be saved by clicking on the **Save** button in the top right corner, doing so will write the changes to the Storage Management database immediately.

4. Other Access



The Other panel in this figure is displaying three **User** level records, one **Class** level record and an **Everyone** record

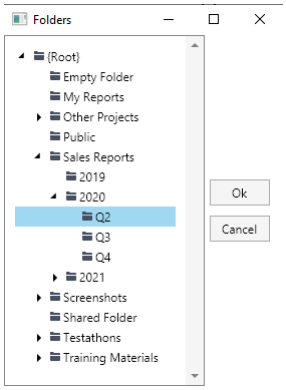
The **Other Access** panel displays all of the content access records that exist for a particular content item, based on the content's **content id**. It also allows for adding, modifying and deleting content access records.

Clicking on any of the records in the panel will update the 3. Access Rights panel with that record's information, as well as highlighting the location of the content item in the Content Tree. Since content can appear in different folders for different parties, it may be in different locations in the tree for different records.

- **Changes** â€" a check will appear in the checkbox if there are unsaved changes to the content access record. Click the **Save All** button to save them.
- **Type** â€" the type of change that will be saved to the Storage Management database upon clicking the **Save** button.
  - *A* â€" a new contrent access record to be **added**
  - *D* â€" this content access record will be **deleted**
  - *M* â€" this content access record is **modified**
- **Party Type** â€" the party type name from the party type table that this content access record applies to
- **Party Id** (editable) â€" the value of the identity key that correlates with the **Party Type** that this content access record applies to
- **Access Flags** â€" displays the access flags as a hexadecimal value.
- **Parent** â€" the content ID of the parent that contains this content item for this particular party
- **Parent** â€" the name of the parent that contains this content item for this particular party

Right-clicking on items in the Other Access panel opens a context-menu with additional options:

- **New Access Record** â€" create a new user level access record with a GUID as the party ID and the currently selected content access record's parent. Modify the record with the 3. Access Rights panel and the Party Id. The new record is not saved until clicking the **Save All** button in the panel.
- **Change Parent** â€" change the parent of the currently selected content access record. A **Folders** dialog will be opened, to choose a new parent by name. Clicking **Ok** in the dialog will save the change to the content access immediately.



- **Delete Access Record** â€" mark the currently selected content access record for expunging from the table. No changes are made until the **Save All** button is clicked.

Configuration

The **Configuration** tab presents an editable list of the database profiles in the configuration file. Changes are stored in memory, and are only committed to the conifiguration file the next time the **Ok** button is clicked in the config dialog.

Log

The **Log** tab displays the contents of the Storage Management Utility's log (StorageMgmtQuery.log in the utility's installation directory). It will only show entires from the time the utility was last started.

Logging for the application may be customized by editing the log4net.config.xml file in the utility's installation directory. See the log4net section of the Application and Performance Logging article for additional information.

Status Bar

The **Status Bar** at the bottom of the SMU window displays information about the utility's communication with the Storage Management database and the local file system, the loaded configuration and the version number of the utility.



Status bar message indicating the Content Tree was last refreshed at 8:45:28 PM

The status bar is broken up into three zones:

1. Messages â€" accompanied with a timestamp indicating when the operation was completed
2. The name of the currently loaded configuration profile. In the figure above, the currently loaded profile is named *My SM Test Environment*.
3. The version number of the Storage Management Utility, and the Exago logo

Scheduler Service Configuration

To configure the **Scheduler Service**, edit the file eWebReportsScheduler.xml. This file can be found in the Scheduler Service's installation directory. By default, this is C:\\Program Files\Exago\ExagoScheduler on Windows and /opt/Exago/Scheduler on Linux.

The following settings are available:

> **IMPORTANT**
>
> Settings that can be true or false are case sensitive and must use lower case.

<smtp_server> â€" The SMTP server used by the Scheduler Service to email reports.

<smtp_enable_ssl> â€" Set to true to enable SSL.

<smtp_user_id> â€" The user id that is used to login into the SMTP server.

> **NOTE**
>
> If the user id is set to a null or empty value or *DefaultCredentials*, the value automatically defaults to the user's current credentials. For anonymous emailing see theÂ **Credential-less SMTP**Â section below.

<smtp_password> â€" The password that is used to login into the SMTP server. This value is only considered when using the user id as an SSL credential.

> **NOTE**
>
> If the password is set to a null or empty value, the value automatically defaults to the user's current credentials.

<smtp_from> â€" The 'From' email address used in the report emails.

<smtp_from_name> â€" The 'From' name used in the report emails.

<error_report_to> â€" The email address to send error reports to.

<channel_type> â€" tcp or http â€" must match the setting of the **Remote Host** in the Admin Console **Scheduler Settings**.

<port> â€" The port number of the .NET remoting object used to communicate with Exago; this should also be entered in the Admin Console **Scheduler Settings**.

<working_directory> â€" The directory where scheduled documents and temporary files are written. The default setting [INSTALLDIR]working creates a working folder in the Scheduler Service's installation location.

<default_job_timeout> pre-v2019.2.31 pre-v2020.1.14 pre-v2021.1.2 â€" The maximum number of seconds any report execution is allowed. If an execution reaches a maximum number of seconds, the job is marked Abended and an email will be sent to the address specified under <error_report_to>. Email jobs use default_job_timeout exclusively for timing out the connection to the SMTP server.

For *v2019.2.31+*, *v2020.1.14+* and *v2020.1.2+*, this value is ignored. Use smtp_timeout to limit SMTP connection time and max_job_execution_minutes exclusively.

> **IMPORTANT**
>
> <default_job_timeout> and <max_job_execution_minutes> work in conjunction. They should both be set to the same time period. Note that <default_job_timeout> is set in seconds and <max_job_execution_minutes> is set in minutes.

<smtp_timeout> v2019.2.31+ v2020.1.14+ v2021.1.2+ â€" limit connection time to the SMTP server in seconds. The default is *30*.

<report_path> â€" A path to specify where to save reports when 'Email Scheduled Reports' is set to *false* in the Admin Console. For more details see Saving Scheduled Reports to External Repository below.

<sleep_time> â€" The time interval (in seconds) used for polling for scheduled reports to execute.

<simultaneous_job_max> â€" The maximum number of report executions that can occur simultaneously. This setting is based on the resources available of the server where the Scheduler Service is installed.

<logging> â€" Logging is on by default. To turn logging off, set to OFF (in all-caps). To configure logging, edit the **Logging Settings** in the eWebReportsScheduler.exe.config file.

<flush_time> â€" The number of hours that a completed, deleted, aborted or abended (*v2021.1.6+*) job will be saved for viewing in the

Schedule Manager. Set to *0* to flush jobs immediately upon completion. Set to *-1* to disable automatic flushing only at the direction of an Exago Support Analyst. Flushing should be done automatically at all other times. The default value is *24* in *v2021.1.9+*.

<sync_flush_time> â€" The number of hours that a completed, deleted, or aborted Remote Execution job will be saved in the Scheduler Service's job list. Set to *0* to flush jobs immediately upon completion (default). Set to *-1* to disable automatic flushing only at the direction of an Exago Support Analyst. Flushing should be done automatically at all other times.

> **NOTE**
>
> Synchronous jobs are never displayed in the
>
> **Schedule Manager**, even if this value is *greater than 0*.

<email_addendum> â€" Text that will be added at the end of email body. Use n to insert lines.

<external_interface> â€" Overrides the Admin Console's setting for the External Interface, a deprecated extensibility feature that should not be used for new installations.

<abend_upon_report_error> â€" This controls how the scheduling service should proceed if an error occurs while loading or executing a report. The default true will stop the running the schedule and set the status to *Abended*. Set to false to continue running the schedule and maintain the status as 'Ready'.

<ip_address> â€" Binding IP address for the Scheduler Service. Most commonly used when the server has multiple Network Interface Cards (NICs).

<encrypt_schedule_files> â€" Set to true to encrypt the files created by the Scheduler Service. All existing schedules will be encrypted the next time the service is started.

<max_temp_file_age> â€" The number of minutes between each flush of the temp files created by the scheduling service. The default is 1440 minutes (24 hours).

> **NOTE**
>
> Setting <max_temp_file_age> too low may result in errors as temp files are used during report execution and for interactive HTML capabilities when using remote execution. It is not recommended setting this value any lower than 60 minutes. Execution cache files will not be flushed.

<email_retry_time> â€" In the case an email fails to send, the number of minutes to wait before retrying to send the email. After five failed attempts the schedule will set itself to 'Aborted'. The default is 10 minutes.

<max_job_execution_minutes> (*v2016.2.12+*) â€" Maximum amount of time (in minutes) to run an execution job before timing out.Â If the job times out, the schedule will be marked as Aborted.

> **NOTE** PRE-V2019.2.31 PRE-V2020.1.14 PRE-V2021.1.2
>
> <default_job_timeout> and <max_job_execution_minutes> work in conjunction. They should both be set to the same time period. Note that <default_job_timeout> is set in seconds and <max_job_execution_minutes>is set in minutes.
>
> For *v2019.2.31+*, *v2020.1.14+* and *v2020.1.2+*, <default_job_timeout> is no longer applicable.

<secure_channel> (*v2016.3+*) â€" Set to true to allow receipt of encrypted data from hosts. The setting **Use Secure Scheduler Remoting Channel** must be true in the Admin Console/XML configuration file.

<security_protocol> (*v2016.3.4+*) â€" Specify which security protocol(s) theÂ scheduler should use. Possible values:Â Ssl3, Tls, Tls11, Tls12, Tls13 (*.NET v4.6+*). Separate multiple values with commas (,).

<service_name_tag>â€" For manualÂ installation of Scheduler Services usingÂ Visual StudioÂ ***installutil.exe***, this field is appended to the end of the service name. UsefulÂ for installing multiple services on the same server. *installutil.exe* must be in the same folder as the scheduler configuration file.

<webappbaseurl> â€" Set to the virtual directory URL where the Exago BI Web Application is installed. In Cross-Origin Resource Sharing (CORS) environments, setting this flag ensures that images are correctly loaded.

<load_assembly_in_external_domain> (*v2018.1+*) â€" Set to *True* to load Assembly Data Sources in an external domain, as opposed to the application domain. When *True*, this feature also should be enabled in the Web Application's appSettings.config and and the Scheduler Service's eWebReportsScheduler.exe.config files.

Example XML File

```
<?xml version="1.0" encoding="utf-8" ?>
<eWebReportScheduler>
 <smtp_server>smtp.office365.com:587</smtp_server>
 <smtp_enable_ssl>true</smtp_enable_ssl>
 <smtp_user_id>[email protected]</smtp_user_id>
 <smtp_password>notmypassword#1234</smtp_password>
 <smtp_from>[email protected]</smtp_from>
 <smtp_from_name>Exago Scheduler</smtp_from_name>
 <error_report_to>[email protected]</error_report_to>
 <channel_type>tcp</channel_type>
 <port>2011</port>
 <working_directory>[INSTALLDIR]working</working_directory>
 <default_job_timeout>3600</default_job_timeout>
 <sleep_time>15</sleep_time>
 <simultaneous_job_max>10</simultaneous_job_max>
 <logging>on</logging>
 <flush_time>-1</flush_time>
 <sync_flush_time>0</sync_flush_time>
 <email_addendum></email_addendum>
 <external_interface></external_interface>
 <report_path>[INSTALLDIR]</report_path>
 <abend_upon_report_error>true</abend_upon_report_error>
 <ip_address></ip_address>
 <security_protocol></security_protocol>
 <encrypt_schedule_files></encrypt_schedule_files>
 <max_temp_file_age>1440</max_temp_file_age>
 <email_retry_time>10</email_retry_time>
 <queue_service></queue_service>
 <load_assembly_in_external_domain>false</load_assembly_in_external_domain>
</eWebReportScheduler>
```

Starting and Changing Scheduler Services

The Windows Service will have to be manually started for new installations of the Scheduler. Starting the service will create the working directory as set in <working_directory> described above.

To start the scheduler open Windows Services. Double click on 'ExagoScheduler' and the Properties menu will appear. Click **Start**.

If any changes are made to the configuration (detailed above) the service must be stopped and restarted for the changes to take effect.

For starting and stopping Scheduler Services on Linux servers, refer to the Scheduler and Monitoring Services section of the Installing Exago on Linux article.

Credential-less SMTP v2018.1+

The values for <smtp_user_id>, <smtp_password>, <smtp_from>, and <smtp_from_name> cannot be removed or left blank. Otherwise, these values will be reset to their default or throw an error when attempting to send an email.

To set up anonymous report emailing:

- Set <smtp_user_id> to *NoSmtpCredentials*
- Set <smtp_password> to any non-null value

Saving Scheduled Reports to a Repository

Instead of sending report output as an e-mail, the output file can be saved to a file repository instead. To enable this:

1. Provide a **Report Path** in the Scheduler Service's configuration file. This is the location where the Scheduler Service will save the report output files. Add the <report_path> element if it does not yet exist and set the location of the repository.
2. In the **Admin Console** setÂ **General** >Â **Scheduler Settings** > **Email Scheduled Reports** to *False*.

After applying these settings, generic scheduled reports (i.e. the schedule does not run as a batch) will be saved directly to the specified repository rather than being sent via email.

If theÂ

General >Â **Scheduler Settings** > **Show Schedule Delivery Type Options** setting is *True*, users can select whether to save a scheduled report to the specified repository or to send it via email.

Output files will be saved to the <report_path> directory with the filename based on the **Scheduler Service**'s working file GUID with _Download appended. For example, 855b5857-38ee-473f-8f2c-f3ce36393ef2_Download.pdf.

Server Events for Handling Scheduled Reports

The **OnScheduledReportExecuteSuccess** Server Event, if defined, is called after a scheduler executes a file, but before emailing it or saving it to disk. This event can override Exago's default email or archival operations. For example, the file names could be changed to better reflect their user and content, or files could be uploaded to an FTP server or other location.

For assistance with this Server Event, contact the Exago Services Team.

Time Zone Calculation Enhancements in v2019.1

Geopolitical time zones present an inherent problem for scheduling and date calculations. Their UTC offset values may vary throughout the year

based on if and when a user's or server's location observes daylight saving time (DST). For example, a user in California and server in England would enter and exit DST on different dates, necessitating different UTC offset values throughout the year. This displacement of the UTC offset may cause schedules to run ahead or behind their intended times or even be ignored entirely.

With the release of *v2019.1*, Exago BI now offers an easy way to accurately handle different users and server spread across different geopolitical time zones.

What Changed

In previous versions of Exago BI, the client and server time offset would be compensated for based on the a static setting called **Server Time Zone Offset**, a specified number of +/- hours to calculate the difference in time between client and server. This setting, however, is incapable of considering the fluctuation in server time offset caused by the non-uniform observations of DST in different geopolitical regions.

To better handle this is issue Exago now has a new setting called **Client Time Zone Name** which works in conjunction with **Noda Time**, an advanced .NET library, to accurately consider geopolitical time zones through the year as schedules and calculations are done.

Client Time Zone Name

To accurately handle time zones there is a new **Culture Settings** field called **Client Time Zone Name** where the user's geopolitical time zone may be specified.

> **NOTE**
>
> Using **Client Time Zone Name** is highly recommended as a best practice. However, if **Client Time Zone Name** is not given a value then Exago BI will fall back on its pre-*v2019.1* behavior and use the **Server Time Zone Offset** setting. In general the **Server Time Zone Offset** setting should only be provided if a **Client Time Zone Name** cannot be determined.

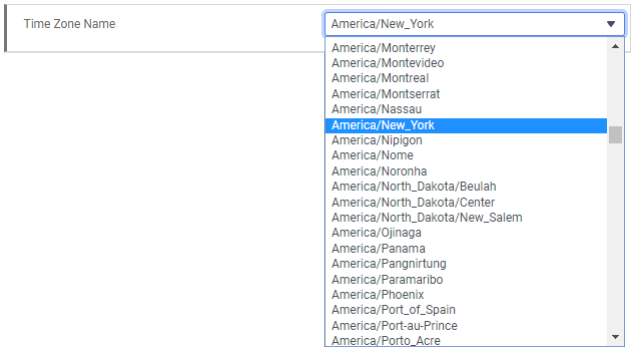
Adjusting the Client Time Zone Name

The **Client Time Zone Name** setting may be set in the following ways:

- Within the **Admin Console**
- Modifying the configuration file XML
- Via the .NET and REST APIs

Admin Console

Within **Culture Settings** in the **Admin Console**, the default **Client Time Zone Name** can be specified. Setting a default value is recommended if all or a majority of clients and users are located within the same geopolitical region. If clients and users are located in many different regions, then a default is not required as this information should be set dynamically through the use of Roles or via the API.



The dropdown contains a complete list of official IANA time zones. Expressed as locations rather than as deltas from UTC, each time zone in this list correlates to a specific geopolitical region and contains information on that region's observations of DST.

Configuration XML

The **Client Time Zone Name** can be manually edited within the configuration file XML using the following tag:

```
<clienttimezonename>America/New_York</clienttimezonename>
```

.NET and REST APIs

To modify this setting via the .NET API, the following property may be used:

```
PageInfo.SetupData.General.ClientTimeZoneName
```

This setting may also be modified via the REST API:

```
curl http://{webservice}/rest/settings?sid={sid} -XPATCH ^
    -d "{
        'General':
         {
          'ClientTimeZoneName': 'America/New_York'
         }
        }"
```

Noda Time Library

The standard .NET date and time library lacks important date/time concepts (e.g., “a date and time in a *specific* geopolitical time zone”). Due to the inherent issues that arise for scheduling processes because of DST and other non-linear time zone conflicts, Exago has replaced this library with a more powerful date and time library: **Noda Time**, a .NET library specifically designed to resolve date and time issues associated with geopolitical time zones and DST.

The **Schedule** module now utilizes the *Instant* and *LocalDateTime* classes of the Noda Time library. The *Instant* class represents a global, linear time in seconds. It does not map onto any calendar system and does not consider DST. The *LocalDateTime* class represents a point in time of a particular IANA time zone. These values can be mapped onto a calendar system and can take DST into consideration. The system operates using *Instants* while users operate in the *LocalDateTime* of their legislative time zone of residence.

The Noda Time library is also available for use by developers via the .NET API. For in-depth information on the Noda Time library, please see the Noda Time documentation.

Updates to the Scheduler Queue

> **NOTE**
>
> If you have not implemented the **Scheduler Queue**, then the following information will not affect your *v2019.1* upgrading process.

Certain DateTime values may have been assigned different meanings in the Scheduler Job XML following the implementation of Noda Time. Depending on the modifications made to your implementation of the **Scheduler Queue**, changes may have to be made to its code:

- If your implementation of the **Scheduler Queue** utilizes the QueueAPI object from WebReports.Api.Scheduler, then there should be no required for the **Scheduler Queue** to function properly after upgrading to *v2019.1*.
- If your implementation of the **Scheduler Queue** has been modified to directly parse the Job XML, then some DateTime values may have been assigned a different meaning and will need to be adjusted for. It is recommended that you contact our Services team at for help in determining which values will need to be adjusted.

Remote Execution

Report execution can be balanced across servers to improve performance. As one execution is being processed subsequent report execution calls will be sent to different servers. For each new job, Exago will prioritize the server with the lowest load (according to CPU and memory load) and ratio of running jobs to max jobs allowed. The number of jobs on a server will not exceed the value specified by the Scheduler Service’s <simultaneous_jobs_max> setting.

The following instructions provide an overview for setting up report execution on remote servers:

On each remote execution server:

1. Install the **Scheduler Service**. For detailed instructions see: **Scheduler Service Installation**.
2. The following conditions must be met:
   - The Scheduler Service version must match the Web Application version.
   - The Schedulerâ€™s language files and the Exago Applicationâ€™s language files must match.
   - Any custom assemblies must be present in the Scheduler Service’s /bin directory.
3. Configure the Scheduler Service. For detailed instructions see: **Configuring Scheduler Services**. By default the execution host will pass the reports back to the Exago Application. In order to save reports to an external repository, see: Saving Scheduled Reports to External Repository.

> **TIP**
>
> Multiple Scheduler Services can point to the same repository.

In the Admin Console:

1. Set **General** > **Scheduler Settings** > **Enable Report Scheduling** to *True*.
2. Set **General** > **Scheduler Settings** > **Enable Remote Report Execution** to *True*.
3. Set **General** > **Scheduler Settings** > **Remote Execution Remoting Host** to a comma or semicolon-separated list of the Scheduler Services. e.g. *http://MyHttpServer1:2001, tcp://MyTcpServer:2121*. The servers will be prioritized based on the listed order.

> **NOTE**
>
> When a Scheduler Service is used for both scheduling and remote execution, Exago will place immediate priority on Remote Execution tasks.

Execution Caching

Execution Caching allows the data from a report execution to be saved so that reports can be viewed more quickly and with less stress to the data source. The frequency and schedule of data source updates is customizable for each report. Updates can be scheduled for off-hours in order to better balance system load. Caching can significantly reduce data source calls for reports that are accessed frequently. Large or data-intensive reports, which can take a long time to run, especially benefit from caching.

> **NOTE**
>
> Execution caching is currently incompatible with the **Scheduler Queue**.

Enabling Execution Caching

> **IMPORTANT**
>
> This feature requires the installation and proper configuration of the Scheduler Service.

Execution Caching uses the **Scheduler Service** to cache the data at set refresh intervals. When caching is enabled for a report, a caching schedule is created that determines how often the data is refreshed. Scheduler Services need to be enabled for the user or users who will be setting the caches; however, users who only need to view the cached reports do not need to have scheduling enabled.

To enable execution caching, set **General** > **Scheduler Settings** > **Enable Execution Cache** in the **Admin Console** to *True*.

Cache Visibility Settings

The **userId and companyId** parameters must exist and have default values assigned to them in order to be able to enable execution caching. These parameters are used to assign visibility permissions to cached data.

The following **Admin Console** settings that affect access to some schedule recurrence options will also affect the recurrence options available in the execution caching window:

- **General** > **Scheduler Settings** > **Show Schedule No End Date Option**
- **General** > **Scheduler Settings** > **Show Schedule Intraday Recurrence Option**

The following **Admin Console** setting will affect the view level options that a user has when assigning permissions for an execution cache:

- **General** > **Scheduler Settings > User Cache Visibility Level** (*v2017.3+*)

The cache data is stored in the scheduler temporary directory, however cache files will not be deleted by the scheduler's routine cleanup.

Setting Cached Reports and Refresh Intervals

To set up caching for an advanced report:

> **CAUTION**
>
> Role-based filters and **column-based tenancy** apply based on the user who *creates* the cache. Users who can *view* the cache will see data filtered by the *creator's* role and tenancy, not the viewer.

1. From the

   Advanced menu, click **Execution Caching**.

2. Check the **Enable execution caching** check box.

3. Choose which users will be able to see the cached data:

   > **NOTE**
   >
   > Not every option may be available.

   - **User**: Users will only see cached data that they have created. When another user runs the report, it will execute against the database instead of the cache.
   - **Company**: Users will see cached data that has been created by anyone with the same **companyId parameter**. When a user from outside the company runs the report, it will execute against the database instead of the cache.
   - **Global**: The report has one data cache which is visible for all users.

4. Choose how often the cache is updated:

   a. Enter a time of day in the **Schedule Time** field, or use the picker to select a time.

   b. Optional: To update the cache multiple times per day, select the **Repeat Every** check box and enter a time for how often it should repeat.

   Optional: To repeat until a certain time of day, enter that time in the **Until** field, or use the picker to select a time.

5. Choose which days of the week the cache is updated. Select either:

   - To repeat every day or every number of days, click **Every __ day(s)** and enter a number of days for how often it should repeat.
   - To repeat every Mon, Tues, Wed, Thurs, and Fri, click **Every weekday**.

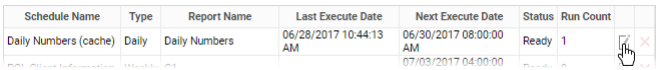
6. Click **OK**.

When execution caching is enabled for a report, the cache is not created immediately. To run the report, you can manually create the cache by clicking the **Cache Status**



icon, or you can wait until the first scheduled update completes. The cache can always be updated manually by clicking this icon.

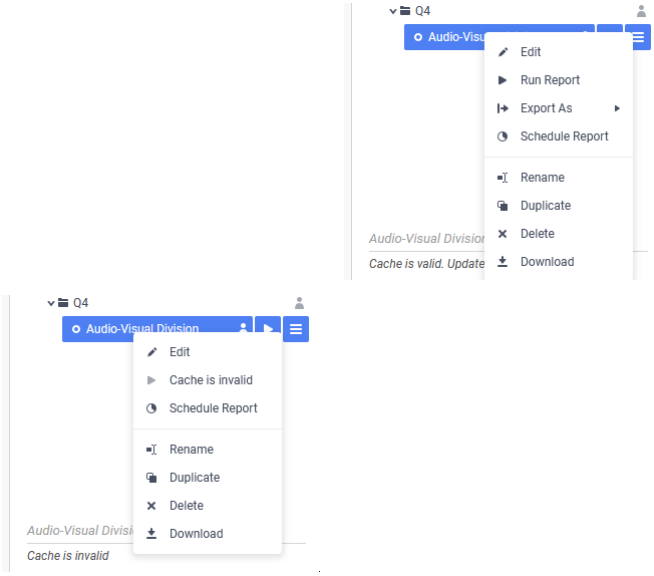
Cache schedules are visible in the schedule manager, with the label *(cache)*. Caches can be edited or deleted using the schedule manager or the advanced report designer.



Viewing a report cache in the schedule manager

Editing cached reports

Editing a cached report invalidates its cache. It cannot be run until the cache is updated. A missing or invalid cache is indicated by the message *Cache is invalid*, both in the report description and in the report tree menu.



Valid and invalid report cache indicators

It is recommended that you fully design a report first, so that you can manually verify the output, before enabling the cache.

Scheduler Queue

The **Scheduler Queue** is a custom-built application library that sits between the Exago core application and any number of **Scheduler Service** instances and handles how schedule traffic is managed. The Scheduler Queue is completely optional, but configurations with multiple scheduler instances for which load balancing is a priority are ideally suited to making use of this feature.

Background

The way in which Exago has historically handled report scheduling and the default behavior without using a queue, is the following.

For this discussion, it's important to define some terms:

- A **Schedule** represents all of the information that a Scheduler Service needs to execute a report. A Schedule contains information such as report name, time and frequency of execution, and how to handle the result. This information is usually stored as an XML file in a repository. Schedules can be accessed from the .NET API using the **ReportScheduler** class.
- Each Schedule contains some interpreted data that tells a Scheduler Service when to run it. This information is called a **Job**. Jobs can also be stored separately from Schedules. Jobs can be accessed from the .NET API using the **QueueApiJob** class.
- The process whereby a Scheduler Service runs a report at a specified time and emails or saves the information is called an **Execution**.

Within the host application, all Scheduler Service instances are listed in the configuration XML file:

When a Schedule is created in the Web Application, the host application sends the job to Scheduler Services starting with the first and moving through the list ("round-robin" style). The queried Scheduler Service stores the Schedule XML in a local working directory. This acts as a repository for the Scheduler Service's unique set of jobs.

From this point, each Scheduler Service works independently. The host application has no idea what happens to schedules after they are sent out successfully. Likewise, the Scheduler Services have no more communication with the host application with regard to report execution.

> **NOTE**
>
> Schedules can be viewed and edited from the Web Application using the
>
> **Schedule Manager**, but this is essentially a combined front-end for the Scheduler Service's existing files. If a Scheduler Service is offline, its schedules do not appear in the list (there will be a warning message). The Schedule Manager has no impact on the host application.

Scheduler Services periodically scan their repository for job execute times. If a job is ready and the current time is equal to or past the execute time, the Scheduler Service knows to run the job. The Scheduler Service will perform its duty and then alter the schedule XML file to indicate success or failure and the next execute time.

This default behavior may be adequate for most cases, but there can be issues. In particular, the Scheduler Queue sets out to solve the following two issues that can arise in default configurations: **Load Balancing** and **Unexpected Outages**.

Load Balancing

Ideally, unoccupied Scheduler Services would receive new jobs. This way, stacks of unexecuted data do not build up on individual Scheduler Services, leading to unbalanced load and potential time loss. But the host application has no idea which schedulers will be busy when, and no idea how long jobs will take to run. The randomness of round-robin job assignment could cause jobs to build up inordinately on one Scheduler Service.

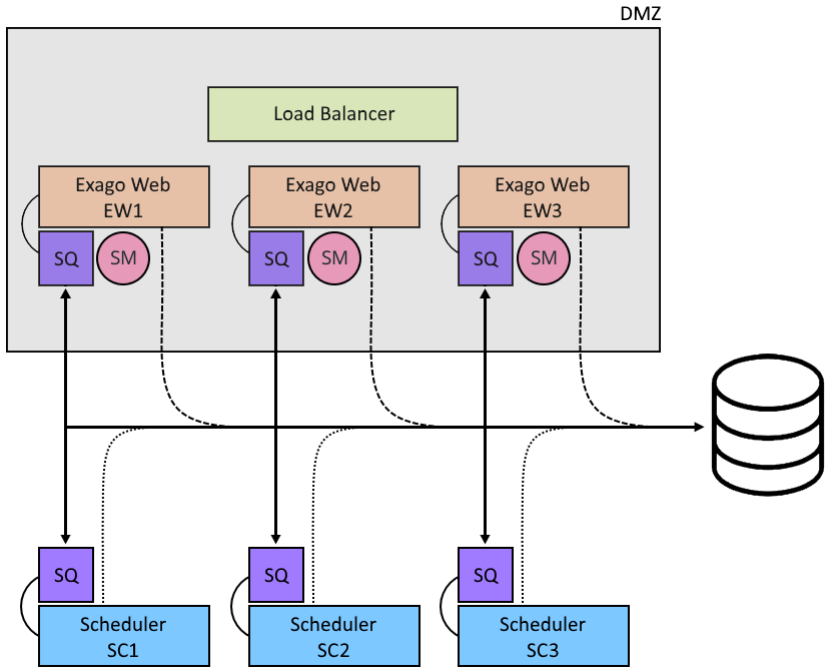
Outages

Once the host application sends out a schedule, as far as it's concerned, it's finished. If a Scheduler Service goes offline unexpectedly the host has no recovery function. The job will simply be delayed until the Scheduler Service is restarted. There is also no function to move schedules from one Scheduler Service to another.

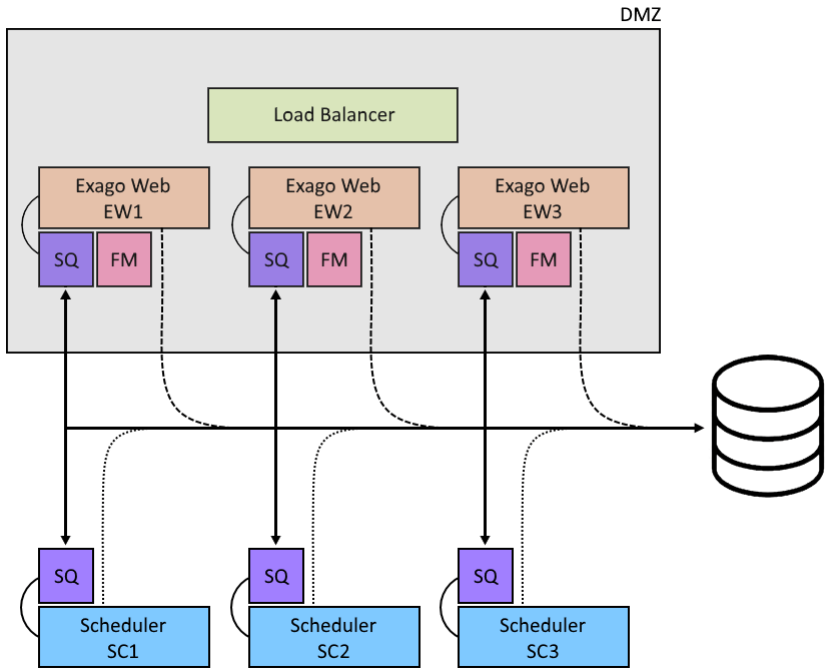
How the Queue Works

The **Scheduler Queue** is a custom .NET or Web Service library which aims to handle scheduling in a more robust manner. It's important to note that the queue is entirely customizable. You are only required to implement the applicable methods; how you do so is up to you. The following section will describe a typical setup which can improve load balancing and help resolve some common issues with multiple Scheduler Services.

The queue sits in between the Exago host application and any number of Scheduler Services and handles logic for all scheduler requests and maintenance.



Architecture Diagram *v2020.1+*



Architecture Diagram

The host applications and Scheduler Services all make calls to the queue at certain points during their runtime. In particular, Scheduler Services will call the queue on three occasions: upon service **startup**, **periodically** while running, and when a job's status is **changed**. The host application calls the queue for various maintenance tasks related to schedule creation and populating the

Schedule Manager. For now, we'll focus on the relationship of Scheduler Services to the Scheduler Queue and how it can aid a typical multi-scheduler configuration.

When Scheduler Services are configured to use the queue, their behavior changes somewhat.

> **NOTE**
>
> In the default configuration, Scheduler Services store their unique schedules in a local working directory, from which jobs are queried for execution.

Scheduler Services now periodically query the queue, which has instructions (**GetNextExecuteJob**) for assigning jobs. (The query time defaults to 15 seconds, but is configurable). In a typical setup, the queue pulls from a central repository of stored schedules. In order to prevent duplication, Scheduler Services lock the queue so that only one may access it at a time. Additionally the Scheduler Queue sets a job's status to "running" while it's active, so that other Scheduler Services know to ignore it.

> **NOTE**
>
> Scheduler Services still use a local working directory for temporary files.

This has several advantages. First, Scheduler Services are no longer responsible for a unique set of schedules. This prevents outages from causing excessive missed executes. Only one job will ever be hung per Scheduler Service as it will be responsible for only one job at a time. If a Scheduler Service goes offline in the middle of a job, the Scheduler Queue can be used to gracefully handle incomplete jobs.

Next, jobs are now distributed much more evenly between the Scheduler Services. Jobs will only be assigned to available Scheduler Services. Finally, since this allows us to control what data is being sent and received to the Scheduler Services and the file system, any custom load balancing solution can be implemented.

Set Up

Setting up the Scheduler Queue is a multi-part process which depends on the desired configuration. We'll discuss some constants and some potential variations.

Write the Scheduler Queue

First we need to write the Scheduler Queue. This can be a .NET Assembly or a Web Service, and it can be part of another library.

All the following methods must be implemented in the Scheduler Queue interface:

```
public static string[] GetJobList(string viewLevel, string companyId, string userId)
```

Called from the Exago Web Application to populate the jobs in the

Schedule Manager.

```
public static string GetJobData(string jobId)
```

Called from the Exago Web Application

Schedule Manager to get the full job XML data.

```
public static void DeleteReport(string reportId)
```

Called from the Exago Web Application when a report is deleted.

```
public static void RenameReport(string reportId, string reportName)
```

Called from the Exago Web Application when a report is renamed.

```
public static void UpdateReport(string reportId, string reportXml)
```

Called from the Exago Web Application when a report is updated.

```
public static void Flush(string viewLevel, string companyId, string userId)
```

Called from the Exago Web Application

Schedule Manager in response to a click on the **Flush** icon.

```
public static void Start(string serviceName)
```

Called from Scheduler Services to indicate when a specific service starts.

```
public static string GetNextExecuteJob(string serviceName)
```

Called from the Scheduler Services to return the next job to execute.

```
public static void SaveJob(string jobXml)
```

Called from both the Scheduler Service and the Exago Web Application to save the job. This method is called when a schedule is added, updated, completed, killed, etc.

The **QueueApi** and **QueueApiJob** helper classes have been added to the API to facilitate writing the Scheduler Queue. You'll need to reference the **WebReports.Api.Scheduler** namespace. **QueueApiJob** wraps a **Job** object and a variety of useful methods for managing jobs. The **QueueApiJob** class will be used extensively in the following example.

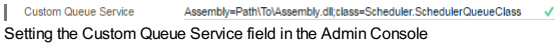
Configuration

The host application config and each Scheduler Service config must contain the path to the Scheduler Queue assembly or Web Service class in the following format:

```
Assembly=PathToAssembly.dll;class=Namespace.Class
```

The path may be set by one of these methods:

- Using the **Admin Console** by setting the **General** > **Scheduler Settings** > **Custom Queue Service** field.



Setting the Custom Queue Service field in the Admin Console

- By setting the field **<schedulerqueueservice>** in the config file,
- By setting the field **Api.SetupData.General.SchedulerQueueService** via the .NET API at runtime.

In each Scheduler Service, set the field **<queue_service>** in the config file.

Next, determine how schedules are accessed. A common solution uses a database to optimize lookup speed. The queue only needs to know the Job ID (filename), Next Execute Time, and the Running status to determine which schedules to run.

Job ID	Next Execute Time	Running?
String	DateTime	Boolean

User Identification

Exago does not have native user authentication. User logins must be handled by a security layer in the embedding host application. After a user logs in, the host application should pass identification parametersÂ to the Exago API, which can be used to set permissions on content and data objects within Exago.

Storage Management Identity Keys

Review the Storage Management: Introduction article's Identity Keys section to learn about how these keys affect the application. These keys should be set along with the System Parameters each time an Exago session is created via the API.

Setting the Identity Keys

> **IMPORTANT**
>
> The identity key values are case-sensitive.

.NET API

Set the **User Id**:

```
api.SetupData.StorageMgmtConfig.SetIdentity(key,value)
```

Replace **key** with the name of one of the identity keys, either:

- userId
- companyId
- classId
- ownerId

REST Web Service API

Use the /rest/StorageMgmt endpoint to PATCH identity key values to the session. For example:

```
"Identities": {
   "userId": "aboy",
   "classId": "user",
   "companyId": "Exago, Inc."
}
```

Storage Management also allows for creating custom identity keys by modifying the XML configuration file and party type table. Any custom identity keys may also be set with these calls.

Reading Values of the Identity Keys

The embedding host application may want to read the values stored for each of the identity keys.

.NET API

The following call will return the corresponding value:

```
api.SetupData.StorageMgmtConfig.Identity(key);
```

Replace **key** with the name of one of the identity keys, either:

- userId
- companyId
- classId
- ownerId

REST Web Service API

Use the /rest/StorageMgmt endpoint to GET a complete Storage Management Configuration JSON object, then examine the object's Identities property. More information can be found in the REST â€" Storage Management article.

System Parameters

Exago has two built-in parameters which are used to store identifying information: **userId** and **companyId**. These parameters are used in conjunction with the **Scheduler Service** and **User Preferences**, and they are automatically passed to any extensions which may need to access authentication. **Scheduler Queue**, and any extension which can access **sessionInfo** (such as **Assembly Data Sources** or **Server Events**) can retrieve these parameters in relevant methods.

> **NOTE**
>
> These are required parameters. They cannot be deleted through theÂ Admin Console, and if they are not present in the Config, then they are automatically added at load time.

Often **userId** corresponds with a unique user or log-in, and **companyIdÂ** represents a group of users with shared characteristics.Â Either parameter can be used without the other.

userEmail v2018.2+

The **userEmail** parameter is a third built-in parameter that can specify an email to be set as the "Reply To" email for any scheduled reports. If using the **userId** or **companyId** parameters to identify the user, it is possible to change the default value of **userEmail** to be specific to the users identified by **userId** or **companyId**. See below for an example of this in the .NET API and REST API.

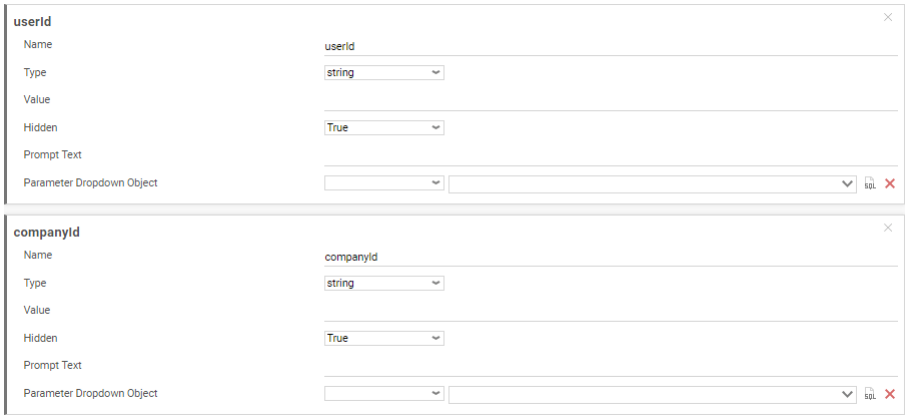
Setting theÂ Current User

In *v2017.2+*, the Id parameters are added by default in the base config file with blank values.

Prior to *v2017.2*, the Id parameters are not instantiated by default, and must be created before use. They can be created in the **Admin Console**, config file, or in the API code. They must be created with the exact names of "userId" and "companyId" (which are case sensitive), with data type "string". Since the values are set in the API, if you create the parameters in the **Admin Console** or config, they should have blank default values.

Admin Console

AsÂ created in the **Admin Console**:



Config File

AsÂ created in the config file:

```
<parameter>
 <id>userId</id>
 <data_type>string</data_type>
 <value />
 <hidden>True</hidden>
 <prompt_text />
</parameter>
<parameter>
 <id>companyId</id>
 <data_type>string</data_type>
 <value />
 <hidden>True</hidden>
 <prompt_text />
</parameter>
```

The **userEmail** parameter (*v2018.2+*):

```
<parameter>
<id>userEmail</id>
<data_type>string</data_type>
<value />
<hidden>True</hidden>
<prompt_text />
</parameter>
```

.NET API

AsÂ created in the .NET API:

> **NOTE**
>
> “DataType” defaults to DataType.String, so the call is omitted.

```
api.Parameters.GetItem("userId").Value = "myUserName";
api.Parameters.GetItem("companyId").Value = "myCompanyName";
```

For setting the **userEmail** parameter (*v2018.2+*):

```
Parameter userId = api.Parameters.GetParameterById("userEmail");
userId.Value = "[email protected]_17.com";
```

REST Web Service API

As created in the REST Web Service API:

> **NOTE**
>
> “DataType” defaults to “String”, so the call is omitted.

POST /rest/Parameters with this payload:

```
{
  "Id": "userId",
  "Value": "user_224"
  ...
}
```

For setting the **userEmail** parameter (*v2018.2+*):
POST /rest/Parameters with this payload:

```
{
  "Id": "userEmail",
  "Value": "[email protected]_17.com"
  ...
}
```

Basic Sandboxing

Setting the **userId** and **companyId** parameters has several effects in the user interface.

Schedule Manager

By default,Â theÂ

Schedule Manager will show only schedules belonging to the current **userId**. This can be changed by changing the **Admin Console** > **General** > **Scheduler Settings** > **Schedule Manager User View Level**:

- Current User (‘*User*’): Filters schedules by current **userId** parameter.
- All Users in Current Company (‘*Company*’): Filters schedules by current **companyId** parameter.
- All Users in All Companies (‘*All*’): No filtering.

This setting can also be overridden by a Role.

Execution Cache

The **userId** and **companyId** parameters are used to set permissions for which users can view cached report data. Users can choose whether a report cache is visible just for their **userId**, for everyone with the same **companyId**, or for all users. The options that are available to a user depends on the **Admin Console** > **General** > **Scheduler Settings** > **User Cache Visibility Level** setting.

User Preferences

User preferences, including which reports should run on startup and **User Reports** (live report customization), are set by **userId**, and will only apply to that user.

Advanced Permissions

userId and **companyId** can be used in many other application areas in order to handle security permissions.

Roles

Additional permissions are typically handled byÂ **Roles**. A check can be made in the API which maps the current **userId** and/or **companyId** to the role which it belongs. This must be handled manually via a lookup table or a similar data structure. Then activate the role for the session.
.NET:

```
api.Roles.GetRole("admin").Activate();
```

REST:

```
PATCH /REST/Roles/admin?sid={sid} { "IsActive": true }
```

For more information, see **Roles**.

Tenanting

userId and **companyIdÂ** can be used as tenant parameters in your data objects.

If your data is set up such that each table, view and stored procedure has columns that indicate which user has access to each row, then you can use **userId** and/or **companyId** to match these columns and act as data row filters. (For this purpose, the parameters cannot be set to 'hidden').

For more information, see **Multi-Tenant Environment Integration**.

Accessing Ids in Extensions

userId and **companyId** are passed to any custom extensions where relevant. For example, in an external interface assembly, you may wish to access the **userId** in order to log user executions. You could do so by implementing the *ReportExecuteStart* method, which passes the **userId** parameter.

```
public static void ReportExecuteStart(string companyId, string userId, string reportName)
{
  string logText = string.Format("{0}: Report '{1}' executed by user '{2}'.", DateTime.Now, reportName, userId);
File.AppendAllText(logFile, logText + Environment.NewLine);
}
```

This would return the following text upon a reportÂ execution by **userId** "Alex":

```
2017-03-07 14:50:49: Report 'TestProduct Sales Report' executed by user 'Alex'.
```

Most extensions have methods which can access **userId** and **companyId**. In addition, theÂ parameters are accessible through sessionInfo.Â So anyÂ extensions which can access sessionInfo can also access **userId** and **companyId,** even if methods do not explicitly implement them.

Setting up a State Server

Exago highly recommends using a state server to manage user sessions. Often, the cause of timeout problems is related to not properly managing session state. In a high availability implementation, a state server is required.

You can create a state service on the same server as the Exago application, or on a different one.

- ASP.NET State Service on Windows
- ASP.NET State Service on Linux
- Elasticache for Redis on AWS EC2

> **WARNING**
>
> When using any of the state server configurations below, make sure to set cookieless="false" as the use of cookieless sessions with a state server is not supported.

ASP.NET State Service on Windows

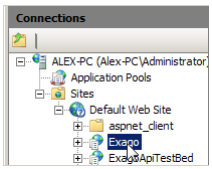
Enable the Service

1. On your state server, click **Start** and then **Run…**. Type *services.msc*, and click **OK**.
2. Locate the **ASP.NET State Service** and check the **Startup Type** property. If it is not set to *Automatic* or *Automatic (Delayed Start)* then you have to enable the service.
3. Right-click on it, and select **Properties**.
4. Change the **Startup Type** to *Automatic*, then click **Apply**. Then click **Start**.

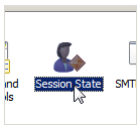
If your state server is on a network, make sure you have allowed inbound connections to the state service on a port.

Configure the Web Server

1. To configure the web server to use the state service, open IIS Manager, then in the left-most **Connections** pane, locate and select your Exago application.

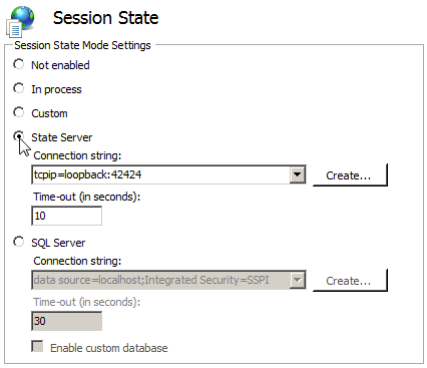


2. Double-click on **Session State**.

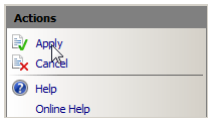


3. Select the *State Server* setting, and input the server port and a desired timeout value.

> **NOTE**
>
> Windows' built-in State Server runs on port 42424 by default. It is suggested that this port is used for the configuration of the State Server. However, if preferred, this may be changed.



4. Finally, in the right-most **Actions** pane, click **Apply**.



After applying these settings, the web server should be fully configured.

Additional Info

For more information about out-of-process session state, see the following external links:

- Configure a State Server to Maintain Session State (IIS 7.0)
- Configuring Step 2: Configure ASP.NET Settings

ASP.NET State Service on Linux

> **NOTE**
>
> This section references <WebApp>/ as a placeholder for the application's install location. The default install location is /opt/Exago/; however, this directory may be changed during installation.

A state service is installed as part of the mono runtime. Exago provides a script that can enable the state service to run headless, that can be downloaded from the Downloads page on the Support Site.

Prerequisites

Using an ASP.NET State Service on Linux requires:

- mono be installed on the state server
- the **expect** package be installed on the state server
- a firewall rule allowing access to port 42424 on the state server for all servers in the web farm. This port may be changed, see Choosing a Different State Service Port below for more information.

Setup

Perform these steps on the state server:

1. Download the **startStateServer** script with the link above. This script should be saved outside of the root or bin directory of the Exago installation (e.g. <WebApp>/StateService) in order to isolate it from the rest of the application.
2. Run the script as root. For example:

```
sudo /opt/Exago/StateService/startStateServer.sh
```

Perform these steps on each Exago application server in the web farm:

1. Open the <WebApp>/web.config file for editing.
2. Locate the <sessionState mode="InProc" cookieless="false" timeout="20" /> line and comment it out using XML comment tags <!– and –>.
3. Uncomment (or add if missing) the <sessionState mode="StateServer" cookieless="false" timeout="15" stateConnectionString="tcpip=localhost:42424" /> line just below the line in step 2.
4. Replace *localhost* in the **stateConnectionString** attribute with the IP address of the server running the state service.When finished, the web.config file should look like this:

```
<!–<sessionState mode="InProc" cookieless="false" timeout="20" /> –>
<sessionState mode="StateServer" cookieless="false" timeout="15" stateConnectionString="tcpip=0.0.0.0:42424" />
```

5. Restart the web server.

Choosing a Different State Service Port

To use a port other than 42424 for the state service:

1. Open the <WebApp>/web.config file for editing.
2. Modify the <sessionState mode="StateServer" cookieless="false" timeout="15" stateConnectionString="tcpip=localhost:42424" /> line with the correct port in the **stateConnectionString** attribute
3. Save and close <WebApp>/web.config.
4. Restart the web server.
5. Repeat steps 1â€“4 on each server in the web farm.
6. On the state sever, open /usr/lib/xsp/4.0/asp-state4.exe.config for editing.
7. Locate the <channel ref="tcp" port="42424" /> line inside of the <channels> node. Modify the **port** attribute to the same port specified in step 2 above.
8. Save and close /usr/lib/xsp/4.0/asp-state4.exe.config.
9. Restart the web server and state service.

Elasticache for Redis on AWS EC2

1. Create an Elasticache for Redis instance on AWS. The Elasticache for Redis instance must be in a security group that has port number 6379 open in order to communicate with Exago. Consult with the AWS and Redis documentation on how to do this.
2. On the Exago EC2 instance, install the Redis command line interface by issuing the following commands. The commands below are for Ubuntu and Debian type Linux distributions, modify them for other operating system variants.
    1. sudo apt-get install gcc
    2. wget http://download.redis.io/redis-stable.tar.gz
    3. tar xvzf redis-stable.tar.gz
    4. cd redis-stable
    5. make
    6. make test

    > **NOTE**
    >
    > Note that the make and make test steps will take several minutes to complete each.
    >
    > make test requires that Tcl be installed on the system. If not installed already, it can be installed with sudo apt-get install tcl.

3. Confirm that you can connect to Redis from the Exago EC2 instance by issuing the following command:
    1. src/redis-cli -c -h <redis-host-name> -p 6379
       Substitute <redis-host-name> for the actual host name of the Elasticache for Redis instance created for step 1. The Redis CLI prompt should appear. If the session store is not correctly setup, this operation will time out instead.
    2. Run a few test commands to insure that Redis is working. Exit the CLI with **Ctrl** + **C** on the keyboard.
4. On the Exago EC2 instance, modify the Web Application's web.config file. Locate the sessionState node and modify it as follows:

    ```
    <sessionState mode="Custom" customProvider="MySessionStateStore"> <providers>  <add name="MySessionStateStore" type="Microsoft.Web.Redis.RedisSessionStateProv
    ```

    Substitute <redis-host-name> for the actual host name of the Elasticache for Redis instance created for step 1.

5. On the Exago EC2 instance, download the NuGet Package Manager and the required packages by issuing the following commands:

    > **NOTE**
    >
    > This article references <WebApp>/, as a placeholder for the application's install location. The default install location is /opt/Exago/ however, this directory may have been changed during installation.

    1. cd <WebApp>/bin
    2. sudo wget https://dist.nuget.org/win-x86-commandline/latest/nuget.exe
    3. sudo mono nuget.exe install StackExchange.Redis.Mono.StrongName
    4. sudo mono nuget.exe install Microsoft.Web.RedisSessionStateProvider -Version 3.0.2
6. Verify that the following directories now exist in <WebApp>/bin:
    - Microsoft.AspNet.SessionState.SessionStateModule.1.1.0
    - Microsoft.Web.RedisSessionStateProvider.3.0.2
    - StackExchange.Redis.Mono.StrongName.1.2.0
7. Move the libraries from their directories to <WebApp>/bin by issuing the following commands:
    1. cd <WebApp>/bin/Microsoft.Web.RedisSessionStateProvider.3.0.2/lib/net452
    2. sudo mv Microsoft.Web.RedisSessionStateProvider.dll <WebApp>/bin
    3. cd <WebApp>/bin/StackExchange.Redis.Mono.StrongName.1.2.0/lib/net45

4. sudo mv StackExchange.Redis.StrongName.dll <WebApp>/bin
5. cd <WebApp>/bin/Microsoft.AspNet.SessionState.SessionStateModule.1.1.0/lib/Net462
6. sudo mv Microsoft.AspNet.SessionState.SessionStateModule.dll <WebApp>/bin

8. Load the **Admin Console** to verify Exago loads without errors and to create at least one session in the cache.
9. Check to make sure the cache is working by issuing the following Redis CLI command from the Exago EC2 instance.
   1. Re-establish a connection to the Redis CLI with the commands in step 3.1 above.
   2. KEYS '*'

Keys for Exago should appear in the listing if the Admin Console loaded in step 8, similar to the figure below.



Redis Resources

- Redis Quick Start
- Redis Command Refernce

Deploying to Production

This guide describes the considerations you should take when deploying an Exago BI installation to a production environment. Suggested steps are listed in the order they should be taken.Â Best practices are recommended for each step. However, every environment is different, so recommendations should be considered in the context of your desired setup.

For personalized support, please file a **support ticket**.

Contents

1. **Installation**: Decide where the Exago BI application and schedulers live
2. **Data**: DetermineÂ how to expose your data to your users
3. **API**: Use the API to control user permissions
4. **Content Management**: Implement a content management solution
5. **Integration**: Visually integrate Exago BI into your host app
6. **Reports**: Make “canned” reportsÂ as examples for your users
7. **User Preferences**: Implement a user preference storage solution
8. **Deployment**: Important steps to follow beforeÂ deploying yourÂ application
9. **Security**: Follow our **Security Checklist** of best practices

Installation

Since Exago BI is an embedded application, it is up to you to decide which of your servers Exago is installed on. Exago BI supports nearly any type of deployment, including cloud, private servers or onsite at your clients.

Exago does not need to be installed on the same server as the host application. One or more scheduling services can be further installed separate servers toÂ handle **Remote Execution** of reports. Additional scaling can be accomplished with a Web Farm. To use the .NET API, Exago BIÂ must be accessible from the host application via a file system path. See **API** for more information.

The scheduling services are capable of acting as standalone report execution applications. The best way to scale Exago BI for performance is to deploy additional scheduling servers, and offload report executions to them.Â This method, called **Remote Execution**, also implements an automatic load balancing solution. The servers with the most available resources are given execution priority in order to keep an even load distribution.

A QA/Staging environment is highly recommended as well. This allows developers to test API changes, config changes, and Exago BI version updates, before moving to production.

Also consider the following:

- Where do you want temp files and report definitions to reside for each server?
- Where does the data source reside, relative to where Exago BI will be deployed?
- Should Exago BI reside in the same domain as your host application?

Data

It is critical to make the right choices in how to present your data to your users. Consider the technical level and reporting experience of your users. You may need to service different classes of users, from technical users like developers and database administrators, to non-technical business analysts and project managers.

Exago BI can manage **data object permissions** to many levels of precision.Â Permissions can be setÂ per-user andÂ per-group, for data objects, fields, rows, and even within field values.Â Data objects are easily restricted for classes of users using **Roles**. Within a data object, fields can be hidden, and row access can be limited by matching a linked user ID with the logged-in user. Roles can also provide filters for data values, in case some fields should be partially, but not fully hidden from view.

It is also recommended to use **aliasing, descriptions, categories, and metadata** to control how your data objects appear in the application. Categories are a way to separate data objects into folders, so you can group together associated data. Aliasing allows you to showÂ more user-friendly names for data objects, instead ofÂ how they are named in the database. Descriptions provide additional information, and metadata can improve report performance.

Also consider the following:

- How normalized or de-normalizedÂ should the visible data be?
- Should any data objectsÂ be available for some users and not for others?
- Within a data object are there row-level permissions or multi-tenant permissions per user?

- Will the underlying data objects change in the future? If so, add IDsÂ toÂ objects to preventÂ naming conflicts.

API

It is critical that Exago BI is only exposed to users through the API. The API allows you to set security and permissions settings, and tailor the reporting experience by user.

.NET environments can use the **.NET API**, which is the most flexible and extensible API. Non-.NET environments can access a subset of API calls through the **REST** Web Service API.

The most basic API implementation begins by initializing a session, then sets the **"userId" and "companyId"** parameters to identify the logged-in user, setsÂ a user-specific **Role** to control permissions, and then launches Exago BI using the **getUrlParamString()** method. Developers should write the API code robustly with checks for null return values and exception handling.

Consider where you want Exago BI to appear within your application, and how users should use it to access reporting. For example, should it be placed in an iFrame container, a redirected page, or a popup window?

Also consider the following:

- Do you want to provide a list of reportsÂ and dashboards to users and directly run them via the API?
- Are there other settings in Exago BI you want to enable or disable based on the user?

Content Management

In *v2020.1+* all reports, folders, themes and templates are stored in a database in the **Storage Management** system. Read more in the Storage Management: Introduction article.

There are many strategies to Storage Management during the evaluation, development and QA stages. As such, there are a few paths to promoting reports from those environments to production. This section will cover the most common use cases clients have encountered.

Cherry-Picking Reports

If only a small number of reports will transition from development to production, the built-in **Download** and **Upload** functions can be used from the Report Tree in either environment.

A Single Database for All Environments

If all environments are sharing the same Storage Management database, this is the simplest migration path. In production, simply set the **Admin Console** > **Storage Management** database settings to the same as those from the development environment.

Unique Databases for Each Environment

If development and production have separate Storage Management databases, use the **ImportExportStorageMgmt** command line utility to easily export one database and import into another.

Then, using one of the APIs it is possible to set the Storage Management connection string programmatically on a per-session basis.

Integration

Exago BI gives you full control over the **CSS, icons**, and **language strings** in the UI. You can have several different **application themes**Â if necessary, and select different ones for different classes of users. You can also build customÂ themes for reports and visualizations. And you can make a **custom start page** that users will see when they first enter the UI.

For integration, we recommend the following best practices:

Home page

Copy the ExagoHome.aspx home page to a new page, and use that as the entry point for users going into the full UI. This copy can be styled at will.

Application Theme

Make a new application theme by extracting a copy of our theme template into a new folder in the ApplicationThemes directory. Enable the theme in the Admin Console, and add any **custom styling** if desired.

Getting Started

Remove the content from the default **Getting Started** page, since thisÂ is only intendedÂ to be a styling example. It is recommended to add custom content, since users with access to the full UI will see this page often. Clients have usedÂ the Getting StartedÂ page toÂ provide announcements, quick tips, helpful formulas, and links to other parts of the application.

Also consider the following:

- Does your application support multiple languages?
- Are there any text strings or tooltips you want to customize?
- Do you want to customize the CSS or swap out any of the icons in Exago BI?
- Do you want to customize the context-sensitive help to match your documentation?

Reports

By providing a folder of "canned"Â reports, you can show users Exago BI's capabilities, and some useful examples of the data that is available to them. This folder should be at the top level so it is easy to locate, and read-only so that users cannot edit the reports. If they want to make changes or see how the reports are made, they can duplicate a reportÂ and edit the copy.

Making these reports can be a good opportunity for your support, sales, and services staff to become familiar with Exago BI. The Exago BI services team can also assist your staff, either by basing your training on these reports, or by building them for you.

Also consider the following:

- What questions can be answered by reporting?
- What data objects do you want to highlight?
- Are there any specific Exago BI features that you want to showcase?

User Preferences

The **User Preferences** store user-level customizations of the application environment in the user's browser local storage by default, but it is recommended to store these in a database via extensibility. An example implementation for *v2020.1+* can be found on the Downloads page of the Support Center.

For more information:

- Admin Console: User Settings
- Admin Support Lab â€" Customizing Color Picker (video)
- Admin Support Lab â€" v2018.2 User Preferences (video)

Deployment

Once the other steps are in place, lay out a plan for moving to production. We recommend keeping detailed notes of the process so it can be replicated for future updates.

After installing in production, move the following files from your staging environment to production:

1. The encrypted configuration file: /Config/*config*.xml.enc
2. Any custom application themes folders: /ApplicationThemes/*themeFolder*
3. Any custom language files: /Config/Languages/*language*.xml. These also need to be added to the /Languages directories for each scheduler installation.
4. Your custom Getting Started page: /Config/Languages/*getting-started*.xml
5. Any other configuration files: /Config/Other/*file*.json, and /appSettings.config
6. Your custom context-sensitive help, if you have one: /NetHelp
7. Your custom home page: /*home*.aspx
8. If you are not using folder management, any custom theme files: /Themes/*theme*
9. If you are using Google Maps, make sure the MapPolygonDataBase.sqlite file is present in the /MapCache folder.

We also recommend adding the non-encrypted config file (/Config/*config*.xml) to your version control after removing any sensitive passwords or connection strings.

Disable the Admin Console. It should not be accessible in a production environment.

Also consider the following:

- Make sure all Exago BI instances and Scheduler Services are on the same versionÂ and buildÂ number.
- If you are using the .NETÂ API make sure the version ofÂ WebReportsApi.dll matches the version and build number of Exago BI. Do the same with any custom assemblies, such as Storage Management, Scheduler Queue, or Server and Action Events.

Security

IMPORTANT:Â FollowÂ our **Security Checklist** beforeÂ turning on access to your application. It is highly recommended to follow these steps to reduce the possibility of unauthorized access.

Transition from HTTP to HTTPS

Many times during the evaluation process, Exago may be setup to access resources over HTTP. As the evaluation progresses, or when transitioning from evaluation to production, it may be desirable to secure the connections between Exago and these resources. This article serves as a checklist of all locations in the product where an HTTP or HTTPS resource may be provided, making it easy to transition from one to the other.

Web Application

- Data Sources (**Admin Console** > **Data** > **Sources**) that use web services
- **Admin Console** > **General Settings** > **Main Settings** > **Temp URL**

Scheduler Service

- The following Scheduler Settings (**Admin Console** > **General** > **Scheduler Settings**):
  - Schedule Remoting Host
  - Remote Execution Remoting Host
  - Custom Queue Service
- The following nodes in the Scheduler Service's configuration file:
  - <webappbaseurl>

REST Web Service API

- The <webreportsbaseurl> node of the WebReportsApi.xml configuration file.

Host Application

The **host application** is the application that embeds Exago into it.

- Any REST Web Service API calls coming from the host application will need to be changed to HTTPS.
- References to the JavaScript API library.

Cross-Origin Embedding of Exago

When a user enters Exago BI, the application creates a session in order to preserve the user's working state and environment while they engage with it. The session information is stored on the web application server or state server (depending on the network configuration).

Whenever requests are made the user's browser sends a unique, randomly generated ID to the web server in order to authenticate the user and preserve the connection between user and session.

By default, this session ID is stored as a browser cookie. However, when embedding Exago BI in a cross-origin environment (where the Exago server and host application are referred to by different protocols, domains or ports) cookies are considered a potential attack vector for cross-site scripting. Most modern web browsers prevent embedded content from separate origins from using cookies without additional configuration to explicitly allow this behavior safely.

Consider these three scenarios for embedding Exago in a host application:

> **WARNING**
>
> In the cases below and a state server is in use, make sure to set cookieless="false" as the use of cookieless sessions with a state server is not supported.

- Exago and Host Application are referenced by the same origin (not covered by this article)
- Exago and Host Application are referenced by different origins, using Cross-Origin Resource Sharing with cookies (recommended)
- Exago and Host Application are referenced by different origins, using cookieless sessions (not recommended)

Cross-Origin Resource Sharing (CORS) with Cookies

> **IMPORTANT**
>
> When using cookies, Exago must be accessed via **HTTPS** in a CORS environment.

If the Exago BI application is being accessed over a network, the Exago web server must be configured to allow **cross-origin requests** from the embedding host application.

By default, a web application making a request for content that originates from a remote server sends a cross-origin HTTP request. For security reasons, browsers restrict these types of requests when they originate from scripts.

> **IMPORTANT**
>
> When using the JavaScript API with versions prior to *v2019.1.23*, *v2019.2.9* and *v2020.1.0* in a CORS environment, cookieless sessions must be used.

To permit content to be accessed by whitelisted resources, you can configure the Exago web server to send aÂ *Access-Control-Allow-Origin* and *Access-Control-Allow-Credentials* headers to the calling server.

v2019.2.29+, v2020.1.13+, v2021.1.1+

As of the version numbers listed above, two new Application Settings elements, **AccessControlAllowCredentials** and **CorsAllowedOrigins** have been added to the Web Application's appSettings.config file. These new nodes are the only supported way of adding *Access-Control-Allow-Origin* and *Access-Control-Allow-Credentials* for new installations.

Setting the headers in the web server's configuration is no longer recommended. However, that method will continue to function for existing clients upgrading from previous versions.

Review the Web Application appSettings.config section of the Application Settings article for full details on setting these headers.

pre-v2019.2.29, v2020.1.13, v2021.1.1

Access-Control-Allow-Origin should be set to a single domain that matches the domain of the page in which Exago is embedded. For example:

```
Access-Control-Allow-Origin: hostapp.server.com
Access-Control-Allow-Credentials: true
```

In **IIS**, this is either set in the **HTTP Response Headers** or added manually to the web.config file.

In **Apache**, the following lines should be added to all relevant .conf files:

```
Header set Access-Control-Allow-Origin "hostapp.server.com"
Header set Access-Control-Allow-Credentials: "true"
```

In **NGINX**, the following line should be added inside the location /Exago/ {…} section of the "exago" NGINX site configuration file:

```
add_header 'Access-Control-Allow-Origin' 'hostapp.server.com';
add_header 'Access-Control-Allow-Credentials' 'true'
```

Cookieless Sessions

> **IMPORTANT**

As of Exago *v2019.1.23+*, *v2019.2.9+* and *v2020.1.0+* the use of cookieless sessions is not recommended when Exago is hosted over a secure connection (HTTPS). Use Cross-Origin Resource Sharing (CORS) with Cookies instead.

Enable Cookieless Sessions

NOTE

This article references <WebApp>as a placeholder for the application's install location. The default install locations are /opt/Exago/ on Linux and C:Program FilesExagoExagoWeb on Windows; however, these directories may be changed during installation.

To enable cookieless sessions, locate the following line in the <WebApp>web.config file and set the cookieless attribute to *true*:

```
<sessionState mode="InProc" cookieless="true" timeout="20" />
```

Cookieless sessions work to preserve state by appending the session ID to the URL string itself, which is used to connect to the web server. For example, a standard application URL generated by the server API after creating a session looks similar to the following:

```
//ExagoBI/ExagoHome.aspx?sn=0
```

With cookieless sessions enabled, the session ID is added to the URL, for example:

```
//ExagoBI/(S(rim43x0e3cz2hf4sjw24ezdk))/ExagoHome.aspx?sn=0
```

This has no direct implications for the embedding process into the host application.

IMPORTANT

In Linux environments, a patched System.Web.dll Mono library is necessary when using cookieless sessions. This patched DLL is installed when the Exago Linux Installer is executed, or it can be installed using our patch script. Exago only supports specific versions of Mono, review Installing Exago on Linux for details.

Secured Authentication

When the web application server is exposed to the internet or any insecure public network or intranet, you need to enable secured authentication to prevent unauthorized access.

Because the session ID is passed as a part of the URL, it can be used by malicious third parties to send unauthorized requests to the web server. Secured authentication causes the web server to require a secondary security token in order to authenticate the session. This does not require any additional effort on the part of the end-user.

To enable secured authentication, edit the <WebApp>appSettings.config file and add the following line:

```
<add key="useSecurityToken" value="True" />
```

The server API generates a security token with the session and embeds it into the page markup. It is sent with every client request as a part of the payload. When using a secured connection, the payload is encrypted, thus making it nearly impossible for a malicious party to derive the token.

CAUTION

Unsecured connections will compromise any security intended by the use of secured authentication. If the payload is unencrypted, the token can be easily intercepted by a third party. The web server must enforce HTTPS in order for secured authentication to function properly.

Additional Resources

Review these third-party resources for additional background information on the topics presented in this article.

- Mozilla Developer Network: Cross-Origin Resource Sharing (CORS)
- IBM: Prevent a cross-site scripting attack

Security Checklist

There are a number of precautions that should be taken before running Exago in a production environment.

- Set an external temp path
- Disable direct access
- Set configuration and API credentials
- Remove the XML configuration file
- Remove the Admin Console
- Encrypt Scheduler Service working data
- Disable SOAP Web Service API
- Content Security Policy HTTP Headers

This article mentions several Admin Console settings. There are matching configuration file nodes and API properties that can be used for modifying these settings as well. For more information, review the Config File XML Reference and Config File and API Setting Reference articles.

NOTE

This article references <WebApp>/, <WebSvc>/ and <Sched>/as a placeholder for the Web Application, Web Service API and Scheduler Service's install location respectively. The default install location is C:\Program Files\Exago\ExagoWeb (/opt/Exago/ on Linux), C:Program Files\Exago\ExagoWebApi (/opt/Exago/WebServiceApi/ on Linux) or C:\Program Files\Exago\ExagoScheduler (/opt/Exago/Scheduler/ on Linux); however, these directories can be changed during installation.

Set an External Temp Path

The Temp directory contains working data that may contain sensitive information. If the **Temp Path** configuration setting is left blank, Exago will default to a Temp folder at the root of the installation directory. This is not recommended because it could expose temporary data to web access.

The Temp Path should be set to a location outside of the Exago installation (and behind the server's firewall). Set the Temp Path in **Admin Console** > **General** > **Main Settings** > **Temp Path**.

Disable Direct Access

Access to Exago should be curated through the API so that user permissions can be set via Roles or the Storage Management identity system. Users should not be able to access the home page directly, which would bypass role restrictions. To disable direct access to Exago, set the **Admin Console** > **General** > **Main Settings** > **Allow direct access to Exago (bypassing API)** to *False*.

Set Configuration and API Credentials

A User ID, Password, and REST Key should be set in the configuration. This safeguards access to the Admin Console and REST API. See REST API for information on accessing a password-protected web service. To do so, set values for the following settings:

- **Admin Console** > **General** > **Other Settings** > **User Id**
- **Admin Console** > **General** > **Other Settings** > **Password**
- **Admin Console** > **General** > **Other Settings** > **Confirm Password**
- **Admin Console** > **General** > **Other Settings** > **REST Key**

Remove the XML Configuration File

The Admin Console generates two copies of the base configuration: a plain-text XML file, typically WebReports.xml and an encrypted version WebReports.xml.enc. Plain-text config files may contain sensitive information, such as database connection strings, schemas, usernames, and passwords.

When your config settings have been finalized, the plain-text config file should be removed from the Config folder and saved in a secure location.

Remove the Admin Console

The Admin Console should not be accessible in a production environment. To permanently remove the Admin Console, in each installation:

1. Delete <WebApp>\Bin\admin.aspx.cdcab7d2.compiled

TIP

Admin.aspx can be deleted as well, or edit it to show a static error message.

Encrypt Scheduler Service Working Data

NOTE

This section applies only in environments where at least one **Scheduler Service** is in use.

Each Scheduler Service stores working data in a local temporary folder. For security, this data should be encrypted. In each Scheduler Service's configuration file:

1. Set the <encrypt_schedule_files> setting to *True*. For example:

```
<encrypt_schedule_files>True</encrypt_schedule_files>
```

2. Restart the Scheduler Service

Disable SOAP Web Service API

NOTE

This section applies only in environments where the REST Web Service API is in use. If the .NET API is being used instead, the REST Web Service may not be installed. It may be uninstalled if it will not be used.

If you are using the REST Web Service API, disable the SOAP API to prevent any unauthorized requests. To do so, in each instance of the Web Service API:

1. Delete <WebSvc>\Bin\api.asmx.cdcab7d2.compiled

TIP

Api.aspx can be deleted as well, or edit it to show a static error message.

Set Additional Response Header Flags When Using Content Security Policy

When using Content Security Policy (CSP) rules with the web server, set these additional HTTP Response Headers to insure compatibility with Exago BI:

```
default-src 'self'; style-src 'self' 'unsafe-inline'; script-src 'self' 'unsafe-inline' 'unsafe-eval';
```

Instructions to configure the web server with these headers can be found at the links below:

- IIS
- Apache
- NGINX

Multi-Tenant Environment Integration

Exago supports a variety of approaches to make sure that users can only access the data that is assigned to them. These approaches can eliminate the need to create different reports for each user. This can be done in one of four ways. Using either column, schema, database, or custom SQL based tenancy.

Column-Based Tenancy

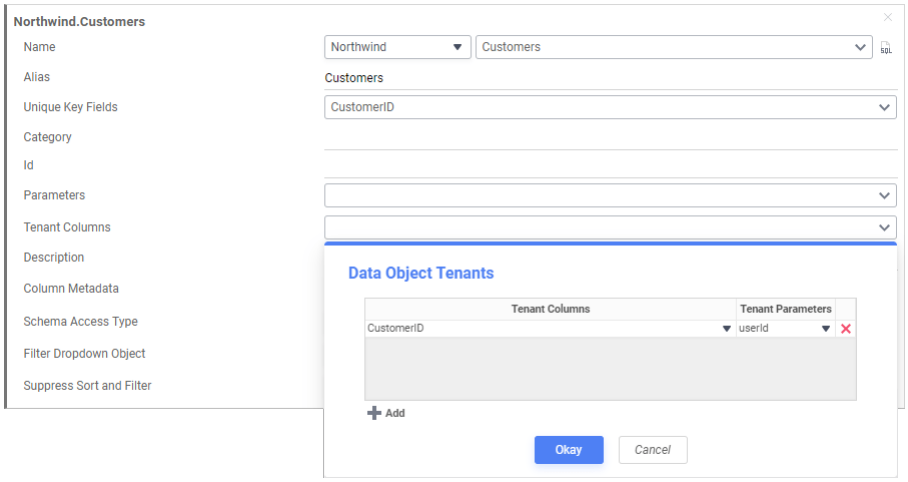
The most basic multi-tenant environment is when each table, view and stored procedure has one or more columns that indicate which users have access to each row.

To set column based tenancy in Exago:

1. Create a **Parameter** for each tenant column.

   > **NOTE**
   >
   > For these parameters set **Hidden** to *False*.

2. For each **Data Object** click the **Tenant Columns** dropdown. Use the **Tenant Columns** menu to match each tenant column in the data object with its corresponding parameter.
3. When initializing Exago through the **API**, set the value of each tenant parameter for the current user.



Disabling Column Tenancy

While disabling tenancy on any tenant columns of a data object is a noted security risk, it proves useful for administrators to be able to briefly disable tenancy for testing purposes. Disabling tenancy simplifies testing by allowing administrators to troubleshoot the dataset without having to remove and re-add tenant columns on each data object.

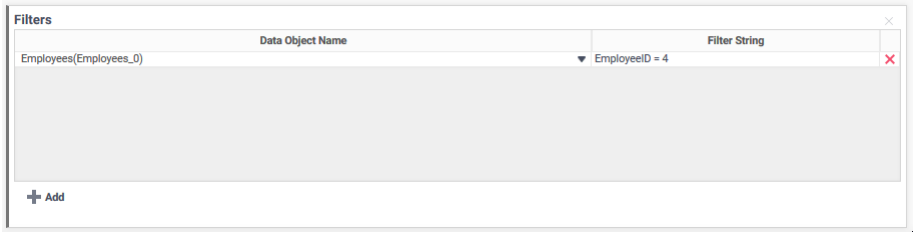
> **IMPORTANT**
>
> Tenancy should be disabled for testing purposes only. Do not allow non-administrative users to access data objects with disabled tenancy columns. Doing so can potentially expose sensitive data.

Following changes to tenant parameter values in *v2019.1+*, any **Tenant Parameters** values set to an empty value will be recognized as an empty string rather than a null value. To continue disabling tenancy in these versions, the parameter value must be set to {disabletenancy}.

Row-Based Tenancy

With Roles, security filters can be added to data objects so that users can only view specific rows in the data object.

In the example below, this role will only be able to see data on reports where the EmployeeID field is equal to 4.



Row Based Filter dialog in Admin Console Role configuration tab

To set row-based tenancy in Exago via the Admin Console, visit the Filters Access section of the Roles article.

Row based filters can also be added with the REST Web Service API and .NET API.

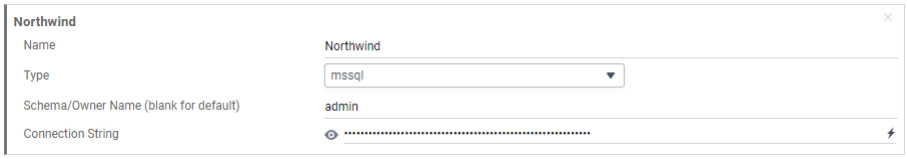
To delete a row based filter, click the **Delete** ✕ icon at the end of the associated row.

Schema-Based Tenancy

Some multi-tenant environments create multiple tables/views/stored procedures with the same name and columns but different database schema. Information is then stored in the appropriate table based on database schema.

To set schema based tenancy in Exago:

1. On the **Data Source** set **Schema/Owner Name** to any valid value.
2. For each table/view/stored procedure create a data object. In the **Name** dropdown select the object that utilizes the schema value used in step (1). This will tell Exago that for this data object it should retrieve the schema from the data source.
3. When initializing Exago through the **API**, set the schema on the data source for the current user.

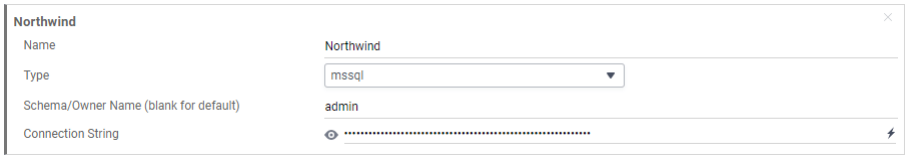


Database-Based Tenancy

Another way to assure that each user can only access their data is to provide a separate database for each user. In this situation each database should have the same tables, views and stored procedures.

To support database based tenancy in Exago:

1. Create a data source and corresponding data objects using any one of the databases.
2. When initializing Exago through the **API**, set the connection string on the Data Source to access the appropriate database for the current user.

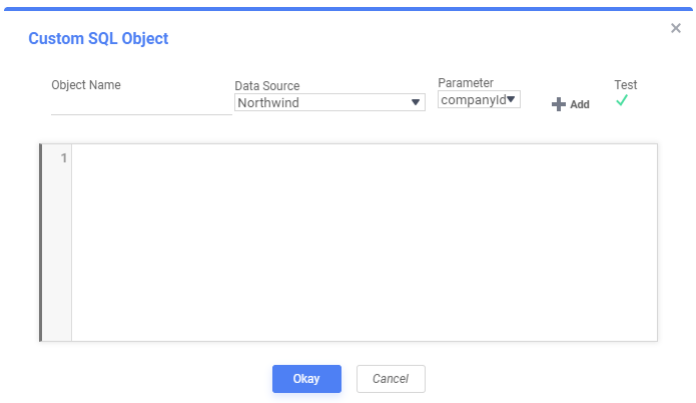


Custom SQL Based Tenancy

Multi-Tenant security can also be assured by using **Custom SQL** for all data objects. Exago can pass parameter values into each SQL statement to filter data based on user.

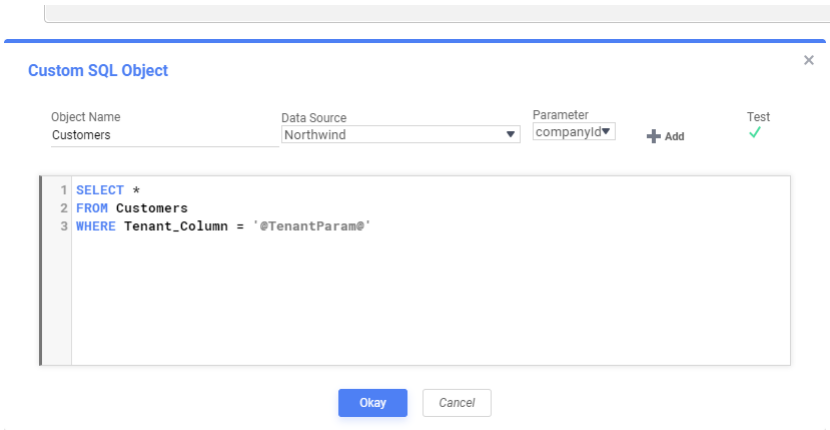
To set Custom SQL based tenancy in Exago:

1. For each data object open the **Custom SQL** menu and create the desired SQL utilizing parameters to assure only appropriate information is available.



2. When initializing Exago through the **API**, set the value of any parameters utilized in the SQL for the current user.

> **NOTE**
>
> Parameters should be surrounded by single quotes.



HTML Tag Sanitization

By default, the Web Application and Scheduler Services scan and remove unsafe HTML tags from any user input. User input areas include report cells, report descriptions, filter fields, and so on. This is to protect the application from any potential script injection attacks. Removing unsafe tags is referred to as a *blacklist* approach, because input is filtered against a blacklist of tags which are deemed unsafe.

Instead ***all*** HTML tags can be removed except those specifically deemed are safe. This is referred to as a *whitelist* approach. This can have a more unpredictable effect on user input, but it is generally safer if script injection attacks are a concern.

HTML formatting generated by the application, such as the Report Viewer, is not vulnerable to script injection.

Enabling Whitelist Filtering

To filter user input by a whitelist:

> **NOTE**
>
> This article references <WebApp>, <Sched> as a placeholder for the application's install location and the Scheduler Service's install location, respectively.

1. Edit the appropriate appSettings file in a text or XML editor:
   - Web Application: <WebApp>appSettings.config
   - Scheduler Application: <Sched>eWebReportsScheduler.exe.config
2. Add the following node between the <appSettings> </appSettings> tags:

```
<add key="inputSanitizationMethod" value="Aggressive" />
```

3. Restart the application or service.

Editing the Whitelist v2016.3.7+

To view and edit the tag whitelists, edit the appropriate whitelist file in a text editor. The whitelist files contain a JSON-formatted array of strings.To strip all HTML tags from user input, remove all tags from the tagwhitelist.json files:

> **TIP**
>
> To learn more about JSON, refer to the Using JSON article.

- Web Application: <WebApp>ConfigOthertagwhitelist.json
- Scheduler Service: <Sched>bintagwhitelist.json

Be cautious about adding potentially unsafe tags such as <input> and <iframe> to the whitelist. Removing HTML tags from the whitelist may cause existing reports to display incorrectly.

Enable FIPS Compliant Encryption on Windows

Exago is FIPS (Federal Information Processing Standard) 140-2 compliant. FIPS is a United States and Canadian government standard which defines a minimum set of security requirements for cryptographic systems. This standard is designed for products to secure sensitive but unclassified information.

Exago is compliant with FIPS Level 2 (140-2) which is the current active version of the standard. Before enabling FIPS, please be aware that you may lose access to certain websites which use SSL 1.0 via Internet Explorer. For more details, see the following Microsoft support articles:

- "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing" security setting effects in Windows XP and in later versions of Windows
- PRB: Cannot visit SSL sites after you enable FIPS compliant cryptography

Enabling FIPS

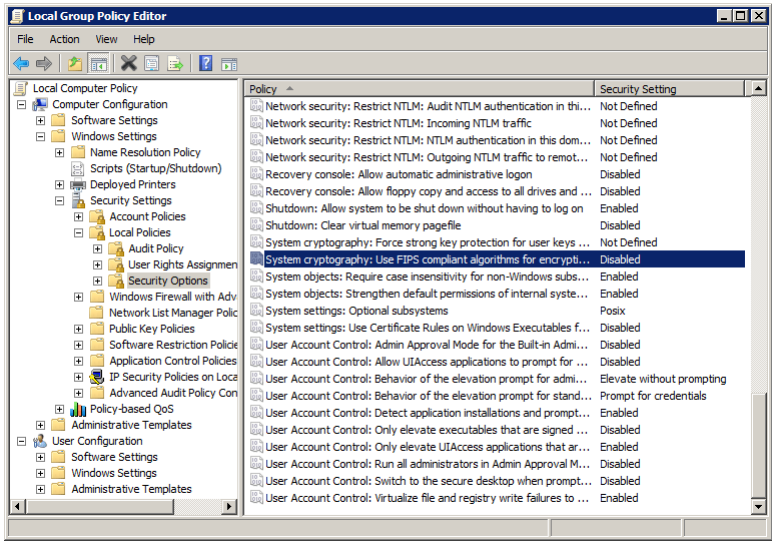
FIPS compliant encryption on Windows can be enabled using a local group policy setting or by editing a registry key.

Group Policy

Log in with an account that has administrative credentials. To open the Group Policy editor, press Start, press Run, type **gpedit.msc**, and press Enter.

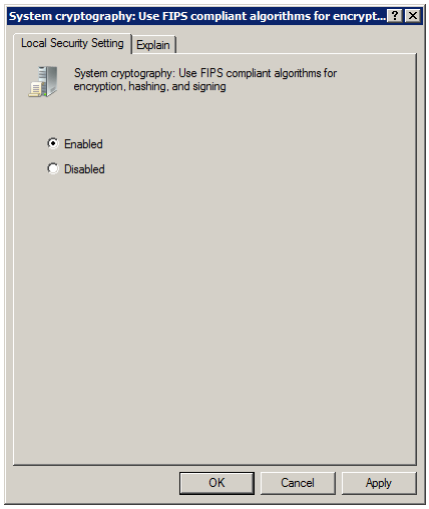
Navigate to the following setting:

Computer Configuration > Windows Settings > Security Settings > Local Policies > Security Options



Windows Local Group Policy Editor window

In the Details pane, double-clickÂ *System cryptography: Use FIPS-compliant algorithms for encryption, hashing, and signing.*



Enabling the System cryptography: Use FIPS compliant algorithm for encryption policy

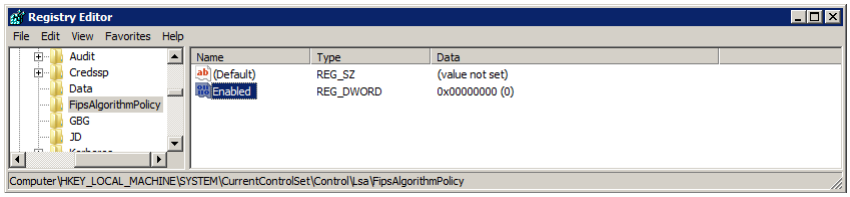
SelectÂ *Enabled*, and press OK or Apply. Then restart your webserver.

Windows Registry

Log in with an account that has administrative credentials. To open the RegistryÂ editor, press Start, press Run, type **regedit**, and press Enter.

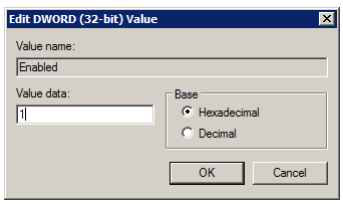
Navigate to the following key:

HKEY_LOCAL_MACHINESystemCurrentControlSetControlLsaFipsAlgorithmPolicy



Windows Registry editor

Double-click on *Enabled*.



Enter 1 in Value data, then press OK. Then restart your webserver.

About the Administration Console

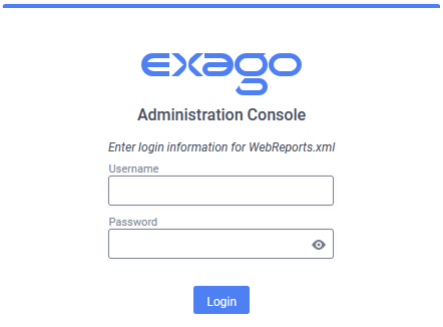
The Exago **Administration Console**, or **Admin Console** for short, serves as a user interface to set up and save administrative preferences. Use the Admin Console to create and modify a base session configuration file, including:

- Config File Options â€" Implement a static/dynamic configuration
- Storage Management â€" Configure report, theme, folder and template file storage
- Data â€" Establish how to connect to data sources and determine what data should be exposed to users. Configure system-wide variables called **Parameters**.
- General â€" Modify global settings of Exago to enable/disable features.
- Roles â€" Create and modify security **Roles** for individuals or groups of users.
- Extensions
  - Functions â€" Create and modify custom functions to make calculations on reports.
  - Aggregate Functions â€" Create and modify custom aggregate functions.
  - Filter Functions â€" Create and modify custom filter functions
  - Server Events â€" Create and modify custom code that is run when reports execute.
  - Action Events â€" Create and modify custom code that activates when a certain condition in the application is met.
  - Custom Options â€" Create and modify custom options that can be set on reports.

Accessing the Administration Console

Once Exago is installed, navigate with a web browser to **http://<Your Server>/<Exago Web Alias>/Admin.aspx**.

It is recommended to protect access to the **Admin Console** by setting values for the **General** > **Other Settings** > **User ID**, **Password** and **Confirm Password** settings. Subsequent access to the Admin Console requires the username and password be provided to gain access.



Admin Console login dialog in *v2021.2+*

After making changes in the Admin Console, click the **Logout** button in the top-right corner to lock the Admin Console back out.

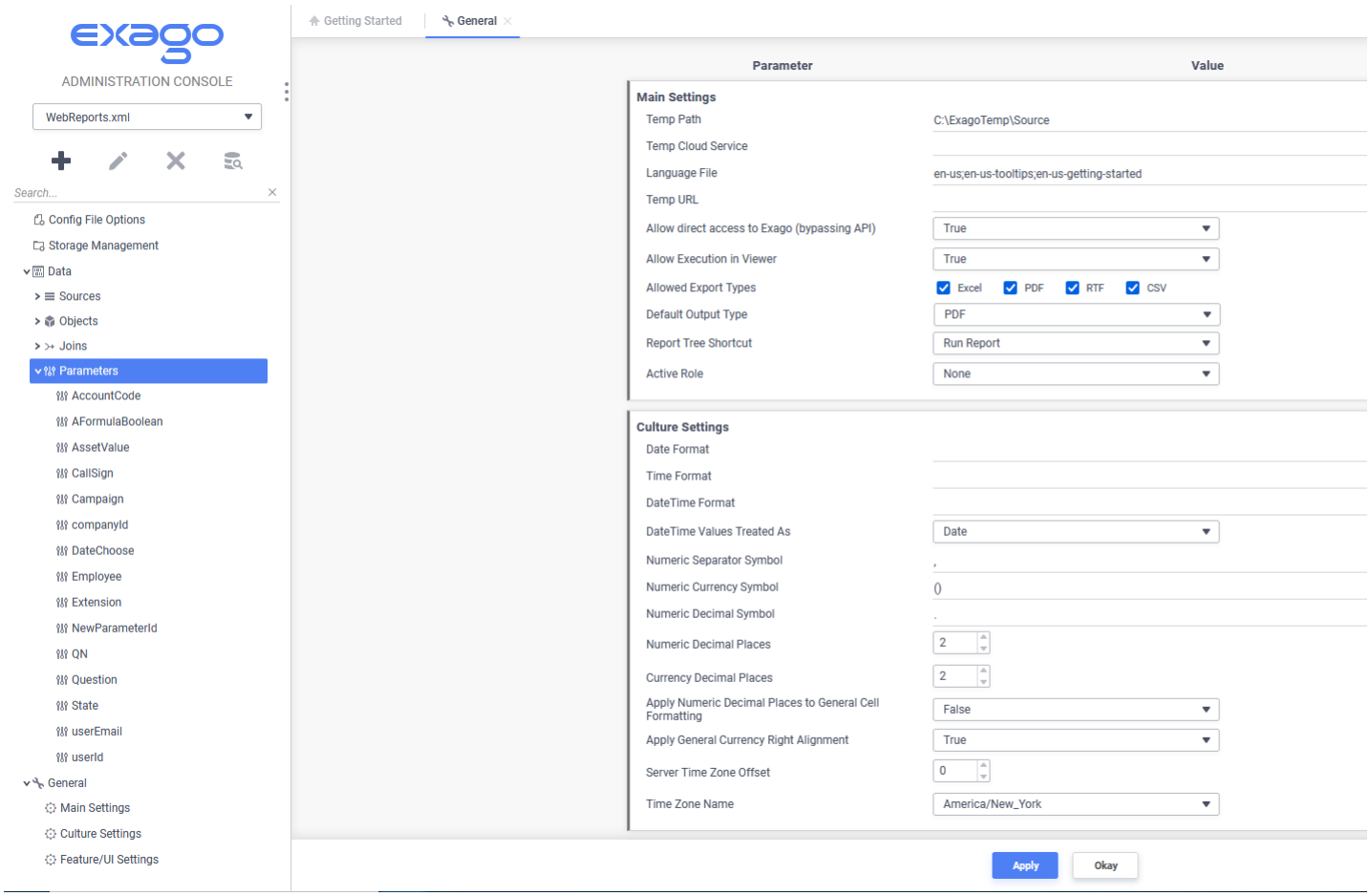
> **NOTE**
>
> The **Logout** button only appears when either the **General** > **Other Settings** > **User ID** or **General** > **Other Settings** > **Password** (or both) are set. For more information, see the Other Settings article.

Once Exago is configured and working, access to the Admin Console should be completely disabled. The Admin Console should never be accessible in a production environment. For more information, review the Security Checklist.

> **WARNING**
>
> Aside from the obvious security concerns by exposing the Admin Console in production, any changes made to the system configuration will invalidate all current user sessions causing an unsatisfactory user experience. For more information, review Modifying Configs in a Production Environment.

Navigation

The **Admin Console** consists of two sections. On the left is the **Main Menu** and on the right are **Tabs** that can contain menus to create and modify the configuration file.



The Exago Admin Console in v2021.1

Main Menu

Through the main menu you can:

- Configure the **Storage Management** system
- Create **Data Sources**, **Data Objects**, **Joins**, **Parameters**, **Roles**, and **Custom Functions**.
- Edit settings for: **Data**, **Roles**, **Functions**, and **General** features.
- Delete **Data Sources**, **Data Objects**, **Joins**, **Parameters**, **Roles**, and **Functions**.
- Automatically discover data source objects and metadata.

Click the **Collapse** icon to hide (or restore) the main menu.

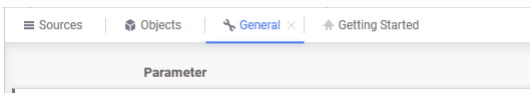
Tabs

The right section of the **Admin Console** is made up of tabs containing menus to create and modify administrative settings.

To save the changes made in a tab click **Apply** or **Okay.**

Tabs can be closed with or without saving by clicking the **Close**

icon to the right of the tab name.



The General tab can be closed by clocking the X icon to the right of its name in the tab

Tabs may be arranged by clicking and dragging them as desired.

The **Admin Console** creates two base configuration files: an XML file named WebReports.xml and an encrypted XML file named WebReports.xml.enc. These files are created and saved in the /Config sub-directory of Exago's installation location. It is recommended to use this as the live version of the config in a production environment. Copy WebReports.xml to a secure backup, and then delete it from the Config directory.

Creating Additional Configuration Files

As part of the integration of Exago you may want to create alternative configuration files in addition to WebReports.xml. Additional configuration files can be utilized in two ways:

- If entering Exago directly, the configuration file to be used is specified in the **Custom Styling.**
- When entering through the API the configuration file to be used is specified in the **API Constructor Methods**.

To create additional configuration files:

1. Navigate to the **Administration Console** in a web browser.
2. Append "*?configFn=NewConfigFile.xml*" to the URL replacing NewConfigFile with the desired configuration file name.
3. Click in the URL bar and press enter.

Storage Management

Â

This article applies to the **Admin Console** > **Storage Management** settings.

Assembly Location

The path and filename of the DLL that implements the IStorageManagement interface. The default value is *<WebApp>binWebReports.ContentDatabase.dll*, which will use Exago's Storage Management implementation, where <WebApp> represents the Web Application's installation directory.

Provide the fully qualified path to the file. If no path is supplied, Exago will check the Web Application's bin directory for the DLL, or in *v2020.1.15+* and *v2021.1.3+* fallback to the internal Exago implementation of Storage Management.

Class Name

The name of the class in the assembly that implements the IStorageManagement interface. The default value is *WebReports.ContentDatabase.StorageMgmtDatabase*, which will use Exago's Storage Management implementation.

Click the **Validate Assembly**

icon to validate that the assembly exists and that all methods of IStorageManagement can be found by Exago in that class.

Database

Out-of-the-box, Exago provides a SQLite implementation of the Storage Management database. The default values noted for each setting below reflect the default SQLite installation.

> **IMPORTANT**
>
> Exago recommends using the included SQLite database for development, testing and demonstration purposes only. Use MSSQL, MySQL, Oracle or PostgreSQL in a production environment.

Database Type

Choose the type of the Storage Management database:

- *SQLite* (default value)
- *MSSQL* â€" Microsoft SQL Server
- *MySQL*
- *Oracle*
- *PostgreSQL*

Database Provider

Specify the name of the database provider Exago will use to connect to the Storage Management database. This matches the *InvariantName* found as a property of **DbProviderFactories** in the server's machine.config file. The default value is *SQLite*

> **NOTE**
>
> The machine.config file is located in the %SystemRoot%Microsoft.NETFramework%VersionNumber%CONFIG directory.

> **TIP**
>
> Use the applicable database provider name from the **Admin Console** >
>
> **General** > **Database Settings**.

Database Connection

Connection string Exago will use to connect to the Storage Management database.

Default value for SQLite is *Data Source=<FILENAME>*. Replace *<FILENAME>* with the absolute path and file name to the database file. For example: Data Source=D:StorageMgmt.sqlite. When using SQLite, the default location will be the /Config directory under the Exago Web Application install directory.

For PostgreSQL on Windows, include Unicode=true in the connection string.

Tables

Review Storage Management: Database Schema and Storage Management: Introduction for more information about the purpose of each of these tables. Each table name may also include a schema (e.g. *dbo.content*).

Table Prefix

Provide an optional prefix for all Storage Management database tables. Review the Storage Management: Database Schema article for information. This option can be used to add a schema if it exists.

Examples:

1. test_ will result in tables named *test_content*, *test_party_type*, *test_content_access* and *test_storage_meta*
2. dbo. will result in tables named *dbo.content*, *dbo.party_type*, *dbo.content_access* and *dbo.storage_meta*

Check Database Settings

Click to check the connection between Exago and the Storage Management database. Also checks that the tables exist in the database.

Show Prepare Database SQL

Click to display the SQL statements that will initialize a Storage Management database with the schema and default party types. Works similar to the Prepare Database button but will display the SQL statements on screen instead of executing them.

This button will also show any SQL statements needed to update the schema if necessary on an existing database.

Prepare Database

Click to create the four Storage Management tables, load default party types into the Party Type table and create the **Public** and **My Reports** folders if they don't already exist.

This button will also update the schema if necessary on an existing database. It is not destructive.

Load Themes

Click to load new chart, ExpressView, CrossTab and GeoChart themes from the <WebApp>\Themes folder into the Storage Management database.

This is only necessary for first time installations of Exago, after upgrading to Storage Management from a legacy storage mechanism and themes were not imported with one of the transitioning utilities or after adding new themes to the Themes directory (either by Exago or by customizing the application).

Update Reports v2021.1+

The **associated_reports** column of the Storage Management Content table is a comma separated list of content_ids for each report that is associated with a particular content record. Reports become associated with others when they are components in a Composite Report such as Chained Report or Dashboard, or if an Advanced Report contains links to other reports (a.k.a. drilldowns).

This **Update Reports** function in the Admin Console reads the contents of a Storage Management database, parses the report file contents and generates the comma separated lists of associated_reports where applicable.

Depending on the size of the Storage Management database, the Update Reports process may take several minutes to complete. Once finished, an Update All Reports dialog will appear with the results of the operation.

For more information, hover the mouse over the button to see the tooltip. Then click on the **Storage Management** link in the tooltip.

Identity

The Storage Management Identity keys are used similar to the system-wide @ and @ parameters but only affect content storage. Like @ and @, they should be set each time an Exago session is created via the API.

The settings here can be used for testing and demonstration purposes. Review the Storage Management: Introduction and Storage Management: Database Schema articles for more information on how these keys are used in the application.

Class Id

Set a value for the **Class Id** identity key.

Company Id

> **NOTE**
>
> This is similar to, but is not the same as the system-wide @ parameter.

Set a value for the **Company Id** identity key.

User Id

> **NOTE**
>
> This is similar to, but is not the same as the system-wide @ parameter.

Set a value for the **User Id** identity key.

When accessing Storage Management properties through the Admin Console (e.g. when viewing Role Folder Permissions) all access filters will

be removed and all content will be available.

Owner Id

Set the value for **Owner Id**. When the Owner Id of a content item matches the value of the Owner Id identity key, full access is granted to that content item automatically.

Default Settings & Content Access Permissions

The default settings determine how permissions are configured for newly created content in the root.

Individual folders have their own inherit flag, default party type and default access flags properties. If for some reason individual folders do not have their own inherit flag, default party type or default access flags set (i.e. they are null or disabled) then the values here will be applied.

Inherit Flag

Set the default value of the **Inherit Flag**.

If *True*, new folders created in the root will have theirÂ **inherit flag** set toÂ *true*.Â When a folder's **inherit flag** is *true*, new content created in that folder will copy all of the access records from its parent.

If *False*, new folders created in the root will have their **inherit flag** set to *false*. When a folder's **inherit flag** is *false*, new content created in that folder will have a single new content access record written with the **default party type** and **default access flags**.

Default value is *True*.

> **NOTE**
>
> This setting determines the value of the **Inherit Flag** on child folders of the root. Folders created in the root do not copy the access records from the root folder. Instead, a single content access record with the **Default Party Type** will be written for new folders in the root.

Party Type

Set the **default party type** for newly created folders in the root. Regardless of the value ofÂ **Party Type**, new folders created in the root will have theirÂ **default party type** set to this value.

When **Inherit Flag** is *True*, this dropdown is disabled as content access records will be inherited from the content's parent and therefore will not have an effect.

Access Flags v2020.1â€“v2021.1

Default Access Permission v2021.1+

Set the **default access flags** that will be used for newly created content when not inheriting access from the parent objects. Choose from:

- *Full Access* â€" there will be no restriction on access to content
- *Read Only* â€" content is marked as read-only in a similar way to Roles. When content is read-only, it cannot be edited, saved or deleted. Read-only folders cannot have new content saved into them. Read-only reports can be duplicated into non-read-only folders.
- *Custom (v2021.1+)* â€" a different content access strategy can be implemented by manually checking the corresponding permission checkboxes.

Each of the access flags are defined in the Storage Management: Introduction article.

Cache Settings

Enable Report List Caching

> **NOTE**
>
> This setting was formerly located in
>
> **General** > **Other Settings**.

Set to *True* to enable the folders and report list in the

Report Tree to be cached, rather than querying the Storage Management database each time they are needed. Default value is *True*.

It is recommended to enable **Report List Caching**, even with a small timeout. Care should be taken when running Exago in a web farm environment, as the caches will become unsynchronized.

Timeout

If **Enable Report List Caching** is *True*, set the timeout in seconds that the folder and report list cache should expire. Default value is *30* seconds.

Enable Report XML Caching

> **NOTE**

This setting was formerly located in

General > **Other Settings**.

If *True*, the XML report definition content will be cached rather than being read from the database each time it is needed. Enabling this setting will reduce the number of calls to the Storage Management database and may help with performance. Default value is *True*.

It is recommended to enable **Report XML Caching**, even with a small timeout. Care should be taken when running Exago in a web farm environment, as the caches will become unsynchronized.

Timeout

If **Enable Report XML Caching** is *True*, set the timeout in seconds that the folder and report list cache should expire. Default value is *30* seconds.

Theme List Caching Timeout

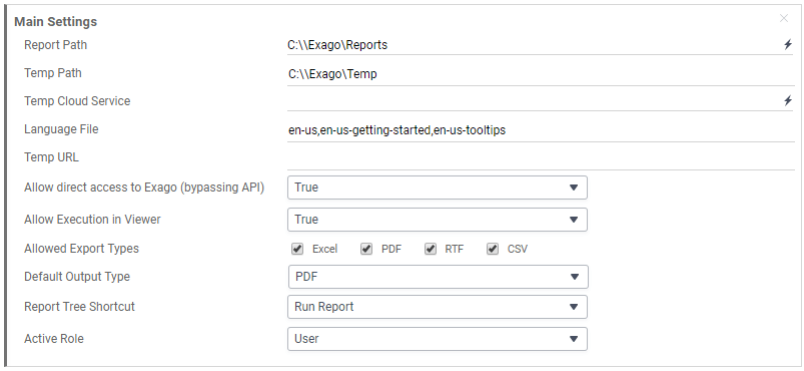
Set the timeout in seconds that the report and chart theme cache should expire. Default value is *30* seconds.

Main Settings

This article applies to the **Admin Console** > **General** > **Main Settings**.

This document covers several versions of the application. Use the **Viewing content for** dropdown to see only the content relevant to a specific application version.

The main settings are the basic options for Exago. The following settings are available:



The Main Settings in the Admin Console pre-v2020.1

Temp Path

The location where temp files are stored. The Temp Path may be:

- **(no value)** â€" All temp files and images will be saved to ./Temp.
- **Virtual Path**
- **Absolute Path** â€" Temp files will be saved to the absolute path and image files will be saved to ./Temp

> **IMPORTANT**
>
> Before deploying into a production environment be sure to set a **Temp Path** that resides behind your serverâ€™s firewall/security system.

Temp Cloud Service

Web Service, .NET Assembly, or Azure Authentication string used to integrate into a cloud environment. For more information see **Cloud Environment Integration**.

Language File

List of the XML files that specify language strings. See **Modifying Select Language Elements** for more details.

Temp URL

Overrides the portion of the temporary URL used to store images from HTML exports. Temp URL can override just the scheme (i.e. https) or the full URL.

Allow Direct Access to Exago

A boolean setting:

- *True*: allows users direct access to Exago with no security.
- *False*: users must be authenticated by the host application to enter Exago. Users that try to directly access Exago will receive a message saying â€˜Access Denied.â€™

IMPORTANT

We highly recommend setting this to *False* before deploying Exago in a production environment.

Allow Execution in Viewer

Enables or disables running reports in the Report Viewer.

NOTE

Setting **Allow Execution in Viewer** to *False* will also prevent creating new Dashboards regardless of the setting of **Admin Console** > **General** > **Feature/UI Settings** > **Allow Creation/Editing of Dashboards**.>

Allowed Export Types

The available formats for exporting all reports. Check the box of the formats that should be available.

Default Output Type

The export format that appears when a new report is selected unless a specific export format is set in the Options Menu of the Report Designer.

NOTE

Allowed Output Types.

Report Tree Shortcut

Whether the report execute button in the report tree defaults to Run the report in the Report Viewer, or Export the report to the default format.

Active Role

Retains the Id of the Role to be made active upon session start. The active Role may be changed manually using the dropdown. For more information please see the **Roles** article.

Culture Settings

This article applies to the **Admin Console** > **General** > **Culture Settings**.

Introduction

The **Culture Settings** give administrators control over formats and symbols that vary amongst geographic location. These settings can be overwritten for a specific user or group of users by modifying the Role. For more information see **Roles**.

Settings

The following settings are available:

Date Format

The format of date values. May be any .NET standard (e.g., *MM/dd/yyyy*).

Time Format

The format of time values. May be any .NET standard (e.g., *h:mm:ss tt*).

DateTime Format

The format of date-time values. May be any .NET standard (e.g. *M/d/yy h:mm tt*).

NOTE

For more information on .NET Date, Time, and DateTime format strings please visit this MSDN article.

Date Time Values Treated As

Choose to format DateTime as Date or DateTime values. To change this setting for specific columns see **Column Metadata**.

NOTE

Column metadata will override culture settings.

Numeric Separator Symbol

Symbol used to separate 3 digit groups (ex. thousandths) in numeric values. The default is a comma (,).

Numeric Currency Symbol

Symbol prepended to numeric values to represent currency. The default is the US currency symbol ($).

Numeric Decimal Symbol

Symbol used for numeric decimal values. The default is a period (.).

Numeric Decimal Places

The default number of decimal places to show when a cell format is set to display a **Number** or when column metadata or the data source identifies the value is a number data type. This value is also the default value presented to users in the Report Designer Cell Formatting dialog. Default isÂ *-1* which will show two decimal places. Values between -1 and 8 are accepted.

Currency Decimal Places

The default number of decimal places to show when a when column metadata or the data source identifies the value is a currency data type. Default isÂ *-1* which will show two decimal places. Values between -1 and 8 are accepted.

Apply Numeric Decimal Places to General Cell Formatting

Set to true to apply the Numeric Decimal Places to any cell that has Cell Formatting set to General but contains a number. Default value is *false*.

Apply General Currency Right Alignment

Set to true to cause currency values to appear right-aligned by default in report cells.

Server Time Zone Offset

A decimal value used to convert server to client time (the negation is used to convert client to server time). This offset is not applied to data coming from Data Sources. It is utilized by Exago features like the Scheduler Service. Set to *0* to use server time to calculate the time zone change. This value should be set to *0* if a **Time Zone Name** is selected.

> **NOTE**
>
> A **Server Time Zone Offset**Â is only required if a **Client Time Zone Name** cannot be provided. Otherwise, all scheduling calculations will be performed using the provided time zone information.

Time Zone Name

Default geopolitical location of the client as determined by the IANA time zone database (e.g., America/New_York). Utilized by Exago to properly resolve scheduling issues centering around DST and other non-linear time zone conflicts. The setting is also used for other time and date related features in the application such as Date/Time functions like Now().

> **NOTE**
>
> For a full list of official IANA time zones and their relative UTC offsets, please reference this Wikipedia article.

Feature/UI Settings

This article applies to the **Admin Console** > 🔧

General > ⚙ **Feature/UI Settings**.

Introduction

The **Feature/UI settings** allow administrators to hide various features in the user interface. As each heading indicates, settings may apply to specific report types or the entire application.

Settings

The parameters are broken up into several sub-sections:

- Available Report Types
- ExpressView Settings
- Advanced Report Designer Settings
- Dashboard Report Designer Settings
- Common Settings

Available Report Types

These settings enable/disable report types.

Allow Creation/Editing of Advanced Reports

Enables/Disables the Report Designer

Allow Creation/Editing of CrossTabs

Enables/disables the creation of new CrossTab reports from the main menu.

When *False*, the **Show CrossTab Wizard** setting will appear under Advanced Report Designer Settings, permitting CrossTabs to be enabled/disabled separately in the Advanced Report Designer.

Allow Creation/Editing of Dashboards

Enables/Disables the Dashboard Designer.

> **NOTE**
>
> Both **Allow Creation/Editing of Dashboards** and **General** > **Main Settings** > **Allow Execution in Viewer** must be *True* to enable the Dashboard Designer.

Allow Creation/Editing of Chained Reports

Enables/Disables the Chained Report wizard.

Allow Creation/Editing of ExpressViews

Enables/Disables the ExpressView designer.

ExpressView Settings

These settings only apply to the ExpressView designer.

Allow Editing ExpressView with Live Data

When *True*, users are allowed to make changes to an ExpressView when the in Live Data (**Run**) mode. When *False*, users must click the **Stop** button on the toolbar before changes may be made.

The default value is *false* for *pre-v2021.1.1* or *true* for *v2021.1.1+*

Fields Enabled in Data Fields TreeÂ v2017.1.2+

This setting controls whether users are allowed to add fields to an ExpressView that are not directly joinable to another field on the report.

- **All joinable fields** (*default*): Users can add any fields with a join path to existing report fields.
- **Direct joins only**:Â Users can only add fields with a direct join to an existing report field.

> **NOTE**
>
> As of *v2018.1.23+*, the join algorithm hasÂ been modified for ExpressViews with **Direct joins only** enabled. The **Disable Non-Joined Data ObjectsÂ** database setting is now taken into consideration when generating a join path, allowing these ExpressViews to properly analyze join weight and encompass constructed SQL.

Join Path Algorithm

Select the join path algorithm to use when running ExpressViews with multiple data objects.

- **Standard** (*default*): More performant in most cases.
- **Legacy**:Â Select if you are experiencing issues with ExpressViews created in an older version.

Advanced Report Designer Settings

These settings only apply to the Advanced Report designer.

Show CrossTab Wizard

If Allow Creation/Editing of CrossTabs in Available Report Types is *False*, this option becomes available. When Allow Creation/Editing of CrossTabs is *True*, this option is hidden from view in the Admin Console.

Set to *True* to allow CrossTab widgets to be added to Advanced Reports in the Designer, or *False* to completely disable the creation of CrossTabs in the application.

Show Chart Wizard

Enables/disables the **Chart Wizard** from the **Insert** menu (*v2021.1+*). Set to *False* to disable users from creating or editing charts.

Chart Colors

Lists the values used for default chart colors, except for heatmap charts. Hexadecimal values should be separated by commas or semicolons.

Heatmap Chart Colors v2020.1.3+

Lists the values used for default heatmap chart colors, either as CSS color names or hexadecimal color codes with a leading # sign. Values must be separated by commas or semicolons. This list of colors will be applied to a heatmap when selecting *Default* from the **Theme** dropdown in the chart wizard.

Maximum Number of Chart Data Points

Upper limit on the number of data points visible on a chart. If the limit is exceeded, a warning will be displayed to the user. Charts with large numbers of data points could cause browser performance issues.

Default Chart Font

Specifies a default font for charts created in the Advanced Report Designer. This setting can be overridden on a per-Report basis. Does not apply to ExpressView charts or Dashboard visualization tiles.

Show Geochart Map Wizard

Enables/Disables **GeoCharts**Â from the **Insert** menu (*v2021.1+*). Set to *False* to disable users from creating or editing GeoCharts.

> **NOTE**
>
> The first time Show Geochart Map Wizard is set to true a dialog appears prompting you to accept the terms of using the Google Charts Api. Type “I accept” in the first box and your full name in the second to accept the terms and enable mapping.

GeoChart Map Key

Optional Google Maps license key for GeoCharts permissions. License must contain the **Google Maps Javascript API** service. See **Legacy Maps (Geocharts)** for more information.

> **NOTE**
>
> Due to a change in Google’s Maps API Terms of Service, if geocharting was enabled after June 2016, or if you had geocharting enabled before, but changed your host domain name after June 2016, you need a license key to use this feature.

Geochart Map Colors

List the values used for default Geochart map colors. Hexadecimal values or CSS color names should be separated by commas (or semicolons).

Show Google Map Wizard

Enables/Disables **Google Maps**Â from the **Insert** menu (*v2021.1+*). Set to *False* to disable users from creating or editing Google Maps.

> **NOTE**
>
> In order to use Google Maps, a license key must be obtained from Google, a polygon file must be downloaded from our **support site** and the Exago server must have an Internet connection. See **Installing Optional Features** for more information.

Google Map Key (unlimited or JS API restricted)

License key for Google Maps permissions. Must contain either:

- Google Maps **JavaScript API** and **Geocoding API** services, unlimited key.
- Google Maps **JavaScript API** service, limited key with referrer URLs. If this is the case, supply a limited **Geocoding API** service key in the following field.

> **NOTE**
>
> When upgrading from a version prior to *v2018.1*, the value previously supplied for the Google Map Key will appear here.

Google Map Key (optional Geocode API restricted)

License key for Google Maps permissions. If the previous field contains a key limited to the **JavaScript API** service, supply a limited **Geocoding API** service key in this field, given server IP addresses. Otherwise leave this blank.

Google Map Colors

List the values used for default Google map colors. Hexadecimal values or CSS color names should be separated by commas or semicolons.

Show Gauge Wizard

Enables/Disables the **Gauge Wizard**Â from the **Insert** menu (*v2021.1+*). Set to *False* to disable users from creating or editing gauges.

Gauge Colors

List the values used for default gauge colors. Hexadecimal values or CSS color names should be separated by commas (or semicolons).

Show Document Template

Enables/Disables working with document templates. Set to *False* to disable users from using the Document Template menu.

Show Document Template Upload Button

Set to *True* to allow users to upload document template files from their local computer. Set to *False* to prevent users from uploading Document Templates.

Show Linked Report

Enables/Disables the **Linked Report**

icon in the toolbar. Set to *False* to disable users from linking reports.

Show Linked Report Fields

Enables/Disables the Fields tab in the Linked Report dialog.

Show Linked Report Formula

Enables/Disables the Formula tab in the Linked Report dialog.

Show Linked Action

Enables/Disables the **Linked Action Event**

icon. When *True*, users will be able to add **Action Events** to reports.

Show Auto Sum Button v2021.1+

Enable/disable theÂ **AutoSum**

icon in the Advanced Report Designer's toolbar. The default value is *False*.

Show Insert Image

Enables/Disables **Image**Â from the **Insert** menu (*v2021.1+*). Set to *False* to disable users from inserting images.

Show Joins Window

Enables/Disables the Joins menu under **Advanced**. Set to *False* to disable users from modifying joins.

Show Advanced Joins

Enables/Disables additional options in the Joins menu. Set to *True* to enable advanced users to create, delete, and modify joins.

Advanced Joins Display v2017.3.1+

Select whether to show complex join options in the report Joins menu.

- *Complex* â€" allow users to modify join operators and expressions, and allow conjoining clauses.
- *Standard* â€" only permit joining data columns on equality.

Allow Data Object Aliasing v2021.1+

Data Object aliasing allows users to add categories multiple times on the same report. This may be necessary for some advanced join operations.

Show Events Window

Enables/Disables the Events item under **Advanced**. Set to *True* to allow users to apply Server Event Handlers in the report. See **Server Events** for more information.

Show Report-Level Parameters Window v2019.1.3+

Enables/Disables the Parameters item under **Advanced**. Set to *True* to allow users to view system-parameters and define their own report-level parameters for use within reports. See **Advanced Options** for more information.

Show SQL Window

Enables/Disables the Show SQL menu under **Advanced** . Set to *True* to allow users to view the SQL that will be sent to the data sources upon execution. See **Advanced Options** for more information.

Linked Report Display v2017.3+

Specify where to display drilldowns for linked reports.

- *Cursor â€"* a popup window appears at the mouse cursor
- *NewTab â€"* the linked report opens in a new user interface tab
- *Center of screen â€"* a popup window appears at the center of the screen

Allow Grouping on Non-Sorts

Enables/disables users to create groups on non-sort formulas.

> **NOTE**
>
> Grouping on non-sort formulas is deprecated and unsupported.

Allow Creation of Custom SQL Objects v2018.1+

Allow end-users to write custom SQL objects at the report level.

> **WARNING**
>
> Carefully review the Report Level SQL article before enabling.

Data Sources to Exclude from Custom SQL Creation v2018.1+

When Allow Creation of Custom SQL Objects is enabled, enter the data sources to exclude from Report-Level SQL. Write each data source in double quotation marks (“), and separate sources by a comma (,). Example: "Northwind","AdventureWorks".

Dashboard Report Designer Settings

These settings only apply to the Dashboard designer. If Allow Creation/Editing of Dashboards is *false* these settings will be ignored.

Prompt user for Parameters/Filters on Execution

Default setting indicating whether to prompt the user for filter and/or parameter values when executing a dashboard. The option can be overridden on an individual dashboard in the Options menu.

Automatically Refresh Reports v2021.1+

In the Dashboard Designer, enable/disable automatically loading data into the tiles, requiring the end user to manually refresh the tile in order to see any data in it.

When *True*, reports and visualization tiles will automatically load data when the Dashboard Designer is first opened, and whenever changes are made to the Dashboard or the tiles on the canvas. This is the *pre-v2021.1* behavior.

When *False*, reports and visualization tiles will display placeholder data until either the **Refresh** 

icon on the tile, or the **Refresh**  icon on the Designer’s toolbar is explicitly clicked. If changes are made to the Dashboard or the tile, a

notification icon  will appear in the tile and toolbar. By limiting the frequency of calls to the Data Sources and execution engine, responsiveness of the Dashboard Designer can increase especially for very complex and data intensive Dashboards.

This functionality is referred to as the **Refresh Reminder** feature in the Dashboards documentation. When *False*, the default value, tiles do not automatically load and therefore the Refresh Reminder feature is enabled.

Show URL Item Button

Display/Hide the URL item in the New Tile Menu of the Dashboard Designer. When *False*, users will not be able to add URL tiles to the Dasboard canvas.

Allow Creation/Editing of Dashboard Visualizations

Display/Hide the Visualization item in the New Tile Menu and Tile Properties Pane of the Dashboard Designer.

Refresh Reports/Visualizations on Dashboards Silently

Set to *True* to disable the refresh hourglass animation for timed automatic dashboard reloads.

Minimum Tile Width for Dashboard Reflow v2019.1+

Specifies the minimum width a tile will be drawn when Dashboard Reflow is enabled. Please see theÂ **Canvas** section of the **Dashboard Designer** article for more information.

Minimum Window Width for Dashboard Reflow v2019.1+

Specifies the minimum window size where Dashboard Reflow will begin to take effect. Please see theÂ **Canvas** section of the **Dashboard Designer** article for more information.

Common Settings

Default Designer Font

Specifies a default font for reports created in the Express Report Wizard, Advanced Report Designer and Dashboard Designer. This setting can be overridden on a per-report basis.

End-users must have the selected font installed locally in order to display. Otherwise, Exago will default to Sans Serif.

Default Designer Font Size

Specifies a default font size for reports created in the Express Report Wizard, Advanced Report Designer, and Dashboard Designer. This setting can be overridden on a per-report basis. Does not apply to CrossTabs.

Show Help Button

Enables/Disables the **Help**

icon in the top right corner of the user interface. Set to *False* to disable users from accessing the context-sensitive help system.

Custom Help Source

Specifies the URL that contains custom context-sensitive help content. See **Custom Context Sensitive Help** for more details.

Show Exports in Tab

Set to *True* to open PDF reports in an Exago tab. Set to *False* to prompt the user to download the PDF.

Show IE Download Button

Set to *True* if Internet Explorer is not automatically prompting users to download PDF, XLS, RTF or CSV export files.

Show Join Fields

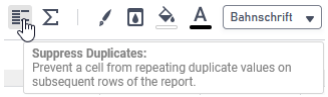
Enables/Disables any **Data Fields** that are used as Unique Keys or Joins. Set to *False* to hide all unique key and join Data Fields from users. To hide specific Data Fields see **Column Metadata**.

Show Grid Lines in Report Viewer

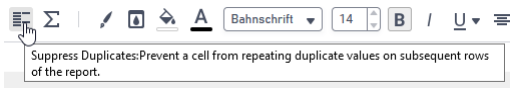
Sets the default output to show grid lines or not. This can be overriden on a per-report basis.

Show Enhanced Tooltips

Sets the style of tooltips to display. If *True*, enhanced tooltips which support HTML will be displayed. If *False*, standard browser tooltips will be shown.



Show Enhanced Tooltips set to *True*



Show Enhanced Tooltips set to *False*

Save on Report Execution

Set to *False* to disable automatic saving of reports when executing from the Report Designer.

Save on Finish Press

Set to *False* to disable automatic saving of reports when finish button is pressed in a wizard.

Enable Right-Click Menus

Set to *False* to disable right click menus.

Enable Reports Tree Drag and Drop

Set to *False* to disable the dragging of reports and folders in the Report Tree.

Show Report Upload/Download Options

Set to *True* to enable users to upload and download report files by right clicking on folders and reports. Default value is *False*.

Allow interactivity in Report Viewer

Set to *False* to disable Interactive Report Viewer capabilities, including: changing column width, styling output, and interactive filters.

Show Toolbar in Report Viewer

Specify if Report Viewer should display paging, search, and export options.

- *Auto* â€" Exago will detect if the report only displays a single page of content from the Report Footer Section. If so the HTML Toolbar will be hidden, otherwise it will show.
- *Show*â€"the toolbar will always show.
- *Hide* â€" the toolbar will never show.

Default interactive report viewer dock is open

Set to *True* to have the Interactive Dock opened by default when a report is executed in the Report Viewer.

Interactive report viewer default dock placement

Specify if the Interactive Dock should appear on the left or right of the Report Viewer. The default value is is *Left pre-v2021.1.5* or *Right v2021.1.5+*

Allow save to report design for report viewer

Set to *False* to prevent users from saving changes to a report made in the Report Viewer back to the original report design. This effectively removes the **Save Changes To This Report** and **Save Changes to New Report** options in the Report Viewer.

Maximum number of fields in a CrossTab header or tabulation source

Specify the maximum allowed fields in a CrossTab header or tabulation source. Note that adding a large number of data fields to a CrossTab will significantly increase the execution time of the report.

Use SVG for Application Icons v2016.3+

Set to *True* to se SVG icons instead of the default PNG icons in user interface elements. SVG icons look nicer on high-pixel density screens, but they may not be compatible with some web browsers.

When using a custom application theme, this setting should be set to *True*.

Application Theme Selection v2016.3+

Choose from a selection of available user interface themes, or choose *Create NewApplication Theme…* to create a new theme. See **Application Themes** for more information.

After choosing a theme, users should clear their browser cache and refresh the user interface in order for the theme change to take effect.

Update All Application Themes v2020.1.3+

Click the ↻

Update button to update all custom application themes with the newest copy of the image and CSS files available in the Basic theme. It is recommended to click this button after updating Exago from one major version to another (e.g. *v2020.1*â€"*v2021.1*).

Depending on the number of the themes available and the number of changes made between versions, this operation may take a minute or so to complete.

> **NOTE**
>
> The context-sensitive help will not be affected by the theme selection.

Show Data Fields Search Box v2017.2+

Data Fields Search Mode

Enables/Disables the data field search tools in the Data Fields Pane of the ExpressView, Advanced Report and Dashboard Designers.

- *Disabled* â€" there is no search box
- *As You Type* â€" search results display as the user types
- *On Submit* â€" search results display after hitting the Enter key

> **NOTE**
>
> We highly recommend setting **Column Metadata**, and setting **Schema Access Type** to *Metadata* for all available objects, before enabling this feature.

Report-Level SQL Objects

This article applies to the **Admin Console** > 🔧

General > ⚙ **Feature/UI Settings >** ⚙ **Allow Creation of Custom SQL Objects** setting.

Beginning with *v2018.1*, administrators have the ability to allow end-users to create reports using custom report-level SQL objects written in the end-user interface.

See **Advanced Reports: Data Objects (v2021.1+)** for info on how end-users will be able to use this feature.

To enable Report-Level SQL, in the Admin Console, set

General > **Feature/UI Settings >** **Allow Creation of Custom SQL Objects** to *True*.

Warning

Protect your data from unauthorized SQL injection

This feature allows report writers to execute arbitrary SQL commands against data sources they can access. By default this is ALL sources except those you have specifically excluded.

Contact your database administrator to ensure that the connection string has READ-ONLY access. **Do not** enable Report-Level SQL without a restricted connection string for each allowed source.

Furthermore, because Report-Level SQL bypasses the Admin Console data model, Role (row-based) and column tenancy restrictions on data objects have **no effect**. Therefore, ensure that the connection string also restricts viewing and joining to unauthorized tables and schema.

Exclude unauthorized sources from Report-Level SQL by entering their names, surrounded by quotes (“) and separated by commas (,), in the Admin Console field *Data Sources to Exclude from Custom SQL Object Creation*.

Example

```
"NorthWind","AdventureWorks"
```

Note: This prohibits creation, but not execution, of Report-Level SQL reports with these sources.

You can deny Roles access to Report-Level SQL by setting the following Role field to False:

(**Role | General**) *AllowCreation of Custom SQL Objects in Advanced Reports*

This prohibits creation and execution of reports with Report-Level SQL. To permit execution, enable the following setting:

(**Role | Objects**) *AllowUser to ViewReport-Level Custom SQL Objects*

Please be cautious with your data.

Programmable Object Settings

This article applies to the **Admin Console** > **General** > **Programmable Object Settings**.

The **Programmable Object Settings** enable you to specify names for parameters that will be passed from Exago to stored procedures, web services, or .NET Assemblies. Using these parameters will allow filtering to be done before the data is sent to Exago. This can increase performance and reduce server resources when using Programmable Objects. For more information on these types of Data Objects see **Web Services & .NET Assemblies**.

> **NOTE**
>
> If performance is a concern, especially for large data sets, database-joinable objects, such as tables, views, and **table-valued functions**, are preferable to programmable objects.

Names for the following Parameters can be set:

Call Type Parameter Name

Integer that specifies what Exago needs at time of the call. There are three possible values.

- **0 : Schema** â€” return a data set with no rows (only the schema information).
- **1 : Data** â€” return a full data set.
- **2 : Filter Dropdown Values** â€” return data for the filter dropdown list. The Data Field requested is passed in the **Column** parameter and the filter value is passed in the **Filter** parameter (see below). The direction the user is scrolling the dropdown is passed in the **Sort** parameter as either ASC (scrolling down) or DESC (scrolling up).

Column Parameter Name

- **Call Type = 1:** List of columns required to execute the report, separated by commas.
- When **Call Type = 2:** Column being requested by the filter dropdown.

Filter Parameter Name

- When **Call Type = 1:** The filter string specific to the Data Object being called passed as standard SQL.
- When **Call Type = 2:** The current value of the filter whose dropdown is being requested.

Full Filter Parameter Name

- When **Call Type = 1:** The filter string for the entire report passed as standard SQL.
- When **Call Type = 2:** The Tenant and Row Level filters passed as standard SQL.

Sort Parameter Name

The sort string for the report.

If using call type 2, it can indicate which direction the user is scrolling a filter dropdown. ASC indicates scrolling down or DESC indicates scrolling up.

Data Object Parameter Name v2021.1+

The Data Objectâ€™s name. Can be used in conjunction with the Data Object ID Parameter.

Data Object ID Parameter Name

Id of Data Object being called. For more information see **Calling a Single Web Service/.Net Assembly/Stored Procedure**.

Filter Settings

This article applies to the **Admin Console** > 🔧

General > ⚙ **Filter Settings**.

Introduction

The **Filter Settings** provide control over what filter options are exposed to users and how the dropdowns within filters behave. This document covers several versions of the application. Use the **Viewing content for** dropdown to see only the content relevant to a specific application version.

Settings

Show Group (Min/Max) Filters

Enables/DisablesÂ the **Min/Max Filter** menu. Set to *False* to disable users from using Min/Max filters.

Show Top N Filters v2017.1+

Enables/Disables the **Top/Bottom Filters** menu in the **ExpressView** and **Advanced Report** designer. Top N filters allow users to see the highest or lowest values for a data set. Set to *False* to disable users from using Top N filters.

Allow New Filters at Execution

Controls the creation of new filters when a user is prompted for a filter value at the time of report execution. Set to *False* to disable new filters from being created at execution.

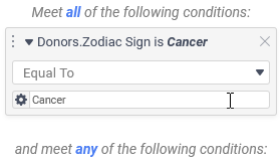
Read Database for Filter Values

Enables/Disables filter dropdowns to contain values from the filter field's data object.

Set to *False* only if retrieving the values for the dropdown will take more than a few seconds.



The Filters tab of the ExpressView Properties Pane with **Read Database for Filter Values** set to *True*



The Filters tab of the ExpressView Properties Pane with **Read Database for Filter Values** set to *False*. Note the lack of the filter dropdown arrow in the bottom figure.

Allow Filter Dependencies

Causes filter drop downs to retrieve values dependent on the filters above them in the menu. Set to *True* to enable. Default value is *False*.

For a full description of how this feature works, review the Filter Dependency section of the Filters article.

> **NOTE**
>
> This setting only works for Data Objects from databases and will not change drop downs from Web Services, .NET Assemblies

and stored procedures.

Show Filter Description

Enables/Disables reports to have description text for the filters menu. The filter description is set in the Description tab of the **New Report Wizard** or from the **Description** item on the Report Designer's Settings menu.

A **Help**

icon will appear in the Filters dialog and display the filter description when clicked.

Default Filter Execution Window

Determines the default type of window that appears when prompting users for filter execution input:

- *Standard* â€" New reports display the standard filter execution window, allowing filters to be modified and new filters to be created.
- *Simple with Operator* â€" New reports display a simplified filter execution window that only allows the operator and value to be changed.
- *Simple without Operator* â€" New reports display a simplified filter window that only allows the value to be changed.
- *Custom* â€" New reports use a custom built filter execution window. When this option is chosen, Allow User to Change Filter Window should be set to *False* and a value be provided for Custom Filter Execution Window. This is a deprecated feature and should not be used in new installations.

Allow User to Change Filter Window

Enables/Disables reports to change the type of filter execution window that is displayed.

Include Null Values for 'NOT' Filters

Indicates to include NULL values for filters with using the operators â€¯not equalâ€™ or â€¯not one ofâ€™.

Custom Filter Execution Window

Specifies a control or URL that contains Custom Filter Execution Window. This is a deprecated feature and should not be used in new installations.

Restore All Default Date Filter Functions v2016.3+

Restores the default **Filter Functions** to the **Extensions** menu.

Restore All Default Formula Functions v2017.2+

Restores the default **Custom Functions** to the **Extensions** menu.

Database Settings

This article applies to the **Admin Console** > **General** > **Database Settings**.

The **Database Settings** allow administrators to adjust how Exago interfaces with the data sources. Additional type-specific settings allow you to specify which driver to utilize when connecting to each data source.

Settings

This document covers several versions of the application. Use the **Viewing content for** dropdown to see only the content relevant to a specific application version.

Database Settings

Database Timeout

A value that is passed to the data provider's CommandTimeout property as Exago queries the business intelligence data source. It is up to the data provider what to do with this value. This timeout has no effect on connections to Storage Management databases.

> **CAUTION**
>
> This value is passed on to the data source provider when the report is executed. Each data source driver, data source or individual database may interpret this setting slightly differently. Consult the official product documentation associated with your data source implementation for complete details how this setting will impact your environment.

> **NOTE**
>
> This setting will also control the maximum number of seconds that a Web Service API method can run. If set to *0* the Web Service API time out will be *infinite*.

Database Row Limit

Maximum number of rows returned on an execution. This only applies to Tables, Views and Functions. Set to *0* to return all rows.

Row Limit Step Size

Maximum number of rows returned on a query. Set to *0* to return all rows. Set greater than *0* to enable **Incremental Loading** for **Advanced Reports** and **ExpressView**. The value determines how many rows are returned for each user-initiated data query.

Disable Non-Joined Data Objects

If *True* users are not able to add Data Objects to a report that do not have a join path with at least one other Data Object on the report. Set to *False* to disable this restriction.

This setting does not have an affect for ExpressView unless **Feature/UI Settings** > **ExpressView Settings** > **Fields Enabled in Data Fields Tree** is set to *Direct Joins Only*.

Enable Special Cartesian Processing

If *True* any one-to-many Joins will cause special processing to avoid data repeating on the report. Set to *False* to disable this behavior.

Aggregate and Group in Database

If *True*, aggregate and grouping calculations will be done in the database when possible. This will provide a performance boost for reports with group sections.

> **IMPORTANT**
>
> Before enabling this, you must ensure that all One-To-Many Joins in your environment are correctly identified and set as *One-To-Many* in the **Join options menu**. If these joins are not properly identified, reports which utilize them will return incorrect aggregate data. See **Database Aggregation** for more information.

Convert Formulas to SQL v2020.1+

If *True*, formula filters and sorts will be converted to SQL when possible. The converted SQL statements will be placed in the WHERE (for formula filters) and ORDER BY (for formula sorts) clauses when querying data for the report. If set to *False*, all formula filtering and sorting will occur in-memory.

For more information, please see the **Converting Formula Sorts to SQL** section below.

> **IMPORTANT**
>
> Due to innate difference between Exago's formula engine and SQL database engines, there may be discrepancies in data returned between two methods. Additionally, because the converted SQL formulas may contain arbitrary input, the connection string must be read-only access. For more information please see the **Database Formulas** article.

Type-Specific Database Settings

Each type of Data Source has the following settings available.

Click the **AutoFill** icon to automatically fill the **Data Provider**, **Table Schema Properties**, **View Schema Properties**, **Function Schema Properties** and **Procedure Schema Properties** fields for each database type. Fields will be filled with data from the drivers provided with the application.

Data Provider

Specify the name that can be used programmatically to refer to the data provider.

This matches the *InvariantName* found as a property of **DbProviderFactories** in the server's machine.config file.

> **NOTE**
>
> The machine.config file is located in the %SystemRoot%Microsoft.NETFramework%VersionNumber%CONFIG directory.

Table Schema Properties

Specifies how to retrieve the schema of tables.

View Schema Properties

Specifies how to retrieve the schema of views.

Function Schema Properties

Specifies how to retrieve the schema of functions.

Procedure Schema Properties

Specifies how to retrieve the schema of procedures.

> **NOTE**
>
> For any of the Schema Property settings you can dynamically refer to properties from the Data Sourceâ€™s connection string by surrounding the property name in @ symbols. For example, “@” will be replaced with the database name from the connection string of the Data Source being queried.

Converting Formulas to SQL

When **Convert Formulas to SQL** is *True*, Exago attempts to push all formulas to the database instead of running them in memory.

For example, a report may be sorted by the following formula, which organizes the data based upon whether or not an *Order* has a *Revenue* of greater than or equal to $1000.

```
=(OrderDetails.Revenue)>=1000
```

This formula sort would be converted to the following SQL statement and appended to the ORDER BY clause of the query during report execution.

```
ORDER BY
 (CASE WHEN dbo.[OrderDetails].[Quantity] * dbo.[OrderDetails].[UnitPrice] >=
  1000 THEN '1' ELSE '0' END) asc
```

Formula Dictionary

All defined Exago formula to SQL statement translations can be found in the dbconfigs.json file, under the **Formula Dictionary** property of each database type. The formula dictionary is comprised of formula names paired with the corresponding SQL syntax translation. In v2021.2+, the and, switch, or, DateAdd, DateDiff and Concatenate functions are represented as JSON objects with zero or more properties. This is helpful when a database has different implementations of internal functions for handling dates and times, for example. See Writing SQL Translations below for more information.

SQL syntax translations may be added in this file for any custom functions that are defined. Translations can be overloaded, meaning multiple versions of a translation with a variable amount of arguments are allowed. Exago will choose the correct translation based on the number of arguments that are passed to it. Functions with an unbounded number of arguments, however, cannot be defined.

See Managing the dbconfigs.json File for more information.

Writing SQL Translations

Each function is paired with its standard SQL translation as a string array, except for and, switch, or, DateAdd, DateDiff and Concatenate in *v2021.2+*.

Arguments are represented as digits enclosed in curly braces, starting from 0, indicating the order in which they are passed from the ExagoBI application. For example, "if": ["(CASE WHEN {0} THEN {1} ELSE {2} END)"] uses the first argument passed to the If() function for the comparison, the second argument if the comparison is true and the third argument if the comparison is false.

> **NOTE**
>
> For the date function translations, the very first argument representing the interval to add or subtract (which would be {0}) is not needed, so only {1} and {2} representing the quantity of the interval, and the date to be changed appear in the translation.

The function translations below use a JSON object instead of an array of strings. This feature requires *v2021.2+*.

And

The and translation has no additional properties. It emits the same SQL regardless of database in use.

Example

```
"and":Â{Â}
```

Or

The or translation has no additional properties. It emits the same SQL regardless of database in use.

Example

```
"or" : { }
```

Switch

The switch translation has no additional properties.

Example

```
"switch" : { }
```

DateAdd

The DateAdd translation may implement 1 or more of the following properties:

- **year** â€" the SQL to add years to the date
- **month** â€" the SQL to add months to the date
- **day** â€" the SQL to add days to the date
- **week** â€" the SQL to add weeks to the date
- **hour** â€" the SQL to add hours to the date
- **minute** â€" the SQL to add minutes to the date
- **second** â€" the SQL to add seconds to the date
- **quarter** â€" the SQL to add quarters to the date
- **default** â€" the SQL when a non-constant is passed for the *interval* argument of the function

Example

```
"dateadd":Â{
ÂÂÂÂ"year":Â"DATE_ADD({2},Â INTERVALÂ{1}Â YEAR)",
ÂÂÂÂ"month":Â"DATE_ADD({2},Â INTERVALÂ{1}Â MONTH)",
ÂÂÂÂ"day":Â"DATE_ADD({2},Â INTERVALÂ{1}Â DAY)",
ÂÂÂÂ"week":Â"DATE_ADD({2},Â INTERVALÂ({1}*7)Â DAY)",
ÂÂÂÂ"hour":Â"DATE_ADD({2},Â INTERVALÂ{1}Â HOUR)",
ÂÂÂÂ"minute":Â"DATE_ADD({2},Â INTERVALÂ{1}Â MINUTE)",
ÂÂÂÂ"second":Â"DATE_ADD({2},Â INTERVALÂ{1}Â SECOND)",
ÂÂÂÂ"quarter":Â"DATE_ADD({2},Â INTERVALÂ({1}Â*Â3)Â MONTH)",
ÂÂÂÂ"default":Â"CASEÂ{0}Â WHENÂ 'yyyÂ THENÂ DATE_ADD({2},Â INTERVALÂ{1}Â YEAR)Â WHENÂ 'mÂ THENÂ DATE_ADD({2},Â INTERVALÂ{1}Â MONTH)Â WHENÂ 'yÂ THENÂ
}
```

DateDiff

The DateDiff translation may implement 1 or more of the following properties:

- **year** â€" the SQL to calculate the difference in years
- **month** â€" the SQL to calculate the difference in months
- **day** â€" the SQL to calculate the difference in days
- **week** â€" the SQL to calculate the difference in weeks
- **hour** â€" the SQL to calculate the difference in hours
- **minute** â€" the SQL to calculate the difference in minutes<.li>
- **second** â€" the SQL to calculate the difference in seconds
- **quarter** â€" the SQL to calculate the difference in quarters
- **default** â€" the SQL when a non-constant is passed for the *interval* argument of the function

Example

```
"datediff":Â{
ÂÂÂÂ"year":Â"TIMESTAMPDIFF(YEAR,Â CAST({1}Â ASÂ DATE),Â CAST({2}Â ASÂ DATE))",
ÂÂÂÂ"month":Â"TIMESTAMPDIFF(MONTH,Â CAST({1}Â ASÂ DATE),Â CAST({2}Â ASÂ DATE))",
ÂÂÂÂ"day":Â"TIMESTAMPDIFF(DAY,Â CAST({1}Â ASÂ DATE),Â CAST({2}Â ASÂ DATE))",
ÂÂÂÂ"week":Â"FLOOR(TIMESTAMPDIFF(DAY,Â CAST({1}Â ASÂ DATE),Â CAST({2}Â ASÂ DATE))/7)",
ÂÂÂÂ"hour":Â"TIMESTAMPDIFF(HOUR,Â{1},Â{2})",
ÂÂÂÂ"minute":Â"TIMESTAMPDIFF(MINUTE,Â{1},Â{2})",
ÂÂÂÂ"second":Â"TIMESTAMPDIFF(SECOND,Â{1},Â{2})",
ÂÂÂÂ"quarter":Â"FLOOR(TIMESTAMPDIFF(MONTH,Â CAST({1}Â ASÂ DATE),Â CAST({2}Â ASÂ DATE))/3)",
ÂÂÂÂ"default":Â"CASEÂ{0}Â WHENÂ 'yyyÂ THENÂ TIMESTAMPDIFF(YEAR,Â CAST({1}Â ASÂ DATE),Â CAST({2}Â ASÂ DATE))Â WHENÂ 'mÂ THENÂ TIMESTAMPDIFF(MONTH,Â CAST
}
```

Concatenate

The Concatenate translation implements these properties

- **mode** â€" determines if the translation will use a function or an operator. Provide either one of these values:
  - *infix-operator* â€" use an operator (e.g. a || b || c) for concatenation
  - *function-variable-args* to use a function (e.g. CONCAT(a, b, c)) for concatenation. When using *function-variable-args*, specify the name of the function with the **functionname** property.
- **functionname** â€" the name of the database function to use for concatenation when **mode** is *function-variable-args*.

Example

```
"concatenate": {
 "mode": "function-variable-args",
 "functionname": "CONCAT"
},
```

For example, in v2021.2+, the entire Formula Dictionary for MySQL is:

```
"FormulaDictionary" : {
 "and": { },
 "if": [ "(CASE WHEN {0} THEN {1} ELSE {2} END)" ],
 "switch": { },
 "or": { },
 "len": [ "CHAR_LENGTH({0})" ],
 "null": [ "NULL" ],
 "dbnull": [ "NULL" ],
 "date": [ "STR_TO_DATE('{0}/{1}/{2}', '%Y/%m/%d')" ],
 "dateadd": {
  "year": "DATE_ADD({2}, INTERVAL {1} YEAR)",
  "month": "DATE_ADD({2}, INTERVAL {1} MONTH)",
  "day": "DATE_ADD({2}, INTERVAL {1} DAY)",
  "week": "DATE_ADD({2}, INTERVAL ({1}*7) DAY)",
  "hour": "DATE_ADD({2}, INTERVAL {1} HOUR)",
  "minute": "DATE_ADD({2}, INTERVAL {1} MINUTE)",
  "second": "DATE_ADD({2}, INTERVAL {1} SECOND)",
  "quarter": "DATE_ADD({2}, INTERVAL ({1} * 3) MONTH)",
  "default": "CASE {0} WHEN 'yyyy' THEN DATE_ADD({2}, INTERVAL {1} YEAR) WHEN 'm' THEN DATE_ADD({2}, INTERVAL {1} MONTH) WHEN 'y' THEN DATE_ADD({2}, INTERVAL {1} D
 },
 "datediff": {
  "year": "TIMESTAMPDIFF(YEAR, CAST({1} AS DATE), CAST({2} AS DATE))",
  "month": "TIMESTAMPDIFF(MONTH, CAST({1} AS DATE), CAST({2} AS DATE))",
  "day": "TIMESTAMPDIFF(DAY, CAST({1} AS DATE), CAST({2} AS DATE))",
  "week": "FLOOR(TIMESTAMPDIFF(DAY, CAST({1} AS DATE), CAST({2} AS DATE))/7)",
  "hour": "TIMESTAMPDIFF(HOUR, {1}, {2})",
  "minute": "TIMESTAMPDIFF(MINUTE, {1}, {2})",
  "second": "TIMESTAMPDIFF(SECOND, {1}, {2})",
  "quarter": "FLOOR(TIMESTAMPDIFF(MONTH, CAST({1} AS DATE), CAST({2} AS DATE))/3)",
  "default": "CASE {0} WHEN 'yyyy' THEN TIMESTAMPDIFF(YEAR, CAST({1} AS DATE), CAST({2} AS DATE)) WHEN 'm' THEN TIMESTAMPDIFF(MONTH, CAST({1} AS DATE), CAST({2} AS
 },
 "day": [ "DAY({0})" ],
 "dayofyear": [ "DAYOFYEAR({0})" ],
}
```

Notes on Behavior

Please take the following information into consideration when utilizing this feature:

- Non-SQL databases (e.g., Mongo, ElasticSearch, Excel) do not support this feature. All sorting for reports using non-SQL databases is done in-memory.
- DB2 is supported by this feature; however, there are limitations regarding what formulas can be translated to SQL when using this database type. In *v2020.1.2+*, most functions are translated to SQL except for DateAdd, DateDiff, MonthName, Time and Mid.
- SQLite supports translating most functions to SQL with the exception of the Financial and Arithmetic & Geometric functions and some of the Database & Data Type and String functions.
- If a sort contains reference to an object from a different data source than the one that is being queried for report execution, then all sorting will occur in-memory.
- If a sort contains reference to an object from an Excel data source, then all sorting will occur in-memory.
- If no sorts exist on a report then no SQL will be appended to the query.
- Boolean expressions will prioritize *False* values when set to ascending and *True* values when set to descending.
- If sorts that cannot be properly translated to SQL exist on a report, then SQL statements will be generated and appended to the database query up to the sort that cannot be translated. For example, if the third sort on a report is untranslatable, then only the first two sorts will be translated and appended.
- Calculations that are pushed to the database may differ in comparison to those run in-memory by Exago. For example, pi (Ï€) is rounded to the 15th digit in Exago but is only rounded to the 6th digit in MySQL by default. These differences in calculations vary depending on the database type and should be taken into consideration when enabling this feature.

Incremental Loading

Â

Incremental loading limits the amount of data that is returned for each database query. This allows users to load reports incrementally, starting with a small set of rows and adding more in steps as desired.

Incremental loading can shorten the amount of time it takes a report to load and be usable. It can also help reduce continuous load on a database which may improve load balancing performance.

To enable it, set **Admin Console** > **General** > **Database Settings** > **Row Limit Step Size** to a value *greater than 0*. This value sets the initial number of rows returned when the report is first executed, as well as the number of rows returned for each subsequent query.

> **NOTE**
>
> We recommend setting **Row Limit Step Size** to a minimum of *1000*.

The step size can be overridden for with a smaller value at the report-level with the Advanced Report Designer's **Report Viewer option Report Row Step Limit**.

Choosing a **Row Limit Step Size** that allows most reports to avoid reaching the limit entirely is ideal since truncated results can affect report accuracy. Summary data may not be accurate if not all of the pertinent rows are available for aggregation. Another consideration is performance. If

there are a lot of slow running reports due to the amount of data being returned, then the need to balance between report completeness in the first execution vs performance by limiting the number of rows returned will need to be considered. Each environment is unique, so the *1000 row* recommendation can be used as a starting point and then fine tuning applied.

Usage

With incremental loading enabled, when an ExpressView is run or a report is run in the Report Viewer (the interactive toolbar must be enabled), the report will only query the first number of rows specified in the configuration setting.

If there are fewer rows than the full data set:

- (*v2021.1+*) a link displays on the toolbar with the message **Showing *X* Results**. Click the link to load more data:
  - **Load X more Records** â€" get the next number of data rows and add them to the existing report
  - **Load All Data** â€" get the remainder of the data set

> **NOTE**
>
> Exporting a truncated report from the Report Viewer generates a file with only the existing rows.

Incremental loading has no effect for Dashboards, Chained Reports, or Scheduled Reports. Exporting from the Report Tree and Advanced Report Designer is not affected by incremental loading.

Infinite Scrolling v2018.1+

With **Incremental Loading** enabled, in certain conditions, scrolling or paging through the Report Viewer will cause additional rows to be loaded automatically instead of needing to click the **Truncated Results Displayed** alert icon each time.

> **NOTE**
>
> When a report qualifies for infinite scrolling, the **Truncated Results Displayed** alert icon and the Report Viewer's interactive sorts and filters dock will be hidden.

Conditions for Infinite Scrolling

Infinite Scrolling requires reports to be designed in a certain manner. If the conditions are not met, users will need to manually step through the rows instead by clicking the icon.

To qualify for infinite scrolling, reports must have:

- a visible Detail section ,
- at least one Sort,
- utilize a data source that supports range selection, such as Oracle, MySQL, DB2, Informix, Microsoft SQL Server, or PostgreSQL.

A report that contains any of the following will be disqualified from infinite scrolling:

- interactive sorts or filters (sorts and filters in the Report Viewer's interactive dock)
- Top N filters
- a CrossTab
- collapsible groups
- cross-source or Cartesian joins
- in-memory aggregate functions (e.g. RunningSum), custom functions, or server events that use the full data set (e.g. OnDataCombined).
- a **Row Limit Step Size** setting that returns less than one full page of data

Range Method

The method used to construct the range limiting SQL statement depends on the Data Source, and can be customized by editing the dbconfigs.json and setting the "RowRangeMethodString" property for each source. See **Constants and Enumerators: wrRowRangeSqlMethod** for valid values.

Scheduler Settings

This article applies to the **Admin Console** > 🔧

General > ⚙ **Scheduler Settings**.

Reports can be emailed or scheduled for recurring automated delivery to an email address. The **Scheduler Settings** are used to configure these services. Before adjusting the settings,Â ensure that the **Scheduler Service** is installed, running, and set to automatically start. For more information see **InstallingÂ ExagoÂ Scheduler**.

The Remote Execution service can be used to move processing to a different server or to provide load balancing across multiple servers. For more information see **Remote Execution**.

Settings

Enable Report Scheduling

Enables/Disables scheduling and remote execution functionality in the application.

If *False,* will override **Show Report Scheduling Option**, **Show Email Report Options**, & **Show Schedule Manager** also to *False*.

If *False*, **Enable Remote Report Execution** is ignored.

Show Report Scheduling Option

Enables/Disables the

Schedule Report option when right-clicking on a report in the Report Tree. Set to *False* to disable users from scheduling reports.

Show Email Report Options

Enables/Disables the

Email Report option when right-clicking on a report in the Report Tree.

Show Schedule Reports Manager

Enables/Disables the **Schedule ManagerÂ** icon on the Main Menu.

Show Schedule No End Date Option

Enables/Disables users ability to create a schedule that never ends. Set to *False* to force users to set an end date limit to the schedule.

Show Schedule Intraday Recurrence Option

Enables/Disables options in the Recurrence tab of the Schedule Wizard to allow a schedule to repeat throughout the day.

Scheduler Manager User View Level

Controls what information each user can see in the Schedule Manager. These levels utilize the Parameters companyId and userId. There are three possible values:

- **Current User:** Can only view and delete report jobs that have been created by that user. This setting will hide the Host and Company Id columns of the Schedule Manager.
- **All Users in Current Company:** User can only view and delete report schedules for their company. This setting will hide the Host column of the Schedule Manager.
- **All Users in All Companies:** User can view and delete report schedules for all companies (administrator).

For more information, see **User Identification**.

Email Scheduled Reports

Set to *False* to have the Scheduling Service save reports to a file repository instead of attaching them to emails. For more details see **Saving Scheduled Reports to a Repository**.

Enable Batch Reports

Set to *True* to allow users to schedule reports which are filtered separately for each recipient user. Batch reporting requires a table or other data structure containing email addresses for the intended recipients associated with a key used to filter the reports. For more information see **Scheduling Reports**.

Show Schedule Delivery Type Options

Enables/disables user option to choose the output type (i.e. email or save report to repository) with each schedule. When *True*, the default value will reflect whatever is set in the **Email Scheduled Reports** setting.

Use Secure Scheduler Remoting Channel

Enables/disables encrypting of data Set to true to cause data sent to remote schedulers to be encrypted. Each scheduler config file must also have <secure_channel> set to true.

Schedule Remoting Host

Sets the protocol, server and port to the **Scheduler Service**. Separate multiple remoting hosts with a comma or semicolon. Click the **Test ConnectionÂ**

icon to test the connection between the Web Application and the Scheduler Services.

The protocol and port must match the **Scheduler Service**'s channel_type and port configuration settings, respectively.

Examples

```
http://localhost:2001
```

```
http://SchedulerOne:2001,tcp://SchedulerTwo:2001
```

Enable Remote Report Execution

Set to *True* to enable Remote Execution of reports by one or more Scheduler Services. **Enable Report Scheduling** must also be *True* and a value must be provided for the **Remote Execution Remoting Host**.

Set to *False* to disable this behavior.

For more information, review the Remote Execution article.

Enable Execution Cache

Enables/disables users to use an execution cache for their reports. An execution cache refreshes report data on a schedule, and report execution calls use the cached data instead of querying the data source each time the report is executed.

This dropdown will only display *False* until all conditions in the Enabling Execution Caching section of the Execution Caching article are met. Then, this setting can be changed to *True* to enable the feature.

User Cache Visibility Level

Controls what visibility permissions users can assign to **Execution Caches**. Available levels utilize the parameters **companyId and userId**. There are three possible values:

- **User:** Users can only view cached report data for caches that they have created.
- **Company:**Â Users can permit cached data to be visible to all other users in the user's company. Users can also select the **User** option.
- **Global:** Users can permit cached data to be visible to all other users for all other companies. Users can also select the **Company** or **User** options.

Enable Access to Data Sources Remotely

When *True*, all calls, including populating Filter Value dropdowns and retrieving Data Object/Data Object schemas made to the Data Sources are handled by the **Remote Execution Remoting Host**, or the **Schedule Remoting Host** if no **Remote Execution Remoting Host** is defined.

When *False*, the Scheduler Services will only make calls to the Data Sources during report execution. All other calls are handled by the Web Application.

Enabling this feature is helpful when whitelisting connections to the Data Sources, and connections should only originate from the the designated Remote Execution Remoting or Schedule Remoting Hosts as opposed to the Web Application servers; or when the Scheduler Services and the databases are on the same firewall level but the Web Application is not.

Remote Execution Remoting Host

Specifies the Scheduler Services to use for remote execution. Separate multiple servers with commas or semicolons. Click the **Test Connection**Â

icon to test the connection between the Web Application and the Scheduler Services.

The protocol and port must match the **Scheduler Service**'s channel_type and port configuration settings, respectively.

Example

```
http://MyHttpServer1:2001,tcp://MyTcpServer:2001
```

Custom Queue Service

Specifies the web or assembly queue service for custom scheduler management and load balancing. See **Scheduler Queue** for details. Click the **Test Connection**Â

icon to test the connection between the Web Application and the Queue Service.

Delete Schedules upon Report Deletion

When a report is deleted corresponding schedules can be deleted automatically by Exago. Set to *True* to enable this behavior.

Default Email Subject

Set a default subject that will be displayed in the schedule report wizard.

Internal Parameters such as @ may be utilized in this area.

Default Email Body

Sets a default body that will be displayed in the schedule report wizard.

Internal Parameters such as @ may be utilized in this area.

Password Requirement (for pdf and excel documents)

Requires a password for PDF or Excel file export. This parameter can be made up of the following values:

- **A:** requires an upper case letter for each **â€¯Aâ€™**
- **a:** requires a lower case letter for each **â€¯aâ€™.**
- **n:** requires a numeric character for each **â€¯nâ€™**
- **4:** password must have at least **4** characters

For example, â€¯AAnna6â€™ would require a password of at least six characters with 2 capitals, 1 lower case and 2 numeric characters.

Custom Scheduler Recipient Window

Provides URL, height and width for custom Scheduler Recipient window. This is a deprecated extensibility feature and should not be used in new

installations.

Show “Reply To” Field in the Scheduler “Recipients” Tab

Displays the email specified in the userEmail parameter in the Recipients tab when scheduling a report. Any replies to the scheduled report will be sent to this email address instead.

Other Settings

This article applies to the **Admin Console** > 🔧

General > ⚙ **Other Settings**.

This document covers several versions of the application. Use the **Viewing content for dropdown** to see only the content relevant to a specific application version.

Excel Export Target

Choose the type of Excel export you would like. Choosing 2003 will automatically split the workbook into multiple worksheets when Excelâ€™s row limit is reached.

> **NOTE**
>
> Linux does not support setting the Excel export target to 2003.

External Interface

Provide a Web Service URL or .NET Assembly to interface with the external module. This is a deprecated extensibility feature that should not be used in new installations.

Enable Paging In the Report Viewer

Controls when data for Report Viewer output is sent to the client. Set to *True* to send data as each page is requested.Â If this is set to *False*, it restricts the interactive HTML hide/show column option in theÂ **Report Viewer**.

> **NOTE**
>
> When enabled, it will cause multiple requests from the server.

Set to *False* to send all the data to the client browser at once.

Renew Session Automatically

This setting is used to bypass the session timeout property set in web.config. Set to *True* to send a server side AJAX callback every two minutes to keep the session from expiring.

> **NOTE**
>
> This will only work if the timeout period set in web.config is greater than two minutes.

Write Log File

> **IMPORTANT**
>
> Setting the log level to **DEBUG** in a production environment can have unintended performance consequences. Only use **DEBUG** level logging in production temporarily and under advisement from the Exago Support, Services or Development teams.

Set the level of detail logged in the Exago log file. See **Application Logging** for information about the different log levels. This setting will be overridden by a log4net.config file.

Enable Debugging

Set to *True* to enable debugging. For more information see Submitting a Debug Package.

Max Report Execution Time (minutes)

Specify how long, in minutes, reports should run before timing out. Default is *240* minutes (4 hours).

Max Chained Report Collation Executions v2021.1+

When executing a collated chained report, the application will run each report for each filtered group. A non-collated chained report containing two reports will only execute those two reports. A collated chained report with the same two reports sorted on a field with 100 values will execute 200 individual reports.

Set the maximum number of individual report executions that are allowed when exporting a collated Chained Report. Setting a limit will prevent very long chains from being executed that could cause performance reductions. If the number of executions exceeds this limit, the chain will be cancelled and an error message displayed to the user (and an e-mail sent to the <error_report_to> address for reports executed by a Scheduler Service).

The default value is *0*, indicating no limit to the number of collated report executions.

Maximum Age for Temp Files

Determines the maximum number of minutes a temp file can exist before Exagoâ€™s automatic cleanup removes it.

> **IMPORTANT**
>
> Understand that setting the maximum age too low may cause an error, as users might spend some time viewing a report executed in HTML, which uses AJAX to read temp paging files.

The default value is *1440* minutes (1 day). The minimum this value can be set to is 30 minutes.

Enable Web Service/Assembly Data Mapping

Allows Web Service and .NET Assembly methods to replace Data Field names.

Limit Reports and Visualizations to One Category

Limits reports and visualizations to Data Objects within a single category. Set to *true* to restrict reports and visualizations to one category, or *false* to allow Data Objects from multiple categories.

Run Aggregate Functions at Record Level by Default v2019.1.8+

The recordLevel argument of all aggregate functions will default to this value if no value for recordLevel is passed when the aggregate function is called. If *True*, aggregate functions run at the record level (including duplicates). If *False*, aggregate functions run at the entity level which is the default behavior.

By default, set to *True*

Cache External Services

Caches external Web Services and .NET Assemblies. Setting to *False* may reduce performance due to loading/unloading of services.

Global Schema Access Type

Specifies whether to query the Data Source for an objectâ€™s schema or to read it from *Column Metadata*. See Retrieving Data Object Schemas for more information.

Allow Multiple Sessions

Allows multiple sessions of Exago per user. Set to *True* to enable this behavior.

Allow MD5 Hashing on FIPS Server

Allows a FIPS-compliant server to encrypt PDF files by using an alternate MD5 library to the built-in System.Cryptography.

Write BOM to CSV Files v2019.1.1+

Attaches a Byte Order Mark (BOM), an optional abstract character used to declare and recognize Unicode encoding of a text file, to the beginning of CSV file exports. Set toÂ *True* to enable.

‘LoadImage’ Cell Function Parameter Prefix

A string that is prepended to the **LoadImage** function when the report is run. This setting allows an administrator to hide the report path of images on your server.Â This field is ignored for images loaded from a database.

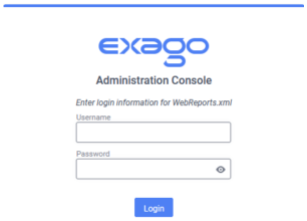
Ignore Inaccessible Report Folders

If *False*, Exago throws an error message if a folder has an accessibility issue. Set to *True* to ignore the error and hide the inaccessible folder.

User ID

Sets the User ID necessary to gain access to the Administration Console and REST API. While a blank user ID is allowed, it is not recommended. For best results, set both the User ID and the Password.

When set, the login screen appears when accessing the Admin Console. For more information, see the About the Administration Console article.

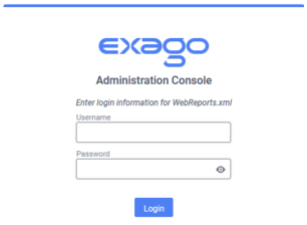


Admin Console login dialog

Password

Used in conjunction with User ID to gain access to the Administration Console and REST API. While a blank password is allowed, it is not recommended. For best results, set both the User ID and the Password.

When set, the login screen appears when accessing the Admin Console. For more information, see the About the Administration Console article.



Admin Console login dialog

Confirm Password

Confirm the **Password** by entering it again in this field.

Debug Password

A password that enables clients to send a debug package directly to Exago Inc. Leave blank to disable Debug Extraction. When set to *True*, correct permissions must be set on the ./Debug Folder. For details see the **Submitting a Debug Package** article.

REST Key

Set a key to authenticate REST requests. Review the Authorization section of the Introduction to REST article for more information.

Exago Expiration Date

A date when users will no longer be able to access Exago.

Custom Code Supplied by Exago

Used for custom functionality.

License Key

Used to activate purchased add-on Exago features.

Admin Console Password Encryption

Beginning with *v2017.3*, the **Admin Console** password is now encrypted by default when entered into the Admin Console or when set through an API call. This is done to increase the security of credential storage by preventing plain text passwords from being saved to disk in the unencrypted version of the configuration file.

> **CAUTION**
>
> If updating from an older version, existing passwords will not be encrypted automatically.

There are two ways to set an encrypted Admin Console password: Using the **Admin Console** or the **API**.

Using the Admin Console

1. Browse to the Admin Console
2. Navigate to **General** > **Other Settings**.
3. Enter the desired password into the **Password** and **Confirm Password** fields



4. Click **Apply** or **Okay**

To verify that the configuration contains the encrypted password, open the XML config file in a text or XML editor and locate the <password>

node.

```
<userid>alex</userid>
<password>[wOrZAqGMuU6+ieAVN60gc31AnUzg5SU+AfShdp5obbDW]</password>
```

The value should be an encrypted string surrounded by brackets [ ].

Using the API

To add an encrypted password to a programmatically generated config file:

```
api.General.Password = api.General.EncryptPassword("mypassword");
api.SaveConfigToFile(); // Save the configuration file to disk
```

To verify whether two passwords match:

```
bool IsMatch = api.General.CheckPassword("mypassword", api.General.Password);
```

To verify if an existing password is encrypted:

```
bool IsEncrypted = api.General.IsHashedPassword(api.General.Password);
```

Additional information

Password encryption is one-way. An encrypted password cannot be decrypted into plain text.

The encryption algorithm used is SHA-256. Passwords are salted.

> **IMPORTANT**
>
> We still recommend that the plain text config file (e.g. *WebReports.xml*) is removed in favor of the encrypted config file (e.g. *WebReports.xml.**enc***) in a production environment. See **Security Checklist** for more information.

Data Sources

This article applies to the **Admin Console** > **Data** > **Sources** settings.

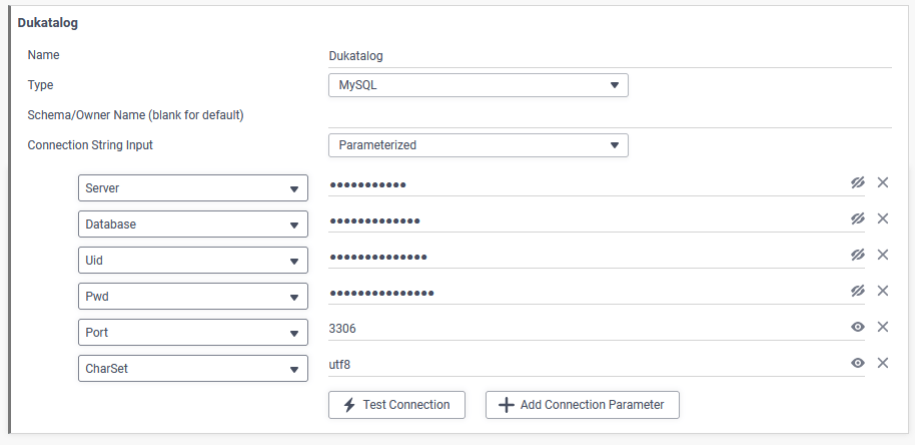
Data Sources establish the connection between Exago and a database or a web service. Although typically only one database is used, Exago can join data from different sources into a single report.

> **NOTE**
>
> To utilize some types of data sources, the appropriate driver may need to be downloaded and installed separately from the Exago install process. Refer to Data Source Drivers.

Creating, Editing or Deleting Data Sources

- To **add** a new data source clickÂ **Sources** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
- To **edit** a source either:
  - double click it
  - select it and click the **Edit**Â icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** a source either:
  - select it and click the **Delete**Â icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new objects click the **Apply** or **Okay** buttons.

Data Source Properties



Creating a new Data Source in the Admin Console in v2021.2+

Each data source must have the following:

Name

A friendly name for the Data Source that will be referenced in the application.

Type

The type of source being used. Valid types include:

- Relational databases
  - **SQL Server** â€" Microsoft SQL Server
  - **MySQL**
  - **Oracle**
  - **Postgres** â€" PostgreSQL
  - **DB2** â€" IBM db2
  - **Informix** â€" IBM Informix
  - **ODBC** â€" ODBC Driver
  - **SQLite** v2021.1.8+
- Non-relational databases
  - **MongoDB** v2018.2+
  - **ElasticSearch** v2018.2+ â€" ElasticSearch/ELK database (For more information see **CData Drivers**.)
- Data warehouses
  - **Redshift** v2019.1.11+
  - **Snowflake** v2019.1.9+
  - **Google BigQuery** v2021.1.10+ (For more information see **CData Drivers**.)
- Other
  - **File** â€" XML or Excel file (For more information see **Excel and XML Files**.)
  - **.NET Assembly** â€" .NET Assembly DLL (For more information see **.NET Assemblies**.)
  - **MS OLAP** â€" OLAP (For more information **OLAP and MDX Queries**.)

Schema/Owner Name (blank for default)

If using schema to provide multi-tenant security, provide a database schema. Otherwise, leave this field blank. For more information, review the **Multi-Tenant Environment Integration** article.

Connection String Input

Describes how Exago shall connect to the Data Source. Typically, a server name or IP address, database name and login credentials are specified here. Different Data Source types have different connection string requirements, although the most common connection string parameters are:

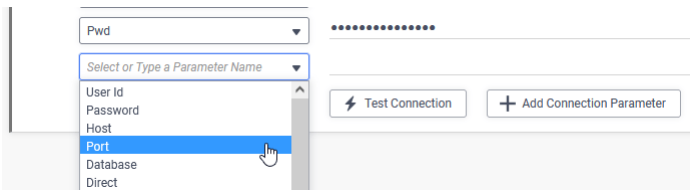
- **Server** â€" the server host name or IP address where the database is physically located
- **Database** â€" the name of the database on the **Server**
- **User ID** and **Password** â€" the user name and password credentials to access the **Database**

Choose either the integrated Connection String Builder (*Parameterized* mode), or to provide a connection string manually (*String* mode) from the **Connection String Input** dropdown. Switching between *Parameterized* and *String* is possible, and Exago will remember the last mode selected. The Data Source connection is still stored as a standard connection string in the configuration file regardless of mode selected.

> **NOTE**
>
> Some Data Source types only support *Parameterized* or *String* mode. In these cases, Exago will select the correct mode and disable the **Connection String Input** dropdown.

In *Parameterized* mode, Exago queries the database driver for all of the available connection string options/parameters. Then, these parameters may be added one at a time in key-value format to build a complete connection string.

> **NOTEIF THE DATABASE DRIVER DOES NOT RETURN ANY CONNECTION STRING PARAMETERS, THE PARAMETER NAMES MUST BE MANUALLY TYPED INSTEAD OF CHOSEN FROM A DROPDOWN LIST.**



Adding the *Port* connection string parameter to a Data Source by choosing its name from the **Parameter Name** dropdown

- To **add** a new connection string parameter to the Data Source:
  1. Click theÂ + **Add Connection Parameter** button.
  2. Enter the name of, or choose the name of a parameter from the dropdown list.
  3. Provide a value for the parameter in the space provided.

- To **hide/show** a connection string parameter, click the **Make Connection String VisibleÂ** or **Connection String HideÂ** icons accordingly.
- To **remove** a connection string parameter, click the **DeleteÂ** icon at the end of the line. This icon is only visible when there are two or more parameters for a Data Source.

Connection strings vary by type:

Type	Connection Strings
mssql, oracle, postgres, mysql sqlite (*v2021.1.8+*) and olap	Refer to connectionstrings.com for database connection strings. If using **Integrated Authentication** from an IIS Application Pool to connect to a remote MSSQL or Microsoft SQL Server Analysis Services (SSAS) server, review these MSDN and Stack Overflow articles to ensure the correct credentials are sent to the SQL server. The database driver specified in Database Settings determines the character set used in the application. If the data in the data source uses a different character set, it should be explicitly set in the connection string. For PostgreSQL data sources using the Npgsql driver and all lowercase schema names, set the UseDelimitersOnSchema directive in dbconfigs.override.json to true for best results. See the Overriding dbconfigs Information section of the Managing the dbconfigs.json File article for more information.
assembly	Required parameters: • *assembly* â€“ The full path of the assembly name. • *class* â€“ The class name in the assembly where the static methods will be obtained.
file	Requires the physical path to the Excel or XML file and the file type. **Example:** File=C:example.xls;Type=excel;
mongodb	For more information see CData Drivers
elasticsearch	For more information see CData Drivers
snowflake	For more information see CData Drivers
redshift	For more information see CData Drivers
Google BigQuery	For more information see CData Drivers

- Click the **Test Connection** icon to verify the connection succeeds.
- Click the **Make Connection String Visible** or **Connection String Hide** icons to show/hide the connection string in the **Data Source** tab.

Data Source Drivers

This content has been moved to the Data Source Drivers article.

.NET Assemblies

.NET Assemblies can be used as data sources. This is possible when the .NET Assemblies underlying methods are setup as data objects.

An advantage of doing this is being able to use high-level language to manipulate the data being reported on at run-time. The main disadvantage is not being able to take advantage of the database to perform joins with other data objects; data from methods can still be joined, but the work to do this is done within Exago. For more information see **Note about Cross Source Joins**.

Parameters

Parameters are passed from Exago to .NET Assemblies. Three types of parameters can be passed but only Call Type is required.

Call Type (required)

Integer that specifies what Exago needs at the time of the call. There are three possible values. You may specify the name of this parameter in the **Programmable Object Settings** of theÂ

Â **General** section.

- **0 : Schema –Â** returns a DataSet with no rows.
- **1 : Data –** returns a full DataSet.
- **2 : Filter Dropdown Values** â€“ returns data for the filter dropdown list. The **Data Field** being requested is passed in the column parameter. The filter type is passed in the filter parameter (see below).

Column, Filter and Sort Strings

To optimize performance Exago can pass user-specified sorts and filters to the .NET Assembly. This process reduces the amount of data sent to Exago. If these parameters are not used, all of the data will be sent to Exago to sort and filter. Column, filter and sort strings are sent as standard SQL. You may specify the name of these parameters in the

General >Â **Programmable Object Settings** section.

Custom Parameter Values (optional)

Additional parameters can be specified to be sent to individual methods in the **Data Object Menu**.

> **IMPORTANT**

To increase performance, when a .NET Assembly is first accessed it is compiled and cached within Exago. Therefore, Exago will not be aware of any changes within .NET Assembly. If the assembly is subsequently changed, reset the internal cache by clicking the **Test Connection** icon restarting the web server.

NOTE

If an **Exago .NET API** application needs to access reports with an assembly data source, it must include a reference to the assembly WebReportsAsmi.dll.

SessionInfo (optional) v2016.2+

Session state variables. See **SessionInfo** for more information.

.NET Assemblies

It is important to note that when a connection string for .NET Assembly is set the class name must match the name of the class where the static methods will be searched. UNC or absolute paths may be used. Make sure that the assembly has read privileges for the IIS user running Exago. Below is an example of a .NET Assembly connection string:

```
assembly=MyServerNameMyShareNameMyAssembly.dll;class=Main
```

.NET Assembly methods must be static. Below is an example of a .NET Assembly method.

```
public class Main{  public static DataSet dotnet_optionees(int callType, string columnStr, string filterStr, int myCustomParameter) {   switch (callType)   {    case 0:     // return schema
```

Method signature using SessionInfo (*v2016.2+*):

```
public class Main{  public static DataSet dotnet_optionees(WebReports.Api.Common.SessionInfo sessionInfo, int callType, string columnStr, string filterStr, int myCustomParameter)
```

Excel and XML Files

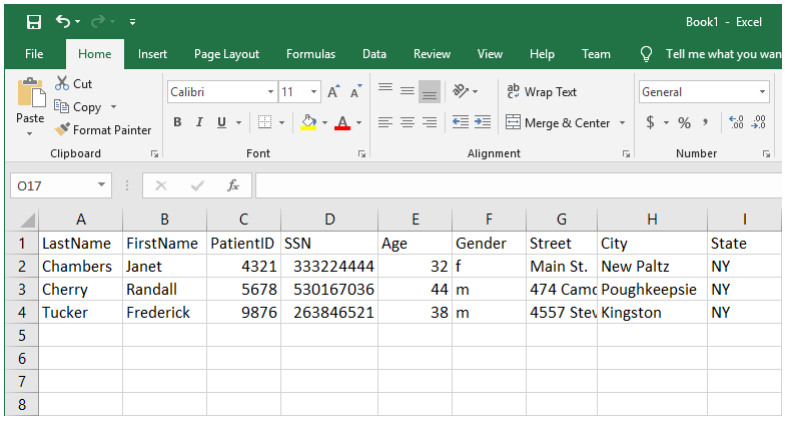
Exago can read Microsoft Excel workbooks and XML files as Data Sources.

CAUTION

Excel workbooks and XML files will have less speed, performance, and security of a database. Using Excel and XML files is recommended only if the dataset is small or if the information is only available in this format.

Excel

Each worksheet in the Excel file will be read as a separate table with the sheet name as the table's name. The top row will be read as the column headers, and the remaining cells will be read as the data. Do not leave any blank rows or columns.



	A	B	C	D	E	F	G	H	I
1	LastName	FirstName	PatientID	SSN	Age	Gender	Street	City	State
2	Chambers	Janet	4321	333224444	32	f	Main St.	New Paltz	NY
3	Cherry	Randall	5678	530167036	44	m	474 Camc	Poughkeepsie	NY
4	Tucker	Frederick	9876	263846521	38	m	4557 Stev	Kingston	NY
5									
6									
7									
8									

Connection String Example

```
File=C:example.xls;Type=excel;
```

XML

The XML document must begin with the schema. After defining the schema the data must be placed into the appropriate tags. For reference see the working example below:

```
<?xml version="1.0" encoding="UTF-8"?><ExagoData>  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" id="E
```

Connection String Example

```
File=C:example.xml;Type=xml;
```

OLAP and MDX Queries

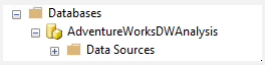
Exago can query OLAP Data Sources using MDX Queries. OLAP Data Sources and Objects are identical to a regular data base type object, with the following exceptions:

- Data Objects belonging to OLAP type Data Sources must have their **Schema Access Type** set to *Metadata* and must have **Column Metadata** set for all fields.
- OLAP Data Objects will always be MDX Queries written in the **Custom SQL Object** dialog. These queries are passed directly through to the data source. Exago simply passes through the MDX query to the data source and does not modify it in any way. Therefore, all filtering and sorting are done in memory, tenanting at the data object level and database aggregation do not work with OLAP data sources.

Connecting to a Microsoft SQL Server Analysis Services (SSAS) data source will use **Integrated Authentication**. See the notes in the Connection String section above how to handle this situation. In a general sense, the IIS Application Pool will need to login with a credential assigned in an appropriately provisioned SSAS role on the data source.

Sample Connection String

```
Data Source=SERVER1MSSQLSERVER2014;Catalog=AdventureWorksDWAnalysis;
```

> **TIP**
>
> The **Catalog** parameter should match the **Database** name in Microsoft SQL Server Manageent Studio.
>
> 
>

Using Azure based SSAS

In order to use an Azure based Microsoft SQL Server Analysis Services Database (SSAS) you need the most up to date ADOMD.NET driver, which is available from Microsoft. Then, in order to properly configure Data Object entities through joins, the following information must be provided:

- The entity containing the primary key should be specified in the “From” section.
- The entity containing the foreign key should be specified in the “To” section.
- The IDs used in the join configuration must be included in the metadata fields and should be available in the design model.

ODBC Drivers

Exago can use ODBC drivers to connect to data sources. When connecting to an ODBC data source, an extra option will appear to set the Column Delimiters. The delimiter character depends on which type of data base you are connecting to.

Examples

MySql

```
' (grave accent)
```

MsSql, **OLAP**

```
[] (brackets)
```

DB2, **Informix**, **Oracle**, **Postgres**, **Sqlite**

```
" (quotation marks)
```

If you don’t know which delimiter character to use, contact your database administrator.

> **WARNING**
>
> If your data objects have spaces in their names, you must set the correct delimiter in order to access the data. Otherwise, improper SQL will be generated and you will see errors or erroneous data.

CData Drivers

As of *v2018.2+*, Exago allows for additional data source types through CData ADO.NET drivers. For more information, see the article on **CData Drivers**.

Data Source Drivers

Supported Driver List

Below is a list and the associated links for recommended ADO.NET drivers for each type of data source.

- **SQL Server**
  - No external ADO.NET driver needed
- **Oracle**
  - Oracle ODAC Connector

    - Oracle.ManagedDataAccess.Client
- **MySQL/MariaDB**
    - Devart Connector
    - MySql.Data.MySqlClient
- **PostgreSQL**
    - Devart Connector
    - Npgsql
- **DB2/Informix**
    - IBM Data Server Driver Package â€" 5.exe or newer
- **SQLite** v2021.1.8+
    - No external ADO.NET driver needed
- **Elasticsearch, Redshift, Snowflake, MongoDB, Google BigQuery**
    - CData ADO.NET drivers

Exceptions, Caveats and Notes

MySQL on Linux v2019.2+

As of *v2019.2* the Exago Linux Installer does not install a MySQL ADO.NET driver at the time of installation. Instead, clients wishing to use a MySQL data source for either reporting or for Storage Management will need to provide their own.

Exago has provided wrappers around two popular MySQL data drivers that clients may choose to install on their own.

- Devart dotconnect free edition
- MySQL ADO.NET

Contact your Customer Success Manager or the Exago Support Team for assistance. Install the driver by extracting the contents of the download and then running either installMySql.sh or installDevartMySql.sh as root. Provide the Exago installation path to the installer script. For example:

```
sudo ./installDevartMySql.sh /opt/Exago
```

Once installed, update the **Admin Console** > **General** > **Database Settings** to reflect the new data provider.

If using the Storage Management command-line transition utilities add this line to LoadReportsToDb.exe.config between <system.data> <DbProviderFactories>:

```
<add name="dotConnect for MySQL" invariant="Devart.Data.MySql" description="Devart dotConnect for MySQL" type="Devart.Data.MySql.MySqlProviderFactory, Devart.Data.MySql, Versi
```

Oracle.ManagedDataAccess.Client and Npgsql Installation via Nuget

The **Oracle.ManagedDataAccess.Client** driver for Oracle Data Sources and the **Npgsql** driver for PostgreSQL Data Sources are installed via the Nuget Package Manager. Therefore, manual editing of Exago's web.config is required to give Exago access to the drivers.

Using Nuget is beyond the scope of Exago's documentation, but more information on installation of these drivers can be found at the links below. Alternatively, contact Exago's Support Team for assistance.

- Using NuGet to Install and Configure Oracle Data Provider for .NET
- Npgsql Getting Started

CData Drivers

As of *v2018.2*, additional database connection types are available using ADO.NET drivers provided by CData Software Inc., in partnership with Exago Inc., to connect to database types not natively included in the application.

Exago BI now officially supports the following database types out-of-the-box via paid CData driver support:

- Elasticsearch v2018.2+
- Google BigQuery v2021.1.10+
- MongoDB v2018.2+
- Redshift v2019.1.11+
- Snowflake v2019.1.8+

> **NOTE**
>
> The Exago BI web server must have full permissions for the **Drivers** folder in the Web Application installation directory.

CData drivers are initially available for a 30 day trial period. Using a CData driver in production requires a paid license. Once a successful connection is made to a data source using a CData driver, a 30 day trial will begin. At the end of the 30 days, a license will have to be obtained to continue using the driver.

Licensing and Support

CData licenses for use with Exago BI are purchased from Exago Inc. (not from CData Software Inc.). Submit a support request to purchase a license key for a CData driver.

> **NOTE**
>
> The required license is an internally generated license specific to Exago, and is not interchangeable with a standard CData license. This license is CData version specific and will need to be re-generated when changing CData driver versions.

Exago Inc. may refuse support for CData drivers that are not included with the Exago BI installation and not licensed through Exago Inc. If such support is requested, Exago may require a purchase of service hours.

License Model

CData driver licenses are **not included** with the base Exago BI license, and are purchased separately or as add-ons to the Exago license agreement. Purchasing a CData driver license grants youÂ **one provider per product per year**. More specifically:

One (1) CData driver license applies to the following:

- One CData driver provider (e.g. “Snowflake ADO.NET Provider”)
- One product in which Exago BI is embedded (products are defined as per your individual Exago license agreement)
- One or more Exago instances comprising the product (an instance is an installation of the Web Application or Scheduler Application)
- One or more data sources (databases) of the provider type and pertaining to the product
- One calendar year of usage, renewed annually

This information may differ according to the terms of your individual license agreement. Please make sure to verify your specific terms with your business department.

Adding Additional Data Source Types

A list of additional data sources supported by CData can be found on the CData website. Not all sources listed are supported by Exago BI. If you are interested in using a CData driver that is not provided with the Exago installation, please submit a support request or contact your Exago **Customer Success Manager**.

When adding an additional CData driver, the driver must be placed in the **Drivers** folder of the web application installation directory.

When setting up an additional data source, create a new configuration in the **cdataconfig.json** configuration file. They will be available as data source types in the **Admin Console** when creating a new connection.

When a full license key is obtained, the value must be placed in the License key value pair in the object within **cdataconfig.json**. The server hosting the application will not have to be restarted. The next time the application is accessed, the connection will be available with the new license.

Configuration File

When using a CData driver, the application will refer to a file in the installation directory under ‘Config/Other’ named **cdataconfig.json**. The application will look to this file for information about the CData driver and how to use it.

Initially, the license key value for built-in CData sources is “TRIAL”.

cdataconfig.json Definition

This configuration file is comprised of JSON objects that contain specific information for how the data type will appear and behave in the application. These objects are explained in more detail in the table below.

More information about JSON and how to use it can be found in the Using JSON article.

Attribute Name	Description
Name	The outer label for the JSON object is the simplified name for the driver being added.
FriendlyName	The name of the data source type that will appear in the Admin Console
License	The license value to use the driver. By default this value will be *TRIAL*.
DateFormat	The specified Date format of dates returned by the data source.
Delimiter	Characters used to surround table names when passing SQL.
TopRowLimitSQL	Top row limiting syntax. For limiting the rows returned when using a Top filter.
BottomRowLimitSQL *(optional)*	Bottom row limiting syntax. For limiting the rows returned when using a Bottom filter.
dbInfo	This is another JSON Object with properties that define schema for the data source. These are equivalent to the corresponding fields in the **Admin Console Database Settings**.
Provider	This is the name of the library that is obtained from CData for the database driver. This will always have the format of System.Data.CData.*{name}* where *{name}* is the proper name of the source type (e.g. Snowflake).
TableSchema	Table schema retrieval properties.
ViewSchema	View schema retrieval properties.
ProcedureSchema	Procedure schema retrieval properties.

Example Configuration File

```
{
 "mongodb":
 {
  "FriendlyName":"MongoDB",
  "License":"TRIAL",
  "DateFormat":"YYYY-MM-DD HH:mm:ss",
  "Delimiter":"|",
  "TopRowLimitSQL":"",
  "BottomRowLimitSQL":" LIMIT {0}",
  "dbInfo":{
      "Provider":"System.Data.CData.MongoDB",
      "TableSchema":"Collection:"Tables",
          Restrictions:[null, "@[email protected]"],
          NameColumn:"TABLE_NAME",
          SchemaColumn:null",
      "ViewSchema":"Collection:"Views",
          Restrictions:[null, "@[email protected]"],
          NameColumn:"TABLE_NAME",
          SchemaColumn:null",
      "ProcedureSchema":"Collection:"Procedures",
          Restrictions:[null, "@[email protected]"],
          NameColumn:"ROUTINE_NAME",
          SchemaColumn:null"
  }
 }
}
```

Sample Connection Strings

Elasticsearch

```
Server=[SERVER IP OR HOSTNAME];Port=[PORT];User=[USER];Password=[PASSWORD]
```

The default port is 9200.

Refer to CData's ADO.NET Provider for ElasticSearch documentation for a complete list of connection string options and further details.

Google BigQuery

```
InitiateOAuth=GETANDREFRESH;ProjectId=[GCP PROJECT];DatasetId=[GCP PROJECT];OAuthJWTCertType=GOOGLEJSON;OAuthJWTCert=[PATH TO GOOGLE SERVICE ACCOUNT
```

- InitiateOAuth should always be set to GETANDREFRESH
- OAuthSettingsLocation should always be set to a local server directory where OAuth values can be saved.
- UseConnectionPooling should explicitly be set to False unless using connection pooling.
- pollinginterval should be set higher than 1

Refer to CData's ADO.NET Provider for Google BigQuery documentation for a complete list of connection string options and further details.

MongoDB

MongoDB data sources should utilize the FlattenArrays and FlattenObjects connection string parameters for best results.

```
Server=[SERVER IP OR HOSTNAME];Port=[PORT];Database=[DATABASE]
```

The default port is 27017.

Refer to CData's ADO.NET Provider for MongoDB documentation for a complete list of connection string options and further details.

Redshift

```
Server=[SERVER IP OR HOSTNAME];Port=[PORT];Database=[DATABASE];User=[USER];Password=[PASSWORD]
```

The default port is 5439.

Refer to CData's ADO.NET Provider for Redshift documentation for a complete list of connection string options and further details.

Snowflake

In *v2021.1.0â€“v2021.1.4*, provide a directory and file name in the CredentialsLocation parameter that is writable by the web server. The CData driver will create the file. For example: CredentialsLocation=C:/Temp/SnowflakeCredentials.txtor CredentialsLocation=/var/tmp/SnowflakeCredentials.txt. A unique location must be provided for each Snowflake user account.

For versions *v2021.1.0â€“v2021.1.4:*

```
Url=[SNOWFLAKE_URL];Warehouse=[WAREHOUSE_NAME];Database=[DB_NAME];User=[USER];Password=[PASSWORD];CredentialsLocation=[LOCATION];
```

Refer to CData's ADO.NET Provider for Snowflake documentation for a complete list of connection string options and further details.

Caveats and Known Limitations of CData Data Sources

- **Elasticsearch**

- Only in-memory joins are supported
- **Google BigQuery**
  - Does not support primary or foreign key constraints. Therefore, the Automatic Database Discovery feature will be unable to add primary keys or find joins.
  - Calculating sums or averages of a floating point column can provide non-deterministic results. Refer to BigQuery's documentation for more information.
- **MongoDB**
  - Utilize the FlattenArrays and FlattenObjects connection string parameters for best results.

Data Objects

This article applies to the **Admin Console** > **Data** > **Objects** settings.

Data Objects are the tables, views, methods, stored procedures, functions and custom SQL made accessible from the Data Sources for reports.

Create, Edit, Delete Data Objects

- To **add** a new data object click **Objects** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
  - or to quickly add data objects, use the **Automatic Database Discovery** tools
- To **clone** an object, right-click and select **Clone** from the context menu
- To **edit** an object either:
  - double click it
  - select it and click the **Edit** icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** an object either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new objects click the **Apply** or **Okay** buttons.

Data Object Properties

A **data object** has the following properties:

Name

Select the data source from the first dropdown. In the second drop-down select a data object. This field is required.

> **NOTE**
>
> This will display all the of the sourceâ€™s tables, views, methods, stored procedures, and functions.

To add custom SQL click the **Add Custom SQL**

icon next to the **Data Sources** dropdown. For more details see **Custom SQL Objects**.

> **IMPORTANT**
>
> The name of data objects may not contain the following characters:
>
> ```
> {}[],.%
> ```
>
> To create a data object for a database table (or other backing model) with one of these characters in its name, create a Custom SQL Object with a simple SELECT clause. For example for a table named sales.figures.2020:
>
> ```
> SELECT * FROM [sales.figures.2020]
> ```
>
> Note that the identifier delimiters (the [ ]) around the table name will vary with the database type.

Alias

The user friendly name for the data object. The alias will be displayed to end-users. This field is required.

> **IMPORTANT**
>
> An alias may not contain the following characters:
>
> ```
> @{}[],.%
> ```

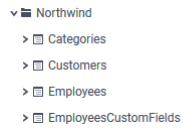
Unique Key Fields

The columns which uniquely identify a record in this data object. This field is required.

Folder v2020.1+

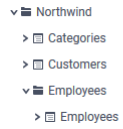
The folder used to group related data objects. A hierarchy can be created by using backslashes between names.

```
Northwind
```



Data objects in a single folder

```
Northwind\Employees
```



A hierarchy of data objects

When configuring Multiple Data Models, separate the different folder names with a comma.

ID

An application-wide unique identifier for the data object. IDs are required when creating multiple data objects with that have the same name but come from distinct data sources. IDs can also be used to optimize Web Service and .NET Assembly calls. For more information see Data Object IDs at the end of this article.

> **NOTE**
>
> By default all objects have an implicit object ID that is set behind-the-scenes. This field allows you to set the IDs explicitly. It is generally recommended to set explicit IDs for all data objects. As of *v2019.1.12*, this field will display the implicit ID by default.

Parameters

Parameters that are passed to stored procedures, table functions, Web Services or .NET Assembly methods. Clicking in the drop-down will bring up a menu. Click the ➕

Add button and select the parameter from the drop-down list. For more information see **Stored Procedures** and **Web Services & .NET Assemblies.**

Parameter values are passed in the order in which they are listed in the data object. It is critical to ensure that the order is correct.

Tenant Columns

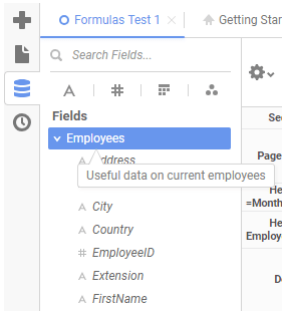
Specify which columns contain tenant information and link the parameters accordingly.

This setting is used to filter data when multiple usersâ€™ information is held within the same table or view, and a column holds information identifying each user. Exago will only retrieve the rows where the column value matches the corresponding parameter.

Description

Provide an optional user-friendly description of this data object. When this object is hovered over in the

Data Fields Pane, the description text will pop up in a tooltip.



Both plain-text can be added verbatim or a language file ID can be supplied. See Column Description v2016.3+ below for more information.

Column Metadata

Specify any columns that should not be filterable, visible, or that should be read as a specific data type. See **Column Metadata** for more information.

Schema Access Type

Specify how Exago should retrieve the schema for the data object. There are three possibilities:

- *Default* â€" Follow the global **Schema Access Type** setting in **Other Settings**.
- *Datasource* â€" Queries the data source for the schema.
- *Metadata* â€" Reads the schema from the stored metadata.

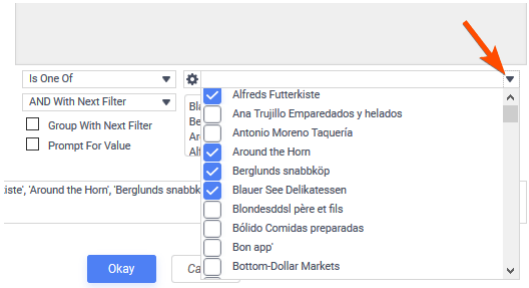
> **NOTE**
>
> For more information see **Retrieving Data Object Schemas**.

As of *v2019.1+*, the **Schema Access Type** defaults to *Metadata*. The optimizations made to the configuration architecture in this version alleviate the performance issues previously associated with loading large configuration filesâ€"metadata playing a primary role in increasing configuration file size. Exago now strongly encourages the full use of metadata in combination with the new configuration optimizations as this will reduce the frequency in which the application queries databases.

> **NOTE**
>
> For more information regarding the performance enhancements of *v2019.1+*, please see this article.

Filter Dropdown Object

Specify an alternative data object or custom SQL to be queried when a user clicks a filter value dropdown (for example in the Report Designer's Filters dialog, the Filters pane of the Dashboard or ExpressView Designer)



Filter value dropdown in the Report Designer's Filters dialog

> **IMPORTANT**
>
> The **Filter Dropdown Object** must have a column with the same name as each column in the main data objects or an error message will be displayed to the user when clicking the dropdown.

To set a Filter Dropdown Object, choose a Data Source and then:

- select a table or view from the dropdown on the right, or
- click the **Custom SQL** icon to write custom SQL statements to retrieve filter values. Writing custom SQL for Filter Dropdown Objects is the same as writing Custom SQL Data Objects.

> **NOTE**
>
> The filter dropdown object data source should return values faster than it's corresponding "parent" data object. Therefore, .NET Assembly data sources are not recommended as Filter Dropdown Object data sources.

To remove a Filter Dropdown object, click the **Delete** ×

icon.

For more information, review the Filter Dropdown Objects section at the bottom of this article.

Interactive Filtering in Database v2021.1+

When *True*, Dashboards incorporating interactive filters will filter in the Data Source instead of in-memory, which can reduce execution time and can reduce the number of calls to the Data Source.

When *False*, the feature is disabled. Dashboards will do all filtering in-memory. Exago will first need to read the filter values by executing the tile, then re-execute the tile with the filter applied. This is the *pre-v2021.1* behavior.

The default value is *True*.

In certain circumstances such as for Stored Procedures, Functions, Custom SQL Objects or Views that take a long time to execute, it may be desirable to disable filtering in the Data Source for an individual data object. This feature should not be disabled in general for these types of objects as they may perform well when collecting filter values or other limited sets of data. Use cases will need to be evaluated individually.

Example Scenarios

Exago suggests setting **Interactive Filtering in Database** to *False* on the Data Object level for:

- A Custom SQL Object that creates a lot of internal joins in the database and requires substantial amounts of memory and/or time to execute.
- A Custom SQL Object whose performance is not affected in any significant way by the presence of hard filters.

Limitations

Even when **Interactive Filtering in Database** is *True*, Exago will bypass this function when one or more of the following conditions are met:

- the filter affects a report with a visible Detail section. In this case, the values in the Detail section will be utilized for the filter values
- the Dashboard contains interactive filers with formulas that cannot be converted to SQL
- **Admin Console** > **General** > **Database Settings** > **Convert Formulas to SQL** is set to *False*
- the filter affects a report that is disqualified from Database Aggregation
- the filter affects a report that contains a Top/Bottom filter
- the filter affects a report with a report-level custom SQL object
- the filter affects a report that references a Data Source that Exago does not generate SQL to query

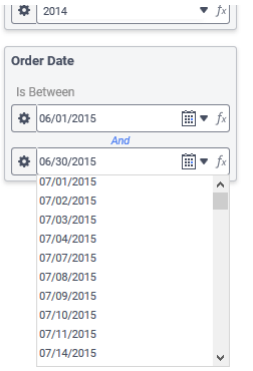
Suppress Sort and Filter v2018.1+

If this object is a programmable object (Stored Procedure or .NET Assembly Method), select whether to suppress the application sorting and filtering for report execution queries. Enable this if the programmable object uses **Programmable Object** parameters to do sorting and filtering in code, as application processing would be redundant and unnecessary. This can allow for better performance for programmable objects.

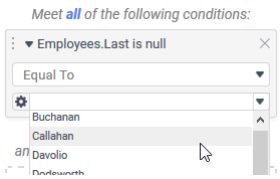
> **NOTE**
>
> If the application requires sorting/filtering in memory, it will not be suppressed. For instance, a report with this object has a cross-source join, an advanced join, a Cartesian join, or a formula sort or filter. Multiple tables from the same programmable data object can be joined with suppressed filtering & sorting if this setting is enabled for all of them.

Filter Dropdown Objects

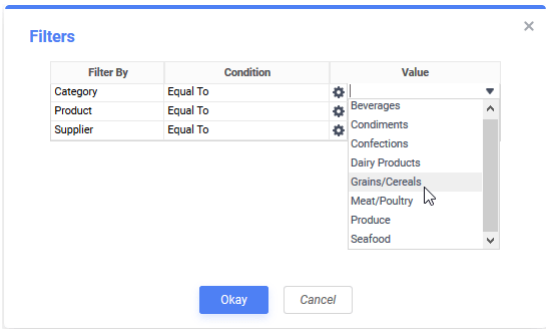
The **Filter Dropdown Object** overrides the source of values that populate a **filter value dropdown** in the user interface (e.g. in a prompting filter dialog; in the Filters dialog in the Report, Dashboard or ExpressView Designer; in the Dashboard or Report Viewer).



Filter Dropdown in the Dashboard Viewer



Filter Dropdown in the ExpressView Designer



Filter Dropdown in the Simple Filter Window of the Report Viewer

By default, filter value dropdowns populate with data from their corresponding data object. Explicitly defining a Filter Dropdown Object in the configuration allows those default values to be replaced by another table, view or by custom SQL statements. This substitution happens

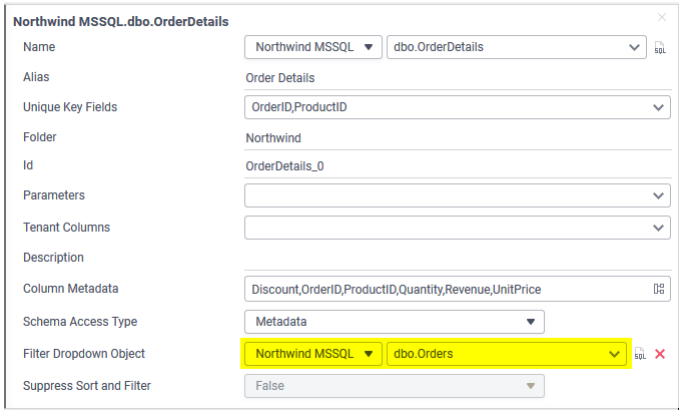
transparently to the end user.

Specifying a **Filter Dropdown Object** may be most useful when the corresponding data object is a Stored Procedure, Web Service, or .NET Assembly that takes more than a few seconds to return data, as this can adversely affect user experience. Filter Dropdown Objects may also be useful when explicitly controlling filter values available to end users is desired (see Caution below).

> **NOTE**
>
> Using a Filter Dropdown Object requires **General** > **Filter Settings** > **Read Database for Filter Values** to be *True*.

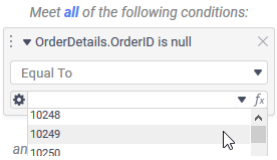
Filter Dropdown Objects may reference Data Sources other than their corresponding data object but there must be an identical column in the source data object and the Filter Dropdown Object.

For example, consider the Orders and OrderDetails tables of the Northwind sample data set. If the Orders table is used as the Filter Dropdown Object for the OrderDetails table, and a filter is created on the OrderId field, the Filter Dropdown Object will work since both tables contain an integer typeOrderId field.

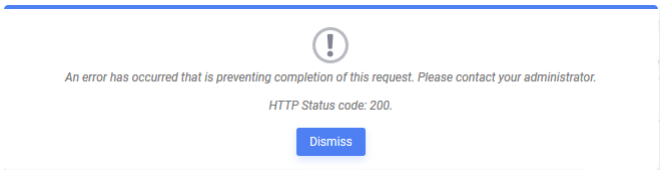


Setting Filter Dropdown Object for Northwind.OrderDetails table

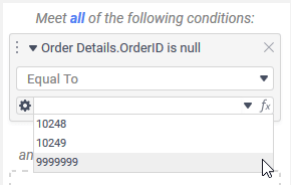
Although not obvious in the figure, the order ID numbers (e.g. *10248*, *10249*, *10250*) in the ExpressView filter dropdown below are read from the Orders table since it is the Filter Dropdown Object for OrderDetails.



On the contrary, there is no Quantity column in the Orders table. When attempting to filter on OrderDetails.Quantity, an error message will be displayed to the end user.



To prevent users from filtering on fields that are not available in the Filter Dropdown Object, set the **Filterable** property in the Filter Dropdown Object's **Column Metadata** to *False*. Note that this will prevent all filtering on that field through the user interface. Filters on the column may still be added via one of the APIs.

> **CAUTION**
>
> Filter Dropdown Objects will completely replace the available filter dropdown values for their corresponding data object. If the Filter Dropdown Object has less or more values than its corresponding data object, users may not be able to filter on all possible values or be able to select values that do not correlate with records in the corresponding object.
>
> The figure below demonstrates a less than ideal Filter Dropdown Objectâ€"the object only contains three Order IDs despite there being many more in the OrderDetails table, as well as an Order ID that does not exist in the OrderDetails table (i.e. *9999999*). A user would only be able to filter by order ID *10248* or *10249*.
>
> 
>

Additional Resources

- Admin Support Lab â€" Filters, segment on Filter Dropdown objects runs from 14:05â€"20:15.

Stored Procedures

Stored Procedures offer the ability to use high level code to modify the data set before it is sent to Exago.

For best performance, stored procedures should know what sorts and filters the user has set and whether to return the schema, a single column, or the entire data set. To accomplish this:

- Use the *Call Type*, *Filter*, *Column* and *Sort Parameters* in the **Programmable Object Settings**. These parameters will be passed from Exago to identically named parameters in the Stored Procedure. It is highly recommended to use these parameters as they can improve performance.
- Additional parameters may be passed by setting them in the **Data** > **Object** > **Parameters** tab for the Data Object.

> **IMPORTANT**
>
> As noted above in the **Parameters** section, parameter values are passed in the order in which they are listed. It is critical to ensure that the order listed is correct.

Important Note for Microsoft SQL Server

SQL Server has an attribute called **FMTONLY** that must be handled by all stored procedures.

FMTONLY has two possible values:

- *ON*: The stored procedure will only return the column schema. However all IF conditional statements are ignored and all of the code will be executed. This setting will fail if the stored procedure contains any temp tables.
- *OFF*: The stored procedure returns all of the data and the column schema. The stored procedure will correctly execute IF conditions.

The *ON* setting will cause problems if there are IF conditions in the procedure; however, only using the *OFF* setting will hurt performance if the *Call Type Parameter* in the **Programmable Object Settings** is not used.

The following example demonstrates how to use the *Call Type*, *Column, Filter and SortStr Parameters* to improve efficiency.

As of *v2019.1.11+* the **Sort Parameter** can be used to indicate in which direction the user is scrolling a filter dropdown. ASC indicates scrolling down or DESC indicates scrolling up.

> **NOTE**
>
> For SQL Servers, FMTONLY is set to OFF.

```
CREATE PROCEDURE [dbo].[sp_webrpt_person]
@callType INT,
@columnStr varchar(1000),
@filterStr varchar(1000),
@fullFilterStr varchar(1000),
@sortStr varchar(1000)
AS
SET NOCOUNT ON /*for performance reasons */
SET FMTONLY OFF /*force procedure to return data and process IF conditions */
declare @sql varchar(2000)
declare @columnInfo varchar(1000)
declare @orderbyClause varchar(1000)
declare @operator varchar(max)
if @callType = 0 /*return schema; no need to return any rows */
begin
 set @sql = 'select * from ww_webrpt_person where 0 = 1'
end
else
if @callType = 1 /*return all data for execution */
begin
 set @orderbyClause = ''
 if @sortStr is not NULL AND @sortStr <> 'null' set @orderbyClause = ' ORDER BY' + @sortStr
 set @sql = 'select ' + @columnStr + ' from ww_webrpt_person where ' + @filterStr + @orderbyClause
end
else
if @callType = 2 /*return filter dropdown values; limit # rows to some value; sortStr will indicate which direction user is scrolling the dropdown*/
begin
set @orderByClause = ''
 set @operator = ' >= '
 if @sortStr is not NULL AND @sortStr <> 'null' AND @sortStr <> '' Begin SET @orderByClause = ' ORDER BY ' + @sortStr END
 if (RIGHT(@sortStr, 4) = 'DESC')Begin SET @operator = ' <= 'END
 set @columnInfo = '[' + @columnStr + ']'
 SET @sql = 'select top 100 ' + @columnInfo + ' from Northwind.dbo.OrderDetails where ' + @columnInfo + @operator + @filterStr + ' AND ' + @fullFilterStr + @orderByClause
end
exec(@sql)
```

Table Value Functions

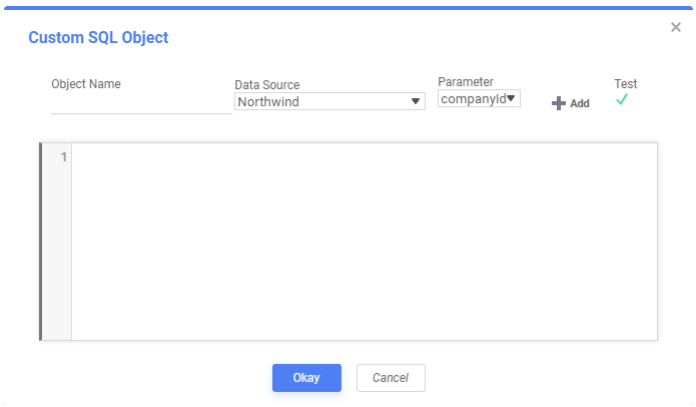
Table Value Functions can be used as data objects. Any available table value functions of a data source will be displayed in the

Extensions tab under **Functions**. Exago handles table value functions similar to views and tables except it will pass any parameters set in the

Data > **Object** tab or in the **Programmable Object Settings**.

For more information, see **Table-Valued Functions**.

Custom SQL Objects



Exago can use custom SQL as a data object.

1. Provide a unique **Object Name** for the new data object. It cannot match any existing data object in the system configuration nor contain any of the following characters, including spaces: [ ] { } . , @
2. Choose the **Data Source** that this Custom SQL Object will retrieve data from. Not all data sources in the system may support Custom SQL objects.
3. In the large **code window** enter the full SQL statement. This statement will be inserted as a subquery when it is sent to the Data Source for processing.

> **CAUTION**
>
> ORDER BY clauses and **Common Table Expressions** are not supported in subqueries.

4. System variables called **parameters**, that change based on certain environmental situations (e.g. who is running the report) may optionally be added to the SQL statement by selecting the name from the dropdown and clicking the **Add** button.
5. Click the **Test** icon to check the SQL statement for validity.
6. Click **Okay** to close the Custom SQL Object dialog and continue editing the data object.

Data Object Macros

Macros can be embedded in **Custom SQL Objects** to make them even more dynamic. Each macro allows for different SQL to be used according to the circumstances in which the data object is being called. Below are the details and examples of available macros.

IfExecuteMode

(string trueCondition, string falseCondition)

Description	Includes the *trueCondition* if a user is executing a report. Includes the *falseCondition* if otherwise.
Example	SELECT * FROM vw_webrpt_optionee IfExecuteMode("WHERE [State] = 'CT'","")

IfExistReportDataObject

(string dataObjectName, string trueCondition, string falseCondition)

Description	Includes the *trueCondition* if *dataObjectName* exists inside the full Exago SQL statement to the data source. Includes the *falseCondition* if otherwise.
Example	SELECT * FROM vw_webrpt_optionee IfExistReportDataObject("fn_webrpt_grant", "JOIN ON fn_webrpt_grant…", "")

Data Object IDs

There are three ways in which you can utilize data object IDs.

Adding Multiple Data Objects with the Same Name

IDs distinguish Data Objects that have the same name but come from different Data Sources. When adding multiple Data Objects with the same name, make sure each data object has a **unique ID**.

Avoiding Issues from Changes to Object Names

Providing IDs for all the Data Objects prevents issues if the name of the underlying table, view or stored procedure is changed.

Calling a Single Web Service/.NET Assembly/Stored Procedure

Web Services, .NET Assemblies and Stored Procedures comprise a group called **Programmable Objects**. These objects can retrieve parameters from Exago and the host application in order to control what data is exposed to the user.

Generally for Web Services and .NET Assemblies each data object calls a distinct method. Similarly each Stored Procedure is its own data object. By using data object IDs a single method/stored procedure can be called. This method can then return data or schema based on the data object ID.

To call a single Web Service/.NET Assembly/Stored Procedure:

1. Provide a name for **Data Object ID Parameter Name** in **Programmable Object Settings**
2. Create a method/procedure in your Service/Assembly/Procedure that utilizes the object ID parameter to return the appropriate data/schema.
3. For each data object:
    1. Select **Object** in the **Main Menu** and click the **Add** button
    2. Select the single Service/Assembly/Procedure
    3. Provide an **Alias** and an **ID** for the object
    4. Select the key columns
    5. Click **Okay** or **Apply** to save the object.

Example: This stored procedure uses the object ID parameter (@) to return different data/schema information for different object IDs.

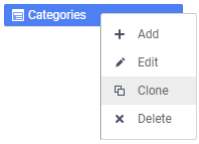
```
ALTER PROCEDURE "dbo"." Exago_Example"  @callType INT, @objectID nvarchar(max) AS
SET
NOCOUNT
  ON
SET
FMTONLY OFF  if @objectID = 'Produce'  begin    if @callType = 0    begin       SELECT
  ProductID,
  ProductName,
  SupplierID,
  UnitPrice,
  UnitsInStock
FROM
  Products
WHERE
  CategoryID = 1001
end
else if @callType = 1    begin       SELECT
  ProductID,
  ProductName,
  SupplierID,
  UnitPrice,
  UnitsInStock
FROM
  Products
ORDER BY
  ProductID
end
else if @callType = 2    begin       SELECT
  ProductID,
  ProductName,
  SupplierID,
  UnitPrice,
  UnitsInStock
FROM
  Products
ORDER BY
  ProductID
end
end  if @objectID = 'Orders0'  begin    if @callType = 0    begin       SELECT
OrderID,
OrderDate,
RequiredDate,
ShippedDate,
CustomerID
FROM
Orders
WHERE
CustomerID = 0
end
else if @callType = 1    begin       SELECT
OrderID,
OrderDate,
RequiredDate,
ShippedDate,
CustomerID
FROM
Orders
ORDER BY
OrderID
end
else if @callType = 2    begin       SELECT
OrderID,
OrderDate,
RequiredDate,
ShippedDate,
CustomerID
FROM
Orders
ORDER BY
OrderID
end
end
```

Cloning Data Objects v2019.1+

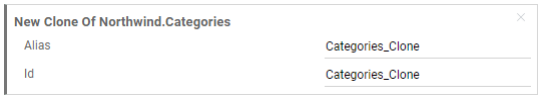
s of *v2019.1+*, data objects may be cloned via the **Admin Console**. A cloned data object inherits all its base attributes from its parent except its Alias and ID, which must be entered manually in order to create the clone.

As of *v2019.2.18+*, a cloned data object may have its own folder and description.

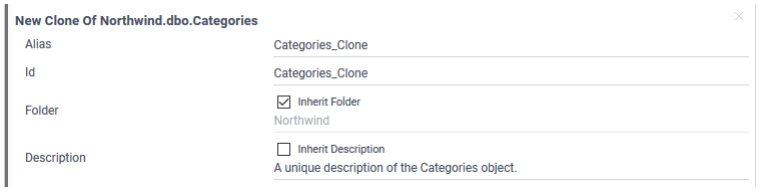
To clone a data object, right-click it and select **Clone**.



Enter a new **Alias** and **ID** for the cloned object.

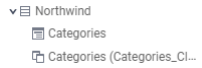


In *v2019.2.18+*, select how to handle the folder and description that will be associated with this cloned object:



- **Inherit Folder** â€" check this checkbox to copy the folder location of the object that this clone was created from. Uncheck to provide a different folder location in the text field below. When checked, the value copied is shown in the field below. The default value is *checked*.
- **Inherit Description** â€" check this checkbox to copy the description text of the object that this clone was created from. Uncheck to provide a unique description in the text field below. When checked, the value copied is shown in the field below. The default value is *checked*.

The cloned object will then appear in the **Objects** menu:



Functionality of Cloned Data Objects

- A cloned data object is linked to its parent object. Any changes made to a parent data object will be reflected in its clone.
- As of *v2020.1.17+* and *v2021.1.5+*, cloned data objects may point to a different (but identical) object in the corresponding data source (e.g. a view, a table). This requires editing the XML configuration file. See the Data Objects section of the Config File XML Reference (All Nodes but General) article for more information.
- Deleting the parent data object will also delete its linked clone.
- Clones do not inherit joins from their parent objects. Joins need to be manually created between clones and other data objects.
- Clones cannot be distinguished from other data objects within the report designers. There is no visual or functional difference between a cloned data object and a normal object outside of the **Admin Console**.
- A cloned data object cannot itself be cloned or duplicated.

Column Metadata

Column metadata refers to the properties of each column in the data object.

> **NOTE**
>
> Column metadata overrides culture settings.

There are several ways to add column metadata to a data object:

- for all data objects in a data source in bulk utilizing the Bulk Metadata feature
- with the **Add All Complete Objects, Joins and Metadata** feature on the corresponding Data Source (*v2021.2+*)
- manually for each data object, as described below

To modify the metadata of a column, either:

- double-click it
- select it then click the ➕ **Add** button.

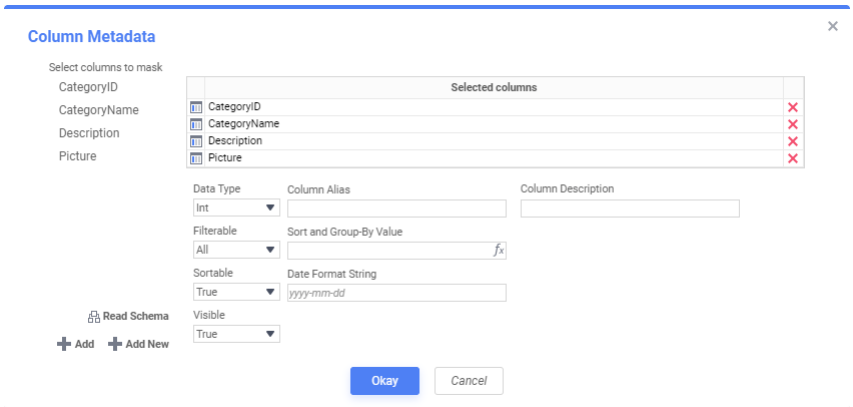
Click the

Read Schema button to quickly create column metadata for each column in the data object.

To remove metadata for a column, select it in the right panel and click the **Delete** ✕

icon.

To save changes to **Column Metadata**, click the **Okay** button. To discard the changes, click the **Cancel** button.



The following properties of each column can be modified:

Column Alias

The name of the data field that the end-users see.

> **IMPORTANT**
>
> Column aliases may not contain the following characters:
>
> {}"

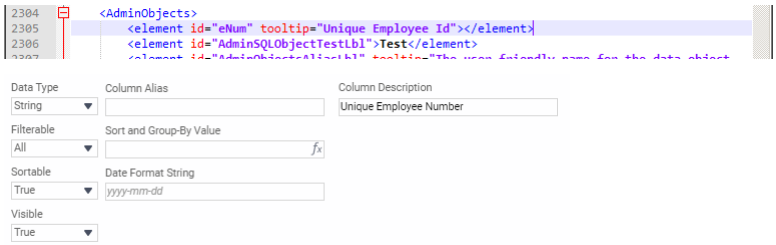
Column Description v2016.3+

Data fields can have description text added. If the data field is hovered over in a selection screen in the **Report Designer**, the description text will pop up in a tooltip.

Plain Text

Tooltip text can be added verbatim in the **Column Description** field. In-line HTML tags like can also be used if desired. Press **Okay** when done, then **Apply** the change.

Language File



You can also add description elements to the language file, and reference them in the **Column Description** field.

In the language file, add new elements to the <AdminObjects> section using the following format:

```
<element id="uniqueIdentifier" tooltip="Description Text"></element>
```

HTML tags must be encoded like so:

- Encode < as <
- Encode > as >
- Encode " as "

For example, the following tooltip string encodes "Description Text", which displays as "**Description** Text."

```
<element id="uniqueIdentifier" tooltip="&lt;b&gt;Description&lt;/b&gt; Text"></element>
```

After adding the element to the language file, add the ID string to the **Column Description** field. Press **Okay** when done, then **Apply** the change.

Data Type

The type of data Exago should treat the data field as any of the following valid types:

- String
- Date
- Datetime
- Time
- Int
- Decimal
- Image
- Float

- Boolean
- Guid
- Currency

Filterable

Whether this field can be used to filter reports. The available options are:

> **IMPORTANT**
>
> To prevent users from attempting to build a filter in the **Report Designer** on an unsupported data type, **Filterable** should be set to *None* if the column's data type is not one of the supported types in the Data Type section above.

Field	Used as a Report Filter	Used as an Interactive Filter
All (*True*)	Yes, for the: • **Advanced Report** Designer • **Express Report** Designer • **ExpressView** Designer • **Dashboard** Designer • **Scheduler**	Yes, for the: • **Report Viewer**
Dynamic (*False*)	No	Yes, for the: • **Report Viewer**
Static (*v2017.1.2+*)	Yes, for the: • **Advanced Report** Designer • **Express Report** Designer • **ExpressView** Designer • **Dashboard** Designer • **Scheduler**	No
None (*v2017.1.2+*)	No	No

> **NOTE**
>
> This setting affects the availability of the column in the user interface. Filters may still be programmatically added to reports with the API regardless of the setting here.

Sortable v2016.3+

If set to *False*, the data field will not be listed in the **Sorts** menu.

Admins can now indicate whether data fields should appear in the **Sorts** menu using the **Sortable** dropdown.

> **NOTE**
>
> This toggle does not prevent data fields from being sorted by. Users can still enter the data fields manually as a formula, or use the data fields within a sort formula.

Date Format String v2019.1+

The format string for datetime data fields. Allows datetime data fields to be accurately translated from generic string columns, in order to support datetime metadata for vertical tables.

The exact form of this string is dependent on the underlying data source. Refer to the data source provider's official documentation for more information. This transformation is supported for any data source that supports converting formulas to SQL, the most popular sources are listed here. For example:

- Microsoft SQL Server CAST and CONVERT Date and Time styles
- MySQL STR_TO_DATE() Format Specifiers
- PostgreSQL Template Patterns for Date/Time Formatting
- ORACLE Database Datetime Format Models

Consider this example vertical table:

Key Value	Field Name	Value	Data Type
1	LastName	Davolio	0
1	HireDate	1/31/1975	8
2	LastName	Callahan	0
2	HireDate	12/7/2000	8

To properly translate the HireDate column, the **Date Format String** would look like the following:

Data Source Type	Date Format String
Microsoft SQL Server	101
MySQL	%c/%e/%Y
PostgreSQL	FMMM/FMDD/YYYY
ORACLE	FMMM/DD/YYYY

Visible

If set to *False*, the data field will not be listed for users.

Sort and Group-By Value v2016.3+

Specify a custom formula by which columns should be sorted and grouped by the application.
This field allows admins to specify how columns should be sorted and grouped by the application. By default, Exago will sort (and group) columns based on the data in the column. You can use this metadata field to specify different data by which the column should be sorted.

> **NOTE**
>
> As of *v2019.1.1+*, **Sort and Group-By Values** are not honored when the data field is being used within a formula sort. This change was made to prevent unexpected behavior from occurring in these instances.

For example, you may have a custom column *Employees.FullName* like the following:

{Employees.FirstName} & ' ' & {Employees.LastName}

By default, Exago would sort this field on the full string. You may want to sort on just the *LastName*, instead. In **Sort and Group-By Value**, enter *{Employees.LastName}*, and the column will sort on *LastName*.

Another common example is sorting a *Month* field by the numeric representation of the month instead of the name. Since this value accepts any valid Exago formula (except aggregates), custom functions can also be used.

> **NOTE**
>
> The **Sort and Group-By Value** field must have a one-to-one relationship with the data field. Otherwise, unexpected behavior could occur.

Custom Columns v2016.3+

Custom Columns are a way to add columns to Exago that don't exist in the Data Source. This is completely transparent for the users; they can then use them like any other column except as noted below. New data columns can be created from composite or interpreted data fields. A formula can be used to create data from scratch. Admins often use custom columns to make popular formula sorts available on an application-wide level.

> **CAUTION**
>
> Custom columns cannot be used as filters, within aggregate formulas or as the collating field on a Chained Report.

Admin Console

To add a custom column using the **Admin Console**:

1. Expand the **Data** > **Objects** group and double-click on the desired data object, or select it and click the **Edit** icon.
2. In the **Object** panel that opens, in the **Column Metadata** field, click on the **Edit Metadata** icon at the end of the line to open the Column Metadata dialog box.
3. Click the **Add New** button at the bottom left of the Column Metadata dialog box. Enter a name for the new column in the dialog box.
4. **Data Type**, **Column Alias**, and **Column Value** are required fields, the others are optional.
5. In the **Column Value** field, click the **Formula Editor** icon to open the Formula Editor and define the data for the new column. For example, to create a new *Revenue* column, you might enter the formula {OrderDetails.Quantity} * {OrderDetails.UnitPrice}.

   > **NOTE**
   >
   > The **Custom Value** field may only reference other fields that exist in the Data Source, and not other Custom Columns including references to itself.

6. Click **Okay** when done, then click **Apply** to save the new column to the configuration.

Config File

To add a **Custom Column** by editing the config file, open the config file in a text or xml editor. Data Objects are <entity> elements. Locate the desired entity and add a new <column_metadata> element:

```
<entity>
 ...
  <column_metadata>
  <col_source>ExagoFormula</col_source>
  <col_name>FullName</col_name>
  <col_type>string</col_type>
  <col_alias>FullName</col_alias>
  <col_description>First and Last Name</col_description>
  <filterable>false</filterable>
  <col_value>{Employees_0.FirstName} &amp;amp; ' ' &amp;amp; {Employees_0.LastName}</col_value>
  <col_sortandgroupbyvalue>{Employees_0.LastName}</col_sortandgroupbyvalue>
 </column_metadata>
</entity>
```

> **NOTE**
>
> Fields in **bold** are required.

<col_source>ExagoFormula</col_source> is static. This is the same for every <column_metadata>.

In <col_value> and <col_sortandgroupbyvalue>, data fields are identified by their ID, not their alias.

> **NOTE**
>
> The **<col_value>** field may only reference other fields that exist in the Data Source, and not other Custom Columns including references to itself.

Acceptable values for <col_type>: string, date, datetime, time, int, decimal, image, float, boolean, guid, currency.
Save the config file when done, and restart the web server.

Examples:

There are a lot of options for what kinds of data fields you can create:
Transform or interpret an existing data field:

```
Right({Employees.SocialSecurityNumber},4)
```

```
Month({Orders.OrderDate})
```

Combine multiple data fields together:

```
{Employees.FirstName} & ' ' & {Employees.LastName}
```

Create new data from scratch:

> **NOTE**
>
> The following example uses a custom function.

```
Random(0,65536)
```

Retrieving Data Object Schema

Many of the dialogs throughout Exago require schema information such as column name, data type, filterable, sortable and visible status, sort-and-group-by value etc…. To enhance performance, schema information can be stored in the application as **column metadata**. Exago can then read this metadata instead of querying the data source.

> **NOTE**
>
> While storing the schema as column metadata improves performance, updates to the column metadata will be required whenever columns are added, removed, or re-titled.

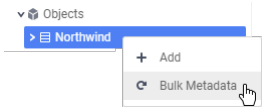
Building Metadata in Bulk

The use of column metadata reduces the frequency in which the databases are queried for schema information. To make the process of adding metadata simple, this information can now be built in bulk through the **Admin Console**.

Metadata can be built in bulk:

- with the **Add All Complete Objects, Joins and Metadata** feature on the corresponding Data Source (*v2021.2+*)
- with the Bulk Metadata feature described below

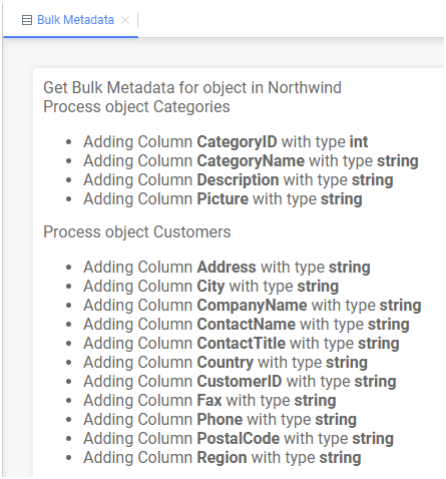
Right-click a data object, then click **Bulk Metadata**.



Building metadata in bulk on the Northwind data source

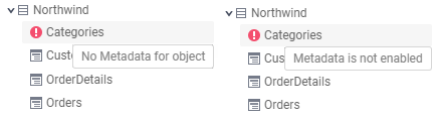
Exago will then process the information for each data object under that data source and add metadata information for each field within these objects. The **Schema Access Type** of each object will automatically be set to *Metadata* for each object.

A new **Bulk Metadata** tab will open in the **Admin Console** displaying the processing information.



The metadata information that has been processed and added to each data object may then be edited or removed in the **Column Metadata** menu.

Please note, however, that if metadata has not been added to or is not enabled for a data object, the following warning will appear next to the data object in the **Admin Console**. This warning serves to further encourage the use of metadata and notify the system administrator that this data object will query the database each time it requires schema information.



Building Metadata Manually

In versions prior to *v2019.1*, data sources are queried for schema information by default. This process, however, may cause performance issues if the data sources take a considerable amount of time to return the schema. To avoid these performance issues, metadata may be added manually via the **Column Metadata** menu.

For Exago to retrieve schema information from the metadata:

1. In **Other Settings**, set **Schema Access Type** to *Metadata*. This will force Exago to get all schema information from the metadata for all data objects.

   > **NOTE**
   >
   > Alternatively this setting can be overwritten for individual data objects by setting the **Schema Access Type** property.

2. For each data object open the **Column Metadata** menu.
   1. Click the **Read Schema** button. A message will appear asking you to confirm you want to continue. Click **Okay**.
   2. Click **Okay** to close the **Column Metadata** menu.
   3. Press **Okay** or **Apply** to save the data objects.

   > **NOTE**
   >
   > Other metadata options such as aliasing can still be utilized.

Reading Images from a Database

Exago can read images from a database and load them directly into a cell of a report. When images are stored in a database as a binary string there are two ways that Exago can load them into a report.

1. Change Column Metadata
   1. Open the **Column Metadata menu** and for the image column set **Data Type** to *Image*.
   2. Place the data field containing the images into the desired cell of a report. Upon execution the images will be loaded into the cell.
2. Without changing Column Metadata
   1. Place the data field that contains the images into the *LoadImage* function. Upon execution Exago will interpret the binary and load the

images into the cell.

Multiple Data Models

In some cases the same Data Objects may need to be joined together differently. To accomplish this, Data Objects and Joins can be placed into Folders to create **multiple data models**. When an end user selects a Data Object from a Folder it indicates which joins to use.

The following steps detail how to create multiple data models:

> **NOTE**
>
> This article references <WebApp>/, as a placeholder for the Web Application's install location. The default install location is C:Program Files\Exago\ExagoWeb on Windows or /opt/Exago/ on Linux, however, these directories can be changed during installation.

1. *Optional:* To limit a report to using fields from a single Data Object, set **Admin Console** > **General** > **Other Settings** > **Limit Reports and Visualizations to One Object** to *True*.
2. Open the XML configuration file (by default this file is located at <WebApp>\Config\WebReports.xml) in a text editor.
3. In the <webreports> section, begin by creating a <category> for each unique data model.

   > **CAUTION**
   >
   > Each XML tag must be closed (e.g. <category> must be closed with </category>).

4. Inside the <category> specify an ID with a <category_id> node. The ID should be a unique identifier for the data model and will be utilized by the Data Objects and Joins.
5. Give the model a name that will be displayed to the end user using the <category_name> node.

   > **TIP**
   >
   > The <category_name> node creates a folder structure to group Data Objects. Child folders can be created by entering the parent name, a backslash, then the then the child folder name name. (e.g. <category_name>Sales\Clients</category_name>)

   Example:

```
<category>
 <category_name>Exago University\Advisors</category_name>
 <category_id>advisorModel</category_id>
</category>
```

6. For each Data Object (<entity> node):
   1. Within the <category> node, create a comma separated list of IDs for each data model in which you want the Object to be available. In the example below two data models are specified by their IDs (advisorModel & classesModel).
      Example:

```
<entity>
  <entity_name>Professors</entity_name>
  <db_name>Professor</db_name>
  <category>advisorModel,classesMode</category>
  <datasource_id>7</datasource_id>
  <object_type>xmltable</object_type>
  <key>
    <col_name>ID</col_name>
  </key>
</entity>
```

7. For each Join (<join> node):
   1. Within the <category> node, create a comma separated list of IDs for each data model in which you want the Join to be utilized. In the example below a Join between two Data Objects is being set to one data model (advisorModel).
      Example:

```
<join>
Â<entity_from_name>Professor</entity_from_name>
Â<entity_to_name>Student</entity_to_name>
Â<join_type>rightouter</join_type>
Â<relation_type>1M</relation_type>
Â<weight>0</weight>
Â<category>advisorModel</category>
Â<joincol>
ÂÂ <col_from_name>ID</col_from_name>
ÂÂ <col_to_name>Advisor</col_to_name>
Â</joincol>
</join>
```

Example

The following configuration example demonstrates how three Data Objects are made available in two different relational models. In the advisor

Model model Students are joined directly to Professors, while in the classes Model model Students are joined to Professors indirectly through Classes.

Models

```
<category>
  <category_name>Exago University\Advisors</category_name>
  <category_id>advisorModel</category_id>
</category>
<category>
  <category_name>Exago University\Classes</category_name>
  <category_id>classesModel</category_id>
</category>
```

Data Objects

```
<entity>
  <entity_name>Classes</entity_name>
  <db_name>Class</db_name>
  <category>advisorModel,classesModel</category>
  <datasource_id>7</datasource_id>
  <object_type>xmltable</object_type>
  <key>
    <col_name>ID</col_name>
  </key>
</entity>
<entity>
  <entity_name>Students</entity_name>
  <db_name>Student</db_name>
  <category> advisorModel,classesModel </category>
  <datasource_id>7</datasource_id>
  <object_type>xmltable</object_type>
  <key>
    <col_name>ID</col_name>
  </key>
</entity>
```

Joins

> **TIP**
>
> The Professors => Classes join is utilized by both Data Models because no <category> is set.

```
<join>
  <entity_from_name>Professor</entity_from_name>
  <entity_to_name>Student</entity_to_name>
  <join_type>rightouter</join_type>
  <relation_type>1M</relation_type>
  <weight>0</weight>
  <category>advisorModel</category>
  <joincol>
    <col_from_name>ID</col_from_name>
    <col_to_name>Advisor</col_to_name>
  </joincol>
</join>
<join>
  <entity_from_name>Professor</entity_from_name>
  <entity_to_name>Class</entity_to_name>
  <join_type>inner</join_type>
  <relation_type>1M</relation_type>
  <weight>0</weight>
  <joincol>
    <col_from_name>ID</col_from_name>
    <col_to_name>ID</col_to_name>
  </joincol>
</join>
```

Automatic Database Discovery

This article applies to the **Admin Console** > **Data** >Â ≡ **Sources** settings.

To discover and add data objects, joins and metadata to the configuration automatically use:

- Add All Complete Objects, Joins and Metadata tool, or
- Discover Database Metadata tool

Automatic Database Discovery is available for all Exago supported Data Sources except files (Excel & XML), .NET Assemblies and Microsoft OLAP. Google BigQuery Data Sources do not support primary and foreign key constraints, so the Automatic Database Discovery tools cannot

add primary key columns or suggest joins.

Add All Complete Objects, Joins and Metadata v2021.2+

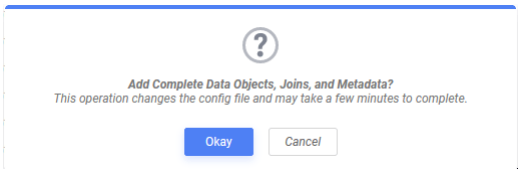
To automatically add all complete data objects, joins and column metadata in a single operation, use the **Add All Complete Objects, Joins and Metadata** feature.

This feature adds new tables, views, stored procedures or database functions that have at least one primary key defined in the data source (called a complete object). Joins are added when there is a foreign key from one complete object to another.

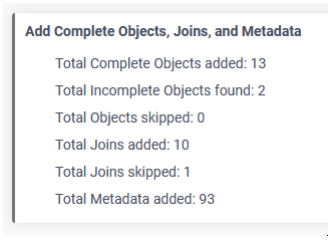
Any data objects or joins that already exist in the configuration will not be updated with information discovered with this method, they are skipped instead. To add missing metadata or join information, use the Discover Database Metadata method described below.

To do so:

1. Right-click on a Data Source.
2. Click + **Add All Complete Objects, Joins and Metadata** in the context menu
3. Click **Okay** on the confirmation dialog, or **Cancel** to abort the process.



4. Wait for the summary tab to open.



The summary tab shows the result of the operation:

- **Total complete objects added** â€" number of complete data objects added to the configuration
- **Total incomplete objects found** â€" number of incomplete objects found. Incomplete objects are not added to the configuration with this feature. Use Discover Database Metadata to add them.
- **Total objects skipped** â€" number of complete data objects found that already exist, or otherwise not added to the configuration
- **Total joins added** â€" number of joins added to the configuration
- **Total joins skipped** â€" number of joins found that already exist, or otherwise not added to the configuration
- **Total metadata added** â€" number of data object columns that had column metadata added

5. If in step 4 there were any incomplete objects found, either use Discover Database Metadata or manually add the metadata to make them complete.

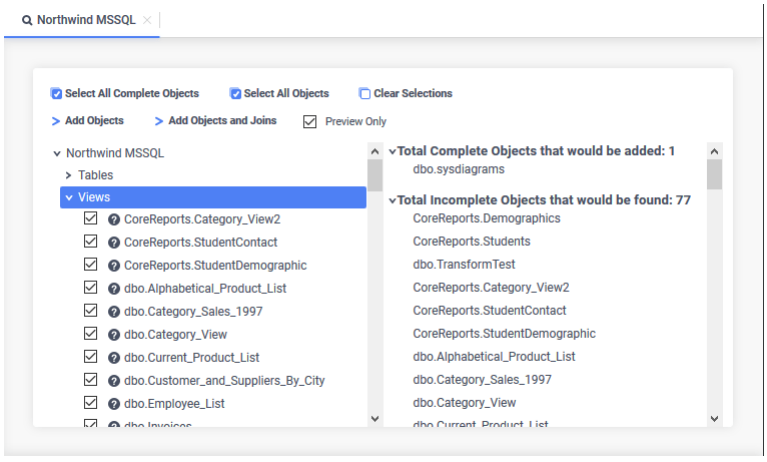
Discover Database Metadata

The **Discover Database Metadata** option is available for granular control over the automatic discovery and import of Data Source schema information.It is also used when incomplete objects exist in a Data Source and are not detected with the Add All Complete Objects, Joins and Metadata v2021.2+ feature.

To start using Automatic Database Discovery, select a Data Source from the list and then either:

- click the **Discover Database Metadata** icon at the top of the main menu
- right-click the Data Source then click **Discover Object/Join Metadata** from the context menu

This will open a discovery tab for the Data Source.



Automatic Database Discovery tab

In the discovery tab you can:

1. Select the Tables, Views, Functions, and Stored Procedures you would like to add by either:
   - checking individual item checkboxes
   - clicking **Select All Complete Objects** to select only those objects with unique key values already defined
   - clicking **Select All Objects** to select all data objects regardless of whether or not unique key values are defined

Table A â€" Icons

Icon	Description
	This object has identified unique key values
	This object has associated joins
	This object is incomplete

2. Add Unique Key fields by right-clicking on an object and selecting **Add/Edit Primary Key Info**.
3. Check **Preview Only** and then **> Add Objects** to preview the selected objects and joins.
4. Uncheck **Preview Only** and then add the selected Data Objects to the system by clicking **> Add Objects** or **> Add Objects and Joins** to add the selected objects and any associated joins.

> **NOTE**
>
> If any selected Objects are missing unique key values they can be completed individually in a new tab which will open entitled **Incomplete Objects**.

Schema Discovery

Exago can retrieve primary and foreign key constraint information from the Data Sources by either:

- Executing a SQL queries to discover the schema
- Using ADO.NET GetSchema() calls (default) v2021.1.8+

SQL Queries

Customizing Data Discovery SQL v2016.3.6+

The SQL statements used for Automatic Database Discovery can be customized if necessary, in order to accommodate non-standard key names.

To customize the discovery SQL:

1. Refer to Managing the dbconfigs.json File for the location of the file to edit. Follow the instructions in the **Overriding dbconfigs Information** section to correctly override the default behavior depending on your version of Exago.
2. Locate the property for your data source type and edit the SQL for either or both primary and foreign keys.
3. Save the file, then run Database Discovery to see your changes.

Excerpt from dbconfigs.json showing the database discovery SQL statements for Primary and Foreign Key discovery on a MySQL Data Source:

```
"mysql": {
    "PrimaryKeySql": "SELECT CONSTRAINT_NAME AS indexname, TABLE_SCHEMA as schemaname, TABLE_NAME as tablename, COLUMN_NAME as columnname FROM informa
    "ForeignKeySql": "SELECT TABLE_SCHEMA As schemaname, TABLE_NAME as tablename, COLUMN_NAME as columnname, REFERENCED_TABLE_SCHEMA as referencedsch
}
```

ADO.NET GetSchema() v2021.1.8+

Exago will use the standard GetSchema() methods to request schema information from a Data Source when:

- The database driver in use to connect to the Data Source is an ADO.NET driver
- The dbconfigs.json (or according dbconfigs.overrides.json) file contains the **PrimaryKeysCollection** and/or the **ForeignKeysCollection** properties.

Exago provides the definition of the **PrimaryKeysCollection** and **ForeignKeysCollection** properties on data source-by-data source basis. The values for each property vary with the Data Source type and the driver in use.

An example of implementing ADO.NET GetSchema() primary/foreign key constraint discovery for a MySQL Data Source follows:

```
{
 "mysql": {
  "PrimaryKeysCollection": {
   "CollectionName": "Indexes",
   "ConstraintNameColumn": "INDEX_NAME",
   "SchemaColumn": "INDEX_SCHEMA",
   "TableColumn": "TABLE_NAME",
   "IsPrimaryColumn": "PRIMARY"
  },
  "PrimaryKeyColumnsCollection": {
   "CollectionName": "IndexColumns",
   "ConstraintNameColumn": "INDEX_NAME",
   "SchemaColumn": "INDEX_SCHEMA",
   "TableColumn": "TABLE_NAME",
   "ColumnColumn": "COLUMN_NAME"
  },
  "ForeignKeysCollection": {
   "CollectionName": "Foreign Key Columns",
   "FromSchemaColumn": "TABLE_SCHEMA",
   "FromTableColumn": "TABLE_NAME",
   "FromColumnColumn": "COLUMN_NAME",
   "ToSchemaColumn": "REFERENCED_TABLE_SCHEMA",
   "ToTableColumn": "REFERENCED_TABLE_NAME",
   "ToColumnColumn": "REFERENCED_COLUMN_NAME"
  }
 }
}
```


Table of Contents

Join Configuration

This article applies to the **Admin Console** > **Data** > **Joins** settings.

Joins specify to Exago the relationship between Data Objects.

All existing Joins are listed in the Main Menu under Data. All the joins you are adding or editing will be displayed in a tab entitled **Joins**.

Create, Edit, Delete Joins

- To **add** a new join click icon at the top of the main menu
- right-click and select **Add** from the context menu
- or to quickly add data joins, use the **Automatic Database Discovery** tool

- To **edit** a join either:
  - double click it
  - select it and click the **Edit** icon at the top of the main menu
  - right-click it and select **Edit** from the context menu

  > **IMPORTANT**
  >
  > Review the Modifying Joins section below when making changes to joins in the Admin Console.

- To **delete** a join either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new joins click the **Apply** or **Okay** buttons.

Join Properties

Every join has the following properties:

From Object

The first Data Object you would like to join.

To Object

The other Data Object you would like to join.

- The order of the Data Objects is important if you have a one-to-many relation type or a Left/Right Outer Join type. See below for details.

Join Column(s)

Specify the field(s) of each Data Object that must match to join an entity in the From Object to an entity(s) in the To Object.

Join Type

Specify whether rows from either Data Object that do not have a match should or should not be included.

- **Inner** â€" include only rows of the *From* Object that have a match in the *To* Object and vice versa.
- **Left Outer** â€" include rows of the *From* Object that do not have a match in the *To* Object but not vice versa.
- **Right Outer** â€" include rows of the *To* Object that do not have a match in the *From* Object but not vice versa.

> **NOTE**
>
> SQLite Data Sources do not support right outer joins.

- **Full Outer** â€" include rows in either Data Object that do not have a match.

> **NOTE**
>
> MySQL Data Sources and SQLite Data Sources do not support full outer joins.

Relationship Type

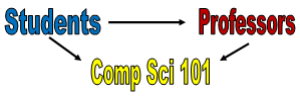
Specify if the join type is one-to-one or one-to-many.

- **One-to-One** â€" each row in the *From* Object can join to at most one row from the *To* Object.
- **One-to-Many** â€" each row in the *From* Object can join to any number of rows from the *To* Object.

Weight

Give a join weight in order to set its precedence when multiple join paths exist between Data Objects. The path with the higher weight will be utilized.

- A report contains three Data Objects â€œStudentsâ€™, â€œProfessorsâ€™ and â€œComp Sci 101.â€™ Students is joined to â€œProfessorsâ€™ and â€œComp Sci 101.â€™ Additionally â€œProfessorsâ€™ is joined to â€œComp Sci 101.â€™ There are two available join paths between â€œStudentsâ€™ and â€œComp Sci 101.â€™ Adding weight to a join will clarify which of the two paths Exago should use.



Modifying Joins

Although joins are created in the Admin Console, they are also saved within each individual report. If changes are made in the Admin Console, the affected reports need to refresh their joins. To do so:

1. Open the report for editing in the Report Designer.
2. From the Advanced Options > Joins menu, click the **Recreate** button.
3. Click **Okay** on the confirmation dialog.

For more information, review Advanced Report Designer: Advanced Options.

Cross Source Joins

Data Objects from different Data Sources can be joined together.

Caveats

- Filter Dependency only works when all data objects originate from the same Data Source.
- The data types and size of the joined columns must match exactly. For example, an Unsigned Int column may only be joined with another Unsigned Int column.

> **IMPORTANT**
>
> Cross Source Joins are done in-memory, so this process may be memory intensive for very large data sets.

Special Cartesian Processing

*Part of the **Performance and Scaling** series.*

Exago BI can detect when a report query will return a **Cartesian product** or **cross join** and process the query in such a way to avoid returning a nonsensical table. This can considerably improve the end-user experience, but it can negatively impact execution performance.

Special Cartesian Processing, enabled by default for all reports, looks at the **join configuration** to see whether a report contains multiple **one-to-many** join relationships.

Assuming that the “many” side of these relationships have no direct join to each other, a Cartesian product will be formed when executing this query. This results in a table where every field in the first object is joined to every field in the second object, causing many duplicate entries.

With Cartesian processing turned on, each joined pair is executed as a separate query, then the resulting tables are joined in the application according to the key fields for the “one” side of the joins.

Example

Take the following object relationships as an example. Let's say we have a report with the following three data objects:

- Cities
- Avenues (by city)
- Boulevards (by city)

We potentially have the following joins:

1. (one) Cities >> Avenues (many)
2. (one) Cities >> Boulevards (many)

There is no given relationship between Avenues and Boulevards. When we execute this query without Special Cartesian Processing, the dataset returned is a Cartesian product:

City	Avenue	Boulevard
Los Angeles	Melrose Ave	Hollywood Blvd
Los Angeles	Melrose Ave	Sunset Blvd
Los Angeles	Melrose Ave	Colorado Blvd
Los Angeles	La Brea Ave	Hollywood Blvd
Los Angeles	La Brea Ave	Sunset Blvd
Los Angeles	La Brea Ave	Colorado Blvd

With Special Cartesian Processing, the resulting dataset is:

City	Avenue	Boulevard
Los Angeles	Melrose Ave	Hollywood Blvd
Los Angeles	La Brea Ave	Sunset Blvd
Los Angeles		Colorado Blvd

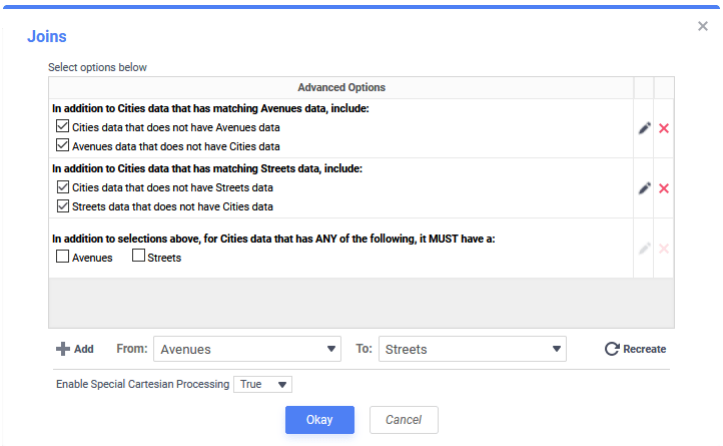
As long as we remember that there is no implied relationship between Avenue and Boulevard, we will get a much more readable table. The downside is that this will take longer to execute than the Cartesian product, especially for large data sets.

Enabling Special Cartesian Processing

By default, Special Cartesian Processing is on for all reports that have multiple one-to-many joins. To turn it off by default, set **Admin Console** > **General** > **Database Settings** > **Enable Special Cartesian Processing** to *False*. Note, this does not make the feature unavailable at the report-level.

To enable or disable Cartesian processing for a specific report, click **Report Options** > **Advanced** > **Joins** to open the Joins window, then select True or False from the **Enable Special Cartesian Processing** dropdown.

> **NOTE**
>
> The **Enable Special Cartesian Processing** dropdown is only available when a report has multiple one-to-many joins. It is hidden otherwise.



Parameters

This article applies to the **Admin Console >** **Data >** **Parameters** settings.

Parameters are used throughout the Exago application to store values. Although parameters can be created and given a default value in the **Administration Console**, parameters are designed to be set at runtime through the **API**.

Parameters can be used to:

- Pass values to Web Services, .NET Assemblies, or custom SQL Data Objects.

IMPORTANT

Parameter values are passed in the order in which they are listed in a data object. It is critical to ensure that the order is correct. For more information please see the **Data Objects** article.

- Set tenant values to assure security in a multi-tenant environment. For more information see **Data Objects**.
- Pass values into cells and formulas of a report. To display a non-hidden parameter in a cell type “@”.
- Pass values into custom functions. For more information see **Custom Functions**.
- Create a custom dropdown list of values for user selection on a report prompt.

Create, Edit, Delete Parameters

All existing parameters are listed in the **Admin Console Main Menu** under **Data**. All the parameters that are added or edited will be displayed in a tab entitled **Parameters**.

- To **add** a new parameter click **Parameters** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
- To **edit** a parameter either:
  - double click it
  - select it and click the **Edit** icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** a parameter either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new parameters click the **Apply** or **Okay** buttons.

NOTE

Currently, the prompt window will not appear when drilling down into a child report that contains prompting parameters. However, this can be partially bypassed by adding the prompting parameter in a hidden cell on the parent report so that the user will be prompted at execution for the parameter value.

Parameter Properties

Each Parameter has the following properties:

Name

A name for the parameter. Prompting parameters are sorted alphabetically by name unless otherwise specified or unless there are dropdown parameters with dependencies.

The following characters are not allowed: @ , . { } [ ]

The following names are reserved by the application: filter, email, userId, companyId, reportName, reportFullName, pageNumber, data_label, data_value, series_label, bubble_size, bubble_value, batch_*x* (where *x* can be any string of characters).

IMPORTANT

Parameter names are case sensitive.

Type

The parameter data type. The following types are available:

- *string*
- *date*
- *integer*
- *decimal*
- *boolean* (*v2019.1+*)

NOTE

Select *date* for DateTimes and *string* for functions.

Value

The default value of a parameter. This is intended to be overwritten at runtime through the API.

Date values should be entered in yyyy-MM-dd format.

Null values may also be passed to parameters. In order to do so, the default value of the parameter needs to be set to {null}. This will allow parameters to process any null values that are passed through the API or application at runtime.

Formulas as Default Parameter Values

Formulas that return strings, integers, dates or decimals may also be passed as parameter default values. Just as entering a function into a cell in the Report Designer, include an equals sign as a prefix (e.g. =QuarterNumber(Today())).

Formulas may include multiple functions. As with other functions, these may also include other parameters.

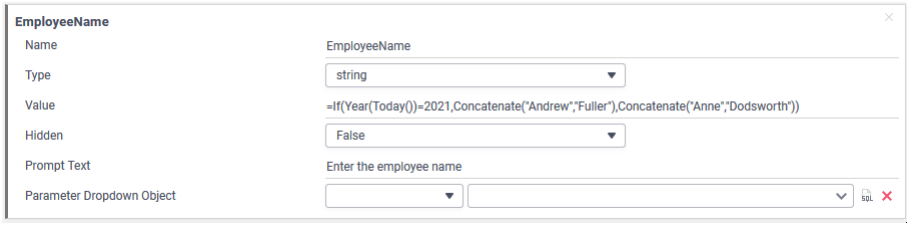
> **CAUTION**
>
> The **Admin Console** will only allow *String*, *Decimal* or *Integer* **Type** parameters to have a formula as their default **Value**. If you need the *Date* **type**, select *String* in the Admin Console, enter the formula into the **Value** field and then manually edit the XML configuration file to choose the desired data type.

Examples

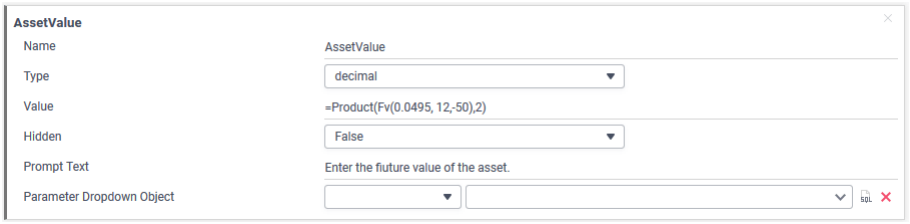
1. This parameter as defined in the XML configuration file is *date* type and by default will return the value of the Today() function, which is today's date.

```
<parameter>
  <id>DateChoose</id>
  <data_type>date</data_type>
  <value>=Today()</value>
  <hidden>False</hidden>
  <prompt_text>Choose a date</prompt_text>
</parameter>
```

2. This parameter defined in the Admin Console returns a *string* with an employee's name. The employee's name is decided based on the year of the current date.



3. This parameter defined in the Admin Console returns a *decimal* with the future value of an asset multiplied by 2.



Hidden

Set hidden to *True* to disable this parameter from being used by users in cells and formulas.

Prompt Text

Give non-hidden parameters a prompt text to query the user for a value at the time of report execution. Leave blank to use the default value.

Parameter Dropdown Object

Optional data object for populating the parameter as a drop-down selection list. Only applicable with prompting parameters. Commonly used in conjunction with programmable data objects (such as stored procedures).

Stored Procedure Parameters

A list of preexisting Exago parameters to be used as variables for a selected stored procedure.

Value Field

A column from the data object or custom SQL that sets that actual value of the parameter at runtime. This represents a set of values that are not displayed to the end user but are instead used when parameter values are required in custom SQL or stored procedures, or other server side processing.

Display Value Field

A column from the data object or custom SQL that sets the display value of the parameter for the dropdown selector. This represents the set of values that should be presented to the end user when they are executing or scheduling a report.

The Display Value is also displayed in reports and used in formulas (except in formula filters).

> **NOTE**
>
> These distinct values can be accessed in formulas with @ and @ respectively.

Display Type

The display value data type.

Report-Level Parameters v2019.1.3+

Report builders can define their own parameters on the report-level. These parameters function identically to parameters existing on the system-level and may be used within filters, functions, and so on within the report. Parameters defined in this way are only accessible within the report that they are created on. For more information, please see the **Report-Level Parameters** section of the **Advanced Options** article.

> **IMPORTANT**
>
> Report-level parameters cannot be used in report-level **Custom SQL Objects** in versions *pre-v2021.1.1*.

This feature is enabled by default and can be disabled by setting **Show Report-Level Parameters Window** to *False*. This setting is located in the **Feature/UI Settings** section of the **Admin Console**.

Modifying Report-Level Parameters via the .NET API

To modify existing report-level parameters:

1. Load the report that the parameter is defined on.
2. Iterate through the parameters collection to find the parameter.
3. Finally, adjust the required values of the parameter
4. Save it back to the API.

> **NOTE**
>
> Currently, report-level parameters cannot be loaded or modified via the JavaScript API.

Example

```
ReportObject report = (ReportObject)api.ReportObjectFactory.LoadFromRepository(@"Reports\Sales Reports");
report.Parameters.Find(x => x.Id == "SalesQuota").Value = "1500";
api.ReportObjectFactory.SaveToApi(report);
```

Roles

This article applies to the **Admin Console** >

Roles settings.

This article explains how to use the **Roles** to control access to data and override the **General Settings**.

Create, Edit, Delete Roles

- To **add** a new role click **Roles** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
- To **edit** a role either:
  - double click it
  - select it, expand the sections (*Main*, *General*, *Folders*, *Objects* or *Filters*) and click the **Edit** icon at the top of the main menu
- To **delete** a role either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new joins click the **Apply** or **Okay** buttons.

The Role that is currently active will be highlighted under the **Roles** menu of the **Admin Console**.



User is the currently active role

About Roles

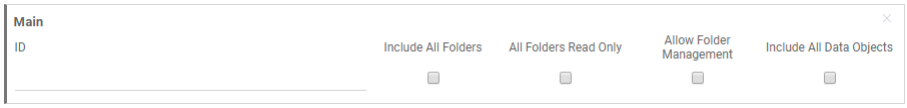
Roles are created to specify how a user or group of users interfaces with Exago. Roles can restrict access to folders or data objects. Roles can also override the **General Settings**.

> **NOTE**
>
> Exago was designed to be an integrated reporting solution for other applications using the applicationâ€™s own security and authentication methods. Although you can create Roles through the **Admin Console**, Roles are typically created through the API to dynamically set a user's access. For more information see the sections regarding the various APIs.

Roles have five sections to control access: **Main**, **General**, **Folders**, **Objects**, and **Filters**.

Main Settings

The **Main** settings control the broad properties of the Role.



Id

A name for the role.

Include All Folders

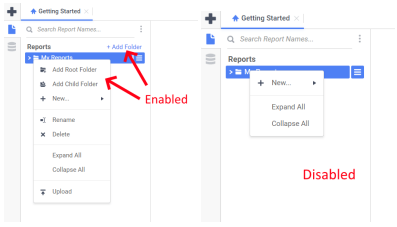
If checked, all folders that are not listed in **Folder Access** will be available. If unchecked, only those listed in **Folder Access** will be available.

All Folders Read Only

If checked, all folders that are not specified in **Folder Access** will be execute-only. If unchecked, only those specified in **Folder Access** will be execute-only.

Allow Folder Management

If *True*, users will be able to create root and child folders, rename folders and delete folders in the user interface. If *False*, users cannot take action on folders and the Add Folder buttons are hidden from the user interface.



Include All Data Objects

If checked, all Data Objects that are not listed in Objects section will be available. If unchecked, only those listed in Objects Access will be available.

General Settings

The **General** settings of a Role override the some of the global **General Settings**. Utilize the API in order to overwrite additional settings for a user or group of users.

Date Format

The format of date values. Can be any .NET standard (ex. MM/dd/yyyy). If left blank, this setting will inherit the environmental settings.

Time Format

The format of time values. Can be any .NET standard (ex. h:mm:ss tt). If left blank, this setting will inherit the environmental settings.

Date Time Format

The format of date-time values. May be any .NET standard (ex. M/d/yy h:mm tt). If left blank, this setting will inherit the environmental settings.

> **NOTE**
>
> For more details on .NET Date, Time and DateTime Format Strings, review the Standard date and time format string document on Microsoft's website.

Numeric Separator Symbol

Symbol used to separate 3 digit groups (ex. thousandths) in numeric values. The default is â€¯,â€™.

Numeric Currency Symbol

Symbol prepended to numeric values to represent currency. The default is â€¯$â€™.

Numeric Decimal Symbol

Symbol used for numeric decimal values. The default is a period (.).

Server Time Zone Offset

Value that is used to convert server to client time (the negation is used to convert client to server time). If left blank, this setting will use server time to calculate value.

Time Zone Name v2019.1+

Default geopolitical location of the client as determined by the IANA time zone database (e.g., America/New_York). Utilized by Exago to properly combat scheduling issues centering around DST and other non-linear time zone conflicts.

Show Grid Lines in Report Viewer

Sets the default value for the HTML output option Show Grid. This can be modified in the Options Menu of the Report Designer.

Allow Creation/Editing of Advanced Reports

Displays/Hides the Report Designer.

> **NOTE**
>
> IfÂ **Allow Creation/Editing of Advanced Reports** is *False* then attempts to edit Advanced or CrossTab reports will cause an â€˜access deniedâ€™ message. Additionally, if *False*, users will not be able to create CrossTab reports.

Allow Creation of CrossTab Reports

When *True*, users can select to add a new CrossTab Report from the New Report icon in the main menu. When *False*, CrossTabs may only be added to an Advanced Report, provided that **Show CrossTab Wizard** is also *True*.

Show CrossTab Wizard

When *True*, users can add CrossTabs to their existing Advanced Reports from within the Advanced Report Designer, with the Insert menu.

Allow Creation/Editing of Dashboards

Enables/Disables the Dashboard Designer.

Allow Creation/Editing of Chained Reports

Enables/Disables the Chained Report wizard.

Allow Creation/Editing of ExpressViews

Enables/Disables the ExpressView designer.

Allow Editing ExpressView with Live Data

When *True*, users are allowed to make changes to an ExpressView when the in Live Data (**Run**) mode. When *False*, users must click the **Stop** button on the toolbar before changes may be made.
The default value is *false* for *pre-v2021.1.1* or *true* for *v2021.1.1+*

> **NOTE**
>
> Prior to *v2021.1.1* we recommend setting this to *False*. Editing live ExpressViews will cause an increase in Data Source requests, and may reduce performance.

Database Timeout

Maximum number of seconds for a single query to run.

> **CAUTION**
>
> This value is passed on to the data source driver when the report is executed. Each data source driver, data source or individual database may interpret this setting slightly differently. Consult the official product documentation associated with your data source implementation for complete details how this setting will impact your environment.

> **NOTE**
>
> This setting will also control the maximum number of seconds that a Web Service API method can run. If set to *0* the Web Service API time out will be *infinite*.

Read Database for Filter Values

Enable/Disables filter drop-downs to contain values from the database. Set to *False* only if retrieving values for the drop-down will take more than a couple of seconds.

Show Report Scheduling Option

Displays/Hides the scheduler icon on the Main Menu. Set to *False* to disable users from creating scheduled reports.

Show Email Report Options

Displays/Hides the email report icon on the Main Menu. Set to *False* to disable users from emailing reports.

Show Schedule Reports Manager

Displays/Hides the scheduler manager icon on the Main Menu. Set to *False* to disable users from editing existing schedules.

Scheduler Manager User View Level

Controls what information each user can see in the Schedule Manager. These levels utilize the Parameters companyId and userId. There are three possible values:

- **Current User:** Can only view and delete report jobs that have been created by that user.
- **All Users in Current Company:** User can only view and delete report schedules for their company.
- **All Users in All Companies:** User can view and delete report schedules for all companies (administrator).

Allow Creation of Custom SQL Objects in Advanced Reports v2018.1+

Allow this role to create and execute reports with report-level custom SQL objects.

> **WARNING**
>
> Carefully review the Report Level SQL article before enabling.

Allow Creation/Editing of Dashboard Visualizations

Display/Hide the Visualization item in the New Tile Menu and Tile Properties Pane of the Dashboard Designer.

Cache Visibility Level

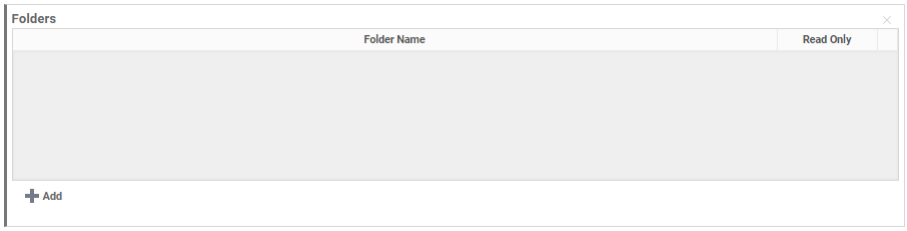
Controls what visibility permissions users can assign to **Execution Caches**. Available levels utilize the parameters **companyId and userId**. See Scheduler Settings for more information.

Folder Access

The **Folder Access** controls which report folders are visible and executable for the Role.

> **CAUTION**
>
> The Report Tree should contain no more than 1,000 items in it for best user experience.

All folders available in the Storage Management database will be available for selection here, regardless of the values of the identity keys. Access can only be granted to content that a user would already have access to through the Storage Management system.



> **NOTE**
>
> If **Include All Folders** is checked this list will deny access to the folders added. If unchecked, the list will allow access to the folders added. If **All Folders Read Only** is checked this list will overwrite the setting when a folder is added without the 'Read Only' option checked.

To add a folder to the list:

1. Click the ➕ **Add** button.
2. Click in the **Folder Name** column and select the folder you want to add.
3. To make the folder execute only check the box in the **Read Only** column.

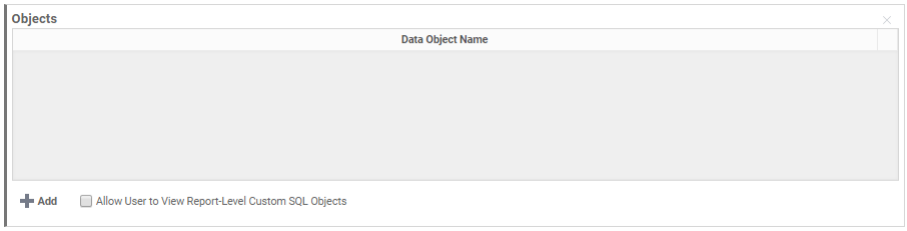
To delete a folder click the **Delete** ✕

icon at the end of the row.

Objects Access

The **Objects Access** controls which data objects are accessible to the Role. A report can only be executed if the Role has access to all the data

Objects on the report.



> **NOTE**
>
> If **Include All Data Objects** is checked this list will deny access to the data objects added. If unchecked the list will allow access to the data objects added.

To add a data object click theÂ ➕

Add button.

Click in the **Data Object Name** column and select the object you want to add.

To delete an object click the **Delete**Â ✕

icon at the end of the row.

Allow User to View Report-Level Custom SQL Objects v2018.1+

IfÂ **Allow Creation of Custom SQL Objects in Advanced Reports** is *False*, enable this setting to allow this role to run reports with report-level custom SQL objects. (Otherwise users will receive an “Access Denied” message when running such reports).

Filters Access

Filter Access provides a means to filter a Data Object by Role. Filters can be added to data objects so that users can only view specific rows in the data object. To set row-based tenancy in Exago:



1. To add a filter click the ➕ **Add** button.
2. Select the **Data Object Name** from the dropdown.

> **IMPORTANT**
>
> A Data Object may only appear in the list once. Adding the same Data Object multiple times is not allowed and will return an error message, “Please address the following errors before saving: Filters table contains the same object in multiple rows.”

3. Enter the **Filter String**. The filter string should be valid, standard SQL to be added to the ‘WHERE’ clause sent to the Data Source. This filter string can include Exago system parameters (e.g. @) or custom SQL including subqueries (e.g. IN (SELECT EmployeeID FROM Employees WHERE EmployeeID <4)). The **Filter String** must contain the actual name of objects in the Data Source (e.g. tables, views, columns, etc…) rather than their Exago aliases.

To delete a filter click the **DeleteÂ** ✕ icon at the end of the row.

Custom Functions

This article applies to the **Admin Console** > **Extensions** > *fx* **Functions** settings.

Exago comes with a number of functions that can be used to make formulas in the **Formula Editor**. Administrators may create additional custom functions using high level coding languages which will be accessible to end-users of the **Report Designer**. Functions can be added to a preexisting folder or a function can be put into a new custom folder.

Custom Functions can be written in C#, VB.NET, or JavaScript (Windows only). Functions can take as many input arguments as needed.

Functions can get and set elements of the current session state of Exago such as **Parameter** values. See **SessionInfo** for more information.

Create, Edit, Delete, Restore Custom Functions

- To **add** a new custom function click **Extensions >** *fx* **Functions** in the Main Menu and either:
  - click the **Add** ➕ icon at the top of the main menu
  - right-click and select ➕ **Add** from the context menu
- To **edit** a function either:
  - double click it

- select it and click the **Edit** icon at the top of the main menu
- right-click it and select **Edit** from the context menu

- To **delete** an object either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **restore** the default custom functions that come with the application either:
  - right-click **Extensions >** **Functions** in the Main Menu and select **Restore All Formula Functions** from the context menu
  - **General** > **Filter Settings** > **Restore All Default Formula Functions** and click on the **Restore** button.
- To **save** changes and new functions click the **Apply** or **Okay** buttons

Custom Function Properties

Each **Custom Function** has the following properties.

Name

A name for the function that will be displayed to the end users. Required.

Description

A description of the function that will be displayed to the end users.

> **NOTE**
>
> To support multi-language functionality, you can supply an Id from a language file instead of a static string. For more information, see **Multi-Language Support**.

> **NOTE**
>
> Arguments are passed to the code as an array of generic objects, accessed as **args[ ]**.

Category

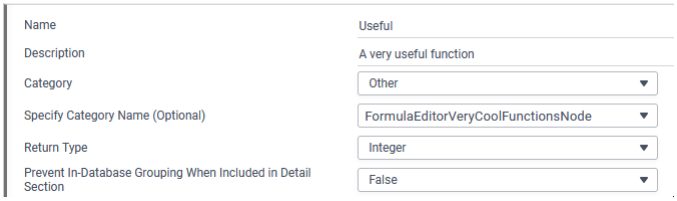
A way of grouping similar functions. You can assign custom functions to an existing Exago category or create a new category.

To create a new category:

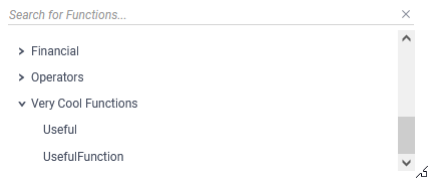
1. For all applicable language files, add a new language translation ID for the new category name. The element ID must begin with FormulaEditor and end with Node. For example:

   ```
   <element id="FormulaEditorVeryCoolFunctionsNode">Very Cool Functions</element>
   ```

2. Select *Other* from the **Category** dropdown list.
3. An input field entitled **Specific Category Name** will appear. Enter the language file ID created in step 1.



The language filed ID is provided in the **Specify Category Name (Optional)** field



Formula Editor dialog showing the custom category name
Very Cool Functions

> **IMPORTANT**
>
> Providing a static text string that does not match a language file ID for the **Specify Category Name (Optional)** field is not valid.

Return Type v2018.2.6+

A return type must be specified for each Custom Function. This field can be defined as a *String*, *Integer*, *Decimal*, or *Date*, and determines the expected return type of each function.

Prevent In-Database Grouping When Included in Detail Section v2019.1.5+

You may use this flag to specify whether this function has side-effects and should disqualify including reports from aggregating in the database (thus skipping the function when in a suppressed Detail section). For example, most functions that build or manipulate memory storage with Get/SetStoredValue should have this flag set to true.

Language

The high-level language of the code for the function. CanÂ be C# or VB.NET, or JavaScript (Windows only).

References

A semicolon-separated list of any DLLs that need to be referenced by the Custom Function. The DLLs must be present in the bin folder of the Exago Web Application, Scheduler Services, and the Web Service API if applicable. This folder can be found in the installation directory of the respective component.

> **TIP**
>
> System.dll does not need to be listed as a reference as it is already available.

Namespaces

A semicolon-separate list of any namespaces that need to be referenced by the Custom Function.

Program Code

The program code for your Custom Function. Press the green check mark to verify the code executes properly. Required.

> **NOTE**
>
> To use a .NET Assembly for custom functions, first add it to the applicable bin folders. Then add the assembly as a reference in the Custom Code window, and invoke the method, e.g., return Extensions.Functions.DayBefore(args);.

Arguments

Starting with version *v2017.2* there are several enhancements to the way function arguments are implemented and used.

Click **Edit Argument Info** to show a dialog for managing arguments. Then click **Add Argument** for each argument the function will have. Arguments have the following properties:

Name

The name of the argument, which will appear as a placeholder in the function parentheses and in the function's description tooltip.

Description

A description for what the argument is used for. You should mention the expected data type, if it is not obvious. This will appear in a tooltip when the placeholder name is selected.

Can also be a **language file ID**.

Optional

Whether this argument is required or optional. Optional arguments are surrounded by brackets [ ] in the function's description tooltip.

Variable Argument Count

If selected, the last argument in the list accepts more than one value. Variable arguments are followed by an ellipsis (…) in the function's description tooltip.

Exago Session Info

Custom Functions can access the Exago session state through a "sessionInfo" object. Access to sessionInfo allows powerful new capabilities such as the ability to persist values across function invocations, allowing each invocation to be aware of previous calls and behave accordingly.

> **NOTE**
>
> **sessionInfo** can also be accessed in **Server Events**, **Action Events**, and **Assembly Data Sources**.

Properties

PageInfo

This is the parent of all information in the current session. Included is the active Report and SetupData objects.

> **NOTE**
>
> Since the Report and SetupData objects are accessed frequently, direct pointers are included for these objects.

Report

An object that contains all of the reportâ€™s Data Object, sort, filter, and layout information.

SetupData

An object that contains all of the sessionâ€™s configuration setting including Filters, Parameters, Data Objects, Joins, Roles, etc.

CompanyId

A direct pointer to the companyId **Parameter** value.

UserId

A direct pointer to the userId **Parameter** value.

Methods

GetReportExecuteHtml(string reportName)

A method that executes the specified report and returns its html output. This could be used to embed a report within a cell of another report.

> **NOTE**
>
> The reportName is relative to the sessionâ€™s report path.

GetParameter(string parameterName)

A method that returns the specified Parameter Object. ‘GetParameter’ first looks in the Report Parameter collection, parameters being utilized by the report, and then in the Config Parameter collection, as well as other parameters such as hidden parameters or multi-tenant values.

GetReportParameter(string parameterName)

A method that returns the specified Parameter object that is utilized by the report being executed.

GetConfigParameter(string parameterName)

A method that returns the parameter object stored in the default configuration.

WriteLog(string text)

A method that writes the specified text to the Exago log file.

> **NOTE**
>
> The following methods utilize Stored Values which are objects that can be created and set by custom functions during report execution to pass data between custom function calls. Stored Values only exist for the duration of report execution.

GetStoredValue(string valueName, [object initialValue = null])

A method that retrieves a Stored Value. If a there is no Stored Value with the specified valueName, then one will be created with the specified initialValue.

SetStoredValue(string valueName, object newValue)

A method that sets the value of a Stored Value. Passing ‘null’ to newValue will delete the Stored Value.

Calling Exago Functions

Cases may arise where you want to call an existing function within your Custom Function. Using the class CellFormula and returning the method CellFormula.Evaluate().

Default Custom Functions v2017.2+

Exago BI ships with several built-in custom functions in *v2017.2+*. These are functions that are common in many reporting environments, but the manner in which they work may be different depending on locality, time zone, or other factors. For this reason, these functions have been exposed in the Admin Console so that administrators may change how they work.

> **NOTE**

If these functions are unavailable, such after an upgrade, in the **Admin Console** navigate to **General** > **Filter Settings** > **Restore All Default Formula Functions** and click on the

Restore button.

The following custom functions ship with Exago BI:

MonthName

Description	Given a date value, returns the name of the month of that date. The month name is retrieved from the active language dictionary. For example, given the date “01/01/2017”, MonthName() will return “January” in an English-speaking environment, and “Enero” in a Spanish-speaking environment.
Arguments	• inputDateOrNumber: A formatted date string, a Date or DateTime value, or an integer value. Required.
References	• mscorlib.dll
Example	These examples are in an English-speaking environment: MonthName("02/29/2020") February MonthName(3) March MonthName(1989-07-06 22:19:21) July

QuarterName

Description	Given a date value, returns the fiscal quarter of the date, as “Q1”, “Q2”, “Q3”, or “Q4”. By default, Q1 encompasses January 01 – March 31, Q2 encompasses April 01 – June 30, Q3 encompasses July 01 – Sept 30, and Q4 encompasses Oct 01 – Dec 31. But since different countries or financial landscapes may use different systems of quarters, the behavior of the function is exposed and customizable.
Arguments	• inputDate: A formatted date string, Date value, or DateTime value. Required.
Example	QuarterName("02/29/2020") Q1 QuarterName(1989-07-06 22:19:21) Q3

QuarterNumber

Description	Given a date value, returns the fiscal quarter of the date, as “1”, “2”, “3”, or “4”. By default, Q1 encompasses January 01 – March 31, Q2 encompasses April 01 – June 30, Q3 encompasses July 01 – Sept 30, and Q4 encompasses Oct 01 – Dec 31. But since different countries or financial landscapes may use different systems of quarters, the behavior of the function is exposed and customizable.
Arguments	• inputDate: A formatted date string, Date value, or DateTime value. Required.
Example	QuarterNumber("02/29/2020") 1 QuarterNumber(1989-07-06 22:19:21) 3

Custom Filter Functions

This article applies to the **Admin Console** > **Extensions** > **Filter Functions** settings.

Custom Filter Functions provide the ability to make functions that will dynamically calculate a value for a filter using high level code.

Filter Functions can be written in C#, JavaScript, or VB.NET. Filter Functions written in C# and VB.NET can get and set elements from the current session of Exago, such as Parameter values. See **Exago Session Info** for more information.

Create, Edit, Delete, Restore Custom Filter Functions

- To **add** a new custom filter function click **Extensions >** **Filter Functions** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
- To **edit** a function either:
  - double click it
  - select it and click the **Edit** icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** an object either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **restore** the default custom filter functions that come with the application either:
  - right-click **Extensions >** **Filter Functions** in the Main Menu and select **Restore All Default Date Filter Functions** from the context menu
  - navigate to **General** > **Filter Settings** > **Restore All Default Date Filter Functions** and click on the **Restore** button.
- To **save** changes and new functions click the **Apply** or **Okay** buttons

Custom Filter Function Properties

Each Custom Filter Function has the following properties:

Name

A name for the filter function that will be displayed to the end users. Required.

Description

A description of the function.

> **NOTE**
>
> To support multi-language functionality, if the filter functionâ€™s name or description can be prepended with _wrFunctionId. If this ID matches the ID of any element in the language files, then the string of that language element will be displayed to the user instead of the function name/description in the Admin Console. For more information see **Multi-Language Support**.

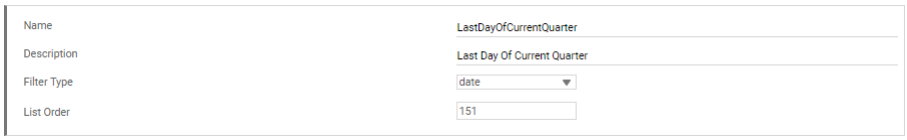
Filter Type

Determines the data type that the filter function should be available for. Either *Date*, *String*, *Integer* or *Decimal*.

List Order

The order the filter function will appear among other filter functions of the same type. Functions with a lower number will appear higher on the list. If two functions have the same list value they will display in alphabetic order.

> **NOTE**
>
> All of the built in filter functions start with list value 100 or greater.



Language

The high-level language of the code for the date function. May be C#, JavaScript, or VB.NET.

References

A semicolon-separated list of any DLLs that need to be referenced by the Custom Filter Function. The DLLs must be present in the bin folder of the Exago Web Application, Scheduler Services, and the Web Service API if applicable. This folder can be found in the installation directory of the respective component.

> **NOTE**
>
> System.dll does not need to be listed as a reference as it is already available.

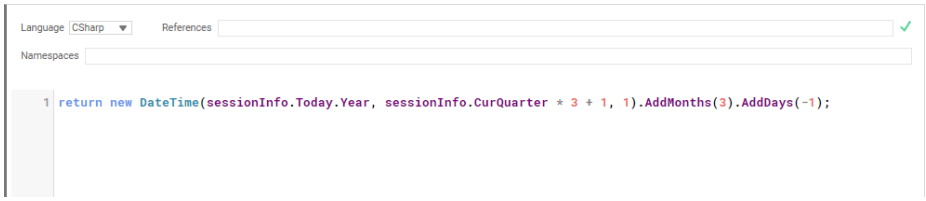
Namespaces

A semicolon-separated list of any namespaces that need to be referenced by the **Custom Filter Function**.

Program Code

The program code for your Custom Filter Function. The code must return the data type that was set in the **Filter Type** setting.

> **NOTE**
>
> Parameters may be referenced within custom functions by placing their name between @â€™s. For example, @



A Custom Filter Function that returns the date value of the last day of the current quarter

Click the **Test custom code execution** ✓

icon to verify that the code properly compiles.

Default Custom Filter Functions v2016.3+

Exago comes with a number of built-in custom filter functions. They are exposed in the Admin Console so they may be modified as needed.

> **NOTE**
>
> If these functions are unavailable, such as after an upgrade, follow the **restore** instructions in the Create, Edit, Delete, Restore Custom Filter Functions section above to add them back to the configuration automatically.

In all of the examples below, today's date is assumed to be **Friday, February 26, 2021**.

FirstDayOfCurrentMonth

Description	Returns a DateTime representing the first day of the current month
Example	2/1/2021

FirstDayOfCurrentQuarter

Description	Returns a DateTime representing the first day of the current quarter
Example	1/1/2021

FirstDayOfCurrentYear

Description	Returns a DateTime representing the first day of the current year
Example	1/1/2021

FirstDayOfLastMonth

Description	Returns a DateTime representing the first day of last month
Example	1/1/2021

FirstDayOfLastQuarter

Description	Returns a DateTime representing the first day of last quarter
Example	10/1/2020

FirstDayOfLastYear

Description	Returns a DateTime representing the first day of last year
Example	1/1/2020

FirstDayOfNextMonth

Description	Returns a DateTime representing the first day of next month
Example	3/1/2021

FirstDayOfNextQuarter

Description	Returns a DateTime representing the first day of next quarter
Example	4/1/2021

FirstDayOfNextYear

Description	Returns a DateTime representing the first day of next year
Example	1/1/2022

FridayOfCurrentWeek

Description	Returns a DateTime representing the Friday of the current week
Example	2/26/2021

FridayOfLastWeek

Description	Returns a DateTime representing the Friday of the previous week
Example	2/19/2021

FridayOfNextWeek

Description	Returns a DateTime representing the Friday of the next week

Example	3/5/2021

LastDayOfCurrentMonth

Description	Returns a DateTime representing the last day of the current month
Example	2/28/2021

LastDayOfCurrentQuarter

Description	Returns a DateTime representing the last day of the current quarter
Example	3/31/2021

LastDayOfCurrentYear

Description	Returns a DateTime representing the last day of the current year
Example	12/31/2021

LastDayOfLastMonth

Description	Returns a DateTime representing the last day of the previous month
Example	1/31/2021

LastDayOfLastQuarter

Description	Returns a DateTime representing the last day of the previous quarter
Example	12/31/2020

LastDayOfLastYear

Description	Returns a DateTime representing the last day of the previous year
Example	12/31/2020

LastDayOfNextMonth

Description	Returns a DateTime representing the last day of the next month
Example	3/31/2021

LastDayOfNextQuarter

Description	Returns a DateTime representing the last day of the next quarter
Example	6/30/2021

LastDayOfNextYear

Description	Returns a DateTime representing the last day of next year
Example	12/31/2022

MondayOfCurrentWeek

Description	Returns a DateTime representing Monday of the current week
Example	2/22/2021

MondayOfLastWeek

Description	Returns a DateTime representing Monday of the previous week
Example	2/15/2021

MondayOfNextWeek

Description	Returns a DateTime representing Monday of the next week
Example	3/1/2021

Today

Description	Returns a DateTime representing today's date
Example	2/26/2021

TodayMinus30Days

Description	Returns a DateTime representing the date 30 days prior to today
Example	1/27/2021

TodayMinus60Days

Description	Returns a DateTime representing the date 60 days prior to today
Example	12/28/2020

TodayMinus7Days

Description	Returns a DateTime representing the date 7 days prior to today
Example	2/19/2021

TodayMinus90Days

Description	Returns a DateTime representing the date 90 days prior to today
Example	11/28/2020

TodayPlus30Days

Description	Returns a DateTime representing the date 30 days in the future from today
Example	3/28/2021

TodayPlus60Days

Description	Returns a DateTime representing the date 60 days in the future from today
Example	4/27/2021

TodayPlus7Days

Description	Returns a DateTime representing the date 7 days in the future from today
Example	3/5/2021

TodayPlus90Days

Description	Returns a DateTime representing the date 90 days in the future from today
Example	5/27/2021

Tomorrow

Description	Returns a DateTime representing tomorrow's date
Example	2/27/2021

Yesterday

Description	Returns a DateTime representing yesterday's date
Example	2/25/2021

Custom Aggregate Functions

Â

Exago comes with a number of functions that can be used to make formulas in the **Formula Editor**. Administrators may create additional custom functions, using high level coding languages, which will be accessible to end-users of the **Report Designer**. In Exago *v2019.2+* administrators may also create custom aggregate functions. Custom Aggregate Functions will be available in any formula editor in which the existing built-in aggregate functions are also available.

Custom Aggregate Functions can be written in C# or VB.NET and can take as many input arguments as needed. They can get and set elements of the current session state of Exago such as **Parameter** values. See the article on **SessionInfo** for more information.

Creating Custom Aggregate Functions

Custom Aggregate Functions may be created programmatically through the .NET API or manually through the Web Application. This section details how to create **Custom Aggregate Functions** through the Web Application but contains important details about programming the aggregate function which will apply to the .NET API as well.

1. In the **Admin Console**, navigate to **Extensions** > **Aggregate Functions** then click the **Add** ➕ icon at the top of main menu. This will open

a new **Custom Aggregate Function** tab.

Each **Custom Aggregate Function** has the following properties:

> **NOTE**
>
> The **Description** and **Category** properties below support multi-language functionality, where an Id from a language file instead of a static string may be supplied. For more information, see the article on **Multi-Language Support**.

Name

A name for the function that will be displayed to the end user and will be used to call the function. This name may only contain letters, numbers and underscores, and must be different from any of the built-in aggregate functions.

Description

A description of the function that will be displayed to the end user when building a formula in the **Formula Editor**.

Category

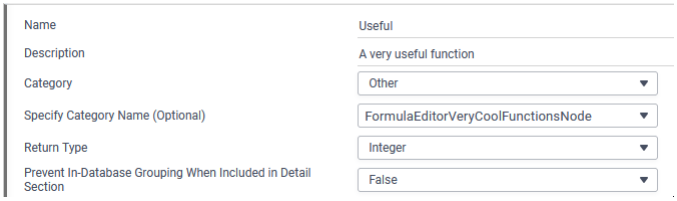
A way of categorizing/grouping similar functions together. **Custom Aggregate Functions** may be assigned to any existing category or to a new category. By default, custom aggregate functions are added to the *Aggregate* category unless a different one is chosen.

To create a new category:

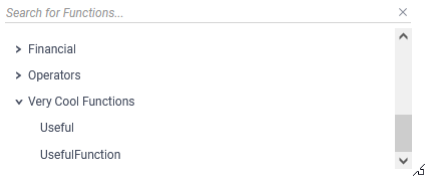
1. For all applicable language files, add a new language translation ID for the new category name. The element ID must begin with FormulaEditor and end with Node. For example:

   ```
   <element id="FormulaEditorVeryCoolFunctionsNode">Very Cool Functions</element>
   ```

2. Select *Other* from the **Category** dropdown list.
3. An input field entitled **Specific Category Name (Optional)** appears. Enter the language file ID created in step 1.



The language filed ID is provided in the **Specify Category Name (Optional)** field



Formula Editor dialog showing the custom category name
Very Cool Functions

> **IMPORTANT**
>
> Providing a static text string that does not match a language file ID for the **Specify Category Name (Optional)** field is not valid.

Return Type

A return type must be specified for each **Custom Aggregate Function**. This field can be defined as a *String*, *Integer*, *Decimal*, or *Date*, and determines the expected return type of the function.

Prevent In-Database Grouping When Included in Detail Section

If the **Custom Aggregate Function** is called from a **Detail** section of a report and that Detail section is suppressed (hidden), setting this flag to *True* will prevent Exago from requesting the aggregation be done in the database. Instead aggregation will be done in memory insuring that the **Custom Aggregate Function** is called for each Detail row.

If *False*, Exago's normal behavior determined by theÂ **Database Aggregation** setting is utilized. If applicable, aggregation will be done in the database and then the **Custom Aggregate Function** will be called afterwards. This might cause trouble if the function is intended to work on unaggregated or ungrouped data.

For example, most functions that build or manipulate memory storage with **GetStoredValue**/**SetStoredValue** should have this flag set to *True*.

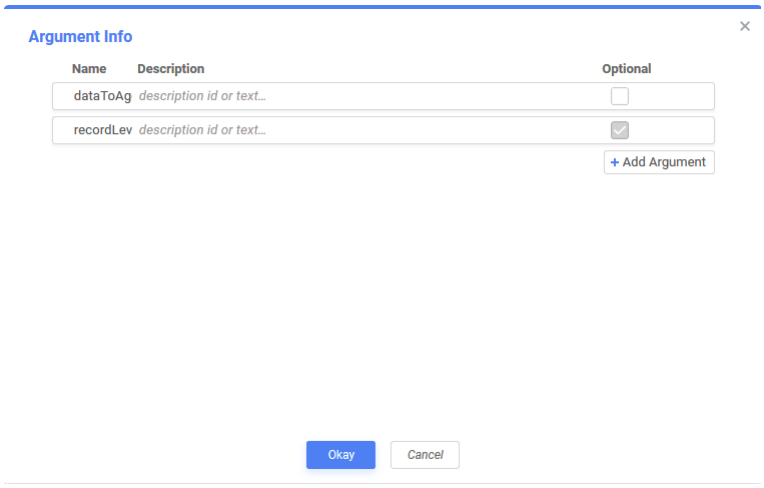
This flag will have no affect if the **Custom Aggregate Function** is called from any other section besides a **Detail** section.

NOTE

If Exago's **Database Aggregation** setting is *False*, then aggregations will never be done in the database regardless of the setting of the **Prevent In-Database Grouping When Included in Detail Section** flag.

Arguments

Click **Edit Argument Info** to show a dialog for managing arguments.



Edit Argument Dialog

Arguments are passed to the code as an array of generic objects, accessed as **args[]**. **Custom Aggregate Functions** have two required arguments, dataToAggregate and recordLevel.

- **dataToAggregate**: the data field or formula to aggregate. Can be renamed, but must always be the first argument.
- **recordLevel**: indicates whether aggregation should occur at the record level or entity level. If *true*, aggregation occurs on the record level. Can be renamed, but must always be the last argument. Refer to the articles on Aggregate Functions and the **Admin Console**'s Other Settings for more information on how this argument works.

IMPORTANT

recordLevel will evaluate to true if it is set to any true value (as returned by functions like True(), a nonzero number, or a non-empty string).

To add additional arguments to the function, click the **+**

Add Argument button.

To remove arguments from the function, click the **Delete Row** ✕ icon at the end of the respective row.

Arguments have the following properties:

NOTE

Both **Name** and **Description** properties below support multi-language, where an Id from a language file instead of a static string may be supplied. For more information, see the article on **Multi-Language Support**.

Name

The name of the argument, which will appear as a placeholder in the function parentheses and in the function's description tooltip.

Description

A description for what the argument is used for which will appear as a tooltip when the placeholder name is selected.

Optional

Check the box if a value for the argument does not need to be supplied when the function is called. Leave the box unchecked if a value must be supplied.

If a user does not supply a required argument, the formula editor will highlight the error.

Optional arguments are surrounded by brackets [ ] in the function's description tooltip.

Program Code

Language

The programming language of the code for the function. Either C# or VB.NET.

References

A semicolon-separated list of any libraries that need to be referenced by the **Custom Aggregate Function**. The DLLs must be present in the bin folder of the Exago Web Application, as well as any **Scheduler** bin folders, and the **Web Service API**, if applicable.

> **NOTE**
>
> System.dll does not need to be listed as a reference as it is already available.

Namespaces

A semicolon-separated list of any namespaces that need to be referenced by the **Custom Aggregate Function**.

WebReports.Api.Custom; WebReports.Api.Common; will be added automatically as they are required.

Code Editor

The code editor contains the program code for the **Custom Aggregate Function**. Unlike **Custom Functions**, the program code for **Custom Aggregate Functions** accommodates a full class definition.

A **Custom Aggregate Function** must contain at most one class that implements the ICustomAggregator interface. The code may include more than one class definition, but no more than one class should implement ICustomAggregator.

Depending on which language chosen, the code editor will pre-fill with a code stub as a general starting suggestion. This stub is an empty class which implements the ICustomAggregator interface. Examples of both C# and VB stubs are shown below.

C# Code Stub

```
public class MyCustomAggregator : ICustomAggregator
{
 public void AddValue(SessionInfo sessionInfo, object value, params object[] args)
 {

 }

 public object Result(SessionInfo sessionInfo)
 {

 }
}
```

VB Code Stub

```
Public Class MyCustomAggregator
   Inherits ICustomAggregator

   Public Overrides Sub AddValue(ByVal sessionInfo As SessionInfo, ByVal value As Object, ParamArray args As Object())
   End Sub

   Public Overrides Function Result(ByVal sessionInfo As SessionInfo) As Object
   End Function
End Class
```

> **NOTE**
>
> To use a .NET Assembly for custom aggregate functions, first add it to the applicable bin folders. Then add the assembly as a reference in the Code Editor.

During report execution, the conditions under which Exago will call AddValue change depending on what is passed to the **Custom Aggregate Function**â€™s *dataToAggregate* argument. This mirrors the behavior of the built-in aggregate functions:

- If a **data field** is passed, and the function's *recordLevel* argument is *false*, AddValue is called when the key of the data fieldâ€™s data object in the current record is different from that of the previous record. If *recordLevel* is *true*, AddValue is called for every record.
- If a **parameter** is passed, AddValue is called for every record.
- If a **formula** or cell reference is passed, AddValue is called for every record.

Result is called when the current break section has terminated. The aggregation result should be returned here. An explanation of when section processing ends is below in the following paragraphs.

When this **Custom Aggregate Function** is used on a report, Exago will instantiate a new instance of the class implementing ICustomAggregator once at the beginning of report execution, and again every time the section that the **Custom Aggregate Function** appears in breaks. This means that ICustomAggregator implementations will only aggregate within whichever section iteration is

currently being processed, and aggregation starts over at the beginning of each new section.

A section breaks when its corresponding sorting entity's unique key changes. For example, consider a report with Products organized into Category groups. To know when a new product Category starts and the previous Category ends (and thus when to display the appropriate section headers/footers and to perform aggregation), Exago looks for the unique key of the Category entity to switch to a new value, this is a section break.

If the custom aggregate function is in a:

- **Detail** section, a new ICustomAggregator is instantiated once before the first record is processed, then again before every other record is processed.
- **Group Header**, a new ICustomAggregator is instantiated once before the first record is processed, then again after every other Group Header with the same group break is processed.
- **Report Footer**, a new ICustomAggregator is instantiated only once, before the first record is processed.

Click the **Test Execution** ✓ icon to test the code for errors.

ICustomAggregator Interface

```
public interface ICustomAggregator
{
   void AddValue(SessionInfo sessionInfo, object value, params object[] args);
   object Result(SessionInfo sessionInfo);
}
```

The ICustomAggregator interface defines two functions, AddValue and Result. There must be one and only one class defined in the **Custom Aggregate Function**'s program code that implements both of these functions.

```
void AddValue(SessionInfo sessionInfo, object value, params object[] args)
```

- value: The value to aggregate. The cell data derived from the dataToAggregate argument passed to the custom aggregate function is passed here.
- args: subsequent arguments passed to the custom aggregate function are made available here. args[0] here will be the argument following dataToAggregate in the custom aggregate function call.

```
object Result(SessionInfo sessionInfo);
```

Called when the current break group has terminated. The aggregation result should be returned.

To complete the creation of the **Custom Aggregate Function**:

2. Click **Apply** or **Okay**. Exago will check the code for compilation errors and display a warning if any are found.
3. If there are no warnings, the **Custom Aggregate Function** is now available for use.

Editing Custom Aggregate Functions

1. In the **Admin Console**, navigate to **Extensions** > **Aggregate Functions**.
2. Click on the name of the **Aggregate Function** to be edited, then click on the **Edit** icon at the top of the main menu.
3. Review the section on creating **Custom Aggregate Functions** to understand all of the available fields.

Deleting Custom Aggregate Functions

1. In the **Admin Console**, navigate to **Extensions** > **Aggregate Functions**.
2. Click on the name of the **Aggregate Function** to be deleted, then click on the **Delete** ✕ icon at the top of the main menu.

Examples

Several examples of **Custom Aggregate Functions** can be found with the Custom Aggregate Functions tag in our Support Center.

Custom Options

This article applies to the **Admin Console** >

Extensions > **Custom Options** settings.

Custom Options allow for theÂ definition of settings that users can modify on a per-report basis in the **Report Designer**. Options can be accessed during report execution by **Server Events** or **Custom Functions**.

The name of each option can be controlled on a per-user basis using the **multi-language** feature. Custom Options can be *Integers*, *Decimals*, *Booleans*, *Strings* or *Lists*. Each data type provides an appropriate UI element for the user to select a value.

Create, Edit, Delete Custom Options

- To **add** a new custom filter function click **Extensions >Â Custom Options** in the Main Menu and either:
  - click the **Add** ✚ icon at the top of the main menu
  - right-click and select + **Add** from the context menu
- To **edit** an option either:
  - double click it
  - select it and click the **Edit** icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** an option either:
  - select it and click the **Delete** ✕ icon at the top of the main menu

- right-click it and select ✕ **Delete** from the context menu
- To **save** changes and new options click the **Apply** or **Okay** buttons

Custom Option Properties

Each **Custom Option** has the following properties:

Id

The unique Id or name of the custom option. The Id is used in accessing the option and may be displayed in the Custom Options Menu as the user sets its value on a report. Required.

The Id may not contain the following characters:

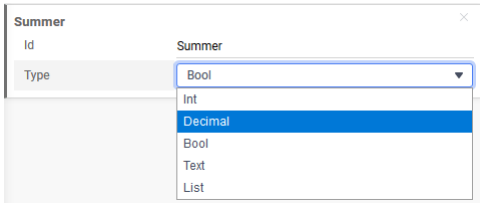
[ ] { } . , @ and whitespace.

> **NOTE**
>
> To support multi-language functionality, create an element in the language file(s) with an Id that matches the Optionâ€™s Id. The string of that language element will be displayed to the user in the Custom Options Menu. For more information see **Multi-Language Support**.

Type

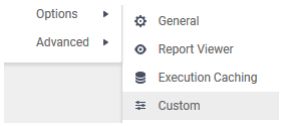
The data type the Custom Option should display. Each data type will display an appropriate input element in the Custom Options Menu. The following types are available.

- **Int**: Represents a whole number.
- **Decimal**: Represents a decimal number.
- **Bool**: Represents a Boolean value. A checkbox is displayed.
- **Text**: Represents a text string and displays a text box.
- **List**: Represents a single choice from among multiple values. Click the **AddÂ** ➕ icon to define list choices.



User Interaction with Custom Options

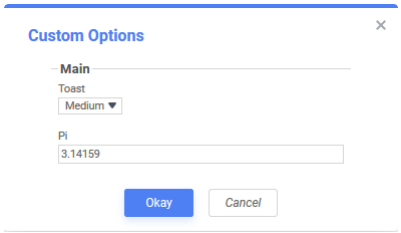
After **Custom Options** are created, the Custom Options Menu will be available in the Report Designer of Advanced and CrossTab Reports.



Custom Options in Report Designer Settings menu

> **NOTE**
>
> The Custom Options Menu will only display if Custom Options exist.

In the Custom Options dialog, values can be set using the user interface elements displayed above.



Custom Options dialog with one list and one decimal option

Accessing Custom Option Values

The .NET API, Server Events, and Custom Functions can access Custom Option values through the SessionInfo.Report object by using the following function:

string GetCustomOptionValue(string id)

Description	Returns the value of the specified **Custom Option** as a string.
	For *Bool* options the value returned will be *true* or *false*.

Remarks	For *List* Options, the chosen Id is returned. **NOTE** List options will return the Id of the selected value and not the displayed language string.
Example	A Custom Function could use the following C# code to return the value of a Custom Option. The Id of the Option is entered as an argument of the Custom Function. return sessionInfo.Report.GetCustomOptionValue(args[0].ToString());

Hidden Flags

The following options are inaccessible from the **Admin Console**, but may be toggled on or off or otherwise modified by editing the field in the XML configuration file.

- <errorondbrowlimitexceeded> v2021.1.16+ â€" When *True*, if the **Admin Console** > **General** > **Database Settings** > **Database Row Limit** is exceeded for a given report execution, the execution halts and a message is displayed to the user indicating that the report tried to pull in too much data. If the execution is the result of a scheduled report, the schedule abends and an error email is sent to the administrator. **.NET API Key:** *ErrorOnDbRowLimitExceeded*
- <convertjsonextracttosql> v2021.2+ â€" Set to *True* to enable all use of the JSONExtract function to be translated to SQL and pushed off to the applicable Data Source. When *False*, which is the default, JSONExtract will be always be done in-memory. Implementation of the JSONExtract function across Data Source types varies quite widely. Test this function in SQL mode (with value set to *True*) in a development or testing environment before enabling on a production server to insure compatibility and expected return results. For a list of supported JSONPath expressions and Data Source types, review the JSONExtract Function article.
- <UseExternalTimeZoneConverter> v2021.2+ â€" Set to *True* to enable the External Interface (a deprecated extensibility feature) to do time zone conversions. This behavior was previously enabled by setting the **Server Time Zone Offset** to null, which is no longer an acceptable value. The default value for this setting is *False*, so time zone conversions are handled internally with the **Time Zone Name** configuration file setting.
- <general><maxexportfilesizebytes></general> v2021.1.17+ â€" The maximum number of bytes that an export file may grow to before report execution is cancelled. *0* disables the limit, or any valid integer value greater than 0 cancels report execution and displays an error message to the user if the output file becomes larger than this number of bytes. .**NET API Key:** *MaxExportFileSizeBytes*
- <allowhtmlinscheduledemails> â€" Set to *True* to allow users to insert HTML tags within the body of scheduled emails.
- <filterdropdownobjecttenancy> â€" Set to *False* to disable tenanting on filter and parameter dropdown objects. If this is disabled and a tenant column limits the dataset for a data object, then a filter or parameter dropdown list within a report will show the full dataset for the data object *rather than* its intended limited subset.

  EXAMPLE

  A tenant column limits the dataset of an *Employees* data object so that only data for an employee with the last name “King” appears. A report that uses this data object will only output data associated with King. If <**filterdropdownobjecttenancy**> is *True* (default), then a filter on *Employees.LastName* added to this report will only allow King to be selected. However, if it is set to *False*, the filter on *Employees.LastName* will allow all employee last names to be selected even though only data for King will appear.

- <showbrowseroutofdatewarning> v2016.3.4+ â€" Set to *False* to prevent a popup error message from appearing if a user accesses Exago with an unsupported browser.
- <webfarmsupport> â€" If not using sticky sessions in a web farm environment, should be *True*. More temp information will be written to disk when *True* allowing other nodes in the farm to read it. Should be *False* if sticky sessions is configured or when Exago runs in a single node to stop unnecessary costly disk writes and keep information in memory instead.
- <expressviewdefaultformattheme> â€" v2017.1+ Select a default **ExpressView** theme. Provide the name of the file in the Themes directory or the name of the theme as it appears in the content table of the Storage Management database.
- <aliasallentities> â€" v2017.2+ Set to *True* to always use the Exago data object ID as table aliases in application generated SQL, rather then auto-generated aliases.
- <allowearlypagebreak> â€" v2017.3+ Set to *True* to allow users to insert a page break at the beginning of a report, which would cause the first page to be empty.
- <canjointransformobjectsindb> â€" v2018.1+ Set to *False* to cause vertical table transformations to be processed in the application instead of in the database. This prevents the use of advanced joins with vertical tables.
- <general><sqlgenerationlevel></general> â€" v2020.1+ Set the level of in-database formula grouping used by Exago. This value should be *1* unless otherwise directed by Exago Support. **.NET API Key:** *sqlgenerationlevel*
- <performancetesting> â€" v2018.2+ Set to *True* to set **Ctrl+Alt+A** to allow for running a report a specified number of times simultaneously for performance testing. Must be run from the report designer for the desired report.
- <reportlistcache> â€" v2018.2â€"v2020.1.0 Set to *False* to opt out of caching in the GetReportList method when using a Folder Management assembly.
- <webrequestkeepalive> â€" v2018.2+ Set to *True* to allow the use of a single TCP connection to send and receive multiple HTTP requests. If set to *False*, a new TCP connection will be opened for every HTTP request. This flag only applies to Google Maps widgets.
- <webrequestprotocolversion> â€" v2018.2+ Defaults to “1.1”, allowing the definition of 24 status codes as well as other advancements made present in HTTP protocol version 1.1. This flag only applies to Google Maps widgets.
- <showexpressreports> â€" v2019.2+ Set to *True* to enable the Express Report Wizard.
- <role><showexpressreportsstylingtoolbar></role> â€" v2019.2+ Set to *True* to display styling tools in the Layout tab of the Express Report Wizard for a role, or *False* to hide them.
- <role><showexpressreportsthemes></role> â€" v2019.2+ Set to *True* to display the Theme drop-down in the Layout tab of the Express Report Wizard for a role or *False* to hide it.

- <role><showexpressreportsgrouping></role> â€" v2019.2+ Set to *True* to show grouping tools in the Layout tab of the Express Report Wizard for a role or *False* to hide it.
- <role><showexpressreportsformulabutton></role> â€" v2019.2+ Set to *True* to show the Formula Editor button in the Layout tab of the Express Report Wizard for a role or *False* to hide it.

Any fields which are not mentioned here are either not intended for external use or not fully implemented and should be ignored.

Configuration Management and Propagation

Exago recommends installing at least three entirely separate environments: development, testing, and production. Designing an implementation strategy to move an Exago configuration from system to system is an inevitable and important process. As a highly-configurable environment, Exago consists of many components. This article will describe these components, where they are located, why they need to be part of the implementation strategy, and how they should be propagated from system to system.

For the sake of being all-encompassing, this article will assume that all components of Exago have been installed and are part of your deployment strategy. If you choose not to use one or more of these components, the respective sections may be skipped.

> **IMPORTANT**
>
> Many of the files listed below are impacted by the installer when updating to a new version of Exago. Please see the **Updating Recommendations** and **Updating to the Latest Version** articles regarding any files that need to be updated or recreated.

Exago Components

An Exago installation could include the following components:

- Web Application
- Web Service API
- Scheduler Service
- Monitoring Service
- Extensibility DLLs (Folder Management, Server Events, Action Events, Custom Functions, Scheduler Queue)

> **NOTE**
>
> This article references <WebApp>/, <WebSvc>/ and <Sched>/as a placeholder for the Web Application, Web Service API and Scheduler Service's install location respectively. The default install location is C:Program Files\Exago\ExagoWeb (/opt/Exago/ on Linux), C:Program Files\Exago\ExagoWebApi (/opt/Exago/WebServiceApi/ on Linux) or C:Program Files\Exago\ExagoScheduler (/opt/Exago/Scheduler/ on Linux); however, these directories can be changed during installation.

Web Application

The primary configuration for any Exago system is done via the **Admin Console**. There are countless configurations made with the Admin Console that are saved to two separate files. One of these files is a clear-text, standards-based XML file named WebReports.xml by default. It is appropriate for source control repositories given its consistent and clear-text nature. The other file, WebReports.xml.enc (derived from the name of the associated XML file), is an encrypted version of the aforementioned WebReports.xml file.

The encrypted version of the XML file should be considered to be the deployable version of the configuration. As such, it would be passed to *Configuration Management* as part of the deployment process. It is not necessary to deploy both the clear-text and the encrypted versions of the configuration created with the Admin Console. All configuration files created by the **Admin Console** are saved in the Config directory of the Exago Web Application.

> **NOTE**
>
> When the configuration settings have been finalized, it is recommended that the **Admin Console** be disabled and the plain-text config file be moved to a secure location outside of the Config directory. For more information refer to the **Security Checklist**.

Due to the fact that the configuration should progress from development to testing environments, and eventually to production via controlled processes, the **Admin Console** should only exist in development. While the **Admin Console** is installed as part of Exagoâ€™s base installation, it should be removed from any system outside of development.

In addition to the primary Exago configuration created using the **Admin Console**, there are numerous additional configuration files that may be relevant. All of these are optional, created or edited manually via a text editor on an as-needed basis, and override existing features or provide additional functionality. These files are saved as clear-text and are appropriate for source control repositories.

Language Files

All language files are saved as clear-text, standards-based XML files in the <WebApp>\Config\Languages directory.

Data Source Customization

Data Source customizations are saved as clear-text, standards-based JSON in the <WebApp>\Config\Other directory.

Application Themes

Application Themes include CSS, fonts, and images typically replicated saved in the <WebApp>\ApplicationThemes directory.

Element-Level Themes

Element-level theme files are saved as clear-text, proprietary-format files, and are created in a variety of ways. These files are used for the styling of Charts, CrossTab Reports, Express Reports, and GeoCharts, and are saved in the <WebApp>\Themes directory for *pre-v2020.1* versions, and in the Storage Management database for *v2020.1+*.

Folder Management and Monitoring Configuration

A separate config file used for providing configurations for Folder Management or Monitoring, located in <WebApp>.

Web Application Server Configuration

A separate config file used for providing configuration details to the web application server and/or Exago itself, this file is frequently modified. Common Exago configurations are used to set cookieless sessions, activate state servers, set connection strings, and add custom headers. This file is located in <WebApp>.

Database Drivers and Fonts

Database drivers may also be added to the system, assuming that the system doesnâ€™t ship with the appropriate divers already. These drivers would be saved as DLLs in the <WebApp>\Drivers directory.

Similarly, custom fonts may be included in the system. These would be saved in the <WebApp>\Fonts directory.

Due to the fact that these files are compiled components and are likely provided by third parties, drivers and fonts are likely not source code repository friendly, and would also need to be propagated from development through to production.

Web Service API

The **REST Web Service API** is an optional component. If applicable the Web Service API requires some configurable components similar to those listed as components of the Web Application. The following pertain to the Web Service API exactly as they pertain to the Web Application:

- Data Source customizations
- Drivers

In addition to the above, there are two clear-text configuration files that can be customized, based on the needs of the Web Service API itself:

XML File

The Web Service API XML file, defaulted as WebReportsApi.xml, is a clear-text, standards-based XML file, and is used to configure the connection between the Web Service API and the Web Application installation. It is located in the <WebSvc>\Config directory.

appSettings.config

The configuration file is a clear-text, standards-based XML file, and is used to activate the Web Service API. It is located in <WebSvc>.

Scheduler Service

The **Scheduler Service** is an optional component that executes reports an requires configurable components similar to those listed as components of the Web Application. The following pertain to the Scheduler Service exactly as they pertain the Web Application:

- Language files
- Data Source customizations
- Drivers
- Fonts

In addition to the above, there are two clear-text configuration files that can be customized based on the needs of the Scheduler itself:

XML File

The Scheduler configuration file, defaulted to <Sched>\eWebReportsScheduler.xml, is a standards-based XML file that includes configurations such as SMTP server connectivity configurations, communication channel and port information for connection to Exago itself, and many other configuration specific to the Scheduler Service.

exe.config

The Scheduler XML configuration file, defaulted to <Sched>\bin\eWebReportsScheduler.exe.config, is a standards-based XML file and is analogous to the Web Application's web.config file and may include configurations described previously.

Monitoring Service

Other than the monitoring configuration mentioned in the Web Application section, additional configurations can be made in the Monitoring.exe.config file. This file is formatted similar to the web application web.config file and saved in <WebApp>\MonitoringService.

Additionally, the current Monitoring implementation produces an SQLite database that is saved in <WebApp>\Monitoring. While this database is not configurable, if Monitoring is configured, its maintenance should be taken under consideration by Configuration Management.

Extensibility DLLs

While these are entirely optional components, Exago is frequently andÂ extensively customized via custom code compiled to DLLs. While these DLLs are configured via the aforementioned **Admin Console**, they need only be accessible via file system, meaning that they do not need to be deployed as part of Exagoâ€™s configuration. However, typical implementations of custom DLLs require components of Exago DLLs to be referenced, and, as such, require the maintenance of these custom implementations to be taken into consideration by Configuration Management.

Configuration File Options and Optimizations

In order to decrease the amount of resources needed to handle the configuration information, especially in situations where a large number of complex database objects are present, the option to implement a *static configuration* has been introduced. This facility is available to all clients as of *v2019.1+*.

In versions prior to *v2019.1*, the configuration file is read and loaded into memory for each interaction of the user with the system. In *v2019.1*, the system has been modified allowing a portion of the configuration to be extracted into a static configuration, which will be loaded into static memory and made available to all sessions. This process reduces the size of the configuration that must be loaded for each session.

These optimizations will considerably decrease configuration processing time, as well as decrease overhead in storing and executing scheduled reports.

> **NOTE**
>
> This is an optional feature and does not have to be implemented. While clients with large and generally invariant configuration files or those who programmatically generate configurations through the API may find this feature particularly useful, its implementation is not mandatory. For information regarding this, review the **Dynamic-Only Configurations** section below.

Configuration Structure

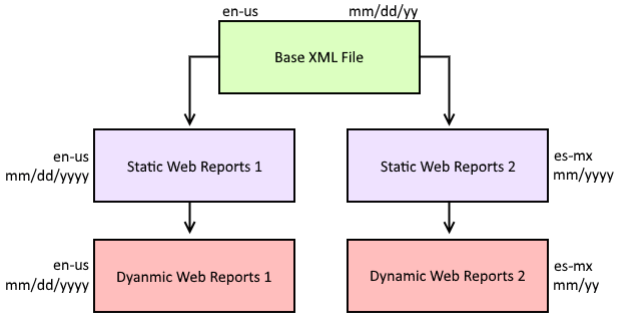
The configuration has been reorganized into a two-part hierarchical structure consisting of **static configurations** acting as *parents* to **dynamic configurations**. The contents of a parent configuration, which are saved to static memory and shared among all user sessions, are then extended directly to each user-specific dynamic configuration.

Configuration Types and Nomenclature

- **Dynamic**: A configuration that is retained in dynamic memory and must be saved or serialized to exist in user environments. This is the configuration type that should be specified when instantiating the API. All configurations *pre-v2019.1* were dynamic.
- **Static**: A configuration that is loaded into static memory and is available to all sessions. A static configuration must be specified as a parent of a dynamic configuration in order to be applied.
- **Parent**: A static configuration that is related to another configuration. The contents of a parent configuration extend its specified attributes to the child configuration.
- **Child**: A configuration that inherits all attributes present in the parent configuration. The child configuration may temporarily overwrite attributes it inherits from the parent configuration by changing their values or recreating them in the configuration file.

Architecture

The architectural diagram below is an example configuration where two static-dynamic configurations branch off of a single base static configuration. For ease of explanation, the only settings highlighted in each configuration file are the **Language File** and **Date Format** settings.



An example architectural diagram

In the first set of configuration files, the language and date formatting settings are identical. In this instance, these settings will be saved in static memory and referenced by each user session. In the second set of configuration files, the date formatting setting of the dynamic configuration differs from the value that is stored in memory by the static configuration. In this instance, the dynamic configuration will override this setting with each new session.

Implementation

In order to utilize this facility, the contents of the standard configuration will need to be separated into two files: the **static configuration** (parent) and the **dynamic configuration** (child).

The static configuration should only contain invariant information as it is not possible to reload the configuration without restarting the IIS thread and disrupting existing user sessions. This generally includes Entities (Data Objects), Joins, Functions, and Custom Options; however, the information that should exist in the static configuration is likely to vary from client to client.

> **NOTE**
>
> Data Objects, settings, and other values in the static configuration cannot be modified via API calls as doing so would disrupt existing user sessions. These items, however, may be temporarily overwritten in the dynamic configuration via the API.

The dynamic configuration should contain any information that is likely to be modified often or prior to the start of a user's session. This often includes the General Settings, Roles, Parameters, and when using database tenancy, Data Sources. Server Events and Action Events should be stored in the dynamic configuration as well.

For simplicity, this article will refer to the dynamic configuration as DynamicWebReports.xml and the static configuration as StaticWebReports.xml.

General Setup

The following list details how to migrate from a dynamic-only configuration to the new configuration architecture. Please note that these instruction assume a basic configuration infrastructure and setup may vary depending on the environment.

1. Duplicate your configuration file (e.g. WebReports.xml) twice, creating two new versions of the configuration. The first duplicate will become the static/parent configuration file and the second will become the dynamic/child configuration file.
2. Appropriately rename these configuration files (e.g. StaticWebReports.xml and DynamicWebReports.xml).Â The file name WebReports.xml is reserved and should not be used. As with pre-*v2019.1* configurations, these files should continue to exist in the **Config** folder of the Exago install directory.
3. Manually edit each configuration file, including the following information:
    - The static configuration file should contain only invariant objects and settings. The **Configuration File XML Reference** article contains a list of typically invariant objects.
    - Edit the dynamic configuration file to contain the <general> section and any other variable objects and settings. This file should essentially contain the remainder of what is not included within the static configuration file. Server Events and Action Events should be stored in the dynamic configuration as well.
4. Using the **Configuration Directive** information listed below, specify the type and parent of each configuration file.
5. Navigate to the **Admin Console** and verify that these changes have properly taken effect. For information on how to do this, please see the **Admin Console** section below.
6. The static configuration should be made available to all components of the system including the **Scheduler Service** and **Web Service API**.

    > **IMPORTANT**
    >
    > All static configuration files must be added to the **Config** sub-directory of each Scheduler Service instance.

7. The dynamic configuration should then be specified as the configuration file for each new session. For example, the file should be passed to the API constructor or specified in the initial POST session when using REST. This configuration will automatically inherent objects and properties from its parent static configuration.

Please continue reading for more information regarding implementation of this facility as well as relevant changes that have been made to Exago in order to accommodate these new configuration optimizations.

Configuration Directive

A new directive has been added to the configuration file:

```
<config type="[STATIC|DYNAMIC]" parent="[config]" />
```

Within this directive, the following information can be specified:

- type: Specifies whether the configuration file is dynamic or static.
- parent: Specifies the name of the parent configuration. This only needs to be specified within child configurations.

Dynamic Configuration

The following is an example of the configuration directive that should be added at the top of the dynamic configuration XML file:

```
<webreports>
  <config type="DYNAMIC" parent="StaticWebReports" />
  <general>
  ...
  </general>
</webreports>
```

Alternatively, the directive will be automatically applied or updated when the configuration options are set in the **Admin Console**. For more information, please see the **Admin Console** section below.

Static Configuration

The following is an example of the configuration directive that should be added at the top of the static configuration XML file:

```
<webreports>
  <config type="STATIC" parent="" />
  ...
</webreports>
```

Alternatively, the directive will be automatically applied or updated when the configuration options are set in the **Admin Console**. For more information, please see the **Admin Console** section below.

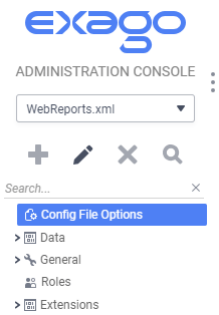
Admin Console

Within the **Admin Console**, the static and dynamic configurations can be fully managed.

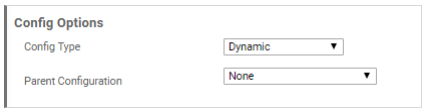
Configuration Options

After the standard, *pre-v2019.1* configuration has been split into its constituent dynamic and static parts, the directive for each configuration type can be implemented within the **Admin Console**.

Within the **Admin Console**, click on the **Config File Options** menu item.



This will open the **Config Options** tab

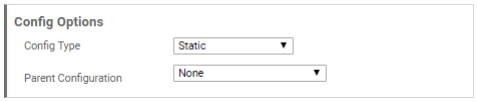


From here, the **Config Type** and **Parent Configuration** of the currently selected configuration can be specified.

First, the static configuration needs to be set up so that it can be visible to the dynamic configuration as a parent. To set up the static configuration, change the current configuration dropdown to StaticWebReports.xml. This option should be visible to the **Admin Console** once the configuration XML file is created.



Next, in **Config Options**, set the **Config Type** to *Static* and leave **Parent Configuration** set to *None*.

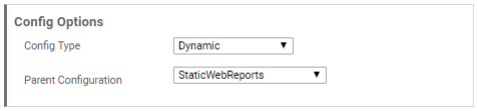


Click **Okay** to save these settings. Saving these settings will automatically update the configuration directive in StaticWebReports.xml to the specified information.

This static configuration should now be selectable as a parent to any dynamic configuration. To set up the dynamic configuration, change the current configuration dropdown to DynamicWebReports.xml.



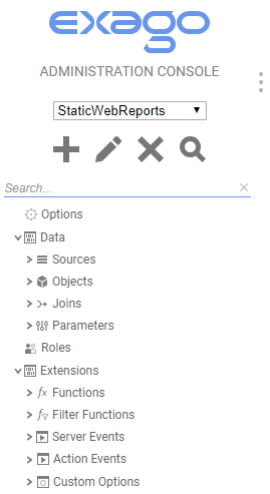
Next, in **Config Options**, set the **Config Type** to *Dynamic* and the **Parent Configuration** to StaticWebReports.xml.



Click **Okay** to save these settings update the configuration directive in DynamicWebReports.xml to the specified information.

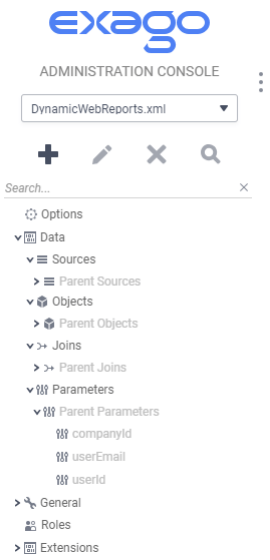
Managing the Static Configuration

The static configuration is managed similarly to *pre-v2019.1* configurations. Data Objects, Joins, Functions, and so on can all be manually added, edited, and deleted as normal.



Managing the Dynamic Configuration

Within the dynamic configuration, the **General** settings can be modified as it would in *pre-v2019.1* configurations. Note that objects like Data Objects, Server Events, and Functions can still be added into the dynamic configuration as well; however, any item or setting added or edited within the dynamic configuration will not be reflected within the static configuration.



Data Objects, Joins, Functions, and other invariant items and settings defined in the parent configuration will be visible but not editable within the dynamic configuration.

Dynamic-Only Configurations

Dynamic-only configurations can still be used in *v2019.1+* and function identically to *pre-v2019.1* configurations. To do so, the standard configuration can be left unmodified, and the following directive referenced at the beginning of the XML file:

```
<config type="DYNAMIC" parent="" />
```

> **NOTE**
>
> This information should automatically be added to the standard configuration file upon updating to *v2019.1*.

This directive specifies that the current configuration is dynamic-only since there is no reference to a parent static configuration.

Dynamic-only configurations may also be configured within the **Config Options** of the **Admin Console**. The *Config Type* of the standard configuration should be set to *Dynamic*, and its *Parent Configuration* should be set to *None*.

Object Adaptations

The following sections detail changes that have been made to Exago in order to accommodate these new configuration optimizations.

> **NOTE**
>
> This information has been provided for those interested in understanding the modifications that have been made to the Exago system and should not be pertinent to the actual implementation of this feature.

> **IMPORTANT**
>
> Please also review **Updating to the Latest Version**. This article highlights any breaking changes associated with updating Exago to newer versions, and it is imperative that this information is reviewed prior to updating.

PageInfo and Thread-Safe Configuration

In order to apply these configuration optimizations, objects in the configuration have been made *thread-safe*.

In *pre-v2019.1* configurations, objects retained references to the PageInfo context object at the time of their creation, which precluded them from being used in other contexts. Accordingly, these references to PageInfo have been replaced with a method that obtains the current context on demand. A single PageInfo context is now used and its instance is registered on the thread, allowing any object to request the current context.

> **NOTE**
>
> For more information on PageInfo, please see the **SessionInfo** article.

As a result, all configuration objects can now be stored in static memory and shared with all execution threads, providing a safe configuration that is always available and is not required to be loaded at every iteration.

Entity Objects

Due to its volatility, the Entity object â€" referred to as Data Objects in the **Admin Console** â€" is managed differently within the static configuration than other objects.

If the application attempts to change an Entity located within the static configuration, a copy of this Entity is created in the local configuration. This Entity object will not contain any data; instead, the user's references will be updated to point to an EntityData object. When the new, local EntityData object is created, the Entity object points to the new data and all references to that Entity automatically point to the new data, reflecting the changes that are made.

The shift in reference from the static to the local configuration happens automatically whenever a setter or update method is called on an Entity. The Entity objects have a delegate instance from the collection they are maintained in that enables registration to the new EntityData object after the copy is made.

Non-Entity Objects

Non-Entity objects do not implement the same methods as described above for Entities as an exception will be thrown when attempting to update the master configuration items.

All non-Entity objects located within the configuration tree implement a simpler structure. These objects are implemented as subclasses of ConfigCollection, and the elements of the collection are subclasses of ConfigObj. The default iterator searches through the local collection first and then processes the master configuration.

> **NOTE**
>
> Parameter objects are the only exception to this rule. Due to the dynamic nature of Parameter objects, the elements are instead copied from the master configuration into the local configuration.

The following list contains the collection classes derived from ConfigCollection and the source of the object identity:

- CategoryCollection | name
- JoinCollection | complex
- ParameterCollection | id
- UdfFunctionCollection | name
- DataSourceCollection | name
- ActionEventCollection | id
- ServerEventCollection | name

Unique Identifiers

All objects within the configuration must have a unique identifier. The following list is a reference for each object type's identifier type:

- Entity | name
- Data Source | id
- Join | complex
- Server Event | id
- Action Event | id
- Function | name
- Parameter | name
- Role | id
- Custom Option | id

> **NOTE**
>
> The Join object's unique identity type is automatically calculated by a concatenation of the object's EntityFromName, EntityToName, Type, Weight, and Category.

The client application may fail if identifiers are not unique.

Implementation FAQ

Can either the static or dynamic configuration file be modified in production without dropping user sessions?

The static configuration cannot be modified in production without having to restart IIS, or the associated pool, and dropping user sessions. Dynamic configurations may be modified without dropping user sessions.

The **OnConfigLoadStart** and **OnConfigLoadEnd** server events may also be used to modify dynamic configurations at session start.

How does the SaveConfigToFile() method work when loading the configuration in this manner? Does it generate two configuration files or just one master one?

The SaveConfigToFile() method will write to the dynamic configuration file as the static configuration file cannot be modified in production without having to restart IIS/dropping the user session.

This method is designed to save the current child configuration, which is always handled as dynamic. If a static configuration is set as the child of a separate static parent configuration and this method is called, then no effect will take place in any currently loaded sessions until an IIS/Pool reset.

In the .NET API, is there a way to know which configuration file each object is from?

Currently, there is no way to know which configuration file each object extends from in the .NET API.

When does the OnConfigLoadEnd server event trigger? After just the static configuration or after the full configuration is loaded?

The **OnConfigLoadEnd** event will be triggered after the **full configuration** is loaded.

This event is triggered after all API changes are made and the host application container is redirected to Exago. Note, however, that if the API is being used but the host application does not redirect to Exago, the event can manually be called using the following public method:

```
Api.SetupData.FireOnConfigLoadEndEvent()
```

For more information, please see the **Global Event: OnConfigLoadEnd** article.

Can I only have a static configuration and generate the full dynamic configuration programmatically at session start?

Yes, the full dynamic configuration can be generated programmatically at the start of a session. This dynamic config will inherit the necessary components from the static configuration upon session start as long as the generated dynamic configuration's directive properly specifies its type and parent.

If in-session configuration storage is set to store the entire configuration, and not just diffs, does this enhancement have any effect?

Yes, this enhancement will still have an effect if in-session configuration storage is set to store the entire configuration file. Since the parent configuration will be loaded into static memory and made available to all user sessions, the only information that will need to be loaded at session start will be the portion of the configuration allotted to the dynamic configuration. This will decrease the required resources that need to be loaded at session start, thus still optimizing the system.

If configurations are programmatically generated in full at session start does this enhancement have any effect?

This enhancement was specifically designed to reduce the size of the configuration file that is required to be loaded into memory at the start of each session. Thus, if configurations are programmatically generated in full at session start, this enhancement would likely not have any effect since the size of the configuration file being loaded into memory has not been reduced.

Are there any breaking changes? Is this a suggested or a mandatory migration?

There are no breaking changes associated with this enhancement.

This migration is not mandatory. While this is a suggested migration for most clients due to its inherent performance optimizations, some may find that a migration to the new configuration architecture may not have an effect on their current system and therefore may decide against implementing it.

Modifying Configs in a Production Environment

test

Using the Administration Console and Exago API

> **NOTE**
>
> This version of the .NET API documentation is deprecated. A new version can be found at /support/administrators/net-api-docs/.

Introduction: Understanding the Relationship

The Exago Administration Console and API are used to control and configure a set of metadata that customizes a given user's session. Â That customization includes elements such as:

- **Data Security**: controlling access at the source, entity, row, column or custom level
- **Report Security**: controlling read/write and visibility of reports through folders
- **Culture Settings**: such as language, date and time format, currency symbol, etc.
- **Available Features**: toggle on/off designers and other features

Each available setting has an XML representation in the *Exago configuration schema,Â* which means that at any given point in time the current state of a user session can be represented as a set of XML values corresponding to the values as set by the administrator and/or developer.

Administration Console: The Base

The Administration Console (or Admin Console) then is used to create aÂ *base configuration. Â* Think of Admin Console settings as (relatively) static and applying to all users. Â When Admin Console data is saved it is stored both to XML as well as to a .enc file that contains an encrypted version of the same XML. Â During the standard launch sequence this XML (or .enc) is part of the initial load, giving the host application the base config as a starting point.

API: The Delta

At runtime the host application then makes a series of API calls to customize the user session. These customizations representÂ *the delta* from the original base config. Â When Exago is launched in this manner the effective user session is thus established as the combination of base and delta orÂ *base plus delta.*

Bringing it all Together

Generally speaking,Â the Administration Console is used for things that applyÂ *universally,* while the API is used for things that applyÂ *specifically* for an individual user. Â A few examples will help illustrate this point.

Example 1: Security at the *Data Source* Level

This example illustrates how to choose the user's data source (represented by a connection string) at runtime. Â For simplicity, we assume that the *entity-levelÂ* schema (tables, views, joins, etc.) is consistent across users; they just need access to different databases.

We have:

- User A, usingÂ Data Source *DS_A*
- User B, using Data SourceÂ *DS_B*

Use the **Administration ConsoleÂ** to:

- Create a singleÂ *Data Source*, using either the connection string forÂ *DS_A*Â or *DS_B*
- Build the metadata, normally using the Admin Discovery capability

At runtime assuming that User A is logged in, use something similar to the following in the API:

```
DataSource ds = api.DataSources.GetDataSource("dsName");
ds.DataConnStr = <Connection String forÂ DS_A>;
```

All of User A's queries will use the connection string set at runtime. Â However all of the metadata established in the Administration Console will be valid regardless of the connection string used in the Administration Console.

When user B needs to access Exago the same sequence is used, substituting the appropriate connection string of course. Â The key point is that different physical data sources can be represented by a single Exago data source, in essenceÂ *sharing*Â metadata.

Of course if the schemas among the data sources differ, the metadata itself can be modified at runtime via the API's Entities and Joins collections. Â When planning your integration consider setting common elements (such as data source type, in-common schema elements, etc.) in the Administration Console, and unique or distinguishing elements (such as connection string in this example) at runtime via the Exago API.

Example 2: *Row-LevelÂ* (AKAÂ *Multi-Tenant)Â* Security

This example assumes you've added a Data Source with one or more entities and wish to enforce row-level security on an entity. Â we assume for this example that the entity has a column called AccountId that is used for the filtering.

Use the **Administration ConsoleÂ** to:

- Create an ExagoÂ *Parameter*Â named "AccountId"
- Use the *Data Objects* tab to associate the AccountId Parameter with the AccountId field of the entities you want to secure

Use the **APIÂ** to:

- Set the AccountId parameter's value to the appropriate one for the currently logged-in user

Assuming that the desired AccountId is "39524" use something similar to the following:

Parameter p = api.Parameters.GetParameter("AccountId");

p.Value = "39524";

Based on the association established in the Admin Console, Exago will ensure that any queries against this entity will include AccountId = '39524' in the Where clause.

Config File XML & API Setting Reference (General Nodes)

When building configuration files without the **Admin Console**, or to change settings in the API, use this reference to map **Admin Console** settings to the relevant config file node or API key.

> **NOTE**
>
> For config file elements not in the <general> node, see **Config File XML Reference (All Nodes but General)**.

How to Use This Document

Setting Name : type

- <*config_file_node*> (within the <general> element)
- *Api_Key* (within the **Api.SetupData.General** namespace)

Example

Allow Filter Dependencies : bool

- <allowdependantfilters>
- AllowDependantFilters

This boolean value enables or disables the Filter Dependency feature. To enable it, set the setting to *True*. To disable it, set it to *False*. The sample configuration file below enables it.

```
<general>
 <allowdependantfilters>True</allowdependantfilters>
</general>
```

Enables Filter Dependency

Alternatively, this setting may be changed when creating a session with the .NET API:

```
Api api = new Api(ConfigurationManager.AppSettings["ExagoAppPath"]);
api.General.AllowDependantFilters = false;

api.Action = wrApiAction.Home;
sring paramString = api.GetUrlParamString(homePage);
```

Disables Filter Dependency

Or the REST Web Service API:

```
curl http://{webservice}/rest/settings?sid={sid} -XPATCH ^ -d "
{
 'General':
  {
    'AllowDependantFilters':false,
  }
}"
```

Disables Filter Dependency

> **TIP**
>
> This document is presented in the order that settings appear in the **Admin Console**. Use your browser's Find function (**Ctrl+F**) to find a specific setting in this document.

Main Settings

The **Main Settings** article includes detailed descriptions of each of these settings.

Temp Path : string

- <temppath>
- TempPath

Temp Cloud Service : string

- <tempcloudservice>
- TempCloudService

Language File : string

- <languagefile>
- LanguageFile

Temp URL : string

- <tempurl>
- TempUrl

Allow direct access to Exago (bypassing API) : bool

- <allowhomedirect>
- AllowHomeDirect

Allow Execution in Viewer : bool

- <allowhtmloutput>
- AllowHtmlOutput

Allowed Export Types – Excel : bool

- <allowexceloutput>
- AllowExcelOutput

Allowed Export Types – PDF : bool

- <allowpdfoutput>
- AllowPdfOutput

Allowed Export Types – RTF : bool

- <allowrtfoutput>
- AllowRtfOutput

Allowed Export Types – CSV : bool

- <allowcsvoutput>
- AllowCsvOutput

Default Output Type : constant

Possible values: *excel, pdf, rtf, csv*

- <defaultoutputtype>
- DefaultOutputType

Report Tree Shortcut : constant

Possible values: *Run*, *Export*

- <reporttreeshortcut>
- ReportTreeShortcutRun : bool

Active Role : string v2019.1+

- <activeroleid>
- ActiveRoleId

Culture Settings

The **Culture Settings** article includes detailed descriptions of each of these settings.

Date Format : string

- <dateformat>
- DateFormat

Time Format : string

- <timeformat>
- TimeFormat

DateTime Format : string

- <datetimeformat>
- DateTimeFormat

DateTime Values Treated As : constant

Possible values: *date, datetime*

- <datetimetreatedas>
- DateTimeTreatedAs

Numeric Separator Symbol : string

- <separatorsymbol>
- SeparatorSymbol

Numeric Currency Symbol : string

- <currencysymbol>
- CurrencySymbol

Numeric Decimal Symbol : string

- <decimalsymbol>
- DecimalSymbol

Numeric Decimal Places : int

- <decimalplaces>
- DecimalPlaces

Currency Decimal Places : int

- <currencydecimalplaces>
- CurrencyDecimalPlaces

Apply Numeric Decimal Places to General Cell Formatting : bool

- <applygeneralformatdecimalplaces>
- ApplyGeneralFormatDecimalPlaces

Apply General Currency Right Alignment : bool

- <applygeneralcurrencyrightalignment>
- ApplyGeneralCurrencyRightAlignment

Server Time Zone Offset : decimal

> **NOTE**
>
> Set to *0* to use the **Client Time Zone Name**. *If null, Exago will use the External Interface (a deprecated extensibility feature) to calculate the time zone change.*

- <servertimezoneoffset>
- ServerTimeZoneOffset

Time Zone Name : string

- <clienttimezonename>
- ClientTimeZoneName

Feature/UI Settings

The **Feature/UI Settings** article includes detailed descriptions of each of these settings.

Allow Creation/Editing of Express Reports : bool

> **NOTE**
>
> This setting was removed from the Admin Console in *v2019.2* but remains available here in the config file. No other functionality was removed.

- <showexpressreports>
- IsShowExpressReports

Allow Creation/Editing of Advanced Reports : bool

- <showadvancedreports>
- IsShowAdvancedReports

Allow Creation of Crosstab Reports : bool

- <showcrosstabreports>
- IsShowCrosstabReports

Allow Creation/Editing of Dashboards : bool

- <showdashboardreports>
- lsShowDashboardReports

Allow Creation/Editing of Chained Reports : bool

- <showchainedreports>
- lsShowChainedReports

Allow Creation/Editing of ExpressViews : bool

- <showexpressviews>
- lsShowExpressViews

Allow Editing ExpressView with Live Data : bool

- <allowexpressviewliveedit>
- lsAllowExpressViewLiveEdit

Show Data Fields Search Box : bool pre-v2017.2

- <showexpressviewdatafieldssearch>
- lsShowExpressViewDataFieldsSearch

Fields Enabled in Data Fields Tree : bool v2017.2+

- <showdatafieldssearch>
- lsShowDataFieldsSearch

Show Styling Toolbar : bool

> **NOTE**
>
> This setting was removed from the Admin Console in *v2019.2* but remains available here in the config file. No other functionality was removed.

- <showexpressreportsstylingtoolbar>
- lsShowExpressReportsStylingToolbar

Show Themes : bool

> **NOTE**
>
> This setting was removed from the Admin Console in *v2019.2* but remains available here in the config file. No other functionality was removed.

- <showexpressreportsthemes>
- lsShowExpressReportsThemes

Show Grouping : bool

> **NOTE**
>
> This setting was removed from the Admin Console in *v2019.2* but remains available here in the config file. No other functionality was removed.

- <showexpressreportsgrouping>
- lsShowExpressReportsGrouping

Show Formula Button : bool

> **NOTE**
>
> This setting was removed from the Admin Console in *v2019.2* but remains available here in the config file. No other functionality was removed.

- <showexpressreportsformulabutton>
- lsShowExpressReportsFormulaButton

Show CrossTab Wizard : bool

- <showcrosstabwizard>
- lsShowCrosstabWizard

Show Chart Wizard : bool

- <showchartwizard>
- IsShowChartWizard

Chart Colors : string

- <chartcolors>
- ChartColors

Maximum Number of Chart Data Points : int

- <maxnumberofchartpoints>
- MaxNumberOfChartPoints

Default Chart Font : string

- <defaultchartfont>
- DefaultChartFont

Show Geochart Map Wizard : bool

- <showmapwizard>
- IsShowMapWizard

Geochart Map Key (optional) : string

- <geochartmapkey>
- GeochartMapKey

Geochart Map Colors : string

- <mapcolors>
- MapColors

Show Google Map Wizard : bool

- <showgooglemapwizard>
- IsShowGoogleMapWizard

Google Map Key (unlimited or JS API restricted) : string v2018.1+

- <googlemapjsapikey>
- GoogleMapJSAPIKey

Google Map Key (optional Geocode API restricted) : string v2018.1+

- <googlemapgeocodeapikey>
- GoogleMapGeocodeAPIKey

Google Map Colors : string

- <googlemapcolors>
- GoogleMapColors

Show Gauge Wizard : bool

- <showgaugewizard>
- IsShowGaugeWizard

Gauge Colors : bool

- <gaugecolors>
- GaugeColors

Show Document Template : bool

- <showpdftemplate>
- IsShowPdfTemplate

Show Document Template Upload Button : bool

- <showtemplateuploadbutton>
- IsShowTemplateUploadButton

Show Linked Report : bool

- <showlinkreport>
- lsShowLinkReport

Show Linked Report Fields : bool

- <showlinkreportfields>
- lsShowLinkReportFields

Show Linked Report Formula : bool

- <showlinkreportformula>
- lsShowLinkReportFormula

Show Linked Action : bool

- <showlinkaction>
- lsShowLinkAction

Show Auto Sum Button : bool v2021.1+

- <showautosum>
- lsShowAutoSum

Show Insert Image : bool

- <showinsertimage>
- lsShowInsertImage

Show Joins Window : bool

- <showjoinswindow>
- lsShowJoinsWindow

Show Advanced Joins : bool

- <showadvancedjoins>
- lsShowAdvancedJoins

Advanced Joins Display : bool v2017.3.1+

- <showcomplexjoins>
- lsShowComplexJoins

Allow Category Aliasing : bool v2017.3.1+

- <showcategoryalias>
- lsShowCategoryAlias

Show Events Window : bool

- <showeventswindow>
- lsShowEventsWindow

Show Report-Level Parameters Window : bool

- <showreportparameterswindow>
- lsShowReportParametersWindow

Show SQL Window : bool

- <showsqlwindow>
- lsShowSQLWindow

Linked Report Display : constant v2017.3+

Possible values: *Cursor*, *NewTab*, *ScreenCenter*

- <linkedreportdisplay>
- LinkedReportDisplay

Allow Grouping on non-Sorts : bool

- <showgroupheadersformulabutton>
- lsShowGroupHeadersFormulaButton

Prompt user for Parameters/Filters on Execution : bool

- <dashboardpromptatexecution>

- IsDashboardPromptAtExecution

Automatically Refresh Reports v2021.1+

- <dashboardautomaticallyrefreshtiles>
- IsDashboardAutomaticallyRefreshTiles

Show URL Item Button : bool

- <dashboardshowurlitembutton>
- IsDashboardShowUrlItemButton

Allow Creation/Editing of Dashboard Visualizations : bool

- <showdashboardnewvisualizationbutton>
- IsShowDashboardNewVisualizationButton

Show Data Fields Search Box : bool

- <showdashboarddatafieldssearch>
- IsShowDashboardDataFieldsSearch

Use Sample Data for Dashboard Visualization Design : bool pre-v2019.2

> **NOTE**
>
> This setting was removed from the Admin Console in *v2019.2* but remains available here in the config file. No other functionality was removed.

- <usesampledatafordashboardvisualizationdesign>
- IsUseSampleDataForDashboardVisualizationDesign

Visualization Database Row Limit : int

- <visualizationdbrowlimit>
- VisualizationDbRowLimit

Refresh Reports/Visualizations on Dashboards Silently : bool

- <silentdashboardrefresh>
- IsSilentDashboardRefresh

Minimum Tile Width for Dashboard Reflow : int v2019.1+

- <mintilewidth>
- MinTileWidth

Minimum Window Width for Dashboard Reflow : int v2019.1+

- <mindesktopwidth>
- MinDesktopWidth

Default Designer Font : string

- <defaultfont>
- DefaultFont

Default Designer Font Size : int

- <defaultfontsize>
- DefaultFontSize

Show Help Button : bool

- <showhelp>
- IsShowHelp

Custom Help Source : string

- <customhelpsource>
- CustomHelpSource

Show Exports in Tab : bool

- <showexportsintab>
- IsShowExportsInTab

Show IE Download Button : bool

- <showiedownloadbutton>
- IsShowIeDownloadButton

Show Join Fields : bool

- <showjoinfields>
- IsShowJoinFields

Show Grid Lines in Report Viewer : bool

- <showgrid>
- IsShowGrid

Show Enhanced Tooltips : bool

- <enhancedtooltips>
- IsShowEnhancedTooltips

Save on Report Execution : bool

- <saveonexecute>
- IsSaveOnExecute

Save on Finish Press : bool

- <saveonfinish>
- IsSaveOnFinish

Enable Right-Click Menus : bool

- <enablerightclickmenus>
- IsEnableRightClickMenus

Enable Reports Tree Drag And Drop : bool

- <enablereportstreedraganddrop>
- IsEnableReportsTreeDragAndDrop

Show Report Upload/Download Options : bool

- <showreportuploaddownloadoptions>
- IsShowReportUploadDownloadOptions

Allow interactivity in report viewer : bool

- <interactivehtml>
- AllowInteractiveHtml

Show Toolbar in Report Viewer : constant

Possible values: *Auto, Show, Hide*

- <showhtmltoolbar>
- IsShowHtmlToolbar

Default interactive report viewer dock is open : bool

- <defaultisdockopen>
- DefaultIsDockOpen

Interactive report viewer default dock placement : wrDockLocation

Possible values: *Left, Right*

- <dockplacement>
- DockPlacement

Allow save to report design from the report viewer : bool

- <executesavetodesign>
- AllowExecuteSaveToDesign

Maximum number of fields in a CrossTab header or tabulation source : int

- <crosstabmaximumfields>
- CrossTabMaximumFields

Use SVG for Application Icons : bool

- <usesvgforappicons>
- UseSVGForAppIcons

Application Theme Selection : string

- <csstheme>
- CssTheme

Join Path Algorithm : const v2018.1+

- <expressviewjoinalgorithm>
- ExpressviewJoinAlgorithm

Allow Creation of Custom SQL Objects : bool v2018.1+

- <allowreportcustomsqlobjects>
- IsAllowReportCustomSQLObjects

Data Sources to Exclude from Custom SQL Object Creation : string v2018.1+

- <excludedatasourcesreportcustomsql>
- ExcludeDataSourcesReportCustomSQL

Programmable Object Settings

The **Programmable Object Settings** article includes detailed descriptions of each of these settings.

Call Type Parameter Name : string

- <calltypeparamname>
- CallTypeParamName

Column Parameter Name : string

- <columnparamname>
- ColumnParamName

Filter Parameter Name : string

- <filterparamname>
- FilterParamName

Full Filter Parameter Name : string

- <fullfilterparamname>
- FullFilterParamName

Sort Parameter Name : string

- <sortparamname>
- SortParamName

Data Category Parameter Name : string

- <datacategoryparamname>
- DataCategoryParamName

Data Object ID Parameter Name : string

- <objectidparamname>
- ObjectIdParamName

DB Row Limit Parameter Name : string v2018.1+

- <dbrowlimitparamname>
- DbRowLimitParamName

DB Row Start Index Parameter Name : string v2018.1+

- <dbrowrangestartparamname>
- DbRowRangeStartParamName

DB Row End Index Parameter Name : string v2018.1+

- <dbrowrangeendparamname>

- DbRowRangeEndParamName

Filter Settings

The **Filter Settings** article includes detailed descriptions of each of these settings.

Show Group (Min/Max) Filters : bool

- <showgroupfilters>
- IsShowGroupFilters

Show Top N Filters : bool

- <showtopn>
- IsShowTopN

Allow New Filters at Execution : bool

- <allowexecutenewfilters>
- AllowExecuteNewFilters

Read Database for Filter Values : bool

- <readfiltervalues>
- IsReadFilterValues

Allow Filter Dependencies : bool

- <allowdependantfilters>
- AllowDependantFilters

Show Filter Description : bool

- <showfilterdescription>
- IsShowFilterDescription

Default Filter Execution Window : constant

Possible values: *Standard, SimpleWithOperator, Simple, Custom*

- <defaultFilterExecutionWindow>
- DefaultFilterExecutionWindow

Allow User to Change Filter Window : bool

- <changeableFilterExecutionWindow>
- IsChangeableFilterExecutionWindow

Include Null Values for 'NOT' Filters : bool

- <includenotfilternullvalues>
- IsIncludeNotFilterNullValues

Custom Filter Execution Window : string

- <filterexecutionwindow>
- FilterExecutionWindow

Database Settings

See **Database Settings** for more information.

Database Timeout : int

- <dbtimeout>
- DbTimeout

Database Row Limit : int

- <dbrowlimit>
- DbRowLimit

Row Limit Step Size : int v2017.2+

- <dbrowlimitstepsize>
- DbRowLimitStepSize

Disable Non-Joined Data Objects : bool

- <detectjoinedobjects>
- IsDetectJoinedObjects

Enable Special Cartesian Processing : bool

- <enablespecialcartesianprocessing>
- IsEnableSpecialCartesianProcessing

Aggregate and Group in Database : bool

- <aggregateandgroupindatabase>
- IsAggregateAndGroupInDatabase

Convert Formulas to SQL v2020.1+

- <evaluateformulasindatabase>
- EvaluateFormulasInDatabase

Scheduler Settings

The **Scheduler Settings** article includes detailed descriptions of each of these settings.

Enable Report Scheduling : bool

- <enablescheduling>
- IsEnableScheduling

Show Report Scheduling Option : bool

- <showschedulereports>
- IsShowScheduleReports

Show Email Report Options : bool

- <showschedulereportsemail>
- IsShowScheduleReportsEmail

Show Schedule Reports Manager : bool

- <showschedulereportsmanager>
- IsShowScheduleReportsManager

Show Schedule No End Date Option : bool

- <showschedulenoenddate>
- IsShowScheduleNoEndDate

Show Schedule Intraday Recurrence Option : bool

- <showscheduleintradayrecurrence>
- IsShowScheduleIntradayRecurrence

Scheduler Manager User View Level : constant

Possible values: *User, Company, All*

- <schedulemanagerviewlevel>
- ScheduleManagerViewLevel

Email Scheduled Reports : bool

- <emailscheduledreports>
- IsEmailScheduledReports

Enable Batch Reports : bool

- <enablebatchreports>
- IsEnableBatchReports

Show Schedule Delivery Type Options : bool

- <showscheduledeliverytypeoptions>
- IsShowScheduleDeliveryTypeOptions

Use Secure Scheduler Remoting Channel : bool

- <usesecureremotingchannel>
- IsUseSecureRemotingChannel

Schedule Remoting Host : string

- <scheduleremotinghost>
- ScheduleRemotingHost

Enable Remote Report Execution : bool

- <executeremotely>
- IsExecuteRemotely

Enable Execution Cache : bool

- <executioncache>
- UseExecutionCache

User Cache Visibility Level : constant v2017.3+

Possible values: *Global*, *Company*, *User*

- <cachevisibilitylevel>
- CacheVisibilityLevel

Enable Access to Data Sources Remotely : bool

- <accessdatasourcesremotely>
- IsAccessDataSourcesRemotely

Remote Execution Remoting Host : string

- <synchronousremotinghost>
- SynchronousRemotingHost

Custom Queue Service : string

- <schedulerqueueservice>
- SchedulerQueueService

Delete Schedules upon Report Deletion : bool

- <deleteschedulesuponreportdeletion>
- IsDeleteSchedulesUponReportDeletion

Default Email Subject : string

- <reportscheduleemailsubject>
- ReportScheduleEmailSubject

Default Email Body : string

- <reportscheduleemailbody>
- ReportScheduleEmailBody

Password Requirements (for pdf and excel documents) : string

- <passwordrequirement>
- PasswordRequirement

Custom Scheduler Recipient Window : string

- <schedulerrecipientwindow>
- SchedulerRecipientWindow

User Settings

The **User Settings** article includes detailed descriptions of each of these settings.

User Preference Storage Method : constant

Possible values: one of the wrUserPreferenceStorage constants.

- <userprefstorage>
- UserPreferenceStorage

Startup Report(s) Replace Getting Started : bool

- <userstartupreportoverwritegettingstarted>
- UserStartupReportOverwriteGettingStarted

Maximum Number of Startup Reports : int

- <userstartupreportmaxnum>
- UserStartupReportMaxNum

Allow User Reports : bool

- <userallowuserreports>
- UserAllowUserReports

Other Settings

The **Other Settings** article includes detailed descriptions of each of these settings.

Excel Export Target : constant

Possible values: *v2003, v2007, v2010, v2013, v2016. v2013* is the default value.

- <excelexporttarget>
- ExcelExportTarget

External Interface : string

- <externalinterface>
- ExternalInterface

Enable Paging in Report Viewer : bool

- <htmlpaging>
- IsHtmlPaging

Renew Session Automatically : bool

- <renewsession>
- IsRenewSession

Write Log File : constant v2017.2+

Possible values: *NONE, ERROR, INFO, DEBUG*

- <logginglevel>
- LoggingLevel

Enable Debugging : bool

- <debugging>
- IsDebugging

Max Report Execution Time (minutes) : int

- <maxjobexecutionminutes>
- MaxJobExecutionMinutes

Max Chained Report Collation Executions : int v2021.1.8+

- <chainedreportmaxcollationexecutions>
- ChainedReportMaxCollationExecutions

Maximum Age for Temp Files : int

- <maxtempfileage>
- MaxTempFileAge

Enable Web Service / Assembly Data Mapping : bool

- <datamapping>
- IsDataMapping

Limit Reports and Visualizations to One Object : bool v2021.1+

- <onecategorylimit>
- IsOneCategoryLimit

Run Aggregate Functions at Record Level by Default : bool

- <isaggregaterecordlevel>
- IsAggregateRecordLevel

Cache External Services : bool

- <cacheservices>
- IsCacheServices

Global Schema Access Type : constant

Possible values: *Datasource, Metadata*

- <schemaaccesstype>
- SchemaAccessType

Allow Multiple Sessions : bool

- <allowmultiplesessions>
- AllowMultipleSessions

Allow MD5 Hashing on FIPS server : bool

- <allowmd5onfips>
- AllowMD5OnFIPS

Write BOM to CSV Files v2019.1.1+

- <writebomtocsv>
- WriteBOMToCSV

'LoadImage' Cell Function Parameter Prefix : string

- <loadimageprefix>
- LoadImageFuncParamPrefix

Ignore Inaccessible Report Folders : bool

- <ignoreinaccessiblereportfolders>
- IgnoreInaccessibleReportFolders

User ID : string

- <userid>
- UserId

Password : string

- <password>
- Password

Debug Password : string

- <debugextractionpassword>
- DebugExtractionPassword

REST Key : string v2017.3+

- <restkey>
- RestKey

Exago Expiration Date : string

- <expirationdate>
- ExpirationDate

Custom Code Supplied by Exago : bool

- <customcode>
- CustomCode

License Key : string

- <licensekey>
- LicenseKey

Hidden Flags

Some settings are not accessible in the **Admin Console** and are only available by editing the Config File. These so called hidden flags are documented in theÂ **Hidden Flags** article.

Unsupported

These settings are not accessible in the **Admin Console**. Changing these settings is not supported.

Formula functions loaded : bool v2017.2+

- <formulafunctionsloaded>
- AreFormulaFunctionsLoaded

Date functions loaded : bool

- <datefunctionsloaded>
- AreDateFunctionsLoaded

Version Number : string

- <versionnumber>
- VersionNumber

Data Cloud Service : string

- <datacloudservice>
- DataCloudService

Window Height : int

- <windowheight>
- WindowHeight

Window Width : int

- <windowwidth>
- WindowWidth

Map Terms Signature : string

- <maptermssignature>
- MapTermsSignature

Join Transform Objects in Database: bool v2018.1+

- <canjointransformobjectsindb>
- CanJoinTransformObjectsInDb

Config File XML Reference (All Nodes but General)

This reference guide describes the structure of the config file XML elements. Use this guide if you build or edit your config XML directly, without using the **Admin Console**.

> **TIP**
>
> For the settings nodes within <general></general>, see Config File XML & API Setting Reference (General Nodes).

The config file's outer nodes are as follows:

```
<?xml version="1.0" standalone="yes"?>
<webreports></webreports>
```

All following nodes are within the <webreports> element. They are organized below in alphabetical order, not necessarily the order they appear in the configuration file.

How to Use This Document

XML objects are represented in tables with the following format:

Node	Description	Type	Required

Node is the XML node name. Nodes in **bold font** are objects, which are represented below the table.

Description is the name of the field as it appears in the **Admin Console**. If there is no mapped field, or the field is implicit, the description is in *italic font.*

Type is the value type. If the type is a constant or enum, the possible values are listed below the table. Constants are case sensitive.

Required indicates whether the node requires a value. (1+) indicates that there can be multiple nodes.

Action Events

The **Introduction to Action Events** article includes detailed descriptions of each of these settings.

Outer node: <actionevent></actionevent>

Node	Description	Type	Required
name	Name	string	yes
id	*Unique Id*	int	yes
event_type	Global Event Type	constant	yes
datasource_id	*Id of assembly data source*	int	no
func_name	*Assembly function name*	string	no
user_action	Event Type	constant	yes
language	Language	constant	no
program_code	*Program code*	string	no
uiaction	Assigned UI Item(s)	object	no (1+)
reference	References	object	no (1+)
namespace	Namespaces	object	no (1+)

Possible values for **event_type**: *None*, or see **Global Action Events**.

Possible values for **user_action**: *None, Load, Click*

Possible values for **language**: *CSharp*, *JavaScript, VB*

uiaction

Node	Description	Type	Required
element_id	*Element Id*	constant	yes

Possible values for **element_id**: See **Identifying Actionable UI Elements**.

reference

Node	Description	Type	Required
name	*Reference name*	string	yes

namespace

Node	Description	Type	Required
name	*Namespace name*	string	yes

Example

```
<actionevent>
 <name>DoubleClick</name>
 <id>0</id>
 <event_type>OnDoubleClickReport</event_type>
 <user_action>Click</user_action>
 <language>CSharp</language>
 <program_code>return Extensions.ActionEvents.DoubleClick(sessionInfo);</program_code>
 <uiaction>
  <element_id>RunReportBtn</element_id>
 </uiaction>
 <reference>
  <name>Extensions.dll</name>
 </reference>
</actionevent>
```

Custom Options

The **Custom Options** article includes detailed descriptions of each of these settings.

Outer node: <customoption></customoption>

Node	Description	Type	Required
option_id	Id	string	yes
option_type	Type	constant	yes
option_list_item	*Option list item*	object	no (1+)

Possible values for **option_type**: *Int, Decimal*, *Bool*, *Text, List*

option_list_item

Node	Description	Type	Required

option_list_item_id	*List item Id*	string	yes

Example

```
<customoption>
 <option_id>Hello</option_id>
 <option_type>List</option_type>
 <option_list_item>
  <option_list_item_id>World</option_list_item>
 </option_list_item>
</customoption>
```

Data Sources

The **Data Sources** article includes detailed descriptions of each of these settings.

Outer node: <datasource></datasource>

Node	Description	Type	Required
id	*Unique Id*	int	yes
name	Name	string	yes
dbtype	Type	constant	yes
dataconnstr	Connection String	string	yes
schema	Schema/Owner Name	string	no
odbcdelim	Column Delimiter(s)	string	no

Possible values for **dbtype**: *mssql, oracle, msolap, odbc, websvc, assembly, mysql, file, postgres, db2, informix, cdata*

Example

```
<datasource>
 <id>0</id>
 <name>Northwind</name>
 <dbtype>odbc</dbtype>
 <schema>dbo</dbtype>
 <dataconnstr>DRIVER=SQLite3 ODBC Driver;Database=C:Northwind.sqlite;</dataconnstr>
 <odbcdelim>`</odbcdelim>
</datasource>
```

Data Objects

The **Data Objects** article includes detailed descriptions of each of these settings.

When a Data Object is a clone of another only the id, entity_name, cloned_from, inherit_category and inherit_description, category, object_description and db_name apply.

Outer node: <entity></entity>

Node	Description	Type	Required
id	Id	string	no
entity_name	Alias	string	no
cloned_from	v2019.1+ *<id> of the parent data object that this one is a clone of.*	string	no
inherit_category	v2019.2.18+ Inherit Folder *For cloned data objects, true if the category/folder the object appears in should be copied from its parent. False if the clone should have its own category folder. If false, <category> must be provided.*	boolean	no
inherit_description	v2019.2.18+ Inherit Description *For cloned data objects, true if the description of the clone should be copied from its parent. False if the clone should have its own description. If false, <object_description> must be provided.*	boolean	no
db_name	Name *For cloned data objects, a value here overrides the object in the Data Source (i.e. table, view) this clone points to while continuing to use the same metadata as its cloned_from object.*	string	yes
sql_stmt	Custom SQL Object	string	no
object_description	Description	string	no
category	Category	string	no

datasource_id	*Id of data source*	int	yes
object_type	*Object type*	constant	yes
schema_access_type	Schema Access Type	constant	no
key	Unique Key Fields	object	yes (1+)
param	Parameters	object	no (1+)
tenant	Tenant Columns	object	no (1+)
filter_dropdown	Filter Dropdown Object	object	no
column_metadata	Column Metadata	object	no (1+)
transform	*Vertical table transform*	object	no
affinity	v2018.1+ Report or config affinity	constant	no
suppress_sort_filter	v2018.1+ Suppress Sort and Filter	boolean	no
canreexecuteindb	v2021.1 Interactive Filtering in Database	boolean	no

Possible values for **object_type**: *table, view, procedure, function, assembly, sqlstmt, websvc, file*

Possible values for **schema_access_type**: *Metadata, Datasource*

Possible values for **affinity**: *Global, Report*

key

Node	Description	Type	Required
col_name	*Column name*	string	yes

param

Node	Description	Type	Required
id	*Parameter name*	string	yes

tenant

Node	Description	Type	Required
col_name	*Column name*	string	yes
parameter_id	*Parameter name*	string	yes

filter_dropdown

Node	Description	Type	Required
db_name	*Data object name*	string	yes
sql_stmt	*SQL statement*	string	no
datasource_id	*Id of data source*	int	yes
object_type	*Object type*	constant	yes

Possible values for **object_type**: *table, view, procedure, function, assembly, sqlstmt, websvc, file*

column_metadata

Node	Description	Type	Required
col_source	*Specify as custom col*	constant	no
col_name	*Column name*	string	yes
col_description	Column Description	string	no
col_type	Data Type	constant	no
col_alias	Column Alias	string	no
visible	Visible	bool	no
filterable	Filterable	constant	no
sortable	Sortable	bool	no
col_sortandgroupbyvalue	Sort and Group-By Value	string	no
col_value	Column Value	string	no

Possible values for **col_source**: *ExagoFormula*

Possible values for **col_type**: *string, date, datetime, time, int, decimal, image, float, boolean, guid, currency*

Possible values for **filterable**: *All, Dynamic, Static, None*

transform

Node	Description	Type	Required
col_name	*Data field names column*	string	yes
val_name	*Data values column*	string	yes
datatype_name	*Data types column*	string	no
non_transform_col	*Fields exempt from transform*	object	no

non_transform_col

Node	Description	Type	Required
col_name	*Data field name*	string	yes
data_type	*Data type*	enum	no

Possible values for **data_type**: 0: string, 1: date, 2: datetime, 3: time, 4: integer, 5: decimal, 6: float, 7: bit, 8: guid, 9: image, 10: currency

Example

```
<entity>
 <id>ODE</id>
 <entity_name>OrderDetails</entity_name>
 <db_name>OrderDetail</db_name>
 <sql_stmt>SELECT *</sql_stmt>
 <category>Order Information</category>
 <datasource_id>1</datasource_id>
 <object_type>table</object_type>
 <key>
  <col_name>OrderId</col_name>
 </key>
 <key>
  <col_name>ProductId</col_name>
 </key>
 <param>
  <id>count</id>
 </param>
 <tenant>
  <col_name>CategoryName</col_name>
  <parameter_id>count</parameter_id>
 </tenant>
 <filter_dropdown>
  <db_name>Category</db_name>
  <sql_stmt>SELECT *</sql_stmt>
  <datasource_id>-1</datasource_id>
  <object_type>table</object_type>
 </filter_dropdown>
 <column_metadata>
  <col_source>ExagoFormula</col_source>
  <col_name>Total</col_name>
  <col_type>currency</col_type>
  <col_alias>Total</col_alias>
  <filterable>false</filterable>
  <col_value>{ODE.Quantity}*({ODE.UnitPrice}-{ODE.Discount})</col_value>
 </column_metadata>
 <column_metadata>
  <col_name>UnitPrice</col_name>
  <col_type>currency</col_type>
 </column_metadata>
</entity>
```

Functions

The following articles include detailed descriptions of each of these settings

- Custom Functions
- Custom Filter Functions
- Custom Aggregate Functions

Outer node: <function></function>

> **NOTE**
>
> Functions and Filter Functions use the same XML syntax. Filter functions have weight > 0 (which indicates its order in the filter list).

Node	Description	Type	Required
name	Name	string	yes
description	Description	string	no
category	Category	constant	no
arguments_json	v2017.2+ *Arguments information*	***json*** object	yes
variable_arguments	v2017.2+ Variable Argument Count	bool	yes
available_in	*Formula*, *filter* or *custom aggregate*	enum	yes
filter_return_type	Filter Type	constant	no
weight	List Order	int	yes
language	Language	constant	yes
program_code	*Program Code*	string	yes
disable_database_grouping	v2019.1.5+ Prevent In-Database Grouping When Included in Detail Section	bool	yes
reference	References	object	no (1+)
namespace	Namespaces	object	no (1+)

Possible values for **category**: *Aggregate, Operators, Logical, Date, Financial, Database, Arithmetic, String, Formatting, Other*

Possible values for **available_in**:

0. Unavailable
1. Formula
2. Filter
3. Aggregate

Possible values for **filter_return_type**: *date, string, integer, decimal*

Possible values for **language**: *CSharp, JavaScript, VB*

reference

Node	Description	Type	Required
name	*Reference name*	string	yes

namespace

Node	Description	Type	Required
name	*Namespace name*	string	yes

arguments_json v2017.2+

A ***JSON*-formatted *array*** of objects with the following format:

Node	Description	Type	Required
Name	Name	string	yes
Required	Required	bool	yes
Description	Description	string	yes

Example

```
<function>
 <name>IncrementBy</name>
 <description>Count up by value steps.</description>
 <category>Other</category>
 <arguments_json>[{"Name":"counter","Required":false,"Description":"Number to increment by. Default: 1."}]</arguments_json>
 <variable_arguments>false</variable_arguments>
 <available_in>1</available_in>
 <filter_return_type>String</filter_return_type>
 <weight>0</weight>
 <language>CSharp</language>
 <program_code>return Extensions.CustomFunctions.IncrementBy(sessionInfo, args);</program_code>
 <reference>
  <name>Extensions.dll</name>
 </reference>
</function>
```

Joins

The **Join Configuration** article includes detailed descriptions of each of these settings.

Outer node: <join></join>

NOTE

For entity names, use Id if set, otherwise use db_name.

Node	Description	Type	Required
entity_from_name	*From object (left)*	string	yes
entity_to_name	*To object (right)*	string	yes
join_type	Join Type	constant	yes
relation_type	Relation Type	constant	yes
weight	Weight	int	yes
clause	v2017.2+ Join Clause	object	yes (1+)
affinity	v2018.1+ *Join affinity*	constant	yes

Possible values for **join_type**: *inner, leftouter, rightouter, fullouter*

Possible values for **relation_type**: 11: one to one, 1M: one to many

Possible values for **affinity**: Global, Report

clause v2017.2+

Node	Description	Type	Required
left_entity	*From object (left) (v2017.3+)*	string	no
left_side	*From expression (left)*	string	yes
right_side	*To expression (right)*	string	yes
conjunction	*Joining conjunction*	constant	yes
comparison	*Comparison operator*	constant	yes
left_side_type	*From expression type*	constant	yes
right_side_type	*To expression type*	constant	yes
level	*Parenthesis nesting level*	int	yes

left_entity overrides the base (implicit) *from object* **entity_from_name** for the specific join clause; this is necessary for some uses of the *IN* operator

Possible values for **conjunction**: *AND*, *OR*

Possible values for **comparison**: *EQ* (=), *NE* (<>), *LT* (<), *GT* (>), *LE* (<=), *GE* (>=), *IN*

Possible values for **left_side_type**, **right_side_type**: *Column*, *Constant*, *SubQuery*, *Expression*

Example

```
<join>
 <entity_from_name>PRO</entity_from_name>
 <entity_to_name>CAT</entity_to_name>
 <clause>
  <left_side>ProductId</left_side>
  <right_side>CategoryId</right_side>
  <conjunction>AND</conjunction>
  <comparison>EQ</comparison>
  <right_side_type>Column</right_side_type>
  <left_side_type>Column</left_side_type>
  <level>0</level>
 </clause>
</join>
```

Parameters

The **Parameters** article includes detailed descriptions of each of these settings.

Outer node: <parameter></parameter>

Node	Description	Type	Required
id	Name	string	yes
data_type	Type	constant	yes
value	Value	string	no
hidden	Hidden	bool	yes
prompt_text	Prompt Text	string	no

parameter_dropdown	Parameter Dropdown Object	object	no

Possible values for **data_type**: *string, date, integer, decimal, boolean*

parameter_dropdown

Node	Description	Type	Required
db_name	*Database object name*	string	yes
sql_stmt	*SQL statement*	string	no
datasource_id	*Unique Id of data source*	int	yes
object_type	*Object type*	constant	yes
value_field	Value Field	string	yes
display_value_field	Display Value Field	string	no
display_data_type	Display Type	constant	yes
sp_params	*Stored proc parameters*	string	no

Possible values for **object_type**: *table, view, procedure, function, assembly, sqlstmt, websvc, file*

Possible values for **display_data_type**: *string, date, integer, decimal*

Example

```
<parameter>
 <id>count</id>
 <data_type>integer</data_type>
 <hidden>False</hidden>
 <prompt_text />
 <parameter_dropdown>
  <db_name>Category</db_name>
  <sql_stmt />
  <datasource_id>1</datasource_id>
  <object_type>view</object_type>
  <value_field>Category_View</value_field>
  <display_value_field />
  <display_data_type>string</display_data_type>
  <sp_params />
 </parameter_dropdown>
</parameter>
```

Quick Functions v2021.1+

Quick Functions are pre-defined calculations or operators that transform the fields in either detail fields or group columns in ExpressView. Learn more in the Quick Functions section of the ExpressView: Formula Columns article.

Outer node: <quickfunction></quickfunction>

Node	Description	Type	Required
id	A unique identifier used internally by the application	string	yes
language_id	The element ID in the language file that will be displayed in the Designer for this Quick Function's name For more information, review the Multi-Language Support article.	string	yes
display_formula	An Exago formula, preceded with the equals sign, that will be applied to the output displayed to the user Use the placeholder {cell_text} to include the contents of the cell in the formula Exago functions, custom functions and parameters may be used in the formula. Do not use aggregate functions or data object fields. Special characters must be XML encoded (e.g. use & for the concatenation character &)	string	yes
	An Exago formula, preceded with the equals sign, that will be applied to the data to order it when the column is sorted or when it is a group		

sort_text	Use the placeholder {cell_text} to include the contents of the cell in the formula Exago functions, custom functions and parameters may be used in the formula. Do not use aggregate functions or data object fields. Special characters must be XML encoded (e.g. use & for the concatenation character &)	string	yes
supported_data_types	The column data types that the Quick Function may be applied to Separate multiple values with this delimiter: |*{*}*|	string from QuickFunctionDataType	yes
return_data_type	The data type that this Quick Function returns A column will be transformed to this data type if different than the column's existing type	int from Constants.DataType	yes

Examples

year

The **year** Quick Function uses the Year() function to transform a Date or DateTime column into an integer value of the year portion of the date.

WebReports.xml

```
<quickfunction>
  <id>year</id>
  <language_id>Year_wrQuickFunction</language_id>
  <display_formula>=Year({cell_text})</display_formula>
  <sort_text>=Year({cell_text})</sort_text>
  <return_data_type>2</return_data_type>
  <supported_data_types>Date</supported_data_types>
</quickfunction>
```

en-us.xml

Below is an except from the English language file where the name of this Quick Function as it appears in the Designer is defined.

```
<DateFunctions>
  ...
 <element id="Year_wrQuickFunction">Year</element>
  ...
</DateFunctions>
```

month-name

The **month-name** Quick Function uses the MonthName() function to transform a Date or DateTime column into a string value representing the name of the month portion of the date (e.g. *January*, *February*, *March*, etc…). Note the use of the Month() function in the <sort_text> node to keep the list ordered chronologically instead of alphabetically.

WebReports.xml

```
<quickfunction>
  <id>month-name</id>
  <language_id>MonthName_wrQuickFunction</language_id>
  <display_formula>=MonthName({cell_text})</display_formula>
  <sort_text>=Month({cell_text})</sort_text>
  <return_data_type>0</return_data_type>
  <supported_data_types>Date</supported_data_types>
</quickfunction>
```

en-us.xml

Below is an except from the English language file where the name of this Quick Function as it appears in the Designer is defined.

```
<DateFunctions>
  ...
 <element id="MonthName_wrQuickFunction">Month Name</element>
  ...
</DateFunctions>
```

Roles

The **Roles** article includes detailed descriptions of each of these settings.

Outer node: <role></role>

Node	Description	Type	Required
id	ID	string	yes
active	Active	bool	yes
rolegeneral	*General settings*	object	yes
security	*Security settings*	object	yes

rolegeneral

> **NOTE**
>
> Settings pertaining to Express Reports were removed from the Admin Console in *v2019.2* but they remain available here in the config file. No other functionality was removed.

Node	Description	Type	Required
reportpath	Report Path	string	no
dateformat	Date Format	string	no
timeformat	Time Format	string	no
datetimeformat	DateTime Format	string	no
separatorsymbol	Numeric Separator Symbol	string	no
currencysymbol	Numeric Currency Symbol	string	no
readfiltervalues	Read Database for Filter Values	bool	no
dbtimeout	Database Timeout	int	yes
showgrid	Show Grid Lines in Report Viewer	bool	no
showcrosstabreports	Allow Creation/Editing of Crosstab Reports	bool	no
showdashboardreports	Allow Creation/Editing of Dashboard Reports	bool	no
showchainedreports	Allow Creation/Editing of Chained Reports	bool	no
showexpressviews	Allow Creation/Editing of ExpressViews	bool	no
allowexpressviewliveedit	Allow Editing ExpressView with Live Data	bool	no
showexpressreports	Allow Creation/Editing of Express Reports	bool	no
showexpressreportsgrouping	Show Grouping	bool	no
showexpressreportsformulabutton	Show Formula Button	bool	no
showexpressreportsstylingtoolbar	Show Styling Toolbar	bool	no
showexpressreportsthemes	Show Themes	bool	no
showadvancedreports	Allow Creation/Editing of Advanced Reports	bool	no
showschedulereports	Show Report Scheduling Option	bool	no
showschedulereportsmanager	Show Schedule Reports Manager	bool	no
schedulemanagerviewlevel	Scheduler Manager User View Level	constant	no
showschedulereportsemail	Show Email Report Options	bool	no
decimalsymbol	Numeric Decimal Symbol	string	no
languagefile	*Language File*	string	no
servertimezoneoffset	Server Time Zone Offset	decimal **NOTE** Set to *0* to use the **ClientTimeZoneName**. *If null, Exago will use the External Interface (a deprecated extensibility feature) to calculate the time zone change.*	no

showdashboardnewvisualizationbutton	Allow Creation/Editing of Dashboard Visualizations	bool	no
allowreportcustomsqlobjects	Allow Creation of Custom SQL Objects in Advanced Reports (*v2018.1+*)	bool	no

Possible values for **schedulemanagerviewlevel**: *user, company, all*

security

Node	Description	Type	Required
folders	Folder Security	object	yes
dataobjects	Data Object Security	object	yes
dataobjectrows	Data Row Security	object	no
includereportcustomsqlobjects	v2018.1+ Allow User to View Report-Level Custom SQL Objects	bool	no

folders

Node	Description	Type	Required
include_all	Include All Folders	bool	yes
read_only	All Folders Read Only	bool	yes
allow_management	Allow Folder Management	bool	yes
folder	Folders	object	no (1+)

folder

Node	Description	Type	Required
name	Folder Name	string	yes
read_only	Read Only	bool	yes
propagate	*Propagate status to sub-folders*	bool	yes

dataobjects

Node	Description	Type	Required
include_all	Include All Data Objects	bool	yes
dataobject	Objects	object	no (1+)

dataobject

> **NOTE**
>
> For name, use Id if set, otherwise use db_name.

Node	Description	Type	Required
name	Data Object Name	string	yes

dataobjectrows

Node	Description	Type	Required
dataobjectrow	Filters	object	no (1+)

dataobjectrow

> **NOTE**
>
> For name, use Id if set, otherwise use db_name.

Node	Description	Type	Required
name	Data Object Name	string	yes
filter	Filter String	string	yes

Example

```
<role>
 <id>Restricted</id>
 <active>False</active>
 <rolegeneral>
  <reportpath />
  <dateformat />
  <timeformat />
  <datetimeformat />
  <separatorsymbol />
  <currencysymbol />
  <readfiltervalues />
  <dbtimeout>0</dbtimeout>
  <showgrid />
  <showcrosstabreports />
  <showdashboardreports />
  <showchainedreports />
  <showexpressviews />
  <allowexpressviewliveedit />
  <showexpressreports />
  <showexpressreportsgrouping />
  <showexpressreportsformulabutton />
  <showexpressreportsstylingtoolbar />
  <showexpressreportsthemes />
  <showadvancedreports />
  <showschedulereports />
  <showschedulereportsmanager />
  <schedulemanagerviewlevel />
  <cachevisibilitylevel />
  <showschedulereportsemail />
  <decimalsymbol />
  <languagefile />
  <servertimezoneoffset>0</servertimezoneoffset>
  <clienttimezonename />
  <showdashboardnewvisualizationbutton />
  <allowreportcustomsqlobjects />
 </rolegeneral>
 <security>
  <includereportcustomsqlobjects>False</includereportcustomsqlobjects>
  <folders>
   <include_all>False</include_all>
   <read_only>False</read_only>
   <allow_management>True</allow_management>
   <folder>
    <name>Sales Reports</name>
    <read_only>False</read_only>
    <propagate>True</propagate>
   </folder>
  </folders>
  <dataobjects>
   <include_all>True</include_all>
  </dataobjects>
  <dataobjectrows />
 </security>
</role>
```

Server Events

T

Server Events

The
Introduction to Server Events
article includes detailed descriptions of each of these settings.

Outer node:
<serverevent></serverevent>

Node	Description	Type	Required
name	Name	string	yes
id	*Unique Id*	int	yes
event_type	Global Event	constant	yes
datasource_id	*Id of assembly data source*	int	no
func_name	*Assembly function name*	string	no

language	Language	constant	no
program_code	*Program code*	string	no
reference	References	object	no (1+)
namespace	Namespaces	object	no (1+)

Possible values for **event_type**: *None*, or refer to the
List of Server Events.

Possible values for **language**: *CSharp, JavaScript, VB*

reference

Node	Description	Type	Required
name	*Reference name*	string	yes

namespace

Node	Description	Type	Required
name	*Namespace name*	string	yes

Example

```
<serverevent>
 <name>ExpandAllFolders</name>
 <id>0</id>
 <event_type>OnAfterLoadReportsList</event_type>
 <language>CSharp</language>
 <program_code>Extensions.ServerEvents.ExpandAllFolders(sessionInfo, (TreeNodeCollection)args[0]);
return null;</program_code>
 <reference>
  <name>Extensions.dll</name>
 </reference>
 <reference>
  <name>WebReports.dll</name>
 </reference>
 <namespace>
  <name>WebReports.UI.Controls</name>
 </namespace>
</serverevent>
```

Storage Management

The Storage Management article contains detailed descriptions of each of these settings.

Each of these settings may be accessed via the .NET API's **WebReports.Api.Api.SetupData.StorageMgmtConfig** namespace. For example:

```
api.SetupData.StorageMgmtConfig.ClassName = "WebReports.ContentDatabase.StorageMgmtDatabase";
```

Outer node: <storagemgmt></storagemgmt>

Node	Description	Type	Required
assembly **API key:** *Assembly*	Full path to the DLL implementing the IStorageManagement interface	string	yes
class **API key:** *ClassName*	Name of the class in the assembly that implements IStorageManagement	string	yes
report_list_cache_timeout **API key:** *ReportListCacheTimeout*	Number of seconds before the report list cache expires	integer	yes
report_xml_cache_timeout **API key:** *ReportXmlCacheTimeout*	Number of seconds before the report definition XML cache expires	integer	yes
theme_list_cache_enabled **API key:** *ThemeListCacheTimeout*	Whether or not theme list caching is turned on or off	boolean	yes
report_xml_cache_enabled **API key:** *ReportXmlCacheEnabled*	Whether or not report definition XML caching is enabled	boolean	yes
default_inherit_flag **API key:** *DefaultInheritFlag*	Determines if new content will inherit permissions of its parent.	boolean	yes
default_party_type_id **API key:** *DefaultPartyTypeId*	If default_inherit_flag is *False*, this is the party_type_id that new content will be assigned.	integer	no
default_access_flags **API key:** *DefaultAccessFlags*	If default_inherit_flag is *False*, this is the access_flags bitmap that new content will be assigned. v2021.1+ If default_inherit_flag is *False*, this is a comma-seperated list of ContentPermission items that new content will be assigned.	ContentPermission v2021.1+	yes

option option nodes are used to store Storage Management options and settings data in name-value pairs	<table><tr><th>Name</th><th>Value Description</th></tr><tr><td>DbType</td><td>Database type</td></tr><tr><td>DbProvider</td><td>Database provider</td></tr><tr><td>ConnectionString</td><td>Connection string to Storage Management database</td></tr><tr><td>ReportListCacheKey</td><td>Key which is used to determine when the report list cache should be replenished.</td></tr></table>		
table_prefix **API key:** *TablePrefix*	Provide an optional prefix for all Storage Management database tables. Review the Storage Management: Database Schema article for information. This option can be used to add a schema if it exists. **Examples**: 1. test_ will result in tables named *test_content*, *test_party_type*, *test_content_access* and *test_storage_meta* 2. dbo. will result in tables named *dbo.content*, *dbo.party_type*, *dbo.content_access* and *dbo.storage_meta*	string	no
identity	<identity> nodes define Storage Management identity keys in name-value pairs. This facilitates adding identity nodes beyond the included ownerId, userId, classId and companyId or changing their names. **NOTE** Adding identity keys will also require changes to the Party Type table.		

Example

This example implements Exago's out-of-the-box implementation of the IStorageManagement interface and includes an additional custom identity key named location.

```
<storagemgmt>
  <assembly>WebReports.ContentDatabase.dll</assembly>
  <class>WebReports.ContentDatabase.StorageMgmtDatabase</class>
  <report_list_cache_timeout>30</report_list_cache_timeout>
  <report_xml_cache_timeout>30</report_xml_cache_timeout>
  <theme_list_cache_timeout>600</theme_list_cache_timeout>
  <report_list_cache_enabled>False</report_list_cache_enabled>
  <report_xml_cache_enabled>False</report_xml_cache_enabled>
  <default_inherit_flag>False</default_inherit_flag>
  <default_party_type_id>0</default_party_type_id>
  <option name="DbType" value="SQLite" />
  <option name="DbProvider" value="SQLite" />
  <option name="ConnectionString" value="Data Source=D:StorageMgmt.sqlite" />
  <option name="ReportListCacheKey" value="" />
  <table_prefix>sm_</table_prefix>
  <identity name="userId" value="Astro Boy" />
  <identity name="classId" value="Exago, Inc." />
  <identity name="location" value="Kingston" />
  <identity name="companyId" value="Development" />
 </storagemgmt>
```

Monitoring: Setup

Monitoring that allows report management, execution, and performance statistics for the web application and schedulers to be tracked in *v2017.1+*. Monitoring data is stored in local SQLite database files and can be reported on using Exago.

Many actions in the application can be monitored:

- Report Management
  - Edit
  - Execute (begin & end)
  - Save
  - Delete
  - Rename
  - Duplicate
- Report Designer usage
  - Gauge Wizard
  - Google Map Wizard
  - GeoChart Wizard
  - Map Wizard
- Scheduling
  - Scheduled report
  - Schedule Manager

Monitoring for these components can be toggled on or off depending on your needs.

Additionally, you can track track CPU and memory load for each scheduler application so you can fine-tune your load balancing setup.

When the Exago Web Application is installed, the monitoring system is automatically installed, but it is disabled by default. It must be configured and enabled manually. To set up monitoring, you need to configure the monitoring application, set your options for which data to collect, and then set the monitoring service to run automatically.

Configuring Monitoring

> **NOTE**
>
> This article references <WebApp>as a placeholder for the application's install location. The default install locations are /opt/Exago/ on Linux and C:Program FilesExagoExagoWeb on Windows; however, these directories may be changed during installation.

The monitoring system is located in a folder MonitoringService, in the same folder as where the Web application is installed. The web application stores its monitoring data in a Monitoring subfolder of the installation. So you should have the following folders:

- <WebApp>
  - Monitoring
- MonitoringService

Windows: Ensure that the IIS user has Full Control permissions for the *ExagoWeb*Monitoring and MonitoringService folders. See **Configuring IIS** for instructions.

To configure monitoring:

1. In MonitoringService, open the file Monitoring.exe.config in a text or xml editor. For each of the following keys in the <appSettings> element, set the values as follows:
   - exagoAppPath, value="*path*" where *path* is the file path to the web app
     Format: "C:*filepath*" (Windows), "/*file*/*path*/" (Linux)
     A trailing slash (\) or (/) is required
   - userConfig, value="*config*" where *config* is the application config file
     Format: "*filename*.xml"
     Use extension .xml for either the encrypted or unencrypted version
   - webAppUri, value="*uri*" where *uri* is the url virtual path to the web app
     Format: "http:///*local*/*path*/"
     A trailing backslash (/) is required
   - Optional: ExtractionIntervalMinutes, value="*i*" where *i* is the number of minutes between updates to the core database. The default is 3.
   - Optional: StatisticsIntervalMinutes, value="*j*" where *j* is the number of minutes between when each scheduler is polled for performance statistics. The default is 1.

Example

```
<appSettings>
<add key="ExtractionIntervalMinutes" value="3" />
<add key="StatisticsIntervalMinutes" value="1" />
<add key="exagoAppPath" value="C:ExagoWeb" />
<add key="userConfig" value="WebReports.xml" />
<add key="webAppUri" value="http://localhost/monitoring/" />
</appSettings>
```

2. In <*WebApp*>, open the file appSettings.config in a text or xml editor. In the <appSettings> element, set the Monitoring.DbPath key to the folder where your web application's monitoring data file is. The default location is <*WebApp*>Monitoring.

Example

```
<appSettings>
<add key="Monitoring.DbPath" value="C:ExagoExagoWebMonitoring" />
...
</appSettings>
```

3. In the same file, set the options for which types of usage data you want to collect. To turn on monitoring for a specific action, set the value for the key to "true". Available options are as follows:
   - Monitoring.CollectDeleteReportUsage
   - Monitoring.CollectRenameReportUsage
   - Monitoring.CollectDuplicateReportUsage
   - Monitoring.CollectExecuteReportUsage
   - Monitoring.CollectSaveReportUsage
   - Monitoring.CollectSaveReportXmlUsage
   - Monitoring.CollectDesignReportUsage
   - Monitoring.CollectGaugeControlUsage
   - Monitoring.CollectGoogleMapControlUsage
   - Monitoring.CollectMapControlUsage
   - Monitoring.CollectChartControlUsage
   - Monitoring.CollectScheduleReportControlUsage
   - Monitoring.CollectScheduleReportManagerControlUsage

Example

```
<appSettings>
...
<add key="Monitoring.CollectDeleteReportUsage" value="true" />
<add key="Monitoring.CollectRenameReportUsage" value="false" />
<add key="Monitoring.CollectDuplicateReportUsage" value="true" />
<add key="Monitoring.CollectExecuteReportUsage" value="true" />
...
</appSettings>
```

4. Restart your web server for the changes to be applied.

Configuring Scheduler Monitoring

If you want to track scheduled report execution, do the following for each **Scheduler Service** installation:

Open the eWebReportsScheduler.exe.config file in a text or XML editor. Add the following key to the <appSettings> element:

```
<add key="Monitoring.CollectExecuteReportUsage" value="true" />
```

Enabling the Monitoring Service

The monitoring system uses a Windows or Linux service that updates the core database with data from the **Web Application** and **Schduler Services** at specified intervals. This way you have the data from every component in a single location.

The service is installed automatically, but it is not enabled by default. To enable the service:

Windows

1. As an administrator, open the Windows Services manager:
    1. Click **Start**>**Run**.
    2. Type services.msc.
    3. Press Enter.
2. Locate the service Exago Monitoring Service *vX.X.X.X*, where *vX.X.X.X* is your Exago version. Right-click the service and select **Properties**.
3. In the Properties window:
    1. From the Startup Type list, select **Automatic**.
    2. Click **Start** to start the service.
    3. Click **OK**.

Accessing the Monitoring Data With Exago

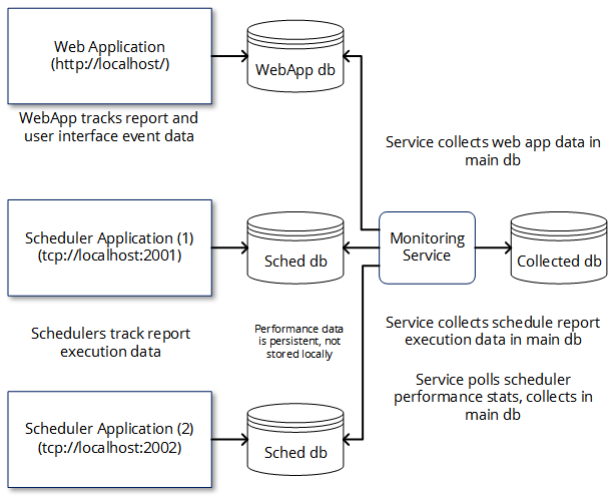
To use monitoring data in reports, you need to add the Monitoring Service's core database file *MonitoringService*/Monitoring.sqlite as an Exago Data Source.

Recommended Resources

- Using SQLite Data Sources
- Data Sources

Monitoring: System Overview

The Monitoring system is structured in the following manner:



Structure of the monitoring system

Web Application Database

The Web Application database stores data for report and user interface events. The type of data tracked depends on your configuration. At the Extract interval, the monitoring service moves this data from the local db to the collected db.

Scheduler Application Databases

Each Scheduler Service has a local database which stores report execution data, if enabled in the configuration. At the Extract interval, the

monitoring service moves this data from the local dbs to the collected db.

Scheduler performance data is “persistent,” that is, always available, and is therefore not stored in the local scheduler dbs. At the Statistics interval, the monitoring service polls the schedulers for their performance statistics, and logs this data in the collected db.

File Paths for Config Files & Databases

*ExagoWeb*MonitoringMonitoring.sqlite – WebApp db

*ExagoWeb*appSettings.config – Select which web app data to track

MonitoringServiceMonitoring.sqlite – Main collected db

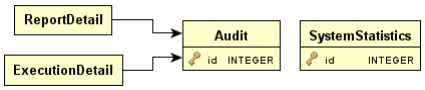
MonitoringServiceMonitoring.exe.config – Set Extraction & Statistics intervals

*ExagoScheduler*Monitoring.sqlite – Scheduler local db

*ExagoScheduler*eWebReportsScheduler.exe.config – Select which scheduled report data to track

Monitoring Database Schema

The monitoring database has three tables that can be used to build reports. This article describes what data is stored, and how to interpret what you see.



Entity-relationship diagram (ERD) for the monitoring database

SystemStatistics

The SystemStatistics table logs the available CPU load and memory for the system on which each scheduler service is installed.

Data is polled at occasional intervals. You can specify the time between polls using the **StatisticsIntervalMinutes** setting in the Monitoring.exe.config file. For instructions, see “Configuring monitoring“.

The table contains the following data columns:

id

An integer used to uniquely identify each row. This is the primary key for the table.

transactionId

(*v2017.2+*) An integer used to associate rows with **type**: cpu available and **type**: free memory to common transactions, in order to facilitate a vertical table transformation and report off both CPU and Memory usage in the same report and chart.

hostId

The scheduler which was polled for system data. Every scheduler is polled at the same time. Schedulers are identified by their host address, as specified in the Administration Console.

Example. tcp://localhost:2010

type

One of:

- cpu available, which indicates that the **value** field in this row shows the CPU load percentage available at this time.
- free memory, which indicates that the **value** field in this row shows the amount of free memory at this time.

value

One of:

- A value indicating the CPU load available at this time, as a percentage of 100%. This field indicates CPU available if the **type** field for this row shows cpu available. This value is “-1” if the scheduler could not be reached at this time.
- A value indicating the amount of free memory at this time, in megabytes (MB). This field indicates free memory if the **type** field for this row shows free memory. This value is “-1” if the scheduler could not be reached at this time.

Note. This field should be formatted as a decimal, either in the metadata for this column, or in the report cell formatting.

timestamp

A datetime value indicating when this scheduler was polled.

Transform

The following **vertical transformation** is recommended for the SystemStatistics table:

```
<entity>
 <!--<entity_name></entity_name>-->
 <db_name>SystemStatistics</db_name>
 <!--<datasource_id></datasource_id>-->
 <object_type>table</object_type>
 <key>
  <col_name>transactionId</col_name>
 </key>
 <transform>
  <col_name>type</col_name>
  <val_name>value</val_name>
  <non_transform_col>
   <col_name>timestamp</col_name>
   <data_type>8</data_type>
  </non_transform_col>
  <non_transform_col>
   <col_name>hostId</col_name>
   <data_type>0</data_type>
  </non_transform_col>
  <non_transform_col>
   <col_name>transactionId</col_name>
   <data_type>0</data_type>
  </non_transform_col>
 </transform>
</entity>
```

Audit

The Audit table records when certain events, which you specify, happen to reports. This table records data for the web application and the schedulers.

Data is logged at the time of each event, but the data is only collected in the core database at occasional intervals. You can specify the time between data collections using the **ExtractionIntervalMinutes** setting in the Monitoring.exe.config file. For instructions, see “Configuring monitoring“.

The table contains the following data columns:

id

An integer used to uniquely identify each row. This is the primary key for the table.

hostId

The application for which this action took place. The web application and schedulers are identified by their host address.

Example. tcp://localhost:2010
Example. http://localhost

transactionType

A string indicating which type of event has triggered this row to be created. One of:

- Execute Report
- Rename Report
- Duplicate Report
- Delete Report
- Design Report
- Save Report XML
- Chart Control
- Gauge Control
- Map Control
- Google Map Control

userId

The userId parameter for this event.

companyId

The companyId parameter for this event.

timestamp

A datetime value indicating when this event happened.

auditId

For rows where the **transactionType** is Execute Report, this field joins up to two rows in the **ExecutionDetail** table that indicate when this execution started and, if successful, when it ended.

This field also joins rows in the **ReportDetail** table which give some information about the report in which the logged event happened.

ExecutionDetail

This table records data for report execution events.

Up to two rows for each event are created:

- The first has **transactionType** Report Execution Begins, which logs when the report execution started.

- The second has **transactionType** Report Execution Ends, which logs when the report execution ends. If the report execution was not successful, this row will not be created.

The table contains the following data columns:

auditId

An integer used to join up to two rows in this table with a row in the **Audit** table.

transactionId

A globally unique identifier (GUID) for this execution. This GUID is used in several places throughout Exago. Notably, it is used as the file name for scheduled reports which have been saved to disk.

transactionType

One of:

- Report Execution Begins, which indicates that the **timestamp** value for this row shows when this execution started.
- Report Execution Ends, which indicates that the **timestamp** value for this row shows when this execution ended.

timestamp

A datetime value indicating when this execution started or finished, depending on the value of **transactionType**.

Note. This table uses columns (**transactionId** and **transactionType**) as a primary key.

Transform

The following **vertical transformation** is recommended for the ExecutionDetail table:

```
<entity>
 <!--<entity_name></entity_name>-->
 <db_name>ExecutionDetail</db_name>
 <!--<datasource_id></datasource_id>-->
 <object_type>table</object_type>
 <key>
  <col_name>transactionId</col_name>
 </key>
 <transform>
   <col_name>transactionType</col_name>
   <val_name>timestamp</val_name>
   <non_transform_col>
    <col_name>auditId</col_name>
    <data_type>2</data_type>
   </non_transform_col>
 </transform>
</entity>
```

ReportDetail

This table records information about the reports which relate to events in the Audit table.

auditId

An integer used to join a row in this table with a row in the **Audit** table.

reportId

The file path and name of the report which the event affected.

reportType

The type of report: advanced, express, expressview, chained, dashboard

Styling Exago

Customizing and Styling Exago comes in many forms. The three major components are:

1. **Application Themes** (The look and feel of the BI platform)
2. **Chart and Designer Themes** (Color scheme for visual components)
3. **Localization** (The language and default text elements of the BI platform)

Application Themes

Application Themes are a way to separate your custom looks from the default look of the Exago Web Application. By maintaining themes separately you are able to have a base reference point to work with in future updates and looks.

Below you can find useful materials regarding **Application Themes**.

- Getting started with Application Themes (Article)
- Application Themes (Article)
- Application Theme Maker (Downloadable Utility)

Another useful component of the applicationâ€™s look and feel is the **Getting Started** content that Exago provides as a default landing page in the Web Aplication. This can be easily modified or removed to meet your particular use case.

Here are materials related to **Getting Started Content**. There is also a support lab walkthrough of making changes to both of these components

should you prefer a guided approach.

- Customizing Getting Started Content (Article)
- Visual Integration and Localization Walkthrough (Support Lab)

Chart and Designer Themes

Exago offers many chart and report designer themes out of the box. These can be easily manipulated, added to, or changed as necessary. You could have a theme specific to some users or clients, or provide a list of options out of the box.

These resources will help with creating your own Exago specific **Chart/Designer Themes**.

- Theme Maker (Web-Based Utility)
- Themes (Article)
- Custom Themes Walkthrough (Support Lab)

Localization

The final piece is customizing Exago's language to fit your application or territories. Whether you are catering to a global audience, or if you want to tweak Exago's default language and naming conventions to better match your own, there should be no issues making these changes with our highly configurable application.

Below are specific walkthroughs for how to access and change Exago Blâ€™s **Language Files and Test Elements**.

- Multi-Language Support (Article)
- Visual Integration and Localization Walkthrough (Support Lab)

Styling the Home Page

Visually modifying and branding the user interface is a simple but effective step toward integrating Exago into a host application. For styling purposes Exago can be thought of as a control that sits within a div on an .aspx (html) page.

Aesthetic changes can be made for single users or groups of users by directing each user/group to different custom .aspx pages. However, we suggest using **Application Themes** instead to encapsulate groups of changes, which can be selected dynamically in the API.

To visually integrate Exago, make a copy of the *ExagoHome.aspx* file and modify the elements surrounding the Exago control or override the CSS of the user interface itself.

> **NOTE**
>
> Do not make changes directly to *ExagoHome.aspx* as they will be overwritten during upgrades. Instead use the example below to create a custom .aspx page.

Exago Control

The example below is the minimum code necessary to embed the Exago control.

```
<%@ Page Language="C#" EnableViewState="false" EnableEventValidation="false" %>
<%@ Register src="WebReportsCtrl.ascx" tagname="WebReportsCtrl" tagprefix="wr" %>

<!DOCTYPE html>

<html>
  <head runat="server">
    <title>Exago</title>
  </head>
  <body>
    <form runat="server">
      <wr:WebReportsCtrl ID="WebReportsCtrl" runat="server" />
    </form>
  </body>
</html>
```

Exago Control Properties

Several properties can be set on the Exago Control to modify various settings and behaviors of Exago. The following properties can be set.

- **ConfigFile** â€” Loads a configuration file other than that created by the Administration Console (e.g. ConfigFile="NorthwindConfig.xml").

  > **NOTE**
  >
  > If entering Exago through the Api this parameter is ignored.

- **LanguageFile** â€” Specify which language files to use in place of the 'Language File' parameter of **Main Settings** in the configuration file. (e.g. LanguageFile="es-mx, getting-started-custom").
- **ForceIECompatMode** â€” Setting to *True* will force certain JavaScript functions to working in 'compatibility' mode. This may be necessary if dragging a Data Field into a cell of the Report Designer does not work properly. (e.g. ForceIECompatMode="true").
- **XUaCompat** â€” Controls whether to remove the meta u-ax-compatible tag when running reports to PDF in IE8. The default is *false* which removes the tag. If you are experiencing issues downloading PDF reports in IE8 setting this flag to True may resolve the issue. (e.g. XUaCompat="true").

> **NOTE**
>
> IE8 is no longer supported by Exago, as of *v2016.1*.

Changing CSS and Images

All of the CSS and images used by Exago can be modified within the aspx page if desired. However we recommend using an **Application Theme** instead.

Any icon in Exago can be changed on a per-company or per-user basis:

1. Create the custom images you would like to display.
2. Identify the Id of the image you want to change. See **Finding Image Ids** for more details.
3. Create a language file that maps the Ids to the location of the custom images. See **Multi-Language Support** for more information.

Example

```
<element id="ExportTypeMenuHtml" image= "ConfigImagesCustomHTMLExecutIconLarge.png"></element>
```

Hovering Images

For icons that have hover effects there is a special naming convention.

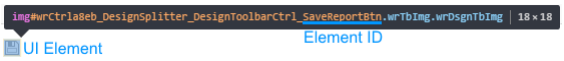
To change custom icons with hover effects:

1. Follow the steps above to create the non-hover icon.
2. Create the custom icon with the hover effect. Save it to have the same name as the non-hover icon and append "_h" to its name.

Finding Image Ids

To find the Ids of icons in Exago:

1. Open Exago in a browser.
2. Use the browserâ€™s developer tools to inspect the icon you want to change. For most browsers this can be done by pressing F12.
3. Look at the id property of the icon. There will be several words separated by underscores. Use the last element as the image Id (see example below).



Customizing Getting Started Content

The **Getting Started** tab is displayed as a user enters the Exago Web Application user interface. This tab can be customized by customizing HTML in a language definition file. This is done by modifying the language element <GettingStartedContent> in the <WebApp>ConfigLanguagesen-us-getting-started.xml

> **NOTE**
>
> This article references <WebApp> as a placeholder for the Web Application's install location.

To assist in customizing the Getting Started tab, Exago provides several JavaScript functions and HTML5 attributes to open the New Report Wizard, execute reports, open other custom tabs and display reports as Dashboards.

The figure below demonstrates a custom Getting Started tab. The two charts are loaded with the data-onloadreportname HTML5 attribute. The links to the New Report Wizard and Dashboard Designer use the void wrStartNewReportWizard([string reportType]) function.



> **NOTE**
>
> It is recommended to make custom tabs in a separate language file to make it easy to change tabs by user or groups of users. See **Modifying Select Language Elements**.

Referencing Another HTML File v2020.1+

Getting Started content may alternatively be sourced from an external HTML file by using the src attribute of the element node. The value must be a file relative to <WebApp>ConfigLanguages.

In the following example, the user's Getting Started page will be replaced by the content in <WebApp>ConfigLanguageses-mx-getting-started.html and the Admin Console's Getting Started Page will be replaced by the content in <WebApp>ConfigLanguageses-mx-admin-getting-started.html.

```
<?xml version="1.0" encoding="utf-8" ?>
<Language>
 <element id="GettingStartedContent" src="es-mx-getting-started.html"></element>
 <element id="AdminGettingStartedContent" src="es-mx-admin-getting-started.html"></element>
```

JavaScript Functions

It is possible to access the JavaScript functions below in the alternative HTML file, by including this line in the file:

```
<script defer src="WrScriptResource.axd?s=GettingStarted"></script>
```

To provide custom JavaScript beyond what is provided by Exago, the WrScriptResource.axd endpoint cannot be used. Instead either:

- write the JavaScript directly in the custom HTML page between standard <script> tags (this is the easiest). Or,
- source JavaScript files using an absolute link to a CDN or other hosting location.

Creating Additional Custom Tabs

Additional custom tabs can be created by creating a language elements with a unique name containing the HTML content. Custom tabs can be opened with the JavaScript function **wrAddTabbedContent**.

This example demonstrates a custom tab that has buttons to launch a report in different formats.

```
<element id="QuickReportsTab">
 <button value="HTML" onclick="wrExecuteReport('My Reports\Revenue', 'html');" />
 <button value="EXCEL" onclick="wrExecuteReport('My Reports\Revenue', 'excel');" />
 <button value="PDF" onclick="wrExecuteReport('My Reports\Revenue', 'pdf');" />
 <button value="RTF" onclick="wrExecuteReport('My Reports\Revenue', 'rtf');" />
 <button value="CSV" onclick="wrExecuteReport('My Reports\Revenue', 'csv');" />
</element>
```

JavaScript Functions

To assist with the creation of custom tab content, Exago provides a number of JavaScript functions to allow custom HTML to call features of Exago.

void wrStartNewReportWizard([string reportType])

Description	Opens the **Report Wizard** in a new tab. Optionally, specify which wizard to open with the *reportType* argument. Available values forÂ *reportType*: • advanced • express • expressview • dashboard • chained
Example	`<p>Click <span style="text-decoration:underline; cursor:pointer;" onclick="wrStartNewReportWizard();">here</span> to create a new report.</p>`

void wrStartDuplicateReportDialog(string reportFolder\reportName)

Description	Opens the Duplicate Report dialog.
Remark	If the report name is null or blank Exago will use the report selected in the Main Menu.
Example	`<p>Click <span style="text-decoration:underline; cursor:pointer;" onclick=" wrStartDuplicateReportDialog();">here</span> to create a duplicate this report.</p>`

void wrExecuteReport(string reportFolder\reportName, string format)

Description	Executes the specified report in the specified format.
Example	`<input type="button" class="Button" value="HTML" onclick="wrExecuteReport('Sales Reports\Revenue by Category','html')`

string wrGetSelectedReportName()

Description	Returns the name of the report that is selected in the Report Tree.
Remark	The returned string will include the folder structure of the report separated by slashes.

void wrAddTabbedContent(string ContentID, string TabName)

Description	Opens a new tab and loads the HTML stored in the element of the Language file that corresponds to the ContentID string.
Remark	The ContentID should match the element ID of the HTML you want to load. The TabName can match the element ID of the name you want the tab to display, or a custom string with the name of the tab.

JavaScript HTML5 data- Attributes

To assist with the creation of custom tab content, Exago provides a number of HTML5 data- attributes to allow custom HTML to call features of Exago.

data-onloadreportname="ReportFolderReportName"

Description	Executes a report as HTML and loads it into a div or iframe. Dashboards are not compatible with data-onloadreportname.
Remark	The report string should be formatted as “Report Folder\Report Name.” **NOTE** When using this function make sure the setting **Enable Debugging** in **Other settings** is **False**.
Example	`<div class="Report"Â data-onloadreportname="Employee Reports\Number of Sales by Employee"></div>`

data-useviewer="True/False"

Description	Specifies to load a report as raw HTML or utilize Exago dynamic report viewer.
Remark	Default value is *true*. In cases where the dynamic capabilities of the Report Viewer is not needed set to *false* to load raw HTML. This has a similar behavior to using either ApiAction vs. GetExecute methods via the APIs.
Example	`<div class="Report"Â data-onloadreportname="Employee Reports\Number of Sales by Employee" data-useviewer= â€œfalseâ€></div>`

data-enablescrolling="*value*"

Description	Specifies whether or not to show scroll bars. This can helpful for certain reports that may not fit exactly within the startup content.
Remark	Available values for *value*: • *true* â€” show scrollbars (default) • *false* â€” do not show scrollbars
Example	`<div class="Report"Â data-onloadreportname="Employee Reports\Number of Sales by Employee" data-enablescrolling= â€œfalseâ€></div>`

data-reloadinterval="*value*"

Description	Reloads a report every *value* seconds.
Remark	This function is used in conjunction with **data-onloadreportname**.
Example	`<div class="Report"Â data-onloadreportname="Employee Reports\Number of Sales by Employee" data-reloadinterval="2"></div>`

data-allowexport="*value*"

Description	Specifies whether or not to show the export menu for the report.
Remark	Available values for *value*: • *0* â€” do not show the menu (default) • *1* â€” show the menu
Example	`<div class="Report"Â data-onloadreportname="Employee Reports\Number of Sales by Employee" data-reloadinterval="1"></div>`

Custom Context Sensitive Help

Exago is installed with context sensitive help. When a user clicks the **Help** icon a tab opens to display the appropriate section of the Exago User Guide. The content of this tab can be replaced with custom content managed by the host application.

To implement **Custom Context Sensitive Help**:

1. Create a minimum of one webpage for the custom help content.
2. Set the **Admin Console** > **General Settings** > **Feature/UI Settings** > **Custom Help Source** setting to the URL of the webpage from step 1. Prefix the URL with url=.
   Example

   ```
   url=http://www.Customhelp.com/Exago;
   ```

> **IMPORTANT**
>
> Supplying a value for the **Custom Help Source** will replace the Exago provided help system throughout the entire application.

Now, when a user clicks the **Help**

icon a tab opens to display the content from the supplied **Custom Help Source** URL.

The userâ€™s language and a context parameter will be appended as a query string to the URL.

- The **language** is determined by the value of the **Admin Console** > **General Settings** > **Main Settings** > **Language File** setting.
- The **context parameter** is determined by where in the application the user clicked the **Help** icon, selected from Table A below.

Example

```
http://www.customhelp.com/Exago?helpKey=tabChartType&language=en-us
```

Table A â€" Custom Context Help Context Parameters

Context Parameter aka “helpKey”	Details
tabexecute	The user is in the **Report Viewer**.
ExpressView	
tabExpressView	The user has the ExpressView Designer active.
tabExpressViewFilter v2019.2.37+ v2020.1.20+ v2021.1.8+	The user is interacting with the Filters tab of the ExpressView Designer’s Properties Pane
tabExpressViewSort v2019.2.37+ v2020.1.20+ v2021.1.8+	The user is interacting with the Sorts tab of the ExpressView Designer’s Properties Pane
tabExpressViewViz v2019.2.37+ v2020.1.20+ v2021.1.8+	The user is interacting with the Visualizations tab of the ExpressView Designer’s Properties Pane
dialogExpressViewFormulaEditor v2019.2.37+ v2020.1.20+ v2021.1.8+	The user has the Formula Builder in the ExpressView Designer open.
Express Report Wizard	
tabExpressName	The user has the Name tab of the Express Report Wizard active.
tabExpressCategories	The user has the Categories tab of the Express Report Wizard active.
tabExpressSorts	The user has the Sorts tab of the Express Report Wizard active.
tabExpressFilters	The user has the Filters tab of the Express Report Wizard active.
tabExpressLayout	The user has the Layout tab of the Express Report Wizard active.
tabExpressOptions	The user has the Options tab of the Express Report Wizard active.
Chained Report Wizard	
tabChainedName	The user has the Names tab of the New Chained Report Wizard active.
tabChainedReports	The user has the Reports tab of the New Chained Report Wizard active.
tabChainedOptions	The user has the Options tab of the New Chained Report Wizard active.
CrossTab Wizard	
tabCrosstabName	The user has the Names tab of the New CrossTab Report Wizard active.
tabCrosstabCategories	The user has the Categories tab of the New CrossTab Report Wizard active.
tabCrosstabFilters	The user has the Filters tab of the New CrossTab Report Wizard active.
tabCrosstabLayout	The user has the Layout tab of the New CrossTab Report Wizard active.
dialogCrosstabOptions	The user has the CrossTab Data Designer dialog open.
Report Designer	
tabDesign	The user is editing an Advanced or CrossTab report in the Advanced Report Designer.
dialogName	The user has the Rename Menu active. This key is never called in *v2021.1+*
dialogDescription	The user has the Description Menu active.

	This key is never called in *v2021.1+*
dialogCategories	The user has the Manage Data Objects dialog open.
dialogSorts	The user has the Sorts dialog open.
dialogFilters	The user has the Filters dialog open.
dialogGeneralOptions	The user has the General Options dialog open.
dialogHtmlOptions	The user has the Report Viewer Options dialog open.
listItemReportHtmlOptionsGeneral	The user has the General section of the Report Viewer Options dialog open.
listItemReportHtmlOptionsFilters	The user has the Filter section of the Report Viewer Options dialog open.
listItemReportHtmlOptionsSorts	The user has the Sorts section of the Report Viewer Options dialog open.
dialogTemplates	The user has the Template dialog open.
dialogJoins	The user has the Joins dialog open.
dialogJoinEdit	The user has the Joins dialog open.
dialogFormulaEditor	The user has the Formula Editor open.
dialogLinkedReport	The user has the Linked Report dialog open.
tabCellFormatNumber	The user has the Number tab of the Cell Format dialog active.
tabCellFormatBorder	The user has the Border tab of the Cell Format dialog active.
tabCellFormatConditional	The user has the Conditional tab of the Cell Format dialog active.
dialogCrosstabDesign	The user has the CrossTab Data Designer dialog open.
dialogGroup	The user has the Group Section dialog open.
dialogSectionShading	The user has the Section Shading dialog open.
dialogShowSQL	The user has the Show Generated SQL open.
dialofReportLevelParameters	The user has the Report Level Parameters open.
tabChartType	The user has the Type tab of the Chart Wizard open.
dialogChartBenchmarkLine	The user has the Benchmark Lines dialog open.
dialogChartDataFormat	The user has the Data Layout… dialog of the Chart Wizard open.
tabChartLabels	The user has the Labels tab of the Chart menu open.
tabChartData	The user has the Data tab of the Chart Wizard open.
tabChartAppeance	The user has the Appearance tab of the Chart menu active.
tabChartSizeAndPreview	The user has the Size and Preview tab of the Chart Wizard open.
tabMapType	The user has the Type tab of the GeoChart Wizard open.
tabMapLocations	The user has the Locations tab of the GeoChart Wizard open.
tabMapData	The user has the Data tab of the GeoChart Wizard open.
tabGaugeType	The user has the Appearance tab of the Gauge Wizard open.
tabGaugeData	The user has the Data tab of the Gauge Wizard open.
tabGoogleMapAppearance	The user has the Appearance tab of the Google Maps Wizard open.
tabGoogleMapLocations	The user has the Locations tab of the Google Maps Wizard open.
tabGoogleMapData	The user has the Data tab of the Google Maps Wizard open.
tabGoogleMapSizeAndPreview	The user has the Size and Preview tab of the Google Maps Wizard open.
listItemReportCacheOptionsGeneral	The user has the Execution Caching dialog open.

Dashboards

tabDashboard	The user has the **Dashboard Viewer** active
tabDashboardDesigner	The user has the **Dashboard Designer** active.
dialogDashboardUrlOptions	The user is adding or editing a URL tile on the canvas.
dialogDasboardImageOptions v2019.2.37+ v2020.1.20+ v2021.1.8+	The user is adding or editing an Image tile on the canvas.
dialogDashboardTextOptions v2019.2.37+ v2020.1.20+ v2021.1.8+	The user is adding or editing a Text tile on the canvas.
dialogDashboardName	The user has the Dashboard Rename menu active.

dialogDashboardDescription	The user has the Dashboard Description menu active.
dialogDashboardOptions	The user has the Dashboard Options menu active.
tabDashboardReportOptions	The user is adding or editing an Existing Report tile.
tabDashboardReportOptionsFilterPrompts	The user is interacting with the Filters menu.
tabDashboardReportOptionsParameterPrompts	The user is interacting with the Parameters menu.
tabDashboardReportOptionsOptions	The user has the Options tab of the Insert Report menu active.
tabDashboardFilterOptionsReports	The user has the Reports tab of the Insert Filter menu active.
tabDashboardFilterOptionsFilter	The user has the Filter tab of the Insert Filter menu active.
dialogDashboardVisualizationOptions	The user is adding or editing a visualization tile.

Scheduler

tabScheduleReportManager	The user has the Schedule Report Manager active.
tabScheduleRecurrence	The user has the Recurrence tab of the New Schedule Wizard active.
tabScheduleParameters	The user has the Parameter tab of the New Schedule Wizard active.
tabScheduleFilters	The user has the Filter tab of the New Schedule Wizard active.
tabScheduleRecipients	The user has the Recipients tab of the New Schedule Wizard active.
tabScheduleBatch	The user has the Batch tab of of the New Schedule Wizard active.

> **TIP**
>
> Create a default page to handle any cases where an undocumented or null context parameter is passed. This guarantees that a valid help page will always be displayed.

Themes

Themes allow a user to quickly stylize reports or elements of reports such as maps and charts.Â This article covers Chart, ExpressView, CrossTab, GeoChart and Express Report Themes. For styling the entire user interface, refer to the Application Themes article. Any further references to Themes throughout this article refers specifically to the aforementioned visualization and report types.

> **NOTE**
>
> This article references <WebApp>/, as a placeholder for the Web Application install location. The default install location is C:Program FilesExagoExagoWeb on Windows or /opt/Exago/ on Linux. These directories can be changed during installation.

Exago comes with several themes pre-installed. Pre-installed themes are saved in the <WebApp>Themes folder of Exago.

Additional custom themes can also be created. By default custom themes are saved in the Storage Management database. Alternatively the host application can manage theme storage by implementing the IStorageManagement interface.

Chart Themes

A user can quickly select colors for Charts by applying a chart theme.

There are two way to create a **chart theme**:

- Manually
  1. In folder specified in the **Admin Console** > **General** > **Main Settings** > **Report Path** create a text file containing a comma separated list of the hexadecimal values of the desired colors.
  2. Save the file with the theme name.
  3. Change the extension to .wrth.
- With the Exago Theme Maker
  1. Launch the Exago Theme Maker, and follow the instructions in the Exago Theme Maker article.
  2. Download the theme created with the Theme Maker.
  3. In folder specified in the **Admin Console** > **General** > **Main Settings** > **Report Path**, save theÂ .wrthÂ file.

> **NOTE**
>
> The file name will be displayed to the end user. To translate the name of a custom theme, see Multi-Language Support for Theme Names below.

Example

The theme file below implements the built-in theme *Autumnal Sunset.*

```
#FFAA5C,#DA727E,#AC6C82,#685C79,#455C7B
```

ExpressView Themes

A user can quickly style ExpressViews by applying a theme. Custom theme creation is covered below:

- Create a Custom ExpressView Theme (*v2021.1+*)
- Create a Custom ExpressView Theme (*v2017.1â€“v2021.1*)

CrossTab Themes

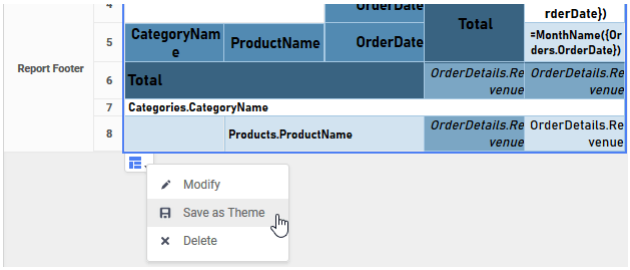
A user can quickly style CrossTabs by applying a theme. CrossTab themes can specify background color, foreground color, section shading, borders, fonts and text size.

To create custom CrossTab themes:

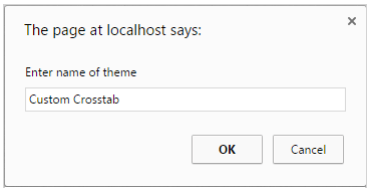
1. Create a CrossTab with several Tabulation Data cells, Row Headers, Column Headers as well as sub-totals and grand totals. We recommend CrossTab Themes have at least five Row Headers, Column Headers, Tabulation Data, sub-total rows, and sub-total columns as well as a grand total row and a grand total column.

   > **NOTE**
   >
   > If a user adds more Tabulation Data, Row Headers or Column Headers than existed on the theme they will appear without styling.

2. In the Report Designer stylize each cell as desired.
3. Hover the mouse over the CrossTab. Notice the **CrossTab Design MenuÂ** icon appears in the bottom left corner.
4. Hold **Ctrl + Alt + Shift** and click on the **CrossTab Design MenuÂ** icon



5. ClickÂ **Save as Theme**.
6. Enter a name for the theme. This name will be displayed to the end-users.



GeoChart Themes

A user can quickly select colors for GeoCharts by applying a theme.

There are two ways to create **GeoChart themes**:

- Manually
  1. In folder specified in the **Admin Console** > **General** > **Main Settings** > **Report Path** create a text file containing a comma separated list of the hexadecimal values of the desired colors.
  2. Save the file with the theme name.
  3. Change the extension to .wrtm.
- With the Exago Theme Maker
  1. Launch the Exago Theme Maker, and follow the instructions in the Exago Theme Maker article.
  2. Download the theme created with the Theme Maker.
  3. In folder specified in the **Admin Console** > **General** > **Main Settings** > **Report Path**, save theÂ .wrtmÂ file.

> **NOTE**
>
> The file name will be displayed to the end user. To translate the name of a custom theme, see Multi-Language Support for Theme Names below.

Express Report Themes

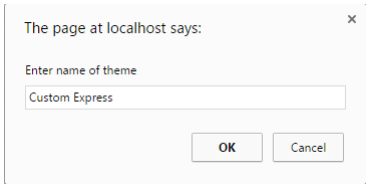
A user can quickly style Express Reports by applying a theme. Express report themes can specify background color, foreground color, section shading, borders, fonts and text size.

To create a custom Express Report theme:

1. Create an Express Report with Headers, Footers and a Page Header/Footer and a Grand Total. We recommend Express Report Themes utilize many Columns, Headers and Footers.

> **NOTE**
>
> If a user adds more Columns, Headers, or Footers than existed on the theme they will appear without styling.

2. In the Layout tab stylize the report as desired.
3. Hold **Ctrl + Alt + Shift** and click on the **SaveÂ** icon.
4. Enter a name for the theme. This name will be displayed to the end-users.



Multi-Language Support for Theme Names

To support multi-language functionality, if the theme name concatenated with â€˜_wrThemeIdâ€™ matches the id of any element in the language files then the string of that language element will be displayed to the user instead of the theme name.

For example, in the Basic theme that is installed with Exago, there exists a language id â€˜Basic_wrThemeIdâ€™. The string associated with this id is displayed. For more information see **Multi-Language Support**.

Create a Custom ExpressView Theme (v2021.1+)

> **NOTE**
>
> This section applies to ExpressView in *v2021.1+* of the application. For previous versions, refer to Create a Custom ExpressView Theme (pre-v2021.1)

A **theme** can be applied to an ExpressView that will change the colors of the elements on the canvas such as the totals, column headers, data rows, borders and divider lines. Themes will also apply in certain situations when the ExpressView is exported.

In addition to several themes provided with Exago, administrators may create their own.

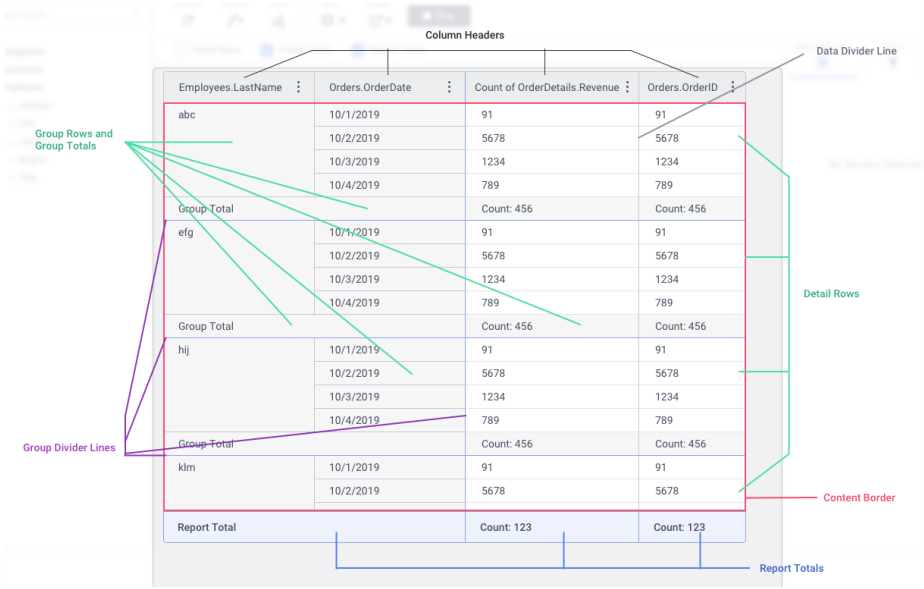
Create an ExpressView Theme

> **NOTE**
>
> This article references <WebApp> as a placeholder for the Web Application's install location. The default install location is C:Program FilesExagoExagoWeb or /opt/Exago/ on Linux. However, these directories can be changed during installation.

1. Copy one of the existing .wrtev2 theme files from the <WebApp>Themes directory.
2. Edit the properties of the JSON objects in the file, as detailed below in the Theme JSON and Color Definition JSON sections.
3. Save the file with the name of the new theme back to the <WebApp>Themes directory.
4. Load the new theme into the Storage Management database by clicking the **Admin Console** > **Storage Management** > **Load Themes** button.

ExpressView Table Elements

Themes apply coloring to common groups of elements. The following figure depicts the uniquely themable elements:



Click the figure for a larger view

The themable elements are:

- Column Headers
- Data Records
  - Detail Rows
  - Innermost Group (when Detail Rows are hidden)
- Group Records
  - Group Rows
  - Group Totals for all levels of grouping (excluding the innermost Group Column when detail rows are hidden)
- Report Totals
- Group Divider Lines, the vertical line separating the groups and detail columns as well as the horizontal line that appears after the last record of the outermost group
- Data Divider Line, separating detail columns
- Content Border

Theme Definition

An ExpressView theme is a JSON file with a .wrtev2 file extension.

Theme JSON

Name	Type	Description
ThemeName	string	The name of the theme
PreviewColors	array of strings	An array of hexadecimal HTML color codes that will appear in the ExpressView Designer's Canvas menu preview window
ColumnHeaders	Color Definition	The font, background and border color definition for the **Column Headers**
DataRows	Color Definition	The font, background and border color definition for the **Detail Rows**
GroupSections	Color Definition	The font, background and border color definition for the **Group Rows** and **Group Totals**
ReportTotals	Color Definition	The font, background and border color definition for the **Report Totals**
GroupDivider	string	A hexadecimal HTML color code for the **Group Divider Lines**
ContentBorder	string	A hexadecimal HTML color code for the **Content Border**
DataDivider	string	A hexadecimal HTML color code for the **Data Divider Line**s
CanvasBackground	string	This property is not implemented and will not affect the theming of the ExpressView

Color Definition JSON

Color Definition objects describe the font color, background color and border color of the **Column Headers**, **Detail Rows**, **Group Rows**, **Group Totals** and **Report Totals**.

Name	Type	Description
FontColor	string	A hexadecimal HTML color code for the font (text) color in the item
BackgroundColor	string	A hexadecimal HTML color code for the background color of the item
BorderColor	string	A hexadecimal HTML color code for the borders of the item

Sample

The following sample theme JSON file creates the built-in *Office Park* theme.

```
{
 "ThemeName": "Office Park",
 "PreviewColors": [ "#90D9FD", "#F0F6F8", "#9FC1CE" ],
 "ColumnHeaders": {
  "FontColor": "#2A2D34",
  "BackgroundColor": "#9FC1CE",
  "BorderColor": "#858992"
 },
 "DataRows": {
  "FontColor": "#2A2D34",
  "BackgroundColor": "#FFFFFF",
  "BorderColor": "#E4E5E8"
 },
 "GroupSections": {
  "FontColor": "#2A2D34",
  "BackgroundColor": "#F0F6F8",
  "BorderColor": "#9FC1CE"
 },
 "ReportTotals": {
  "FontColor": "#2A2D34",
  "BackgroundColor": "#90D9FD",
  "BorderColor": "#51AAD5"
 },
 "GroupDivider": "#637A82",
 "ContentBorder": "#C7CACF",
 "DataDivider": "#C7CACF",
 "CanvasBackground": "#F1F1F3"
}
```


Adding Fonts to Exago

Additional fonts may be added to Exago by installing them on the web server. Fonts must be available to the user account the web server is running under. The font must also be available on each user's computer in order to appear in reports.

Windows

Adding additional fonts to Azure App Services is not supported.

> **IMPORTANT**
>
> This procedure requires restarting the web server which will terminate any user sessions in progress.

1. Right-click on the .ttf or .otf font file.
2. Click **Install for all users**.
3. Once all fonts are installed on the server, restart IIS.
4. Verify the font now appears in the Report Designer's font list.

Linux

1. As root, copy the .ttf or .otf font file to the font directory.
   - For CentOS, Debian, Fedora, Red Hat Enterprise: usr/share/fonts/custom
   - For Ubuntu: /usr/local/share/fonts
2. As root, re-build the font-cache by running fc-cache -fv.
3. Verify the font now appears in the Report Designer's font list.

Keyboard Shortcuts

This article lists all the available keyboard shortcuts in the Web Application User Interface and describes their behavior in their applicable contexts.

Where a plus sign **+** appears between two keys, it means hold down the key before it and press the key after it. For example, **Ctrl+V** means hold down the Control key and then press the V key. Release both keys after pressing V.

> **IMPORTANT**
>
> The key mappings are the same for PC and Mac keyboards. Use the Mac Control key when directed.

Global

- **Ctrl+Down Arrow or Ctrl+Right Arrow** â€" Next tab
- **Ctrl+Up Arrow or Ctrl+Left Arrow** â€" Previous tab
- **Ctrl+Shift+V** â€" Show application version number
- **Ctrl+Shift+X** â€" Create debug package (only if debugging is enabled in system configuration)

Report / Data Tree

- **[** â€" Toggle last open tree pane

- **Up Arrow** â€" Previous item
- **Down Arrow** â€" Next item
- **Delete** â€" Delete selected report
- **Enter** â€" Edit selected report
- **/** â€" Search for a report. The cursor will move to theÂ **Search** field on top of the Report Tree.

New Report Menu

- **Up Arrow** â€" Previous item
- **Down Arrow** â€" Next item
- **Enter** â€" Select item
- **Esc** â€" Close menu

Advanced Report Designer

- **Arrows** â€" Select cells
- **Ctrl+S** â€" Save report
- **F2** â€" put focus (move the cursor to) the Formula Bar
- **Delete** â€" Delete cell contents
- **Ctrl+C** â€" Copy cell contents
- **Ctrl+X** â€" Cut cell contents
- **Ctrl+V** â€" Paste cell contents
- **Ctrl+Click** â€" Select multiple cells, rows, or columns
- **Shift+Click** â€" Select a range of cells, rows, or columns
- **Ctrl+Z** â€" Undo
- **Ctrl+Y** â€" Redo
- **Ctrl+B** â€" Bold cell text
- **Ctrl+I** â€" Italicize cell text
- **Ctrl+U** â€" Underline cell text
- **Tab** â€" Select next cell
- **Shift+Tab** â€" Select previous cell
- **Enter** â€" Open cell text editor / close cell text editor and apply changes
- **Esc** â€" Close cell text editor without applying changes / close wizard or modal window

Report Filters Dialog

These shortcuts apply to the Report Filters dialog in the Advanced Report Designer.

- **Ctrl+[** â€" Add an opening parenthesis before the selected filter
- **Ctrl+]** â€" Add a closing parenthesis before the selected filter
- **Ctrl+Shift+[** â€" Remove an opening parenthesis before the selected filter
- **Ctrl+Shift+]** â€" Remove a closing parenthesis before the selected filter

Report Joins Menu (Complex)

- **Ctrl+.** â€" Add parentheses around last clicked condition
- **Ctrl+,** â€" Remove parentheses from around last clicked condition

Formula Editor

- **Up Arrow** â€" Select previous item in search list
- **Down Arrow** â€" Select next item in search list
- **Right Arrow** â€" Select next argument in formula
- **Left Arrow** â€" Select previous argument in formula

Color Picker Menu

- **Enter** â€" Apply selected color and close picker
- **Esc** â€" Close picker without applying color

ExpressView

- **Page Up** â€" Scroll up
- **Page Down** â€" Scroll down
- **Ctrl+Home** â€" Start of report
- **Ctrl+End** â€" End of report
- **Ctrl+Arrows** â€" Scroll the report
- **N** â€" Next page
- **P** â€" Previous page
- **S** â€" Toggle sorts pane
- **F** â€" Toggle filters pane
- **O** â€" Toggle formatting pane
- **E** â€" Toggle settings pane
- **V** â€" Toggle visualizations menu
- **C** â€" Toggle selected cell menu
- **Shift+V** â€" Show/hide visualization
- **Ctrl+R** â€" Refresh data
- **Ctrl+S** â€" Save ExpressView
- **Ctrl+F** â€" Show search report menu

ExpressView Search

- **Enter** â€" Find next result
- **Shift+Enter** â€" Find previous result
- **Esc** â€" Close menu

Dashboard Designer

- **E** â€" Open/close options menu *pre-v2019.2*, opens the Save menu in *v2019.2*
- **Ctrl+S** â€" Save
- **Esc** â€" Deselect all tiles
- **Arrow keys** â€" Moves the selected tiles
- **Delete** â€" Delete the selected tiles
- **Ctrl+A** â€" Select all tiles
- **Ctrl+C** â€" Copy selected tiles
- **Ctrl+V** â€" Paste copied tiles
- **F** â€" Toggle filters menu
- **O** â€" Toggle formatting menu
- **Ctrl+Y** â€" Redo
- **Ctrl+Z** â€"Undo

Dashboard Viewer

- **F** â€" Open/close the Filters pane
- **P** â€" Open/close the Parameters pane

Multi-Language Support

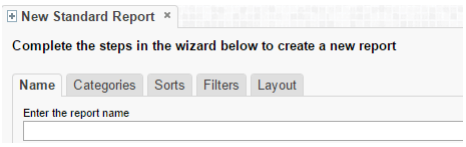
> **NOTE**
>
> The language elements discussed in this article do not include those created by users or administrators such as reports, folders, express report/crosstab themes or Data Field names. To modify Data Field names please see **Column Metadata**. To modify theme names please see **Express Report and CrossTab Themes**.

Any text in the application can be translated or modified. This can be accomplished by creating XML files in the ConfigLanguage directory under Exago's Web Application installation directory. These files map IDs to strings. Any place within Exago that displays text has an associated ID. When a text element is required in the application, Exago will read the file(s) specified in the â€˜Language Fileâ€™ parameter of **Main Settings** and use the string that is mapped to the ID.

Exago comes with both a standard English file en-us.xml and a Spanish translation es-mx.xml. Below is an example of the multi-language functionality. Notice that the prompt text in the **New Report Wizard** can be set by changing the string associated with the ID NewReportLb1.

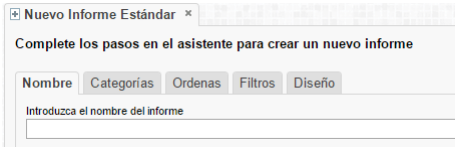
En-us.xml:

```
<NewReport>
 <element id="NewReportLbl">Complete the steps in the wizard below to create a new report</element>
</NewReport>
```



Es-mx.xml:

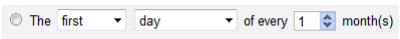
```
<NewReport>
  <element id="NewReportLbl">Complete los pasos en el asistente para crear un nuevo informe</element>
</NewReport>
```



> **CAUTION**
>
> Some language strings contain special place holders between curly brackets (ex. {0}). These hold the place of elements that must be filled in dynamically by Exago. **Do not translate anything inside curly brackets**. The place holders may be moved within the string but do not delete them.

The example below demonstrates three place holders that will be replaced by dropdown menus in the Scheduling Wizard.

```
<element id="ScheduleRecurrenceRelativeMonthlyTxt">The {DayPosition} {DayOfWeek} of every {MonthNumber} month(s)</element>
```



Translating Exago

To translate the entire interface, make a copy of the file en-us.xml and give it a different name. Make sure this copy is in the folder <webapp_dir>/Config/Languages, where <webapp_dir> represents the Exago Web Application installation directory. Without changing the IDs translate the strings as desired (see example above). Then set the **Language File** setting in **Main Settings** to point at your translated copy.

> **NOTE**
>
> If you are using the Exago Scheduler Service be sure to copy all custom language XML files to the â€˜<scheduler_dir>/Languagesâ€™ folder of each Scheduler Service instance.

Modifying Select Language Elements

To change specific language elements without copying the entire mapping you can use a base file and specify changes in separate language files. When you set the **Language File** setting, list the all of the files you want to load separated by commas or semicolons. Exago will load the files from left to right, meaning the first file listed will be used as a base and can be changed by the files loaded after it.

As an example you can create the file en-custom.xml which only contains the lines:

```
<?xml version="1.0" encoding="utf-8" ?>
<element id="GettingStartedTab">Home</element>
```

Set the **Language File** setting to *en-us, en-custom* and the Getting Started tab will reflect the change made in the custom file.

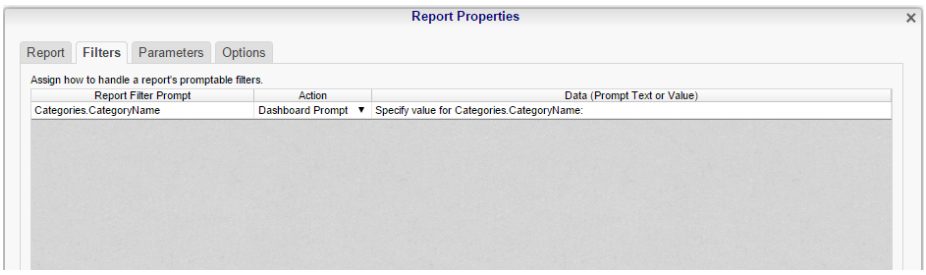
> **NOTE**
>
> Begin all language XML files with the line <?xml version="1.0" encoding="utf-8"
> ?>

Text of Prompting Filters and Parameters on Dashboards

When adding a Report to a Dashboard a user can specify text for any prompting Filters or Parameters. By default this text will match the strings associated with the ids *CompositeReportOptionsFilterDefaultPromptText* and *CompositeReportOptionsParameterDefaultPromptText* respectively.

If a user changes the default and enters a different language Id then the associated text for that new Id will display when the dashboard is executed.

If a user enters text that does not match any language Id the text will be displayed when the dashboard is executed.



Theme Names

Names of custom chart, GeoChart, CrossTab, ExpressView and Express Report themes can also be translated. Review the Multi-Language Support for Theme Names section of the Themes article for more information.

An Overview of Exago Extensions

Exago extensions allow developers to access, extend and modify the platform's capabilities to suit particular needs that can't be handled "out-of-the-box".

The interface for most capabilities is via coded elements, done either via Code Editor in the Administration Console or external code modules provided via .NET Assemblies or Web Services. Other situation-specific options are also available, such as providing custom parameterized SQL.

The following extension types are supported. More information on each type is provided in the remainder of this document:

- Custom SQL
- Custom Functions and Filter Functions
- Server Events
- Action Events
- Custom Data Sources
- Custom Options

Custom SQL

Exago supports a **Custom SQL** type Data Object. Objects of this type are very similar to pre-defined database views. They consist of a SQL statement, have an implied field schema and return a relational value.

There are a few differences from traditional database views however.

- Exago Custom SQL statements can be parameterized. In other words Exago parameters can be embedded within the SQL statement. Parameters serve as placeholders for runtime values. Upon execution of a report any parameter values will be replaced with the value of the parameter at runtime. Any part of the SQL statement can be parameterized in this way.
- Exago Custom SQL statements behave like any other entity type. Administrators can attach column metadata. Report designers can filter, sort and layout fields from the entity, etc.
- Since the SQL is stored in the Exago configuration no intervention by database administrators is required.

Adding Custom SQL

Add a new Data Object and select the Data Source from the dropdown. Instead of selecting a table or view from the name field, click the SQL button on the right of the name field.

You will be presented with the SQL code editor. Specify a name for your Data Object (no whitespace or special characters) then enter your SQL as shown.

Notice the @ and @ parameters. They can be manually typed or inserted by selecting from the list of Parameters and selecting the Add button.

When finished select Ok to save your object's information. You can then specify and Alias and select Unique Key Fields like any other object.

The SessionInfo Object

The global SessionInfo object is available to Custom Functions, Server Events, and Action Events. It is similar to the API object in that it contains the full running state of the Exago system during execution.

Collections of metadata and settings normally set in the Administration Console and/or via the API are accessible through the sessionInfo.SetupData property. Note that for the most part the accessible properties are settable via SessionInfo.

SessionInfo also contains a key object store accessible via two methods:

```
void sessionInfo.SetStoredValue(Object key, Object value)
Object sessionInfo.GetStoredValue(Object key)
```

Custom Functions and Custom Filter Functions

Custom Functions and **Custom Filter Functions** allow developers to create abstracted high level language routines that are used by report designers. Standard Custom Functions appear in the Exago Formula Editor as if they were part of the product. Custom Filter Functions are available to use as values in user formulas.

Both types of functions have access to the SessionInfo object, creating powerful opportunities for context-sensitive processing as well as storage. In the case of Custom Functions, global storage can be used to maintain counts as the report is processing.

Server Events

Server Events are handlers invoked on the *server side* during the normal execution lifecycle. They allow developers to inspect and/or modify running state to achieve situation-specific objectives. Server Event handlers also have access to the global SessionInfo object, providing the same global storage and state variables as mentioned above.

A commonly used event is the OnReportExecuteStart event, which allows the handler to make modifications to a report just prior to execution. Another commonly-used event is OnExecuteSqlStatementConstructed which is invoked after Exago generates SQL but before it is shipped to the data source. This event can be used to simply log the SQL for audit or diagnostics or to modify it to e.g. substitute actual fields for placeholders.

Action Events

Action Events are handlers invoked on the *client side* during the normal execution lifecycle and include an abstracted interface for server-side callbacks to gather data as needed and perform similar tasks. They offer a rich interface for modify charts and dashboard reports in response to user actions. There are also specific interfaces to allow trapping and instrumentation of things like user saves.

Custom Data Sources

In additional to traditional relational data sources Exago can consume data from *programmable* data sources such as .NET assemblies. Connection to and marshalling of the data sources is handled at the metadata layer, just like standard sources. This capability provides two key benefits to end users:

- Data from multiple disparate sources can be combined into a single report, visualization or dashboard
- The end user view is the same regardless of the type of data source. Users will see no difference between relational and programmable data sources when building reports and dashboards.

Custom data sources are often used when it is desirable to make use of an existing business logic layer such as a means to pull data from non-traditional or distributed sources.

Custom Options

Custom Options allow administrators to alter the Exago UI to capture custom input from report designers. The values chosen by the designer are available via the sessionInfo object and are thus accessible to Server Event and Action Event handlers.

Introduction to Action Events

This article applies to the **Admin Console** > **Extensions** >Â **Actions Events** settings.

Action Events can be grouped into two general categories: **Local** and **Global** events.

- **Local** events have two sub-categories:
  - Handlers attached to items in reports and set to fire automatically, or when the item is interacted with in the Report Viewer.
  - Handlers attached to items in the Web Application User Interface and set to fire when that item is clicked.
- **Global** events are active throughout the application, and fire when specific events occur.

This article explains how to create **Local** and **Global Action Events**, describes the ways in which **Action Events** can interact with the Exago application, and lays out examples for common usage.

Create, Edit, Delete Action Events

- To **add** a new action event click **Extensions >Â Action Events** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
- To **edit** an action event either:
  - double click it
  - select it and click the **Edit**Â icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** an action event either:
  - select it and click the **Delete**Â icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new functions click the **Apply** or **Okay** buttons

They can also be added or modified on a per-session basis via the .NET API using the Api.SetupData.ActionEvents server call. This section details how to create **Action Events** through the Web Application but contains important details about programming them which will apply to the .NET API as well.

Action Event Properties

Each **Action Event** has the following properties:

Name

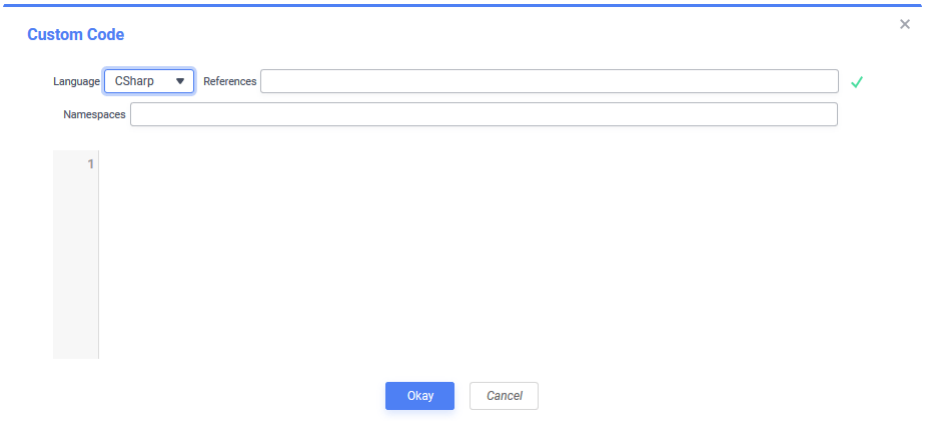
A unique identifier for each **Action Event**.

Function

Can either be custom code or a .NET Assembly method.

Custom Code

To save code directly in Exago, select *Custom Code* from the first **Function** dropdown. Clicking on the second dropdown opens the **Custom Code** dialog.Â See the Writing **Action Events** section below for information on how to access the arguments for each Event.



Custom Code has four additional properties:

- **Language**: Code can be written in C#, JavaScript (in Windows only) or VB.NET.
- **References:** A semicolon-separated list of any libraries that need to be referenced by the **Action Event**. The DLLs must be present in the bin folder of the Exago Web Application, as well as any **Scheduler** bin folders, and the **Web Service API**, if applicable.

  > **NOTE**
  >
  > System.dll does not need to be listed as a reference as it is already available.

- **Namespaces:** A semicolon-separated list of namespaces in the referenced DLLs or the **Exago API** library.
- **Code:** The code that will be executed.
  When done writing code, click the **Test Execution** icon to test the code for errors.

.NET Assembly Method

There are two ways to reference a .NET Assembly method.

- Create a .NET Assembly **Data Source**. Select the desired assembly from the first **Function** dropdown. Clicking on the second dropdown will open a list of available methods.
- Add the .NET Assembly to the Bin folder (for the Web Application, Web Service API, and all Scheduler Services, if applicable). Then in the **Custom Code**, add the assembly as a reference, then invoke the method, e.g.
  `return Extensions.Events.CensorEmployeeBirthYear(sessionInfo, args);`

> **NOTE**
>
> The Assemblyâ€™s DLL will be locked by Exago when it is first accessed. To replace the DLL, unlock it by restarting the IIS App pool.

> **NOTE**
>
> If you want to utilize the **sessionInfo** object that is passed to all **Action Events** the Assembly must include a reference to WebReportsApi.dll. For more information see the article on **SessionInfo**.

> **NOTE**
>
> All methods used as **Action Events** must be static.

Event Type

Select an option in this dropdown to create an local event that will be executed when certain client-side actions are taken.

- **None:** This **Action Event** is a **Global Event**. A value must be set for **Global Event Type**.
- **Load:** TheÂ **Action Event** will execute when a report item is loaded in the **Report Designer**, **Report Viewer**, or upon **Export**. This type of handler is typically used to interpret and then apply alterations to report data, e.g.Â conditionally changing the colors on charts or maps.
  - As of *v2016.2.5+*Â **Load** events can affect Export formats (*PDF*, *Excel*, *RTF*, *CSV*).
- **Click:** The **Action Event** will execute when a user clicks on an item in a report or in the Exago UI. This type of handler is typically used to add additional interactive elements to reports or to the **Report Designer**. Click events will not function onÂ Export formats.
  For information on adding action events to specific reports, see **Adding Action Events to a Report**.

Global Event Type

Select an option in this dropdown to create an event that will be triggered when a condition is met in the Exago application. The **Global Events** article contains descriptions of all of the available event handlers.

> **NOTE**
>
> Selecting a **Global Event Type** will cause Exago to ignore any selected Local **Event Type**.

Assigned UI Item(s)

This field designates a semicolon-separated list of UI item IDs for items in the Exago interface. These elements can be intercepted and modified by entering them in this field.

> **IMPORTANT**
>
> This selection field only applies when the **Event Type** is **Click**. This field will be ignored when any other options are selected.

Writing Action Events

When an **Action Event** is fired, two primary parameter objects are passed: **sessionInfo** and **clientInfo**. These are the main points of interaction with the Exago application.

- **sessionInfo**: Provides access to all the elements of the current Exago session. This is the server-side information. For more information see the article on **sessionInfo**.
  Note: To access **sessionInfo** from a .NET Assembly, a reference to WebReportsApi.dll must be included.
- **clientInfo**: A JavaScript object that is accessed from within the client-side script. Provides access to any specified client-side parameters and information about the item attached to the **Action Event**. For a breakdown of the elements in clientInfo, see **ClientInfo**.

arguments array â€“ The server-side portion of action events can also access an array of input values called **args**. These parameters are passed manually from client code to server code using the functionÂ clientInfo.ServerCallback(eventName, args...).

Global Action Events

Global Actions Events are actions that can be attached to one of a specific list of events that will occur within the Exago application. These events usually trigger in response to user input, but they are not necessarily directly related to the input action, and thus will not transfer information about the user input. However, global events are more reliable than capturing user clicks, especially in response to actions that can be taken in a

variety of ways, such as saving a report.

A subset of global events, namely the ones which are used to handle report tree interaction, require a boolean return value or a Promise of a boolean in the client script. *True* indicates to Exago to stop the normal course of action, which we have replaced with our custom code. *False* indicates that we should continue with the normal action. If any action event returns a Promise Exago will await the Promise before taking action based on the boolean result.

For example, when double-clicking on a third party (non-Exago) report, we may want to launch an external editor instead of the Exago report designer. We would check the report type, and if it is a third party report, we would insert our callout and then return *True*. If it is a regular Exago report, we would continue with the normal course of action by returning *False*.

For these events to be able to have a return value, they must be enclosed within a JavaScript function. This means that if you want to write the full client scripts in the admin console (rather than calling out to a separate function) each event will need to be wrapped in an auto-executing anonymous function, like so:

```
string jsCode = @"(function()
 {
  /* javascript stuff; */
  return true;
 })()";

sessionInfo.JavascriptAction.SetJsCode(jsCode);
return sessionInfo.JavascriptAction;
```

List of Global Events

Events which require a boolean or Promise of a boolean return value are labeled.

OnSaveReport

Description	Fires when a user attempts to save an open report. **WARNING** This action event replaces the default save handling. If this action event is implemented, reports will not be saved by the normal means.
Remarks	Passes the report object.

OnDuplicateReport

Description	Fires when an open report is duplicated.
Remarks	**NOTE** This action event can return a Promise. Passes the report object. If the return value resolves to *true*, then Exago will not launch the report name dialog to allow the user to select the duplicate report name. If the return value resolves to *false*, then Exago will launch the name dialog as usual.

OnEditReport v2016.3+

Description	Fires when a report is opened for editing.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true*, then Exago will not launch the corresponding Designer. If the return value resolves to *false*, then Exago will launch the Report Designer as usual.

OnSelectReport v2016.3+

Description	Fires when a report item in the folders tree is selected.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true*, then Exago will not fetch report information and update the report description box. If the return value resolves to *false*, then Exago will fetch information about the report and update the report description box as usual.

OnDeleteReport v2016.3+

Description	Fires when a report is deleted from within the folders tree.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true*, then Exago will not prompt the user to delete the report. If the return value resolves to *false*, then Exago will prompt the user to delete the report as usual.

OnRenameReport v2016.3+

Description	Fires when a report is renamed from within the folders tree.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true*, then Exago will prevent the rename text box from appearing. If the return value resolves to *false*, then Exago will show the rename text box as usual.

OnExecuteReport v2016.3+

Description	Fires when a report is executed from within the folders tree.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true*, then Exago will not execute the selected report. If the return value resolves to *false*, then Exago will execute the selected report as usual.

OnDoubleClickReport v2016.3+

Description	Fires when a report item in the folders tree is double-clicked.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true*, then Exago will not launch the report editor. If the return value resolves to *false*, then Exago will launch the report editor as usual.

OnRightClickReport v2016.3+

Description	Fires when a report item in the folders tree is right-clicked.
Remarks	**NOTE** This action event can return a Promise. Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. If the return value resolves to *true* and the selected node is a â€œcustomâ€ node created by the client integration, then Exago will hide certain built-in menu items. If the return value resolves to *false*, then Exago will show the normal context menu.

OnAfterAddDataObject

Description	Fires after a data object is added to a report.

OnBeforeRemoveDataObject

Description	Fires before a data object is removed from a report.

OnChangeParameterValue

Description	Fires when the value of a parameter in a prompt is changed

Remarks	This is commonly used in conjunction with parameter drop-downs in order to selectively enable, disable, and populate fields.

OnDashboardResize

Description	Fires when a running dashboard has its container size changed, by either the web page or the browser window
Remarks	This is commonly used to enable dashboards to re-format their contents in response to changing screen size.

OnBeforeCloseApiWindow

Description	Fires when the user clicks the cancel button in an iFrame or modal window containing a report wizard.
Remarks	This can be used to provide a JavaScript callback to close the window automatically, rather than returning to a blank page.

OnSaveReportSuccess v2017.1+

Description	Fires after an existing report is successfully savedÂ from a Wizard, Designer, or Viewer.Â Works for all report types. Also fires whenÂ an Advanced Report is created from an ExpressView, Express Report or a Dashboard Visualization. Does not fire when a report is duplicated, or when a User Report is saved as a new Advanced Report from the Report Viewer. Use **OnDuplicateReport**Â for these cases.
Remarks	ThisÂ can be useful for implementingÂ handlers to update the host application after reports are saved. This does not interfere with the default report saving behavior. Report name (path and filename) is available to the action event handler through the reportName property of the ClientInfo object.

OnClassifyUserInputURL v2018.2.6+

Description	Fires when a URL is loaded (e.g., when a URL is added to a URL tile in the Dashboard Designer or when a URL is loaded upon dashboard execution) or clicked (e.g., when an end-user clicks a URL in an executed report or report description). Only fires on loading in the Dashboard designer or on an executed Dashboard.
Remarks	This event is used for determining whether a URL is internal or external to the host domain, and can be utilized to prompt users for verification when attempting to access external URLs. The **urltoclassify** property is an available property of the ClientInfo object allowing for the specification of the URL Please see **Action Event: External URL Confirmation** for more information.

Actionable UI Elements

Advanced Reports

The following **Report Designer** toolbar items can be attached to a **Click Action Event**.

ID Description	Icon *pre-v2019.2*	Icon *v2019.2â€“v2021.1*	Icon *v2021.1+*
ReportOptionsBtn Report Settings menu			
SaveReportBtn Save Report			
DesignNewReportBtn Create New Report			not available *v2021.1+*
UndoBtn Undo			
RedoBtn Redo			
FormatCellsBtn Cell Format dialog			
FormatPaintbrushBtn Format Paintbrush			
BoldBtn Bold			
ItalicBtn Italic			
UnderlineBtn Underline			
UnderlineSelect Underline type select			
MergeCellsBtn Merge Cells			
SplitCellsBtn			not available *v2021.1+*, use MergeCellsBtn

Split Cells			always
AlignLeftBtn Horizontal alignment – left		not available *v2019.2+*	
WrapTextBtn Wrap Text			
AutoSumBtn AutoSum			
EditFormulaBtn Formula Editor			
SuppressDuplicatesBtn Hide Repeated Values			
CrossTabWizardBtn CrossTab Report Wizard			not available *v2021.1+*
LinkedReportBtn Linked Report/Drilldown			
LinkedActionBtn Linked Action Event			
GoogleMapBtn Google Maps Wizard			not available *v2021.1+*
MapBtn GeoCharts Wizard			not available *v2021.1+*
InsertBtn Insert Item menu	not available *pre-v2021.1*		

ExpressView v2019.1.12+

Beginning with *v2019.1.12+*, the following **ExpressView** toolbar items can be attached to a **Click Action Event**.

ID	Icon	Description
ExpressViewLiveDataButton	Run Stop	Start/Stop using live data

Action Event Example

The return value of the embedded JavaScript determines whether the result of clicking the button should take place.

- If the Action Event returns *True*, then toolbar item action is cancelled
- If the Action Event returns *False*, then the toolbar item action is carried out
- *ExpressViewLiveData* only: If the Action Event implementation returns a Promise, then the toolbar item action continues if and when that Promise resolves *False* (for *v2019.1.12+*).

```
string JsCode = @"(function() {
  return !confirm('Continue?');
})()";

sessionInfo.JavascriptAction.SetJsCode(JsCode);
return sessionInfo.JavascriptAction;
```

ClientInfo

This article describes the properties and methods in the **clientInfo** object and what they are commonly used for.

> **NOTE**
>
> If an element is not listed here, it is likely intended for internal use and should not be accessed.

Properties

showHourglass

Description	Set to *False* to disable the progress icon that appears when data is being saved or loaded.

includeReportData

Description	Set to *False* to prevent the client from passing the sessionInfo object to the server whenever a server callback is done.
Remarks	It may be useful to disable this to limit overhead if access to sessionInfo is not needed for a specific callback.

includeReportSaveData

Description	Set to *False* to prevent the client from passing the report save data to the server whenever a server callback is done.

Remarks	The SaveData is an additional set of data passed whenever a report is saved. This information is only passed by an **onSaveReport** global event. It may be useful to disable this to limit overhead if the save data is not needed for a specific callback.

refreshDataOnReturn

Description	Set to *False* to prevent the client Viewer from refreshing the report whenever a server callback alters report data.
Remarks	If a SaveReport callback does not alter the appearance of the report, it may be useful to disable this to limit overhead.

Utilities

Description	Access to a large variety of utilities and controls.
Remarks	Likely unnecessary in most cases. A pre-written action event provided to you by a support analyst may make use of this.

webReportsCtrl

Description	Access to the Exago Web Application user interface elements.
Remarks	Often used in order to add or remove items from the report tree sidebar. Useful for allowing Exago to handle third-party report objects.

contextObject

Description	A generic class for the object which the action event call was attached.
Remarks	The more specific context items below provide a superset of this class.

dashboard, dashboardItem, report, chartData, chartSeriesDataPoint, chartItemDataPoint, reportWidgets, categoriesCtrl, parameterListCtrl

Description	Specific classes which are set depending on the context of the call. Contain information about the object for which the action event call was attached.
Remarks	These are set contextually depending on the object of the call. For example, chartData will only be set if the action event was attached to a chart or gauge.

uiElement

Description	Provides information about the UI element called by a “click” local action event. For a list of supported elements, see Actionable UI Elements.

isSandboxMode

Description	*True* if an action event is running in a non-interactive environment, i.e. any non-html environment, where javascript interactivity is not permitted. Includes all export types: PDF, Excel, RTF, CSV.

urlToClassify

Description	String value of a URL. To be classified as “external” or “internal”. Requires user confirmation if classified as external.
Remark	This property corresponds specifically with the **OnClassifyUserInputURL** Global Action Event. For an example of a use case, please see **Action Event: External URL Confirmation**.

Methods

ServerCallback(args[])

Description	Call back to the server with any given arguments.

GetLanguageData(id)

Description	Returns the text and tooltip info from the language file for the specified UI item.

ExecuteParentFunction(func, args), GetParentFunction(func), GetParentByFunctionName(func)

Description	If the Exago Web Application is running in an iframe these are helper functions to call JavaScript functions in the parent frame.
Remarks	These functions are for convenience and safety. They are generally the same as calling Parent.FunctionName.

LoadHtmlDialog(html, options)

Description	Creates and loads an HTML dialog box. Accepts an HTML string or an html element. Accepts several options.

SetDialogValue(elementId, value)

Description	Populates the given element of a dialog with a given value.

GetDialogElementById(elementID)

Description	Finds and returns the element given by its ID.

Alert(alertText)

Description	Creates and loads an HTML alert dialog with the given text.

UpdateChart(chartWidget, chartData)

Description	Updates the given chart with the given data and re-renders it in the report.

GetDashboardReports(options)

Description	Returns all the reports on the dashboard as report objects.

GetDashboardWidgets()

Description	Returns all the widgets on the dashboard (i.e. all dashboard elements besides embedded reports).

EditReport(reportName, options)

Description	Opens the Report Designer for the given report with options. See **.NET API**.

ExecuteReport(reportName, exportType, options)

Description	Executes the given report with options. See **.NET API**.

StartNewReportWizard(reportType)

Description	Starts the Designer for the given report type.

GetClientReportObject(reportName)

Description	Returns the given report object by name.

LoadUrlToNewTab(string url)

Description	Opens a new tab with the provided URL as the contents.

Introduction to Server Events

This article applies to the **Admin Console** > **Extensions** > **Server Events** settings.

Server Event Handlers provide code that Exago can execute when certain events happen during the application runtime. This code can either come from a .NET Assembly or be written in a high-level programming language.

Create, Edit, Delete Server Event Handlers

- To **add** a new server event click **Extensions > Server Events** in the Main Menu and either:
  - click the **Add** icon at the top of the main menu
  - right-click and select **Add** from the context menu
- To **edit** an event either:
  - double click it
  - select it and click the **Edit** icon at the top of the main menu
  - right-click it and select **Edit** from the context menu
- To **delete** a server event either:
  - select it and click the **Delete** icon at the top of the main menu
  - right-click it and select **Delete** from the context menu
- To **save** changes and new server events click the **Apply** or **Okay** buttons

Server Event Handler Properties

Each **Server Event Handler** has the following properties:

Name

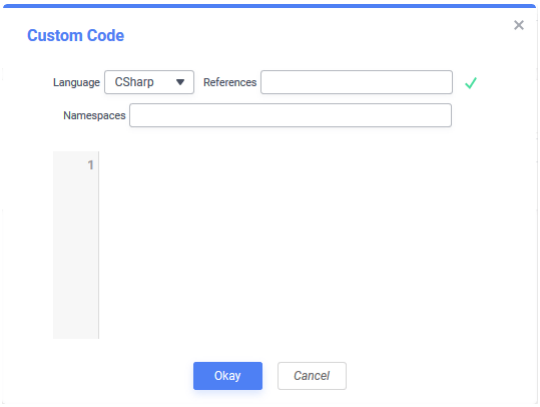
Provides a unique identifier for each Event Handler

Function

Choose between *Custom Code* or the name of a .NET Assembly. If there are no .NET Assembly Data Sources available, *Custom Code* will be the only available option.

Custom Code

To save code directly in Exago, select *Custom Code* from the first function dropdown. Clicking on the second dropdown opens the custom code dialog.



Custom Code dialog

Custom Code has three additional properties:

- **Language**: Code can be written in C#, JavaScript (in Windows only) or VB.NET.
- **References**: A semicolon-separated list of any DLLs that need to be referenced by the code. The DLLs must be present in the bin folder of the Exago Web Application, Scheduler Services, and the Web Service API if applicable. This folder can be found in the installation directory of the respective component.

  > **NOTE**
  >
  > System.dll does not need to be listed as a reference as it is already available.

- **Namespaces:** A semicolon-separated list of namespaces in the referenced DLLs or the Exago API library.
- **Code:** The code that will be executed by Exago when called. See Arguments for information on how to access the arguments for each Event. When finished, click the **Test Custom Code** ✓ icon to check if the code compiles.

.NET Assembly

There are two ways to reference a .NET Assembly method.

1. Create a .NET Assembly **Data Source**. Select the desired assembly from the first **Function** dropdown. Clicking on the second dropdown will open a list of available methods.
2. Add the .NET Assembly to the bin folder of the Exago Web Application, Scheduler Services, and the Web Service API if applicable. This folder can be found in the installation directory of the respective component. Then in the **Custom Code** window, add the assembly as a reference and invoke the method, for example:

```
return Extensions.Events.CensorEmployeeBirthYear(sessionInfo, args);
```

Global Event

Determine when this **Server Event Handler** will be called, either whenever specific event type occurs, or only when that event occurs for specific reports.

- Select a **Global Event** from the list to indicate that the **Event Handler** should be called whenever this event occurs in the application for all report executions.
- Leave **Global Event** set to *None* to indicate the **Event Handler** is meant for a specific report. See **Setting Event Handlers on Specific Reports** for more information. For example, selecting *OnReportExecuteStart* from this dropdown will cause the Event Handler to be called at the start of any Report Execution.

Arguments

Server Event Handlers can access the following information in order to inspect the session state, and utilize built-in methods:

- **sessionInfo** â€“ The **sessionInfo** object provides access to elements of Exago's current session such as parameters, filters, and the current report.
- **args** â€“ Events may have access to an array of values called ***args***. For each Event the content of the array will be different. For details on the arguments that each event provides, see **Full Description of Events**.

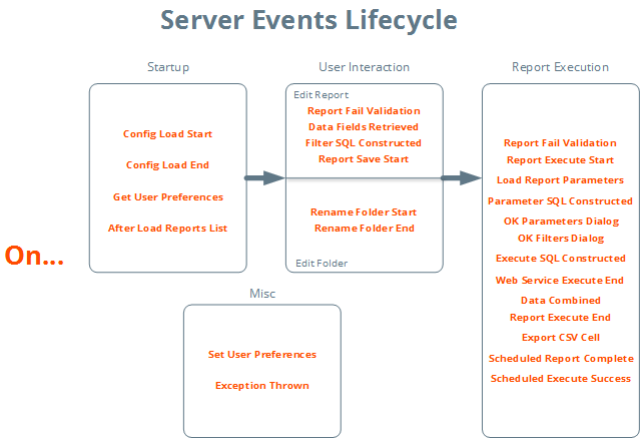
.NET Assemblies

The following are important details for using .NET Assemblies as **Server Event Handlers**.

- The Assembly DLL will be locked by Exago when it is first accessed. To replace the DLL, unlock it by restarting the IIS App Pool (and Scheduler Services, if applicable).
- The first argument of all Event Handlers is the **sessionInfo** object which can be used to access elements within the Exago session. To make use of this object the assembly must reference WebReportsApi.dll.
  If the code does not need to make use of sessionInfo then the method signature in the assembly can declare sessionInfo as an object instead of as a sessionInfo data type.
- All methods used as Event Handlers must be static.

> **NOTE**
>
> If WebReportsApi.dll or another Exago DLL is referenced by the assembly, then it must be recompiled to the current version whenever Exago is updated.

List of Server Events



Server events categorized by activity

Startup

OnConfigLoadStart â€" Occurs when the configuration of Exago is initially loaded. Expects a void return.

OnConfigLoadEnd â€" Occurs after the last API changes have been made to Exago's configuration. Expects a void return.

OnGetUserPreferences â€" Called to retrieve user preferences when entering the application and editing/executing reports.

OnAfterLoadReportsList (*v2016.3+*) â€" Occurs after reports created in Exago have been loaded in the report tree object, for the purpose of allowing additional items to be loaded in the report tree.

User Interaction

OnReportFailValidation (*v2017.1+*) â€" Occurs when a user attempts to edit or run a report which has errors.

OnDataFieldsRetrieved â€" Occurs after Data Fields are retrieved from specific Data Objects. Expects a Data Table to be returned to indicate how to display the Data Fields.

OnFilterSqlStatmentConstructed â€" Occurs before the data source is queried to populate the filter dropdown. Expects an SQL string to be returned.

OnReportSaveStart â€" Occurs at the beginning of the report save process.

OnRenameFolderStart â€" Occurs when a user attempts to rename a folder. Expects a string to be returned to indicate if execution should proceed.

OnRenameFolderEnd â€" Occurs when a folder has been renamed. Any return value will be ignored.

Report Execution

OnReportFailValidation (*v2017.1+*) â€" Occurs when a user attempts to edit or run a report which has errors.

OnScheduledReportExecuteStart (*v2018.1+*) â€" Occurs when report execution begins on a scheduler service. Expects a string to be returned to indicate if execution should proceed.

OnReportExecuteStart â€" Occurs when report execution begins. Expects a string to be returned to indicate if execution should proceed.

OnReportExecuteInit (*v2019.1.9+*) â€" Occurs slightly before **OnReportExecuteStart** before Report Viewer metadata is created. Expects a string to be returned to indicate if execution should proceed.

OnLoadReportParameters â€" Passes a list of Parameter elements that can be reordered or modified before they are sent to the client for display.

OnParameterSqlStatementConstructed â€" Occurs after a parameter dropdown object is constructed. Allows for modifying the object SQL.

OnOkParametersDialog â€" Occurs when Okay is clicked on the Parameter Execution Window. Expects a string to be returned to indicate if execution should proceed.

OnOkFiltersDialog â€" Occurs when Okay is clicked on the Filter Execution Window. Expects a string to be returned to indicate if execution should proceed.

OnExecuteSqlStatmentConstructed â€" Occurs before the data source is queried for report execution. Expects an SQL string to be returned.

OnWebServiceExecuteEnd â€" Occurs when a web service data source returns data. Expects an xml string to be returned.

OnDataCombined â€" Occurs when data is combined and initially processed. Expects a Data Table to be returned.

OnReportExecuteEnd â€" Occurs when a report execution finishes. Return value will be ignored.

OnExportCsvCell â€" Occurs prior to exporting a CSV cell for the purpose of overriding the standard export results.

OnScheduledReportComplete (*v2017.1+*) â€" Occurs when a scheduled report execution completes, regardless of whether it was successful or not.

OnScheduledReportExecuteSuccess â€" Occurs when a scheduled report is executed. Expects a boolean to be returned to indicate if the report should be sent as scheduled or intercepted.

OnAfterReportExportSuccess (*v2019.1.1+*) â€" Occurs when a report successfully exports.

Miscellaneous

OnSetUserPreferences â€" Called to save user preferences when a user specifies startup reports or saves interactive HTML changes as a user report.

OnExceptionThrown â€" Occurs when an exception is thrown in the user interface. Used to log additional information to the log file.

Global Event: OnAfterLoadReportsList

Occurs after reports created in Exago have been loaded into the Report Tree, but before the Report Tree is drawn into the user interface.

Signature

For custom code the arguments array is structured as follows:

args[] contains one object:

0. a WebReports.UI.Controls.TreeNodeCollection, a list of WebReports.UI.Controls.TreeNode objects to populate the Report Tree

> **NOTE**
>
> The **TreeNodeCollection** and **TreeNode** classes are defined in the WebReports.dll library, which is not automatically referenced in the Admin Console. To use this Server Event, add WebReports.dll as a reference, and include the WebReports.UI.Controls namespace.

For .NET Assemblies the method signature is as follows:

```
void EventHandlerName(SessionInfo sessionInfo, TreeNodeCollection nodes)
```

Expected Return

The event does not return a value.

Global Event: OnAfterReportExportSuccess

The *OnAfterReportExportSuccess* event (*v2018.2.14+, v2019.1.1+*) will be called after a report execution export file has been generated for a specified report. The server event implementation may load the export file from disk using the supplied file path, *exportFilePath*, modify the file, then save the changes back to disk. If the server event implementation saves changes to the export file, the file must be saved with the same filename that is passed in as the *exportFilePath*.

> **CAUTION**
>
> If the file is saved to a different path, modifications will not be honored by Exago.

This server event will be called for all report execution export files generated, including those generated via:

- Execution in the web app process (remote execution disabled)
- Remote execution
- Scheduled execution (including those using the **Scheduler Queue**)
- Batch scheduled executions
- GetExecute()

It is compatible with HTML, PDF, RTF, Excel, and CSV export types.

Signature

For custom code the args array is structured as follows:

args[] contains two arguments: args[0] should contain a string representing the file path of the exported report, and args[1] should contain a wrExportType object representing the export type of the report.

For .Net Assemblies the method signature is as follows:

```
void EventHandlerName(SessionInfo sessionInfo, string exportFilePath, wrExportType exportType)
```

Expected Return

The return value of this event is ignored.

Global Event: OnConfigLoadEnd

The **OnConfigLoadEnd** Event occurs after all API changes are made and the host application container is redirected to Exago. If entering Exago directly this event occurs immediately after **OnConfigLoadStart**. If the API is being used but the host application does not redirect to Exago (such as using the direct Report.GetExecuteData method) the event can manually be called using the public method *Api.SetupData.FireOnConfigLoadEndEvent()*.

Similar to the OnConfigLoadStart event, this event can also be used to change configuration information on-the-fly via the sessionInfo object. However making these changes after the API calls can provide extra convience. For example if the host application is using the Web Service API it can set a single parameter value using the WebService and then based on that parameter make further configuration changes within this event. This provides better performance, security and a reduction of http requests.

Signature

For custom code the args array is structured as follows:

args[] is empty.

For .Net Assemblies the method signature is as follows:

```
void EventHandlerName(SessionInfo sessionInfo)
```

Expected Return

The OnConfigLoadEnd Event has a void return value.

Global Event: OnConfigLoadStart

The **OnConfigLoadStart** Event occurs after the configuration file is loaded. This may happen in the API when the API object is initialized or in Exago when entering the application directly. This event can be used to change any configuration information on-the-fly via the SessionInfo object, such as decrypting database connection strings.

Signature

For custom code the args array is structured as follows:

args[] is empty.

For .Net Assmblies the method signature is as follows:

```
void EventHandlerName(SessionInfo sessionInfo)
```

Expected Return

The OnConfigLoadStart Event has a void return value.

Global Event: OnDataCombined

The **OnDataCombined** Event allows the inspection and/or modification of the raw data set after retrieval from the Data Sources and initial combining within Exago. A common use of this event is to modify or blank sensitive data fields in a Report depending on the authorizations available to the user executing the report.

In *v2018.1* this event was extended to provide access to the active execution object, as well as the ability to modify report execution schema via the **SessionInfo** object. After modifying the execution schema, the method Report.ReprocessAfterAlteration() must be called in order to process the changes.

Signature v2018.1+

For custom code the args array is structured as follows:

args[] contains a single DataTable of the combined data in position zero and the ActiveLocalExecution object in position one.

For .NET Assemblies the method signature is as follows:

```
DataTable EventHandlerName(SessionInfo sessionInfo, DataTable combinedData, ActiveLocalExecution execution)
```

> **NOTE**

The **ActiveLocalExecution** object is defined in the WebReports.Api.Execute namespace.

Expected Return

The OnDataCombined Event expects a DataTable to be returned. The schema of the DataTable must match that of combinedData.

Notes

In the DataTable, if a Data Object has an **Id** then that will be used as the column names, otherwise the database name will be used. Data Fields will always use their database names despite any **Column Metadata**.

Global Event: OnDataFieldsRetrieved

The **OnDataFieldsRetrieved** Event occurs after Data Fields are retrieved for a specific Data Object. This event is commonly used to change the order Data Fields are displayed in the Data Menu of the Report Designer.

Signature

For custom code the arguments array is structured as follows:

args[] contains three objects:

0. a System.Data.DataTable containing the names and metadata of the Data Fields
1. the Data Object as a WebReports.Api.Reports.Entity object
2. a reference to a WebReports.Api.Data.DataObjectBase object which calls the event.

For .NET Assemblies the method signature is as follows:

```
DataTable EventHandlerName(SessionInfo sessionInfo, DataTable originalDataFields, Entity dataObject, DataObjectBase eventCaller)
```

Notes

The DataTable being being passed to the event in the first argument has already applied Column Metadata.

Expected Return

Expects a System.Data.DataTable return value, which represents the modified data.

Global Event: OnExceptionThrown

OnExcpetionThrown is called when an application exception is thrown in the user interface. Generally used to pass additional or different information to the log file.

Signature

For custom code, the args[] array is structured as follows:

0. the System.Exception object
1. the WebReports.Api.Common.Logger object â€" provides write access to the log file
2. string of JSON data with the AJAX arguments which can provide more information in the log regarding the users requests which resulted in the exception being thrown.

For .NET Assemblies the method signature is as follows:

```
bool EventHandlerName(SessionInfo sessionInfo, Exception exception, Logger logger, string jsonArgs)
```

Expected Return

The **OnExceptionThrown** event expects a boolean return value. When *true*, logging of the exception is stopped.

Global Event: OnExecuteSqlStatementConstructed

The **OnExecuteSqlStatementConstructed** Event occurs just before SQL is sent to the Data Source to retrieve data for report execution. This Event could be used to inspect, log or modify the SQL that is being used for report execution.

Signature

For custom code the arguments array is structured as follows:

args[] contains:

0. a string representing the execution SQL
1. a SqlObject class wrapping the execution SQL string in position one.

For .NET Assemblies the method signature is as follows:

NOTE

The WebReports.Api.Data namespace needs to be included.

```
string EventHandlerName(SessionInfo sessionInfo, string executionSql, SqlObject sqlObject)
```

Expected Return

The **OnExecuteSqlStatementConstructed** Event expects either a SQL string return, to override the default execution SQL, or null, in which case the execution will continue as normal.

Global Event: OnExportCsvCell

Called prior to exporting a CSV cell for the purpose of overriding the standard export results. If the server event is defined, it is called for every visible cell in a report on CSV export. Allows for customizing cell contents on a cell-by-cell basis.

Signature

For custom code the args array is structured as follows:

args[] is contains two objects: The first argument args[0] is the Cell object in its entirety, which contains a variety of cell attribute information including row and col indices, cell type, format, convenience flags, linked report data, etc. The second argument args[1] provides the raw text of the cell as a string.

For .Net Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, Cell cell, string rawCellText)
```

Expected Return

The event expects a string or null return value. If null is returned, the event will interpret this as a flag indicating that no custom CSV data is being provided, and to return the report's CSV output. If a string is returned (including an empty string), the cell will output the string as provided, overriding the report data for that cell.

Global Event: OnFilterSqlStatementConstructed

The **OnFilterSqlStatementConstructed** Event occurs just before SQL is sent to the Data Source to retrieve data to populate a filter dropdown. This Event could be used to inspect, log or modify the SQL that is being used to populate the dropdown.

Signature

For custom code the arguments array is structured as follows:

args[] contains:

0. a string representing the filter SQL

For .NET Assemblies the method signature is as follows:

> **NOTE**
>
> Include the WebReports.Api.Data namespace.

```
string EventHandlerName(SessionInfo sessionInfo, string filtersSql, SqlObject sqlObject)
```

Expected Return

The **OnFilterSqlStatementConstructed** Event expects a string value to be returned.

Note

This Event will provide the SQL for the Filter Dropdown Object if that feature is being utilized. See **Data Objects** for more information on Filter Dropdown Objects.

Global Event: OnGetUserPreferences

The **OnGetUserPreferences** Event is used to retrieve user preferences when entering the application and when editing/executing reports.

Signature

For custom code the args array is structured as follows:

args[] contains one object, a string with the user preferenceâ€™s id.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, string id)
```

Expected Return

Expects a string return value, which represents the user preferenceâ€™s value.

Notes

The event will only be called if the â€˜User Preference Storage Methodâ€™ is set to Server Events in the **User Settings**.

Global Event: OnLoadReportParameters

The OnLoadReportParameters event passes a list of Parameter elements that can be reordered or modified before they are sent to the client for display. Called when report parameters are loaded, but before any processing has occurred. By default, parameters are sorted alphabetically by name.

Signature

For custom code the args array is structured as follows:

args[] is contains one object, a list of Parameter elements.

For .NET Assemblies the method signature is as follows:

```
void EventHandlerName(SessionInfo sessionInfo, List<Parameter> parameters)
```

Expected Return

The event has a void return value.

Global Event: OnOkFiltersDialog

The **OnOkFiltersDialog** Event occurs when a user clicks on the Ok button in the Filter Execution Window. This window only displays if prompt for value was checked for a filter. This Event could be used to see what filters are being used on the report and/or assure that a filter exists.

Signature

For custom code the args array is structured as follows:

args[] is empty.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo)
```

Expected Return

The OnOkFiltersDialog Event expects a string to be returned. Based on the returned string there are three possible results.

- **Null / Whitespace** â€” If the string is null or whitespace then the report execution will continue as expected.
- **LanguageId** â€” If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user and the report execution will terminate. For more information see **Multi-Language Support**.
- **Other** â€” If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user and the report execution will terminate.

Notes

The filters of the report being executed can be accessed through the sessionInfo object by using sessionInfo.Report.ExecFilters.

Global Event: OnOkParametersDialog

The OnOkParametersDialog Event occurs when a user clicks on the Ok button of the Parameter Prompt Window. The window will only displays if the report has a non-hidden parameter with a prompt text. This Event could be used to see what values the user is setting for each prompting parameter.

Signature

For custom code the args array is structured as follows:

args[] is empty.

For .NET Assmblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo)
```

Expected Return

The OnOkParametersDialog Event expects a string to be returned. Based on the returned string there are three possible results.

- **Null / Whitespace** â€” If the string is null or whitespace then the report execution will continue as expected.
- **LanguageId** â€” If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user. For more information see **Multi-Language Support**.
- **Other** â€” If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user.

Notes

This Event cannot override the value of Parameters for the report execution.

The Parameters of the report being executed can be accessed through the sessionInfo object by using sessionInfo.Report.

Global Event: OnParameterSqlStatementConstructed

Called after a parameter dropdown object is constructed. Allows for modifying the object SQL.

Signature

For custom code the args array is structured as follows:

args[] contains one object, the SQL string which is passed to the server to construct the parameter dropdown.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, string sqlStatement)
```

Expected Return

The event expects a string return value: The (modified) SQL statement to be passed to the server to construct the parameter dropdown.

Global Event: OnRenameFolderEnd

The OnRenameFolderEnd Event occurs after user has renamed a folder.

Signature

For custom code the args array is structured as follows:

args[] is contains two strings, the first represents the fully qualified old folder name, the second is the new folder name.

For .Net Assemblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo, string currentFolderName, string newFolderName)

Expected Return

Anything can be returned to the OnRenameFolderEnd Event. Any return value will be ignored.

Global Event: OnRenameFolderStart

The **OnRenameFolderStart** Event occurs when a user attempts to rename a folder. This event happens before the folder is renamed permitting you to stop the renaming if desired.

Signature

For custom code the args array is structured as follows:

args[] contains two strings:

1. the fully qualified current folder name
2. the new folder name.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, string currentFolderName,
 string newFolderName)
```

Expected Return

The OnRenameFolderStart Event expects a string to be returned. Based on the returned string there are three possible results.

- **Null / Whitespace** â€" If the string is null or whitespace then the report execution will continue as expected.
- **LanguageId** â€" If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user. For more information see Multi-Language Support.
- **Other** â€" If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user.

Global Event: OnReportExecuteEnd

The OnReportExecuteEnd Event occurs at the end of the Report Execution process. This Event could be used to track which report executions return data.

This event will be called for all report execution types, including those via:

- Execution in the web app process (remote execution disabled)
- Remote execution
- Scheduled execution (including those using the **Scheduler Queue**)
- Batch scheduled executions
- GetExecute()

Signature

For custom code the args array is structured as follows:

args[] contains a single Boolean indicating if Data qualified (True), or not (False).

For .Net Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, bool DataQualified)
```

Expected Return

Anything can be returned to the OnReportExecuteEnd Event. Any return value will be ignored.

Global Event: OnReportExecuteInit

Available in *v2019.1.9+*

The **OnReportExecuteInit Event** occurs at the beginning of the report execution process similar to **OnReportExecuteStart**, but triggers before the Report Viewer's rendering information metadata file has been created and sent to the client. This is in contrast to **OnReportExecuteStart** which triggers after the metadata is created.

This Event could be used to make modifications to column labels or column widths based on schema information, for example, prior to the generation of the metadata file being created.

This event will be called for all report execution types, including those via:

- Execution in the web app process (remote execution disabled)
- Remote execution
- Scheduled execution (including those using the **Scheduler Queue**)
- Batch scheduled executions
- GetExecute()

Signature

For custom code the args[] array is structured as follows:
args[] is empty

For .NET Assemblies the method signature is as follows:

```
string OnReportExecuteInit(SessionInfo sessionInfo)
```

Expected Return

The **OnReportExecuteInit** Event expects a string to be returned. Based on the return string there are three possible results.

- **null**: If the string is null then the report execution will continue as normal.
- **LanguageId**: If the string matches the ID of any element in the language files then the string of that language element will be displayed as a message to the user and the report execution will terminate. For more information refer to the article on **Multi-Language Support**.
- **Other**: If the string does not match the ID of any element in the language files then the returned value will be displayed as a message to the user and the report execution will terminate.

Notes

The report being executed can be accessed through the sessionInfo object by using sessionInfo.Report

Global Event: OnReportExecuteStart

The **OnReportExecuteStart** Event occurs at the beginning of the Report Execution process. This Event could be used to check and modify properties of a report and log or stop execution.

This event will be called for all report execution types, including those via:

- Execution in the web app process (remote execution disabled)
- Remote execution
- Scheduled execution (including those using the **Scheduler Queue**)
- Batch scheduled executions
- GetExecute()

Signature

For custom code the args array is structured as follows:

args[] is empty.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo)
```

Expected Return

The OnReportExecuteStart Event expects a string to be returned. Based on the return string there are three possible results.

- **Null / Whitespace** â€" If the string is null or whitespace then the report execution will continue as expected.
- **LanguageId** â€" If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user and the report execution will terminate. For more information see **Multi-Language Support**.
- **Other** â€" If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user and the report execution will terminate.

Notes

The report being executed can be accessed through the sessionInfo object by using sessionInfo.Report.

Global Event: OnReportFailValidation

This server event is available in version 2017.1 and later.

The OnReportFailValidation server event occurs when a user attempts to edit or run a report which has errors. When a user edits a report which fails validation, a message dialog shows the user the list of errors with the report. This server event can be used to customize the error messages.

Signature

For custom code the args array is structured as follows:

args[] contains one argument, the list of ValidationErrors. A ValidationError is an object containing an enum for the error type, and two strings for more detailed message information about the error. The message strings can be a text string or a language file id element.

For .Net Assemblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo, List<ValidationErrors> errorList)

Expected Return

The OnReportFailValidation server event expects a List<ValidationErrors> return object.

The list of error types is as follows:

DataObjectNotFound,
SortDataFieldNotFound,
FilterDataFieldNotFound,
LinkedDataFieldNotFound,
MinMaxFilterDataFieldNotFound,
JoinDataObjectNotFound,
JoinNotFound,
JoinDataFieldNotFound,
ChartDataFieldNotFound,
MapDataFieldNotFound,
CellDataFieldNotFound,
RowGroupNameNotFound,
RowGroupFormulaNameNotFound,
ChartCellIdNotFound,
MapCellIdNotFound,
MergedCellsAcrossSections,
CrossTabIdNotFound,
CrossTabCellIdNotFound,
ColumnSortByFieldNotFound,
ChildReportNotFound,
ExpressViewColumnMissingEntity,
ExpressViewColumnMissingField,
ExpressViewGroupMissingField,
ExpressViewFilterMissingField,
ExpressViewChartMissingField,
None

Global Event: OnReportSaveStart

The **OnReportSaveStart** Event occurs at the beginning of the save process for Advanced Reports, CrossTabs, Dashboards, Chained Reports and ExpressViews.

Support for Dashboards and Chained Reports was added in *v2019.1.26*, *v2019.2.12* and *v2020.1.1*.

Signature

For custom code the arguments array is structured as follows:

args[] is empty.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo)
```

Notes

The report being saved can be accessed through the sessionInfo object by using sessionInfo.Report for Advanced Reports, CrossTabs and ExpressViews. For Dashboards and Chained Reports, use sessionInfo.ReportObject.

Expected Return

Expects a string return value. If null, the report will continue processing; if not null, represents the language file ID, or string text to display to the user.

Global Event: OnScheduledReportComplete

This server event is available in version 2017.1 and later.

The OnScheduledReportComplete server event occurs when a scheduled report execution completes, regardless of whether it was successful or not. It precedes creation of the output file. The schedule job object is passed in the args array so the job can be examined for potential errors.

Signature

For custom code the args array is structured as follows:

args[] contains one argument, a SchedulerJob object representing the completed scheduled execution.

For .Net Assemblies the method signature is as follows:

```
bool EventHandlerName(SessionInfo sessionInfo, SchedulerJob job)
```

Expected Return

The OnScheduledReportComplete server event expects a boolean return value to indicate whether to create the report output for the job. Returns **TRUE** if the report output should be created, returns **FALSE** if the report output should not be created.

Global Event: OnScheduledReportExecuteStart

This server event is available in *v2018.1+*. The **OnScheduledReportExecuteStart** Event occurs at the beginning of the Report Execution process when run by a Scheduler Service. Occurs for any job run by a scheduler service, including scheduled executions, remote executions, and execution cache rebuilds. This Event could be used to check and modify properties of a report and log or stop execution.

Signature

For custom code the args array is structured as follows:

args[] contains a **SchedulerJob** object at position 0, which can be used to access and modify properties of the active execution and schedule, including the report object, email and schedule information.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, SchedulerJob job)
```

Expected Return

The **OnScheduledReportExecuteStart** Event expects a string to be returned. Based on the string returned there are three possible results:

- **Null / Whitespace** â€“ If the string is null or whitespace then the report execution will continue as expected.
- **LanguageId** â€“ If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user and the report execution will terminate. For more information see **Multi-Language Support**.
- **Other â€“** If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user and the report execution will terminate.

Notes

The report being executed can be accessed through the ScheduleJob object by using job.Report. The schedule information can be accessed through job.ScheduleInfo.

Global Event: OnScheduledReportExecuteSuccess

The **OnScheduledReportExecuteSuccess** Event occurs when a scheduled report execution is finished. This event can be used to create an audit log of scheduled reports.

Signature

For custom code the args array is structured as follows:

args[] is empty.

For .NET Assemblies the method signature is as follows:

```
bool EventHandlerName(SessionInfo sessionInfo)
```

Expected Return

The **OnScheduledReportExecuteSuccess** Event expects a Boolean to be returned. Returning **True** will prevent the scheduled report from being sent. Returning **False** will allow the report schedule to proceed with processing.

> **NOTE**
>
> This server event is called for remote execution of reports. However, the return value will be ignored as there is no email to be prevented.

Global Event: OnSetUserPreferences

The **OnSetUserPreferences** Event is used to store user preferences when setting startup reports, saving interactive Report Viewer

modifications as user reports or color chooser user colors. The event will only be called if the **Admin Console** > **General** > **User Settings** > **User Preference Storage Method** is set to *Server Events*.

Signature

For custom code the args array is structured as follows:

args[] contains two objects:

0. a string with the user preferenceâ€™s id
1. a string with the user preferenceâ€™s value

For .NET Assemblies the method signature is as follows:

```
void EventHandlerName(SessionInfo sessionInfo, string id, string value)
```

Expected Return

The event has no return value.

Additional Resources

- Admin Support Lab â€" Customizing Color Picker

Global Event: OnWebServiceExecuteEnd

The **OnWebServiceExecuteEnd** Event occurs when data is returned from a Web Service Data Source. This Event could be used to decompress or decrypt data being returned from a Web Service Data Source.

Signature

For custom code the args array is structured as follows:

Args[] contains a single string of the data coming from the Web Service in position zero.

For .NET Assemblies the method signature is as follows:

```
string EventHandlerName(SessionInfo sessionInfo, string webServiceXml)
```

Expected Return

The OnWebServiceExecuteEnd Event expects a string to be returned.

> **NOTE**
>
> This Event only occurs when the **callType** Parameter has the value *1*.

Introduction to the .NET API

> **NOTE**
>
> This version of the .NET API documentation is deprecated. A new version can be found at /support/administrators/net-api-docs/.

The Exago interface consists of two primary elements:

- the User Interface, and
- the API

The User Interface is built directly onto the .NET API. This means that .NET applications can interface directly with Exago. Non-.NET applications use the **REST Web Service API.**

REST – Introduction

The Exago API provides programmatic access to create session configurations based on user credentials. The REST API is an alternative to the .NET API for applications not built in .NET environments. REST web services provide access to API functions for any client capable of passing HTTP requests over a network.

Installing REST

Install the REST Web Service API following the steps in REST â€" Getting Started.

The API

This is the documentation for the Exago REST API. This information will always reflect the latest available Exago version. Changes are made periodically to the API. For details, please refer to **REST Updates**.

The API is a JSON API. For more information, see **Using JSON**.

Endpoints are documented with the HTTP request type and a URI sub-path:

```
POST /rest/Sessions
```

Prepend the URL path to your web service to get the full endpoint URL:

```
http://{myapp}/{exago web service}/rest/Sessions
```

Curly braces, {}, are values that you must supply:

```
GET /rest/Roles/{Id}
```

Usually the value to supply is found in JSON given by a request to the base endpoint. For example, to retrieve a particular Role by its “Id”, first run GET /rest/roles to see the roles available to you:

```
[
 {
  "Id": "Admin"
 },
 {
  "Id": "User"
 }
]
```

Then to see detailed data on a Role, run GET /rest/roles/{Id} where {Id} is the value of its “Id” property (case sensitive):

```
GET /rest/Roles/Admin
```

Some methods may accept optional URL parameters. Append any url parameters to the end of the endpoint URL using the following format:

```
/rest/{endpoint}?{param1}={value1}&{param2}={value2}&{param3}={value3}
```

For all endpoints except /rest/sessions, you must append the session Id as a url parameter to the end of the URL:

```
VERB /rest/{endpoint}?sid={sid}
```

For more information, see **Starting a REST Session**.

Some examples in this documentation use **cURL** (“curl”), which is a command-line tool for transferring data using various protocols. Curl is compatible with Windows or Linux, and can be downloaded from **https://curl.haxx.se/**. If curl is installed on your machine, you can copy-and-paste the examples into a command line, replacing any item in {braces} with your local variables, in order to test them out for yourself.

Long examples may be broken into multiple lines in order to improve readability. The caret symbol (^) tells the command line to ignore the following linebreak. On Linux, replace the caret with a backslash ().

```
curl http://{webservice}/rest/Settings?sid={sid} ^
-H "Accept: application/json" ^
-H "Content-Type: application/json" ^
-H "Authorization: Basic Og==" ^
-X PATCH ^
-d "{'ShowExpressReports':false}"
```

For testing, we recommend a graphical application such asÂ **Advanced Rest Client**Â or Postman.

Authentication

The REST API requires authorization to be accessed. To make an authorized request, the authorization header must be supplied. There are two different authorization methods depending on your needs. Both rely on the **Admin Console** > **General** > **Other Settings** > **User ID** and **Admin Console** > **General** > **Other Settings** > **REST Key**.

Basic Authorization

When using basic authorization, the authorization header is constructed as follows:

1. The User ID and REST Key are combined into a string *User ID:REST Key*.
2. The resulting string literal is encoded using Base64.
3. “Basic” and a space are placed before the encoded string.

For example, if the User ID is “Brian” and the REST Key is “open sesame” then the authorization header would be constructed as follows:

1. Combine User ID and REST Key into a string

   ```
   Brian:open sesame
   ```

2. Encode the string using Base 64

   ```
   QWxhZGRpbjpvcGVuIHN1c2FtZQ==
   ```

3. Append “Basic ” to the front

   ```
   Basic QWxhZGRpbjpvcGVuIHN1c2FtZQ==
   ```

The auth key is sent in clear text with each request. If this is a concern, the REST API should be deployed in an SSL environment or the more secure ExagoKey authorization should be used.

A configuration with a blank User ID and REST Key can be accessed using the following authorization header:

```
Authorization: Basic Og==
```

ExagoKey Authorization

ExagoKey authorization uses the HMAC-SHA256 algorithm for authorization. When using ExagoKey authorization, the authorization header is constructed as follows:

1. The string to sign is UTF-8 encoded, then signed with the UTF-8 encoded REST key using the HMAC-SHA256 algorithm.
2. The resulting signature is then encoded using Base64.
3. The User ID and a colon is put before the encoded signature.
4. “ExagoKey” and a space are placed before the encoded string literal.

For example, if the User ID is “Brian” and the REST Key is “open sesame” then depending on the request the authorization header might be something like:

```
Authorization: ExagoKey Brian:6HZE5tCWjsbJY+VXQg3UzXlK/jeoGhbm25YDXiHWdE=
```

Using ExagoKey does not send the password with each request, making it more secure than Basic Authorization. To ensure greater security the REST API should be deployed in an SSL environment.

ExagoKey String

The ExagoKey string that is to be signed is constructed using the following information from the request, in the following order, with “n” after each item (including the last one).

1. The HTTP Method, must be in uppercase.
2. The absolute request path, up to but not including the query string if one should exist. For example, if the request is to “http://myserver.com/exago/rest/Sessions?config=myconfig” the absolute request path would be “/exago/rest/Sessions”.
3. The contents of the Content-Length header.
4. The contents of the Content-Type header, or a string of zero length if no header exists.
5. The contents of the Content-MD5 header, or a string of zero length if no header exists.
6. The session ID, or a string of zero length if no session ID exists.
7. The contents of the X-Exago-Date header, or the contents of the Date header if the X-Exago-Date header does not exist, or a string of zero length if neither header exists.

> **NOTE**
>
> If a date is supplied, the REST API will reject any request that is older than 15 minutes from the supplied date. The date supplied is in GMT (UTC).

Request Format

The REST API uses JSON. It will not accept data in any other format. In the case of an error, it may return plain text, but this is not session data. You must set the following headers on all requests:

- “Content-Type: application/json”
- “Accept: application/json”
- “Authorization: {type} {authstring}”

Most of the examples in this documentation omit the headers in order to improve readability; however they are required for all requests.

Response Format

Successful requests return HTTP status codes in the 200 range. When you create a resource with POST, the API returns the resource in the response body:

```
Status: 201 Created
Location: /{webservice}/rest/Entities/Employees

{
  "Id":          "Employees",
  "Name":        "Employees",
  "Schema":      "dbo",
  "CategoryName": "",
  "DataName":    "Employees",
  "DataSourceId": "0",
  "DataType":    "Table",
  ...
}
```

You may get a plain text response in the case of errors or bad requests. The content will not contain any session data.

Responses have one of the following status codes:

200

The request was completed successfully. The document in the body, if any, is a representation of some resource. This code is usually returned for a successful GET request.

201

The request was completed successfully. A new resource has been created at the URL specified in the Location header of the response. The document in the body, if any, is a representation of the resource created. This code is usually returned for a successful POST request.

204

The request was completed successfully. There is no content in the body. This code is usually returned for successful PATCH, PUT, and DELETE requests.

400

The request was bad on the client side. The document in the body, if any, is error data describing the problem. This is usually the result of using an invalid method type.

401

The request wasnâ€™t authorized to access the resource. The document in the body, if any, is error data describing the problem.

404

The requested resource was not found. The document in the body, if any, is error data describing the problem. Often this is the result of a malformed URL request string.

409

The request caused a conflict between two resources. The document in the body, if any, is error data describing the problem.

500

There was a problem on the server side. The document in the body, if any, is error data describing the problem. Often this is the result of a malformed JSON request package.

Request Data

The API returns and accepts JSON values, which can be strings, numbers, objects, arrays, **true**, **false**, or **null**. See **Using JSON** for more information.

Each endpoint uses a unique JSON object for input and output of variables. The JSON is documented with each resource in a table:

Name	Type	Writable	Description

Each row represents a property. The **Name** field is the name of the property. The **Type** field indicates what type of data it accepts.

The **Writable** field indicates whether this property can be written:

- no – this property is read-only
- yes – this property can be written
- yes (“value”) – this property can be written; its default value is in parentheses
- required – this property must be written; it cannot be null
- required-create – this property must be written for all POST calls; it is read-only for all other calls

The **Description** field is any relevant information about the property.

Id Values

Most resources are identified by an “Id” property, which is an integer or string. The Id must be unique among the set of resources. If you are permitted write-access to the Id, avoid conflicting names.

Constant or Enum Values

Some properties can only accept one of a set of string or integer values (or sometimes null).

Properties which accept enumerated values are indicated as having an “enum” type. The description field in its JSON representation links to the list of possible values, which can usually be represented by their string or equivalent integer value. For example:

Name	Type	Writable	Description
DataType	enum	yes	**Parameter Type**

Properties which accept constant string values are indicated as having an “const” type. The description field in its JSON representation links to the list of possible values. For example:

Name	Type	Writable	Description
JoinType	const	yes	**Join Type**

This Documentation

This documentation is broken down by section according to the use cases for different API elements. Each section represents a single endpoint or set of related endpoints and use cases. To find out how to do something using the REST API, first browse to the relevant section for the element you want to use. Then check the table of contents for what actions you want to take; Or read from top to bottom to gain a fuller understanding of the resource.

Check out **Getting Started with REST** to get started!

See **List of REST Endpoints** for the available endpoints and their usages.

Additional Resources

- Admin Support Lab â€” REST Web Service API Setup & Security
- Admin Support Lab – Batch REST API
- Admin Support Lab – .NET and REST APIs

Using JSON

The Exago REST API is JSON-based.Data sent to the API methods is formatted in JSON, and the methods return JSON formatted response objects. In order to use the REST API you need to convert your data to, and from JSON.

What is JSON?

JSON is a data-interchange format that is designed to be text-based, and easy to read and parse. JSON objects are formatted using the following rules:

1. Objects contain a collection of properties in “key”: “value” format. Braces {} enclose objects.
2. Keys are strings, which are enclosed by single- (‘ ‘) or double-quotes (” “). Each key has a value, indicated by a colon (:). Properties are separated by commas (,).
3. Properties can accept sets of values using arrays, enclosed by brackets [ ]. Values are separated by commas (,).
4. Values can be strings, numbers, arrays, other JSON objects, true, false, or null.
5. The order of properties within an object doesn’t matter.

Example

```
{
 "users": [
  {
   "id": "358",
   "username": "alex224",
   "admin": true,
   ...
  },
  ...
 ]
}
```

> **NOTE**
>
> Ellipses ... indicate that one or more properties or objects have been omitted for clarity.

Using JSON with Code

JSON data usually has to be converted into a format that your code can understand. Most modern programming languages have JSON compatible libraries, if they are not built into the language.

To convert JSON strings into objects:

JavaScript (see JSON.parse() at MDN)

```
var json_obj = JSON.parse('{"user":{"id":"358"}}');
```

Python 3 (see json at Pydoc)

```
json_obj = json.loads('{"user":{"id":"358"}}')
```

JSON Object Documentation

Although JSON objects can be formatted in a variety of ways, each endpoint requires an object with a specific set of properties. The required format can be viewed in the documentation for each endpoint. Some properties are mandatory; some are read-only; some are create-only, which means that they are required for POST calls, but read-only for other call types; and some are optional. Accessing certain resources in an un-authorized state may only return a subset of data; in general we recommend that all REST calls be authorized.

For example, the documentation for the JSON object above might read like the following:

Name	Type	Writeable	Description
users	array of **User**	yes	List of users belonging to the session

User JSON

Name	Type	Writeable	Description
id	integer	no	This user’s unique Id
username	string	required	This user’s login key
admin	boolean	yes (false)	Whether this user has admin privileges

> **NOTE**
>
> Read (-only) permissions indicate that the property cannot be edited with PATCH or PUT. This does not indicate whether a DELETE call can be used on the object.

Using the API with cURL

Throughout the REST documentation there are examples which use cURL, a free command-line tool for using various protocols. Download cURL at: https://curl.haxx.se/download.html. If you're using Google Chrome, the Advanced REST Client browser extension is an excellent alternative. We recommend using either program to experiment with the API and test your method calls.

> **DISCLAIMER**
>
> We cannot provide support for any third-party tools such as cURL or Advanced REST Client.

API calls using cURL require the following at a bare minimum: Three headers, Accept: application/json, Content-Type: application/json, and your authorization header; The JSON data, which should be empty if the endpoint expects no data; And the "verb" e.g. POST, GET, etc.

All cURL calls are formatted like the following:

```
curl http://{webservice}/rest/{endpoint}?sid={sid}&{param1}={value1}&{param2}={value2} ^
 -d "{json}" ^
 -H "Accept: application/json" ^
 -H "Content-Type: application/json" ^
 -H "Authorization: {type} {authstring}" ^
 -X{verb}
```

> **NOTE**
>
> Replace text in {braces} with the applicable data for your environment, method, and JSON. End-of-line carets (^) terminate each line to improve readability, but are not necessary.

For example, a new session call is formatted like the following:

```
curl http://{webservice}/rest/Sessions ^
 -d "" ^
 -H "Accept: application/json" ^
 -H "Content-Type: application/json" ^
 -H "Authorization: Basic Og==" ^
 -XPOST
```

For an un-authenticated installation, you could copy and paste this example into a command line, replacing {webservice path} with the path to your web service, in order to test whether REST is working.

If it succeeds, it should return data similar to the following:

```
{"AppUrl":"ExagoHome.aspx?d={appUrl}","Id":"{sid}","Page":"ExagoHome","ApiAction":"Default","ExportType":null,"ShowTabs":true,"ShowErrorDetail":true,"ReportSettings":{"Id":null,"Rep
```

Note that all the data was returned on a single line. For improved readability, copy and paste this into a "pretty-printer" (there are many free online solutions).

```
{
  "AppUrl": "ExagoHome.aspx?d={appUrl}",
  "Id": "{sid}",
  "Page": "ExagoHome",
  "ApiAction": "Default",
  "ExportType": null,
  "ShowTabs": true,
  "ShowErrorDetail": true,
  "ReportSettings": {
    "Id": null,
    "ReportPath": null,
    "SortsResource": null,
    "FilterItems": null
  }
}
```

To pass JSON with cURL, either insert it inline:

```
curl http://{webservice}/rest/Sessions ^
 -d "{'ReportSettings':{'ReportPath':'TestTestReport'}}" ^
 -H "Accept: application/json" ^
 -H "Content-Type: application/json" ^
 -H "Authorization: Basic Og==" ^
 -XPOST
```

Or reference a text file containing the JSON object:

```
curl http://{webservice}/rest/Sessions ^
 -d @json.txt ^
 -H "Accept: application/json" ^
 -H "Content-Type: application/json" ^
 -H "Authorization: Basic Og==" ^
 -XPOST
```

json.txt

```
"{'ReportSettings':{'ReportPath':'TestTestReport'}}"
```

List of REST Endpoints

The following REST Web Service API endpoint paths are available. All calls require the following headers:

```
Content-Type: application/json
Accept: application/json
Authorization: {type} {authstring}
```

For more information, review REST â€" Introduction

/rest/Batch v2018.1.1+

- POST

/rest/Sessions

- POST
- GET

/rest/Sessions/{sid}

- GET
- PATCH
- DELETE

All of the following calls require the Session ID URL parameter: sid={sid}

Example

```
GET /rest/Settings?sid={sid}
```

/rest/DataSources

- GET

/rest/DataSources/{Id}

- POST
- GET
- PATCH

/rest/Entities

- POST
- GET

/rest/Entities/{Id}

- GET
- PATCH
- DELETE

/rest/Entities/{Id}/Fields

- GET

/rest/Entities/{Id}/Fields/{Field Id}

- GET
- PATCH
- POST v2019.1.13+

/rest/Folders/{Name}

- POST
- DELETE

/rest/Folders/Rename

- POST

/rest/Functions

- POST
- GET

/rest/Functions/{Id}

- GET
- PATCH
- DELETE

/rest/Joins

- POST
- GET

/rest/Joins/{Id}

- GET
- PATCH
- DELETE

/rest/Parameters

- POST
- GET

/rest/Parameters/{Id}

- GET
- PATCH
- DELETE

/rest/Reports/List

- GET

/rest/Reports/Execute/{Type}

- POST

/rest/Roles

- POST
- GET

/rest/Roles/{Id}

- GET
- PATCH

/rest/Roles/{Id}/DataObjectRows

- GET
- PATCH

/rest/Roles/{Id}/Entities

- GET
- PATCH

/rest/Roles/{Id}/Folders

- GET
- PATCH

/rest/Roles/{Id}/Settings

- GET
- PATCH

/rest/SchedulesV2 v2019.2.29+ v2020.1.3+ v2021.1.1+

- GET
- POST

/rest/SchedulesV2/{ScheduleID} v2019.2.29+ v2020.1.3+ v2021.1.1+

- GET
- PATCH
- DELETE

/rest/SchedulesV2/Flush v2019.2.29+ v2020.1.3+ v2021.1.1+

- POST

/rest/SchedulesV2/{ScheduleID}/Filters v2019.2.29+ v2020.1.3+ v2021.1.1+

- GET
- PATCH

/rest/SchedulesV2/{ScheduleID}/Parameters v2019.2.29+ v2020.1.3+ v2021.1.1+

- GET
- PATCH

/rest/ServerEvents

- GET

/rest/ServerEvents/{Id}

- GET
- DELETE

/rest/Settings

- GET
- PATCH

/rest/StorageMgmt v2020.1+

- GET

- PUT
- PATCH

REST – Batch

Use the **Batch** REST endpoint to group together multiple sequential calls into a single network request. For environments which make a number of programmatic alterations to the session configuration, Batch can significantly reduce the network load and time to create the session.

The session ID can be passed as a URL parameter just as it is for the other REST endpoints.

When supplied as part of a Batch call, the individual requests are evaluated sequentially in the order they are supplied.

> **NOTE**
>
> The GET|PATCH|DELETE /Sessions/{sid} endpoints are not supported by Batch REST.

Batch JSON

The body of a Batch call is formatted as an **array of Batch objects**, each of which has the following JSON format:

Name	Type	Writable	Description
Url	string	required	REST endpoint URL (e.g. the part following /rest/). **IMPORTANT** Unlike standalone calls, the service names are case-sensitive in the Batch endpoint. Reference the service names from this list: • DataSources • Entities • Fields • Folders • Functions • Joins • Parameters • Reports • Roles • SchedulesV2 • ServerEvents • Sessions • Settings • StorageMgmt
Method	constant	required	Request method, one of: POST, PATCH, GET, DELETE **NOTE** The GET|PATCH|DELETE /Sessions/{sid} endpoints are not supported by Batch REST.
Payload	**Payload**	yes	Optional payload for POST and PATCH calls.

Example

```
[
 {
  "Url": "Sessions",
  "Method": "POST"
 },
 {
  "Url": "Roles/MyRole",
  "Method": "PATCH",
  "Payload": { ... }
 }
]
```

Payload JSON

The format of the Payload property depends on the **Url**, and mirrors the payload supplied for the non-batch versions of the endpoint. For example, when the **Url** is *Parameters*, the payload has the following format (as detailed in REST â€" Parameters):

```
{
 "Id":       "MyParameter",
 "DataType":   "String",
 "Value":     "",
 "PromptText": "Input a value",
 "IsHidden":   true
}
```

> **TIP**
>
> When the **Url** is *Sessions/Save*, only the **FileName** property in the payload is required. The current session ID will be used automatically.
>
> For example:
>
> ```
> {
>    "Url": "Sessions/Save",
>    "Method": "POST",
>    "Payload": {
>       "FileName" : "MyConfig.xml"
>    }
> }
> ```

Making Batch Calls

POST /rest/batch

All **Batch** calls are made using the POST method. Otherwise, calls to the Batch endpoint require the same Headers as all other REST calls. Each request object is supplied in order of evaluation. The response object contains the responses from each request in the same order that they were sent.

> **IMPORTANT**
>
> Adding additional Data Source column metadata or performing a GET for the Data Source Fields will result in fixing the current column metadata list as the only defined columns. The column metadata can be further altered, but will not be reflected in the Entity Columns collection until the session is launched in a browser.

Available parameters

Name	Type	Description
config	string	Specify which config file to create the session with. Format: CustomConfigFile **CAUTION** All config files must use the same Temp path. Otherwise, you may experience errors when loading sessions.

Response JSON

Name	Type	Description
ReturnCode	string	**Response code** indicating whether the call was successful
ResponseContent	object	If the response code is **200** or **201**, this is the content of the response. Otherwise this indicates whether an error occurred and the error type.

ResponseContent JSON

ReturnCode: 204

Name	Type	Description
StatusCode	int	Status code
StatusDescription	string	Status description

ReturnCode: 400-500

Name	Type	Description
reason	string	Error message
stacktrace	string	Stack trace, if available

Using curl

```
curl http://{webservice}/rest/batch -X POST ^
 -d "[{'Url':'Sessions','Method':'POST'}, ^
        {'Url':'Roles/Example','Method':'PATCH','Payload':{'IsActive':true}}]"
```

Example response


```
[
 {
  "ReturnCode": "201",
  "ResponseContent": {
   "AppUrl": "ExagoHome.aspx?...",
   "ApiKey": "...",
   "Id": "...",
   "Page": "ExagoHome",
   "ApiAction": "Default",
   "ExportType": null,
   "ShowTabs": true,
   "ShowErrorDetail": true,
   "ReportSettings": {
    "Id": null,
    "ReportPath": null,
    "SortsResource": null,
    "FilterItems": null,
    "IsError": false,
    "ErrorList": null
   }
  }
 },
 {
  "ReturnCode": "204",
  "ResponseContent": {
   "StatusCode": 204,
   "StatusDescription": null
  }
 }
]
```


REST – Sessions

Sessions are the means by which REST persists changes through multiple API calls. A REST session allows for you to easily encapsulate a group of user-specific changes. Most REST changes will only affect the current session, with folder changes being the exception. However, once an API session is committed and Exago is launched, a user will be able to make such changes as their role permits.

> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Entering the API

A **POST /Sessions** call should always be the first made when entering the API. This creates a session, which contains a Session Id and an AppUrl.

Session Id

The **Session Id** is a GUID which uniquely identifies the instance of Exago you're working in.
Example:

```
3c04c155-3317-4dab-a2ee-09572e3a8e44
```

This Id is passed as a URL parameter to all other method calls.

AppUrl

The AppUrl is a string which encodes the session information in a format readable by the host application.
Example:

```
ExagoHome.aspx?d=fk7hInneiNI7yUR1n8fCjjbTuldTWgQrj8AaUlzO8l7K5JMaf7DhbZxzaSAXthP3bWLA%2fzG4AtXApO6pGoBpzQ%3d%3d&showerrordetail=true
```

The AppUrl consists of three components: the Home Page, an alphanumeric string, and the ShowErrorDetail URL parameter.
Once you have committed your API changes and want to launch Exago, append the AppUrl to the end of your Exago base URL and launch the full URL string in a browser:

```
http://yoursite.com/exago/ExagoHome.aspx?d=fk7hInneiNI7yUR1n8fCjjbTuldTWgQrj8AaUlzO8l7K5JMaf7DhbZxzaSAXthP3bWLA%2fzG4AtXApO6pGoBpzQ%3d%3d&showerrordetail=tr
```

An AppUrl can only be used once. Launching an API session will permanently close it, and apply your customization to a new user session in the host application.

TIP

Retrieving the AppUrl via **GET /Sessions/{Id}** should always be the last call made when finishing in the API if you didn't store it after the initial **POST /Sessions** call.

Session JSON

The Sessions endpoint allows for creation and deletion of sessions and access to some general settings. It also allows you to specify and configure the action that the session will take when launched. To launch a report in the full user interface, it needs to be set in Sessions as the output report. Sessions are represented as JSON objects with the following properties:

Table A â€" Session JSON definition

Name	Type	Writable	Description
Id	string	no	The unique Id of this session ("sid")
AppUrl	string	no	The AppUrl of this session; changes to reflect any modifications made to this session in the API
Page	string	yes ("ExagoHome")	The Exago home page
ApiAction	enum	yes ("Default")	**Api Action**, to take when this session is launched
ExportType	enum	yes	**Export Type**, if ApiAction is 'ExecuteReport'
ShowTabs	boolean	yes (true)	Whether the tab bar should be shown
ShowErrorDetail	boolean	yes (true)	Whether detailed error messages should be shown
ReportSettings	**Report**	yes	A report with optional sorts & filters if ApiAction is 'ExecuteReport' or 'EditReport'
ApiKey	string	no	Used when instantiating the JavaScript Api, if applicable.

Session JSON Example

```
{
  "AppUrl": "ExagoHome.aspx?d={alphanumeric}&showerrordetail=true",
  "Id":          "{sid}"
  "Page":        "ExagoHome",
  "ApiAction":    "ExecuteReport",
  "ExportType":   "Html",
  "ShowTabs":     true,
  "ShowErrorDetail": true,
  ...
}
```

Report JSON

To edit or launch a report, specify a report in the Session object's **ReportSettings** property. Reports are represented as JSON objects with the following properties:

Table B â€" Report JSON definition

Name	Type	Writable	Description
Id	string	yes (*v2021.1+*)	The unique ID of the this report Can be used to retrieve a report from the Storage Management database in lieu of the **ReportPath** property below in *v2021.1+*
ReportPath	string	yes	Full path from root folder to this report
SortsResource	Sorts	yes	Any sorts to apply to this report
FilterItems	array of **Filter**	yes	Any filters to apply to this report
IsError	boolean	no	Whether there were any validation errors when loading this report
ErrorList	array of ReportValidationErrorType	no	If IsError == *true*, a list of errors encountered when loading this report. **Possible Errors**

Example

```
"ReportSettings": {
  "Id": "ad0f442e-275b-48a0-893f-0b1255ee7880",
  "ReportPath": "ReportsProduct Sales Report",
  "SortsResource": null,
  "FilterItems": [],
  "IsError": false,
  "ErrorList": null
}
```

NOTE

Windows file paths are delineated by double-backslashes: \\

Sorts JSON

Sorts can be added to the report either in addition to, or replacing existing sorts. This only affects the current session, and does not edit the base report. The collection of sorts to be added is represented as a JSON object with the following properties:

Table C â€" Sorts JSON definition

Name	Type	Writable	Description
ReplaceFlag	string	yes (false)	Whether to replace all existing report sorts
SortItems	array of **Sort**	yes	List of sorts to apply to this report

Example

```
{
  "ReplaceFlag": false,
  "SortItems": [
   {
    "Id":  "Products.ProductName",
    "Asc": false
   },
   {
    "Id":  "Categories.CategoryName",
    "Asc": false
   }
   ...
  ]
}
```

Sort JSON

Each sort in the **SortItems** property is represented as a JSON object with the following properties:

Table D â€" Sort definition

Name	Type	Writable	Description
Id	string	required	Data field to sort on; format: “EntityName.ColumnName”
Asc	boolean	yes (false)	Whether sort direction is ascending

Example:

```
{
  "Id":  "Products.ProductName",
  "Asc": false
}
```

Filter JSON

Filters can be added to the report in addition to existing filters. This only affects the current session, and does not edit the base report. Each filter to be added is represented as a JSON object with the following properties:

Table E â€" Filter JSON definition

Name	Type	Writable	Description
Name	string	required	Data field to filter on; format: “EntityName.ColumnName”
EntityName	string	yes	Entity name to filter on (*deprecated*)
ColumnName	string	yes	Column name to filter on (*deprecated*)
DataType	enum	yes (“string”)	**Data Field Type**
Operator	enum	yes (“EqualTo”)	**Filter Operator Type**
Values	array of strings	yes	Values to filter with
AndFlag	boolean	yes (true)	Whether this filter should be AND-ed with the next (versus OR-ed)
Prompt	boolean	yes (false)	Whether this filter should prompt the user for a value (Advanced or Express Reports)
GroupWithNext	boolean	yes (false)	Whether this filter should be grouped with the next (Advanced or Express Reports)

> **NOTE**
>
> Filter values that are DateTime strings must be able to be parsed by the .NET method DateTime.Parse using the current thread culture.

Example

```
{
  "Name": "Categories.CategoryName",
  "DataType": "string",
  "Operator": "OneOf",
  "Values": [
    "Beverages", "Seafood", "Produce"
  ],
  "AndFlag": true,
  "Prompt": false,
  "GroupWithNext": false
}
```

Create a Session

```
POST /rest/Sessions
```

Available parameters

Table F â€" Parameters for creating a Session

Name	Type	Description
config	string	Specify which config file to create the session with. Format: CustomConfigFile

Using curl

```
curl http://{webservice}/rest/Sessions -X POST ^
-d "{'Page':'Reporting','ApiAction':'Home','ShowTabs':false}"
```

Example response

```
Status: 201 Created
Location: /{webservice}/rest/Sessions/{sid}

{
  "AppUrl": "Reporting.aspx?d={alphanumeric}",
  "Id":          "{sid}",
  "Page":        "Reporting",
  "ApiAction":     "Home",
  "ExportType":    null,
  "ShowTabs":      false,
  "ShowErrorDetail": true,
  "ReportSettings": {
    "ReportPath": null,
    "SortsResource": null,
    "FilterItems": null
  }
}
```

> **CAUTION**
>
> Currently, if a new XML configuration file and temp path are passed via a POST session, the API session .enc file will first be written to the original temp path of the default config file rather than the newly assigned path. This limitation prevents this action from being properly completed. However, a new XML configuration file may still be passed with the default temp path, allowing this limitation to be bypassed.

Save a Session to Storage v2021.2+

Save the current session to persistent storage (e.g. WebReports.xml).

```
POST /rest/Sessions/Save/
```

Payload

Name	Type	Description
sid	string	The session ID (sid) of the session to save.
FileName	string	The file name to save to. Required.

Example Payload

This example writes the current session (with session ID 4a9b1ca6-b2e3-4994-8f7d-8033f5f0d356) to a new file *WebReports2.xml*.

```
{
 "sid" : "4a9b1ca6-b2e3-4994-8f7d-8033f5f0d356",
 "filename" : "WebReports2.xml"
}
```

> **TIP**
>
> When saving a session as part of a Batch REST request, only the **FileName** property in the payload is required. The current session ID will be used automatically.
>
> For example:
>
> ```
> {
>    "Url": "Sessions/Save",
>    "Method": "POST",
>    "Payload": {
>       "FileName" : "MyConfig.xml"
>    }
> }
> ```

Using curl

```
curl http://{webservice}/rest/Sessions/Save -X POST ^
 -d "{'sid':'4a9b1ca6-b2e3-4994-8f7d-8033f5f0d356','FileName':'WebReports2.xml'}"
```

Example Response

```
204 No Content
```

Show a Session

```
GET /rest/Sessions/{sid}
```

If the session is launched, the session object is deleted and can no longer be viewed.

> **NOTE**
>
> Unlike other resources, the {sid} is passed as part of the endpoint string, not as a URL parameter.

Using curl

```
curl http://{webservice}/rest/Sessions/{sid} -X GET
```

Example response

```
Status: 200 OK

{
  "AppUrl": "Reporting.aspx?d={alphanumeric}",
  "Id":   "{sid}",
  "Page": "Reporting",
   ...
}
```

Edit a Session

```
PATCH /rest/Sessions/{sid}
```

Only supply the properties to be edited. If the session is launched, the session object is deleted and it can no longer be edited.

> **NOTE**
>
> Unlike other resources, the {sid} is passed as part of the endpoint string, not as a URL parameter.

Using curl

```
curl http://{webservice}/rest/Sessions/{sid} -X PATCH ^
 -d "{'Page':'CompanyHome'}"
```

Example response

```
Status: 200 OK

{
  "AppUrl": "Reporting.aspx?d={alphanumeric}",
  "Id":  "{sid}",
  "Page": "CompanyHome",
  ...
}
```

Delete a Session

```
DELETE /rest/Sessions/{sid}
```

If a session is deleted, the AppUrl can no longer be used. If the session is launched, the session object is deleted and it can no longer be edited.

> **NOTE**
>
> Unlike other resources, the {sid} is passed as part of the endpoint string, not as a URL parameter.

Using curl

```
curl http://{webservice}/rest/Sessions/{sid} -X DELETE
```

Example response

```
Status: 204 No Content
```

Launch a Report with Sorts and Filters

```
POST /rest/Sessions
```

Set the **ApiAction** to *ExecuteReport*, and specify a report, with any optional sorts and filters, in the **ReportSettings** property. If you want to save the report to a file, set the **ExportType** to the desired file type.

Using curl

```
curl http://{webservice}/rest/Sessions -X POST ^
 -d @sessionSettings.txt
```

sessionSettings.txt

```
{
  "ApiAction":"ExecuteReport",
  "ExportType":"html",
  "ReportSettings":{
    "ReportPath":"SalesProducts",
    "FilterItems":[
      {
        "Name":"Categories.CategoryID",
        "DataType":"Integer",
        "Values":[
          "123"
        ]
      }
    ],
    "SortsResource":{
      "ReplaceFlag":true,
      "SortItems":[
        {
          "Id":"Categories.CategoryID",
          "Asc":true
        }
      ]
    }
  }
}
```

Response

```
Status: 201 Created
Location: /{webservice}/rest/Sessions/{sid}

{
 ...
 "ApiAction":      "ExecuteReport",
 "ExportType":     "Html",
 ...
 "ReportSettings": {
  ...
  "ReportPath": "SalesProducts",
  "SortsResource": {
   "ReplaceFlag": true,
   "SortItems": [
    {
     "Id": "Categories.CategoryID",
     "Asc": true
    }
   ]
  },
  "FilterItems": [
   {
    "Name": "Categories.CategoryID",
    "DataType": "Integer",
    "Values": ["123"]
   }
  ]
 }
}
```

To launch the report in Exago, take the AppUrl, and append it to the URL to your Exago installation to create the full URI path:

```
http://{website}/{exago}/{AppUrl}
```

Example

```
http://yoursite.com/reporting/homepage?d={alphanumeric}&showerrordetail=true
```

REST – Data Sources

A **Data Source** is a resource which contains or accessesÂ the information whichÂ is reported on. Often this is a database, but a variety of types are supported. Data Sources are accessed using a connection string,Â which passes the location of the resource and authentication params. This endpoint allows for session-based read/write access to all data sources for the current config.

> **IMPORTANT**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted for clarity.

Data Source JSON

Data Sources are represented as JSON objects with the following properties:

Name	Type	Writable	Description
Id	string	no	The unique Id of this data source
Name	string	yes	The name of this data source
DbType	enum	yes (“MsSql”)	Data Source Type
Connection	string	yes	The connection string of this data source
Schema	string	yes	The data source schema name (blank for default)

Example

```
{
 "Id":       "1",
 "Name":     "My Data Server",
 "DbType":   "MsSql",
 "Connection": "Server=DataServer;Database=Northwind;uid=username;pwd=password;",
 "Schema":   "dbo"
}
```

List Data Sources

```
GET /rest/DataSources
```

List all the data sources in the current configuration. Output is an array of JSON objects, each representing an individual data source.

```
GET /rest/DataSources?config=myconfig.xml
```

List all of the data objects only in the user's session specific configuration. Replace *myconfig.xml* with the file name of the corresponding config file name. Output is an array of JSON objects, each representing an individual data source.

Name	Type	Description
Id	string	The unique Id of this data source
Name	string	The name of this data source

Using curl

```
curl http://{webservice}/rest/DataSources?sid={sid} -X GET
```

Example response

```
Status: 200 OK
[
 {
  "Id":  "0",
  "Name": "Northwind"
 },
 {
  "Id":  "1",
  "Name": "AdventureWorks"
 }
]
```

Show Data Source

```
GET /rest/DataSources/{Id}
```

Show the properties of the data source specified by its Id.

Using curl

```
curl http://{webservice}/rest/DataSources/{Id}?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
 "Id":       "0",
 "Name":     "Northwind",
 "DbType":   "MsSql",
 "Connection": "Server=DataServer;Database=Northwind;uid=username;pwd=password;",
 "Schema":   "dbo"
}
```

Create Data Source

```
POST /rest/DataSources
```

May be useful if you don't want your connection strings in the config file.

Using curl

```
curl http://{webservice}/rest/DataSources?sid={sid} -X POST ^
 -d "{'Name':'NW','DbType':'MsSql','Connection':'Server=SRV;Database=NW;uid=ID;pwd=PW;'}"
```

Example response

```
Status: 201 Created
Location: /{webservice}/rest/DataSources/2

{
 "Id":       "2",
 "Name":     "NW",
 "DbType":   "MsSql",
 "Connection": "Server=SRV;Database=NW;uid=ID;pwd=PW;",
 "Schema":   ""
}
```

Edit Data Source

```
PATCH /rest/DataSources/{Id}
```

OnlyÂ supply the properties to be edited.

Using curl

```
curl http://{webservice}/rest/DataSources/{Id}?sid={sid} -XPATCH ^
 -d "{'Connection':'Server=SRV;Database=NW;uid=Admin;pwd=Password;'}"
```

Example response

```
Status: 204 No Content
```

Delete Data Source

```
DELETE /rest/DataSources/{Id}
```

Using curl

```
curl http://{webservice}/rest/DataSources/{Id}?sid={sid} -XDELETE
```

Example response

```
Status: 204 No Content
```

REST – Data Objects

Data Objects (referred to as Entities internally and in the API) are the manner by which Exago views and accesses the tables, views, procedures, etc… from the **Data Sources**. Data objects represent the structure of the data, but the actual data is only accessed at report run-time.

> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Data Object JSON

Data Objects are represented as JSON objects with the following properties:

Name	Type	Writable	Description
Id	string	required-create	The unique Id of this data object
Name	string	required	The display name (“alias”) of this data object
Schema	string	yes	The schema of this data object
SchemaAccessType	string	yes	How Exago should retrieve the schema for the data object. There are three possibilities: • *Default* â€” Follow the global **Schema Access Type** setting in **Other Settings**. • *Datasource* â€” Queries the data source for the schema. • *Metadata* â€” Reads the schema from the stored metadata.
CategoryName	string	yes	The Category/Folder group this object will appear in

DataName	string	required	The name of this data object in its data source
			IMPORTANT
			The name of data objects may not contain the following characters:
			`{}[],.%`
			To create a data object for a database table (or other backing model) with one of these characters in its name, create a Custom SQL Object with a simple SELECT clause. For example for a table named sales.figures.2020:
			`SELECT * FROM[sales.figures.2020]`
			Note that the identifier delimiters (the [ ]) around the table name will vary with the database type.

DataSourceId	integer	required	The Id of the data sourceÂ of this data object (see **Data Sources**)
DataType	enum	yes (“Table”)	**Data Object Type**
SqlStatement	string	yes	The custom SQL of this data object if it’sÂ **DataType** is *Sql/Stmt*
canreexecuteindb	Boolean	yes	See **Interactive Filtering in Database** in the Data Object Properties section of the

			Data Objects article for more information
Parameters	array of strings	yes	Any parameters for stored procedure, .NET Assembly or Web Service API calls
KeyColumns	array of strings	yes	The unique key fields of this data object
TenantColumns	array of **Tenant Column**	yes	The tenant fields of this data object
FilterDropdownObject	**Filter Dropdown**	yes	The filter dropdown object of this data object

Example

```
{
 "Id":         "Employees_0",
 "Name":       "Employees",
 "Schema":     "dbo",
 "CategoryName": "",
 "DataName":    "Employees",
 "DataSourceId": "0",
 "DataType":    "Table",
 "SqlStatement": "",
 "canreexecuteindb": true,
 "Parameters":   [],
 "KeyColumns":   ["EmployeeID"],
 "TenantColumns": [
  {
   "Column":   "EmployeeID",
   "Parameter": "UserId"
  }
 ],
 "FilterDropdownObject": {
  "FilterDbName":     "Employee_List",
  "FilterDataSourceId": -1,
  "FilterObjectType":  "view",
  "FilterSchema":     "",
  "FilterSqlStmt":    ""
 }
}
```

Tenant Column JSON

Tenant Columns are represented as JSON objects with the following properties:

Name	Type	Writable	Description
Column	string	required	The tenant data field
Parameter	string	required	The tenant parameter

Example

```
"TenantColumns": [
  {
   "Column":   "EmployeeID",
   "Parameter": "UserId"
  }
 ]
```

Filter Dropdown JSON

A Data Object's Filter Dropdown is represented as a JSON object with the following properties:

Name	Type	Writable	Description
FilterDbName	string	required	The name of this data object in its data source
FilterObjectType	enum	yes	**Data Object Type**
FilterSchema	string	yes	The schema for this data object
FilterSqlStmt	string	yes	The custom SQL for this data object if it is of type SqlStmt

Example

```
"FilterDropdownObject": {
  "FilterDbName":     "Employee_List",
  "FilterObjectType":  "view",
  "FilterSchema":     "",
  "FilterSqlStmt":    ""
}
```

Data Field JSON

Data fields for each object are represented as JSON objects with the following properties. The actual data in the fields is not accessible via REST. Data fields cannot be created or deleted. However, some metadata for existing fields can be edited.

> **IMPORTANT**
>
> The **IsFilterable** property name changed to **Filterable** in *v2019.1.13*.

Name	Type	Writable	Description
Id	string	no	The unique nameÂ for this data field
Name	string	yes	The display name for this data field
Type	enum	yes	**Data Field Type**
IsFilterable (*pre-2019.1.13*) Filterable (*v2019.1.13+*)	bool (*pre-v2017.2*) const (*v2017.2+*)	yes	Whether this field is filterable Filterable Type
IsVisible	boolean	yes	Whether this field is visible to end users. Set to *false* to hide it or *true* to show it.
Value	string	yes	The content of the data field. Enclose other Exago data fields in curly braces { }. There are three possibilities: • *An Exago formula* â€" A custom function used to define the field value. Can use other fields from the same data object. (e.g. â€œConcatenate({Orders.ShipCountry},'+ ', {Orders.ShipCity})â€) • *A SQL formula* â€" SQL statement used to define the the field value. (e.g. â€œConcat(ShipCountry, ' + ', ShipCity)â€) • *A static value* â€" Static content (e.g. "*12345*")
Description	string	yes	A text description of the data field what will be shown to the end user
ColumnSource	string	yes	Describes the content of the **Value** property. Either: • *Data* â€" use for either a static value, or when **Value** contains a single column name • *ExagoFormula* â€" use when the **Value** is an Exago formula • *SqlFormula* â€" use when the **Value** is a SQL formula
SortAndGroupBy	string	yes	Either an Exago formula or another data field name that will be used to sort/group the values in this column. Enclose column names in curly braces { }. If *null*, sorting/grouping on a field is not enabled.

List All Data Objects

List all the data objects in the current configuration. Output is an array of objects, each representing an individual data object.

GET /rest/Entities

Name	Type	Description
Id	string	The unique Id of this dataÂ object
Name	string	The display name ("alias") of this dataÂ object

Using curl

```
curl http://{webservice}/rest/Entities?sid={sid} -X GET
```

Example response

```
Status: 200 OK
[
 {
  "Id":  "Customers_0",
  "Name": "Customers"
 },
 {
  "Id":  "Employees_0",
  "Name": "Employees"
 },
 ...
]
```

List Properties of a Single Data Object

Show the properties of the data object specified by its Id.

GET /rest/Entities/{Id}

Using curl

```
curl http://{webservice}/rest/Entities/{Id}?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
"Id":"system_departments_0",
"Name":"Departments",
"Schema":"",
"CategoryName":"Roster",
"DataName":"system_departments",
"DataSourceId":"4",
"DataType":"Table",
"SqlStatement":"",
"Parameters":[],
"KeyColumns":["id"],
"TenantColumns":[],
"FilterDropdownObject":null
}
```

Add a New Data Object

```
POST /rest/Entities
```

> **NOTE**
>
> Requires a DataName or a custom SqlStatement. One or more KeyColumns are required for most data types.

Using curl

```
curl http://{webservice}/rest/Entities?sid={sid} -X POST ^
 -d @newDataObject.txt
```

newDataObject.txt

```
{
  "Id":"Employees_1",
  "Name":"Employees",
  "Schema":"dbo",
  "DataName":"Employees",
  "DataSourceId":0,
  "KeyColumns":[
    "EmployeeID"
  ]
}
```

Example response

```
Status: 201 Created
Location: /{webservice}/rest/Entities/Employees_1

{
  "Id":         "Employees_1",
  "Name":       "Employees",
  "Schema":     "dbo",
  "CategoryName":  "",
  "DataName":    "Employees",
  "DataSourceId":  "0",
  "DataType":    "Table",
  "SqlStatement":  "",
  "Parameters":   [],
  "KeyColumns":   ["EmployeeID"],
  "TenantColumns": [],
  "FilterDropdownObject": null
}
```

Edit a Data Object

```
PATCH /rest/Entities/{Id}
```

Only supply the properties to be edited.

Using curl

```
curl http://{webservice}/rest/Entities/{Id}?sid={sid} -XPATCH ^
-d "{'Name':'Staff List'}"
```

Example response

```
Status: 204 No Content
```

Delete a Data Object

DELETE /rest/Entities/{Id}

Using curl

```
curl http://{webservice}/rest/Entities/{Id}?sid={sid} -XDELETE
```

Example response

```
Status: 204 No Content
```

List the Data Fields of a Data Object

List all the data fields in the data object specified by its Id. Output is an array of objects, each representing an individual data field.

GET /rest/Entities/{Id}/Fields

Name	Type	Description
Id	string	The unique Id of this dataÂ field
Name	string	The display name of this data field

Using curl

```
curl http://{webservice}/rest/Entities/{Id}/Fields?sid={sid} -XGET
```

Example response

```
Status: 200 OK

[
 {
  "Id":  "Address",
  "Name": "Address"
 },
 {
  "Id":  "BirthDate",
  "Name": "Date of Birth"
 },
 {
  "Id":  "EmployeeID",
  "Name": "ID Number"
 },
 {
  "Id":  "FirstName",
  "Name": "First Name"
 },
 {
  "Id":  "LastName",
  "Name": "Last Name"
 },
 ...
]
```

List Data Field Properties

List the properties of the data field specified by its Id, of the Data Object specified by its Id.

> **IMPORTANT**
>
> Performing a GET for the data fields will result in fixing the current column metadata list as the only defined columns. The column metadata can be further altered, but will not be reflected in the Entity Columns collection until the session is launched in a browser.

GET /rest/Entities/{Id}/Fields/{Field Id}

Using curl

```
curl http://{webservice}/rest/Entities/{Id}/Fields/{Field Id}?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "Id":         "LastName",
  "Name":       "Last Name",
  "Type":       "String",
  "IsFilterable": true,
  "IsVisible":   true
}
```

Add Data Field/Column Metadata

Add new metadata that doesn't already exist to a data field/column.

```
POST /rest/Entities/{Id}/Fields/{Field Id}
```

> **CAUTION**
>
> Setting the **Type** property to a data type that does not match the actual type in the data source will cause errors when reports are run or designed.

> **IMPORTANT**
>
> Adding additional column metadata will result in fixing the current column metadata list as the only defined columns. The column metadata can be further altered, but will not be reflected in the Entity Columns collection until the session is launched in a browser.

Edit Data Field/Column Metadata

Only supply the properties to be edited.

```
PATCH /rest/Entities/{Id}/Fields/{Field Id}
```

Using curl

```
curl http://{webservice}/rest/Entities/{Id}/Fields/{Field Id}?sid={sid} -X PATCH ^
 -d "{'Name':'Surname'}"
```

Example response

```
Status: 204 No Content
```

REST – Data Joins

Joins are the relationship data objects have to each other. Join information is accessible and editable via REST.

> **IMPORTANT**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted for clarity.

Join JSON

Joins are represented as JSON objects with the following properties:

Name	Type	Writable	Description
Id	string	no	The unique Id of this join
EntityFrom	string	required-create	The "from" (or "left") data object of this join
EntityTo	string	required-create	The "to" (or "right") data object of this join
JoinType	const	yes ("Inner")	**Join Type**
RelationshipType	const	yes ("OneToOne")	**Join Relation Type**
Weight	integer	yes (0)	The weight of this join
JoinColumns	array of **JoinColumn**	required	The data fields which are joined

Example

```
{
 "Id":           "Shippers.Orders",
 "EntityFrom":    "Shippers",
 "EntityTo":      "Orders",
 "JoinType":      "Inner",
 "RelationshipType": "OneToMany",
 "Weight":        0,
 "JoinColumns": [
  {
   "ColumnFrom": "ShipperID",
   "ColumnTo":  "ShipVia"
  }
 ]
}
```

JoinColumn JSON

The **JoinColumn** objects of a join indicate which columns are used to join the data objects. Objects can be joined on multiple join columns (which are AND-ed). Each join requires one or more sets of join columns.

JoinColumn objects have the following properties:

Name	Type	Writable	Description
ColumnFrom	string	required	The join data field for the “from” (or “left”) data object
ColumnTo	string	required	The join data field for the “to” (or “right”) data object

Example

```
"JoinColumns": [
 {
  "ColumnFrom": "ShipperID",
  "ColumnTo":  "ShipVia" }
]
```

List All Joins in the Configuration

GET /rest/Joins

List all the joins in the current configuration. Output is an array of objects, each representing an individual join.

Name	Type	Description
Id	string	The unique Id of this join

Available parameters

Name	Type	Description
entity	string	Show only joins that join this data object

Using curl

```
curl http://{webservice}/rest/Joins?sid={sid} -X GET
```

Example Response

```
Status: 200 OK

[
 {
  "Id": "Employees.Orders"
 },
 {
  "Id": "Employees.EmployeeTerritories"
 },
 ...
]
```

Show Properties of a Specific Join

Show the properties of the join specified by its Id.

GET /rest/Joins/{Id}

Using curl

```
curl http://{webservice}/rest/Joins/{Id}?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "Id":             "Orders.OrderDetails",
  "EntityFrom":     "Orders",
  "EntityTo":       "OrderDetails",
  "JoinType":       "Inner",
  "RelationshipType": "OneToOne",
  "Weight":         0,
  "JoinColumns": [
    {
      "ColumnFrom": "OrderID",
      "ColumnTo":   "OrderID"
    }
  ]
}
```

Create a New Join

```
POST /rest/Joins
```

Using curl

```
curl http://{webservice}/rest/Joins?sid={sid} -X POST ^
 -d @newJoin.txt
```

newJoin.txt

```
{
  "EntityFrom":"Shippers",
  "EntityTo":"Orders",
  "RelationshipType":"OneToMany",
  "JoinColumns":[
    {
      "ColumnFrom":"ShipperID",
      "ColumnTo":"ShipVia"
    }
  ]
}
```

Example response

```
Status: 201 Created

{
  "Id":             "Shippers.Orders",
  "EntityFrom":     "Shippers",
  "EntityTo":       "Orders",
  "JoinType":       "Inner",
  "RelationshipType": "OneToMany",
  "Weight":         0,
  "JoinColumns": [
    {
      "ColumnFrom": "ShipperID",
      "ColumnTo":   "ShipVia"
    }
  ]
}
```

Edit Properties of a Join

```
PATCH /rest/Joins/{Id}
```

Only supply the properties to be edited.

Using curl

```
curl http://{webservice}/rest/Joins/{Id}?sid={sid} -X PATCH ^
 -d "{'JoinType':'LeftOuter'}"
```

Example response

```
Status: 204 No Content
```

Delete a Join

```
DELETE /rest/Joins/{Id}
```

Using curl

```
curl http://{webservice}/rest/Joins/{Id}?sid={sid} -X DELETE
```

Example response

```
Status: 204 No Content
```

REST – Folders

Management of the folder and report tree is available with the REST Web Service API. Unlike other resources, managing the tree affects the base configuration, and is not temporary to a session. Therefore it also affects concurrent users.

> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

> **IMPORTANT**
>
> Beginning with *v2020.1* it is strongly recommended to set the Storage Management identity keys before manipulating the folder tree as it can be arranged differently for different parties of users.

Node JSON

The Report Tree is represented as an output-only (i.e. non-writable) resource. Each object in the tree is a Node. Nodes areÂ represented as JSON objects with the following properties:

> **NOTE**
>
> The top-level node has only an Entity property.

Name	Type	Description
name	string	The name of this folder or report
leaf_flag	boolean	Whether this node is a report. *true* if the item is a report, *false* if it is not
readonly_flag	boolean	Whether this node is read-only for the active role
type	enum	**Report Type** (null if this is a folder)
user	string	The active role
Entity	array of **Node**	If this is a folder, an array of the nodes contained within
content_id v2021.1+	string	the content_id of this item in the Storage Management database
content_type v2021.1+	enum	integer value from wrContentType â€" the type of item this is (e.g. report, folder)
report_type v2021.1+	enum	integer value from wrReportType â€" the type of report this item is (e.g. ExpressView, Advanced). This is a duplicate of the type property above.
access_flags v2021.1+	integer	a decimal representation of the access_flags from the content_access table of this content item

Example

```
{
 "Entity": [
  {
   "name":       "Asset Reports",
   "leaf_flag":   false,
   "readonly_flag": false,
   "type":       null,
   "user":       "Admin",
   "Entity": [
    {
     "name":       "Sales Report",
     "leaf_flag":   true,
     "readonly_flag": false,
     "type":       "advanced",
     "user":       "Admin",
     "Entity":     null
    },
    {
     "name":       "Financial Holdings",
     "leaf_flag":   false,
     "readonly_flag": false,
     "type":       null,
     "user":       "Admin",
     "Entity": [
      {
       "name":       "Liquidity Report",
       "leaf_flag":   true,
       "readonly_flag": false,
       "type":       "expressview",
       "user":       "Admin",
       "Entity":     null
      },
      {
       "name":       "Regional Holdings",
       "leaf_flag":   true,
       "readonly_flag": false,
       "type":       "dashboard",
       "user":       "Admin",
       "Entity":     null
      }
     ]
    }
   ]
  }
 ]
}
```


Show Folders Tree

Show the full Report Tree for the current configuration.

```
GET /rest/Reports/List
```

Using curl

```
curl http://{webservice}/rest/Reports/List?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
 "Entity": [
  {
   "name": "Asset Reports",
   ...
   "Entity": [
    {
     "name": "Sales Report",
     ...
     "Entity": null
    },
    {
     "name": "Financial Holdings",
     ...
     "Entity": [
      {
       "name": "Liquidity Report",
       ...
       "Entity": null
      }
     ]
    }
   ]
  }
 ]
}
```

Folder JSON

The /rest/folders endpointÂ represents each folder as an output-only (i.e. non-writable) resource. Folders are represented as JSON objects with the following properties:

Name	Type	Description
Name	string	Full path from root folder to this folder
Status	const	**Folder Status**

Create a Folder

Name is the full path from the root folder to the name of the folder to be created. Spaces and unusual characters must be URL-encoded. Folder names are case sensitive.

```
POST /rest/Folders/{Name}
```

Using curl

```
curl http://{webservice}/rest/Folders/Asset%20Reports/Overseas%Holdings?sid={sid} -XPOST
```

Example response

```
Status: 200 OK

{
 "Id":   null,
 "Name":  "Asset Reports/Overseas Holdings",
 "Status": "Created"
}
```

> **NOTE**
>
> The **Id** property is not used.

Move or Rename a Folder

```
POST /rest/Folders/Rename
```

Folders are moved or renamed by supplying the following JSON resource:

Name	Type	Writable	Description
OldName	string	required	Full path from the root to the folder to be moved or renamed
NewName	string	required	Full path from the root to the new folder location and name

Using curl

```
curl http://{webservice}/rest/Folders/Rename?sid={sid} -X POST ^
 -d "{'OldName':'Asset Reports\Overseas Holdings', ^
 'NewName':'Asset Reports\Regional Holdings'}"
```

> **NOTE**
>
> Windows file paths are delineated by double-backslashes:

Example response

```
Status: 200 OK

{
  "Id":    null,
  "Name":   "Asset Reports/Regional Holdings",
  "Status": "Renamed"
}
```

Delete a Folder

Name is the full path from the root to the name of the folder to be created. Spaces and special characters must be URL-encoded. Folder names are case sensitive. The folder must be empty to be deleted.

```
DELETE /rest/Folders/{Name}
```

Using curl

```
curl http://{webservice}/rest/Folders/Asset%20Reports/Overseas%Holdings?sid={sid} -X DELETE
```

Example response

```
Status: 200 OK

{
  "Id":    null,
  "Name":   "Asset Reports/Overseas Holdings",
  "Status": "Deleted"
}
```

REST – Storage Management

Viewing and modifying the Storage Management configuration via the REST Web Service API is available in Exago *v2020.1+*.

> **IMPORTANT**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Storage Management Configuration JSON

A complete Storage Management configuration is represented as a JSON object with the following properties:

Name	Type	Writable	Description
IsAdmin	boolean	yes	If *true*, user can see the entire contents of the Storage Management database.
Assembly	string	yes	The fully qualified path to the DLL implementing IStorageManagement
ClassName	string	yes	The class name in the **Assembly** implementing IStorageManagement
ReportListCacheEnabled	boolean	yes	*True* if report list caching is enabled
ReportXmlCacheEnabled	boolean	yes	*True* if report definition XML caching is enabled
ReportListCacheTimeout	integer	yes	Number of seconds before the report list cache times out
ThemeListCacheTimeout	integer	yes	Number of seconds before the theme list cache times out
ReportXmlCacheTimeout	integer	yes	Number of seconds before the report XML definition cache times out
TablePrefix	string	yes	Prefix for the Storage Management database tables. For more information, review Storage Management Admin Console.
DefaultInheritFlag	boolean	yes	Value of the Inherit Flag in the configuration. For more information, review Storage Management Admin Console.
DefaultPartyTypeId	integer	yes	ID from the Party Type table to create a content access record with. For more information, review Storage Management Admin Console.
DefaultAccessFlags	integer	yes	Access flags to create a content access record with. For more information, review Storage Management Admin Console.
Identities	string	yes	A collection of name-value pairs for the identity keys. For more information, review

			Storage Management Admin Console.
Options	string	yes	A collection of name-values pairs for Storage Management options. • **DbType** â€" the type of Storage Management database • **DbProvider** â€" the database provider used to connect to the Storage Management database • **ConnectionString** â€" connection string to connect to the Storage Management database • **RptListCacheKey** â€" a prefix for the report list cache key. In Exago's implementation of Storage Management, a report list cache key is constructed from the values of the identity keys. Setting this value will apply a prefix to the cache key. Setting a value here should only be necessary if users with the same identity key values should somehow be handled differently. • **ReportListCacheKey** â€" reserved for future use In a custom Storage Management implementation, this **Options** collection can also be used to pass additional name-value pairs around the application. For more information, review Storage Management Admin Console.

Retrieve the Storage Management Configuration

```
GET /rest/StorageMgmt
```

Returns a complete Storage Management Configuration object.

Update the Storage Management Configuration

```
PUT /rest/StorageMgmt
```

Send a complete Storage Management Configuration JSON object. All existing values will be cleared and replaced with the values in the object. Values that are not in the object will be set to their system defaults.

Example

```
{
 "IsAdmin": false,
 "Assembly": "C:\\Exago\\bin\\WebReports.ContentDatabase.dll",
 "ClassName": "WebReports.ContentDatabase.StorageMgmtDatabase",
 "ReportListCacheEnabled": true,
 "ReportXmlCacheEnabled": false,
 "ReportListCacheTimeout": 31,
 "ThemeListCacheTimeout": 63,
 "ReportXmlCacheTimeout": 32,
 "TablePrefix": "dbo.",
 "DefaultInheritFlag": true,
 "DefaultPartyTypeId": 3,
 "DefaultAccessFlags": 65535,
 "EffectiveReportXmlCacheEnabled": false,
 "SaveValuesImplicit": true,
 "Identities": {
  "userId": "Astro Boy",
  "classId": "admin",
  "companyId": "Exago",
  "ownerId": "Astro Boy"
 },
 "Options": {
  "DbType": "SQLite",
  "DbProvider": "SQLite",
  "ConnectionString": "Data Source=C:\\Exago\\mydatabase.sqlite",
  "ReportListCacheKey": "",
  "RptListCacheKey": "mykey"
 }
}
```

Edit an Individual Storage Management Setting

```
PATCH /rest/StorageMgmt
```

Include only those properties to be updated. Returns with HTTP 200 to indicate success and a copy of the updated configuration as a Storage Management Configuration object.

Example

This example changes the value of three of the Storage Management identity keys.

```
"Identities": {
   "userId": "aboy",
   "classId": "user",
   "companyId": "Exago, Inc."
}
```

REST – Functions

Functions are custom formulas that can be used in reports to manipulate data. Only custom functions can be edited, not the built-in functions.

> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Function JSON

Functions are represented as JSON objects with the following properties:

Name	Type	Writeable	Description
Id	string	required-create	The unique Id of this function
Description	string	yes	The description of this function
MinArgs	integer	no	*(Deprecated in v2017.2)* Minimum number of arguments for this function
MaxArgs	integer	no	*(Deprecated in v2017.2)* Maximum number of arguments for this function
Language	enum	yes	**Code Language**
Namespaces	array of strings	yes	Namespaces for this function
References	array of strings	yes	References for this function
ProgramCode	string	yes	Function code
IsCustom	boolean	no	Whether this is a custom function

Example

```
{
 "Id":        "DayOfWeek",
 "Description": "Returns the day of the week as a string given a date value",
 "MinArgs":    null,
 "MaxArgs":    null,
 "Language":   "cs",
 "Namespaces":  [],
 "References":  ["System.dll"],
 "ProgramCode": "return System.Convert.ToDateTime(args[0].ToString());",
 "IsCustom":    true
}
```

List Functions

```
GET /rest/Functions
```

List all the functions in the current configuration. Output is an array of objects, each representing an individual function.

Name	Type	Description
Id	string	The unique Id of this function

Available parameters

Name	Type	Description
filter	string	Possible values: **custom –** Return only custom functions.

Using curl

```
curl http://{webservice}/rest/Functions?sid={sid} -X GET
```

Example response

```
Status: 200 OK

[
 {
  "Id": "Today"
 },
 {
  "Id": "Tomorrow"
 },
 ...
]
```

Show Function

```
GET /rest/Functions/{Id}
```

Show the properties of the function specified by its Id.

Using curl

```
curl http://{webservice}/rest/Functions/{Id}?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "Id":        "Today",
  "Description": "",
  "MinArgs":    null,
  "MaxArgs":    null,
  "Language":   "cs",
  "Namespaces":  [],
  "References":  [],
  "ProgramCode": "return sessionInfo.Today;",
  "IsCustom":   true
}
```

Create Function

```
POST /rest/Functions
```

Using curl

```
curl http://{webservice}/rest/Functions?sid={sid} -X POST ^
 -d @customFunction.txt
```

customFunction.txt

```
"{'Id':'DayOfWeek','Description':'Returns the day of the week as a string given a date value','References':['System.dll'],'ProgramCode':'return System.Convert.ToDateTime(args[0].ToString
```

Example response

```
Status: 201 Created
Location: /{webservice}/rest/Functions/DayOfWeek

{
  "Id":        "DayOfWeek",
  "Description": "Returns the day of the week as a string given a date value",
  "MinArgs":    null,
  "MaxArgs":    null,
  "Language":   "cs",
  "Namespaces":  [],
  "References":  ["System.dll"],
  "ProgramCode": "return System.Convert.ToDateTime(args[0].ToString());",
  "IsCustom":   true
}
```

Edit Function

```
PATCH /rest/Functions/{Id}
```

Only supply the properties to be edited. Only custom functions can be edited.

Using curl

```
curl http://{webservice}/rest/Functions/{Id}?sid={sid} -X PATCH ^
 -d "{'Namespaces':['WebReports.Api']}"
```

Example response

```
Status: 204 No Content
```

Delete Function

```
DELETE /rest/Functions/{Id}
```

Only custom functions can be deleted.

Using curl

```
curl http://{webservice}/rest/Functions/{Id}?sid={sid} -XDELETE
```

Example response

```
Status: 204 No Content
```

REST – Parameters

Parameters are used throughout the application to store values. The **userId and companyId** parameters are used for identifying the current user. Non-hidden parameters can be accessed in reports.

> **IMPORTANT**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted for clarity.

Parameter JSON

ParametersÂ are represented as JSON objects with the following properties:

Name	Type	Writable	Description
Id	string	required-create	The unique Id of this parameter
DataType	enum	yes (“String”)	**Parameter Type**
Value	string	yes	The value of this parameter
PromptText	string	yes	If set, text to prompt the user for a value
IsHidden	boolean	yes (true)	Whether the parameter is accessible in reports

Example

```
{
 "Id":       "MyParameter",
 "DataType":  "String",
 "Value":     "",
 "PromptText": "Input a value",
 "IsHidden":  true
}
```

ListÂ Parameters

```
GET /rest/Parameters
```

List all the parametersÂ in the current configuration. Output is an array of objects, each representing an individual parameter.

Name	Type	Description
Id	string	The unique Id of thisÂ parameter

Using curl

```
curl http://{webservice}/rest/Parameters?sid={sid} -XGET
```

Example response

Status: 200 OK

```
 {
   "Id": "MyParameter"
 },
 {
   "Id": "HelloWorld"
 }
```

ShowÂ Parameter

```
GET /rest/Parameters/{Id}
```

Show the properties of the parameterÂ specified by its Id.

Using curl

```
curl http://{webservice}/rest/Parameters/{Id}?sid={sid} -XGET
```

Example response

```
Status: 200 OK

{
  "Id":       "MyParameter",
  "DataType":  "String",
  "Value":     "",
  "PromptText": "Input a value",
  "IsHidden":  true
}
```

CreateÂ Parameter

```
POST /rest/Parameters
```

Parameters are designed to be set in the API. If they do not already exist in config, create them programmatically.

Using curl

```
curl http://{webservice}/rest/Parameters?sid={sid} -X POST ^
 -d "{'Id':'Hello','DataType':'String','Value':'World','IsHidden':false}"
```

Example response

```
Status: 201 Created
Location: /{webservice}/rest/Parameters/Hello

{
  "Id":       "Hello",
  "DataType":  "String",
  "Value":     "World",
  "PromptText": "",
  "IsHidden":  false
}
```

EditÂ Parameter

```
PATCH /rest/Parameters/{Id}
```

OnlyÂ supply the properties to be edited.

Using curl

```
curl http://{webservice}/rest/Parameters/{Id}?sid={sid} -X PATCH ^
 -d "{'IsHidden':true}"
```

Example response

```
Status: 204 No Content
```

DeleteÂ Parameter

```
DELETE /rest/Parameters/{Id}
```

Using curl

```
curl http://{webservice}/rest/Parameters/{Id}?sid={sid} -X DELETE
```

Example response

```
Status: 204 No Content
```

REST – Role Security

Roles are collections of security settings for users of the application. They are used to distinguishÂ classes of users by access rights. Setting an active role will cause itÂ to take effect for the application session, once the AppUrl is launched in the browser.Â Once in an application session, the active role cannot be changed. Only one role can be active at a time.

Roles do not affect your access to the API.

> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Role JSON

A role's base status is represented as a JSON object with the following properties:

Name	Type	Writable	Description
Id	string	required-create	The unique Id of this role
IsActive	boolean	yes (true)	Whether this role is active

Example

```
{
 "Id":     "Client",
 "IsActive": false
}
```

ListÂ Roles

```
GET /rest/Roles
```

List all the roles in the current configuration. Output is an array of objects, each representing an individual role.

Name	Type	Description
Id	string	The unique Id of this role

Using curl

```
curl http://{webservice}/rest/Roles?sid={sid} -X GET
```

Example response

```
Status: 200 OK

[
 {
  "Id":  "Admin"
 },
 {
  "Id":  "Client"
 },
 ...
]
```

Create a Role

```
POST /rest/Roles
```

Creating a new role activates it by default.

Using curl

```
curl http://{webservice}/rest/Roles?sid={sid} -X POST ^
 -d "{'Id':'User'}"
```

Example response

```
Status: 201 Created
Location: /{webservice}/rest/Roles/User

{
 "Id":     "User",
 "IsActive": true
}
```

Show Role Status

```
GET /rest/Roles/{Id}
```

Using curl

```
curl http://{webservice}/rest/Roles/{Id}?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "Id":      "User",
  "IsActive": true
}
```

Activate or Deactivate a Role

```
PATCH /rest/Roles/{Id}
```

OnlyÂ supply the properties to be edited.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}?sid={sid} -XPATCH ^
 -d "{'IsActive':false}"
```

Example response

```
Status: 204 No Content
```

DeleteÂ a Role

```
DELETE /rest/Roles/{Id}
```

Using curl

```
curl http://{webservice}/rest/Roles/{Id}?sid={sid} -XDELETE
```

Example response

```
Status: 204 No Content
```

Role Settings JSON

Each role has a group of settings that may be used to override the base config. Each key corresponds with a setting in the Admin Console, which is noted in the Description field. If a setting is blank or null, then the value from the base config is not overridden for this role. The settings are represented as a JSON object with the following properties:

> **NOTE**
>
> All properties are read/write. No properties are required. Default values are inherited from the base config.

Name	Type	Description
ReportPath	string	Report Path
LanguageFile	string	Language File
ServerTimeZoneOffset	integer	Server Time Zone Offset
ReadFilterValues	boolean	Read Database for Filter Values
DbTimeout	integer	Database Timeout
ScheduleManagerViewLevel	enum	**SchedulerÂ Manager User View Level**
DateFormat	string	Date Format
TimeFormat	string	Time Format
DateTimeFormat	string	DateTime Format
SeparatorSymbol	string	Numeric Separator Symbol
CurrencySymbol	string	Numeric Currency Symbol
ShowGrid	boolean	Show Grid Lines in Report Viewer
ShowCrossTabReports	boolean	Allow Creation/Editing of CrossTab Reports
ShowExpressReports	boolean	Allow Creation/Editing of Express Reports
ShowExpressReportsGrouping	boolean	Show Grouping in the Express Report Designer
ShowExpressReportsFormulas	boolean	Show Formula Button in the Express Report Designer
AllowReportCustomSQLObjects v2018.1+	boolean	Allow Creation of Custom SQL Objects in Advanced Reports
ShowExpressReportsStyling	boolean	Show Styling Toolbar in the Express Report Designer
ShowExpressReportsThemes	boolean	Show Themes in the Express Report Designer
ShowScheduleReports	boolean	Show report scheduling options

ShowScheduleReportsManager	boolean	Show Schedule Manager in the main menu
ShowScheduleReportsEmail	boolean	Show Email Report Options
DecimalSymbol	string	Numeric Decimal Symbol

The following settings have been added to versions *v2018.1.27+*,Â *v2018.2.16+*,Â *2019.1.2+* of Exago:

Name	Type	Description
ShowAdvancedReports	boolean	Allow Creation/Editing of Advanced Reports
ShowDashboardReports	boolean	Allow Creation/Editing of Dashboard Reports
ShowDashboardNewVisualizationButton	boolean	Allow Creation/Editing of Dashboard Visualizations
ShowExpressViews	boolean	Allow Creation/Editing of ExpressViews
AllowExpressViewLiveEdit	boolean	Allow Editing ExpressView with Live Data
ShowChainedReports	boolean	Allow Creation/Editing of Chained Reports
CacheVisibilityLevel	wrCacheVisibility	Cache Visibility Level

Example

```
{
 "ReportPath":              "C:\\Exago\\Reports",
 "LanguageFile":            null,
 "ServerTimeZoneOffset":    0,
 "ReadFilterValues":        true,
 "DbTimeout":               0,
 "ScheduleManagerViewLevel": "All",
 "DateFormat":              "MM/dd/yyyy",
 "TimeFormat":              "hh:mm:ss a",
 "DateTimeFormat":          "MM/dd/yyyy hh:mm:ss a",
 "SeparatorSymbol":         ",",
 "CurrencySymbol":          "$",
 "ShowGrid":                true,
 "ShowCrossTabReports":     true,
 "ShowExpressReports":      true,
 "ShowExpressReportsGrouping": true,
 "ShowExpressReportsFormulas": true,
 "ShowExpressReportsStyling": true,
 "ShowExpressReportsThemes": true,
 "ShowAdvancedReports":     true,
 "ShowDashboardReports":    true,
 "ShowDashboardNewVisualizationButton": true,
 "ShowExpressViews":        true,
 "AllowExpressViewLiveEdit": true,
 "ShowChainedReports":      true,
 "CacheVisibilityLevel":    "Global",
 "ShowScheduleReports":     true,
 "ShowScheduleReportsManager": true,
 "ShowScheduleReportsEmail": true,
 "DecimalSymbol":           "."
}
```

Show Role Settings

```
GET /rest/Roles/{Id}/Settings
```

Show the settings for the role specified by its Id.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}/Settings?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
 "ReportPath":              "C:\\Exago\\Reports",
 "LanguageFile":            null,
 "ServerTimeZoneOffset":    0,
 "ReadFilterValues":        true,
 "DbTimeout":               0,
 "ScheduleManagerViewLevel": "All",
 "DateFormat":              "MM/dd/yyyy",
 "TimeFormat":              "hh:mm:ss a",
 ...
}
```

Edit Role Settings

```
PATCH /rest/Roles/{Id}/Settings
```

OnlyÂ supply the properties to be edited.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}/Settings?sid={sid} -X PATCH ^
 -d "{'ShowAdvancedReports':false}"
```

Example response

```
Status: 204 No Content
```

Folder Permissions JSON

Each role has a set of folder permissions which allow or disallow access to folders for the current session. The permissions are represented as a JSON object with the following properties:

> **NOTE**
>
> All properties are read/write. No properties are required.

> **CAUTION**
>
> The Report Tree should contain no more than 1,000 items in it for best user experience.

Name	Type	Description
IncludeAll	boolean	If *True*, all folders are visible to this role except the ones specified in Folders. If *False*, only the folders specified in Folders are visible to this role.
ReadOnly	boolean	If *True*, all folders are read-only except the folders marked ReadOnly = 'true'. If *false*, only theÂ folders marked ReadOnly = 'true' are read-only.
AllowManagement	boolean	If *True*, users will be able create, rename and delete folders via the user interface. If *False*, users cannot take action on folders and the Add Folder buttons are hidden from the user interface.
CreateFolders	boolean	If *True*, when adding Folders to the Role via the API, Exago will check if the folder exists and add it if not. If *False* no check will be performed. Once the session is launched, this property no longer applies.
Folders	array of **Folder**	The folders with properties specific to this role

Folder JSON

Each folder in the Folders propertyÂ is represented as a JSON object with the following properties:

Name	Type	Description
Name	string	Full path from the root folder to this folder
ReadOnly	boolean	Whether this folder is ReadOnly
Propagate	boolean	Whether this folder's subfolders share its ReadOnly property

Example

```
{
 "IncludeAll":    false,
 "ReadOnly":      false,
 "AllowManagement": true,
 "CreateFolders":  false,
 "Folders": [
  {
   "Name":    "ExamplesClient",
   "ReadOnly":  true,
   "Propagate": true
  }
 ]
}
```

List Folder Permissions

```
GET /rest/Roles/{Id}/Folders
```

Show the folder permissions for the role specified by its Id.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}/Folders?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
 "IncludeAll":     false,
 "ReadOnly":       false,
 "AllowManagement": true,
 "CreateFolders":   false,
 "Folders": [
  {
   "Name":     "Examples",
   "ReadOnly":  true,
   "Propagate": true
  }
 ]
}
```

Edit Folder Permissions

```
PATCH /rest/Roles/{Id}/Folders
```

Only supply the properties to be edited. Supplying a Folders property will overwrite the entire collection.

Using curl

```
curl http://{webservice}/rest/roles/{Id}/folders?sid={sid} -X PATCH ^
-d "{'ReadOnly':true}"
```

Example response

```
Status: 204 No Content
```

Data Object PermissionsÂ JSON

Each role has a set of data object permissions which allow or disallow access to data objects for the current session. The permissions are represented as a JSON object with the following properties:

> **NOTE**
>
> All properties are read/write. No properties are required.

Name	Type	Description
IncludeAll	boolean	If *true*, all data objects are available to this role except the ones specified in Entities. If *false*, only the data objects specified in Entities are visible to this role.
Entities	array of **Data Object**	The data objects with properties specific to this role

Data Object JSON

Each Data Object in the Entities property is represented as a JSON object with the following properties:

Name	Type	Description
Id	string	The unique Id of this data object

Example

```
{
 "IncludeAll": true,
 "Entities": [
  {
   "Id": "EMP"
  },
  {
   "Id": "ETE"
  }
 ]
}
```

List Data Object Permissions

```
GET /rest/Roles/{Id}/Entities
```

Show the data object permissions for the role specified by its Id.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}/Entities?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "IncludeAll": false,
  "Entities": [
    {
      "Id": "EMP"
    }
  ]
}
```

Edit Data Object Permissions

```
PATCH /rest/Roles/{Id}/Entities
```

Only supply the properties to be edited. Supplying an Entities property will overwrite the entire collection.

Using curl

```
curl http://{webservice}/rest/roles/{Id}/entities?sid={sid} -X PATCH ^
 -d "{'IncludeAll':false}"
```

Example response

```
Status: 204 No Content
```

Row Filters JSON

You can add security filters to data objects so that users can only view specific rows in the data object. The collection of row filters for this role is represented as a JSON object with the following properties:

Name	Type	Description
DataObjectRows	array of **Row Filter**	The row filters for this role

Row FilterÂ JSON

Each row filterÂ in the **DataObjectRows** propertyÂ is represented as a JSON object with the following properties:

Name	Type	Description
Id	string	The Id of the data object to filter
Filter	string	The filter string. The filter string should be valid, standard SQL to be added to the 'WHERE' clause sent to the Data Source. This filter string can include Exago system parameters (e.g. @) or custom SQL including subqueries (e.g. IN (SELECT EmployeeID FROM Employees WHERE EmployeeID <4)). The **Filter String** must contain the actual name of objects in the Data Source (e.g. tables, views, columns, etc…) rather than their Exago aliases.

Example

```
{
  "DataObjectRows": [
    {
      "Id":    "EMP",
      "Filter": "EmployeeID = 1"
    },
    {
      "Id":    "ETE",
      "Filter": "EmployeeID = 2"
    }
  ]
}
```

List Row Filters

```
GET /rest/Roles/{Id}/DataObjectRows
```

Show the row filters for the role specified by its Id.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}/DataObjectRows?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "DataObjectRows": [
    {
      "Id":    "EMP",
     "Filter": "EmployeeID = 3"
    },
    ...
  ]
}
```

Edit Row Filters

```
PATCH /rest/Roles/{Id}/DataObjectRows
```

This will overwrite the entire filter collection.

Using curl

```
curl http://{webservice}/rest/Roles/{Id}/DataObjectRows?sid={sid} -X PATCH ^
-d "{'DataObjectRows':[{'Id':'EMP','Filter':'Client'}]}"
```

Example response

```
Status: 204 No Content
```

REST – SchedulesV2

Viewing, creating, modifying, and deleting schedules via the **REST Web Service API** is available in Exago *v2021.1.1+*. This feature requires the installation and proper configuration of the Scheduler Service.

> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Schedule JSON

A schedule is represented as a JSON object with the following properties:

Name	Type	Writable	Description
JobId	string	no	A GUID which uniquely identifies this schedule in the application.
ScheduleName	string	required-create	The descriptive name of the schedule.
ReportName	string	required-create	The name of the report being scheduled including the relative folder path.
Status	string from ScheduleJobStatus	no	ScheduleJobStatus The read-only current status of the schedule.
ExportType	string from ScheduleExportType	yes	ScheduleExportType The file type of the created report when the schedule runs.
Password	string	yes	The PDF or Excel output file password in plain-text.
IsArchive	boolean (*false*)	no	*true* if this schedule object represents an execution that was saved to disk as opposed to being e-mailed. *false* if this schedule object represents an execution that was e-mailed.
JobType	string from ScheduleJobType	no	ScheduleJobType The read-only type of job this schedule will perform.
Recurrence	ScheduleRecurrence	yes	A JSON object representing the schedule's recurrence pattern (e.g. time, day and range of running).
FilterValues	array of ScheduleCompositeFilter	yes	An array of JSON objects representing the prompting filters to apply to the component reports of a composite report (Chained Report or Dashboard)
Filters	array of ScheduleFilterItem	yes	An array of JSON objects representing the prompting filters to apply to the report in this schedule, if the **ReportName** refers to a report that is not a Chained Report or Dashboard.

Parameters	array of ScheduleParameter	yes	An array of JSON objects representing the parameters for this schedule.
Batch	ScheduleBatchInfo	yes	A JSON object representing the batch processing information.
Recipients	ScheduleRecipientInfo	yes	A JSON object representing the e-mail recipients of this schedule's report.
ParentJobId v2021.1.15+	string	no	The GUID of the scheduler job that created this one. A scheduled job that recurs 4 times will ultimately create 5 job filesâ€"1 for the original schedule, and one for each of the 4 executions of the schedule. This field holds the GUID of the original schedule. If this field is an empty string, then it is a parent job.

Schedule Example

This example represents a schedule named **Report Totals** which will run a report named **Employee Orders** from the **My Reports** folder. When it runs, a PDF file with the password "abcd" will be generated an attached to an e-mail sent to This schedule will run every hour starting at 9:00 AM until 5:00 PM every day of the week from March 1, 2021â€"June 1, 2021.

```
{
  "JobId": "4c835b79-56e3-4051-a0c3-6d21d9576bbc",
  "JobType": "Report",
  "ScheduleName": "Report Totals",
  "ReportName": "My Reports\Employee Orders",
  "ReportId": "9b5d7d7f-53fe-4db3-8914-c5c3fad9fb70",
  "Status": "Ready",
  "ExportType": "Pdf",
  "Password": "abcd",
  "IsArchive": false,
  "Recurrence": {
    "IsImmediate": false,
    "RecurrenceType": "Daily",
    "RecurrenceRange": "EndByDate",
    "ScheduleTime": "09:00",
    "RepeatEvery": true,
    "RepeatEveryHours": 1,
    "RepeatEveryMinutes": 0,
    "RepeatEveryEndTime": "17:00",
    "RangeStartDate": "2021-03-01",
    "RangeEndDate": "2021-06-01",
    "OccurrencesToEnd": 1,
    "RecurrencePattern": {
      "Once": null,
      "Daily": {
        "IsEveryWeekday": true,
        "DailyNumberOfDaysInterval": 0
      },
      "Weekly": null,
      "Monthly": null,
      "Yearly": null
    }
  },
  "Filters": null,
  "FilterValues": null,
  "Parameters": [],
  "Batch": null,
  "Recipients": {
    "EmailResults": true,
    "AttachOutputToEmail": true,
    "ToAddresses": [
      "[email protected]"
    ],
    "CcAddresses": [],
    "BccAddresses": [],
    "ReplyToAddresses": [],
    "EmailSubject": "Hourly order update",
    "EmailBody": "Please find the hourly order update attached to this e-mail."
  },
  "ParentJobId" : "9f0b39c0-3170-4ca1-ac21-e830356bf6fb"
}
```

List All Schedules

```
GET /rest/SchedulesV2
```

List all of the schedules in the current configuration. Returns an array of **ScheduleListItem** objects, each representing an individual schedule.

Create New Schedule

```
POST /rest/SchedulesV2
```

Create a new schedule. The **ScheduleName** and **ReportName** are required. Returns HTTP 201 Created with a **Schedule** object representing

the new schedule.

Example

This example creates a new schedule named MyTestSchedule. This schedule will run a report named **Products by Category** from the **ExpressViews** folder 10 times, starting on October 17, 2019. The report will run at 2:45 PM only on weekdays. The output will be saved as a PDF file to a directory on the server's file system.

```
{
  "ScheduleName": "MyTestSchedule",
  "ReportName": "ExpressViewsProducts by Category",
  "ExportType": "Pdf",
  "Recurrence": {
    "IsImmediate": false,
    "RecurrenceType": "Daily",
    "RecurrenceRange": "EndAfterNOccurences",
    "ScheduleTime": "14:45",
    "RepeatEvery": false,
    "RangeStartDate": "2019-10-17",
    "RangeEndDate": null,
    "OccurrencesToEnd": 10,
    "RecurrencePattern": {
      "Daily": {
       "IsEveryWeekday" : true,
        "DailyNumberOfDaysInterval": 0
      }
    }
  }
}
```

Show a Schedule

```
GET /rest/SchedulesV2/{scheduleID}
```

Show the properties of the schedule specified by the *scheduleID*. Returns a **Schedule** object representing the *schduleID* requested.

Edit a Schedule

```
PATCH /rest/SchedulesV2/{scheduleID}
```

Edit the properties of the schedule specified by the *scheduleID*. Include only those properties to be updated. Returns no JSON objects but will return HTTP status code 204 to indicate success.

Example

This example changes the **ExportType** of the schedule with ID aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee to be RTF.

```
PATCH /schedulesv2/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee
```

```
{
 "ExportType": "Rtf"
}
```

Delete a Schedule

```
DELETE /rest/SchedulesV2/{scheduleID}
```

If the specified schedule is active (that is, it is not *Completed* or *Deleted*) it will be set to *Deleted* status. If the specified schedule is inactive, it will be flushed from the scheduler. Therefore, two calls must be made or a separate call to flush the scheduler must be made if the schedule was active when it deleted and the intention is to flush it. Returns no JSON objects but will return HTTP status code 204 to indicate success.

Example

This example marks schedule with ID aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee as *Deleted*.

```
DELETE /rest/SchedulesV2/aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeee
```

Flush the Scheduler

```
POST /rest/SchedulesV2/Flush
```

Flush the scheduler of all *Completed* and *Deleted* schedules. An array of **ScheduleListItem** will be returned with the resulting schedule list after the flush has occurred.

ScheduleListItem JSON

A JSON object which represents an item in the schedule list. An array of **ScheduleListItem** is returned when calling GET /SchedulesV2

Name	Type	Writable	Description

Id	string	no	The GUID which uniquely identifies this schedule.
ParentJobId v2021.1.15+	string	no	The GUID of the scheduler job that created this one. A scheduled job that recurs 4 times will ultimately create 5 job filesâ€"1 for the original schedule, and one for each of the 4 executions of the schedule. This field holds the GUID of the original schedule. If this field is an empty string, then it is a parent job.
ScheduleName	string	no	The descriptive name of the schedule. The same as entering a **Schedule Name** into the Scheduled Report Wizard in the user interface
Type	string from ScheduleRecurrenceType	no	ScheduleRecurrenceType Indicates the type of recurrence this schedule has.
ReportName	string	no	The name of the report being scheduled
LastExecDate	string	no	The date and time of the most recent execution of this schedule, in UTC time as YYYY-MM-DDThh:mm:ssz format. If the schedule has never run, returns null.
NextExecDate	string	no	The date and time of the next execution of this schedule, in UTC time as YYYY-MM-DDThh:mm:ssz format. If the schedule will not run again, returns null.
Status	string from ScheduleJobStatus	no	ScheduleJobStatus The read-only current status of the schedule.
RunCount	integer	no	The number of times this schedule has been executed.
UserId	string	no	The value of the built-in @ parameter when the schedule was created.
CompanyId	string	no	The value of the built-in @ parameter when the schedule was created.
JobType	string from ScheduleJobType	no	ScheduleJobType The type of job that this schedule represents

Example

This array contains one **ScheduleListItem** representing a daily recurring schedule that has already run once and will run again the following hour.


```
{
  "Id": "4c835b79-56e3-4051-a0c3-6d21d9576bbc",
  "ParentJobId": "9f0b39c0-3170-4ca1-ac21-e830356bf6fb",
  "ScheduleName": "Report Totals",
  "Type": "Daily",
  "ReportName": "Employee Orders",
  "LastExecDate": "2021-02-12T21:46:37Z",
  "NextExecDate": "2021-02-12T22:00:00Z",
  "Status": "Ready",
  "RunCount": 1,
  "UserId": "Fiona Light",
  "CompanyId": "Tip O' The Nose Services, LLC",
  "JobType": "Report"
}
```


ScheduleRecurrence JSON

A JSON object representing a schedule recurrence pattern.

Name	Type	Writable	Description
IsImmediate	boolean	yes (*false*)	Determines if the schedule will run immediately or not. If *true* the schedule runs immediately. Must be *false* in order to use the other recurrence properties.
RecurrenceType	string from RecurrenceType	yes	RecurrenceType The type of recurrence to use.
RecurrenceRange	string from RecurrenceRange	yes	RecurrenceRange Determines when the schedule's recurrence pattern will end.
ScheduleTime	string	required	The time at which the schedule should run. Must be formatted in hh:mm format utilizing a 24-hour clock. (e.g. 9:00 AM = 09:00, 1:37 PM = 13:37)
RepeatEvery	boolean	yes (*false*)	If *true*, the schedule will repeat every **RepeatEveryHours** hours and **RepeatEveryMinutes** minutes until **RepeatEveryEndTime**.
RepeatEveryHours	integer	yes	If **RepeatEvery** is *true*, the schedule will run every this number of hours.
RepeatEveryMinutes	integer	yes	If **RepeatEvery** is *true*, the schedule will run every this number of minutes.
RepeatEveryEndTime	string	yes	If **RepeatEvery** is *true*, the schedule will stop repeating at this time. Must be formatted in hh:mm format utilizing a 24-hour clock. (e.g. 9:00 AM = 09:00, 1:37 PM = 13:37)
RangeStartDate	string	yes	The date on which the schedule will start recurring, in yyyy-mm-dd format. Set to {null} to clear an existing value when editing a schedule.
RangeEndDate	string	yes	If **RecurrenceRange** is *EndByDate*, the schedule will stop recurring on this date, in yyyy-mm-dd format.
OccurrencesToEnd	integer	yes	If **RecurrenceRange** is *EndAfterNOccurences*, the schedule will stop recurring after this many occurences have run.

RecurrencePattern	ScheduleRecurrencePattern	yes	A JSON object representing the pattern that the recurrence should follow.

ScheduleRecurrencePattern JSON

A JSON object representing a schedule recurrence pattern. The pattern selected here should match the *RecurrenceType* set in **ScheduleRecurrence**.

> **IMPORTANT**
>
> Pick one and only one of the properties.

Name	Type	Writable	Description
Once	OncePattern	yes	A JSON object that represents a schedule that should run only once.
Daily	DailyPattern	yes	A JSON object that represents a schedule which will run on a daily basis.
Weekly	WeeklyPattern	yes	A JSON object that represents a schedule which will run on a weekly basis.
Monthly	MonthlyPattern	yes	A JSON object that represents a schedule which will run on a monthly basis.
Yearly	YearlyPattern	yes	A JSON object that represents a schedule which will run on a yearly basis.

OncePattern JSON

A JSON object representing a schedule that does not recur, but runs only one time.

Name	Type	Writable	Description
OnceDate	string	required	The date on which the schedule shall run, in yyyy-mm-dd format.

DailyPattern JSON

A JSON object representing a schedule recurrence pattern which occurs on a day-to-day basis

Name	Type	Writable	Description
IsEveryWeekday	boolean	required	If *true*, the schedule runs every weekday, Mondayâ€“Friday. If *False*, set **DailyNumberOfDaysInterval**.
DailyNumberOfDaysInterval	integer	yes	If **IsEveryWeekday** is *false*, the schedule will skip this number of days between exections. For example, setting this value to *2* will result in a report executing on the *1st, 4th, 7th, etc…* day of the month as there are two days between those dates.

Example

This Daily recurrence pattern runs every weekday.

```
{
 "IsEveryWeekday": true
}
```

WeeklyPattern JSON

A JSON object representing a schedule recurrence pattern which occurs on a weekly basis.

Name	Type	Writable	Description
WeeklyNumberOfWeeksInterval	integer	required	The schedule will execute this number of weeks.
WeeklyDaysOfWeek	array of boolean	required	The schedule will run only on the days contained in the array, with the first element of the array representing Sunday, the second element representing Monday, and so on. The array must contain exactly seven boolean values. Set the value to *true* to have the schedule run for that day, or *false* for it to skip that day.

Example

In this example, the schedule will run every Sunday, Wednesday and Saturday of the week.

```
{
 "WeeklyNumberOfWeeksInterval": "1",
 "WeeklyDaysOfWeek": [
  true,
  false,
  false,
  true,
  false,
  false,
  true
 ]
}
```

MonthlyPattern JSON

A JSON object representing a schedule recurrence pattern which occurs on a monthly basis.

Name	Type	Writable	Description
IsOnSpecificDayOfMonth	boolean	yes	Determines whether a schedule will recur on a specific day of the month, or on a relative position in the month (e.g. third Monday of the month). Set to *true* for a specific day, or *false* for a relative day.
MonthlySpecificDayNumber	integer	yes	If **IsOnSpecificDayOfMonth** is *true*, this is the calendar day of the month that the schedule will recur. Otherwise, null.
MonthlySpecificEveryNMonths	integer	yes	If **IsOnSpecificDayOfMonth** is *true*, the monthly schedule will run every this number of months. Otherwise, null.
MonthlyDayPosition	string	yes	If **IsOnpecificDayOfMonth** is *false*, the monthly recurrence will run on a relative position in the month. One and only one of the following values are accepted: *first*, *second*, *third*, *fourth*, *last*. Works in combination with **MonthlyDayOfWeek**.
MonthlyDayOfWeek	string	yes	If **IsOnSpecificDayOfMonth** is *false*, the days on which the relative monthly recurrence occurs. One and only one of these values may be used: *day*, *weekday*, *weekendday*, *sun*, *mon*, *tue*, *wed*, *thu*, *fri*, *sat*.
MonthlyRelativeEveryNMonths	integer	yes	If **IsOnSpecificDayOfMonth** is *false*, the schedule will recur every this number of months.

Example

In this example, the schedule will run on the second Monday of every other month.

```
{
  "IsOnSpecificDayOfMonth" : false,
  "MonthlyDayPosition" : "second",
  "MonthlyDayOfWeek" : "mon",
  "MonthlyRelativeEveryNMonths" : 2
}
```

YearlyPattern JSON

A JSON object which represents a yearly schedule recurrence pattern.

Name	Type	Writable	Description
IsOnSpecificDayOfYear	boolean	required	If *true*, the yearly schedule will recur on a specific day of the year. Use **YearlySpecificMonth** and **YearlySpecficDayNumber** to set the date. If *false*, the schedule will recur on a relative part of the year. Use **YearlyRelativeDayPosition**, **YearlyRelativeDayOfWeek**, **YearlyRelativeMonth** to select the relative day of recurrence.
YearlySpecificMonth	integer	yes	If **IsOnSpecificDayOfYear** is *true*, this is the month portion of the date on which the schedule will recur. Provide an integer representing the month of the calendar (1 = January, 2 = February … 12 = December)
YearlySpecficDayNumber	integer (*1*)	yes	If **IsOnSpecificDayOfYear** is *true*, this is the day portion of the date on which the schedule will recur. Any positive integer 1â€“31 is accepted.
YearlyRelativeWeekPosition	string	yes	If **IsOnSpecificDayOfYear** is *false*, this is the relative week in the month that this schedule will recur. One and only one of the following values are accepted: *first*, *second*, *third*, *fourth*, *last*. Works in combination with **YearlyRelativeDayOfWeek** and **YearlyRelativeMonth**.
YearlyRelativeDayOfWeek	string	yes	If **IsOnSpecificDayOfYear** is *false*, this is the relative day of the week in the month that the schedule will recur. One and only one of these values may be used: *day*, *weekday*, *weekendday*, *sun*, *mon*, *tue*, *wed*, *thu*, *fri*, *sat*.
YearlyRelativeMonth	string	yes	If **IsOnSpecificDayOfYear** is *false*, this is the number of the month in the year that the schedule will recur. (e.g. (1 = January, 2 = February … 12 = December))

Examples

This example will run the schedule every year on the last Sunday in September.

```
{
  "IsOnSpecificDayOfYear": false,
  "YearlyRelativeDayPosition": "last",
  "YearlyRelativeDayOfWeek": "sun",
  "YearlyRelativeMonth": 9
}
```

This example will run the schedule on April 15th of every year.

```
{
  "IsOnSpecificDayOfYear": true,
  "YearlySpecificMonth" : "April",
  "YearlySpecficDayNumber" : 15
}
```

ScheduleFilterItem JSON

A JSON object representing a group of one or more ScheduleFilterElement objects, defining the grouping of the prompting filters on a non-composite (ExpressView, Express, Advanced, CrossTab) report.

Name	Type	Writable	Description
Group	array of ScheduleFilterItem	yes	The contents of a group of filters. If the filters are not grouped together, this value is null.
Filter	array of ScheduleFilterElement	yes	The filter statements contained in the group.

ScheduleFilterElement JSON

A JSON object representing an individual filter on a non-composite (ExpressView, Express, Advanced, CrossTab) report.

Name	Type	Writable	Description
FilterText	string	yes	The field to be filtered, which can include applied formulas. This field must be one of the fields on the scheduled report.
Operator	string from ScheduleFilterOperator	yes	ScheduleFilterOperator The filter operator.
AndFlag	boolean	yes	If *true*, this filter will be logically ANDed with the next filter in the array. If *false*, this filter will be logically ORed with the next filter in the array.
Values	array of number, string or boolean	yes	The values to filter by. When using **Operator.Between** or **Operator.NotBetween** the array must contain exactly two values. When using **Operator.OneOf** or **Operator.NotOneOf** the array may contain any number of values. All other operators use only one value, which must still be in an array. Date and DateTimes must be represented as strings in yyyy-mm-dd format.

Example

This Filters array represents the following filter summary from the Advanced Report Designer: Work Order Tickets.Open Date > '05/14/2017' And (Work Order Tickets.Description Contains 'lightning' And Work Order Tickets.Description Contains 'RMA')

```
"Filters": [
  {
   "Group": null,
   "Filter": {
    "FilterText": "Work Order Tickets.Open Date",
    "Operator": "GreaterThan",
    "AndFlag": true,
    "Values": [
     "2017-05-14 23:59:59.997"
    ]
   }
  },
  {
   "Group": [
    {
     "Group": null,
     "Filter": {
      "FilterText": "Work Order Tickets.Description",
      "Operator": "Contains",
      "AndFlag": true,
      "Values": [
       "lightning"
      ]
     }
    },
    {
     "Group": null,
     "Filter": {
      "FilterText": "Work Order Tickets.Description",
      "Operator": "Contains",
      "AndFlag": true,
      "Values": [
       "RMA"
      ]
     }
    }
   ],
   "Filter": null
  }
]
```

ScheduleCompositeFilter JSON

A JSON object representing a prompting filter on a composite (Chained or Dashboard) report. ScheduleCompositeFilter objects appear in an array in the Schedule object's FilterValues property.

Name	Type	Writable	Description

Reports	array of int	yes	An array of component report indices on a composite report (a Chained Report or Dashboard) that this filter applies to. *0* represents the first report, *1* represents the second report, and so on.
FilterText	string	yes	The data field or formula that contains the data that will be filtered. It must be on all of the reports indicated by the **Reports** array.
Operator	string from ScheduleFilterOperator	yes	ScheduleFilterOperator The filter operator.
AndFlag	boolean	yes	If *true*, this filter will be logically ANDed with the next filter in the array. If *false*, this filter will be logically ORed with the next filter in the array.
Values	array of number, string or boolean	yes	The values to filter by. When using **Operator.Between** or **Operator.NotBetween** the array must contain exactly two values. When using **Operator.OneOf** or **Operator.NotOneOf** the array may contain any number of values. All other operators use only one value, which must still be in an array. Date and DateTimes must be represented as strings in yyyy-mm-dd format.

Example

This Schedule JSON object excerpt is from a scheduled Chained Report with two reports in the chain. It's **Filters** property is null, but the **FilterValues** property contains an array of ScheduleCompositeFilter JSON objects, one for each prompting filter. Note the Employees.Eid applies to both reports in the chain.

```
"Filters": null,
  "FilterValues": [
   {
    "Reports": [
     0
    ],
    "FilterText": "=Year({Employees.HireDate})",
    "Operator": "EqualTo",
    "AndFlag": true,
    "Values": [
     "2009"
    ]
   },
   {
    "Reports": [
     0,
     1
    ],
    "FilterText": "Employees.EID",
    "Operator": "OneOf",
    "AndFlag": true,
    "Values": [
     "4",
     "5",
     "6"
    ]
   },
   {
    "Reports": [
     1
    ],
    "FilterText": "=Year({Orders.OrderDate})",
    "Operator": "EqualTo",
    "AndFlag": true,
    "Values": [
     "2016"
    ]
   }
  ],
```

ScheduleParameter JSON

A JSON object representing a prompting parameter on the scheduled report.

Name	Type	Writable	Description
Id	string	required	The Id (name) of the parameter on the scheduled report. To update the parameter's value, the parameter with this Id must appear on the report being scheduled and must be prompting.
Value	integer, string or date	yes	The value to set the parameter to, which is the same as setting the value of a prompting parameter when running a report. It does not change the underlying default parameter value. For boolean parameters, use *0* for *false* or *1* for *true*.
DisplayValue	integer, string or date	yes	The value that is displayed in the

			report, if different than the **Value**.%
Reports	array of integer	yes	An array of component report indices on a composite report (a Chained Report or Dashboard) that this parameter applies to. *0* represents the first report, *1* represents the second report, and so on.

ScheduleBatchInfo JSON

A JSON object representing the information for processing individualized scheduled reports, also known as a **batch report**. The individualized reports will be e-mailed to recipients based on a **Data Object** field contained within **BatchEmailEntityCategory** > **BatchEmailEntity** > **BatchEmailField**. A summary of the batch report's execution will be sent by e-mail to **ToAddresses** and CC'ed to **CcAddresses**.

More information about **Batch E-Mailing** can be found in the article on Scheduling Reports.

Name	Type	Writable	Description
AttachReportToEmail	boolean (*false*)	yes	If *true*, the report output will be attached to the individualized e-mail messages.
ToAddresses	array of string	yes	An array of e-mail addresses to send the batch report results summary to.
CcAddresses	array of string	yes	An array of e-mail addresses to carbon copy e-mail the batch report results summary to.
BatchEmailEntityCategory	string	yes	The Data Object that **BatchEmailEntity** belongs to. The folder must exist in Exago.
BatchEmailEntity	string	yes	The Data Object containing the batch recipient information.
BatchEmailField	string	yes	The field in **BatchEmailEntity** that contains the e-mail address of each recipient of the individualized batched report.

Example

This example sends a copy of the schedule's execution summary to , and sends the individualized executed report to each entry with an EmailAddress in the Employees data object.

```
{
 "AttachReportToEmail": true,
 "ToAddresses": [
  "[email protected]"
 ],
 "CcAddresses": [],
 "BatchEmailEntityCategory": "",
 "BatchEmailEntity": "Employees",
 "BatchEmailField": "EmailAddress"
}
```

ScheduleRecipientInfo JSON

A JSON object representing an e-mail to send when the schedule has run and to optionally attach the report to the e-mail.

Name	Type	Writable	Description
EmailResults	boolean	yes	If *true*, the report is attached to an e-mail message. If *false*, the report is saved to disk instead. If this is a batch schedule, this is always *true*. Has the same effect as checking or unchecking the **Email Results** checkbox on the **Recipients** tab of the Scheduled Report Wizard.
AttachOutputToEmail	boolean (*true*)	yes	If this is a batch schedule, if *true*, the completed report is attached to the e-mail message for each recipient. If *false*, the report is not attached to the e-mailâ€"the e-mails can be used for delivering personal data-driven alerts or updates. Has the same effect as checking or unchecking the **Attach Report Output to Email** checkbox on the **Recipients** tab of the Scheduled Report Wizard for batch reports.
ToAddresses	array of string	required	An array of e-mail addresses to send the e-mail to.
CcAddresses	array of string	yes	An array of e-mail addresses to CC to.
BccAddresses	array of string	yes	An array of e-mail addresses to BCC to.
ReplyToAddresses	array of string	yes	An array of e-mail addresses that will appear as the Reply-To: addresses in the e-mail.
EmailSubject	string	yes	The subject line of the e-mail message.
EmailBody	string	yes	The body text/content of the e-mail message.

Example

This example attaches the report output file to an e-mail and sends it to and The e-mail is CCed to and , and BCCed to All e-mail messages will have a subject line of "Here is the report you

needed" and the message "Attached is the report you had asked for!" will appear in the body of the e-mail.

```
{
 "EmailResults": true,
 "AttachOutputToEmail": true,
 "ToAddresses": [
  "[email protected]",
  "[email protected]"
 ],
 "CcAddresses": [
  "[email protected]",
  "[email protected]"
 ],
 "BccAddresses": [
  "[email protected]"
 ],
 "ReplyToAddresses": [],
 "EmailSubject": "Here is that report you needed",
 "EmailBody": "Attached is the report you had asked for!"
}
```

REST – Server Events

Server events can be accessed for reference and disabled for the session. Server events cannot currently be created or edited using REST.

> **IMPORTANT**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted for clarity.

Server Event JSON

Server events are represented as JSON objects with the following properties:

Name	Type	Writable	Description
Id	string	no	The unique Id of this server event
Name	string	no	The name of this server event

> **NOTE**
>
> Access to other properties of server events via REST is not yet implemented.

Example

```
{
 "Id":  "0",
 "Name": "RunAfterReportExecution"
}
```

List Server Events

```
GET /rest/ServerEvents
```

List all the server events in the current configuration. Output is an array of objects, each representing an individual server event.

Name	Type	Description
Id	string	The unique Id of this server event
Name	string	The name of this server event

Using curl

```
curl http://{webservice}/rest/ServerEvents?sid={sid} -X GET
```

Example response

```
Status: 200 OK

[
 {
  "Id":  "0",
  "Name": "RunAfterReportExecution"
 },
 {
  "Id":  "1",
  "Name": "AddDisclaimerToOutput"
 }
]
```

Show Server Event

```
GET /rest/ServerEvents/{Id}
```

Show the properties of the server event specified by its Id.

Using curl

```
curl http://{webservice}/rest/ServerEvents/{Id}?sid={sid} -XGET
```

Example response

```
Status: 200 OK

{
 "Id":  "0",
 "Name": "RunAfterReportExecution"
}
```

Delete Server Event

```
DELETE /rest/ServerEvents/{Id}
```

Using curl

```
curl http://{webservice}/rest/ServerEvents/{Id}?sid={sid} -XDELETE
```

Example response

```
Status: 204 No Content
```

REST – Config Settings

The **Settings** endpoint allows access to many settings of the current configuration and session.

Settings JSON

Settings are represented as JSON objects with the following properties. Each corresponds with a setting in the **Admin Console** and base configuration file. Some settings appear both in the Settings JSON (Table A) and in General Settings JSON (Table B). When this happens, use the Table B setting.

Table A â€" Settings JSON Properties

Name	Type	Writable
General	**General Settings**	yes
DateFormat *Deperecated. Use General/DateFormat instead.*	string	yes
TimeFormat *Deprecated. Use General/TimeFormat instead.*	string	yes
DateTimeFormat *Deprecated. Use General/DateTimeFormat instead.*	string	yes
SeparatorSymbol *Deprecated. Use General/SeperatorSumbol instead.*	string	yes
CurrencySymbol *Deprecated. Use General/CurrencySymbol instead.*	string	yes
DecimalSymbol *Deprecated. Use General/DecimalSymbol insread.*	string	yes
ShowGrid *Deprecated. Use General/IsShowGrid instead.*	boolean	yes
ShowCrossTabReports *Deperecared. Use General/IsShowCrossTabReports instead.*	boolean	yes
ShowExpressReports *Deprecated. Use General/IsShowExpressReports instead.*	boolean	yes

ShowExpressReportsGrouping *Deprested. Use General/IsShowExpressReportsGrouping instead.*	boolean	yes
ShowExpressReportsFormulas *Deprecated. Use General/IsShowExpressReportsFormulaButton instead.*	boolean	yes
ShowExpressReportsStyling *Deperecated. Use General/IsShowExpressReportsStylingToolbar instead.*	boolean	yes
ShowExpressReportsThemes *Deprecated. Use General/IsShowExpressReportsThemes instead.*	boolean	yes
ShowAdvancedReports *Deprecated. Use General/IsShowAdvancedReports instead.*	boolean	yes
ShowScheduleReports *Deperecated. Use General/IsShowScheduleReports instead.*	boolean	yes
ShowScheduleReportsManager *Deperecated. Use General/IsShowScheduleReportsManager instead.*	boolean	yes
ShowScheduleReportsEmail *Deperecated. Use General/IsShowScheduleReportsEmail instead.*	boolean	yes
ReadFilterValues *Deprecated. Use General/IsReadFilterValues instead.*	boolean	yes
ShowTabs	boolean	yes
WebReportsBaseUrl	string	yes
DefaultFolderName	string	yes
DefaultReportName	string	yes

General Settings JSON

Some settings appear both in the Settings JSON (Table A) and in General Settings JSON (Table B). When this happens, use the Table B setting.

Table B – General Settings JSON Properties

Name	Type	Writable
ReportPath	string	yes
AllowHomeDirect	boolean	yes
AllowHtmlOutput	boolean	yes
AllowExcelOutput	boolean	yes
AllowPdfOutput	boolean	yes
AllowRtfOutput	boolean	yes
AllowCsvOutput	boolean	yes
DefaultOutputType	**wrExportType**	yes
IsWriteLog	boolean	yes
CustomCode	string	yes
LicenseKey	string	yes
DateFormat	string	yes
TimeFormat	string	yes
DateTimeFormat	string	yes
DateTimeTreatedAs	integer 0. Date 1. Time 2. DateTime	yes
IsReadFilterValues	boolean	yes
DbTimeout	integer	yes
DbRowLimit	integer	yes
DbRowLimitStepSize	integer	yes
VisualizationDbRowLimit	integer	yes
CallTypeParamName	string	yes
ColumnParamName	string	yes
FilterParamName	string	yes
FullFilterParamName	string	yes
SortParamName	string	yes
DataCategoryParamName	string	yes
ObjectIdParamName	string	yes
IsShowGrid	boolean	yes

ActiveRoleId	string	yes
IsDebugging	boolean	yes
IsShowPdfTemplate	boolean	yes
IsDashboardPromptAtExecution	boolean	yes
IsDashboardShowUrlItemButton	boolean	yes
IsVisualizationShowKeyword	boolean	yes
IsShowChartWizard	boolean	yes
IsShowMapWizard	boolean	yes
IsShowGoogleMapWizard	boolean	yes
IsShowGaugeWizard	boolean	yes
IsSaveOnExecute	boolean	yes
IsSaveOnFinish	boolean	yes
IsEnableRightClickMenus	boolean	yes
IsEnableReportsTreeDragAndDrop	boolean	yes
IsShowLinkReportFields	boolean	yes
IsShowLinkReportFormula	boolean	yes
IsShowLinkAction	boolean	yes
IsShowInsertImage	boolean	yes
ExpirationDateStr	string	yes
IsHtmlPaging	boolean	yes
WindowHeight	integer	yes
WindowWidth	integer	yes
SeparatorSymbol	string	yes
CurrencySymbol	string	yes
DecimalSymbol	string	yes
IsDataMapping	boolean	yes
IsShowJoinFields	boolean	yes
IsDetectJoinedObjects	boolean	yes
IsEnableSpecialCartesianProcessing	boolean	yes
IsIncludeNotFilterNullValues	boolean	yes
TempPath	string	no
TempCloudService	string	yes
DataCloudService	string	yes
IsShowScheduleReports	boolean	yes
IsShowScheduleReportsManager	boolean	yes
ScheduleManagerViewLevel	**Schedule Manager View Level**	yes
IsShowScheduleReportsEmail	boolean	yes
ScheduleRemotingHost	string	yes
SynchronousRemotingHost	string	yes
SchedulerQueueService	string	yes
ReportScheduleEmailSubject	string	yes
ReportScheduleEmailBody	string	yes
IsRenewSession	boolean	yes
PasswordRequirement	string	yes
IsExecuteRemotely	boolean	yes
IsAccessDataSourcesRemotely	boolean	yes
MaxTempFileAge	integer	yes
AllowExecuteNewFilters	boolean	yes
ExternalInterface	string	yes
DecimalPlaces	integer	yes
CurrencyDecimalPlaces	integer	yes
ApplyGeneralFormatDecimalPlaces	boolean	yes
ApplyGeneralCurrencyRightAlignment	boolean	yes
ChartColors	string	yes

GeochartMapKey	string	yes
MapColors	string	yes
GoogleMapKey pre-v2018.1	string	yes
GoogleMapJSAPIKey v2018.1+	string	yes
GoogleMapGeocodeAPIKey v2018.1+	string	yes
GoogleMapColors	string	yes
GaugeColors	string	yes
IsShowGroupFilters	boolean	yes
IsShowJoinsWindow	boolean	yes
IsShowAdvancedJoins	boolean	yes
IsShowEventsWindow	boolean	yes
IsShowReportParametersWindow	boolean	yes
IsShowCrosstabReports	boolean	yes
IsShowDashboardReports	boolean	yes
IsShowChainedReports	boolean	yes
IsAllowExpressViewLiveEdit	boolean	yes
IsShowExpressViews	boolean	yes
IsShowDashboardNewVisualizationButton	boolean	yes
IsShowDashboardDataFieldsSearch	boolean	yes
IsShowExpressViewDataFieldsSearch	boolean	yes
IsUseSampleDataForDashboardVisualizationDesign	boolean	yes
IsDashboardPromptKeywordByDefault	boolean	yes
IsShowExpressReports	boolean	yes
IsShowExpressReportsGrouping	boolean	yes
IsShowExpressReportsFormulaButton	boolean	yes
IsShowExpressReportsStylingToolbar	boolean	yes
IsShowExpressReportsThemes	boolean	yes
IsShowAdvancedReports	boolean	yes
IsShowGroupHeadersFormulaButton	boolean	yes
IsOneCategoryLimit	boolean	yes
IsCacheServices	boolean	yes
SchedulerRecipientWindow	string	yes
FilterExecutionWindow	string	yes
IsShowIeDownloadButton	boolean	yes
IsDeleteSchedulesUponReportDeletion	boolean	yes
AllowMultipleSessions	boolean	yes
AllowMD5OnFIPS	boolean	yes
WriteBOMToCSV v2019.1.1+	boolean	yes
IsShowScheduleNoEndDate	boolean	yes
IsShowScheduleIntradayRecurrence	boolean	yes
TempUrl	string	yes
UserPreferenceStorage	integer	yes
UserStartupReportOverwriteGettingStarted	boolean	yes
UserStartupReportMaxNum	integer	yes
ExcelExportTarget	integer 0. 2003 1. 2007 2. 2010 3. 2013 4. 2016	yes
AllowDependantFilters	boolean	yes
LinkedReportsInNewTab	boolean	yes
LoadImageFuncParamPrefix	string	yes
LanguageFile	string	yes
IsShowHelp	boolean	yes

CustomHelpSource	string	yes
IsEmailScheduledReports	boolean	yes
IsEnableBatchReports	boolean	yes
IsShowScheduleDeliveryTypeOptions	boolean	yes
IsUseSecureRemotingChannel	boolean	yes
IsShowTemplateUploadButton	boolean	yes
UseSVGForAppIcons	boolean	yes
CrossTabMaximumFields	boolean	yes
IsShowExportsInTab	boolean	yes
IsShowHtmlToolbar	integer 1. auto 2. show always 3. hide always	yes
IsEnableScheduling	boolean	yes
IsShowFilterDescription	boolean	yes
DefaultFilterExecutionWindow	integer 0. standard 1. operator 2. simple 3. custom	yes
IsChangeableFilterExecutionWindow	boolean	yes
IgnoreInaccessibleReportFolders	boolean	yes
SchemaAccessType	integer 1. datasource 2. metadata	yes
ServerTimeZoneOffset	decimal **NOTE** Set to *0* to use the **ClientTimeZoneName**. *If null, Exago will use the External Interface (a deprecated extensibility feature) to calculate the time zone change.*	yes
ClientTimeZoneName	string	yes
IsShowEnhancedTooltips	boolean	yes
IsShowReportUploadDownloadOptions	boolean	yes
AllowInteractiveHtml	boolean	yes
AllowHtmlInScheduledEmails	boolean	yes
DefaultIsDockOpen	boolean	yes
DockPlacement	integer 0. left 1. right	yes
CssTheme	string	yes
AllowExecuteSaveToDesign	boolean	yes
UserAllowUserReports	boolean	yes
MaxNumberOfChartPoints	integer	yes
DefaultFont	string	yes
DefaultFontSize	integer	yes
DefaultChartFont	string	yes
IsWebFarmSupport	boolean	yes
IsSilentDashboardRefresh	boolean	yes
MaxJobExecutionMinutes	integer	yes
ChainedReportMaxCollationExecutions v2021.1.8+	integer	yes
IsAggregateAndGroupInDatabase	boolean	yes
IsEvaluateFormulasInDatabase	boolean	yes

IsShowBrowserOutOfDateWarning	boolean	yes
AreDateFunctionsLoaded	boolean	yes
VersionNumber	string	no
DbRowLimitParamName v2018.1	string	yes
DbRowRangeStartParamName v2018.1+	string	yes
DbRowRangeEndParamName v2018.1+	string	yes
ExpressviewJoinAlgorithm v2018.1+	string “Standard” or “Legacy”	yes
ExcludeDataSourcesReportCustomSQL v2018.1+	boolean	yes
LoadAssemblyInExternalDomain v2018.1+	boolean	yes
ShowTipsExpressView v2018.2+)	boolean	yes
ShowTutorialExpressView v2018.2+	boolean	yes
IsShowSQLWindow v2019.1+	boolean	yes
SqlGenerationLevel v2020.1+	integer	yes

Table B

Example:


```
{
  "General": {
    "ReportPath": "C:ExagoReports",
    "AllowHomeDirect": true,
    "AllowHtmlOutput": true,
    "AllowExcelOutput": true,
    "AllowPdfOutput": true,
    "AllowRtfOutput": true,
    "AllowCsvOutput": true,
    "DefaultOutputType": 2,
    "IsWriteLog": true,
    "CustomCode": "",
    "LicenseKey": "",
    "DateFormat": "",
    "TimeFormat": "",
    "DateTimeFormat": "",
    "DateTimeTreatedAs": 0,
    "IsReadFilterValues": true,
    "DbTimeout": 600,
    "DbRowLimit": 0,
    "DbRowLimitStepSize": 1000,
    "VisualizationDbRowLimit": 0,
    "CallTypeParamName": "",
    "ColumnParamName": "",
    "FilterParamName": "",
    "FullFilterParamName": "",
    "SortParamName": "",
    "DataCategoryParamName": "",
    "ObjectIdParamName": "",
    "IsShowGrid": true,
    "ActiveRoleId": "",
    "IsDebugging": false,
    "IsShowPdfTemplate": true,
    "IsDashboardPromptAtExecution": false,
    "IsDashboardShowUrlItemButton": false,
    "IsVisualizationShowKeyword": false,
    "IsShowChartWizard": true,
    "IsShowMapWizard": false,
    "IsShowGoogleMapWizard": false,
    "IsShowGaugeWizard": true,
    "IsSaveOnExecute": true,
    "IsSaveOnFinish": true,
    "IsEnableRightClickMenus": true,
    "IsEnableReportsTreeDragAndDrop": true,
    "IsShowLinkReport": true,
    "IsShowLinkReportFields": true,
    "IsShowLinkReportFormula": false,
    "IsShowLinkAction": false,
    "IsShowInsertImage": true,
    "ExpirationDateStr": "",
    "IsHtmlPaging": true,
    "WindowHeight": 0,
    "WindowWidth": 0,
    "SeparatorSymbol": ",",
    "CurrencySymbol": "$",
```

```
"CurrencySymbol": "$",
"DecimalSymbol": ".",
"IsDataMapping": false,
"IsShowJoinFields": true,
"IsDetectJoinedObjects": true,
"IsEnableSpecialCartesianProcessing": true,
"IsIncludeNotFilterNullValues": false,
"TempPath": "",
"TempCloudService": "",
"DataCloudService": "",
"IsShowScheduleReports": false,
"IsShowScheduleReportsManager": false,
"ScheduleManagerViewLevel": 0,
"IsShowScheduleReportsEmail": false,
"ScheduleRemotingHost": "",
"SynchronousRemotingHost": "",
"SchedulerQueueService": "",
"ReportScheduleEmailSubject": "",
"ReportScheduleEmailBody": "",
"IsRenewSession": true,
"PasswordRequirement": "",
"IsExecuteRemotely": false,
"IsAccessDataSourcesRemotely": false,
"MaxTempFileAge": 1440,
"AllowExecuteNewFilters": true,
"ExternalInterface": "",
"DecimalPlaces": -1,
"CurrencyDecimalPlaces": -1,
"ApplyGeneralFormatDecimalPlaces": false,
"ApplyGeneralCurrencyRightAlignment": true,
"ChartColors": "#e85d61, #e5d08f, #00d995",
"GeochartMapKey": "",
"MapColors": "Lightblue,Navy",
"GoogleMapKey": "",
"GoogleMapColors": "Lightblue,Navy",
"GaugeColors": "#e85d61, #e5d08f, #00d995",
"IsShowGroupFilters": true,
"IsShowJoinsWindow": true,
"IsShowAdvancedJoins": true,
"IsShowEventsWindow": false,
"IsShowReportParametersWindow": true,
"IsShowCrosstabReports": true,
"IsShowDashboardReports": true,
"IsShowChainedReports": true,
"IsAllowExpressViewLiveEdit": false,
"IsShowExpressViews": true,
"IsShowDashboardNewVisualizationButton": true,
"IsShowDashboardDataFieldsSearch": false,
"IsShowExpressViewDataFieldsSearch": true,
"IsUseSampleDataForDashboardVisualizationDesign": true,
"IsShowDashboardExistingVisualizationReportButton": false,
"IsDashboardPromptKeywordByDefault": false,
"IsShowExpressReports": true,
"IsShowExpressReportsGrouping": true,
"IsShowExpressReportsFormulaButton": true,
"IsShowExpressReportsStylingToolbar": true,
"IsShowExpressReportsThemes": true,
"IsShowAdvancedReports": true,
"IsShowGroupHeadersFormulaButton": false,
"IsOneCategoryLimit": false,
"IsCacheServices": true,
"SchedulerRecipientWindow": "",
"FilterExecutionWindow": "",
"IsShowIeDownloadButton": false,
"IsDeleteSchedulesUponReportDeletion": false,
"AllowMultipleSessions": true,
"AllowMD5OnFIPS": false,
"WriteBOMToCSV": false,
"IsShowScheduleNoEndDate": true,
"IsShowScheduleIntradayRecurrence": true,
"TempUrl": "",
"UserPreferenceStorage": 0,
"UserStartupReportOverwriteGettingStarted": false,
"UserStartupReportMaxNum": 1,
"ExcelExportTarget": 1,
"AllowDependantFilters": false,
"LinkedReportsInNewTab": false,
"LoadImageFuncParamPrefix": "",
"LanguageFile": "en-us,en-us-getting-started,en-us-tooltips",
"IsShowHelp": true,
"CustomHelpSource": "",
```

```
  "IsEmailScheduledReports": true,
  "IsEnableBatchReports": false,
  "IsShowScheduleDeliveryTypeOptions": false,
  "IsUseSecureRemotingChannel": false,
  "IsShowTemplateUploadButton": false,
  "UseSVGForAppIcons": true,
  "CrossTabMaximumFields": 5,
  "IsShowExportsInTab": true,
  "IsShowHtmlToolbar": 0,
  "IsEnableScheduling": true,
  "IsShowFilterDescription": false,
  "DefaultFilterExecutionWindow": 0,
  "IsChangeableFilterExecutionWindow": true,
  "IgnoreInaccessibleReportFolders": false,
  "SchemaAccessType": 1,
  "ServerTimeZoneOffset": 0,
  "ClientTimeZoneName": "America/New_York",
  "IsShowEnhancedTooltips": true,
  "IsShowReportUploadDownloadOptions": false,
  "AllowInteractiveHtml": true,
  "AllowHtmlInScheduledEmails": false,
  "DefaultIsDockOpen": false,
  "DockPlacement": 0,
  "CssTheme": "Basic",
  "AllowExecuteSaveToDesign": true,
  "UserAllowUserReports": true,
  "MaxNumberOfChartPoints": 300,
  "DefaultFont": "Arial",
  "DefaultFontSize": 8,
  "DefaultChartFont": "Arial",
  "IsWebFarmSupport": false,
  "IsSilentDashboardRefresh": false,
  "MaxJobExecutionMinutes": 240,
  "IsAggregateAndGroupInDatabase": false,
  "IsEvaluateFormulasInDatabase": true,
  "IsShowBrowserOutOfDateWarning": true,
  "AreDateFunctionsLoaded": true,
  "VersionNumber": "2016.3.4.100"
 },
 "DateFormat": "",
 "TimeFormat": "",
 "DateTimeFormat": "",
 "SeparatorSymbol": ",",
 "CurrencySymbol": "$",
 "DecimalSymbol": ".",
 "ShowGrid": true,
 "ShowCrossTabReports": true,
 "ShowExpressReports": true,
 "ShowExpressReportsGrouping": true,
 "ShowExpressReportsFormulas": true,
 "ShowExpressReportsStyling": true,
 "ShowExpressReportsThemes": true,
 "ShowAdvancedReports": true,
 "ShowScheduleReports": false,
 "ShowScheduleReportsManager": false,
 "ShowScheduleReportsEmail": false,
 "ReadFilterValues": true,
 "ShowTabs": true,
 "WebReportsBaseUrl": null,
 "DefaultFolderName": "",
 "DefaultReportName": ""
}
```


> **NOTE**
>
> All requests require a Session ID URL parameter and basic request headers. In the following document, headers are omitted in the interest of brevity.

Show All Settings

.Show the available settings for the current configuration.

```
GET /rest/Settings
```

Using curl

```
curl http://{webservice}/rest/Settings?sid={sid} -X GET
```

Example response

```
Status: 200 OK

{
  "General": {
    "ReportPath": "C:ExagoReports",
     ...
  }
  "DateFormat":              "MM/dd/yyyy",
  "TimeFormat":              "hh:mm:ss a",
  "DateTimeFormat":          "MM/dd/yyyy hh:mm:ss a",
  "SeparatorSymbol":         ",",
  "CurrencySymbol":          "$",
  "ShowGrid":                true,
  "ShowCrossTabReports":     true,
  "ShowExpressReports":      true,
  ...
}
```

Edit Settings

PATCH /rest/Settings

Only supply the properties to be edited

Using curl

```
curl http://{webservice}/rest/settings?sid={sid} -XPATCH ^
-d "{
        'General':
        {
          'ReportPath':'C:ExagoReports',
          'IsShowLinkReportFormula':false,
           ...
        }
        'ShowScheduleReports':false,
        'ShowExpressReports':false,
        ...
       }"
```

Example response

```
Status: 204 No Content
```

Constants and Enumerators

There are many variables in the APIs that require a specific type of value from a set of values. These sets of values are called e*numerated types*, and each value in a type is called an enumerator, or *enum*. Enumerated types are no more than a set of named values, which act as Constants in the API.

When an API variable requires an enum, it must be set to one of the constants that are defined in the specified enumerated type. For example, if a variable requires a Report Type enum, it accepts the values *Advanced*, *Express*, *Dashboard*, *Chained* or *ExpressView*. Other values will cause an invalid type error.

Some API variables are not enumerated types, but may require a set of constants regardless. These constants have no numeric equivalent.

> **CAUTION**
>
> Numeric values should not be used as enum types as these values are not constants.

The following article is a reference guide that lists the most important enumerated types and lists of constants that are used throughout the API.

wrApiAction

Enum Name	Numeric Value	Description
Default	0	If a report is loaded, this is equivalent to ExecuteReport. Otherwise, this is equivalent to Home.
Home	1	Open the home page.
ExecuteReport	2	Execute the loaded report (specify format with wrExportType).
EditReport	3	Open the loaded report in the appropriate editor.
NewAdvancedReport	4	Open the New Advanced Report Wizard.
NewReport	4	*Deprecated*: Use NewAdvancedReport instead.
NewCrossTabReport	5	Open the New CrossTab Report Wizard.

Enum Name	Numeric Value	Description
NewExpressReport	6	Open the New Express Report Wizard.
NewDashboardReport	7	Open the Dashboard Designer for a new dashboard.
NewExpressView	8	Open the ExpressView Designer for a new ExpressView.
ScheduleReport	9	Open the Schedule Report Wizard for the loaded report.
ScheduledReportsManager	10	Open the Scheduled Report Manager.

Cache Visibility Level

Enum Name	Numeric Value	Description
User	1	Cache visibility set at the UserId parameter level
Company	2	Cache visibility set at the CompanyId parameter level
Global	3	Global cache visibility

Code Language

> **NOTE**
>
> Each value has multiple names.

Enum Name	Numeric Value	Description
c# cs csharp	0	C Sharp
js javascript	1	JavaScript
vb visualbasic	2	Visual Basic

ConfigStorageType v2021.2+

Used in the Configuration Management system to describe the type of configuration storage mechanism in use. Most users encounter these values when creating or editing a bootstrap configuration file.

Enum Name	Numeric Value	Description
FileSystem	0	Configuration files are stored on the server's file system. This is the pre-v2021.2 behavior.
Database	1	Configuration files are stored in a Storage Management database. This is the recommended value for all installations.
Cloud	2	Configuration files are stored on a cloud service such as Amazon S3 or Azure.

wrContentType v2020.1+

Enum Name	Numeric Value	Description
Report	0	Item is a report.
Folder	1	Item is a folder.
Theme	2	Item is a chart or map theme.
Template	3	Item is a template file.
All	99	Reference all content types.

ContentPermission v2021.1+

The **ContentPermission** enumeration was updated in *v2021.1*. Review the Access Flags part of the Permissions section of the Storage Management: Introduction article for more detailed descriptions of each of these access flags and their impact on the user experience.

> **TIP**
>
> To determine the content_access.access_flags value, sum the decimal values of the desired flags from the table below. For example, to only allow renaming and deleting, the value is *20* since *4+16=20*.

Enum Name	Numeric Value	Description
None	Hex: 0x0000 Dec: 0	User has no access to the content.
CanEdit	Hex: 0x0001 Dec: 1	User can edit the content.
Reserved	Hex: 0x0002 Dec: 2	This enum is reserved for future use.

CanRename	Hex: 0x0004 Dec: 4	User can rename the content.
CanShare	Hex: 0x0008 Dec: 8	User can share the content.
CanDelete	Hex: 0x0010 Dec: 16	User can delete the content.
Reserved	Hex: 0x0020 Dec: 32	This enum is reserved for future use.
CanCopy	Hex: 0x0040 Dec: 64	User can duplicate the content.
Reserved	Hex: 0x0080 Dec: 128	This enum is reserved for future use.
CanView	Hex: 0x0100 Dec: 256	The content appears in the user's Report Tree.
CanSchedule	Hex: 0x0200 Dec: 512	The user may schedule or e-mail the report.
CanMove	Hex: 0x0400 Dec: 1024	The user may move this item to another folder.
FullAccess	Hex: 0xFFFF Dec: 65535	All access is granted.
ReadOnly	Hex: 0x0348 Dec: 840	The user may view, copy, schedule it and share the item but nothing else.

Data Source Type

Enum Name	Numeric Value	Description
MsSql	0	Microsoft SQL Server
MySql	1	MySQL
ODBC	2	ODBC
Postgres	3	PostreSQL
Oracle	4	Oracle
DB2	5	IBM DB2
Informix	6	IBM Informix
Assembly	7	.NET Assembly
WebService	8	Web Service
File	9	Excel file
MsOlap	10	OLAP
Sqlite	11	SQLite v2021.1.8+

DataType

The **DataType** enumeration is used in the REST Web Service API when processing data object column metadata and when getting the the value of the EntityColumn.DataTypeEnum property in the .NET API.

Enum Name	Numeric Value	Description
String	0	String
Date	1	Date, i.e. day, month, year
DateTime	2	Date & time
Time	3	Time, i.e. hour, minute, second (am/pm)
Integer	4	Integer number
Decimal	5	Decimal number
Float	6	Floating-point number
Bit	7	Bit
Guid	8	Globally (universally) unique identifier, aka. UUID
Image	9	Image
Currency	10	Currency

DataType

The **DataType** constants are used throughout the application including when defining the return type (the <return_data_type>) of Quick Functions.

Constant Name	Integer Value	Description
StringDataType	0	A string
DateDataType	1	A date

IntegerDataType	2	An integer
OldBitDataType	3	Deprecated â€" a single bit. Users should use BitDataType (13) instead
NumericDataType	4	A number
FloatDataType	5	A floating point number
DecimalDataType	6	A decimal number
GuidDataType	7	Globally (universally) unique identifier, aka. UUID
DateTimeDataType	8	A DateTime
TimeDataType	9	Time â€" this data type is currently not in use in the application
ImageDataType	10	An binary image
CurrencyDataType	11	A currency value
AmbiguousFormulaDataType	12	A formula returns a different data type based on its outcome (e.g. =if(temperature > 98.6, temperature, false)
BooleanDataType	13	A boolean (true/false) value
BitDataType	13	A single bit

DataObjectType

Constant Name	Description
Assembly	.NET Assembly
File	Excel file
Function	User-defined function
Procedure	Stored procedure
SqlStmt	Custom SQL statement
Table	Data table
View	SQL view
WebSvc	Web Service method
MdxStmt	OLAP MDX query

DashboardLayoutFit

Enum Name	Description
All	All tiles are resized horizontally and vertically to fit on the canvas Same as selecting **Scale to Screen** in the Dashboard Designer Canvas Format menu
Width	Tiles are resized horizontally to fit within a specified canvas width Same as selecting **Specify Height** in the Dashboard Designer Canvas Format menu
None	The tiles are not resized. The canvas has a specified height and width Same as selecting **Specify Height and Width** in the Dashboard Designer Canvas Format menu

wrDockLocation

Used to set the location of the interactive dock in the Report Viewer with the .NET API's SetupData.General.DockPlacement setting.

Enum Name	Numeric Value	Description
Left	0	The pane is docked on the left.
Right	1	The pane is docked on the right.

ExecuteDataType v2021.1+

Used in the REST Web Service API GetExecute() methods to determine how to return the data to the host application.

Enum Name	Numeric Value	Description
Auto	0	Automatically determine the apprpriate return data action. This is the default if no other value is explicity chosen.
Data	1	Return a base64 encoded representation of the output file. To display the output data, process the information from the ExecuteData property and send it to the browser. When deploying Exago in a web farm architecture, this is the only acceptable value when using GetExecute() methods.
Url	2	Return a URL that can be passed back to the browser. The browser can then display or download the resulting file.

ExportFlag v2020.1+

Used in the Storage Management System.

Enum Name	Numeric Value	Description
Default	0	Use the system default export type.
None	0	There is no default export type, so revert to the system default.
Html	1	Run in the browser, using the appropriate viewer.
Pdf	2	Export as a PDF file.
Rtf	4	Export as an RTF file.
Csv	8	Export as a comma-separated value text file.
Excel	16	Export as a Microsoft Excel workbook.

wrExportType

Enum Name	Numeric Value	Description
Html	0	Run in the browser, using the appropriate viewer.
Excel	1	Export as an Excel file.
Pdf	2	Export as a PDF file.
Rtf	3	Export as an RTF file.
Csv	4	Export as a comma-separated value text file.
Word	5	Export as a Word file.
Default	6	Default type specified by the loaded report; if not specified, default type specified by the current config.
PdfSnapshot	10	Export a Dashboard screenshot as a PDF file. See Exporting Dashboards (v2019.2+) for more information.

Filterable Type

Enum Name	Description
All	Field supports all filter types
True	Field supports all filter types
None	Field cannot be filtered
False	Field cannot be filtered
Dynamic	Field supports only dynamic (interactive) filters
Static	Field supports only static (report) filters

Filter Operator Type

Enum Name	Numeric Value	Description
EqualTo	0	Data value is equal to the filter value
NotEqualTo	1	Data value is less than the filter value (number, date)
LessThan	2	Data value is less than or equal to the filter value (number, date)
LessThanOrEqualTo	3	Data value is greater than the filter value (number, date)
GreaterThan	4	Data value is greater than or equal to the filter value (number, date)
GreaterThanOrEqualTo	5	Data value is not equal to the filter value
StartsWith	6	Data value starts with the filter value (string, number)
NotStartsWith	7	Data value does not start with the filter value (string, number)
EndsWith	8	Data value ends with the filter value (string, number)
NotEndsWith	9	Data value does not end with the filter value (string, number)
Contains	10	Data value contains the filter value (string, number)
NotContains	11	Data value does not contain the filter value (string, number)
Between	12	Data value is between the two filter values
NotBetween	13	Data value is not between the two filter values
OneOf	14	Data value is equal to one of the filter values
NotOneOf	15	Data value is not equal to any of the filter values

FolderStatus

Enum Name	Description
Exists	This folder exists
DoesNotExist	This folder does not exist
Created	This folder was successfully created

Deleted	This folder was successfully deleted
Renamed	This folder was successfully renamed

JobStatus v2019.2+

A read-only property showing the current status of a schedule. Used in REST – SchedulesV2.

Enum Name	Numeric Value	Description
Ready	0	The schedule will run at its next scheduled run time.
Running	1	The schedule is currently running and the reports are executing.
Completed	2	The schedule has completed its final event, and will not run again. It will be removed from the list when the scheduler is flushed.
Suspended	3	The schedule has been suspended and will not be run again.
Deleted	4	The schedule has been deleted, and will be removed from the list when the cache is flushed.
Abended	5	The last run failed due to an error. The schedule will not run again.
UserAbort	6	Schedule is running, but user has requested to delete it.
Transmitting	7	Schedule is finished and waiting for data to be transmitted to client for synchronous execution.
Removed	8	Schedule should be removed if it hasn't already.
Immediate	9	As soon as resources are available, the schedule will be run.
Timedout v2019.2.32+ v2020.1.15+ v2021.1.3+	10	The job has hit the execution time out set in the Scheduler Service's configuration file.

Join Type

Enum Name	Description
Inner	Inner join
LeftOuter	Left outer join
RightOuter	Right outer join
FullOuter	Full outer join (left outer join + right outer join)

Join Relation Type

Enum Name	Description
OneToOne	One-to-one relationship
OneToMany	One-to-many relationship

Parameter Type

Enum Name	Numeric Value	Description
String	0	String
Date	1	Date, i.e. day, month, year
Integer	2	Integer number
Decimal	3	Decimal number

QuickFunctionDataType v2021.1+

Used for defining the input type (the <supported_data_types>) of a Quick Function.

Enum Name	Description
Date	A Date or DateTime
Number	A numeric value
Text	Text

wrRowRangeSqlMethod

Used in the dbconfigs.json file. See Managing the dbconfigs.json File.

Enum Name	Description
None	Disable range limiting SQL.
LimitOffset	SELECT ... LIMIT rangeSize OFFSET offsetRow
OffsetFetch	SELECT ... OFFSET offsetRow ROWS FETCH NEXT rangeSize ROWS ONLY

RowNumberWithTop	SELECT TOP endingRow ROW_NUMBER() Over(Order By (...)) as wrRowNum ... as wrRowNumbered WHERE wrRowNumbered.wrRowNum BETWEEN startingRow AND endingRowField
RowNumber	SELECT ROW_NUMBER() Over(Order By (...)) as wrRowNum ... as wrRowNumbered WHERE wrRowNumbered.wrRowNum BETWEEN startingRow AND endingRowField
RowNumberNoAlias	SELECT ROW_NUMBER() Over(Order By (...)) as wrRowNum ... WHERE wrRowNum BETWEEN startingRow AND endingRowField

RecurrenceRange v2019.2+

Determines the ending of a scheduled report recurrence. Used in REST – SchedulesV2.

Enum Name	Description
NoEndDate	This schedule has no end date and will recur indefinitely.
EndAfterNOccurences	This schedule shall run only for a specified number of instances. The number of instances is specified in the OccurencesToEnd property.
EndByDate	This schedule shall run until a specified date is reached. The date is specified in the RangeEndDate property.

RecurrenceType v2019.2+

Determines the frequency of scheduled report recurrence. Used in REST – SchedulesV2.

Enum Value	Description
Once	This schedule will only run one time.
Daily	The schedule will repeat every day, every weekday, or every number of days.
Weekly	The schedule will repeat every week, or every number of weeks, on one or more days
Monthly	The schedule will repeat every month, or every number of months, on a certain day.
Yearly	The schedule will repeat every year on a certain day.

wrReportType

Enum Name	Numeric Value	Description
Advanced	0	Advanced Report (CrossTab Reports are also considered Advanced Reports)
Standard	0	*Deprecated*: Use Advanced instead.
Express	1	Express Report
Dashboard	2	Dashboard
Chained	3	Chained Report
ExpressView	5	ExpressView
Folder	6	Report Folder

ReportValidationErrorType

The types of report validation errors that can happen. Not all validation error types are applicable to all report objects.

Enum Name
SortFormulaSyntaxError
FilterFormulaSyntaxError
DataObjectNotFound
SortDataFieldNotFound
FilterDataFieldNotFound
LinkedDataFieldNotFound
MinMaxFilterDataFieldNotFound
JoinDataObjectNotFound
JoinNotFound
JoinDataFieldNotFound
ChartDataFieldNotFound
MapDataFieldNotFound
CellDataFieldNotFound
RowGroupNameNotFound
RowGroupFormulaNameNotFound
ChartCellIdNotFound
MapCellIdNotFound
MergedCellsAcrossSections

CrossTabIdNotFound
CrossTabCellIdNotFound
ColumnSortByFieldNotFound
ChildReportNotFound
ExpressViewColumnMissingEntity
ExpressViewColumnMissingField
ExpressViewGroupMissingField
ExpressViewFilterMissingField
ExpressViewChartMissingField
EtlReportDesignerModeNotEtl
EtlReportTypeNotAdvanced
EtlDataTypeNotSet
EtlDataTypeNotFound
EtlReportFormatInvalid
EtlCellBlank
EtlHeaderCellValueDuplicate
EtlPossibleDataTypeError
EtlCellSpan
None
ReportParameterNameUsed

ScheduleExportType v2019.2+

The export type for a report run on a schedule. Used in REST – SchedulesV2.

Enum Name	Description
Pdf	Export as a PDF file.
Rtf	Export as an RTF file.
Excel	Export as a Microsoft Excel workbook file.
Csv	Export as a CSV text file.

ScheduleFilterOperator v2019.2+

Utilized when adding filters to scheduled reports via the REST Web Service API. Used in REST – SchedulesV2.

Enum Name	Description
EqualTo	Evaluates as *true* when **FilterText** is equal to **Values**, otherwise *false*.
NotEqualTo	Evaluates as *true* when **FilterText** is not equal to **Values**, otherwise *false*.
LessThan	Evaluates as *true* when **FilterText** is less than **Values**, otherwise *false*.
LessThanOrEqualTo	Evaluates as *true* when **FilterText** is less than or equal to **Values**, otherwise *false*.
GreaterThan	Evaluates as *true* when **FilterText** is greater than **Values**, otherwise *false*.
GreaterThanOrEqualTo	Evaluates as *true* when **FilterText** is greater than or equal too **Values**, otherwise *false*.
StartsWith	Evaluates as *true* when **FilterText** starts with **Values**, otherwise *false*.
NotStartsWith	Evaluates as *true* when **FilterText** does not start with **Values**, otherwise *false*.
EndsWith	Evaluates as *true* when **FilterText** ends with **Values**, otherwise *false*.
NotEndsWith	Evaluates as *true* when **FilterText** does not end with **Values**, otherwise *false*.
Contains	Evaluates as *true* when **Values** is contained within **FilterText**, otherwise *false*.
NotContains	Evaluates as *true* when **Values** is not contained within **FilterText**, otherwise *false*.
Between	Evaluates as *true* when **FilterText** is between two values contained in **Values**, otherwise *false*.
NotBetween	Evaluates as *true* when **FilterText** is not between two values contained in **Values**, otherwise *false*.
OneOf	Evaluates as *true* when **FilterText** matches at least one of **Values**, otherwise *false*.
NotOneOf	Evaluates as *true* when **FilterText** does not match any of **Values**, otherwise *false*.

ScheduleJobType v2019.2+

A read-only property that shows the type of job assigned to a schedule. Used in REST – SchedulesV2.

Enum Name	Description

Report	This job runs a report.
Batch	This job runs a batch report.
Email	This job handles the disposition of a **Report** or **Batch** job by sending the output as an e-mail.
Disk	This job handles the disposition of a **Report** or **Batch** job by saving the output to disk.

ScheduleRecurrenceType v2019.2+

A read-only property that shows the type of job recurrence assigned to a schedule. Used in REST – Schedules.

Enum Name	Description
Once	This schedule only runs once.
Daily	This schedule recurs daily.
Weekly	This schedule recurs weekly.
Monthly	This schedule recurs monthly.
Yearly	This schedule recurs yearly.

ScheduleManagerViewLevel

Enum Name	Numeric Value	Description
User	1	Filter schedules by current UserId parameter
Company	2	Filter schedules by current CompanyId parameter
All	3	Show all schedules

TreeShortcut

Used in the Storage Management System.

Enum Name	Numeric Value	Description
Default	0	The report will take the default action defined in the system's configuration.
Run	1	The report will run in the Report Viewer
Export	2	The report will be exported

wrUserMessageType

Used when Displaying User Messages.

Enum Name	Numeric Value	Description
Id	0	The user message will display the string associated with the Id in the Language Files.
Text	1	The user message will display the string message

wrUserPreferenceStorage

Used when setting the **User Preference Storage Method** setting programatically.

Enum Name	Numeric Value	Description
Cookie	0	User preferences are stored in the user's local browser storage.
ExternalInterface	1	User Preferences are stored and retrieved via an External Interface. This refers to a deprecated extensibility feature that is no longer supported. It should not be used in new installations.
ServerEvent	2	User Preferences are stored and retrieved via Server Events.
None	3	Users cannot modify User Preference features, and the user interface icon is hidden.

Executing Reports with the API

> **NOTE**
>
> This version of the .NET API documentation is deprecated. A new version can be found at https://www.exagoaccess.com/api-docs/.

There are two different ways to use the Exago APIs to perform report execution. This guide discusses the main differences and provides examples for both types in each API.

- **API Action** is the most comprehensive way to run executions, and supports all types of reports. This is the only way to run composite reports such as Dashboards and Chained Reports. This method launches an Exago session into the browser via URL, and thus usually requires the use of an Iframe. This also means that all interactive Report Viewer or ExpressView (*v2016.3+*) features are supported.
- **GetExecute()** executes reports on the back-end and returns bare HTML, JSON, CSV, or binary data. GetExecute supports non-composite

(Advanced, Express and CrossTab) reports and ExpressView (as of *v20191.31+*, *2019.2.19+* and *v2020.1.5+*). Using this method you do not have to launch any visible instance of Exago for the user, and can simply use it as a calculation engine.

Overview

	Launch Method	Supported Report Types	.NET Supported Output Types	REST Supported Output Types
API Action	Redirect browser frame to Exago session URL	All types	Interactive Report Viewer, Dashboard Viewer, ExpressView designer, or PDF, RTF, Excel, CSV	
GetExecute	Data returned directly to calling method	Advanced, Express, CrossTab, ExpressView*	HTML, CSV, PDF, RTF, Excel, JSON	HTML, CSV, PDF, RTF, Excel, JSON

* GetExecute() for ExpressView is supported in *v20191.31+*, *2019.2.19+* and *v2020.1.5+* only.

API Action

API sessions in Exago have a property called *action type*, which determines what part of Exago should be launched when the session is opened. Action types include executing a report, loading a report into the editor, loading a report into the scheduler, opening a section of the UI, etc.

> **NOTE**
>
> API Action is also referred to as *GetUrlParamString*, because this is the general term for the methods which return the session redirect URL.

To tell the session to execute a report, set the action type to **ExecuteReport**.

Actions which load reports, such as Execute or Edit, work on the *active report* object. This is another property that must be set. This is done differently for each API: details are in the included examples.

> **IMPORTANT**
>
> For security reasons, always set the **action type** and the **active report** explicitly. Although setting an active report defaults to execute, if a report fails to load and an action has not been specified, Exago will launch into the full Web Application user interface. This could cause users to have unintended levels of access.

Once you've finished setting the session variables, call GetUrlParamString(). This finalizes the session and creates a single-use URL string. This is done differently for each API; details are in the included examples. The URL is used to direct a browser window or iFrame to an Exago instance where the specified action takes place. The user can then interact with the report like normal.

See the following sections for examples. Variable names and arguments are placeholders.

.NET API

```
//Create an API object and load a report object into it
Api myApi = new Api(appPath);
Report myReport = (Report)myApi.ReportObjectFactory.LoadFromRepository(reportPath);
```

```
//Set the desired output type.
myReport.ExportType = wrExportType.Html;
```

```
//Save the modified report back to the API object
myApi.ReportObjectFactory.SaveToApi(myReport);
```

```
//Set the API Action to execute the report
myApi.Action = wrApiAction.ExecuteReport;
```

```
//Call GetUrlParamString() to close the session and retrieve the URL for it
string url = netApi.GetUrlParamString(homePage);
```

REST Web Service API

To use REST, create the session and pass the session variables. Take note of the sessionId and UrlParamString.

POST /sessions

Payload:

```
{
â€ƒ"Page": homePage,
â€ƒ"ApiAction": "ExecuteReport",
â€ƒ"ExportType": "Html",
â€ƒ"ReportSettings":
â€ƒ{
â€ƒâ€ƒ"ReportPath": reportPath
â€ƒ}
}
```

Response (*some params omitted*):

```
{
â€ƒ"sid": sessionId,
â€ƒ"AppUrl": urlParamString
}
```

GetExecute

There are four provided *GetExecute* methods. Each return a different data representation of the report. Not every API supports every data type; see the **Overview** for details.

- **GetExecuteHtml:** Typically used for web viewing
- **GetExecuteCsv:** Plain-text format readable by spreadsheet applications
- **GetExecuteData:** Byte array of binary data
- **GetExecuteJson** (*v2016.3+*): Typically used for asynchronous client-server communication

Since GetExecute does not require loading the Exago UI, calling *GetUrlParamString* is not required. Report interactivity is also not supported when executing reports via the GetExecute functions. However, the following Server Events are triggered in order to ensure that the data returned accurately reflects the report design:

- **OnReportExecuteStart**
- **OnReportExecuteEnd**
- **OnAfterReportExportSuccess**

See the following sections for examples. Variable names and arguments are placeholders.

.NET

```
//First create an API session and load a report object.
Api myApi = new Api(appPath);
Report myReport = (Report)myApi.ReportObjectFactory.LoadFromRepository(reportPath);
```

```
//Then call the appropriate GetExecute() method for the desired data type.
string reportHtml = myReport.GetExecuteHtml();
```

REST Web Service API

When using the REST Web Service API, the initialization call creates a session ID string, which is a required URL parameter in all subsequent method calls. Note that the session URL is generated immediately, and altered dynamically as modifications are made to the session.

1. First create an API session. Take note of the session ID.**POST /sessions**

   Response (*some params omitted*):

   ```
   {
   â€ƒ"Id": sessionId
   }
   ```

2. Then execute the selected report. Supported types are HTML, CSV, PDF, RTF, Excel, JSON.**POST /reports/execute/{type}?sid=”sessionId”**

   Payload:

   ```
   {
   â€ƒ"ReportPath": "InventoryCategories",
     "DataType" : "Data"
   }
   ```

   Sample Response:

   ```
   Status: 200 OK


   {
     "ReportPath":  "InventoryCategories",
     "ExportType":  "csv",
     "DataType":    "Data",
     "ExecuteData": ""Categories"rn"Seafood"rn"Produce"rn"Beverages"rn",
     "IsError":     false,
     "ErrorList":   null
   }
   ```

For more information, refer to the REST â€” GetExecute article.

Excel: Grouping on Separate Worksheets

The default behavior when exporting an report to a Microsoft Excel workbook is that all data appears in a single worksheet (tab).

If a report contains Group Sections, an Excel export can optionally create a new worksheet for each new instance of the outermost group.

The name of each worksheet will be the data field on which the group breaks. If the group breaks on a data category, the **Unique Key Field** from the Data Object's configuration will be used to name the worksheet.

> **NOTE**
>
> This feature only applies to Excel export file outputs. To display groups on different pages in PDF and RTF export file outputs, add a Page Break in the report design.

Implementation

The report must contain at least one **Group Header** section. Elements preceding the first group header will appear on the first worksheet as normal. Each time a new group header is reached for the outermost group, including the very first one, the workbook will create a new worksheet with the contents of that group.

There are two ways to enable grouping on separate worksheets. Either:

- Use this .NET API property of the Report class:

```
myReport.GroupsOnSeparateWorksheets = true;
```

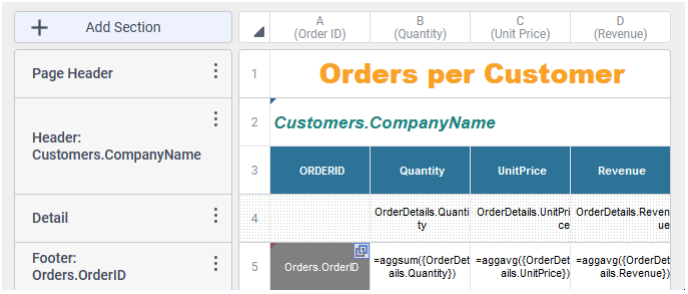
- Set the following property in the <main> section of the report's XML definition to *True*:

```
<groups_on_separate_worksheets>True</groups_on_separate_worksheets>
```

> **CAUTION**
>
> Search the XML for this node first, adding it to a report that already has it will break the report and cause errors to appear in the user interface.

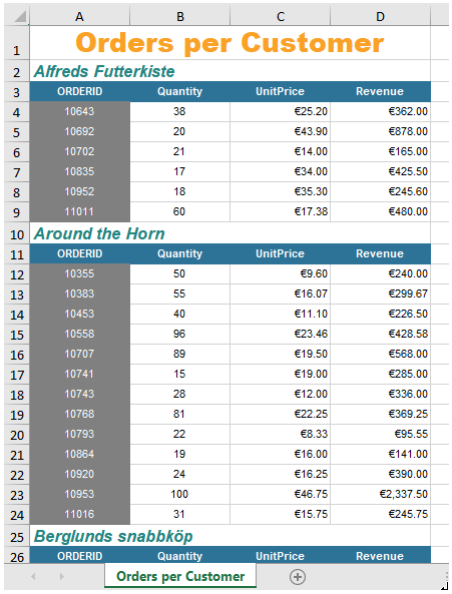
Example

Consider a report that displays summary data of the orders made by each customer of the organization. The report below includes a **Group Header** section that breaks on the sort field Customers.CompanyName.



Group On Separate Worksheets Disabled (Default)

If this report is exported to an Excel workbook, the resulting output looks like something like this:

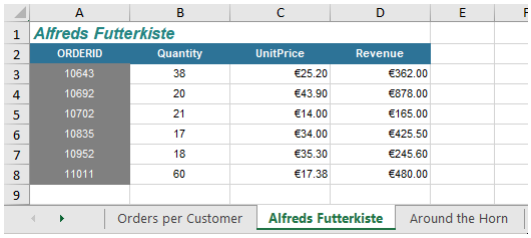


Note that the entire report, including multiple customer company names (the outermost group header), is displayed on a single tab (worksheet),

and that the name of the worksheet is the title of the report: *Orders per Customer*.

Group On Separate Worksheets Enabled

By enabling the **Group on Separate Worksheets** property on this report, the resulting output looks like something like this:



	A	B	C	D	E	F
1	*Alfreds Futterkiste*					
2	ORDERID	Quantity	UnitPrice	Revenue		
3	10643	38	€25.20	€362.00		
4	10692	20	€43.90	€878.00		
5	10702	21	€14.00	€165.00		
6	10835	17	€34.00	€425.50		
7	10952	18	€35.30	€245.60		
8	11011	60	€17.38	€480.00		
9						

Note that the first worksheet is still named the title of the report, *Orders per Customer* but now there is a new worksheet created each time a new Customer's Company Name occurs, one worksheet for each customer's name.

JavaScript API

The Exago **JavaScript (JS) API** allows Exago functionality to be embedded directly into HTML containers.

s can inherit code and styles from the host application. Since CSS cascades down to Exago as well as up to the parent app, a single base of styles can be used, rather than separate ones for the host app and for Exago. The Exago DOM is accessible from the host application, so custom scripting is possible without being limited to **Action Events**.

Background

The JS API implements asynchronous calls to Exago BI in the client browser. Besides the advantages of embedding in s and interacting programmatically, the API allows for multiple calls without needing to generate a new session for each one. As sessions are created once per page load, this can increase the feeling of responsiveness in the host application.

As the JS API runs on the client-side, it is not standalone. Session objects are still required to be generated, either with the **.NET** or **REST Web Service API**. Session objects must include any relevant **user permissions** and **configuration settings**.

A parameter called the **ApiKey** encodes the session object in a string, which is passed from the server-side API to the JS API. The JS API then initializes a JS API object, which is analogous to an Exago user session.

> **NOTE**
>
> JS API objects are static and single-use. They cannot persist between page loads, and multiple JS API objects cannot be active on the same page.

The JS API object provides a set of functions that are used to implement elements of Exago BI functionality.

> **NOTE**
>
> API Actions and active reports should *not* be set when using the JS API as doing so may produce errors within the API session. Any desired actions, such as executing a report or editing a report on the page, should be done using the appropriate JavaScript API functions instead.

Setup and Configuration

These steps describe how to configure the environment to use the JS API, as well as how to implement it in an application.

1. Create the Session
2. Instantiate the JS API object

1. Create the Session

1. Use the .NET API or REST Web Service API to set up the session. Make all configuration changes there, as these settings cannot be changed once the JS API object is loaded.

.NET API

Instantiate an API object with one of the available constructors.
Example:

```
Api api = new Api("appPath");
```

REST Web Service API

The `Id` is the session ID (sid) that will be needed for subsequent REST Web Service API calls. Save the value of the **ApiKey** property, as it will be needed to instantiate the JS API object in step 2.
`POST http://server/webservice_virtual_path/rest/sessions` with this payload:

```
{
 "AppUrl": "UrlParamString",
 ...
 "Id": "1234-5678-9012-3456",
 ...
}
```

2. Set the **WebAppBaseUrl** property to the virtual directory where Exago BI is installed:

.NET API

```
api.SetupData.WebAppBaseUrl = "http://server/Exago/";
```

REST Web Service API

Do either one of the following:

- PATCH **/Rest/Settings**?sid={sid}. This method requires the session be established first in step 1. Payload:

  ```
  {
   "WebReportsBaseUrl" : "http://server/Exago"
  }
  ```

- Add to the web service config file ({WebService}/Config/WebReportsApi.xml):

  ```
  <webreportsbaseurl>http://server/Exago/</webreportsbaseurl>
  ```

In contrast to the .NET and REST Web Service APIs, the JavaScript API has no concept of an **Active Report** or an **API Action**. Instead, the action and report are specified by calling the individual JS function calls.

> **NOTE**
>
> A side-effect of this is that per-session report changes cannot be made in memory, since the JS API function can only act on saved reports. Changes will need to be saved to storage before execution.

3. Once the session is ready, get the **ApiKey** if not saved in step 1.1 (skip this step if you already have the ApiKey). This encodes the session settings to pass to the JS API. This API key will need to be passed to the page as it will be used in client side JavaScript calls.

.NET API

```
return api.GetApiKey();
```

REST Web Service API

GET **Rest/Sessions/**{sid}, then get the ApiKey property from the response object:

```
{
 ...
 "ApiKey": "encodedAlphanumericApiKey"
}
```

2. JS API Object

1. Load the JS API library into the host web application via a <script> tag:

```
<script src="http://server/Exago/WrScriptResource.axd?s=ExagoApi"></script>
```

> **NOTE**
>
> WrScriptResource.axd is not a file on the file system, it is a virtual file that contains the necessary scripts to load the API. "http://server/Exago" is the URL to the virtual path of your Exago web application.

2. Using the **ApiKey**, instantiate an **ExagoApi** JavaScript API object. There are two constructors available:

Constructor 1

```
var api = new ExagoApi(ExagoBaseURL, ApiKey, onLoadCallback, [showErrorDetail]);
```

- **ExagoBaseURL** â€" URL to the installation of Exago BI
- **ApiKey** â€" key generated when the session was created
- **onLoadCallback** â€" function to execute once the JS API has been fully loaded and is ready for use
- *Optional*: **showErrorDetail** â€" set to *true* to see more detailed application error messages. (Default: False).

Example

```
new ExagoApi("https://exago.example.com", "tCq0n6dM6kQEbgofptKsdinMNJpqsCIF5kVhzz2wcgcEfGjYE8eEeKjR3NBkjayZY%2fbs7RLTcpH9EtDyhRBA%3d%3d", () => {
    console.log("Exago is ready for further API calls!");
  }, true);
```

Constructor 2 v2019.1.22+ v2019.2.8+

This constructor contains an additional OnError() callback function. The definitions below are written using TypeScript syntax.

```
var api = new ExagoApi(options: wrExagoApiOptions);
```

This constructor takes one argument, an object of type **wrExagoApiOptions**. **wrExagoApiOptions** objects are defined by the following interface:

```
interface wrExagoApiOptions {
  WebEndpoint: string;
  ApiKey: string;
  OnLoad: () => void;
  OnError?: (err: wrJsApiError) => void;
  OnDisposeContainer?: (container: HTMLElement) => void;
  ShowErrorDetail?: boolean;
  IncludeCredentials?: boolean;
}
```

- **WebEndPoint** â€" URL to the installation of Exago BI
- **ApiKey** â€" key generated when the session was created
- **OnLoad()** â€" function to execute once the JS API has been fully loaded and is ready for use
- **OnError()** â€" *optional*: function to execute if the loading of the JS API fails. Takes one argument, an object of type **wrJsApiError** which simply represents an error message if loading of the API fails. A **wrJsApiError** object is defined by the following interface:

  ```
  interface wrJsApiError {
    Message: string;
  }
  ```

- **OnDisposeContainer** â€" *optional*: function to execute when the containing Exago is no longer needed and can be disposed. Takes one argument of type **HTMLElement** object, the ID of the containing . This is the recommended way of passing this function to the ExagoApi object. For more information, see Disposing Containers below.
- **ShowErrorDetail** â€" *optional*: *true* if full error messages should be displayed to the end user in the browser, *false* if they should be saved to the log only
- **IncludeCredentials** v2019.2.29+ v2020.1.13+ v2021.1.1+ â€" *optional*: determines the credentials mode to use for any XHR or fetch request.

Request Type	IncludeCredentials Value	Result
fetch	omitted or *true*	credentials property of the Request object is set to include
fetch	*false*	credentials property of the Request object is set to same-origin
XHR	true	withCredentials property of the XMLHttpRequest object is set to true
XHR	false	withCredentials property of the XMLHttpRequest object is set to false

OnError()

The OnError() callback function will be called under the following conditions:

- Any HTTP 4xx or 5xx status code is returned from any XHR or fetch request involved with initialization

There are several use cases for the OnError() callback function, such as (but not limited to):

- An expired **ApiKey** is passed
- An incorrect **WebEndPoint** is passed
- In a cross-origin environment, cookieless sessions are not enabled
- The Exago application server is down

Example

```
new ExagoApi({
 WebEndpoint: "https://exago.example.com",
 ApiKey: "tCq0n6dM6kQEbgofptKsdinMNJpqsCIF5kVnzz2wcgcEfGjYE8eEeKjR3NBkjayZY%2fbs7RLTcpH9EtDyhRBA%3d%3d",
 ShowErrorDetail: true,
 OnLoad: () => {
  console.log("Exago is ready for further API calls!");
 },
 OnError: () => {
  console.log("There was an error loading Exago"); }, });
```

> **NOTE**
>
> ApiKeys are single-use. Multiple instances are not supported nor necessary. Functions can be called against the object for the duration of the session.

Functions

The following functions are available for loading and managing Exago functionality.

> **NOTE**

Functions can only be used once the JS API is fully loaded. Wait for onLoadCallback() or OnLoad() to indicate that the API is ready.

CAUTION

Due to a known issue, the callbacks mentioned in the following JavaScript API functions only work as described as of *v2018.1.12+* or *v2018.2+*.

LoadFullUI(container)

Load the full web application user interface in to a container.

Argument	Description
container	container to place the user interface into

NOTE

The Full UI being loaded will block almost all other actions, so while the Full UI is displayed on screen, the host application cannot perform any other Exago functions such as executing reports or creating new reports.

ExecuteReport(container, exportType, reportPath, [udfOrOptions], [updateCallback], [errorCallback])

ExecuteReport(container, exportType, reportID, [udfOrOptions], [updateCallback], [errorCallback], true) v2020.1.23+ v2021.1.11+

ExecuteReport(container, exportType, reportPath, [udfOrOptions], [updateCallback], [errorCallback], false) v2020.1.23+ v2021.1.11+

Execute a report to a specific output type in a defined container.

Argument	Description
container	Div container to place the executed report into
exportType	html|pdf|pdfsnapshot (v2021.2+)|csv|excel|rtf
reportPath	Relative path to report to execute **Example:** "MyFolderMyReport"
reportId v2020.1.23+ v2021.1.11+	The content ID of the report from the Storage Management database. When using this option, the **useReportId** argument must be passed as *true*.
udf pre-v2019.1	Optional: Report UDF information for use with folder management
udfOrOptions v2019.1+	Optional: Report UDF information for use within folder management or a Report Options object for modifying reports before execution **NOTE** If the **wrJsApiExecuteReportOptions** object is passed to this parameter, the *updateCallback* and *errorCallback* parameters will be ignored and **ExecuteReport** will change its behavior based upon the contents of the object being passed. For more information, please see the **Interfaces** section below and the **JavaScript API: Filters and Parameters** article.
updateCallback (container, updateType)	Optional: Callback to execute when the execution status changes, called when the report execution is starting and again when it is ready to be viewed • Parameter â€˜containerâ€™: The same container HTMLElement that was passed in to the call • Parameter â€˜updateTypeâ€™: The type of update as a string, either The parameter 'updateType' will assume one of these string values: • â€œexecutionstartâ€: The report has been loaded and is starting execution, or • â€œinitialcontentloadâ€: The execution viewer has been loaded and populated with at least the first page of the report, if executing an Advanced Report or ExpressView to HTML) **Note:** This callback will occur once for each status change unless errors occur.
errorCallback (container, errorMessage) => string	Optional: Callback to execute in the event an execution blocking error occurs • Argument â€˜containerâ€™: The same container HTMLElement that was passed in to the call • Argument â€˜errorMessageâ€™: The error text Return value: See **errorCallback return values** below.
useReportId v2020.1.23+v2021.1.11+	When calling a report by its Storage Management content ID, set this value to *true*. When calling a report by its report path, set this value to *false*. **NOTE** Exago recommends calling reports by their ID in all versions that support doing so.

ExecuteStaticReport(exportType, reportPath, udfOrOptions , successCallback, [errorCallback])

Execute an Advanced Report, and return its output to the *successCallback* function. Report is not interactive.

> **NOTE**
>
> This function is only compatible withÂ **Advanced Reports**.

Argument	Description
exportType	html|pdf|csv|excel|rtf|json
reportPath	Relative path to report to execute **Example:** “MyFolderMyReport”
udf pre-v2019.1	Report UDF information for use with folder management
udfOrOptions v2019.1+	Report UDF information for use within folder management or a Report Options object for modifying reports before execution **NOTE** If the **wrJsApiExecuteStaticReportOptions** object is passed to this parameter, the *sucessCallback* and *errorCallback* parameters will be ignored and **ExecuteReport** will change its behavior based upon the contents of the object being passed. For more information, please see the **Interfaces** section below and the **JavaScript API: Filters and Parameters** article.
successCallback (executionData)	Callback to execute when execution request returns • Parameter ‘executionData’: a string containing the executed reportâ€™s data is passed as a parameter, whose formatting depends on the specified exportType. If the export type is â€œhtmlâ€, â€œcsvâ€, or â€œjsonâ€, the returned string will contain all of the data in the corresponding format. If the export type is â€œpdfâ€, â€œexcelâ€, or â€œrtfâ€, the returned string will be partial URL to the file, e.g. â€œExecuteExport.aspx?eid=â€¦â€. See the below example for prefixing the returned partial URL with the base address.
errorCallback (errorMessage)	Optional: Callback to execute in the event an error occurs. • Parameter ‘errorMessage’: The error text. Return value: See **errorCallback return values** below.

```
const container  = ...
 const exportType = ...
 const reportPath = ...

 api.ExecuteStaticReport(exportType, reportPath, null,
    (executionData) =>
    {
       if (exportType == "excel" || exportType == "rtf" || exportType == "pdf")
          container.innerHTML = "<iframe src='" + api.GetBaseAddress() + executionData + "'></ iframe>";
       else
          container.innerHTML = executionData;
    },
    (errorMessage) =>
    {
       container.innerHTML = errorMessage;
    });
```

ScheduleReportWizard(container, reportPath, [udf], [errorCallback], [successCallback])

Start the Schedule Report Wizard for the specified report.

> **NOTE**
>
> The optional successCallback is available in *v2021.1.14+*.

Argument	Description
container	Div container to place the scheduled report wizard into
reportPath	Relative path to report to schedule **Example:** “MyFolder/MyReport”
udf	Optional: Report UDF information for use with folder management
errorCallback (container, errorMessage) => string	Optional: Callback to execute in the event an error occurs, such as the scheduler being disabled • Parameter â€˜containerâ€™: The same container HTMLElement that was passed in to the call • Parameter â€˜errorMessageâ€™: The error text

	Return value: see **errorCallback return values** below.

ScheduleReportManager(container, [errorCallback], [successCallback])

Open the

Schedule Manager.

> **NOTE**
>
> The optional successCallback is available in *v2021.1.14+*.

Argument	Description
container	Div container to place the scheduled report manager into
errorCallback (container, errorMessage) => string	Optional: Callback to execute in the event an error occurs, such as the scheduler being disabled • Parameter â€˜containerâ€™: The same container HTMLElement that was passed in to the call • Parameter â€˜errorMessageâ€™: The error text Return value: see **errorCallback return values** below.

LoadReportTree(successCallback, [errorCallback])

Load the report tree as JSON, returned to the *successCallback* method.

Argument	Description
successCallback (reportTree)	Callback to execute once the report tree has been loaded. • Argument ‘reportTree’: A JSON object representing the report tree.
errorCallback (errorMessage)	Optional: Callback to execute in the event an error occurs • Argument â€˜errorMessageâ€™: The error text

EditReport(container, reportPath, [udf], [errorCallback], [successCallback])

Load the report designer for a report.

> **NOTE**
>
> The optional successCallback is available in *v2021.1.14+*.

Argument	Description
container	Div container to place the report designer into
reportPath	Relative path to report to edit **Example:** “MyFolderMyReport”
udf	Optional: Report UDF information for use with folder management
errorCallback (container, errorMessage) => string	Optional: Callback to execute if the report fails to load • Parameter â€˜containerâ€™: The same container HTMLElement that was passed in to the call • Parameter â€˜errorMessageâ€™: The error text Return value: see **errorCallback return values** below.

NewReport(container, reportType, [successCallback])

Create a new report in the corresponding Report Designer. For *pre-v2020.1*, passing *advanced* to the reportType argument will start the New Report Wizard.

> **NOTE**
>
> The optional successCallback is available in *v2021.1.14+*.

Argument	Description
container	Div container to place the report designer into
reportType	advanced|express|dashboard|chained|expressview

DisposeContainerContent(container)

Disposes the contents of a container and resets the system state to be aware of what containers are open.

Argument	Description
container	Div container to dispose

Disposes the contents of a page, removing any event listeners added to the page and performing other cleanup necessary for complete removal of the ExagoAPI instance. After calling *DisposePage()*, the ExagoAPI instance should be considered “closed.” Any new Exago JavaScript API calls must be performed on a new ExagoAPI instance.

IsAllowedReportType(reportType)

Returns whether or not a specified report type is allowed for the session.

> **TIP**
>
> **CrossTab Reports** are a type of **Advanced Report**.

Argument	Description
reportType	advanced|express|dashboard|chained|expressview

GetAllowedReportTypes()

Returns an array of the report types allowed for this session.

Example:

```
function RunReportJS() {
  var container = document.getElementById("ExagoDiv");
  api.ExecuteReport(container, "html", "ExamplesClientReport");
}
```

> **NOTE**
>
> Container divs must be empty or disposed before loading. Additionally, you should specify size and position constraints for each div.

```
div#ExagoDiv {
  width: 1200px;
  height: 600px;
  position: relative;
}
```

Disposing Containers

It is important to properly dispose of containers when they are finished being used by explicitly calling the *DisposeContainerContent(container)* method.

Optionally, an *OnDisposeContainer* callback can be defined that will execute when a container has been disposed either implicitly or explicitly. This allows the host application to safely reset the container to whatever state necessary, or remove the container from the page entirely. When a user encounters an error that prevents the requested action, e.g., *ExecuteReport(…)*, the container will auto-dispose and execute the *OnDisposeContainer* callback if one is defined.

Example:

```
api.OnDisposeContainer = function(container) {  container.parentElement.removeChild(container); };
```

Disposing Pages v2018.2+

Once an ExagoAPI instance has been instantiated, *DisposePage()* must be called before instantiating subsequent ExagoAPI instances. *DisposePage()* will dispose the contents of an entire page, thereby disposing each instantiated container. Furthermore, it will remove any event listeners added to the page by the ExagoApi instance and perform other cleanup necessary for complete removal of the ExagoAPI instance.

After calling *DisposePage()*, the ExagoAPI instance should be considered “closed.” Any new Exago JavaScript API calls must be performed on a new ExagoAPI instance.

errorCallback return value

Whenever an error occurs that prevents the JS API action from updating a container (such as an undefined report name in a call of *ExecuteReport*), the following will happen:

- If **errorCallback** is not defined, an alert dialog will appear with the error message (or a placeholder error message, if the URL parameter ‘showerrordetail’ is set to false).
- If **errorCallback** is defined, it will first be called. If errorCallback returns a string (either the original error message or a custom message), an alert dialog will appear with the given string, or a placeholder error message if the URL parameter ‘showerrordetail’ is set to false.
- If **errorCallback** does not return a string, or the string is empty, no alert dialog will be shown and the *DisposeContainerContent()* method will run immediately.

Clicking on the error dialogâ€™s dismiss button will close the dialog and call *DisposeContainerContent()*. The containerâ€™s content will be closed, the inner HTML will be cleared, and the *OnDisposeContainer* callback will be called.

Interfaces v2019.1+

The following interfaces are available for managing and modifying report settings at runtime.

wrJsApiExecuteReportOptions

Allows information for the modification of filters and parameters of a report to be passed to the *udfOrOptions* parameter of the **ExecuteReport** function. If this interface is passed to **ExecuteReport** the *updateCallback* and *errorCallback* parameters of this function will be ignored; however, they may be specified within this interface if necessary

> **NOTE**
>
> All properties in the wrJsApiExecuteReportOptions object are optional.

Property	Description
Udf	The UDF information used to load the report if required
UpdateCallback (container, updateType) => void	Callback to execute when there has been an update in the status of the report execution
ErrorCallback (container, errorMessage) => string	Callback to execute in the event an error occurs
NewFilters	An array of **wrJsApiExecuteReportFilter** objects defining the new filters to add to the report before execution begins
PromptingFilterValues	An array of prompting filter values to modify before execution begins • When executing an **Advanced Report**, an array of **wrJsApiPromptingFilterValue** objects should be passed • When executing a **Dashboard** or **Chained Report**, an array of **wrJsApiCompositePromptingFilterValue** objects should be passed **NOTE:** It is assumed that the value provided by the given item will be the final value for the filter before the report executes. As such, any filter with a value modified by an item in this array will not prompt the user upon execution.
Parameters	A list of **wrJsApiExecuteReportParameter** objects defining the non-hidden parameters to add or modify before execution begins

wrJsApiExecuteStaticReportOptions

Allows information for the modification of filters and parameters of a non-interactive report to be passed to the *udfOrOptions* parameter of the **ExecuteStaticReport** function. If this interface is passed to **ExecuteStaticReport** the *successCallback* and *errorCallback* parameters of this function will be ignored; however, they may be specified within this interface if necessary.

> **NOTE**
>
> All parameters in the wrJsApiExecuteStaticReportOptions object are optional.

Parameter	Description
Udf	The UDF information used to load the report if required
SuccessCallback (execution) => void	Callback to execute when the report execution succeeds
ErrorCallback (errorMessage) => string	Callback to execute in the event an error occurs
NewFilters	An array of **wrJsApiExecuteReportFilter** objects defining the new filters to add to the report before execution begins
PromptingFilterValues	An array of prompting filter values to modify before execution begins • When executing an **Advanced Report**, an array of **wrJsApiPromptingFilterValue** objects should be passed • When executing a **Dashboard** or **Chained Report**, an array of **wrJsApiCompositePromptingFilterValue** objects should be passed **NOTE** It is assumed that the value provided by the given item will be the final value for the filter before the report executes. As such, any filter with a value modified by an item in this array will not prompt the user upon execution.

Parameters	A list of **wrJsApiExecuteReportParameter** objects defining the non-hidden parameters to modify before execution begins.

wrJsApiExecuteReportFilter

Defines the necessary information for a new filter. To be passed to **wrJsApiExecuteReportOptions** in order to add new filters to a report prior to execution.

Property	Description
FilterText	The filter string (e.g., “Employees.FirstName” or “=MonthName({Employees.BirthDate})”) **Note:** This must take the form of a data field, formula, or parameter.
Operator	The condition used to match the data values to the specified filter value or values type **wrJsApiFilterOperator** = • “EqualTo” (Default) • “NotEqualTo” • “LessThan” • “LessThanOrEqualTo” • “GreaterThan” • “GreaterThanOrEqualTo” • “StartsWith” • “NotStartsWith” • “EndsWith” • “NotEndsWith” • “Contains” • “NotContains” • “Between” • “NotBetwen” • “OneOf” • “NotOneOf”
PromptFlag	Select whether or not to prompt the user for a value upon report execution • Defaults to *False*
AndFlag	Set to *True* to insert an AND condition after this filter, set to *False* to insert an OR condition after this filter • Defaults to *True*
GroupWithNextFlag	Select whether or not to group this filter with the filter following it • Defaults to *False*
Values	The values to add to the filter. Can be a string, number, or date, or an array of any of these types if multiple values are required (e.g., for “Between” and “OneOf” filters). **NOTE** All date values must be passed as JavaScript Date objects (e.g., *var date = newDate(‘December 17, 1995 03:24:00);*.
Title	The name of the filter • Defaults to an empty string (“”)

wrJsApiPromptingFilterValue

Defines the necessary information to pass a value to a prompting filter for **Advanced Reports**. To be passed to **wrJsApiExecuteReportOptions** in order to set prompting filter values of a report prior to execution.

Property	Description
FilterText	The filter text used to identify which filter to modify
Index	Optional: In the case of multiple filters having the same *FilterText*, the index into the matching group of filters • Defaults to 0 (zero)
Operator	Optional: Reassign the condition used to match the data values to the specified filter value or values type **wrJsApiFilterOperator** = • “EqualTo” • “NotEqualTo” • “LessThan” • “LessThanOrEqualTo” • “GreaterThan” • “GreaterThanOrEqualTo” • “StartsWith” • “NotStartsWith” • “EndsWith” • “NotEndsWith” • “Contains” • “NotContains” • “Between” • “NotBetwen” • “OneOf” • “NotOneOf”
	The values to add to the filter. Can be a string, number, or date, or an array of any of these types if multiple values are required (e.g., for “Between” and “OneOf” filters).

Values	**NOTE** All date values must be passed as JavaScript Date objects (e.g., *var date = newDate('December 17, 1995 03:24:00)*;.

wrJsApiCompositePromptingFilterValue

Defines the necessary information to pass a value to a prompting filter for **Dashboards** and **Chained Reports**, also known as "composite reports". To be passed to **wrJsApiExecuteReportOptions** in order to set prompting filter values of a report prior to execution.

Property	Description
Index	The index of the filter as it appears in the prompting UI, beginning at 0 (zero) and increasing by 1 for each following filter
Operator	Optional: Reassign the condition used to match the data values to the specified filter value or values type **wrJsApiFilterOperator** = • "EqualTo" • "NotEqualTo" • "LessThan" • "LessThanOrEqualTo" • "GreaterThan" • "GreaterThanOrEqualTo" • "StartsWith" • "NotStartsWith" • "EndsWith" • "NotEndsWith" • "Contains" • "NotContains" • "Between" • "NotBetwen" • "OneOf" • "NotOneOf"
Values	he values to add to the filter. Can be a string, number, or date, or an array of any of these types if multiple values are required (e.g., for "Between" and "OneOf" filters). **NOTE** All date values must be passed as JavaScript Date objects (e.g., *var date = newDate('December 17, 1995 03:24:00)*;.

wrJsApiExecuteReportParameter

Defines the necessary information to add or modify non-hidden parameters. To be passed to **wrJsApiExecuteReportOptions** in order to add or set parameter values of a report prior to execution.

Property	Description
Name	The name of the non-hidden parameter to modify. This value must not contain the '@' symbol.
Value	The value to give to the parameter. Can be a string, number, or date depending on the parameter's datatype.
PromptText	The text that will display when prompting the user for a value. If this field is set to a null value, the parameter will be considered non-prompting.

JavaScript API: Filters and Parameters

As of *v2019.1*, runtime filters and parameters can now be modified via the JavaScript API prior to a report's execution. New behaviors have been defined for the for *ExecuteReport* function in order to allow the addition of new filters, the modification of existing filters, and the addition or modification of parameters. These modifications only apply at runtime and do not alter report definitions.

> **IMPORTANT**
>
> Review the **JavaScript API** article for in depth descriptions of the **Functions** and **Interfaces** utilized by these features.

For the code snippets in the following sections,Â assume the following framework:

```
function executeReport() {
 var reportPath = document.getElementById('exagoReport').value;
 var exportSelector = document.getElementById('execSelection');
 var exportType = exportSelector.value;
 var containerId = document.getElementById('containerSelection').value;
 var container = document.getElementById('container' + containerId);

  var reportOptions =
 {
  Udf: null,
  UpdateCallback: updateCallback,
  ErrorCallback: errorCallback,

 //For adding new filters to a report
  NewFilters: filters,

 //For modifying prompting filters in Advanced Reports
  PromptingFilterValues: promptingFilters,

 //For modifying prompting filters Dashboards and Chained Reports
 //PromptingFilterValues: promptingCompositeFilters,

 //For adding or modifying the non-hidden parameters of a report
  Parameters: parameters
 };

 api.ExecuteReport(container, exportType, reportPath, reportOptions);
}
```

Adding a New Filter

To add runtime filters to a report via the JavaScript API:

1. Define the report filter objects by specifying their FilterText, Operator, Values, and other necessary information.
2. Add these definitions to an array of report filters.
3. Reference this array in the **NewFilters** property of the report options interface.
4. Call *ExecuteReport* referencing the report options parameter.

For a full list of properties and a description of this feature, please see the **wrJsApiExecuteReportFilter** section of the **JavaScript API** article.

Please consider the following when adding a new filter via the JavaScript API:

- FilterText may not be null
- FilterText must take one of the following forms:
  - a data field (Employees.LastName)
  - a function (=Month({Employee.BirthDate}))
  - a parameter (@)
- If the Operator type is OneOf or NotOneOf, then Values must be an array. If the type is Between or NotBetween, then the array may only be a length of two.
- Filter functions, like *DayAfterTomorrow()*, may not be used as a value.

> **NOTE**
>
> Currently, runtime filters may not be added to **Dashboards** or **Chained Reports** via the JavaScript API.

Example

The following code sample creates two runtime filters on the *Employees* data category.

```
var filters = [];

var filter =
{
  FilterText: "Employees.EmployeeID",
  Operator: "Between",
  PromptFlag: true,
  AndFlag: true,
  GroupWithNextFlag: false,
  Values: [2, 5],
  Title: "Employee Id"
};
filters.push(filter);

filter =
{
  FilterText: "=Year({Employee.HireDate})",
  Operator: "GreaterThanEqualTo",
  PromptFlag: false,
  AndFlag: true,
  GroupWithNextFlag: false,
  Values: 2010,
  Title: "Employee Hire Date"
};
filters.push(filter);
```

Modifying an Existing Prompting Filter

To modify and set the values for existing prompting filters on a report via the JavaScript API:

1. Define the prompting filter objects by specifying their FilterText, Index, and Values.
2. Add these definitions to a designated array of prompting filters.
3. Reference this array in the **PromptingFilterValues** property of the report options interface.
4. Call *ExecuteReport* referencing the report options parameter.

> **NOTE**
>
> Only prompting filters may be modified prior to report execution. Non-prompting filters cannot currently be modified via the JavaScript API.

For a full list of properties and a description of this feature, please see the **wrJsApiPromptingFilterValue** and **wrJsApiCompositePromptingFilterValue** sections of the **JavaScript API** article.

> **NOTE**
>
> Any prompting filter with a value set via the JavaScript API will be treated as non-prompting and will not appear in the prompting **Filters** menu upon report execution.

Examples

The following sections detail how to modify prompting filters on **Advanced Reports** and non-interactive filters on **Dashboards** and **Chained Reports**:

Modifying Prompting Filters on an Advanced Report

For the following example, assume that the following existing prompting filters on an **Advanced Report**:

```
Employees.LastName Starts With 'D' AND
Employees.ID Is Between '2' And '5' AND
Employees.LastName Is Not Equal To 'Dodsworth'
```

These filters are ordered as they appear in the **Filters** menu in the **Advanced Report** designer. FilterText values are treated as the primary identifier of each filter. The Index value serves as a secondary identifier, increasing its value at each instance of identical FilterText. The following list details filter indexing for **Advanced Reports**:

```
Employees.LastName  [Index 0]
Employees.ID        [Index 0]
Employees.LastName  [Index 1]

/** Continuing the example: ************************/
=MonthName({Employees.BirthDate}) [Index 0]
Employees.ID        [Index 1]
Employees.BirthDate [Index 0]
Employees.LastName  [Index 2]
```

> **NOTE**

If the Index value is left null, the value of 0 (zero) will be passed by default.

The following code modifies and sets the values for two existing prompting parameters on a report prior to its execution. It sets the value of "Davolio" to the second instance of the Employees.LastName filter and modifies the Operator of the first instance of the Employees.EmployeeID filter.

```
var promptingFilters = [];

var promptingFilter =
{
  FilterText: "Employees.LastName",
  Index: 1,
  Values: "Davolio"
};
promptingFilters.push(promptingFilter);

promptingFilter =
{
  FilterText: "Employees.EmployeeID",
  Index: 0,
  Values: [2, 5],
  Operator: "NotBetween"
};
promptingFilters.push(promptingFilter);
```

Modifying Prompting Filters on a Dashboard or Chained Report

For the following example, assume the following existing prompting filters on the **Dashboard** or **Chained Report**:

```
=Year({Employees.HireDate}) = '1996' AND
Employees.FirstName Is Not Equal To 'Janet' AND
Employees.Title = 'Sales Representative'
```

These filters are ordered as they appear in the **Filters** menu on the composite report. However, within composite reports, Index values are the primary identifiers of each filter and are set on a global scale. TheÂ Index value increases for each new filter on a composite report. The following list details filter indexing for composite reports:

```
=Year({Employees.HireDate}) [Index 0]
Employees.FirstName       [Index 1]
Employees.Title           [Index 2]

/** Continuing the example ************************/
Employees.ID              [Index 3]
Employees.BirthDate       [Index 4]
Employees.FirstName       [Index 5]
```

NOTE

Index values for composite reports do not default to 0 (zero) and are a required field for each prompting filter.

The following code modifies and sets the values for three existing prompting parameters on a composite report prior to its execution.

```
var promptingCompositeFilters = [];

var promptingCompositeFilter =
{
  Index: 0,
  Values: [1996, 2005],
  Operator: "Between"
};
promptingCompositeFilters.push(promptingCompositeFilter);

promptingCompositeFilter =
{
  Index: 1,
  Values: ["Nancy"],
};
promptingCompositeFilters.push(promptingCompositeFilter);

promptingCompositeFilter =
{
  Index: 2,
  Values: "Sales Manager"
};
promptingCompositeFilters.push(promptingCompositeFilter);
```

Adding Parameters and Modifying Parameter Values

New parameters may be added to a report and existing non-hidden parameter values may be modified prior to report execution via the JavaScript API. For a full list of properties and a description of this feature, please see the **wrJsApiExecuteReportParameter** section of the

JavaScript API article.

To add a new parameter to a report:

- The Name value must be unique and cannot be identical to the names assigned to existing non-hidden parameters or to reserved parameters. If the Name value is identical to an already existing non-hidden parameter, then this parameter's value will be overwritten.
- The Value being assigned to the new parameter must be a number, string, or date. It may not be an array of multiple values.
- The PromptText value set represents the text that will display when prompting the user for a value. If PromptText is set to a null or empty string value, the parameter will be set to non-prompting. Otherwise, the parameter will prompt the user for input upon report execution.

> **NOTE**
>
> For more information on parameters and a list of reserved parameters please see the **Parameters** article.

To modify the value of an existing non-hidden parameter:

- The Name value must be identical to an existing non-hidden parameter on the report.
- The Value being assigned to the parameter must be a number, string, or date, and must match the existing parameter's datatype. It may not be an array of multiple values.
- The PrompText value set represents the text that will display when prompting the user for a value. If PromptText is set to a null or empty string value, the parameter will be set to non-prompting. Otherwise, the parameter will prompt the user for input upon report execution.

> **NOTE**
>
> Modifying parameters via the JavaScript API will overwrite existing parameters at the application/system level for that specific report execution. For example, if an existing prompting parameter is modified to be non-prompting, it is equivalent to setting this parameter to be non-prompting within the **Admin Console**.

Example

The following code will add two parameters. In this example, the first parameter, @, will prompt the user with the specified prompting text, while the second parameter, @, will be non-prompting.

```
var parameters = [];

var parameter =
{
  Name: "FirstName",
  Value: "Anne",
  PromptText: "Enter your first name"
};
parameters.push(parameter);

parameter =
{
  Name: "Id",
  Value: 9,
  PromptText: ""
};
parameters.push(parameter);
```

JavaScript API: Report and Dashboard Execution Timing

As of *v2019.1.3+*, Exago BI exposes a JavaScript API that may be invoked in the browser to execute reports and dashboards and retrieve information about how long they take to run. This makes it easier to do benchmarking, especially for dashboard executions where a test would need to wait until the last tile is finished loading.

> **NOTE**
>
> All report types and execution formats are supported in this API.

wrWebReportsCtrl.ExecuteReport now returns a Promise that resolves with the class that controls the given execution (one of wrChainedExecuteClass, wrDashboardDesignCtrl, wrExecuteClass, wrExpressViewDesignCtrl). Each of these classes exposes the Timing property, which is a Promise that resolves when execution timing information has finished being collected for the given execution.

```
const executor = await wrWebReportsCtrl.ExecuteReport("Reports\Report", "html");
await executor.Timing; // Evaluates to { load: 5243 }
```

The Timing property resolves with an object describing timing information about the report or dashboard execution:

```
{
  load: 5243
}
```

Returns the time (in milliseconds) it took for the report or last dashboard tile to finish loading.

Application and Performance Logging

An administrator can configure how Exago handles logging in order to change or extend functionality.

Logging Defaults

By default Exago saves a log file WebReportsLog.txt to the applicationâ€™s **Temp** path. The logger maintains a lock on the file for the lifespan of the application. The log file cannot be edited or deleted without restarting the application or releasing the lock.

> **NOTE**
>
> As of version 2018.1, the .NET API logs to the file <Temp>WebReportsApiLog.txt

There are five configurable verbosity levels for the logger. By default, Exago logs at the *Info* level. Set the level by changing the **Admin Console** > 🔧 **General** > ⚙ **Other Settings** > **Write Log File** setting.

- *None* â€” Turns logging off.
- *Error* â€” Logs application errors.
- *Warn* â€” Logs application warnings, which may be indicative of problems in the configuration, as well as all **Error** messages.
- *Info* â€” Logs SQL statements, number of rows returned from each statement, and report execution information, as well as all **Warn** and **Error** messages. Report execution information includes the following:
  - Execution start â€” Start time, @ and @ parameter values, full report name and filter summary.
  - Execution end â€” End time, runtime, @ and @ parameter values amd full report name.
- *Debug* â€” Logs a variety of debugging information that can be used to time specific parts of the application as well as all **Info**, **Warn**, and **Error** messages.

> **IMPORTANT**
>
> Setting the log level to **DEBUG** in a production environment can have unintended performance consequences. Only use **DEBUG** level logging in production temporarily and under advisement from the Exago Support, Services or Development teams.

log4net

The logger can load its configuration from a file and continually watch the file for changes. A config file can be used to lock or unlock the log file, change the log file name and path, as well as customize and extend logging capability.

A custom log configuration file overrides the application's configuration settings.

> **NOTICE**
>
> Enabling or disabling logging with log4net will require restarting the web server.

To use a custom log configuration:

> **NOTE**
>
> This article references <WebApp>/ and <Sched>/as a placeholder for the Web Application and Scheduler Service's install location respectively. The default install location is C:Program FilesExagoExagoWeb (/opt/Exago/ on Linux), or C:Program FilesExagoExagoScheduler (/opt/Exago/Scheduler/ on Linux); however, these directories can be changed during installation.

1. Create a file named log4net.config in the <WebApp>Config directory of the Exago Web Application, or open the <Sched>/eWebReportsScheduler.exe.config for the Scheduler Service. The following shows a sample config file:

> **NOTE**
>
> For versions *pre-v2019.2.32*, *pre-v2020.1.15*, or *pre-v2021.1.3*, replace #%-3property{ThreadId} with #%-3thread in the conversionPattern nodes.

```
<log4net>
  <appender name="RollingFile" type="log4net.Appender.RollingFileAppender" >
 <file value="WebReportsLog.txt" />
     <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date #%-3property{ThreadId} %-5level [%-8.8property{ReportId}][%-8.8property{SessionId}] - %message%newline" />
     </layout>
 <maxSizeRollBackups value="10" />
 <maximumFileSize value="1MB" />
 <rollingStyle value="Size" />
  </appender>

  <appender name="PerfRollingFile" type="log4net.Appender.RollingFileAppender" >
 <file value="WebReportsPerfLog.txt" />
     <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date #%-3property{ThreadId} %-5level %logger [%-8.8property{ReportId}][%-8.8property{SessionId}] - %message%newline" />
     </layout>
 <maxSizeRollBackups value="10" />
 <maximumFileSize value="1MB" />
 <rollingStyle value="Size" />
  </appender>

  <root>
    <level value="DEBUG" />
    <appender-ref ref="RollingFile" />
  </root>

<!-- Set logging level to DEBUG and use logger define at the ROOT level -->
  <logger name="Exago Logger">
    <level value="DEBUG" />
  </logger>


<!-- Disable Performance Logging
  <logger name="Perf Logger">
    <level value="OFF" />
  </logger>
-->

<!-- Enable Perf Logging in general log file -->
  <logger name="Perf Logger">
    <level value="INFO" />
  </logger>

<!-- Perf logging in its own file
  <logger name="Perf Logger" additivity="False">
    <level value="INFO" />
    <appender-ref ref="PerfRollingFile" />
  </logger>
-->

</log4net>
```

2. Determine the type of logging needed: either **Application**, **Performance** or both.
3. Edit the log4net.config file as follows:

Anatomy of the log4net.config file

For full details about the log4net.config file, refer to Apache's official log4net documentation. Below are some common changes that can be made that apply to Exago.

Appenders

The **appender** sections control the log output. There can be seperate log files output for both the standard Exago application log and for performance logging. In the example two appenders are defined, one named *RollingFile* and the other *PerfRollingFile* which can be used for application and performance logging, respectively.

Change Log File Location

This node determines the location of the log file. This should be a fully qualified file path.

```
<file value="PathToLog.txt" />
```

Changing the Logging Pattern

```
<conversionPattern value="%date %-5level [%property{SessionId}] %message%newline" />
```

Configures which information is logged and the format of how it is written.

See Apache log4Net SDK Documentation – PatternLayout Class for details on how to format the conversionPattern.

Unlock the Log File

```
<lockingModel type="log4net.Appender.FileAppender+ExclusiveLock" />
```

Configures the locking model in use for the log file. To temporarily disable the write lock, you can use: log4net.Appender.FileAppender+MinimalLock

> **NOTE**
>
> This will result in a performance reduction until it is reset.

Loggers

The **root** and **logger** nodes establish *loggers* that receive information from Exago and then send it to the *appenders* that actually write the files.

The *Exago Logger* is used by the main application. The *Perf Logger* records application performance. By default, logs will then be passed to the the **root** logger unless their corresponding **additivity** attribute is *False*.

In the example file, both **Application** logging (via the <logger name="Exago Logger"> node) and **Performance** logging (via the <logger name="Perf Logger"> node) are enabled. They will capture the data then will use the <root> logger since the **additivity** attribute is not specified. The <root> logger uses the *RollingFile* appender, so both the Application and Performance log data will be written to the same file.

To write the application and performance logs to different files, comment out the <logger name="Perf Logger"> node and uncomment the <logger name="Perf Logger" additivity="False"> node. The log level and destination appender are then further defined:

Change Logging Level of a Logger

Specifies the Exago logging level: OFF, ERROR, INFO, or DEBUG.

> **IMPORTANT**
>
> Setting the log level to **DEBUG** in a production environment can have unintended performance consequences. Only use **DEBUG** level logging in production temporarily and under advisement from the Exago Support, Services or Development teams.

```
<level value="INFO" />
```

Change Appender a Logger Uses

Specify the name of an appender in the ref attribute of this node.

```
<appender-ref ref="PerfRollingFile"
```

Resolving log4net.dll version conflicts

Exago uses a specific version of the log4net.dll file that is shipped with the application. When a .NET embedding application implements a different version of log4net, there can be conflicts. The solution is to add a runtime assembly binding redirect to the **Exago web.config**.

Example

```
<dependentAssembly>
 <assemblyIdentity name="log4net" publicKeyToken="669e0ddf0bb1aa2a" culture="neutral"/>
 <bindingRedirect oldVersion="1.2.11.0" newVersion="1.2.15.0" />
</dependentAssembly>
```

See Redirecting Assembly Versions (MSDN) for more information.

Troubleshooting log4net

log4net does provide the ability to troubleshoot logging. Should you need to troubleshoot, add the following to Exagoâ€™s web.config file below appSettings:

```
<system.diagnostics>
 <trace autoflush="true">
   <listeners>
     <add
       name="textWriterTraceListener"
         type="System.Diagnostics.TextWriterTraceListener"
         initializeData="C:tmplog4net.txt" />
 </listeners>
</trace>
</system.diagnostics>
```

In addition to the above, add the following entry to the appSettings collection:

<add key=”log4net.Internal.Debug” value=”true”/>

Save the file, refresh IIS and give IIS_IUSRS full permission to the folder where logging logs are set to be stored. In the example above, that would

be â€œC:tmpâ€. Then, attempt to execute Exago reports and check the logging log file to see why Exago logging is failing.

Logging to a SQL Database

The following will describe the necessary steps to store Exago logging information in a database. The sample referenced here is configured for MS SQL Server, but log4net provides examples for other database types.

1. Create a new database table to store the log transactions

```
CREATE TABLE [dbo].[exagoweb] (
[id] [int] IDENTITY (1, 1) NOT NULL,
[timestamp] [datetime] NOT NULL,
[logger] [varchar] (255) NOT NULL,
[thread] [varchar] (255) NOT NULL,
[level] [varchar] (50) NOT NULL,
[reportId] [varchar] (255) NOT NULL,
[sessionId] [varchar] (255) NOT NULL,
[message] [varchar] (4000) NOT NULL
)
```

2. Use the following example log4net configuration file as a reference to configure log4net logging to the database:

```
<?xml version="1.0" encoding="UTF-8"?>
<log4net>
  <appender name="AdoNetAppender" type="log4net.Appender.AdoNetAppender">
    <bufferSize value="100" />
    <connectionType value="System.Data.SqlClient.SqlConnection, System.Data, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
    <connectionString value="database connection string" />
    <commandText value="INSERT INTO exagoweblog ([timestamp],[logger],[thread],[level],[reportId],[sessionId],[message]) VALUES (@timestamp, @logger, @thread, @level,
    <parameter>
      <parameterName value="@timestamp" />
      <dbType value="DateTime" />
      <layout type="log4net.Layout.RawTimeStampLayout" />
    </parameter>
    <parameter>
      <parameterName value="@logger" />
      <dbType value="String" />
      <size value="255" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%logger" />
      </layout>
    </parameter>
    <parameter>
      <parameterName value="@thread" />
      <dbType value="String" />
      <size value="255" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%-3thread" />
      </layout>
    </parameter>
    <parameter>
      <parameterName value="@level" />
      <dbType value="String" />
      <size value="50" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%-5level" />
      </layout>
    </parameter>
    <parameter>
      <parameterName value="@reportId" />
      <dbType value="String" />
      <size value="255" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%-8.8property{ReportId}" />
      </layout>
    </parameter>
    <parameter>
      <parameterName value="@sessionId" />
      <dbType value="String" />
      <size value="255" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%-8.8property{SessionId}" />
      </layout>
    </parameter>
    <parameter>
      <parameterName value="@message" />
      <dbType value="String" />
      <size value="4000" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%message" />
      </layout>
    </parameter>
  </appender>
  <root>
    <level value="INFO" />
    <appender-ref ref="AdoNetAppender" />
  </root>
</log4net>
```

3. Restart the web server and relaunch Exago.

You should no longer see information being saved to disk, and should see entries being populated in the database table instead.

Version, Connections, and Permissions

This section will detail a few useful things to check when troubleshooting an issue within Exago.

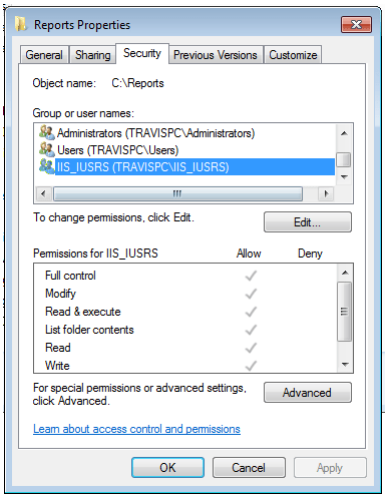
> **NOTE**
>
> This article references <WebApp>/, <WebSvc>/ and <Sched>/as a placeholder for the Web Application, Web Service API and Scheduler Service's install location respectively. The default install locations on Windows are C:Program Files\Exago\ExagoWeb,

C:Program Files\Exago\ExagoWebApi and C:Program Files\Exago\ExagoScheduler. The default install locations on Linux are /opt/Exago/, /opt/Exago/WebServiceApi/ and /opt/Exago/Scheduler/. However these paths may be changed at installation.

Verifying Folder Permissions

A common issue when installing or updating Exago is to not set read/write permissions on various folders the application uses. For these folders the user accessing them via the IIS app pool must have read write permissions. For the default app pool this is the IIS user. The folders that require read/write permissions are listed below.

- **<WebApp>/Config** â€“ this folder contains the configuration settings loaded and modified by the Admin Console.
- **<WebApp>/Temp** â€“ this folder is used to store some temporary files.
- The folder specified in the **Temp Path** of the **Main Settings**.



Security tab of the Properties dialog for the Reports folder

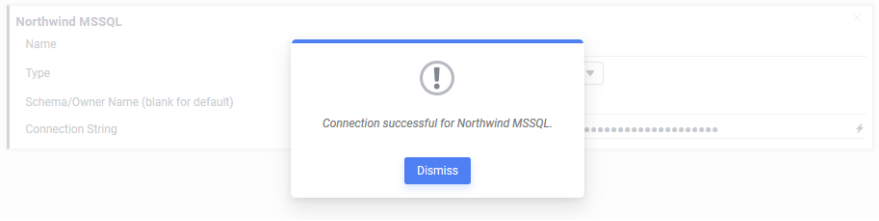
Verifying Administration Settings

Some settings in the **Administration Console** are used to connect to other programs such as databases, Web Services, .NET Assemblies or the External Interface module. Each of these items will have a **Test Connection**Â ⚡

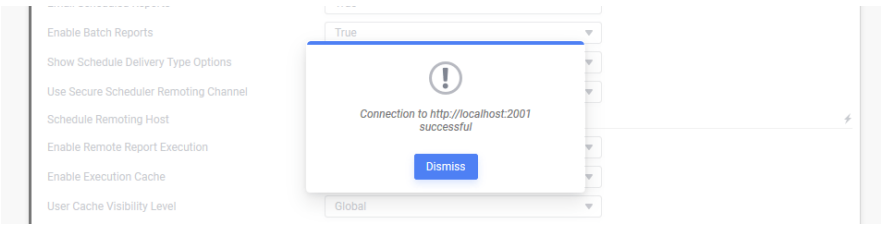
icon next to them. Click **Test Connection** to verify the connection.

The following settings can be checked:

- The **Data Source**(s) being utilized by the report you are investigating.



- **General** > **Scheduler Settings** > **Schedule Remoting Host** and **Remote Execution Remoting Host**



- **General** > **Other Settings** > **External Interface**
- **Storage Management** > **Class Name**
- **Storage Management** > **Check Database Settings**

Verifying Versions

When updating Exago occasionally issues arise because the Storage Management database schema, Scheduler Service or Web Service API have not also been updated. For the Scheduler Service and Web Service API to function properly they must be running the same version and build as the Web Application.

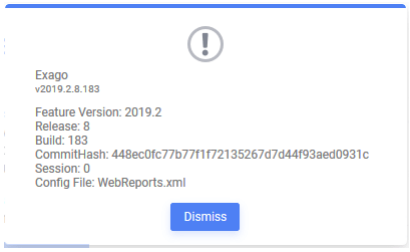
NOTE

When using the .NET API you will need to copy the DLLs from the updated Exago application to the bin directory of your host application.

Exago Web Application

To check the version of **Exago**:

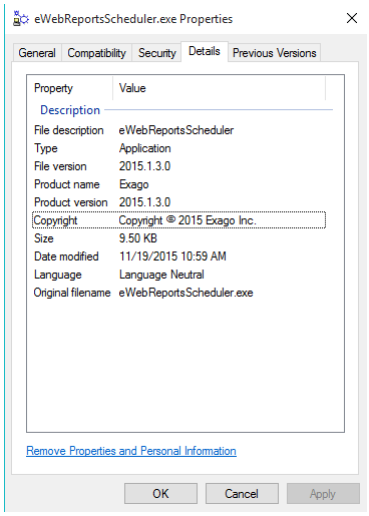
1. Navigate to the home page.
2. Press **Ctrl + Shift + V**.



Scheduler Service

To check the version of the **Scheduler Service** on Windows:

1. Locate the file <Sched>/eWebReportsScheduler.exe.
2. Right click on this file and select **Properties**.
3. Navigate to the **Details** tab.



Checking the version number of the Scheduler Service, this one is *v2015.1.3.0*

To check the version ofÂ the **Scheduler Service** on Linux:

1. Open a terminal.
2. Issue the following command

```
ikdasm -assembly <Sched>/eWebReportsScheduler.exe | grep Version
```

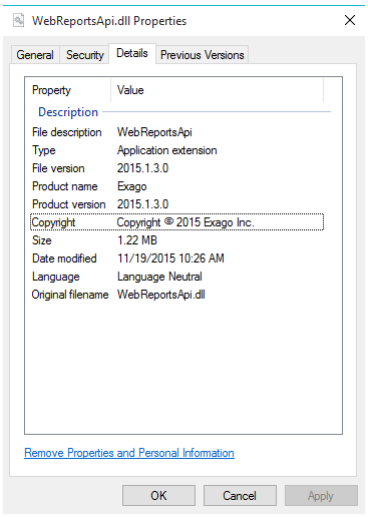
The result will resemble the following:

```
Version: 2020.1.0.108
```

Web Service API

To check the version of **Web Service API** on Windows:

1. Locate the file <WebSvc>/bin/WebReportsApi.dll.
2. Right click on this file and select **Properties**
3. Navigate to the **Details** tab.



Checking the version number of the Web Service API DLL, this one is v2015.1.3.0

To check the version ofÂ the **Web Service API** on Linux:

1. Open a terminal.
2. Issue the following command

```
ikdasm -assembly:<WebSvc>/WebReportsApi.dll | grep Version
```

The result will resemble the following:

```
Version: 2020.1.0.108
```

Storage Management Database Schema v2020.1+

To check the version/update the **Storage Management database schema**:

1. Navigate to **Admin Console** > **Storage Management**.
2. Click **Check Database Settings**.

To update the database schema:

1. Click **Prepare Database** or **Show Prepare Database SQL**.

An explanation of these two different buttons can be found in the Storage Management: Admin Console article.

Troubleshooting Report Output

If you are receiving report output that you believe to be incorrect, you have the ability to verify that the SQL query being generated by your Report returns the results that are expected. You can find the SQL query generated by the report in two different ways:

1. Report Designer Show SQL Dialog
2. By viewing the SQL statement in the log file
3. Implementing a Server Event to show the SQL instead of the report output

Report Designer Show SQL Dialog

Use the **Show SQL** option in the Advanced Report Designer to show the constructed SQL queries that will be executed with the report. See the Show SQL section of the Advanced Options article for more information.

SQL Statement in Log File

The SQL statement that is generated with every report execution can be viewed within the log file, allowing you to take the query and and run it against your data to validate its correctness.

Within the log file, the SQL statement will be formatted as such:

```
2016-01-27 10:08:31,297 INFO [founspz1bwdhpiixlvelew34] [Api.Data.DbConnect.Execute] SQL Stmt: SELECT dbo.[Categories].[CategoryName] as c0,dbo.[Products].[ProductID] as c1
FROMdbo.[Products] inner join dbo.[OrderDetails] on (dbo.[Products].[ProductID] = dbo.[OrderDetails].[ProductID]) inner join dbo.[Categories] on (dbo.[Products].[CategoryID] = dbo.[Ca
```

Show SQL Server Event

In addition to viewing the SQL statement within the log file, you can implement a Server Event to display the SQL query at runtime either globally or on a per report basis. Below you can find the code you will need to get started with the Server Event.

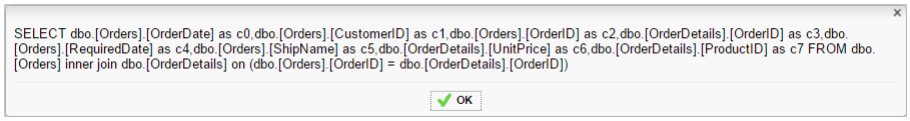
```
throw new WebReports.Api.Common.WrUserMessage(args[0].ToString(), WebReports.Api.Common.WrUserMessageType.Text);

return null;

//This event should be set to run on 'OnExecuteSQLStatementConstructed'Â
```

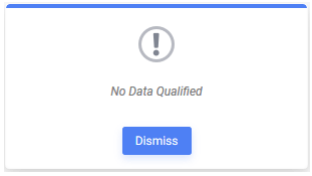
Global Event

To set the Server Event to run globally, the event will need to be implemented **OnExecuteSQLStatementConstructed**. Every report execution that occurs will then be halted, and the SQL statement will show instead of the execution as such.



No Data Qualified Options

If the data source returns zero rows, the message below is automatically generated, and report execution will end.



No Data Qualified warning message

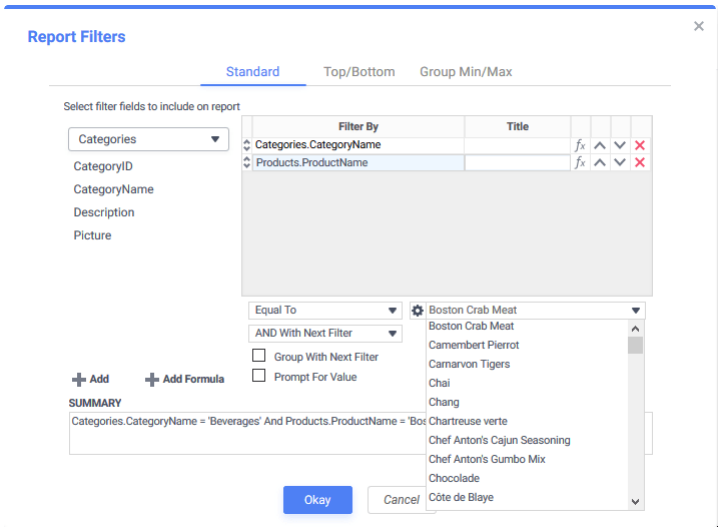
This condition is called **No Data Qualified** or **NDQ**. There are a number of ways to handle or prevent this situation:

- Enabling filter dependency can prevent improper filter values from selecting no data rows
- Allowing execution when NDQ will allow report execution to continue, with the possibility to display a custom error message in the report

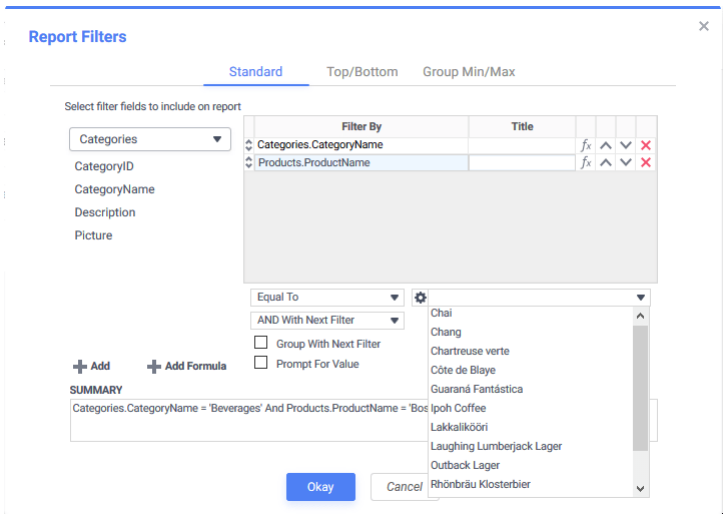
Enable Filter Dependencies

Enabling **Filter Dependencies** will prevent filters from being configured that would cause “No Data Qualified”.

For example, choosing *Beverages* as the Category and *Boston Crab Meat* as the Product Name would result in NDQ since there are no products named *Boston Crab Meat* in the *Beverages* Category. With **Allow Filter Dependencies** enabled Exago will only show Products that are in the Category selected.



With Allow Filter Dependencies disabled, all products regardless of Category are available in the dropdown



With Allow Filter Dependencies enabled, only those products in the Category previously selected in the dropdown are available

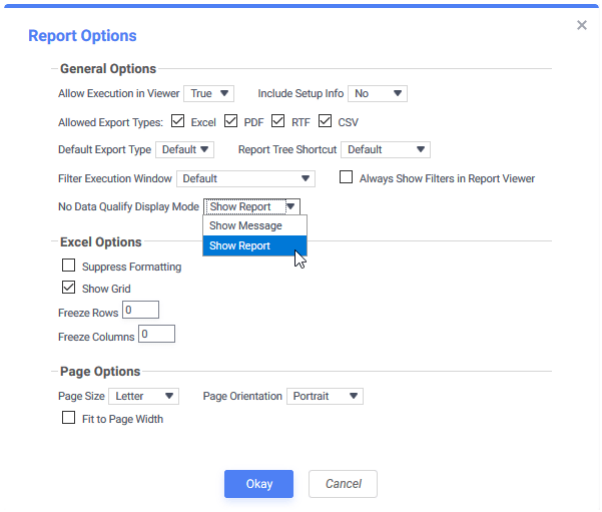
To enable filter dependency, the system administrator must:

1. Set **Admin Console** > **General Settings** > **Filter Settings** > **Allow Filter Dependencies** to *True*.

Allowing Execution When No Data Qualified

A lesser known feature of this application is the ability to execute a report even if No Data is Qualified. This functionality is useful in situations where a user still wants to see the report or when a custom message is conditionally set for when no data is qualified. When using *ShowReport* instead of *ShowMessage*, the only items that will appear in the report are those that aren't connected to the data.

1. (*v2021.1+*) Click on the **Advanced** menu on the toolbar, then choose **General Options**.
2. Set **No Data Qualify Display Mode** to *ShowReport*.
3. Click **Okay**.



Choosing *ShowReport* from **No Data Qualify Display Mode** in the Report General Options dialog

Common Use Case

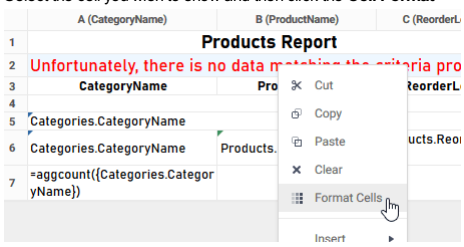
This functionality is commonly used when reports are being shown via a Dashboard and users want at least the title to appear, and/or display a custom message rather than the intrusion of an error message on the Dashboard.

Custom Message Option

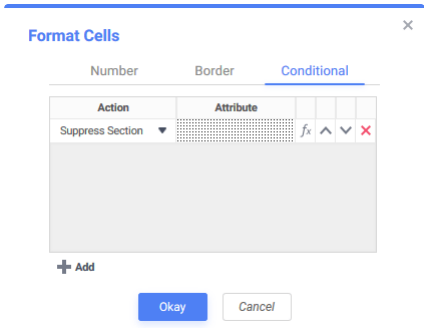
When using display mode *ShowReport* it may be preferable to leave a custom message for the viewer upon execution. This can be done using Custom Formatting.

Using a built in function, Exago can conditionally suppressÂ a cell to only show when the report has returned NDQ.

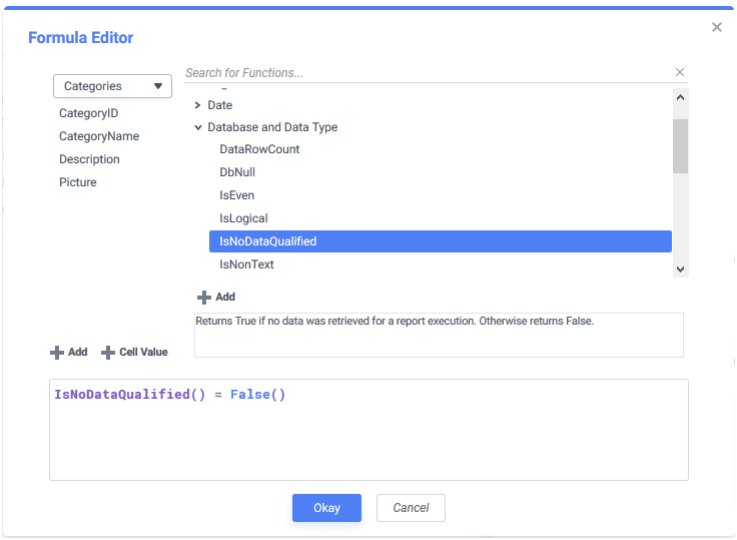
1. Select the cell you wish to show and then click the **Cell Format** icon on the toolbar or right-click the cell and click **Format Cells**.



2. Select the **Conditional** tab in the Cell Format dialog.
3. Click theÂ **Add** button to add a condition.
   1. Set the **Action** to *Suppress Section* or *Suppress Row*based on how you've setup your custom message cell.
   2. Click on the **Formula Editor**Â icon.



4. Add the formula IsNoDataQualified() = False()



5. Click **Okay**.

Upon execution, all the elements that are not data as well as the conditional message should appear within the execution.

Products Report

Unfortunately, there is no data matching the criteria provided.

CategoryName	ProductName	ReorderLevel

For more information about **Conditional Formatting**, refer to the Advanced Reports: Cell Formatting article.

Accessing the Log File

Exago keeps a text log of when certain tasks are performed. For example each time a page or report is loaded, each time an error occurs or when various phases of execution happen.

> **NOTE**
>
> This article references <WebApp>/, <WebSvc>/, <ScdlSrvc>/ and <Temp>/ as a placeholder for the install location of the Web Application, Web Service API and the Scheduler Service and Exago's **Temp Path** configuration flag respectively.

To configure logging, refer to the Application and Performance Logging article.

Web Application Log

To access the log file:

1. Ensure that in the **Admin Console** that **General** > **Other Settings** > **Write Log File** is set to *Debug*.
2. Recreate the error you are investigating.
3. Navigate to the folder specified in **General** > **Main Settings** > **Temp Path**. If this is blank go to <WebApp>Temp. If **General** > **Scheduler Settings** > **Enable Remote Report Execution** is set to *True* the report execution will instead be be recorded in the Scheduler Log.
4. Open the file **WebReportsLog.txt**. Scroll to the bottom of the file for the most recent activity.

> **NOTE**
>
> Occasionally IIS may lock this file and prevent the log from being written. To correct this problem, see **Application Logging**.

Scheduler Service & Remote Execution Log

Similar to the Web Application, the **Scheduler Service** maintains a log file. Considering the Scheduler can reside on a different machine than the main application, the log file is written where the Scheduler is installed.

Since **Remote Execution** is handled by Scheduler Services, if Remote Execution is enabled in the environment, all report execution will be logged into the Scheduler Service's log.

To access the Scheduler Service log file:

1. Set <logging> to *on* in the Scheduler Service's configuration file.
2. Rerun the scheduled report under investigation.
3. Open the file <ScdlSrvc>\eWebReportsScheduler.log. Scroll to the bottom of the log for the most recent activity.

REST Web Service API Log

Similar to the Web Application, the **Web Service API** maintains a log file. Considering the Web Service API can reside on a different machine than the Web Application, the log file is written where the Web Service is installed.

To access the Web Service API log file:

1. Set <writelog> to *True* in the Web Service API's configuration file.
2. Rerun the project that makes the API calls under investigation.

3. Open the file <Temp>\WebReportsApiLog.txt. Scroll to the bottom of the file for the most recent activity.

.NET API Log

Some implementations of the .NET API may write to a log file as well. In *v2018.1+*, the log file for the built-in logging methods defaults to <Temp>\WebReportsApiLog.txt.

Submitting a Debug Package

If the **Admin Console** > **General** > **Other Settings** > **Debug Password** is set, a client will have the ability to submit **Debug Packages** automatically to Exago, Inc. via the internet.

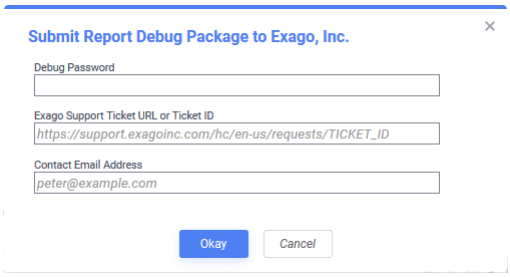
A copy of the report definition, the Exago configuration file, and the dataset used to generate the report will be transmitted to Exago's Support Team.

Automatically Creating a Debug Package

To send a Debug Package to Exago Support automatically through the Web Application user interface:

> **IMPORTANT**
>
> The automatic debug packager only works with single
>
> **Advanced Reports**, **CrossTab Reports**, **Express Reports** and **ExpressViews**. Contact Exago's Support Team directly when troubleshooting **Advanced Reports** with linked reports (a.k.a. drilldowns), **Chained Reports** or **Dashboards**.

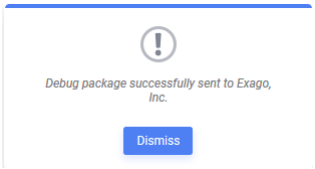
1. From the **Report Tree**, select the report under test.
2. Press **Ctrl + Shift + X**.
3. In the Debug Package dialog:



Debug Package dialog

   1. Enter the **Debug Password** (set in the **Admin Console** > **General** > **Other Settings**).
   2. Enter either the ticket ID number or the entire URL to the ticket in the **Exago Support Ticket URL or Ticket ID** field.
   3. Enter a **Contact Email Address**.
4. Click **Okay** to begin the report execution process. If the report has any prompting filters or parameters, enter values for them when prompted.

A success/failure message will display when the process finishes. Click **Dismiss**. A confirmation e-mail will be sent to the e-mail in the **Contact Email Address** field.



Manually Creating a Debug Package

If a debug package cannot be sent automatically, a package may be created manually instead. The files can be zipped and directly attached to a ticket request.

> **IMPORTANT**
>
> Before manually creating a debug Package verify that **Admin Console** > **General** > **Scheduler Settings** > **Enable Remote Report Execution** is set to *False*

To manually create a Debug Package:

> **NOTE**
>
> This section references <WebApp>/ as a placeholder for the Web Application install location. The default install location is C:Program FilesExagoExagoWeb on Windows or /opt/Exago/ on Linux. However, these directories can be changed during installation.

1. Create the <WebApp>/Debug folder if it doesn't already exist in the file system.
2. Make sure <WebApp>/Debug has the same read/write permissions as the Temp Folder.

3. Set **Admin Console** > **General** > **Other Settings** > **Enable Debugging** to *True*.
4. Execute the report under test.
5. Zip the contents of the <WebApp>/Debug together and attach them to your next response to your request.

> **IMPORTANT**
>
> Each time a report is executed, the Debug directory will be cleared. Be sure to save any needed files before creating a new package.

WebReportsDebug.xml

For special debugging purposes, a WebReportsDebug.xml file may be given to you by one of our support staff. This file should be placed in the Config directory of the Exago BI web application.

The file uses the following format:

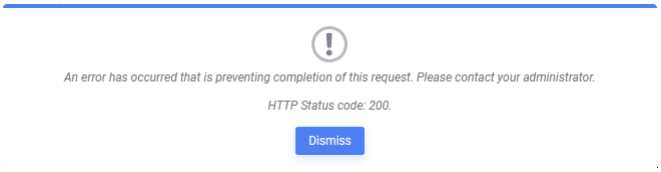
```
<?xml version="1.0" standalone="yes"?>
<webreports>
 <webreportsfn>ConfigFile.xml</webreportsfn>
 <breakoutdatasources>False</breakoutdatasources>
</webreports>
```

The following settings may be set:

- <webreportsfn> â€" the base configuration file that Exago BI should use when no config file has been specified (default: *WebReports.xml*)
- breakoutdatasources â€"- when *True*, each data object is treated as though they came from separate data sources. Causes any filtering and joins to be done in the application rather than in the Data Source. Will likely cause reduced performance.

See Full Error Details

When an error occurs in Exago, a generic error message may be displayed.



Sample error message that could occur when executing a report with errors on it

This generic message is meant to prevent end users from seeing the full stack trace of the error.

There are two ways to see detailed error messages.

1. If you are accessing Exago directly in a browser:
    1. Append **?showerrordetail=true** to the URL.
       Example: /Exagohome.aspx?showerrordetail=true
    2. Refresh the page and recreate the error.
2. If you are accessing Exago through one of the APIs:
    - With the .NET API call the method **GetUrlParamString** and set **showErrorDetail** to *True*.
    - With the REST Web Service API, set the **showErrorDetail** property of the **Session** object to *True* before starting the session.
3. Enter Exago through the API and recreate the error.

> **NOTE**
>
> The status code on the generic error message corresponds to standard HTTP status codes. For example if the status code is 408 it means there was a request timeout. For status code 200 the request completed successfully and the error lies elsewhere.

If you would like more details after seeing the full error message please see the section Read the Log File.

Error Message Glossary

This article serves as a glossary of error messages that may be displayed during the configuration or use of Exago BI. A list of potential causes and remedies is also included. For additional assistance, contact the Exago Support Team.

Choose the applicable application version from the **Viewing content for** dropdown to see only information relevant to the selected version.

Web Application User Interface

> **NOTE**
>
> This article references <WebApp>/ as a placeholder for the Web Application's install location. The default install location is C:Program FilesExagoExagoWeb (/opt/Exago/ on Linux); however, these directories can be changed during installation.

These error messages may appear in several locations throughout the Web Application user interface.

Error Message	No Data Qualified

Possible Causes	Based on the fields, formulas and filters set on this report, no records meet their criteria so there is no data to be included in the report.
Potential Remedies	Adjust the fields, formulas and filters on the report so that at least one record is returned from the data source. For more information review these resources: • No Data Qualified Options • User Support Lab â€" No Data Qualified (video)

Error Message	Session has timed out; browser page will need to be reloaded or browser restarted
Possible Causes	1. The configuration file, or the static configuration file in a static-dynamic pair was changed 2. Exago is dropping sessions 3. The web server is restarting unexpectedly. 4. Exago is accessed from a different origin as the host application over HTTP
Potential Remedies	1. Changes made to the configuration file in a single configuration file environment, or the static configuration file of static-dynamic file pair will always invalidate all user sessions. Avoid modifying these files when users are engaged with the system. For more information, review Configuration File Options and Optimizations and Modifying Configs in a Production Environment. 2. Exago recommends utilizing a State Server. Review the Setting Up a State Server article to do so. If a state server is already in use, make sure it is configured properly and running. 3. Check for and troubleshoot disruptions to the web server's uptime. 4. Review the Cross-Origin Embedding of Exago for full details about the proper way of deploying Exago cross-origin.

Error Message	An error has occurred. Please contact your administrator. HTTP Status code: 200
Possible Causes	This generic error message may occur in a variety of circumstances. Use the instructions in **Potential Remedies** below to gather more information, then search this glossary again.
Potential Remedies	Append ?ShowErrorDetail=true to the end of the URL, and try to replicate the behavior which caused the error. (Refer to the See Full Error Details article for more detail) This will allow you to identify the error in more detail and search for a solution. `http://localhost/exago/exagohome.aspx?showerrordetail=true` The error message will be displayed as a full stack trace. Use this information to correct issues such as data source connectivity trouble, incorrect web server configuration or to send the information to the Exago Support Group for additional support. Additional information can be found in the system log. Review the Accessing the Log File article to locate the log files.

Error Message	There are errors with this report, which prevents it from being executed or scheduled. Please edit report to view and fix errors.
Possible Causes	Trying to run a report that has errors on it. This may be due to data objects that are no longer available, incorrect formulas, reports on a Dashboard or Chained Report that are no longer available, or other reasons.
Potential Remedies	Instead of running the report, open it in the Report Designer. A **Report Errors** dialog will display listing the errors present on the report with potential resolutions to those errors. Follow the instructions in the Report Errors dialog and then try to execute the report again.

Error Message	HTTP 200 An error has occurred that is preventing completion of this request. Please contact your administrator.
Possible Causes	See An error has occurred. Please contact your administrator above.
Potential Remedies	See An error has occurred. Please contact your administrator above.

Error Message	Error saving report
Possible Causes	1. The Scheduler Service is not running, or the Web Application server cannot reach the Scheduler Service.
Potential Remedies	1. Start the Scheduler Service, or check the **Admin Console** > **General** > **Scheduler Settings** and the Scheduler Service Configuration file to insure they are correct and the service is reachable.

Error Message	Access Denied
Possible Causes	1. The report being executed requires access to a Data Object that is not accessible in the current Role.
Potential Remedies	1. Check the settings for the current Role and insure that all of the Data Objects that are needed are available. For more information, review Roles.

Error Message	Error: Access to the path 'C:\Program Files\Exago\Config\WebReports.xml' is denied.
Possible Causes	Permissions are not set on the config folder.
Potential Remedies	Set the IIS user permissions to Full Control on the config folder, or follow the instructions in the Folder Configuration section of the Installing Exago on Linux article to properly configure permissions.

Error Message	*Abended* appears in the **Schedule Manager**
Possible Causes	See Report has abended below.
Potential Remedies	See Report has abended below.

Error Message	Exception occurred configuring log4net: Access to the path 'C:\Program Files\Exago\WebTemp' is denied.
Possible Causes	Permissions are not set on the temp folder.
Potential Remedies	Set the IIS user permissions to Full Control on the temp folder.

Error Message	Folder or Virtual Directory not found: path=[path], mappedPath=[path]
Possible Causes	Report folder path is wrong or permissions are not set on the report folder.
Potential Remedies	1. Verify the report path is set correctly in the Admin Console. 2. Set the IIS user permissions to Full Control on the report folder.

Error Message	**NOTE** The placeholders <report path> and <content ID> are used below to reference the folder and report name, and the content ID (as a GUID) in the system. Report '<report path>:<content ID>' not found. **Example** Report 'My Reports Sales Products 2020 Profit:70aa978f-7f00-42ab-a2ec-c4548ee4ab53' not found.
Possible Causes	The report specified in the error message text was set as a Startup Report, but that report has been deleted from the system.
Potential Remedies	Remove the report specified in the error message from the Startup Reports tab of the **User Preferences**. For more information, review the Startup Reports section of the User Preferences and Context Sensitive Help article.

Error Message	Report has abended
Possible Causes	1. The execution cache for this report is invalid or out of date. 2. A fatal error has occurred on a report being executed by the Scheduler Service, and report execution has stopped.
Potential Remedies	1. Click the Execution Cache icon in the toolbar to refresh the report's execution cache 2. Investigate the Scheduler Service's log file to determine root cause.

Error Message	ERR_CONNECTION_REFUSED Unable to connect
Possible Causes	1. The Exago server or website is not running 2. The port is not open to the Exago server
Potential Remedies	1. Restart the web server and website. 2. Open the necessary port in the firewall.

Error Message	Error: No connection could be made because the target machine actively refused it
Possible Causes	The Scheduler Service is not running, or the Web Application server cannot reach the Scheduler Service.
Potential Remedies	Start the Scheduler Service, or check the **Admin Console** > **General** > **Scheduler Settings** and the Scheduler Service Configuration file to insure they are correct and the service is reachable.

Error Message	Error: Please select a cell value.
Possible Causes	1. A **Top N Filter** was added to the report, but a **Value** field was not chosen.
Potential Remedies	1. Return to the **Top/Bottom** tab of the Report Filters dialog, and either remove the **Top N Filter** by unchecking the **Limit the report to the top/bottom values** checkbox or choose a **Value** field from the dropdown.

Error Message	Error loading language file; unable to continue properly The following language file is invalid: [path]. Please check the current value in General->Main Settings.
Possible Causes	1. A language file that is specified in the configuration does not exist, or cannot otherwise be found.
Potential Remedies	1. Verify that every language file in Admin Console > General > Main Settings > Language File exists in the <WebApp>/Config/Langugesdirectory.

Error Message	HTTP 400 Bad Request
Possible Causes	1. The Web Application and the REST Web Service API are unable to communicate because their is no common encryption protocol between them.
Potential Remedies	1. Add the SecurityProtocol node to both the Web Application's and the REST Web Service API's **appsettings.config** files. There must be at least one common encryption protocol between the two. For more information, review the Application Settings article.

Error Message	Object reference not set to an instance of an object.
Possible Causes	This is a generic error message that may occur in a variety of situations in the application. If this error message appears right after installing Exago, it is typically caused by the configuration file (WebReports.xml) missing or corrupt or inadequate permissions to reach the configuration file.
Potential Remedies	Set the IIS user permissions to Full Control on the config folder, or follow the instructiuons in the Folder Configuration section of the Installing Exago on Linux article to properly configure permissions.

Error Message	HTTP 403.14 Forbidden The Web Server is configured to not list the contents of this directory.
Possible Causes	No file name is included in the URL to reach the home page or the Admin Console.
Potential Remedies	1. Add the file name to the URL. `http://<YourServer>/Exago/admin.aspx` 2. Add exagohome.aspx as the default document in the web server's configuration.

Error Message	HTTP 404 Not Found HTTP 404 An error has occurred that is preventing completion of this request. Please contact your administrator.
Possible Causes	1. Using the incorrect URL to the Exago resource 2. In the REST Web Service API, the wrong method was being used (e.g. GET when a POST was expected) 3. In an environment using cookieless sessions, the session has expired.
Potential Remedies	1. Check that the request URL path is correct. On Linux, paths are case-sensitive. 2. Refer to the List of REST Endpoints to make sure the endpoint supports the method in use. Change to a supported method if necessary. 3. Renew the session. Consider using another cross-origin embedding strategy. Review the Cross-Origin Embedding of Exago article.

Error Message	HTTP 500.19 Internal Server Error
Possible Causes	The configuration section standardEndpoints cannot be read because it is missing a section declaration. Incorrect or missing Application Pool.
Potential Remedies	Select the correct Application Pool

Error Message	System.UriFormatException
Possible Causes	1. An incorrect version of the mono runtime is installed. 2. Unpatched System.Web.dll when using cookieless sessions
Potential Remedies	1. Insure one of the supported Mono versions is installed. 2. Replace System.Web.dll with our patched version

Error Message	Uncaught (in promise) Attempting to get language for ID that doesn't exist, "FormulaEditor__Node"
Possible Causes	One or moreÂ Custom Function or Custom Aggregate Functions exist in a custom named function category, but no language file node exists for that category name.
Potential Remedies	Follow the instructions in the Custom Functions or Custom Aggregate Fucntions articles (links above) and the Multi-Language Support article to add a language file node.

Error Message	Error: Groups must go from inner to outer
Possible Causes	1. A **Top N Filter** with a **For Each** group was added to the report, but a **Group** field was not chosen.
Potential Remedies	1. Return to the **Top/Bottom** tab of the Report Filters dialog, and either remove the **Top N Filter** by unchecking the **Limit the report to the top/bottom values** checkbox or choose a **Group** field from the dropdown.

Error Message	Conditional Format Error: Conditional format return value should evaluate to true or false
Possible Causes	**Conditional Formatting** has been applied to a cell on a report, but the conditional formula that is comparing the cell value to the formatting criteria did not return a *true* or *false* value, so the conditional formatting has failed.
	1. The data type that was returned from the data source for this cell does not match the data types required in the conditional formula. Either apply a filter to eliminate invalid records from the report,

Potential Remedies	or modify the formula accordingly.

Admin Console

Error Message	Error: An attempt was made to load a program with an incorrect format.
Possible Causes	1. The SQLite driver being accessed is the wrong version.
Potential Remedies	1. Delete the SqliteInterop.dll file from the /bin directory. 2. Replace the SqliteInterop.dll in the /bin/x86/ and /bin/x64/ directories. Exago can supply these if necessary.

Error Message	**NOTE** The placeholders <data object name> and <database name> are used below to reference the name of a data object and data source name in the environment that will be referenced by name in the error message. Connection failed for <data object name>: Either the user, 'IIS APPPOOLDefaultAppPool', does not have access to the '<database name>' database, or the database does not exist.
Possible Causes	1. The application pool is attempting to access a data source using **Integrated Authentication** but the data source is not setup for that. **NOTE** A variant of this error message may also appear in the Web Application's log file.
Potential Remedies	1. Review the notes about Integrated Authentication in the **Connection String** section of the Data Sources article to configure the environment properly.

Error Message	Connection to database failed: Storage Management Failed to Initialize. Could not find either the single or double constructor. **NOTE** This error message may also appear in the Web Application main user interface as well.
Possible Causes	Exago was able to locate the Storage Management library but was not able to find one of the two possible expected constructors.
Potential Remedies	1. Verify that the **Admin Console** > **Storage Management** > **Assembly Location** and **Class Name** are correct. Click the **Validate Assembly** ⚡ icon to make sure a connection can be made to the Storage Management assembly. **TIP** It is recommended to always use an absolute file path to the Storage Management assembly. 2. The Storage Management assembly in the **Assembly Location** noted above is out of date. Check the Web Application's log file to find the absolute file path that Exago is using to load the library, and update it as needed.

Error Message	**NOTE** The placeholder <GUID> is used below to reference a Globally Unique Identifier that is appended to the end of the filename. This will vary with each instance of the error message. Error: Could not find file '/opt/Exago/Config/WebReports.xml.<GUID>' Error: Could not find file 'C:Program FilesExagoExagoWebConfigWebReports.xml.<GUID>'
Possible Causes	This error message typically appears when temporary files are not accessible 1. The permissions on the /Config directory and temp path do not allow the web server to read from one or both. 2. Exago is installed in a web farm and temporary files created by one node in the farm are unreachable by another node in the farm. 3. A cloud temporary path is configured, but the cloud service is not supported or accessible by Exago.
	1. Configure the permissions appropriately, as described in the corresponding installation article (Windows, Linux or Azure) 2. Set the <webfarmsupport> node in the configuration XML file to true. Review the Hidden Flags and

Potential Remedies	Set Up Exago in a Web Farm articles. 3. Make sure the **Admin Console** > **General** > **Main Settings** > **Temp Cloud Service** is a valid connection string to a supported cloud storage method such as Azure blob storage or Amazon S3 or EC2.

Error Message	**NOTE** The placeholder <database> is used below to reference the name of a CData driver in the environment that will be referenced by name in the error message. DbConnect Exception Could not find a valid license for using CData ADO.NET Provider for <database> on this system
Possible Causes	Licenses are generated by Exago for a specific version of the CData driver. If the driver version has changed, this error message might appear.
Potential Remedies	Check the Updating to the Latest Version article to see if the applicable driver has been updated. If it has, contact your Customer Success Manager who can have a new license key generated.

Exported File Output

These messages may appear in export output files (e.g. Excel, RTF, PDF, CSV)

Error Message	Chart Error: This chart references one or more cells that are no longer on the report.
Possible Causes	A chart has added to a report that references cells that have since been hidden, removed or relocated.
Potential Remedies	1. Correct the cell references on the chart to reference the correct locations.

Error Message	There was an error rendering the chart image. Please contact your administrator.
Possible Causes	There is an incompatibility with items on the report and the wkHtmlToImage chart rendering library.
Potential Remedies	Switch to the Puppeteer chart rendering library, by setting useWkHtmlToImage to *False* in the appSettings.config files. For more information, see the Application Settings article.

Error Message	To render a chart, please uncheck “Flatten Groups in Excel and CSV” under “Export Settings” in the Settings menu
Possible Causes	A report that includes a visualization was exported as a Microsoft Excel workbook, but the **Flatten Groups in Excel and CSV** option is enabled.
Potential Remedies	1. In the report’s Export Settings, disable **Flatten Groups in Excel and CSV**. 2. Select a different report export format 3. Remove the chart from the report.

Error Message	To see more data, please uncheck “Flatten Groups in Excel and CSV” under “Export Settings” in the Settings menu, or check “Include Detail Rows” in the Report Section Display Options dropdown
Possible Causes	A report was exported as a Microsoft Excel workbook with the **Flatten Groups in Excel and CSV** option enabled, and there are no detail rows to include on the report.
Potential Remedies	1. In the report’s Export Settings, disable **Flatten Groups in Excel and CSV**. 2. Make sure detail rows are included in the report.

Web Application Log File

Error Message	SQL aggregate requirement failed due to:
Possible Causes	A report has been disqualified for Database Aggregation.
Potential Remedies	Review the List of Database Aggregation exceptions and the Best Practices When Report Building section of the Database Aggregation article for more information.

Error Message	Error reading configuration file: /tmp/7be7125d-46ec-4cc1-998d-bf9146c6f7ed.api **NOTE** The path and file name will vary based on environment and configuration.
Possible Causes	Exago cannot read from its temporary files location.
Potential Remedies	1. Change the **Admin Console** > **General** > **Main Settings** > **Temp Path** to a directory outside of the operating system’s temporary directory. Exago will handle its own removal of temporary files. It is possible the operating system may delete files before Exago is done with them.

	NOTE

Error Message	The placeholders a.b.c.d and w.x.y.z have been used below to represent two different Exago version numbers. These numbers will vary in each environment Exception during Page_Load Type: System.Exception Message: Incompatible eWebReports API versions: API=a.b.c.d, eWebReports application=w.x.y.z. If using Web Service API, use Setup application to update, or update API reference if using .NET Assembly API.
Possible Causes	The version of WebReportsApi.dll that the host application is referencing is different than that of the installed version of Exago.
Potential Remedies	Update the reference to the current WebReportsApi.dll as described in step #4 of the Updating Recommendations article.

Error Message	Exception during Page_Load Type: System.ArgumentException Message: Illegal characters in path. or Exception during Page_Load Type: System.NullReferenceException Message: Object reference not set to an instance of an object.
Possible Causes	1. One or more reports in the storage management database contains one of the illegal characters such as ? : / * â€œ > <. 2. The ROOT content item in the storage management database contains anything in place of null for values that should be null such as the name and description fields.
Potential Remedies	1. Remove illegal characters from report names. A simple SELECT can be used to find illegal names. 2. Remove any non-null values from the ROOT item, including the string “null”.

Error Message	Exception Getting AllDbConfigs System.IO.FileNotFoundException: Could not load file or assembly ‘Newtonsoft.Json, Version=9.0.0.0, Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed’ or one of its dependencies. Report Threw Exception System.Exception: Compiler error: Metadata file `WebReportsApi.dll’ could not be found.
Possible Causes	The installed Mono version is not one supported by Exago.
Potential Remedies	Exago only supports the Mono versions that are listed in our Technical Specifications article. It is very highly recommended to let the Exago installer install the Mono runtime.

Error Message	Storage Management Failed to Initialize: Could not invoke any constructor
Possible Causes	The path in the **Admin Console** > **Storage Management** > **Database Connection** is invalid and the Storage Management database cannot be reached.
Potential Remedies	Verify that the path is correct by clicking the **Check Database Settings** button. Correct the connection string if necessary.

Error Message	Could not find file ‘<path>js.es6.resource’
Possible Causes	1. In *pre-v2019.2*, the Puppeteer rendering library is enabled and shadow copying is also enabled.
Potential Remedies	1. Disable shadow copying. More information is available from MSDN. 2. Update the installed version of Exago to the latest version.

Error Message	WebReports.Api.CloudStorage.CloudStorageBase — Exception obtaining file from cloud : The process cannot access the file because it is being used by another process.
Possible Causes	1. Using an unsupported temp file cloud storage service, or the service is misconfigured. 2. Using an unsupported session state service.
Potential Remedies	1. Ensure that the temporary file cloud storage service is setup correctly. Review the Amazon S3 File Storage or Azure 2. Ensure that the session state service is setup properly. Review the Setting up a State Server article.

Scheduler Service Log File

Error Message	System.IO.FileNotFoundException: Could not find file ‘C:working6468deff-5fcb-45e8-822f-441be90aec26.xml’. **NOTE** The path and file name will vary based on environment and configuration.
Possible Causes	The Scheduler Service could not locate some or all of its working files. This may be because the Scheduler Service’s working directory was changed or has become unavailable.
Potential Remedies	1. Verify that the <working_directory> in the Scheduler Service configuration file exists and is available for reading and writing by the Scheduler Service. 2. Verify that the <working_directory> in the Scheduler Service configuration file has not changed. If it has, copy all of the working files from the old location to the new one then restart the Scheduler Service.

Error Message	Could not create WrUserMessage for â€˜ExecuteReportEmptyMsgâ€™â€ In *v2020.1+*, the Report ID will be included in the error message e-mail (if an e-mail address is defined in the Scheduler Service's configuration file.
Possible Causes	A scheduled job produced an empty report. This is different from NDQ where the report has content but no data, it is when the execution returns a completely empty report.
Potential Remedies	Locate the report with the ID indicated in the error e-mail or in the Scheduler's log. Edit the report as necessary.

REST Web Service API Log File

Error Message	[Rest.RESTErrorHandler.OnException] Error in REST Service Type: WebReports.Rest.ExagoRestException Message: The requested report is either missing, or is not supported for this service. Source: WebReportsRest
Possible Causes	1. The report that is trying to be loaded does not exist anymore, or the path and report name provided are incorrect. 2. One of the GetExecute() methods was called on a Dashboard. 3. One of the GetExecute() methods was called on an ExpressView in a version of the application *pre-v2019.1.31*, *pre-v2019.2.19* or *pre-v2020.1.1.5*.
Potential Remedies	1. Check that the report exists in the folder that it is expected to be in. Check spelling of path and report name. 2. Dashboards are not supported by the GetExecute() methods. Use a different execution method. For more information, review the Executing Reports with the API article. 3. GetExecute() for ExpressView is supported in *v20191.31+*, *2019.2.19+* and *v2020.1.5+* only. Update to one of those versions, or use a different execution method. For more information, review the Executing Reports with the API article.

Error Message	Type: System.Exception Message: setupDs is null; setupFn: <path>.api.enc
	Source: WebReportsRest TargetSite: WebReports.Api.Api LoadApi(System.String)
	StackTrace: at WebReports.Rest.RESTHelper.LoadApi(String sid)
	
	This error message will be accompanied by an HTTP 500 error in the REST API along with this response:
	
	{
	"reason": "setupDs is null; setupFn: <path>.api.enc",
	"stacktrace": null
	}
	
	where <path> represents Exago's temp files path and a GUID

Possible Causes	The session cannot be found because one of the following may have happened: 1. The session was consumed because the **AppUrl** has already been visited. 2. An invalid session ID was supplied in the ?sid= parameter when making API calls 3. The web server was restarted
Potential Remedies	1. Do not browse to the **AppUrl** until all calls have been made, as this closes the session and prevents any further changes to it. 2. Check to verify that the session ID is correct. Discard the session and start a new one if required.

Additional Resources

For more troubleshooting assistance, review these resources:

- Troubleshooting section of the Support Site
- Admin Support Lab â€” Troubleshooting for Admins (video)