

Tech Guide

Version 2018.2

Table of Contents

Table of Contents	2
Technical Specifications	14
Windows	14
Linux	14
Data Sources	14
Supported Browsers	14
Desktop	14
Mobile	14
Types of Objects	15
AJAX Enabled	15
System Requirements	15
Configuring IIS for Exago	15
Install Prerequisites	15
Install Exago	16
Set the Application Pool	17
Set Folder Paths and Permissions	18
Check that the Web Site is Running	20
Open the Admin Console	21
Additional Notes	21
Application Pool Settings	21
Selecting the Correct App Pool	21
Ensuring the App Pool is Running	21
Installing Exago on Windows	22
Prerequisites	22
Installation	22
Create the Directory Structure	22
What's Next	23
Resources	23
Installing Exago on Linux	23
Supported distros	23
Requirements	23
Apache 2.4+	23
Nginx	24
Installation	24
Silent Installation with Parameters	24
Guided Installation	24
Apache	24
Nginx	25
Run Exago at Startup	25
Example	25
Configure Nginx	25
Example	25
Folder Configuration	26
Scheduler and Monitoring Services	26
Scheduler	27
Monitoring	27
Installing Exago on Azure	27
Which solution should I use?	27
App Service	28
Hosting Exago BI in an App Service	28
Using the .NET API with Azure	29
File Storage	29
Config File	29
Azure Connection String	29
appSettings.config	29
API	30
Reports Storage	30
Temporary Files Storage	30
Temp Cloud Service	30
Azure Affinity Cookie	31
Virtual Machine	31
Installing Optional Features	31

Legacy Maps (GeoCharts)	31
Google Maps	31
Application Themes	32
Install and Configure the Web Service	32
Web Services API	32
For Windows	32
For Linux	32
Configuring Web Services API	33
Installing the Scheduler Service	33
For Windows	33
For Linux	33
Saving Schedules to a Repository	33
About this Guide	33
Initial Setup	33
Set a Report Path	33
Installation Troubleshooting	34
Administration Console Setup	36
Data Sources	36
Building Metadata	36
Verifying the Report Path	37
Admin Console Password Encryption	37
Using the Admin Console	37
Using the API	38
Additional information	38
Scheduler Configuration	38
Starting and Changing Scheduler Services	39
Scheduler Queue	39
Background	40
How the Queue Works	40
Getting Set Up	41
Examples	42
Basic Example	42
Production Example	43
User Identification	43
userId and companyId	43
userEmail (v2018.2+)	43
Setting the current user	43
Admin Console	43
Config File	44
.NET API	44
REST API	44
Basic sandboxing	45
Schedule Manager	45
Execution Cache	45
User Preferences	45
Advanced permissions	45
Roles	45
Tenanting	46
Accessing Ids in extensions	46
Remote Execution	46
Set Up Exago in a Web Farm	47
Load Balancer	47
State Preservation	48
State Server	48
Sticky Sessions	48
Shared Folders	48
Report & Temp Folders	48
Network share	48
Cloud drive	48
Config File	49
Cloud drive	49
Additional Notes	49
Setting up a State Server	49

Setup ASP.NET State Service	49
Configure the Web Server	50
Additional Info	51
Deploying to Production	51
Contents	51
Installation	51
Data	52
API	52
Folders	52
Integration	52
Home page	53
Application theme	53
Getting Started page	53
Reports	53
Deployment	53
Security	53
Security Checklist	54
Set an external temp path	54
Disable direct access	54
Set a config password	54
Remove the plain-text config	54
Remove the Admin Console	54
Encrypt scheduler data (if applicable)	54
Disable SOAP (if applicable)	54
About the Admin Console	55
Important Security Notes:	55
Creating Additional Configuration Files	55
Accessing the Administration Console	56
Navigation	56
Main Menu	56
Tabs	57
Main Settings	57
Report Path	57
Temp Path	57
Temp Cloud Service	57
Language File	57
Temp URL	57
Allow Direct Access to Exago	57
Allow Execution in Viewer	58
Allowed Export Types	58
Default Output Type	58
Report Tree Shortcut	58
Culture Settings	58
Date Format	58
Time Format	58
DateTime Format	58
Date Time Values Treated As	58
Numeric Separator Symbol	58
Numeric Currency Symbol	58
Numeric Decimal Symbol	58
Numeric Decimal Places	58
Currency Decimal Places	58
Apply Numeric Decimal Places to General Cell Formatting	59
Apply General Currency Right Alignment	59
Server Time Zone Offset	59
Feature/UI Settings	59
Available Report Types	59
Allow Creation/Editing of Express Reports	59
Allow Creation/Editing of Advanced Reports	59
Allow Creation/Editing of Crosstabs	59
Allow Creation/Editing of Dashboards	59
Allow Creation/Editing of Chained Reports	59
Allow Creation/Editing of ExpressViews (v2016.3+)	59
ExpressView Settings	59
Allow Editing ExpressView with Live Data	59

Fields Enabled in Data Fields Tree (v2017.1.2+)	59
Join Path Algorithm (v2018.1+)	59
ExpressView Tutorial (v2018.2+)	60
ExpressView Hints (v2018.2+)	60
Express Report Designer Settings	60
Show Styling Toolbar	60
Show Themes	60
Show Grouping	60
Show Formula Button	60
Advanced Report Designer Settings	60
Show Chart Wizard	60
Chart Colors	60
Maximum Number of Chart Data Points	60
Default Chart Font	60
Show Geochart Map Wizard	60
Geochart Map Key (v2016.3+)	61
Geochart Map Colors (v2016.3+)	61
Show Google Map Wizard	61
Google Map Key (v2016.3-v2018.1)	61
Google Map Key (unlimited or JS API restricted) (v2018.1+)	61
Google Map Key (optional Geocode API restricted) (v2018.1+)	61
Google Map Colors (v2016.3+)	61
Show Gauge Wizard	61
Gauge Colors	61
Show Document Template	61
Show Document Template Upload Button	61
Show Linked Report	62
Show Linked Report Fields	62
Show Linked Report Formula	62
Show Linked Action	62
Show Insert Image	62
Show Joins Window	62
Show Advanced Joins	62
Advanced Joins Display (v2017.3.1+)	62
Allow Category Aliasing (v2017.3.1+)	62
Show Events Window	62
Show Linked Reports in New Tab (pre-v2017.3)	62
Linked Report Display (v2017.3+)	62
Allow Grouping on Non-Sorts	62
Allow Creation of Custom SQL Objects (v2018.1+)	62
Data Sources to Exclude from Custom SQL Creation (v2018.1+)	62
Dashboard Report Designer Settings	63
Prompt user for Parameters/Filters on Execution	63
Show URL Item Button	63
Allow Creation/Editing of Dashboard Visualizations	63
Use Sample Data for Dashboard Visualization Design	63
Visualization Database Row Limit (pre-v2017.2)	63
Refresh Reports/Visualizations on Dashboards Silently	63
Common Settings	63
Default Designer Font	63
Default Designer Font Size	63
Show Help Button	63
Custom Help Source	63
Show Exports in Tab	63
Show IE Download Button	63
Show Join Fields	63
Show Grid Lines in Report Viewer	64
Save on Report Execution	64
Save on Finish Press	64
Enable Right-Click Menus	64
Enable Reports Tree Drag and Drop	64
Show Report Upload/Download Options	64
Allow interactivity in Report Viewer	64
Show Toolbar in Report Viewer	64
Default interactive report viewer dock is open	64
Interactive report viewer default dock placement	64
Allow save to report design for report viewer	64
Maximum number of fields in a crosstab header or tabulation source	64
Use SVG for Application Icons (v2016.3+)	64
Application Theme Selection (v2016.3+)	64
Show Data Fields Search Box (v2017.2+)	64
Programmable Object Settings	64
Call Type Parameter Name	65
Column Parameter Name	65
Filter Parameter Name	65

Full Filter Parameter Name	65
Sort Parameter Name	65
Data Category Parameter Name	65
Data Object ID Parameter Name	65
DB Row Limit Parameter Name	65
DB Row Start Index ID Parameter Name	65
DB Row End Index ID Parameter Name	65
Filter Settings	65
Show Group (Min/Max) Filters	65
Show Top N Filters	65
Allow New Filters at Execution	66
Read Database for Filter Values	66
Allow Filter Dependencies	66
Show Filter Description	66
Default Filter Execution Window	66
Allow User to Change Filter Window	66
Include Null Values for 'NOT' Filters	66
Custom Filter Execution Window	66
Restore All Default Date Filter Functions	66
Restore All Default Formula Functions	66
Database Settings	66
Database Timeout	66
Database Row Limit	66
Row Limit Step Size (v2017.2+)	66
Disable Non-Joined Data Objects	67
Enable Special Cartesian Processing	67
Aggregate and Group in Database (v2016.3+)	67
Type-Specific Database Settings	67
Data Provider	67
Table Schema Properties	67
View Schema Properties	67
Function Schema Properties	67
Procedure Schema Properties	67
Scheduler Settings	67
Enable Report Scheduling	67
Show Report Scheduling Option	67
Show Email Report Options	67
Show Schedule Reports Manager	68
Show Schedule No End Date Option	68
Show Schedule Intraday Recurrence Option	68
Scheduler Manager User View Level	68
Email Scheduled Reports	68
Enable Batch Reports	68
Show Schedule Delivery Type Options	68
Use Secure Scheduler Remoting Channel	68
Schedule Remoting Host	68
Enable Remote Report Execution	68
Enable Execution Cache	68
User Cache Visibility Level	68
Enable Access to Data Sources Remotely	68
Remote Execution Remoting Host	69
Custom Queue Service	69
Delete Schedules upon Report Deletion	69
Default Email Subject	69
Default Email Body	69
Password Requirement (for PDFs only)	69
Custom Scheduler Recipient Window	69
Show "Reply To" Field in the Scheduler "Recipients" Tab	69
Other Settings	69
Excel Export Target	69
External Interface	69
Enable Paging In the Report Viewer	69
Renew Session Automatically	70
Write Log File	70
Enable Debugging	70

Max Report Execution Time	70
Maximum Age for Temp Files	70
Enable Web Service/Assembly Data Mapping	70
Limit Report to One Category	70
Cache External Services	70
Global Schema Access Type	70
Allow Multiple Sessions	70
Allow MD5 Hashing on FIPS Server	70
'LoadImage' Cell Function Parameter Prefix	70
Ignore Inaccessible Report Folders	70
User ID	70
Password	70
Confirm Password	71
Debug Password	71
Exago Expiration Date	71
Custom Code Supplied by Exago	71
Automatic Database Discovery	71
Customizing Data Discovery SQL	71
Data Sources	72
Name	72
Type	72
Schema/Owner Name (blank for default)	72
Connection String	73
Data Source Drivers	73
Web Services and .NET Assemblies	73
Parameters	74
Call Type (required)	74
Column, Filter and Sort Strings (optional)	74
Custom Parameter Values (optional)	74
SessionInfo (optional) (v2016.2+)	74
NET Assemblies	74
Web Services	75
Excel and XML Files	75
Connection string	75
Excel	75
XML	75
OLAP and MDX Queries	76
ODBC Drivers	77
Examples	77
CData Drivers	77
Data Objects	77
Name	77
Alias	78
Unique Key Fields	78
Category	78
ID	78
Parameters	78
Tenants Columns	78
Column Metadata	78
Schema Access Type	78
Filter Dropdown Object	78
Suppress Sort and Filter (v2018.1+)	78
Stored Procedures	79
Important Note for SQL Server:	79
Table Value Functions	79
Custom SQL Objects	80
Data Object Name	80
Data Source	80
Parameter/Insert	80
Data Object Macros	80
IfExecuteMode	80
IfExistReportDataObject	80
Column Metadata	81
Column Alias	81
Column Description	81
Plain Text	82

Language File	82
Data Type	83
Filterable	83
Sortable	83
Visible	83
Sort and Group-By Value (v2016.3+)	83
Custom Columns	84
Admin Console	84
Config File	84
Examples:	85
Retrieving Data Object Schemas	85
Data Object IDs	85
Adding Multiple Data Objects with the Same Name	85
Avoiding Issues from Changes to Object Names	85
Calling a Single Web Service/.Net Assembly/Stored Procedure	85
Reading Images from a Database	88
Joins	88
Join Types	88
Relationship Types	89
Cartesian Processing	89
Must Constraints	90
Modifying Joins	91
Advanced Joins	91
Type	91
Operator	91
Grouping	92
Parameters	92
Name	92
Type	92
Value	92
Hidden	92
Prompt Text	93
Parameter Dropdown Object	93
Stored Procedure Parameters	93
Value Field	93
Display Value Field	93
Display Type	93
Parameter Support for Dashboard URL Tiles	93
URL formatting	93
Example	93
Roles	94
About Roles	94
Main	95
General	95
Folders	95
Object	95
Filter	95
Main Settings	95
Id	95
Active	95
Include All Folders	95
All Folders Read Only	95
Allow Folder Management	95
Include All Data Objects	95
General Settings	95
Report Path	96
Date Format	96
Time Format	96
Date Time Format	96
Numeric Separator Symbol	96
Numeric Currency Symbol	96
Numeric Decimal Symbol	97
Server Time Zone Offset	97
Show HTML Export Grid Lines	97
Show Crosstab Reports	97
Show Express Reports	97
Show Styling Toolbar	97
Show Themes	97
Show Grouping	97

Show Formula Button	97
Show Advanced Reports	97
Database Timeout	97
Read Database for Filter Values	97
Show Report Scheduling Option	97
Show Email Report Options	97
Show Schedule Manager	97
Scheduler Manager User View Level	97
Allow Creation of Custom SQL Objects in Advanced Reports	98
Folder Access	98
Objects Access	98
Allow User to View Report-Level Custom SQL Objects	98
Filters Access	99
Custom Functions	99
Creating Functions	99
Name	99
Description	99
Minimum Number of Arguments	99
Maximum Number of Arguments	99
Category	99
Language	99
Reference	100
Program Code	100
Arguments	100
Name	100
Description	100
Optional	100
Variable Argument Count	100
Exago Session Info	100
Properties	100
PageInfo	100
Report	100
SetupData	100
CompanyId	100
UserId	101
Methods	101
GetReportExecuteHtml(string reportName)	101
GetParameter(string parameterName)	101
GetReportParameter(string parameterName)	101
GetConfigParameter(string parameterName)	101
WriteLog(string text)	101
GetStoredValue(string valueName, [object initialValue = null])	101
SetStoredValue(string valueName, object newValue)	101
Calling Exago Functions	101
Examples	101
JavaScript Example	101
C# Example	101
Default Custom Functions	102
MonthName	102
QuarterName	102
QuarterNumber	102
Custom Filter Functions	102
Creating Filter Functions	102
Name	102
Description	102
Filter Type	102
List Order	103
Language	103
Reference	103
Program Code	103
Example	103
Custom Options	103
About Options	103
Creating Options	104
Id	104
Type	104
Setting Options	104
Accessing Options	104
Hidden Flags	104
Setting Up Monitoring	105

Configuring monitoring	105
Example	106
Example	106
Example	106
Configuring scheduler monitoring	106
Enabling the polling service	107
Windows	107
Using the monitoring database with Exago	107
Monitoring System Overview	107
Web Application Database	107
Scheduler Application Databases	108
File Paths for Config Files & Databases	108
Monitoring Database Schema	108
SystemStatistics	108
Transform	109
Audit	109
ExecutionDetail	110
Transform	110
ReportDetail	110
Introduction to Integration	111
Styling the Home Page	111
Exago Control	112
Exago Control Properties	112
Changing CSS and Images	112
Example	112
Hovering Images	112
Finding Image Ids	112
Styling the Administration Console	113
Customizing Getting Started Content	113
Creating Additional Custom Tabs	113
Available JavaScript Functions	114
Custom Context Sensitive Help	115
Themes	116
Chart Themes	117
Crosstab Themes	117
Express Report Themes	118
Geochart Themes	118
Multi-Language Support	118
Translating Exago	119
Modifying Select Language Elements	119
Text of Prompting Filters and Parameters on Dashboards	119
Multi-Tenant Environment Integration	120
Column Based Tenancy	120
Schema Based Tenancy	120
Database Based Tenancy	121
Custom SQL Based Tenancy	121
An Overview of Exago Extensions	121
Overview	122
Custom SQL	122
Adding Custom SQL	122
The SessionInfo Object	122
Custom Functions and Custom Filter Functions	122
Server Events	123
Action Events	123
Custom Data Sources	123
External Interface	123
Custom Options	123
Introduction to Server Events	123
Event Handlers	123
C# Example	124
Arguments	124
.NET Assemblies	124
Adding Server Events to Specific Reports	124

Displaying User Messages	125
SessionInfo	125
Properties	126
PageInfo	126
Report	126
SetupData	126
CompanyId	126
UserId	126
Methods	126
GetReportExecuteHtml (string reportName)	126
GetParameter (string parameterName)	126
GetReportParameter (string parameterName)	126
GetConfigParameter (string parameterName)	126
WriteLog (string text)	126
GetStoredValue (string valueName, object initialValue = null)	126
SetStoredValue (string valueName, object newValue)	126
Calling Functions	127
Example	127
Introduction to Action Events	127
Creating Event Handlers	127
Writing Action Events	128
JavaScript	129
Example of writing client-side JS in the Custom Code window	129
Adding Action Events to a Report	129
Global Action Events	130
List of Global Events	130
Actionable UI Elements	132
Example	132
ClientInfo	132
Properties	133
Methods	133
How to Inspect Session Data and Debug Extensions	134
Server Data	135
Example	135
Example	135
Example	135
Client Data	137
Example	137
Example	138
Introduction to the .NET API	139
Referencing the Api	139
Creating an Api Object	139
Loading a Report	139
Retrieving the Role Security	140
Modifying the Report	140
Closing the Session and Executing	140
Load Reports in the .NET API	141
The ReportObject Class	141
Accessing Reports via the API	141
.NET API General Reference	141
Getting Started	142
Contents	142
API Object	142
Constructors	142
API Action	142
Active Report	143
Launch Exago and Execute Report	143
GetExecute	144
Sorts and Filters	144
Sorts	144
Filters	144
Settings	146
Parameters	146
Save to Disk	147
Role Permissions	147

Advanced Configuration	147
Data Sources	147
Data Objects	148
Joins	149
Custom Functions	149
Server Events	150
Scheduling	150
Recurring schedules (additional options)	151
Managing Files and Folders	153
Introduction to REST	154
Installing REST	154
The API	154
Authentication	155
Basic Authorization	155
ExagoKey Authorization	155
ExagoKey String	156
Request Format	156
Response Format	156
Request Data	157
Id Values	157
Constant or Enum Values	157
This Documentation	157
Using JSON	158
What is JSON?	158
Using JSON with Code	158
JSON Object Documentation	158
User JSON	158
Using the API with cURL	159
List of REST Endpoints	160
Example	160
Executing Reports with the API	162
Overview	162
API Action	162
.NET	163
REST	163
SOAP	163
GetExecute	164
.NET	164
REST	164
SOAP	164
JavaScript API	165
Background	165
Setup and Configuration	165
Create the Session	165
.NET	165
REST	165
.NET	165
REST	166
JS API Object	166
Functions	166
LoadFullUI(container)	166
ExecuteReport(container, exportType, reportPath, [udf], [updateCallback], [errorCallback])	166
ExecuteStaticReport(exportType, reportPath, udf, successCallback, [errorCallback])	167
ScheduleReportWizard(container, reportPath, [udf], [errorCallback])	167
ScheduleReportManager(container, [errorCallback])	168
LoadReportTree(successCallback, [errorCallback])	168
EditReport(container, reportPath, [udf], [errorCallback])	168
NewReport(container, reportType)	168
DisposeContainerContent(container)	169
IsAllowedReportType(reportType)	169
GetAllowedReportTypes()	169
Example:	169
Disposing Containers	169
Example:	169
errorCallback return value	169
Cookieless Sessions	170

Secured Authentication	170
Cross-Origin Resource Sharing	170
Assembly Data Sources	171
Interface	171
Method Examples	171
Additional Notes	172
Class Definition	172
Implementation	172
Data Format	172
Optimizing for Query Types	173
Using the Call Type Parameter	173
Column, Filter, and Full Filter Parameters	173
Supressing Application Filtering & Sorting:	173
Additional Considerations	175
Sort Parameter	175
Handling Multiple Data Tables	175
Multiple Methods	175
Single Method with Parameters	175
Configuration	176
Maintenance	176
System Locks	176
WebReportsApi.dll Version	176
Example	176
Application Logging	176
Logging Defaults	176
Execution start	177
Execution end	177
log4net	177
Change Logfile Location	177
Change Logging Level	177
Unlock the Log File	178
Changing the Pattern	178
Resolving log4net.dll version conflicts	178
Example	178

Technical Specifications

Windows

- Windows Server 2012+ / Windows Vista / 7 / 8 / 10
- Internet Information Services 7+
- .NET Framework 4.5+

Linux

- Red Hat Enterprise Linux 7 / SLES 12 / CentOS 7 / Fedora 21-25 / Debian 8+ / Ubuntu 14+
- Mono 4+

Apache or Nginx:

- **Apache 2.4+**
 - mod-mono 3.2.8+
- **Nginx**
 - mono-fastcgi-server4

Optional: mono-basic (support for VB.NET)

Note. SELinux is **not supported**

Data Sources

Database servers:

- Microsoft SQL Server 2000+
- PostgreSQL 7.1+
- Oracle Version 9i+
- MySQL 5.0+
- Amazon Aurora
- IBM DB2
- IBM Informix
- MongoDB
- SAP Sybase
- Apache Cassandra
- OLAP*

Database APIs:

- ODBC
- Web Services
- .NET Assembly Methods

Files:

- XML
- Excel

Supported 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.

- Google Chrome
- Apple Safari

Types of Objects

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

AJAX Enabled

- .NET API and REST Web Service API for communication between Exago and host application

System Requirements

Disclaimer: This document describes the baseline hardware recommended to operate Exago in a typical production environment. These recommendations should be taken as general advice, not as strict rules. Additional users, higher request frequency, and more complex data may increase the necessary processing power and memory. Administrators should conduct performance benchmarks for CPU and RAM utilization in order to determine an appropriate hardware level.

Minimum specifications:

Intel Xeon Multi-Core Processor, or equivalent

16GB RAM

200MB storage space

- + approx 85MB per scheduler application
- + approx 100MB per additional host application

Recommended specifications:

2 Intel Xeon Multi-Core Processors, or equivalent

64GB RAM

700MB storage space

- + approx 85MB per scheduler application
- + approx 100MB per additional host application

An additional 500MB of disk space is required to use Google Mapping.

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**

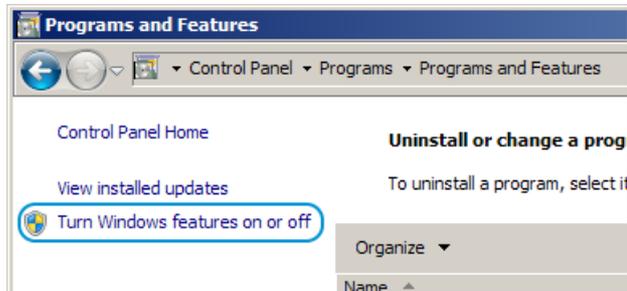
Install Prerequisites

Exago runs on Internet Information Services (IIS) version 5.1 or later, and requires Microsoft .NET Framework version 4.5 or later. In addition, Exago requires the following Windows Features to be installed:

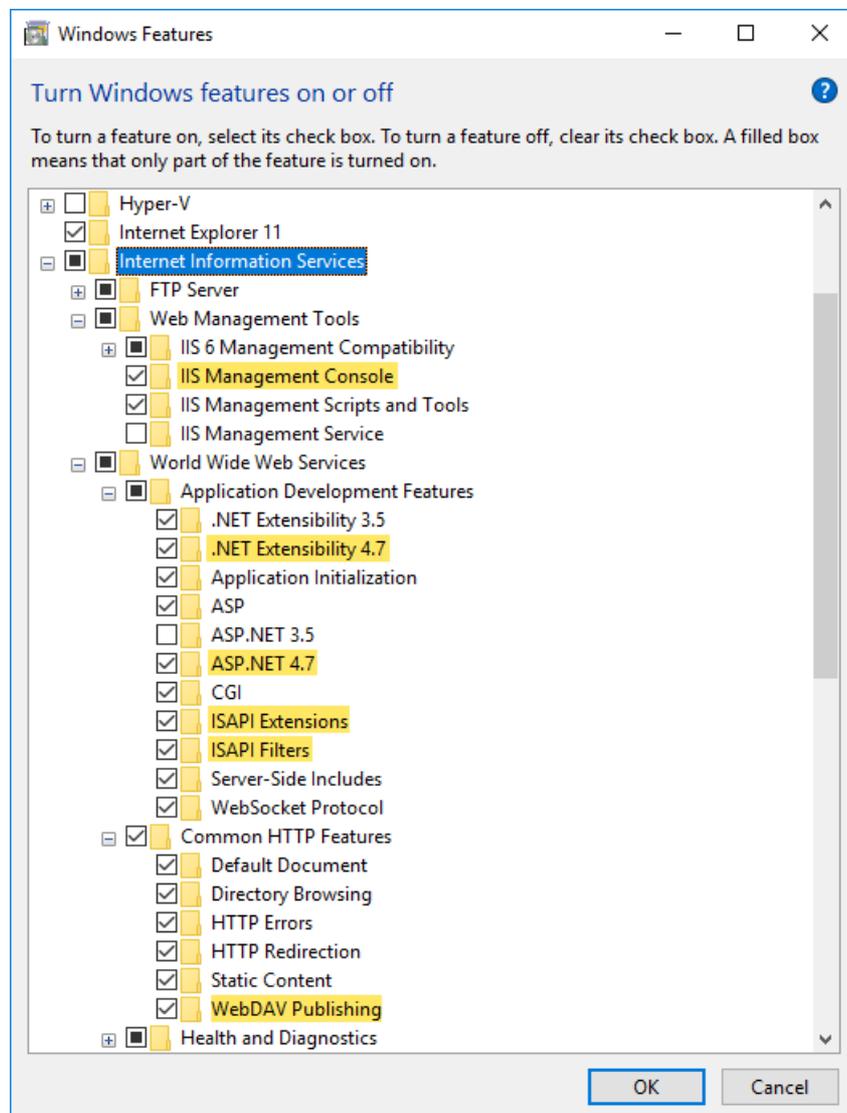
- Internet Information Services
 - Web Management Tools
 - IIS Management Console

- World Wide Web Services
 - Application Development Features
 - .NET Extensibility
 - ASP.NET
 - ISAPI Extensions
 - ISAPI Filters
 - Common HTTP Features
 - 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 click **Turn Windows features on or off**.



In the Windows Features dialog, expand **Internet Information Services** 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 necessary in order for the proper Application Pools to be available.

Install Exago

At this point, Exago can be installed.

Run the installer and follow the instructions for installing the Exago Host Application.

We do not recommend installing Exago in a managed directory such as Windows, Users, Downloads, and Desktop folders. Using one of these file paths can cause permissions conflicts.

Take note of the website which Exago is installed, and the virtual path.

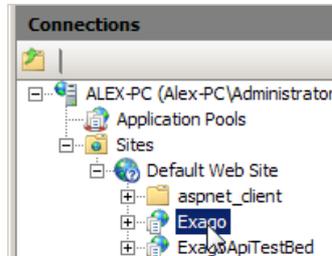
For a more detailed walkthrough, see [Installing Exago on Windows](#).

Set the Application Pool

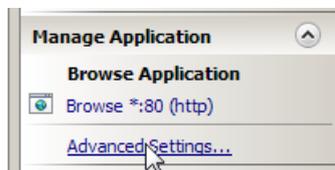
Next, verify that Exago is using a valid application pool.

Open IIS via **Control Panel > Administrative Tools > Internet Information Services (IIS) Manager**.

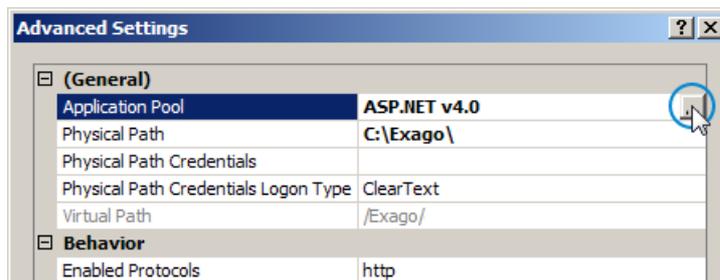
In the left-most Connections pane, select the Exago application.



In the right-most Actions pane, click on **Advanced Settings...**

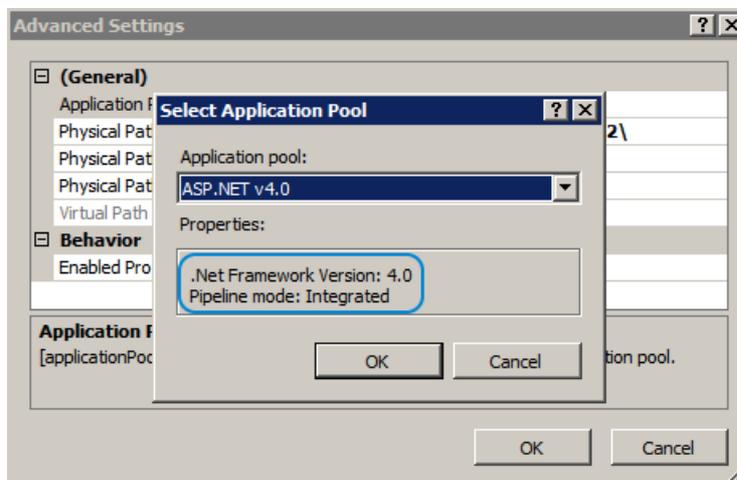


In this menu, click on the [...] button to the right of the **Application Pool**.



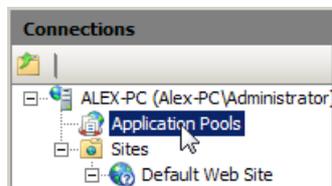
In the **Select Application Pool** menu, determine which of the app pools has the properties:

- .Net Framework Version: 4.0
- Pipeline mode: Integrated

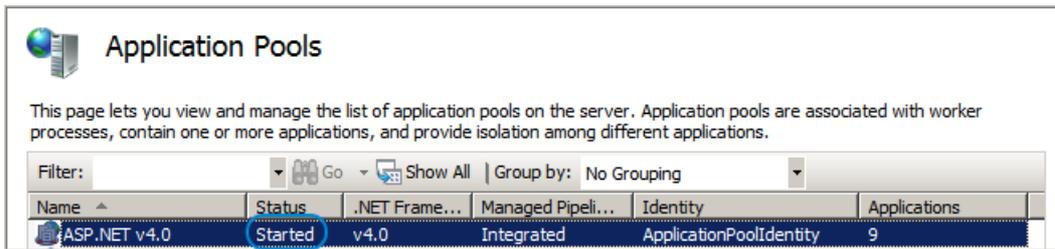


Select this app pool, then press **OK** to close the App Pool menu, and then **OK** to close the Settings menu.

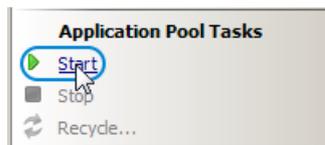
Next, ensure that the app pool is running. In the left-most Connections pane, select **Application Pools**.



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



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



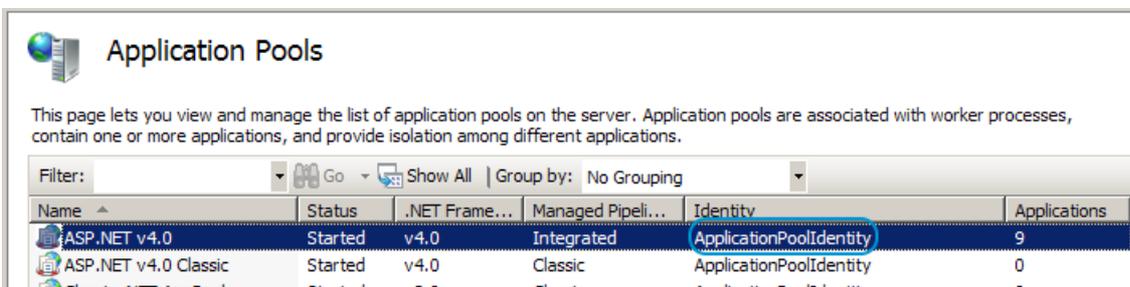
Set Folder Paths and Permissions

Exago uses Temp and Reports folders to store working data and the reports file system, respectively. Create these folders in locations suitable to your environment.

The Reports folder should not be within the Exago application directory. This could cause timeout errors while using the application.

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

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



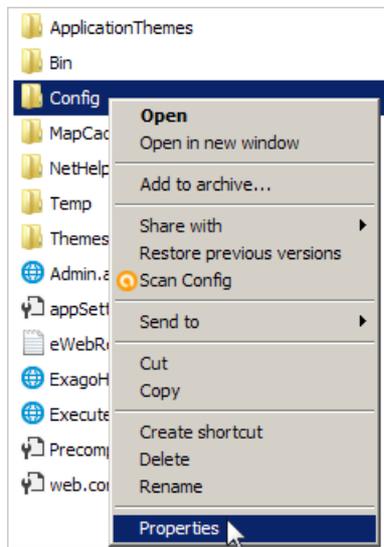
By default, this is set to *ApplicationPoolIdentity*. This corresponds with the built-in user *IIS_IUSRS*. If this is a different user, then set permissions for that user account instead.

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

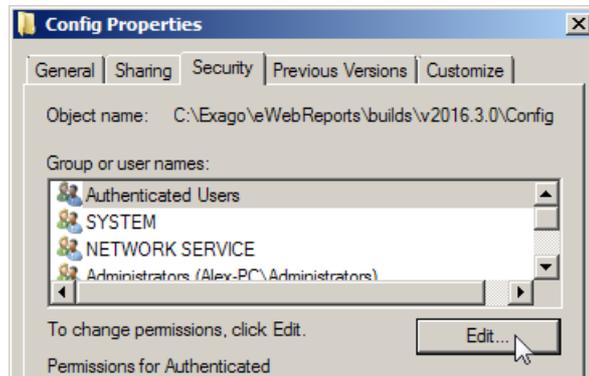
- Config
- Temp
- MapCache
- Reports

NOTE. This process may differ slightly depending on your version of Windows.

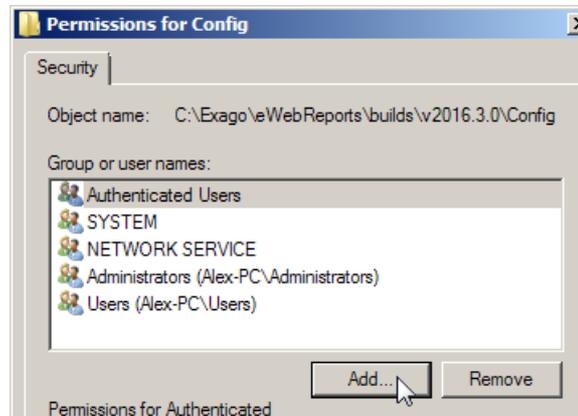
First, right-click on the folder and select **Properties**.



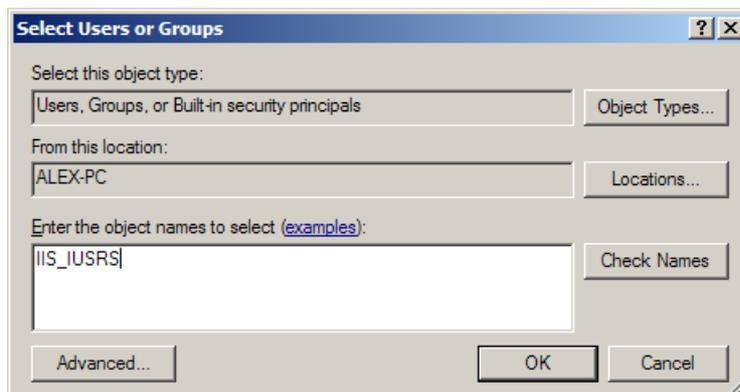
Open the **Security** tab. Then click the **Edit...** button.



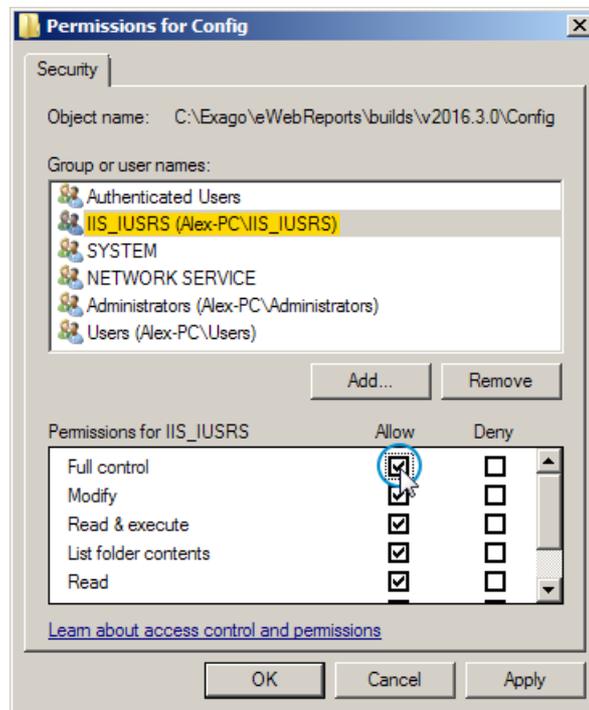
If the application pool user is not available to select, you need to add it. Press the **Add...** button.



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



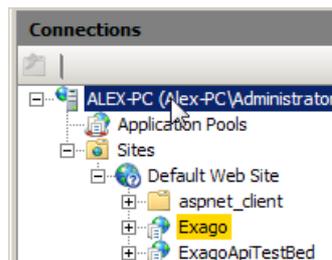
With the user selected, check the box for Allow **Full control**. Then press OK.



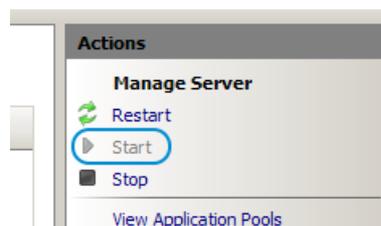
Repeat this process for every folder listed above.

Check that the Web Site is Running

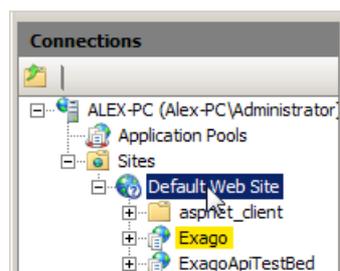
Before starting Exago, ensure that the Web Site is running. Open IIS, and in the left-most Connections pane, locate and select the web server under which Exago was installed



Verify in the right-most 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, locate and select the web site under which Exago was installed.



Verify in the right-most 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.



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 was successful.

In the Admin Console **General Settings**, set your Temp and Reports paths to the folders you previously created. Then press OK.

Then navigate to **http://[YourServerAddress]/[YourExagoVirtualPath]/ExagoHome.aspx**

If you can see the Exago UI, and you have an empty folder/reports tree, then you've set your paths correctly.

Congratulations! You've completed 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**, and a representative will be happy to help you get set up.

Application Pool Settings

When installing Exago on certain Windows configurations using .NET Framework 4.5 or higher, an additional step may be necessary.

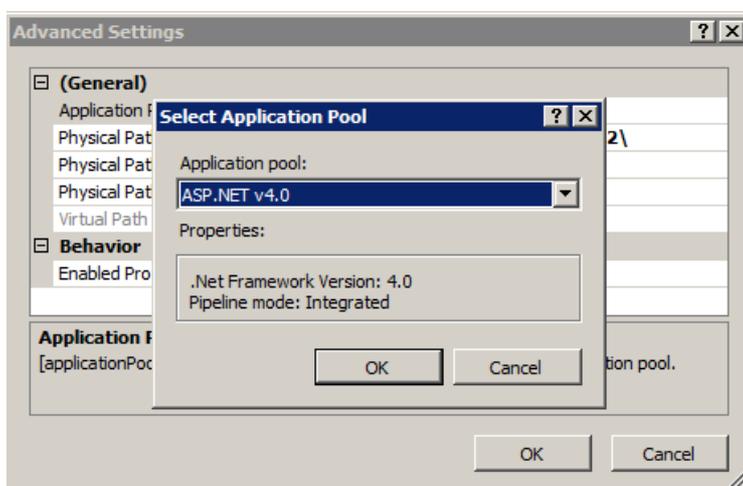
By default the Exago installer will create a website in Internet Information Services (IIS) using the **ASP .NET v4.0** app pool. In some configurations of .NET 4.5+, this app pool may be called by a different name, such as **.NET v4.5** or **.NET v4.6**. In such cases, the app pool must be set manually.

Selecting the Correct App Pool

After you've installed the Exago Application, Web Service, or Scheduler, open your IIS configuration by going to the **Control Panel**, opening **Administrative Tools**, and opening **Internet Information Services (IIS) Manager**.

In the left sidebar, select the application you've just installed. Click on **Advanced Settings...** in the right sidebar. In this menu, click on the [...] button next to **Application Pool**. In the menu that opens, determine which of the app pools is correct by looking at the Properties box.

The correct app pool may have different names, but it will be using **.Net Framework Version: 4.0, Pipeline Mode: Integrated**.

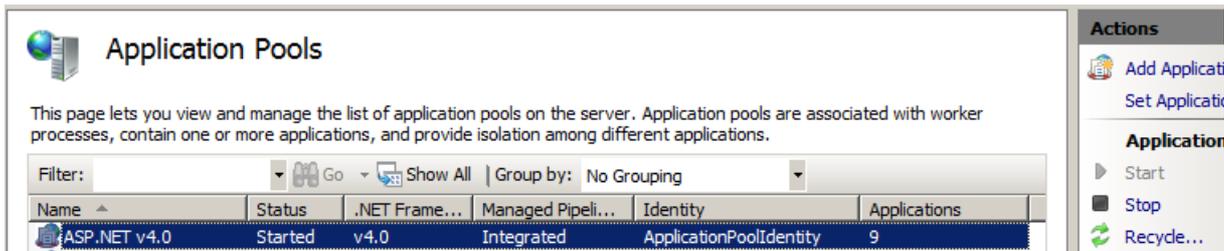


Once you've selected the correct app pool, select 'OK' to confirm your choice.

If you cannot locate an app pool with the correct configuration, please file a **Support Ticket**.

Ensuring the App Pool is Running

Once you've selected the correct app pool, you should ensure that it is running. In the IIS Manager, select the **Application Pools** page in the left column. Check the **Status** column next to the selected app pool. If does not say **Started**, select the app pool and click **Start** in the right column.



If you are still running into issues, 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 for Windows based systems.

See [Linux Installation](#) or Installing Exago on Azure for alternative operating systems.

Prerequisites

See System Requirements to ensure that you have sufficient hardware to run Exago.

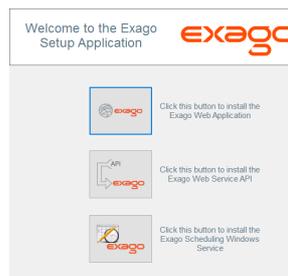
Before installing Exago, ensure that IIS is installed and configured correctly. See Configuring IIS for Exago for details. Please refer back to this guide for necessary configuration settings.

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.



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:
Right click on the folder named "Config" and click **Properties**.
In the security tab click "Edit" then "Add." Enter the IIS application pool user (default **IIS_IUSRS**).
In the "Permissions for Config" window select the user that was just created and select "Modify" or "Full Control" permissions.
Repeat this process for the **MapCache** folder.
2. Create a folder for storing reports. This folder needs to be accessible from the web server, but is not required to be on the web

server. It can reside on any server accessible by Exago via direct UNC or virtual path created in IIS.

IMPORTANT. Do not create the reports folder within the Exago application structure. Doing so will cause ASP.NET sessions to crash when report folders are created or deleted within the Exago application.

Give the Report Folder full control privileges for the IIS application pool user. Below are three examples of report paths to the folder **\ReportsRepository**:

C:\ReportsRepository – Folder is on a file system.

\\Server Name\ReportsRepository – Physical folder is on a separate server.

/ReportsRepository – Assumes an IIS virtual directory called '**ReportsRepository**' has been created to point to the folder.

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**

Specify the location of the **Report** Folder in the '**Report Path**' setting.

Specify the location of the **Temp** Folder in the '**Temp Path**' setting.

What's Next

Point your browser to the **Home Page** to verify that your installation was successful. By default this is **http://<YourServer>/Exago/ExagoHome.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 - Baseline hardware requirements.
- Configuring IIS for Exago - Necessary config details for IIS.
- Installing the Scheduler Service - Scheduler config info.
- Install and Configure the Web Service - Web service config info.
- Installation Troubleshooting - Common install problems & their solutions.
- Administration Console Setup - Initial data sources setup guide.
- Installing Optional Features - How to set up GoogleMaps, GeoCharts, and Application Themes.

Installing Exago on Linux

The following article walks through the installation process for Linux systems.

Supported distros

- Red Hat Enterprise Linux 7
- SUSE Linux Enterprise Server 12
- CentOS 7
- Fedora 21-25
- Debian 8+
- Ubuntu 14+

Requirements

- mono 4.0+

Apache 2.4+

- mod-mono

Nginx

- mono-fastcgi-server4

Optional:

- mono-basic, which provides support for VB.NET

Note: The Exago installer can automatically download and install supported versions of mono and mod-mono. Apache or Nginx must be installed prior to installing Exago.

WARNING: Exago is incompatible with SELinux.

Installation

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.

Note: Apache or Nginx must be installed prior to installing Exago.

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

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

Then run `installExago.sh` as root:

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

The installer can be run in guided or silent mode. The Linux distribution and the type and version of web server software will be detected automatically.

Silent Installation with Parameters

Usage:

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

-d <Install Path>	Default is /opt/Exago
-m <TRUE FALSE>	Whether or not to install Mono
-i <WEBAPP WEBAPI SCHEDULER>	Which component(s) to install
-a <Web App URL Alias>	Default is /Exago
-s <Web Service URL Alias>	Default is /ExagoWebApi
-y	Do not prompt for final verification before installing
-h	Show this help screen

Guided Installation

Specify an install path when prompted. Default is /opt/Exago.

If the proper versions of mono (and mod-mono) are not present in your distribution's package repository then the Exago installer can be used to download and install the correct versions. If so, the mono repositories will be added to the package manager's repository list so that they can be updated in the future.

Select which components to install:

1. Web Application
2. **Web Service API**
3. **Scheduler Service**

Note: It is possible to install any components at a later time by running `installExago.sh` again.

See either **Apache** or **Nginx** for web server configuration details.

Apache

If Apache is detected, the installer will additionally do the following:

- Download and install mod-mono, if it is not already present.
- Generate an Apache configuration file `exago.conf` in the Apache site path.
- Set read/write permissions for the current Apache user on the install paths.
 - **Config** - Permissions and path set automatically

- **Temp** - Requires read and write permissions
- **Reports** - Requires read and write permissions
- **MapCache** - Requires read and write permissions

See **Folder Configuration** to continue with the installation.

Ngixn

Ngixn 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 Ngixn. The installer will not download this automatically, so be sure to install it beforehand. (Some distributions may include it by default).

Run Exago at Startup

Two Exago scripts created during installation need to be started manually or configured to run at startup:

```
<Web App Install Dir>/bin/startExago.sh
<Web Service Install Dir>/bin/startWebService.sh
```

Example

Note: These steps are applicable for Ubuntu and Debian. They may differ for other distributions.

Link the scripts to `/etc/init.d`:

```
sudo ln startExago.sh /etc/init.d/startExago.sh
sudo ln startWebService.sh /etc/init.d/startWebService.sh
```

Make them executable:

```
sudo chmod 775 /etc/init.d/startExago.sh
sudo chmod 775 /etc/init.d/startWebService.sh
```

Add the necessary symbolic links:

```
sudo update-rc.d /etc/init.d/startExago.sh defaults
sudo update-rc.d /etc/init.d/startWebService.sh defaults
```

Then restart the server. Check that the scripts will run on startup:

```
sudo service --status-all | grep start
```

Output should include the following:

```
[ - ] startExago.sh
[ - ] startWebService.sh
```

Finally, check that Exago is running:

```
ps aux | grep Exago
```

Output should include (something like) the following:

```
... /usr/bin/mono /usr/lib/mono/4.5/fastcgi-mono-server4.exe /applications=/Exago:/opt/Exago ...
... /usr/bin/mono /usr/lib/mono/4.5/fastcgi-mono-server4.exe /applications=/ExagoWebApi:/opt/Exago/WebServiceApi
```

Configure Ngixn

The required configuration is created in a separate site file located at `/etc/nginx/sites-available/exago`. The site file is not enabled by default.

Example

```
server {
    listen 80;
    listen [::]:80;
    server_name _;
    root /var/www;

    location /<Web App Alias>/ {
        include /etc/nginx/fastcgi_params;
        root <Web App Install Dir>;
        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;
    }
}
```

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.

Note: 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
```

See **Folder Configuration** to continue with the installation.

Folder Configuration

The **Config** sub-folder of the Exago installation has read and write permissions set by default and the default path `/opt/Exago/Config` is preferred.

Create a folder for storing reports. This folder needs to be accessible from the web server, but is not required to be on the web server. The report folder can reside on any server accessible by Exago, provided a mount point is accessible on the Exago server.

IMPORTANT. Do not create the reports folder within the Exago application structure. This can cause sessions to crash when report folders are created or deleted within Exago.

1. Set the **Report** folder's read and write permissions to **775**.

```
sudo chmod 775 Reports
```

2. (Apache) Set the default ownership to the specific **<apache user>:<apache group>**
3. Specify the location of the Report Folder in the "Report Path" setting in the Admin Console:
(**Main Settings**) Report Path

Note: The group ownership on the reports directory is not mandatory, and can be changed to have other group ownership as needed for access outside of Exago.

Default **UMASK** for files written by Exago is **027** and can be changed by updating the `MonoUnixUmask` option in the generated `exago.conf` Apache configuration file.

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

1. Set the **Temp** folder's read and write permissions to **775**.

```
sudo chmod 775 /opt/Exago/Temp
```

2. (Apache) Set the default ownership to **<apache user>:root**
3. Specify the location of the Temp Folder in the Temp Path setting of the Admin Console:
(**Main Settings**) Temp Path

Set the **MonitoringService** folder's read and write permissions for the Apache user to **775**, and set the default ownership to **<apache user>:root**.

Scheduler and Monitoring Services

If necessary, configure the Scheduler and Monitoring services to run at startup.

First see **Scheduler Configuration** and **Setting Up Monitoring** to configure the services correctly.

Then do the following for Ubuntu or Debian. (This may differ for other distributions).

Add the following scripts to `/etc/init.d`:

Scheduler

```
sudo vi /etc/init.d/startScheduler.sh
```

Add the following lines, then save the file:

```
#!/bin/bash
mono-service /opt/Exago/Scheduler/eWebReportsScheduler.exe
```

Monitoring

```
sudo vi /etc/init.d/startMonitoring.sh
```

Add the following lines, then save the file:

```
#!/bin/bash
mono-service /opt/Exago/MonitoringService/Monitoring.exe
```

Make them executable:

```
sudo chmod 775 /etc/init.d/startScheduler.sh
sudo chmod 775 /etc/init.d/startMonitoring.sh
```

Add the necessary symbolic links:

```
sudo update-rc.d /etc/init.d/startScheduler.sh defaults
sudo update-rc.d /etc/init.d/startMonitoring.sh defaults
```

Then restart the server. Check that the scripts will run on startup:

```
sudo service --status-all | grep start
```

Output should include the following:

```
[ - ] startScheduler.sh
[ - ] startMonitoring.sh
```

Finally, check that they are running:

```
ps aux | grep Exago
```

Output should include (something like) the following:

```
... /usr/bin/mono /usr/lib/mono/4.5/mono-service.exe /opt/Exago/Scheduler/eWebReportsScheduler.exe ...
... /usr/bin/mono /usr/lib/mono/4.5/mono-service.exe /opt/Exago/MonitoringService/Monitoring.exe ...
```

Installing Exago on Azure

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

App Service: The Exago BI web app, web service API, and .NET API host apps can be installed as Azure app services.

Virtual Machine: Exago BI can be installed on a Windows virtual machine on Azure.

File Storage: Exago BI data can be stored and accessed from Azure storage containers.

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?

The primary limitation to Azure App Services is that the Exago Scheduler and Monitoring services are not supported. If scheduling is a

requirement then virtual machines must be used, either in addition to using an App Service for the web application and web service API, or to host the full application stack.

If scheduling is not a requirement, then App Service and File Storage are recommended. This is a more integrated solution that may deliver better performance with less overhead than a Windows Server VM. And it can be easier to scale up this solution into a web farm environment.

App Service

Exago can run as a Azure App Service. This is necessary in order to run a .NET API based host application in Azure. The following Azure resources are required:

 App Service plan

 App Service

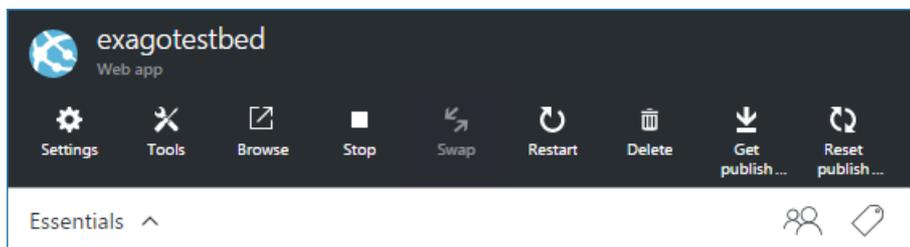
NOTE. You may require a Storage Account to run Exago as a scalable app. See **File Storage** for details.

This section is divided into two parts: **Hosting Exago in an App Service** and **Using the .NET API with Azure**. If your host application is not hosted on Azure, you can skip the second part.

Hosting Exago BI in an App Service

NOTE. This walkthrough requires a local Exago installation. See the **Exago Installation Guide** for details.

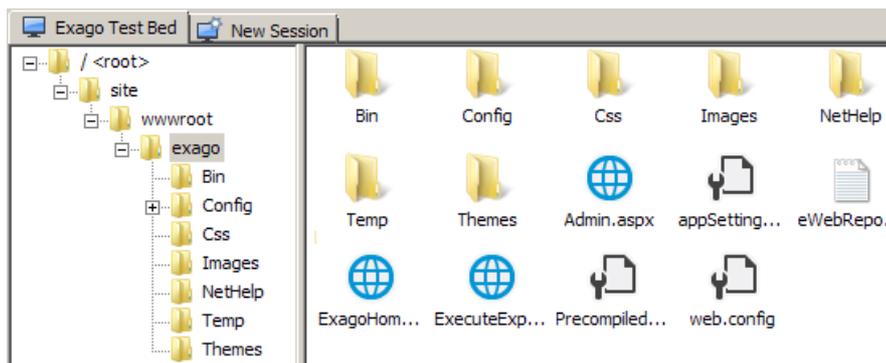
In your Azure Dashboard, create a new App Service container or navigating to an existing one.



In the App Service, navigate to **Deployment Credentials**. Add a username and password. This will allow you to FTP into your web app to transfer files.

Next, you'll need an FTP application. Open a connection using the deployment credentials you just created.

Copy your local Exago BI web app installation directory to the app service container.



In the App Service, navigate to **Application Settings**. Set the following:

- .NET Framework version: **v4.0** or Later
- Managed Pipeline Version: **Integrated**

Under **Virtual applications and directories** create a virtual directory path to the installation directory and select the **Application** check box.



Click **Save** to save your settings.

Test your installation and by navigating to the `WebAppUrl\Exago\Admin.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.

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 libraries 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");
```

Note. .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.

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

The following Azure resource is optional:

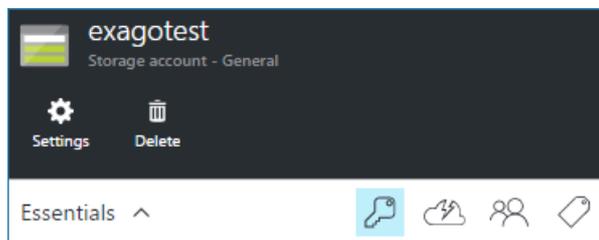
 Files storage

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.

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

Reports can be stored in File storage or Blob storage. Config files, templates, themes, and temp files must be stored in blob storage.

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 of 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.

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:

```
<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.

API

.NET API host apps cannot access the appSettings.config file. Instead, you must 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=reports");
```

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

Reports Storage

To use an Azure storage resource for report and folder management, enter a formatted connection string in the *Report Path* field in your config file.

Report Path `pathtype=azure;credentials=DefaultEndpointsProtocol=https;AccountName=acctName;AccountKey=encryptedalphan` ✓

Note. This is different from (but contains) an Azure Connection String.

The connection string uses the following format:

```
pathtype=azure;credentials='Azure Connection String';storagekey='reports';usefilestorage=false
```

- *credentials*: Your **Azure Connection String**.
- *storagekey*: The prefix of the container used to store report files. Reports are stored in "*storagekey-reports*", templates in "*storagekey-templates*", and themes in "*storagekey-themes*". This key is optional and defaults to "*wreports*".
- *usefilestorage*: If true, Reports are stored in File storage. If false, Blob storage is used. Templates and themes always use blob storage. This key is optional and defaults to false.

NOTE. Templates are automatically stored in blobs when the template upload button is used in Exago. Themes need to be manually uploaded to blob storage.

Temporary Files Storage

Azure allows a Web app to be scaled up to multiple instances on separate servers. If you are implementing this configuration, you must 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 you can (optionally) specify in the Temp Path setting in the config file (defaults to %INSTALLDIR%\Temp).

Temp Path `/exago/temp`

This is Exago's working directory — setup data for most user activity is stored and queried from here.

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** and **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 temp files a shared Blob container whenever necessary. 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.

To set up an Azure storage resource for temporary files, input a formatted connection string in the *Temp Cloud Service* field in your config file.

Temp Cloud Service `type=azure;credentials=DefaultEndpointsProtocol=https;AccountName=acctName;AccountKey=encryptedalphaname` ✓

Note. This is different from a Report Path string. This is different from (but contains) an Azure Connection String.

The connection string uses the following format:

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

The Temp files container name defaults to "*wtemp*". Currently this cannot be changed. Temp files can only be stored in blob storage.

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**.

ARR Affinity

Off

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.

Using a Remote Desktop application:

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**).

Using a 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.

Installing Optional Features

Several features require some additional configuration before they can be used. This may entail downloading some additional files.

Legacy Maps (GeoCharts)

The GeoCharts feature which was present since v2013.2 has an additional requirement for use if you are enabling it for the first time, or if you are implementing your application under a new domain name.

Our mapping features use the **Google Maps API**. Historically, this was a free solution. However, in June 2016, Google began to require paid licenses for commercial usage.

If you had GeoCharts enabled prior to June 2016, and you have not since changed the domain name of your application, this section does not affect you because you have been grandfathered in.

If you are a new user in need of mapping, we strongly suggest implementing the new **Google Maps** feature instead of GeoCharts.

If you intend to implement GeoCharts under a new domain name, then you must acquire a Google Maps API License in order to use this feature. See [this page](#) for details. Your license must include the **Google Maps Javascript API**.

For help with API keys, see [Google API Console Help](#).

To install your license file, place your license key in the following setting in the Admin Console:

(**Feature/UI Settings**) Geochart Map Key

Google Maps

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

First, you need to download and install a polygon file. This is a free download located on our **support site**. The file is named 'MapPolygonDataBase.sqlite'. Once you've obtained this file, place it in the following location in your install path (if the folder does not exist, create it):

"Application root"\MapCache

Next, if you're using Windows, you must give the IIS instance user full control permissions for this folder.

- Right click on the folder and click **Properties**.
- In the security tab click "Edit" then "Add."
 - Enter "**iis_iusrs**" then click "Ok."
- In the "Permissions for Config" window select the **IIS_IUSRS** that was just created and check the box for Allow Full Control. Then click "Ok."

If you're using Linux, you must give the Apache instance user read and write permissions for this folder.

```
chown $apacheUserID MapCache
chmod -R u+wr MapCache
```

The new 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. See [this page](#) 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.

For help with API keys, see [Google API Console Help](#).

To install your license file, first toggle the following setting to True:

(**Feature/UI Settings**) Show Google Maps Wizard

Place your unlimited license key or Google Maps JavaScript API key in the following setting in the Admin Console:

(**Feature/UI Settings**) Google Map Key (unlimited or JS API restricted)

If you used a Google Maps JavaScript API limited key in the previous setting, place your Google Maps Geocoding API key in the following setting:

(**Feature/UI Settings**) Google Map Key (optional Geocode API restricted)

Application Themes

Application themes are customizations that change the look of the Exago UI. App themes are applied for all users of the application. These are not included in the installer, and must instead be downloaded from our **support site**. All app themes can be fully customized using CSS and image editing. We will periodically introduce new ones over time.

App themes are provided as compressed folders. To add an app theme to Exago, uncompress the folder into the following location in your install path:

"Application root"\ApplicationThemes

There should be separate folders for each app theme (the default is "Basic").

Then, use the Admin Console to select the app theme from the following dropdown:

(**Feature/UI Settings**) Application Theme Selection

Install and Configure the Web Service

Web Services API

Use the following steps to install the Web Service API on Windows:

- Download the latest Windows or Linux installers [here](#).
- Make sure your antivirus software is disabled and run the installation wizard as an Administrator.

For Windows

- Click the Web Service button.
- Click **Next** to bring up the 'Select Installation Location' menu.
- In this menu specify the web site, virtual directory and physical directory where you want Exago installed. Click **Next**.

NOTE. The Web Service API must be installed on the **same server and web site** as the Exago Application.

- Confirm your location selections by clicking **Next**.
- Monitor the installation and click **Finish** when it is complete.

For Linux

- Run "**installExago.sh**" as root.
- Enter 2 for the Web Services API

NOTE. If you elect to install the Web Service API the installer will create the sub directory "**WebServiceApi**" in your previously specified install path.

- Follow the prompts

Configuring Web Services API

To configure the Web Service API edit the file 'WebReportsApi.xml' which is located in the Config sub-folder where the Web Service API is installed. The location of the Config sub-folder is determined when the Web Service API is installed. Set the following items:

- **apppath** – file path to the Exago web application. E.g. "C:\Exago"
- **throwexceptiononerror** – set to true if you want to catch exceptions in your application thrown by Exago.
- **writelog** – To turn on logging for the web service, set this to true. Then configure log4net by adding a `log4net.config` file to the \Config folder, **as described in this article**. Set web service logging to the INFO level unless directed otherwise.
Note: Grant write permissions to the ASP.NET app pool user for the log file directory.

Installing the Scheduler Service

The version and build number of the Scheduler Service must match that of the Web Application.

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 on Windows:

- Download the latest Windows or Linux installers [here](#).
- Make sure your antivirus is software disabled and run the installation wizard as an Administrator.

For Windows

- Click the Scheduler button.
- Click **Next** to bring up the 'Select Installation Location' menu.
- Specify if you want to create a new service or if you want to update an existing one.
- To create a new service set a name and location.
- Select to who the Exago Scheduler Windows Service will be installed. By default, "Everyone" is selected. Click **Next**.
- Confirm your location selections by clicking **Next**
- Monitor the installation and click **Finish** when it is complete.

As of version 2016.2.12, 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

- Run "**installExago.sh**" as root.
- Enter 3 for the Scheduler Service

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

- Follow the prompts

To configure the Scheduler Service, continue to this article.

Saving Schedules to a Repository

About this Guide

This guide will walk administrators through the process of saving schedules to a repository rather than emailing them

Initial Setup

Set a Report Path

Navigate to the scheduler install path and locate a configuration file called "**eWebReportsScheduler.xml**".

Note: This xml must be configured properly before the scheduler service will function. For more on configuring your scheduler see [configuring the Scheduler](#)

```

<working_directory>[INSTALLDIR]working</working_directory>
<default_job_timeout>3600</default_job_timeout>
<sleep_time>15</sleep_time>
<simultaneous_job_max>1</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>C:\Reports\scheduled_report_repository</report_path>
<abend_upon_report_error>true</abend_upon_report_error>
<ip_address></ip_address>
<encrypt_schedule_files>false</encrypt_schedule_files>
<max_temp_file_age>1440</max_temp_file_age>
<email_retry_time>10</email_retry_time>
</eWebReportScheduler>

```

In the eWebReportsScheduler.xml file set a repository for the "report Path" parameter.

For this example, a local directory is set

After configuring the scheduler navigate to the admin console.

Parameter	Value
Scheduler Settings	
Enable Report Scheduling	False
Show Report Scheduling Option	False
Show Email Report Options	False
Show Schedule Reports Manager	False
Show Schedule No End Date Option	True
Show Schedule Intraday Recurrence Option	True
Scheduler Manager User View Level	Current User
Email Scheduled Reports	False
Show Schedule Delivery Type Options	False
Schedule Remoting Host	
Enable Remote Report Execution	False
Enable Access to Data Sources Remotely	False
Remote Execution Remoting Host	
Custom Queue Service	
Delete Schedules upon Report Deletion	False
Default Email Subject	
Default Email Body	
Password Requirements (for pdf and excel documents)	
Custom Scheduler Recipient Window	

In the Admin Console set the "Email Scheduled Reports" parameter is set to False.

Note: Enabling "Show Schedule Delivery Options" will allow users to select per scheduler whether to send the report to a repository or to email it.

From here, schedule a report and instead of receiving it as an email the output will be saved to the set directory.

For more on scheduling reports see our [scheduling a report](#) guide.

Installation Troubleshooting

These are some common issues that may arise during an installation of Exago and some solutions for resolving them. If you are encountering problems that are not listed below, please file a support ticket.

- An error has occurred. Please contact your administrator.
- HTTP Error 500.19 - Internal Server Error
- HTTP Error 403.14 - Forbidden
- ERR_CONNECTION_REFUSED
- Error: Access to the path 'C:\[Exago]\Config\WebReports.xml' is denied.

- Object reference not set to an instance of an object.
- Exception occurred configuring log4net: Access to the path 'C:\[Exago]\Temp' is denied.
- Folder or Virtual Directory not found: path=[path], mappedPath=[path]
- Error saving report. Please contact your administrator.
- Session has timed out; browser page will need to be reloaded or browser restarted

An error has occurred. Please contact your administrator.

Append "?ShowErrorDetail=true" to the end of the URL, and try to replicate the behavior which caused the error. This will allow you to identify the error in more detail and search for a solution.

HTTP Error 500.19 - Internal Server Error

The configuration section 'standardEndpoints' cannot be read because it is missing a section declaration.

Most likely problem: Incorrect or missing Application Pool.

Solution: Select the correct Application Pool.

HTTP Error 403.14 - Forbidden

The Web server is configured to not list the contents of this directory.

Most likely problem: You're connecting to "http://[Exago]/" without specifying the home page or admin console.

Solution: Connect to "http://[Exago]/Admin.aspx" or "http://[Exago]/ExagoHome.aspx". OR set the IIS Default Document to "ExagoHome.aspx".

ERR_CONNECTION_REFUSED

Unable to connect.

Most likely problem: The Exago server or website is not running or the port is not open.

Solution: Restart the server and website. AND/OR add a firewall rule for the inbound http port.

Error: Access to the path 'C:\[Exago]\Config\WebReports.xml' is denied.

Most likely problem: Permissions are not set on the config folder.

Solution: Set the IIS user permissions to Full Control on the config folder.

Object reference not set to an instance of an object.

Most likely problem: Config file is missing or corrupt, or permissions are not set on the config folder.

Solution: Load the Admin console to generate a blank config file. If you continue to encounter the error, set the IIS user permissions to Full Control on the config folder.

Exception occurred configuring log4net: Access to the path 'C:\[Exago]\Temp' is denied.

Most likely problem: Permissions are not set on the temp folder.

Solution: Set the IIS user permissions to Full Control on the temp folder.

Folder or Virtual Directory not found: path=[path], mappedPath=[path]

Most likely problem: Report folder path is wrong or permissions are not set on the report folder.

Solution: Verify the report path is set correctly in the Admin Console. Set the IIS user permissions to Full Control on the report folder.

Error saving report. Please contact your administrator.

Most likely problem: Permissions are not set on the report folder.

Solution: Set the IIS user permissions to Full Control on the report folder.

Session has timed out; browser page will need to be reloaded or browser restarted

Most likely problem: Exago is dropping sessions or web server is restarting unexpectedly.

Solution: Make sure the reports folder is located outside the Exago filesystem. You may need to configure a state service. Check for any disruptions in your web server's uptime.

My error is not listed above.

Please file a support ticket.

Administration Console Setup

Configuring the database metadata can be done within the Exago Administration Console. These bullets detail the terms used in Exago that are consistently referred to in both the guides and the application.

- **Data Source** refers to a database or similar programmable file.
- Tables, views, functions, and procedures are stored as **Data Objects** (also called **Entities**) within Exago which can then be connected using **Joins**.
- Exago can automatically discover metadata from traditional databases (MsSQL, MySQL, etc.)

Data Sources

Add a data source by entering the Administration console <http://<YourServer>/Exago/Admin.aspx>

- In the Main Menu click the **Data** dropdown and select **Sources**.
- Add a Data Source by clicking the **Add** button ().
- Give the source a name in the **Name** row.
- Select the intended type of data source in the **Type** row. In addition to traditional db's, Exago can consume data from ODBC sources, Web services, and .NET assembly files.
- In the next row enter the **Connection String**. Connection strings vary according to database and data connector. .NET assemblies, web services, and files use the following connection strings:

Assemblies - `Assembly=C:\PATH\DLL_NAME.dll;class=ASSEMBLYNAME.NAMESPACE`

Web Services - `url=http://HOSTNAME/PATH/SERVICENAME.asmx?OPTIONALPARAM=VALUE`

Files - `File=C:\PATH\FILENAME;Type=FILETYPE` where FILETYPE is 'excel', 'xml', or 'excelXml'.

- Verify the connection to the database using the green check, if the source is through a connection string or non-RDBMS data source.
- Click **OK** to save these changes.

Building Metadata

Multiple Data Objects and joins can be created at once via Exago's automated **Discover Database Metadata** function.

- In the Admin Console (<http://<YourServer>/Exago/Admin.aspx>), select the desired **Data Source** to build metadata.

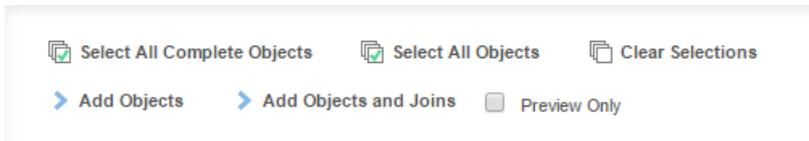


Click either the **Discover Database Metadata** icon () or right click the source name and select **Discover Object/Join Metadata**, as shown on left.

- The fields that appear with checkboxes associated to them (**Tables, Views, Functions, Procedures**, etc.) represent all the Data Objects.

NOTE. Items with a (🔒) next to their checkbox, are incomplete items.

- To add this metadata to the existing setup, use the **Select All Objects** button, **Select Complete Objects** button, or manually check the desired items. Examples of these options are below.



- Finalize the selections with the **Add Objects** or **Add Objects and Joins** options.

NOTE. Incomplete items that are added to the Data Source will require completion of the fields before adding them. If the data source does not have the metadata set, it is possible to complete this process manually, explained in [Manually Creating and Adjusting Metadata](#).

Verifying the Report Path

- In the Main Menu open the **General** drop-down and double click **Main Settings**. The first parameter in Main Settings is the **Report Path**.
- Make sure that the Report Path is pointing to the folder where read/write permissions were set earlier.

NOTE. If this step isn't complete, see [Manually Creating and Adjusting Metadata](#).

- Check this connection by clicking the green checkmark.

Parameter	Value
Main Settings [Close]	
Report Path	C:\Reports ✓
Temp Path	
Temp Cloud Service	
Language File	en-us,en-us-getting-started,en-us-tooltips ✓
Temp URL	
Allow direct access to Exago (bypassing API)	True
Allow Execution in Viewer	True
Allowed Export Types	<input checked="" type="checkbox"/> Excel <input checked="" type="checkbox"/> PDF <input checked="" type="checkbox"/> RTF <input checked="" type="checkbox"/> CSV
Default Output Type	PDF

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.

Note. 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

- Browse to the Admin Console
- Navigate to **General > Other Settings**
- Enter the desired password into the **Password** and **Confirm Password** fields

User ID	<input type="text" value="alex"/>
Password	<input type="password" value="*****"/>
Confirm Password	<input type="password" value="*****"/>

4. Click **Apply** or **OK**

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>[wOrZAgGMuU6+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.SaveData(); // 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**.

Note. 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.

Scheduler Configuration

To configure the Scheduler Service API, edit the file `eWebReportsScheduler.xml` in the folder where the scheduler service was installed.

The following settings are available:

Note: Settings that can be `true` or `false` are case sensitive and must use lower case.

`smtp_server` – The smtp server used to email reports.

`smtp_enable_ssl` - Set to `true` to enable SSL.

`smtp_user_id` - The user id that is used to login into the smtp server.

`smtp_password` - The password that is used to login into the smtp server.

`smtp_from` - The 'From' email address used in the report emails.

`smtp_from_name` - The 'From' name used in the report emails.

`error_report_to` - The email address to send error reports to.

`channel_type` - `tcp` or `http` – must match the setting of the Remote Host in the **Scheduler Settings** admin config.

`port` - The port number of the .NET remoting object used to communicate with Exago; this should also be entered in the **Scheduler Settings** of the admin config.

`working_directory` - The directory where scheduled documents and temporary files are written. The default setting `[INSTALLDIR]working` creates a `working` folder in the scheduler location.

`default_job_timeout` - The maximum number of seconds any report execution is allowed. If an execution reaches a maximum number of seconds an email will be sent to the address specified under `error_report_to`.

Note: This setting is outdated. Use `max_job_execution_minutes` instead. Do not use both, as this could result in inconsistent

timeout behavior.

`report_path` - A path to specify where to save reports when 'Email Scheduled Reports' is set to `False` in the admin config. For more details see **Saving Scheduled Reports to External Repository**.

`sleep_time` - The time interval (in seconds) used for polling for scheduled reports to execute.

`simultaneous_job_max` - The maximum number of report executions that can occur simultaneously. This setting is based on the resources available of the server where the scheduler is installed.

`logging` - 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` - The number of hours that a completed, deleted, or aborted job will be saved for viewing in the schedule reports manager. Set to 0 to flush jobs immediately upon completion. Set to -1 to disable automatic flushing.

`sync_flush_time` - The flush time for synchronous (non-scheduled remote) jobs.

`email_addendum` - Text that will be added at the end of email body. Use `\n` to insert lines.

`external_interface` - This is optional and overrides the value set in the admin config. The advantage of setting the value here is that the existing scheduled reports that have a previous external interface value will take the new value. For more details see **External Interface**.

`abend_upon_report_error` - 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` - Binding IP address for the Scheduling Service. Most commonly used when the server has multiple Network Interface Cards (NICs).

`encrypt_schedule_files` - Set to `true` to encrypt the files created by the scheduling service. All existing schedules will be encrypted the next time the service is started.

`max_temp_file_age` - The number of minutes between each "flush" of the temp files created by the scheduling service. The default is 1440 minutes (24 hours).

Note: Making this value 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` - 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+) - Maximum amount of time (in minutes) to run an execution job before timing out. If the job times out, the schedule will be marked as 'Aborted'.

Note: Do not set both this setting and `default_job_timeout` at the same time. This could result in inconsistent timeout behavior.

`secure_channel` (v2016.3+) - Set to `true` to allow receipt of encrypted data from hosts. The setting **Use Secure Scheduler Remoting Channel** must be `true` in the admin config.

`security_protocol` (v2016.3.4+) - Specify which security protocol(s) the scheduler should use. Possible values: `Ssl3`, `Tls`, `Tls11`, `Tls12`, `Tls13` (.NET v4.6+). Separate multiple values with commas (,).

`service_name_tag` - For manual installation of scheduler services using Visual Studio *installutil.exe*, this field is appended to the end of the service name. Useful for installing multiple services on the same server. *installutil.exe* must be in the same folder as the scheduler configuration file.

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.

Scheduler Queue

The 2016 release of Exago introduces a new powerful feature to the Report Scheduler: The **Scheduler Queue**. The Queue is a custom-built application library that sits in between the Exago core application and any number of scheduler instances and handles how schedule

traffic is managed. The 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

First, some background. The way in which Exago has historically handled report scheduling, and the default behavior without using a queue, is the following.

NOTE. For this discussion, it's important to define some terms:

A *Schedule* is a term for all of the information that is set when creating a schedule in the Schedule Manager. This information is usually stored as an xml file in a repository. Schedules can be accessed from the API using the ReportSchedule class.

Each Schedule contains some interpreted data that tells the schedulers when to run it. This information is called a *Job*. Jobs can also be stored separately from schedules. Jobs can be accessed from the API using the QueueApiJob class.

The process whereby a scheduler runs a report at a specified time and emails or saves the information is called an *Execution*.

Within the host application, all scheduler instances are listed in the configuration xml file:

```
| Schedule Remoting Host    tcp://schedulerHost1:2001, tcp://schedulerHost2:2001, tcp://schedulerHost3:2001 ✓
```

When a schedule is created in the UI, the host application sends the job to schedulers starting with the first and moving down the list ("round-robin" style). The queried scheduler stores the schedule xml in a local working directory. This acts as a repository for the scheduler's unique set of jobs.

From this point, each scheduler acts independently. The host application has no idea what happens to schedules after they are sent out successfully. Likewise, the schedulers have no more communication with the host application with regard to report execution.

A word about the Schedule Manager: You can view and edit schedules from the UI using the schedule manager, but this is essentially a combined front-end for the schedulers' existing files. If a scheduler is offline you will simply not see its schedules in the list (there will be a warning message). The schedule manager has no impact on the host application.

Schedulers 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 knows to run the job. The scheduler will perform its duty and then alter the schedule xml 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**.

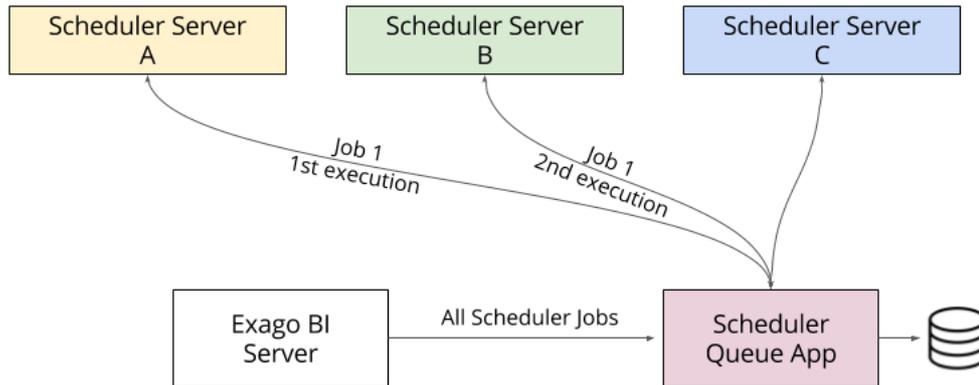
Load Balancing issues: Ideally, unoccupied schedulers would receive new jobs. This way stacks of unexecuted data do not build up on individual schedulers, leading to imbalanced load and potential time loss. But the host application has no idea which schedulers will be busy when, and no idea how long jobs will take to run. The randomness of round-robin job assignment could cause jobs to build up inordinately on one scheduler.

Outages: Once the host sends out a schedule, as far as it's concerned, it's finished. If a scheduler goes offline unexpectedly the host has no recovery function. The job will simply be delayed until the scheduler is restarted, which, to some extent, defeats the purpose of running jobs on a schedule. There is also no function to move schedules from one scheduler to another.

How the Queue Works

The Scheduler Queue is a custom .NET or Web Service library which aims to handle scheduling in a much more robust manner. It's important to note that the queue is entirely customizable. You are only required to implement all 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 schedulers. Later on, we provide a pre-built **Example** that can be used as-is with minimal modifications, or altered as you see fit.

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

The host and scheduler applications all make calls to the queue at certain points during their runtime. In particular, schedulers 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 schedulers to the queue and how it can aid a typical multi-scheduler configuration.

When schedulers are configured to use the queue, their behavior changes somewhat.

Recall that in the default configuration, schedulers store their unique schedules in a local working directory, from which jobs are queried for execution.

Now, schedulers 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, schedulers lock the queue so that only one may access it at a time. Additionally the queue sets a job's status to "running" while it's active, so that other schedulers know to ignore it. (The provided **Example** also saves a temporary file in the job repository to indicate which scheduler is handling a running job).

NOTE. Schedulers still use a local working directory for temporary files.

This has several advantages. First, schedulers 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, since a scheduler will be responsible for only one job at a time. If a scheduler goes offline in the middle of a job, the queue can be used to gracefully handle incomplete jobs (this is not present in the provided **Example**).

Next, jobs are now distributed much more evenly between the schedulers. We no longer have the problem where, due to their independence, schedulers will build up excessive numbers of jobs. Jobs will only be assigned to available schedulers.

Finally, since this allows us to control what data is being sent and received to the schedulers and the file system, we could implement any custom load balancing solution we wanted.

Getting Set Up

Setting up the queue is a multi-part process which depends on your desired configuration. We'll discuss some constants and some potential variations.

First we need to write the scheduler queue. This is discussed in more detail in the **Example** section. 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 queue interface:

```
public static string[] GetJobList(string viewLevel, string companyId, string userId)
```

Called from the Exago UI to populate the jobs in the Schedule Manager.

```
public static string GetJobData(string jobId)
```

Called from the Exago UI Schedule Manager to get the full job XML data for a job.

```
public static void DeleteReport(string reportId)
```

Called from the Exago UI when a report is deleted.

```
public static void RenameReport(string reportId, string reportName)
```

Called from the Exago UI when a report is renamed.

```
public static void UpdateReport(string reportId, string reportXml)
```

Called from the Exago UI when a report is updated.

```
public static void Flush(string viewLevel, string companyId, string userId)
```

Called from the Exago UI Scheduler Manager in response to a click on the Flush button.

```
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 services and the Exago UI 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 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.

The host application config and each scheduler config must contain the path to the scheduler queue assembly or web service class in the following format:

```
Assembly=Path\To\Assembly.dll;class=Namespace.Class
```

You can set the path in the host app by using the Admin Console and setting the following field in the Scheduler Settings:

Custom Queue Service `Assembly=Path\To\Assembly.dll;class=Scheduler.SchedulerQueueClass` ✓

Or by setting the field `<schedulerqueueservice>` in the config file,

Or by setting the field `Api.SetupData.General.SchedulerQueueService` via the API at runtime.

In each scheduler application, set the field `<queue_service>` in the scheduler config file.

Next, determine how you'll be accessing your schedules. 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

If you're using folder management, you can implement the those methods in the queue assembly (see **Report and Folder Storage/Management** for more information).

Examples

We provide the following two examples for reference.

Basic Example

The first is a basic example designed to showcase how the Scheduler Queue works. It is not suited for use in a production environment. However, it can be quickly compiled and used for testing, with minimal setup.

This example uses a directory for schedule storage and fully implements the Schedule Manager. It supports unlimited scheduler services, and implements simple versions of load balancing and error recovery.

Download the example here. To compile, set the `QueueDirectory` global variable, rename the file with a `.cs` extension, and add it to a Visual Studio project.

Production Example

The second example, generously provided by **SofterWare**, is a full-fledged production-ready implementation of the Scheduler Queue. This demonstrates how the Scheduler Queue can use a database for schedule storage, which has significant performance advantages over using a file system.

This Queue generates schedule data dynamically and on-the-fly, implements advanced tenanting, and uses a Server Event to implement custom emailing behavior. Note that temporary execution files must still be written to the file system.

Exago Inc. clients may download this example from the Downloads page. This example will require significant customization for your environment.

This code was originally created by Dave Killough and SofterWare, Inc. SofterWare has released it for Exago customer use in August 2017.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS, ORIGINAL COPYRIGHT HOLDERS, OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

User Identification

Exago does not have native user authentication. User logins must be handled in a security layer in the embedding application. After a user logs in, the application should pass identification parameters to the Exago API, which you can use to set permissions.

userId and companyId

Exago has two built-in parameters which are used to store identifying information: **userId** and **companyId**. These parameters are used in conjunction with the **Schedule Manager** and **User Preferences**, and they are automatically passed to any extensions which may need to access authentication. **Folder Management**, **External Interface**, **Scheduler Queue**, and any extension which can access sessionInfo (such as **Assembly Data Sources** or **Server Events**) can retrieve these parameters in relevant methods.

Often **userId** corresponds with a unique user or log-in, and **companyId** represents a group of users with shared characteristics. 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. For more information about this parameter see Scheduler Reply To Address.

Setting the current user

(v2017.2+)

The **Id** parameters are added by default in the base config file with blank values.

(pre-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

As created in the **Admin Console**:

userId	
Name	userId
Type	string
Value	
Hidden	True
Prompt Text	
Parameter Dropdown Object	

companyId	
Name	companyId
Type	string
Value	
Hidden	True
Prompt Text	
Parameter Dropdown Object	

Config File

As 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

As created in the .NET API:

Note: "DataType" defaults to `DataType.String`, so the call is omitted.

```
Parameter userId = api.Parameters.NewParameter();
userId.Id = "userId";
userId.Value = "user_224";

Parameter companyId = api.Parameters.NewParameter();
companyId.Id = "companyId";
companyId.Value = "company_17";
```

For setting the **userEmail** parameter (v2018.2+):

```
Parameter userId = api.Parameters.GetParameterById("userEmail");
userId.Value = "user_224@company_17.com";
```

REST API

As created in the REST API:

Note: "DataType" defaults to "String", so the call is omitted.

POST /parameters

```
{
  "Id": "userId",
  "Value": "user_224"
  ...
}
```

POST /parameters

```
{
  "Id": "companyId",
  "Value": "company_17"
  ...
}
```

For setting the **userEmail** parameter (v2018.2+):

POST /parameters

```
{
  "Id": "userEmail",
  "Value": "user_224@company_17.com"
  ...
}
```

Basic sandboxing

Setting the **userId** and **companyId** parameters has several effects in the Exago interface.

Schedule Manager

By default, the **Schedule Manager** will show only schedules belonging to the current **userId**. This can be changed by modifying the following **Admin Console** setting:

Scheduler Manager User View Level (**Scheduler Settings**)

- 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 following setting:

User Cache Visibility Level (**Scheduler Settings**)

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 by **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 **companyId** 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 report execution by **userId** "Alex":

```
2017-03-07 14:50:49: Report 'Test\Product Sales Report' executed by user 'Alex'.
```

Most extensions have methods which can access **userId** and **companyId**. In addition, the parameters are accessible through *sessionInfo*. So any extensions which can access *sessionInfo* can also access **userId** and **companyId**, even if methods do not explicitly implement them.

The following server event automatically adds the **userId** to the description text whenever a report is saved.

Global Event Type: *OnReportSaveStart*, **References:** *WebReports.Api.Reports*

```
Report report = sessionInfo.Report;
string userId = sessionInfo.UserId;

if (!report.Description.EndsWith(userId)) { report.Description += ("\n" + userId); }

return null;
```

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 *simultaneous_jobs_max* setting.

NOTE. In versions prior to v2016.2.12, machine load is not taken into account, and jobs are sent to different servers in the order that they are specified ("round-robin").

The following instructions provide an overview for setting up report execution on remote servers:

On each remote server:

- Install the Exago Scheduler Service. For detailed instructions see: **Scheduler Service Installation**.
- The following conditions must be met:
 - The Scheduler version must match the Exago Application version.
 - The Scheduler's language files and the Exago Application's language files must match.
 - Any custom assemblies must be present in the Scheduler directory.
- Configure the Exago Scheduler. 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**.

NOTE. Multiple scheduler services can point to the same repository.

In the Exago Application:

- Using the Admin Console, open the **Scheduler Settings**
 - Set 'Enable Remote Report Execution' to True in the **Report Scheduling Settings**.
 - In 'Remote Execution Remoting Host' list the servers you want to use delineated by commas or semicolons (ex. http://MyHttpServer1:2001, tcp://MyTcpServer:2121). The servers will be prioritized based on the listed order.

NOTE. When multiple remote execution hosts are enabled, the Exago application will prioritize the one with the lowest number of queued jobs.

NOTE. When an execution host is used for both scheduling and remote execution, the Exago application will place immediate priority on Remote Execution tasks.

Set Up Exago in a Web Farm

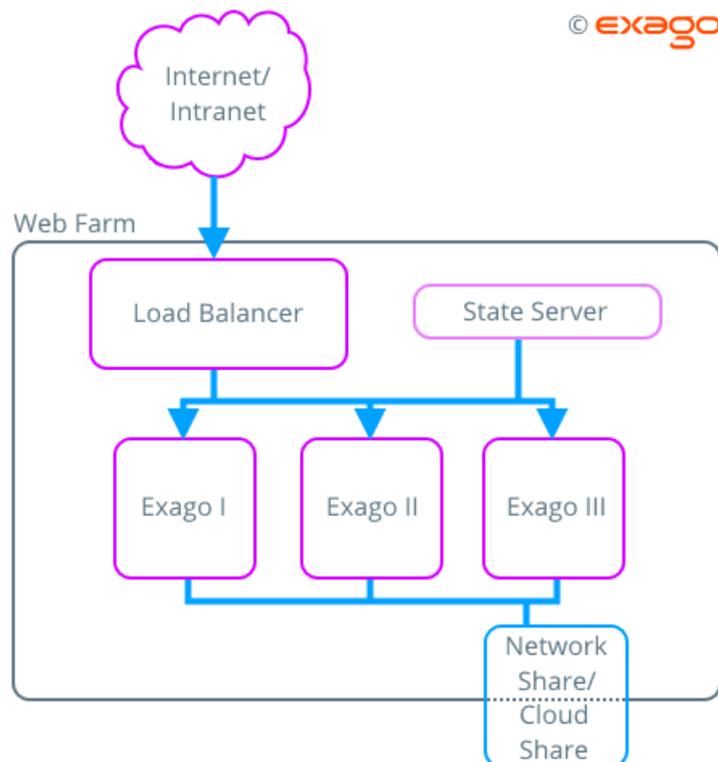
One method for running Exago in a distributed environment is using a Web Farm to run multiple instances of the web application. This may be desirable for load balancing purposes.

There are several guidelines for setup. Each will be discussed in detail in separate sections.

- **Load Balancer** A load balancer server is required as the point of entry for client requests.
- **State Preservation** Since Exago is a stateful application, either a State Server or sticky sessions (affinity cookie) must be enabled to prevent data loss.
- **Shared Folders** Instances must have common temp & reports folders, either on a network share or on a cloud drive. Config files must be mirrored or shared.

NOTE. A single Exago server with multiple worker processes could also be considered a "web farm." In this guide we'll consider the general term "web farm" to be synonymous with "server farm," but refer to that specific setup as a "single-server web farm."

A typical setup will look like the following diagram.



Load Balancer

The load balancer server is the point of entry for most web farms. This server receives and directs client traffic to available application instances. Here, you'll set the list of web farm servers, and set any custom options for load balancing.

Please refer to your software's documentation for the specifics on configuration.

It is good practice to have identical Exago installations, i.e. physical path, virtual path, port number, permissions, on each web farm server.

NOTE. If you're using a single-server web farm, then a load balancer is not necessary, as the server's CPU will handle the load balancing.

State Preservation

Exago is a stateful application that preserves user information in a session, a temporary storage space for client-server communication. When a user enters the application, a session is created which persists while the application is in use. A maximum timeout length can be set in the config file to unload sessions after a period of disuse (defaults to 15 minutes).

In a web farm, it is important to manage sessions in order to prevent data loss. This can be done in two ways: **State Server** (*highly recommended*), or **Sticky Sessions**.

State Server

A state server is a server which stores session data. This server should be highly reliable, and we recommended setting up a dedicated server for this task. However, the load balancer server can act as a state server if necessary.

When a user opens a session, the load balancer routes the user to one of the web farm servers. That server then creates and stores the session data on the state server and sets a browser cookie on the user's machine. On subsequent calls, if a user is routed to a different web farm server, that server will look for the cookie, and load the relevant session from the state server.

State server applications are typically services which listen on a port. Each web farm server needs to be configured to point to the state server address.

IIS comes with a state server called ASP.net State Service. Please refer to your software's documentation for the specifics on configuration.

NOTE. If you're using a single-server web farm, then the state service can run on the same server.

Sticky Sessions

Using sticky sessions preserves user state in a different way. When a user opens a session, the load balancer routes the user to one of the web farm servers, which creates a local session. The load balancer then sets a cookie on the user's machine that ensures that all further calls within that session will go to the same server.

To set up sticky sessions, it's usually as simple as enabling the option on the load balancer application settings. Please refer to your software's documentation for the specifics. It may also be called "affinity cookie."

We generally consider sticky sessions to be a less robust solution than using a state server. Performance can be reduced, and there is no recovery mechanism if a server goes offline.

Shared Folders

A shared location for storing Report and Temp data is required for a web farm. This can be within the network, or at an external location, such as a cloud server. In addition, it is recommended to use a shared location for the WebReports.xml config file. Otherwise, any changes to the config would have to be manually pushed to every single instance.

Report & Temp Folders

For each installation, either using the Admin Console or by editing the config file, use the following keys to set a shared storage location:

Network share

(**Main Settings** > Report Path) <reportpath>

```
\\Path\to\reports
```

(**Main Settings** Temp Path) <temppath>

```
\\Path\to\temp
```

Cloud drive

(see the **cloud setup doc** for more information)

(**Main Settings** Report Path) <reportpath>

```
pathtype=azure;credentials='connection_string'
```

(**Main Settings** Temp Path) <temppath>

```
Local\temp\path
```

(**Main Settings** Temp Cloud Service) <tempcloudservice>

```
type=azure;credentials='connection_string'
```

Config File

The config setting `<webfarmssupport>` must be set to True. This is only accessible by editing the config file manually.

The config file xml needs to be mirrored across installations, either manually by copying the file when you change it, or by setting a config cloud path.

Each installation contains a file called appSettings.config in the install directory. To set a cloud path, use the following key in the appSettings.config file for each instance:

Cloud drive

(see the **cloud setup doc** for more information)

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

Additional Notes

Keep in mind that the information in this guide may not apply to every single configuration.

For assistance in getting set up, please consider filing a **support ticket** on our support site. Our staff will be happy to give additional tips or to help walkthrough some of the steps.

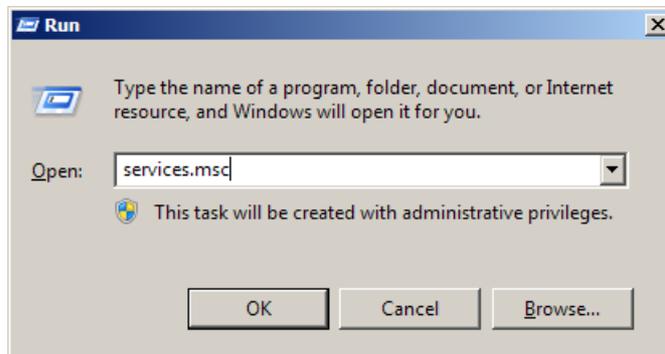
Setting up a State Server

We highly recommend using a state server to manage Exago sessions. Often, the cause of timeout problems is related to not properly managing session state.

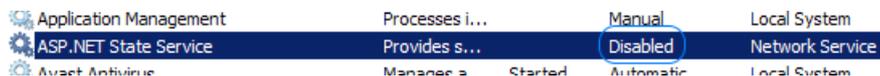
You can create a state service on the same server as the Exago application, or on a different one. This guide will explain how to do so using the ASP.NET State Service which is built into Windows.

Setup ASP.NET State Service

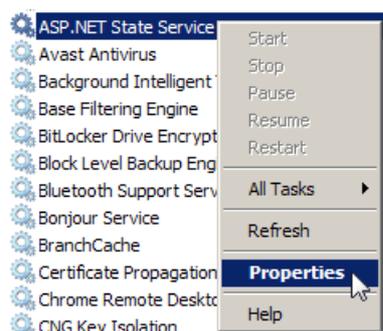
First, enable the State Service. On your state server, press Start > Run, type `services.msc`, and press OK.



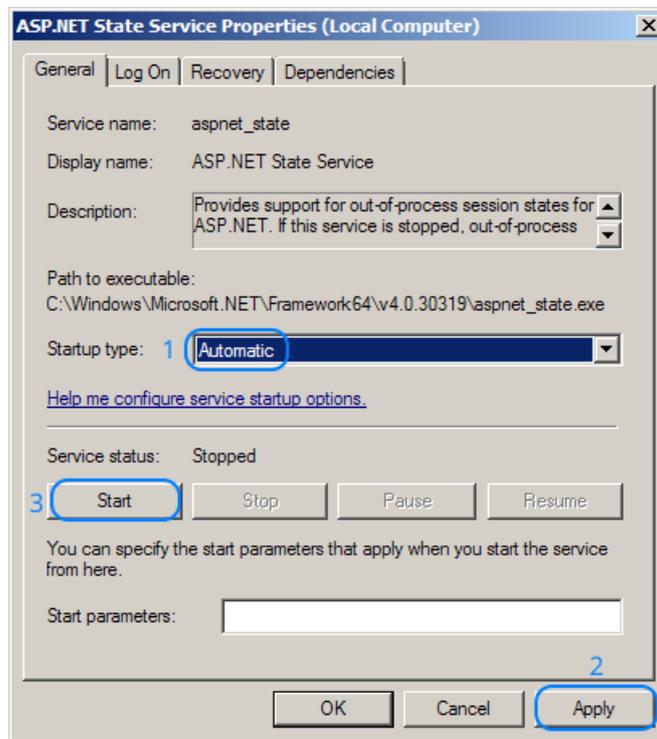
Locate the ASP.NET State Service and check the Startup Type property. If it is not set to *Automatic* or *Automatic (Delayed Start)* then you have to enable the service.



Right-click on it, and select **Properties**.



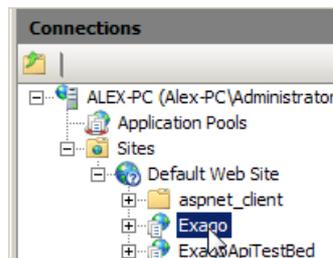
Change the Startup Type to *Automatic*, and press Apply. Then press Start.



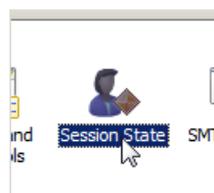
If your state server is on a network, make sure you have allowed inbound connections to the state service on a port.

Configure the Web Server

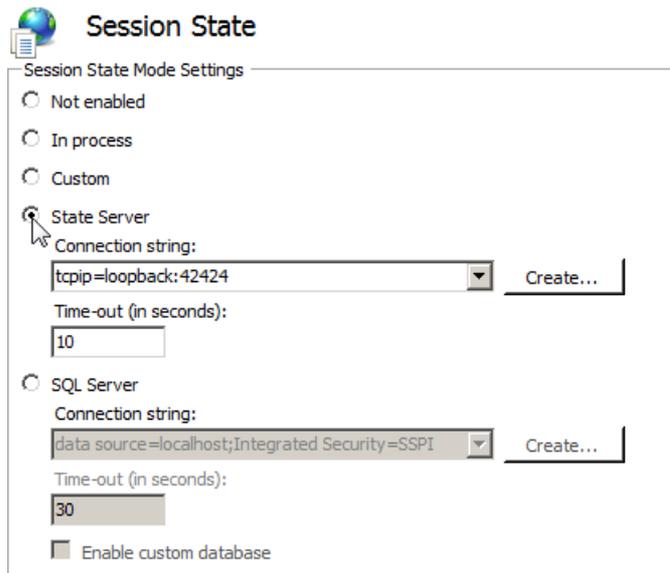
To configure your web server to use the state service, open IIS, then in the left-most **Connections** pane, locate and select your Exago application.



Double-click on **Session State**.



Select the State Server setting, and input the server port and a desired timeout value.



Then in the right-most **Actions** pane, press Apply.



That's it! You should be all set.

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](#)

For personalized assistance, please file a support ticket.

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. 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:** Determine how to expose your data to your users
3. **API:** Use the API to control user permissions
4. **Folders:** Implement a Folder Management solution
5. **Integration:** Visually integrate Exago BI into your host app
6. **Reports:** Make "canned" reports as examples for your users
7. **Deployment:** Important steps to follow before deploying your application
8. **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.

It is recommended to deploy Exago BI on the same server as your application, and to deploy one or more scheduling services on separate servers to handle **Remote Execution** of reports. To use the .NET API, Exago BI 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. 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. Permissions can be set per-user and per-group, for data objects, fields, rows, and even within field values. 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 show more user-friendly names for data objects, instead of 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-normalized should the visible data be?
- Should any data objects 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 IDs to objects to prevent 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 either the **REST** or **SOAP** web service APIs. REST is highly recommended over SOAP, because it supports stronger security and a more modern feature set.

The most basic API implementation begins by initializing a session, then sets the **"userId"** and **"companyId"** parameters to identify the logged-in user, sets 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 reports 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?

Folders

By default, Exago can handle the storage and retrieval of reports on a local or **cloud-based** file system. However, it is strongly recommend to use the **Folder Management** extensibility feature to customize where report definitions reside. Using folder management can allow you to store reports in a database instead of on a file server. This can make it easier to control user access permissions, helps to scale your deployment, and provides additional benefits such as soft deletes and report usage tracking.

Your custom folder management definitions are accessed from a custom assembly, so this does require some additional development. This **sample code** can be downloaded and used as a guide or a starting point.

Also consider the following:

- How do you want the folder tree to appear for new users?
- Will you create common reports that are available for all users?
- Are there multiple levels of report permissions beyond individual users and published reports?

Integration

Exago BI gives you full control over the **CSS, icons, and language strings** in the UI. You can have several different **application themes** if

necessary, and select different ones for different classes of users. You can also build custom 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 page

Remove the content from the default **Getting Started** page, since this is only intended 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 used the Getting Started page to 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" 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 report 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?

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 version and build number.
- If you are using the .NET API make sure the version of `WebReportsApi.dll` matches the version and build number of Exago BI. Do the same with any custom assemblies, such as Folder Management, Scheduler Queue, or Server and Action Events.

Security

IMPORTANT: Follow our **Security Checklist** before turning on access to your application. It is highly recommended to follow these steps to reduce the possibility of unauthorized access.

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 a config password**
- **Remove the plain-text config**
- **Remove the admin console**
- **Encrypt scheduler data (if applicable)**
- **Disable SOAP (if applicable)**

Set an external temp path

The Temp directory contains working data, and may contain sensitive information. If the Temp Path config parameter is left blank, Exago will default to a Temp folder at the root of the install directory. This is not recommended because it could expose your 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).

(**Main Settings** Temp Path) <temppath>

Disable direct access

Access to Exago should be curated through the API so that user permissions can be set via Roles. Users should not be able to access the home page directly, which would bypass role restrictions. To disable direct access to Exago, set the following config setting to *False*:

(**Main Settings** Allow direct access to Exago (bypassing API)) <allowhomedirect>

Set a config password

A User ID, Password, and REST Key should be set in the config file. This safeguards access to the Admin Console and REST API. See **REST API** for information on accessing a password-protected web service.

(**Other Settings** User ID) <userid>

(**Other Settings** Password) <password>

(**Other Settings** Confirm Password)

(**Other Settings** REST Key) <restkey>

Remove the plain-text config

The Admin console generates two copies of the configuration whenever the OK or Apply button is pressed: a plain-text xml document, *WebReports.xml* by default, 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 from your installation, remove the following file from the web application directory (in a web farm, do so for every application instance):

```
Bin\admin.aspx.cdca7d2.complied
```

Note: You can delete the *Admin.aspx* page as well, or edit it to show a static error message.

Encrypt scheduler data (if applicable)

Each scheduler stores working data in a local temporary folder. If you're using scheduler services for report scheduling or remote execution, you should set them to encrypt their data. For each scheduler, edit the *WebReportsScheduler.xml* configuration file and set the following setting to *True*:

```
<encrypt_schedule_files>True</encrypt_schedule_files>
```

Then restart the service.

Disable SOAP (if applicable)

If you are using the REST web service API, then you should disable the SOAP API to prevent any unauthorized web service requests. To do so, remove the following file from the web service directory (in a web farm, do so for every application instance):

```
Bin\api.asmx.cdca7d2.compiled
```

Note: If you are using the .NET API then you do not need the web service, and you can remove it from your environment.

Note: You can delete the *Api.asmx* page as well, or edit it to show a static error message.

About the Admin Console

The Exago Administration Console serves as a user interface to set up and save administrative preferences. Using the Administration Console you can create and modify:

- **Data:** Establish how to connect to databases and determine what data should be exposed to users.
- **General:** Modify global settings of Exago to enable/disable features.
- **Roles:** Create and modify security Roles for individuals or groups of users.
- **Custom Functions:** Create and modify custom functions to make calculations on reports.
- **Server Events:** Create and modify custom code that is run when reports execute.
- **Custom Options:** Create and modify custom options that can be set on reports.

The Administration Console creates two configuration files: an XML file called `WebReports.xml` and an encrypted XML file called `WebReports.xml.enc`. These files are created and saved in the `Config` folder where Exago was installed.

Notice. A backup copy `WebReports.xml.backup` is no longer created.

Important Security Notes:

- Each time you save the Administration Console settings an encrypted copy of `WebReports.xml`, called `WebReports.xml.enc`, is created. This copy cannot be edited with the Admin Console. 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 `WebReports.xml` from the `Config` directory.
- Before deploying Exago into a production environment be sure to set a value for the 'Temp Path' in Main Settings to a location that resides outside of your server's firewall/security system.

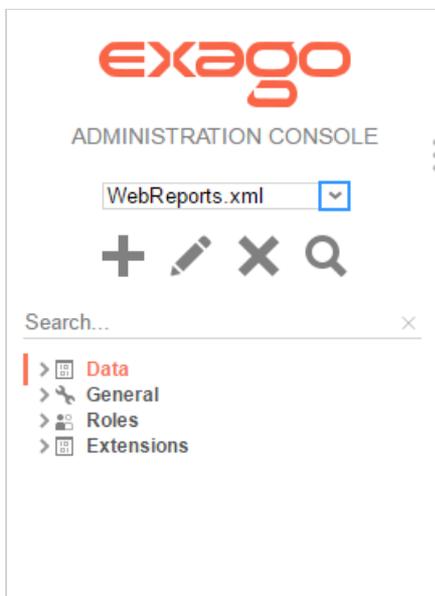
Creating Additional Configuration Files

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 browser.
2. Append `?configFn=NewConfigFile.xml` to the URL replacing 'NewConfigFile' with the name you want to give the configuration file.
3. Click in the URL bar and press enter.



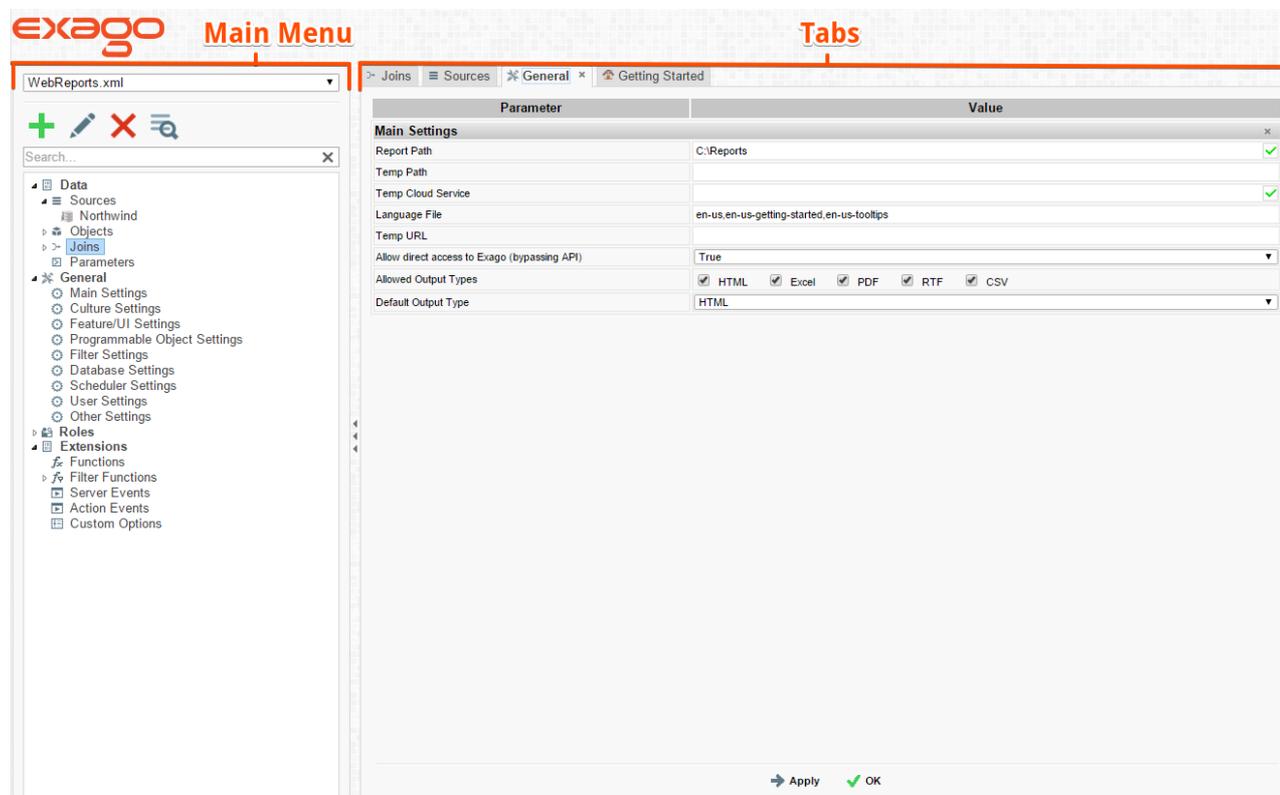
Accessing the Administration Console

Once Exago is installed, navigate the browser to <http://Your Server/Exago/Admin.aspx>.

IMPORTANT. In the **Other Settings** menu under the 'General Section' you can set a login and password to restrict future access to the Admin Console.

Navigation

The Administration 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 Data Sources, Data Objects, Parameters, Roles, and other settings.



Main Menu

Through the main menu you can:

- 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.



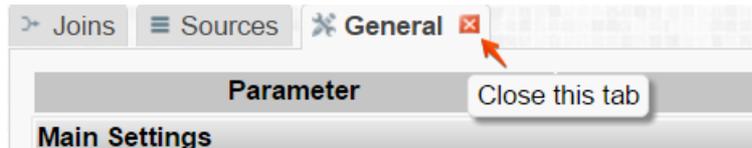
Click the arrows () to hide the main menu.

Tabs

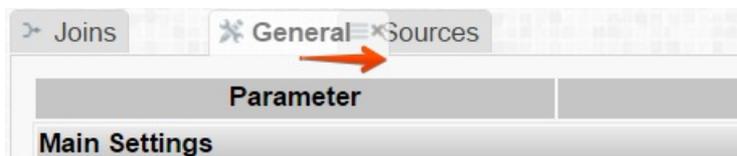
The right section of Exago is made up of tabs containing menus to create and modify administrative settings.

To save the changes made in a tab click 'Ok' or press 'Apply'.

Tabs can be closed without saving by clicking the 'x' to the right of the tab name.



Tabs can also be rearranged by clicking and dragging them as desired.



Main Settings

The main settings are the basic options for Exago. The following settings are available:

Report Path

The parent folder for all reports. The Report Path may be:

- **Virtual Path**
- **Absolute Path** – used to provide increased security (ex. C:\Reports)
- **Web Service URL or .NET Assembly** – using a Web Service or .Net Assembly allows reports and folders to be managed in a database. For more information see **Report Folder Storage & Management**

Temp Path

The location where temp files are stored. The Temp Path may be:

- **Blank** – All temp files and images will be saved to ./Temp.
- **Virtual Path**
- **Absolute Path** – 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 server'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 'Access Denied.'

NOTE. 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.

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. The Default Output Type must be one of the available Allowed Output Types.

Report Tree Shortcut

(v2017.2+) 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.

Culture Settings

The culture settings give administrators control over formats and symbols that vary amongst geographic location (e.g. \$ is the currency symbol in the U.S.A but € is the symbol used in Europe). These settings can be overwritten for a specific user or group of users by modifying the Role. For more information see **Roles**.

The following settings are available:

Date Format

The format of date values. May be any .NET standard (ex. MM/dd/yyyy).

Time Format

The format of time values. May be any .NET standard (ex. h:mm:ss tt).

DateTime Format

The format of date-time values. May be any .NET standard (ex. M/d/yy h:mm tt).

NOTE. For more details on .NET Date, Time and DateTime Format Strings please visit <http://msdn.microsoft.com/en-us/library/8kb3ddd4%28v=vs.71%29.aspx>.

Date Time Values Treated As

Choose to format DateTime as Date or DateTime values. To change this setting for specific columns see **Column Metadata**.

Numeric Separator Symbol

Symbol used to separate 3 digit groups (ex. thousandths) in numeric values. The default is ','.

Numeric Currency Symbol

Symbol prepended to numeric values to represent currency. The default is '\$'.

Numeric Decimal Symbol

Symbol used for numeric decimal values. The default is '.'.

Numeric Decimal Places

Default number of decimal places for numeric fields to show. Leave blank to keep variable by field.

Currency Decimal Places

(v2016.3+) Default number of decimal places for currency fields to show. Leave blank to keep variable by field.

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

(v2016.3+) Set to true to cause currency values to appear right-aligned by default in report cells.

Server Time Zone Offset

Value that is used to convert server to client time (the negation is used to convert client to server time). Leave blank to use server time, or to use **External Interface** to calculate value.

NOTE. This offset is NOT applied to data coming from Data Sources. It is utilized by the Exago UI such as the Scheduling Service.

Feature/UI Settings

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.

Available Report Types

These settings enable/disable report types.

Allow Creation/Editing of Express Reports

Enables/Disables the Express Report Wizard.

Allow Creation/Editing of Advanced Reports

Enables/Disables the Advanced Report Wizard and Report Designer.

Allow Creation/Editing of Crosstabs

Enables/Disables the Crosstab Report Wizard and Insert Crosstab button in the Report Designer.

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 (v2016.3+)

Enables/Disables the ExpressView designer.

ExpressView Settings

(v2016.3+)

These settings only apply to the ExpressView Designer.

Allow Editing ExpressView with Live Data

Allows users to make changes to ExpressViews while in Live Mode.

Note: We recommend setting this to False. Editing live ExpressViews will cause a large increase in database calls, and may reduce performance.

Fields Enabled in Data Fields Tree (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**: Users can only add fields with a direct join to an existing report field.

Join Path Algorithm (v2018.1+)

Select the join path algorithm to use when running ExpressViews with multiple entities .

- **Standard** (*default*): More performant in most cases.
- **Legacy**: Select if you are experiencing issues with ExpressViews created in an older version.

ExpressView Tutorial (v2018.2+)

Enables an instructive tutorial guide using ExpressViews to guide new users through the process of building an ExpressView report. The user can opt out during the course of the tutorial.

ExpressView Hints (v2018.2+)

Enable context-sensitive hints when using different features in ExpressViews.

Express Report Designer Settings

These settings only apply to the Express Report Wizard.

Show Styling Toolbar

Enables/Disables the styling tools in the Layout tab of the Express Report Wizard.

Show Themes

Enables/Disables the Theme dropdown in the Layout tab of the Express Report Wizard.

Show Grouping

Enables/Disables the grouping tools in the Layout tab of the Express Report Wizard.

Show Formula Button

Enables/Disables the formula editor button in the Layout tab of the Express Report Wizard.

Advanced Report Designer Settings

These settings only apply to the Report Designer.

Show Chart Wizard

Enables/Disables the Insert Chart button in the Report Designer. Set to False to disable users from creating or editing charts.

Chart Colors

Lists the values used for default chart colors. Hexadecimal values should be separated by commas (or semicolons).

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

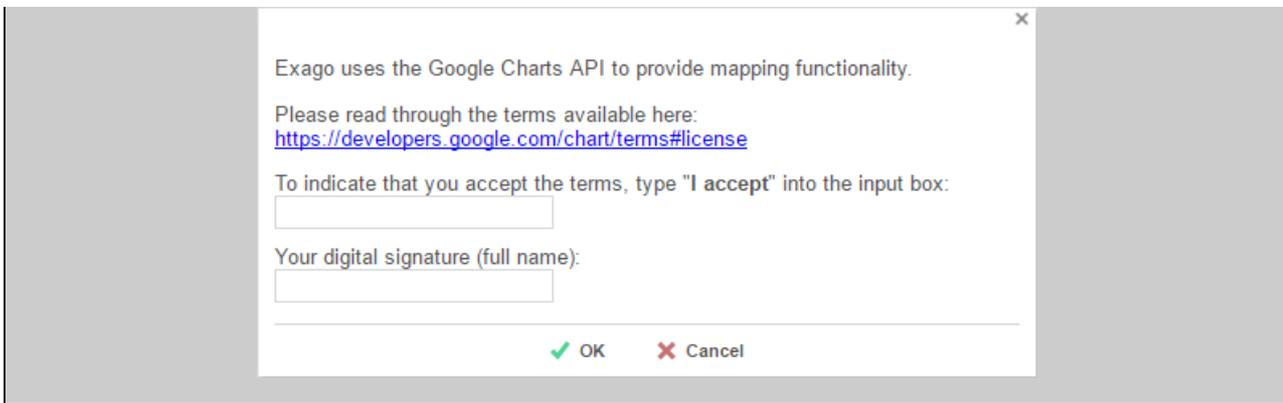
Specifies a default font for charts created in the Report Designer. This setting can be overridden on a per-Report basis. Does not apply to Data Visualizations.

Show Geochart Map Wizard

Enables/Disables the Geochart Maps button in the Report Designer. Set to False to disable users from creating or editing Geochart maps.

Note: Geocharts refers to the legacy maps feature, which was available in v2013.2+.

Note: The first time Show 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 (v2016.3+)

Optional Google Maps license key for geochart 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 (v2016.3+)

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 the Google Maps button in the Report Designer. 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, and a polygon file must be downloaded from our **support site**. See **Google Maps** for more information.

Google Map Key (v2016.3-v2018.1)

License key for Google Maps permissions. This is required to use the new Google Mapping feature. License must contain the **Google Maps JavaScript API** and **Geocoding API** services. See **Google Maps** for more information.

Google Map Key (unlimited or JS API restricted) (v2018.1+)

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 2018.1, the value previously supplied for the Google Map Key will appear here.

Google Map Key (optional Geocode API restricted) (v2018.1+)

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 (v2016.3+)

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 Insert Gauge button in the Report Designer. 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 the Document Template Menu. 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 Templates to the Report Path. Set to False to prevent users from uploading Document Templates.

Show Linked Report

Enables/Disables the Linked Report button in the Report Designer. Set to False to disable users from creating Linked Reports.

Show Linked Report Fields

Enables/Disables the Fields selector tab in the Linked Report dialog.

Show Linked Report Formula

Enables/Disables the Formula editor tab in the Linked Report dialog.

Show Linked Action

Enables/Disables the Linked Action button.

Show Insert Image

Enables/Disables the Insert Image button in the Report Designer. 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. Choose Complex to allow users to modify join operators and expressions, and allow conjoining clauses. Choose Standard to only permit joining data columns on equality.

Allow Category Aliasing (v2017.3.1+)

Select whether to allow data categories to be aliased in the report Categories Menu. Enabling 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 Menu under Advanced. Set to True to enable advanced users to apply Event Handlers for the report. See **Server Events** for more information.

Show Linked Reports in New Tab (pre-v2017.3)

Specify how to display Linked Reports. Set to True to open Linked Reports in a new tab. Set to False to display Linked Reports in a floating window above the parent report.

Linked Report Display (v2017.3+)

Specify where to display drilldowns for linked reports.

- **Cursor:** Window at the cursor
- **New Tab:** New Exago tab
- **Center of screen:** Window centered in the screen

Allow Grouping on Non-Sorts

Enables/Disables the group formula button in the Group Header/Footer Menu. Enabling this will allow users to group 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: See [this article](#) before enabling Report-Level SQL.

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 Show Dashboard Reports 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.

Show URL Item Button

Display/Hide the New URL item in the Toolbox of the Dashboard Designer.

Allow Creation/Editing of Dashboard Visualizations

Display/Hide the New Data Visualization item in the Toolbox and the Data Fields of the Dashboard Designer.

Use Sample Data for Dashboard Visualization Design

Set to True to use sample data while creating and editing Dashboard Visualizations. This will reduce the number of calls to the database. Set to False to query the Data Source for each change made while editing Dashboard Visualizations.

Visualization Database Row Limit (*pre-v2017.2*)

Maximum number of rows returned on a queries made by Data Visualizations. This only applies to Tables, Views and Functions. Set to 0 to return all rows.

Refresh Reports/Visualizations on Dashboards Silently

Set to 'True' to disable the refresh hourglass for timed automatic dashboard reloads.

Common Settings

Default Designer Font

Specifies a default font for reports created in the Advanced Report Wizard, Express Report Wizard, Advanced Report Designer, and Dashboard Designer. This setting can be overridden on a per-Report basis. Does not apply to CrossTabs.

Note: End-users must have the selected font installed locally in order to display. Otherwise, Exago will default to Sans Serif. We suggest using a font-face CSS tag in your **custom home page** to tell the browser to download the font automatically:

```
@font-face {
  font-family: 'Open Sans';
  src: url('myFonts/OpenSans.ttf');
}
```

Default Designer Font Size

Specifies a default font size for reports created in the Advanced Report Wizard, 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 button in the top right corner of Exago. Set to False to disable users from accessing Context Sensitive help.

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 a tab in Exago. 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 reports.

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. This can be modified in the Options Menu of the Report Designer.

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 Main Menu.

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** - 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** - The toolbar will always show.
- **Hide** - The toolbar will never show.

Default interactive report viewer dock is open

Set to False to have the Interactive report Viewer Dock minimized by default.

Interactive report viewer default dock placement

Specify if the Interactive Report Viewer Dock should appear on the right or left of the default output.

Allow save to report design for report viewer

Set to False to prevent users from saving Interactive Report Viewer changes onto the report.

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 enable Exago to use SVG (scalable vector graphics) icons instead of the default PNG icons for the UI elements. SVG icons look nicer on high-pixel density screens, but they may not be compatible with older web browsers.

Application Theme Selection (v2016.3+)

Choose from a selection of downloadable UI themes. See **Application Themes** for more information.

Show Data Fields Search Box (v2017.2+)

Enables/Disables the data field search tools in the sidebar of the ExpressView and Dashboard designers.

Note: We highly recommend setting **Column Metadata**, and setting **Schema Access Type** to Metadata for all available objects, before enabling this feature.

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** - return a DataSet with no rows.
- **1 : Data** - return a full DataSet.
- **2 : Filter Dropdown Values** – 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).

Column Parameter Name

- **Call Type = 1:** List of columns required to execute the report, separated by commas.
- **Call Type = 2:** Column being requested by the filter dropdown.

Filter Parameter Name

- **Call Type = 1:** The filter string specific to the Data Object being called passed as standard SQL.
- **Call Type = 2:** The current value of the filter whose dropdown is being requested.

Full Filter Parameter Name

- **Call Type = 1:** The filter string for the entire report passed as standard SQL.
- **Call Type = 2:** The Tenant and Row Level filters passed as standard SQL.

Sort Parameter Name

The sort string for the report. This is for informational purposes only as Exago sorts data upon return from stored procedures and Web Services.

Data Category Parameter Name

The Data Object's Category. 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**.

DB Row Limit Parameter Name

(v2018.1+) Maximum number of rows to return for this query.

DB Row Start Index ID Parameter Name

(v2018.1+) Index of the first row to return for this query. Used, when **Incremental Loading** is enabled, to return the data set in incremental queries, rather than all at once.

DB Row End Index ID Parameter Name

(v2018.1+) Index of the last row to return for this query. Used, when **Incremental Loading** is enabled, to return the data set in incremental queries, rather than all at once.

Filter Settings

The Filter Settings provide control over what filter options are exposed to users and how the drop downs within filters behave.

Names for the following Parameters can be set:

Show Group (Min/Max) Filters

Enables/Disables 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 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.

Allow Filter Dependencies

Causes filter drop downs to retrieve values dependent on the filters above them in the menu. Set to True to enable.

NOTE: This setting only works for Data Objects from databases and will not change drop downs from Web Services, .NET Assemblies, stored procedures, etc.

NOTE: Drop downs after an 'OR' filter will not be dependent on previous filters.

Show Filter Description

Enables/Disables reports to have a description text for the filters menu. The filter description is set in the Description tab of the New Report Wizard or the Description Menu. A help button will appear in the Filters menu and display the filter description when clicked.

Default Filter Execution Window

Specifies the type of filter execution window to new reports should use by default.

- **Standard** – New reports display the standard filter execution window.
- **Simple with Operator** – New reports display a simplified filter execution window that only allows the operator and value to be changed.
- **Simple without Operator** – New reports display a simplified filter window that only allows the value to be changed.

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 'not equal' or 'not one of'.

Custom Filter Execution Window

Specifies a control or URL that contains Custom Filter Execution Window. See **Custom Filter Execution Window** for more details.

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

The Database Settings allow administrators to adjust how Exago interfaces with databases. Additional type-specific settings allow you to specify which driver to utilize when connecting to each data source.

The following Database Settings are available:

Database Timeout

Maximum number of seconds for a single query to run.

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 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 (v2017.2+)

Maximum number of rows returned on a query. Set to '0' to return all rows. Set to > '0' to enable **Incremental Loading** for **Advanced Reports** and **ExpressViews**. 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 does not have a join path with at least one other Data Object on the report. Set to *False* to disable this behavior.

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 (v2016.3+)

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.

Type-Specific Database Settings

Each Type of Data Sources has the following settings available.

Data Provider

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 machine.config file. See [this link](#) for more information.

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 Source's connection string by surrounding the property name in @ symbols. For example, "@database@" will be replaced with the database name from the connection string of the Data Source being queried.

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, ensure that the scheduler service 'ExagoScheduler' is installed, running, and set to automatically start. For more information see **Installing Exago 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 **Load Balancing**.

The following Scheduler Settings are available:

Enable Report Scheduling

If 'False' will override **Show Report Scheduling Option**, **Show Email Report Options**, & **Show Schedule Manager** to False.

Enables/Disables the scheduler icon on the Main Menu. Set to 'False' to disable users from creating scheduled reports.

Show Report Scheduling Option

Enables/Disables the scheduler icon on the Main Menu. Set to False to disable users from creating scheduled reports.

Show Email Report Options

Enables/Disables the email report icon on the Main Menu. Set to 'False' to disable users from emailing reports.

Show Schedule Reports Manager

Enables/Disables the scheduler manager icon on the Main Menu. Set to 'False' to disable users from editing existing schedules.

Show Schedule No End Date Option

Controls if users must set an end date for recurring report schedules. Set to 'False' to force users to set a limit to the schedule.

Show Schedule Intraday Recurrence Option

Enables/Disables options in the Recurrence tab to have a schedule repeat throughout the day it is scheduled.

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, User Id, 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 and User Id columns 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 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

Set to true to allow users to choose the output option (e.g. email or archiving) with each schedule. When enabled the default value will reflect whatever is set in the 'Email Scheduled Reports' setting.

Use Secure Scheduler Remoting Channel

(v2016.3+) 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 server and port for the 'ExagoScheduler' windows service.

Enable Remote Report Execution

Permits report execution to be done on a different server via the scheduler service. Set to 'True' to enable this behavior.

Enable Execution Cache

(v2017.1+) Permits users to use an execution cache for specified reports. An execution cache refreshes report data on a schedule, and report execution calls use the cached data instead of querying the database.

User Cache Visibility Level

(v2017.3+) 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:** 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

Permits all non-execution data base calls to be done on a different server via the scheduler service. Set to 'True' to enable this behavior. Example calls include Filter value drop downs, Data Object Schema retrieval, and Data Source schemata retrieval in the Administration Console.

Remote Execution Remoting Host

Specifies the server(s) to use for remote execution. The Port is set in the schedule remoting configuration of the scheduler. Separate multiple servers with commas or semicolons.

Ex. `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.

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. Parameters such as `@reportName@` may be utilized in this area.

Default Email Body

Sets a default body that will be displayed in the schedule report wizard. Parameters such as `@reportName@` may be utilized in this area.

Password Requirement (for PDFs only)

Requires a password for PDF export. This parameter can be made up of the following values:

- **A:** requires an upper case letter for each 'A'
- **a:** requires a lower case letter for each 'a'.
- **n:** requires a numeric character for each 'n'
- **4:** password must have at least 4 characters

Ex. 'AAnna6' 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. See **Custom Scheduler Recipient Window** for more information.

Show "Reply To" Field in the Scheduler "Recipients" Tab

(v2018.2+) 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

Administrative options that do not fall into any of the previous categories are found in the "Other" category.

The following Other Settings are available:

Excel Export Target

Choose the type of Excel export you would like. Choosing 2003 will automatically split the workbook into multiple worksheets when Excel'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. For more information see **External Modules**.

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.

NOTE. This will cause multiple hits to 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

(v2017.2+) 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.

(pre-v2017.2) Set to 'True' to write a log file for debugging purposes. For more information see **Reading the Log File**.

Enable Debugging

Set to 'True' to enable debugging. For more information see **Manually Creating a Debug Package**.

Max Report Execution Time

(v2016.2.12+) Specify how long reports should run before timing out. Default is 240 minutes (4 hours).

Maximum Age for Temp Files

Determines the maximum number of minutes a temp file can exist before Exago's automatic cleanup of temp files will remove 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 Report to One Category

Limits reports to Data Objects within a single category. Set to 'True' to enable this behavior.

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 Object'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

(v2016.3+) Allows a **FIPS-complaint** server to encrypt PDF files by using an alternate MD5 library to the built-in System.Cryptography.

'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. 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. Leave blank to permit unverified access to the Administration Console.

Password

Used in conjunction with User ID to gain access to the Administration Console and REST API.

Confirm Password

Used to confirm the value of "Password."

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 more details see [Submitting a Debug Package](#).

Exago Expiration Date

A date when users will no longer be able to access Exago.

Custom Code Supplied by Exago

Used for custom functionality.

Automatic Database Discovery

Automatic Database Discovery enables you to quickly and easily add many Data Objects and Joins from a single Data Source. Discovery can only be performed on the following database types: mssql, oracle, mysql, postgresql, db2, and Informix.

To start using Database Discovery, select a Data Source and click the Discovery button (). This will open a discovery tab for the Data Source.

In the discovery tab you can do the following:

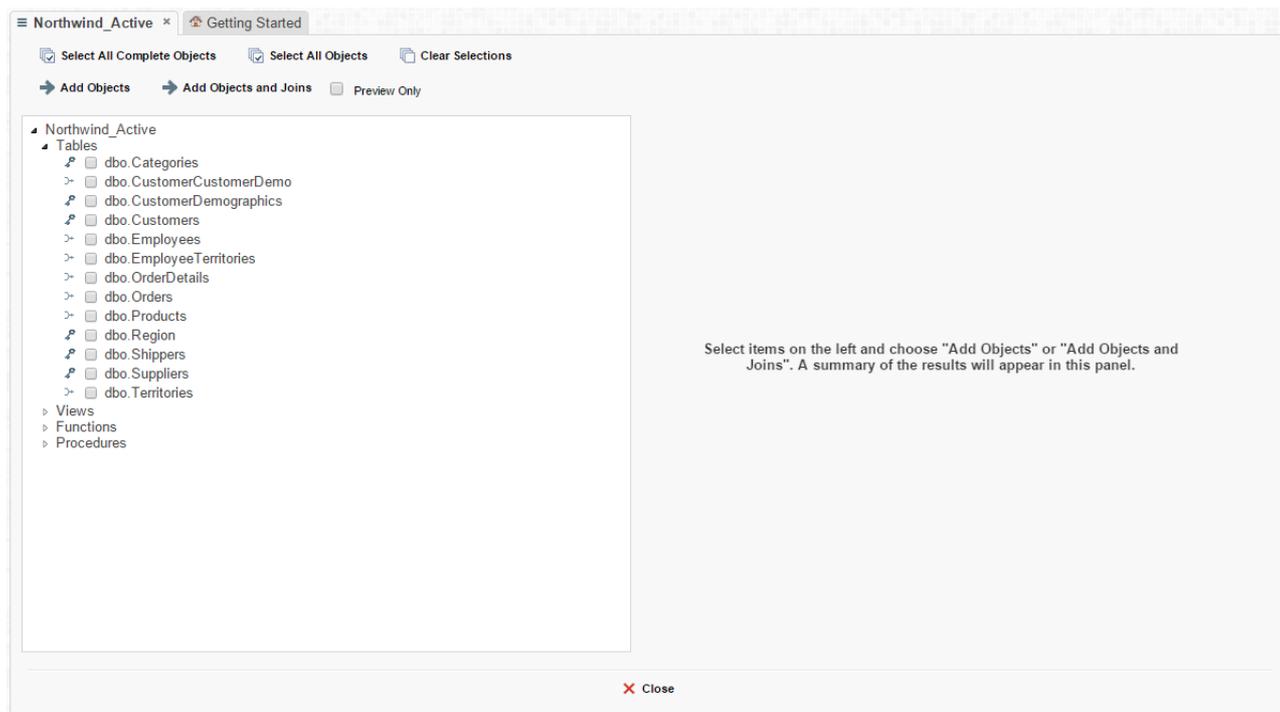
- Select the Tables, Views, Functions, and Stored Procedures you would like to add by either checking individual boxes or pressing 'Select All Objects' or 'Select All Complete Objects'.

Note. Objects with identified unique key values will have a key icon () next to them and objects with associated joins will have a join icon () next to them.

- Set any missing Unique Key fields by right clicking on an object.
- Check 'Preview Only' and then 'Add Objects' to preview the selected objects and joins.
- Add the selected Data Objects by pressing 'Add Objects'.

Note. If any selected Objects are missing unique key values they can be completed individually in a new tab entitled 'Incomplete Objects'.

- Add the selected Objects and any associated Joins by pressing 'Add Objects and Joins'.



Customizing Data Discovery SQL

(v2016.3.6+) The SQL used for Automatic Database Discovery can be customized if necessary, in order to accommodate non-standard key names.

To customize the discovery SQL, locate the file `dbconfigs.json` in the `Config/Other` folder in the host application's install directory, and open it in a text editor.

Locate the property for your data source type and edit the SQL for either or both primary and foreign keys. Save the file, then run Database Discovery to see your changes.

```

6      },
7      "mysql":
8      {
9          "PrimaryKeySql": "SELECT CONSTRAINT_NAME AS indexname, TABLE_SCHEMA as schemaname, TABLE_NAME
as tablename, COLUMN_NAME as columnname FROM information_schema.key_column_usage WHERE
table_schema = schema() AND CONSTRAINT_NAME = 'PRIMARY'",
10         "ForeignKeySql": "SELECT TABLE_SCHEMA As schemaname, TABLE_NAME as tablename, COLUMN_NAME as
columnname, REFERENCED_TABLE_SCHEMA as referencedschemaname, REFERENCED_TABLE_NAME as
referencedtablename, REFERENCED_COLUMN_NAME as referencedcolumnname FROM
information_schema.key_column_usage WHERE table_schema = schema() AND CONSTRAINT_NAME Like
'FK%' "
11     },

```

Note. Data discovery is not currently supported for ODBC data sources.

Data Sources

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 you may need to download and install the appropriate driver. Please see [Data Source Drivers](#) for more information.

All existing data sources are listed in the **Main Menu** under **Data**. All the sources you are adding or editing will be displayed in the **Data > Sources** tab.

- To add a new data source click **Sources** in the **Main Menu** then click the **+ Add** button.
- To edit a data source either double click it or select the data source and click the **Edit** button.
- To delete a data source select it and click the **Delete** button.
- To save changes click the **Okay** or **Apply** button.

Each data source must have the following:

Name

A name for the data source.

Type

The type of source being used. Valid types include:

Type	Description
mssql	Microsoft SQL Server
mysql	MySQL
oracle	Oracle
postgres	PostgreSQL
db2	IBM db2
informix	IBM Informix
websvc	Web Service (For more information see Web Services .)
assembly	.NET Assembly dll (For more information see .NET Assemblies .)
file	XML or Excel file (For more information see Excel and XML Files .)
msolap	OLAP (For more information OLAP and MDX Queries .)
odbc	ODBC Driver (For more information see ODBC drivers .)
mongodb (v2018.2+)	MongoDB database (For more information see CData and MongoDB .)

Schema/Owner Name (blank for default)

Provide a default database schema for the data source.

Note: Only use this if you are using schema to provide Multi-Tenant security. For more details see [Multi-Tenant Environment](#)

Integration.**Connection String**

The method that is used to connect to the data source. Connection strings vary by type:

Type	Connection Strings
mssql, oracle, postgres, mysql and olap	Please refer to connectionstrings.com for database connection strings. Required Parameter: <ul style="list-style-type: none"> <i>url</i> – The url of the web service. Optional Parameters:
websvc	<ul style="list-style-type: none"> <i>authentication</i> – Set to 'basic' to utilize basic authentication through IIS. This will send the user ID and password as clear text (unless https is used). <i>uid</i> – User ID is passed to the web service. <i>pwd</i> – Password is passed to the web service. Required Parameters:
assembly	<ul style="list-style-type: none"> <i>assembly</i> – The full path of the assembly name. <i>class</i> – The class name in the assembly where the static methods will be obtained. Requires the physical path to the excel or xml file and the file type. Ex. File=C:\example.xls;Type=excel;
file	Requires the hostname, port, and database name. Ex. Host=localhost;Port=27017;Database=test;
mongodb	

Click the **connection verification** ⚡ icon to verify the connection succeeds.

Data Source Drivers

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**
 - ODAC1120320_x64 or newer
 - [Oracle ODAC Connector](#)
- **MySQL/MariaDB**
 - dcmysqfree.exe
 - [Devart Connector](#)
- **PostgreSQL**
 - dcpostgresfree.exe
 - [Devart Connector](#)
- **DB2/Informix**
 - 5.exe or newer
 - [IBM Data Server Driver Package](#)
- **MongoDB**
 - CData ADO.NET driver for MongoDB
 - This comes standard with installations of v2018.2+.

Web Services and .NET Assemblies

Web Services and .NET Assemblies can be used as data sources. This is possible when the Web Service and .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 Web Services and .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 the  **General** section.

- **0 : Schema** - returns a DataSet with no rows.
- **1 : Data** - returns a full DataSet.
- **2 : Filter Dropdown Values** – 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 (optional)

To optimize performance Exago can pass user-specified sorts and filters to the Web Service or .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 **Programmable Object Settings** of the  **General** section.

Custom Parameter Values (optional)

Additional parameters can be specified to be sent to individual methods in the **Data Object Menu**.

Important: When a Web Service or .NET Assembly is first accessed it is compiled and kept in an internal cache within Exago. This is done in order to increase performance. Due to this internal cache, Exago will not be aware of any changes within the Web Service or .NET Assembly. If the service or assembly is subsequently changed, Exago will execute the prior compiled version. Thus, when you modify the Web Service or .NET Assembly reset the internal cache of Exago by clicking the **connection verification** icon of the Data Source or by restarting IIS.

Note: If an **Exago .NET API** application needs to access reports, which use 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=\\MyServerName\MyShareName\MyAssembly.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
            case 1:
                // return data
            case 2:
                // return filter values for dropdown
        }
    }
}
```

Method signature using SessionInfo (v2016.2+):

```

public class Main
{
    public static DataSet dotnet_optionses(WebReports.Api.Common.SessionInfo sessionInfo, int callType, string colu
    {
        switch (callType)
        {
            ...
        }
    }
}

```

Web Services

Web Services are accessed via SOAP. Below is an example of a Web Service connection string:

url=http://MyServer/MyWebService.asmx

Web services methods are similar to .NET Assembly methods with the following exceptions:

- Methods do not need to be static
- Methods must return a serialized XML string. The returned XML must follow the structure used by the C# method DataSet.GetXML. An example of XML format can be found in the following section.

Excel and XML Files

Exago can use Microsoft Excel and XML files as data sources. Remember though that Excel and XML files are not databases. Simply put, these data sources do not offer the speed, performance, security or heavy lifting of a real database. Using Excel and XML files is recommended only if your dataset is small or if the information is only available in this format.

Connection string

File=C:\example.xls;Type=excel;

Excel

Each worksheet in the Excel file will be read as a separate table. Each worksheet's name will be read as the table's title. The top row will be read as the column header, 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 Camo	Poughkeepsie	NY
4	Tucker	Frederick	9876	263846521	38	m	4557 Stev	Kingston	NY
5									
6									
7									
8									

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="
    <xs:element name="ExagoData" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="Call">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="CallID" type="xs:unsignedInt" minOccurs="0" />
                <xs:element name="StaffID" type="xs:string" minOccurs="0" />
                <xs:element name="VehicleUsed" type="xs:unsignedInt" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
          <xs:element name="Staff">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="StaffID" type="xs:unsignedInt" minOccurs="0" />
                <xs:element name="Rank" type="xs:string" minOccurs="0" />
                <xs:element name="LastName" type="xs:string" minOccurs="0" />
                <xs:element name="FirstName" type="xs:string" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:schema>
    <Call>
      <CallID>890</CallID>
      <StaffID>134</StaffID>
      <VehicleUsed>12</VehicleUsed>
    </Call>
    <Call>
      <CallID>965</CallID>
      <StaffID>228</StaffID>
      <VehicleUsed>4</VehicleUsed>
    </Call>
    <Call>
      <CallID>740</CallID>
      <StaffID>1849</StaffID>
      <VehicleUsed>2</VehicleUsed>
    </Call>
    <Staff>
      <StaffID>134</StaffID>
      <Rank>Captain</Rank>
      <LastName>Renolyds</LastName>
      <FirstName>Malcom</FirstName>
    </Staff>
    <Staff>
      <StaffID>228</StaffID>
      <Rank>Lieutenant</Rank>
      <LastName>Brown</LastName>
      <FirstName>Bill</FirstName>
    </Staff>
    <Staff>
      <StaffID>1849</StaffID>
      <Rank>Sergeant</Rank>
      <LastName>John</LastName>
      <FirstName>Pepper</FirstName>
    </Staff>
  </ExagoData>

```

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.

- OLAP Data Objects will always be MDX Queries written in the **Custom SQL Object**
- Data Objects must have **Schema Access Type** set to Metadata and must have **Column Metadata** set for all fields.

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.

Caution: 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 our article on **CData**.

Data Objects

Data objects are the tables, views, methods, stored procedures, functions and custom SQL that you want to make accessible for reports.

All existing data objects are listed in the **Main Menu** under **Data**. All data objects that are added or edited will be displayed in the **Objects** tab.

- To add a new data object click **Objects** in the **Main Menu** then click the **+ Add** button.

Note: Data objects can be added quickly using **Automatic Database Discovery**.

- To edit a data object either double click it or select it and click the **Edit** button.
- To delete a data object select it and click the **Delete** button.
- To save changes click the **Okay** or **Apply** button.

Each data object has the following properties:

Name

Select the data object's source from the first drop-down. In the second drop-down select a data object.

Note: This will display all the of the source's tables, views, methods, stored procedures, and functions.

- To add custom SQL click the **Add Custom SQL** button () next to the **Data Sources** drop-down. For more details see **Custom SQL Objects**.

Note: The name of tables or views may not contain the following characters:

```
{ } [ ] , . %
```

Alias

The user friendly name for the data object. The alias will be displayed to end-users.

Note: An alias may not contain the following characters: @ { } [] , . %

Unique Key Fields

The columns which uniquely identify a row.

Category

The 'folder' used to group related data objects. Sub-categories can be created by entering the category name followed by a backslash then the sub-category name.

Ex. Sales\Clients

ID

A unique value 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**.

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 **Parameters, 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.

Tenants Columns

Specify which columns contain tenant information and link the parameters accordingly.

- This setting is used to filter data when multiple users' 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.

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** – Follow the global *Schema Access Type* setting in **Other Settings**.
- Datasource** – Queries the data source for the schema.
- Metadata** – Reads the schema from the stored metadata.

Note: For more information see **Retrieving Data Object Schemas**.

Filter Dropdown Object

Specify an alternative data object to be queried when a user clicks the value drop-down in the **Filters** menu. This setting is most likely to be used when the data object is a Stored Procedure, Web Service, or .Net Assembly that takes more than a few seconds to return data. In this scenario a table or view can be designated to increase performance.

Note: The **Filter Dropdown Object** must have a column with the same name as each column in the main data objects.

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, and 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 PDO can be joined with suppressed filtering & sorting if this setting is enabled for all of them.

Stored Procedures

Stored Procedures offer the ability to use high level code to modify the data set before it is sent to Exago.

Stored procedures must 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.
- Additional parameters may be passed by setting them in the  **Data** >  **Objects** tab.

Important Note for 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 *Sort Parameters* to maintain efficiency.

Note: For SQL Servers, FMTONLY is set to OFF.

```
ALTER PROCEDURE [dbo].[sp_webrpt_person]
@callType INT, --optional but should be implemented for efficiency
and dropdown support
@columnStr varchar(1000), --optional; used for limiting data for efficiency
@filterStr varchar (1000), --optional; used for limiting data for efficiency
@fullFilterStr varchar (1000), --optional; used for limiting data for efficiency
@sortStr varchar(1000) --optional; may improve performance a bit if used
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)
if @callType = 0 --return schema; don't need to return any rows
begin
    set @sql = 'select * from vw_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 vw_webrpt_person where ' + @filterStr + @orderByClause
end
else
if @callType = 2 --return filter dropdown values; limit # rows to some value
begin
    set @columnInfo = '[' + @columnStr + ']'
    set @sql = 'select top 100 ' + @columnInfo + ' from vw_webrpt_person where ' + @columnInfo + ' >= ' + @filterStr
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 **fx 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 data objects. Parameters can be embedded in these SQL statements to enable you to change the statement at runtime.

To add or edit a **Custom SQL Object** click the **Custom SQL** button () and a dialog box will appear.

Data Object Name

The name of the data object to be displayed in the **Administration Console**.

Data Source

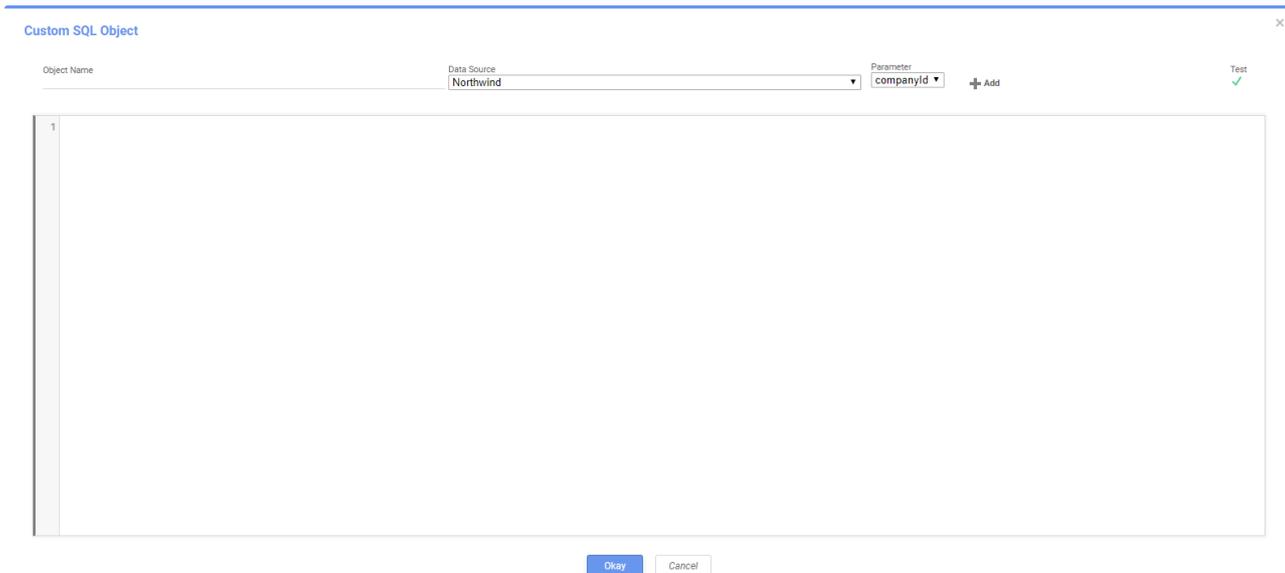
The data source that will be sent the SQL.

Parameter/Insert

Select the parameter you want to embed in the statements. Use the **+ Add** button to move the selected parameter into the SQL statement where your cursor is located. Parameters may also be added manually between **@** symbols (ex. **@userId@**).

Use the **Test**  button to verify that the SQL statement is correct.

Press **Okay** to save the SQL statement or **Cancel** to close the dialog without saving.



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 ww_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 ww_webrpt_optionee IfExistReportDataObject("fn_webrpt_grant", "JOIN ON fn_webrpt_grant...", "")`

Column Metadata

Column metadata refers to the properties of each column in the data objects. Normally Exago gets the metadata for each column directly from the data source, however, in some cases it may be helpful to override or add additional information to the metadata.

To modify the metadata of a column, select it and click the **+ Add** button or double click it. Enter a **Column Alias** or use the **Data Type**, **Filterable**, and **Visible** drop-downs to set the desired properties.

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 Row** button.

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:

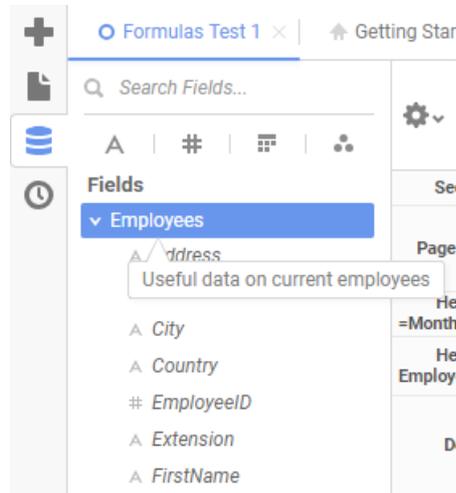
Column Alias

The name of the data field that the end-users see.

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:



Admins can add description text to data fields on an application-wide level. To do so, using the **Admin Console**, expand the **Objects** tab, and double-click on the desired data object, or select it and press **Edit**. Then, in the object menu click on the **Column Metadata** field. This will open the **Column Metadata** dialog.

Double-click on the desired data field, or click-and-drag it to the Selected Columns pane, or select it and press the **Add** button. You have two options for adding description text: Using **Plain Text** or editing the **Language File**.

Plain Text

Data Type	Column Alias	Column Description
String		Unique Employee Number
Filterable	Sort and Group-By Value	
All		
Sortable		
True		
Visible		
True		

Hover-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

Data Type	Column Alias	Column Description
String		eNum
Filterable	Sort and Group-By Value	
All		
Sortable		
True		
Visible		
True		

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 appears as "**Description** Text" in the hover 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, and Guid

Filterable

Whether this field can be used to filter reports. The available options are:

Field	Used as a Report Filter	Used as an Interactive Filter
All (<i>True</i>)	Yes, for the: <ul style="list-style-type: none"> • Advance Report designer • Express Report designer • ExpressView designer • Dashboard designer • Scheduler 	Yes, for the: <ul style="list-style-type: none"> • Report Viewer
Dynamic (<i>False</i>)	No	Yes, for the: <ul style="list-style-type: none"> • Report Viewer
Static (<i>v2017.1.2+</i>)	Yes, for the: <ul style="list-style-type: none"> • Advance Report designer • Express Report designer • ExpressView designer • Dashboard designer • Scheduler 	No
None (<i>v2017.1.2+</i>)	No	No

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.

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.

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 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 database. This is completely transparent for the users; they can then use them like any other column. New data fields can be created from composite or interpreted data fields. You could even use a formula to create data from scratch. Admins often use custom columns to make popular formula sorts available on an application-wide level.

Admin Console

To add a custom column using the **Admin Console**, expand the **Objects** tab and double-click on the desired data object, or select it and press **Edit**. Then, in the **Object** menu click on the **Column Metadata** button next to the **Column Metadata** field. This will open the **Column Metadata** dialog box.

Press the **Add New** button. Enter a name for your data field in the dialog box.

Data Type, **Column Alias**, and **Column Value** are required fields. In the **Column Value** field, press the formula button to bring up the **Formula Editor**.

Press **Okay** when done, then **Apply** the change.

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 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} & ' ' & {Employees_0.LastName}</col_value>
    <col_sortandgroupbyvalue>{Employees_0.LastName}</col_sortandgroupbyvalue>
  </column_metadata>
</entity>
```

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.

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:

- `Random(0,65536)`
Uses a custom function

And much more!

Note: Custom columns cannot be used as filters, or inside **Aggregate Formulas**.

Retrieving Data Object Schemas

Many of the dialogs throughout Exago require schema information (ex. column name, data type, etc.). By default these dialogs query the data sources for the schema. This process, however, may cause performance issues if the data sources take a considerable amount of time to return the schema.

To enhance performance, schema information can be stored as column metadata. Exago can then read the column 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.

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.

Data Object IDs

There are three ways in which you can utilize data object IDs.

Adding Multiple Data Objects with the Same Name

IDs are used 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 will avoid issues if the name of the underlying tables, views, or stored procedures, 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:

- Provide a name for **Data Object ID Parameter Name** in **Programmable Object Settings**
- Create a method/procedure in your Service/Assembly/Procedure that utilizes the object ID parameter to return the appropriate data/schema.
- For each data object:
 - Select  **Object** in the **Main Menu** and click the **+ Add** button
 - Select the single Service/Assembly/Procedure
 - Provide an **Alias** and an **ID** for the object
 - Select the key columns
 - Click **Okay** or **Apply** to save the object.

Example: This stored procedure uses the object ID parameter (@objectID) 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
end

```

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. In the **Administration Console** edit the data object that contains the images. Open the **Column Metadata** menu and for the image column set **Data Type** to *Image*. Next, simply 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. 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.

Joins

Caution

The Joins window is recommended for advanced users only.

Joins describe how the categories on a report are related to each other. When two categories are joined, a field in the first category is associated with a field in the second category. Wherever a value in the first category's field matches a value in the second category's field, that value's rows from each category come together to form a composite row. The table produced by all the composite rows is the resulting data that appears on the report.

For example, take the following categories, Orders and Products. The Orders.ProductId field corresponds with the Products.Id field. When the categories are joined from Orders.ProductId to Products.Id, the rows are connected wherever those two fields have matching values.

Orders		Products	
Orders.Id	Orders.ProductId	Products.Id	Products.ProductName
16702	13	12	Blanton's Original
16703	13	13	Henry McKenna
16704	14	14	Russell's Reserve
16705	16	15	Hillrock Estate
16706	15	16	Buffalo Trace

Categories joined on Orders.ProductId >> Products.Id

The result of this join is the following composite rows. These categories have a *one-to-one relationship*, because each row in the "left" category joins at most one row in the "right" category.

Orders.Id	Orders.ProductId	Products.ProductName
16702	13	Henry McKenna
16703	13	Henry McKenna
16704	14	Russell's Reserve
16705	16	Buffalo Trace
16706	15	Hillrock Estate

Joined categories. Products.Id is omitted.

Tip

Categories could be joined along more than one set of fields; composite rows are formed only when all sets have matching values.

For two categories to be copresent on a report, there must be a join path between them. They are either directly joined, or there is a path through one or more intermediate categories. You do not have to configure joins manually - they already exist in the environment. However, if you want to learn how to add or adjust joins on a per-report basis, this topic will explain the options that are available.

Join Types

The join that was previously described is the most common type of join, an *inner join*. When an inner join is applied, rows in either category that have no matching row in the other are excluded from the resulting table. However, you may not want to exclude these rows. To do so, you can change the type of join to an *outer join*.

For example, the row in the Products category with Id: 12 has no matching row in the Orders category. With an inner join, this row is excluded from the output. If you want to see the Products rows that have no matching Orders row, you can change the join type.

To do so, from the **Report Options** > **Advanced** > **Joins** window, select the **Products data that does not have Orders data** check box.

In addition to Orders data that has matching Products data, include:

Orders data that does not have Products data

Products data that does not have Orders data

Left outer join

This changes the join between these categories to a *left outer join*, because all rows from the *left* category are included. The following rows result:

Orders.Id	Orders.ProductId	Products.ProductName
		Blanton's Original
16702	13	Henry McKenna
16703	13	Henry McKenna
16704	14	Russell's Reserve
16705	16	Buffalo Trace
16706	15	Hillrock Estate

Joined categories with all Products rows. Products.Id is omitted.

Similarly, selecting the **Orders data that does not have Products data** check box changes the join to a *right outer join*, which includes all rows from the *right* category. Selecting both check boxes includes all rows from both categories; this is a *full outer join*.

Relationship Types

There are two types of join relationships: *one-to-one* and *one-to-many*.

In the previous example, the relationship between the categories is *one-to-one*, because each row in the left category joins at most one row in the right category. Some categories have a *one-to-many* relationship, where each row in the left category joins zero or more rows in the right category.

A one-to-many relationship from categories X to Y is represented in the following diagram:

One 1-to-Many	
x	y
	y
	y
x	y
	y
	y
	y
x	y
	y

Each X is joined to one or more Y

Reports with a single one-to-many join are well suited to grouping by the left category. The data in these reports is generally well-formed and understandable.

However, when a report has multiple categories with one-to-many joins, data can appear more disorganized and confusing. For example, the following diagram represents data from three categories, X, Y, and Z, where the relationships between X - Y and X - Z are both one-to-many:

Two 1-to-Many		
x	y	z
	y	z
		z
		z
	y	
x	y	z
		z
		z
	y	
x	y	z
	y	
		z
	y	

Each X has 1 or more Y, and 1 or more Z

Because Y and Z are not directly related to each other, there are many rows with only Y or only Z. This can cause the report to be significantly larger, and to be difficult to read and interpret. This occurs even with inner joins, the most restrictive type, because by default there is no logic that deals with the relationship between Y and Z.

Read on for different ways of improving the structure of a report with multiple one-to-many joins.

Cartesian Processing

You could fill the blank spaces with supplementary data by disabling **Special Cartesian Processing** from the **Joins** window. Blank cells are filled in with data that is repeated directly from the previous row. The following diagram demonstrates how this works:

Two 1-to-Many, No Cartesian		
x	y	z
	y	z
	y	z
	y	z
	y	z
x	y	z
	y	z
	y	z
	y	z
x	y	z
	y	z
	y	z
	y	z
	y	z

Disabling Special Cartesian Processing

The shaded cells represent data that has been repeated from the previous row. This can make the report more readable. However, this approach poses a problem: Blank cells indicate a lack of a relationship between two fields, so filling in these spaces with artificial data can obfuscate any relationship between Y and Z. This can decrease the accuracy of the report.

There are better ways to improve the readability of such a report without sacrificing accuracy:

- Use repeating groups to show the X- Y and X- Z relationships in entirely separate sections. This is suitable if any relationship between Y and Z is irrelevant or nonexistent. See **Sections** for more information.
- Hide some or all of the rows which do not have data for both Y and Z. This is suitable if you want to highlight an implicit or indirect relationship between Y and Z. This is done by imposing *Must* constraints. Read on for more information.

Must Constraints

Although Y and Z are not directly joined, they are both related to X, so there is an implicit relationship between them. If you examine Y and Z alone, you will notice that they technically exhibit a *full outer join*.

y	z
y	z
	z
	z
y	
	z
y	z
	z
	z
y	
y	
y	z
y	
	z
y	

Y and Z, without X

Because all the rows from Y and Z that relate to X are shown, there are rows with both Y and Z, or with only one of either. *Must* constraints allow you to change the implicit join type, and in doing so, eliminate rows that lack data from one or both categories.

To set *Must* constraints, from the **Joins** window, locate the **MUST** panel for the applicable categories:

In addition to selections above, for X data that has ANY of the following, it MUST have a:

Y Z

Setting Must constraints

Do one of the following:

- Select the **Y** check box - *Left outer join*: Any rows without Z are removed
- Select the **Z** check box - *Right outer join*: Any rows without Y are removed
- Select the **Y** and **Z** check boxes - *Full outer join*: Any rows without Y or Z are removed

Two 1-to-Many / MUST Y			Two 1-to-Many / MUST Z			Two 1-to-Many / MUST Both		
x	y	z	x	y	z	x	y	z
	y	z		y	z		y	z
		z		y			y	
		z	x	y	z			
x		z		y		x	y	z
	y	z	x	y				
		z		y	z		y	z
		z		y		x	y	z
x	y	z		y				
		z		y		x	y	z

Effect of setting various Must constraints

Modifying Joins

The **Joins** window shows all direct and implicit joins on the report. Direct joins can be added, modified, or removed from the report.

To add a new join:

1. Select **From** and **To** categories.
2. Click **+ Add**.
3. Click **+ Add Condition** then select **From** (left column) and **To** (right column) fields.

Tip

If there are multiple conditions, only the rows that satisfy all the conditions are joined.

4. Click **OK**.

To remove conditions, click the Delete **X** icon next to the condition to delete.

To modify a join's fields:

1. Click the Edit **E** icon next to the join to edit.
2. Add, remove, or modify conditions.
3. Click **OK**.

To remove a join, click the Delete **X** icon next to the join to delete, then click **OK**.

To restore the default joins, click **Recreate**, then click **OK**.

Advanced Joins

You may be able to specify join conditions that are more complex than column equality.

Note: Advanced Joins cannot be applied across different data sources.

Type

Instead of joining between two columns, one or both sides of the join may instead be an arbitrary expression, constant, or SQL sub-query that you specify.

To change the expression type for one side of a join condition, select one of the following from the **Type** list:

- **Value:** One or more constant values separated by commas
- **Expression:** Formula or calculation
- **SubQuery:** SQL query

Then enter the value in the **Value** field.

Operator

As opposed to the default equality (=) operator, which joins fields from the left expression to matching fields from the right expression, a join condition can use one of several alternative operators instead. For example, the inequality (!=) operator joins fields on the left to non-matching fields on the right. To do so, select one of the alternative operators from the **Operator** list:

- **!=** not equal
- **>** greater than
- **>=** greater than or equal

Parameters

- < less than
- <= less than or equal
- **IN** match one or more values, either specified or calculated from a subquery

Grouping

When a join has two or more conditions, you can specify how the conditions should be met as a group in order for the join to take effect.

To specify that either one of two conditions will satisfy part of the clause, select **OR** from the **Conjunction** list for the first condition of the two.

To add parentheses around two conditions, select the **Group** check box for the first condition of the two.

You can preview the full join clause in the **Summary** field.

Parameters

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**.

In Exago parameters can be used to:

- Pass values to Web Services, .NET Assemblies, or custom SQL Data Objects.
- 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 '@ParameterName@'.

NOTE. Parameters ARE case sensitive.

- 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.

All existing Parameters are listed in the **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 then click the 'Add' button.
- To edit a parameter either double click it or select it and click the 'Edit' button.
- To delete a parameter select it and click the 'Delete' button.
- To save changes click the 'OK' button or press the 'Apply' button.

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:

@ (at sign), { } (curly braces), [] (square brackets), ',' (comma), '.' (period/full stop)

Note: 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 is anything).`

Type

The parameter data type. Select **date** for DateTimes. Select **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.

Note: Functions and date functions are also usable as parameter values. The Type must be **string**.

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.

Display Type

The display value data type.

Parameter Support for Dashboard URL Tiles

A new feature of Exago BI v2017.3 is the ability to use **parameters** in dashboard URL tiles. Dashboard reports intended for use by multiple classes of users can implement URL tiles that change based on user variables.



URL tile with parameter values

Parameters are key-value pairs that are reachable from the Exago API, application extensibility, and, optionally, the user interface. They are intended to be instantiated via the API at session initialization, the value of which are set depending on the user accessing the session. They can either be created in the API or created in the config file or Admin Console and modified in the API.

Note. Parameters must have the **Hidden** property set to False in order to be usable in the application user interface. **Date** type parameters cannot be used in URL tiles.

New Parameter ✕

Name

Type

Value

Hidden

Prompt Text

Parameter Dropdown Object SQL ✕

Setting a parameter's **Hidden** property to False in the Admin Console

If the Prompt Text property is set on a URL tile parameter, users will be prompted to set its value when they run the dashboard.

Note. Dashboard URL tiles cannot be modified after the dashboard is run. Prompting URL parameters cannot be made accessible in the Interactive Parameters pane.

URL formatting

Parameters may be utilized anywhere within a URL string.

Example

```

http://@address@.com
http://example.@domain@
http://@subdomain@.example.com
http://example.com/@path@
@protocol@://example.com
http://example.com:@port@
http://example.com?@key@=value
http://example.com?key=@value@
http://example.com#@fragment@

```

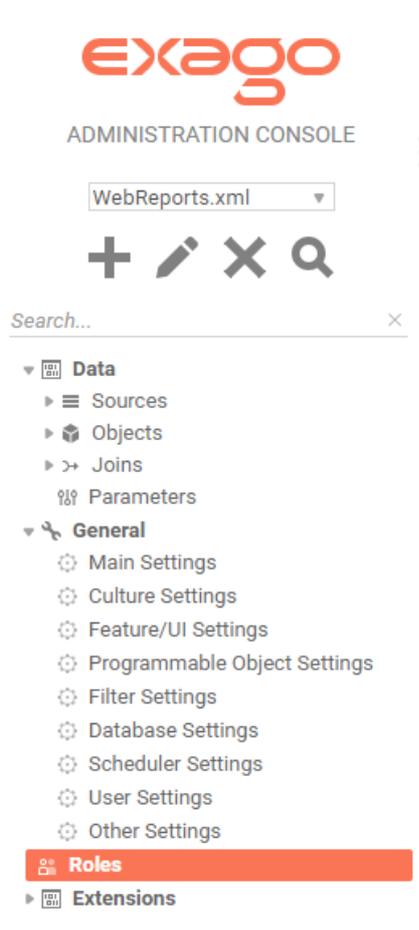
It is recommended that spaces and other special characters are URL-encoded where appropriate. For example, to pass "Hello World" as part of a URL query string, encode the space between the two words as "%20". This will **not** be done automatically.

```
http://example.com?key=Hello%20World
```

For more information, see [Percent-encoding](#) (Wikipedia)

Roles

This chapter explains how to use the Roles to control access to Data and override the General Settings.



- To add a new role select 'Roles' in the Main Menu then click the 'Add' button.
- To edit a role either double click it or select it and click the 'Edit' button.
- To delete a role select it and click the 'Delete' button.

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 application's own security and authentication methods. Although you can create Roles through the Administration Console, Roles are typically created through the API to dynamically set a user's access. For more information see the articles regarding **Integration** and **API**.

Roles have five sections to control access: Main, General, Folders, Objects, and Filters.

Main

Controls the broad properties of the Role.

General

Overrides General Settings.

Folders

Controls which report folders a role can see and edit.

Object

Controls which Data Objects a role can access.

Filter

Provides row level filters on Data Objects.

Main Settings

The main settings control the broad properties of the Role.

Main					
ID	Active	Include All Folders	All Folders Read Only	Allow Folder Management	Include All Data Objects
_____	<input type="checkbox"/>				

The Main role settings are:

Id

A name for the role.

Active

Check to activate 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

Displays/Hides the Folder Management Icon and functionality.

Include All Data Objects

If checked, all Data Objects that are not listed in Objects Access will be available. If unchecked, only those listed in Objects Access will be available.

General Settings

The General Settings of a Role override the Global General Settings. Utilize the API in order to overwrite additional settings for a user or group of users. For more information see [API](#).

General	
Report Path	<input type="text"/>
Date Format	<input type="text"/>
Time Format	<input type="text"/>
DateTime Format	<input type="text"/>
Numeric Separator Symbol	<input type="text"/>
Numeric Currency Symbol	<input type="text"/>
Numeric Decimal Symbol	<input type="text"/>
Server Time Zone Offset	<input type="text" value="0"/>
Show Grid Lines in Report Viewer	<input type="text"/>
Allow Creation/Editing of Express Reports	<input type="text"/>
Allow Creation/Editing of Advanced Reports	<input type="text"/>
Allow Creation of Crosstab Reports	<input type="text"/>
Show Crosstab Wizard	<input type="text"/>
Allow Creation/Editing of Dashboards	<input type="text"/>
Allow Creation/Editing of Chained Reports	<input type="text"/>
Allow Creation/Editing of ExpressViews	<input type="text"/>
Allow Editing ExpressView with Live Data	<input type="text"/>
Show Styling Toolbar	<input type="text"/>
Show Themes	<input type="text"/>
Show Grouping	<input type="text"/>
Show Formula Button	<input type="text"/>
Database Timeout	<input type="text" value="0"/>
Read Database for Filter Values	<input type="text"/>
Show Report Scheduling Option	<input type="text"/>
Show Email Report Options	<input type="text"/>
Show Schedule Reports Manager	<input type="text"/>
Scheduler Manager User View Level	<input type="text"/>
Allow Creation/Editing of Dashboard Visualizations	<input type="text"/>

The following settings can be overwritten:

Report Path

The parent folder for all reports. The Report Path can be:

- **Virtual Path**
- **Absolute Path:** used to provide increased security (ex. C:\Reports)
- **Web Service URL or .NET Assembly:** using a Web Service or .NET Assembly allows reports and folders to be managed in a database. For more information see **Report Folder Storage & Management**.
 - A Web Service should be formatted as 'url=http://WebServiceUrl.aspx'. A .NET Assembly should be formatted as 'assembly = AssemblyFullPath.dll;class=Namespace.ClassName'.

Date Format

The format of date values. Can be any .NET standard (ex. MM/dd/yyyy). Leave blank to use the browser culture.

Time Format

The format of time values. Can be any .NET standard (ex. h:mm:ss tt). Leave blank to use the browser culture.

Date Time Format

The format of date-time values. May be any .NET standard (ex. M/d/yy h:mm tt). Leave blank to use the browser culture.

NOTE. For more details on .NET Date, Time and DateTime Format Strings please see [here](#).

Numeric Separator Symbol

Symbol used to separate 3 digit groups (ex. thousandths) in numeric values. The default is ','.

Numeric Currency Symbol

Symbol prepended to numeric values to represent currency. The default is '\$'.

Numeric Decimal Symbol

Symbol used for numeric decimal values. The default is '.'.

Server Time Zone Offset

Value that is used to convert server to client time (the negation is used to convert client to server time). Leave blank to use server time, or to use **External Interface** to calculate value.

Show HTML Export Grid Lines

Sets the default value for the HTML output option Show Grid. This can be modified in the Options Menu of the Report Designer.

Show Crosstab Reports

Displays/Hides the Crosstab Report Wizard and Insert Crosstab button in the Report Designer.

Show Express Reports

Displays/Hides the Express Report Wizard.

Show Styling Toolbar

Displays/Hides the styling tools in the Layout tab of the Express Report Wizard.

Show Themes

Displays/Hides the Theme drop-down in the Layout tab of the Express Report Wizard.

Show Grouping

Displays/Hides the grouping tools in the Layout tab of the Express Report Wizard.

Show Formula Button

Displays/Hides the Formula Editor button in the Layout tab of the Express Report Wizard.

Show Advanced Reports

Displays/Hides the Advanced Report Wizard and Report Designer.

NOTE. If 'Show Advanced Reports' is False then attempts to edit Advanced or Crosstab reports will cause an 'access denied' message. Additionally if 'False', users will not be able to create Crosstab reports.

Database Timeout

Maximum number of seconds for a single query to run.

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 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.

Folder Access

The Folder Access controls which report folders are visible and executable for the Role.

Folders	
Folder Name	Read Only

+ Add

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 click 'New'.

Click in the 'Folder Name' column and select the Folder you want to add.

To make the folder execute only check the box in the 'Read Only' column.

To delete a folder click the 'Delete' button.

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.

Objects	
Data Object Name	

+ Add

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 'New'.

Click in the 'Data Object Name' column and select the Object you want to add.

To delete an Object click the 'Delete' button.

Allow User to View Report-Level Custom SQL Objects

(v2018.1+) If **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

The Filter Access provides a means to filter a Data Object by Role.

Filters	
Data Object Name	Filter String

+ Add

To add a filter click 'New'.

Click in the 'Data Object Name' column and select the filter you want to add.

Enter the filter string in the Filter String Column. The filter string should be Standard SQL. This string will be added to the 'Where' clause.

To delete a filter click the 'Delete' button.

Custom Functions

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.

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.

Creating Functions

To create a custom function, select 'Functions' in the Admin Console and click the 'Add' button. This will open a Custom Function tab.

Each Custom Function has the following properties:

Name

A name for the function that will be displayed to the end users.

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**.

Minimum Number of Arguments

(pre-v2017.2) The minimum number of values that an end user must enter in the function separated by commas.

Maximum Number of Arguments

(pre-v2017.2) The maximum number of values that an end user may enter in the function separated by commas.

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, select 'Other'. An input field will appear. Leaving this field blank will assign your Function to the 'Other' Category in the Exago Formula Editor. A non-empty value in this field will be new Category with the specified name.

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**.

Language

The high-level language of the code for the function. Can be C#, VB.NET, or JavaScript (Windows only).

Reference

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, 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.

Program Code

The program code for your Custom Function. Press the green check mark to verify the code executes properly.

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 2017.2 there are several enhancements to the way function arguments are implemented and used. See **Formula Editor** for details on how argument and function information will appear for end-users.

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 report's Data Object, sort, filter, and layout information.

SetupData

An object that contains all of the session'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 session'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.

E.g. If a parameter is prompting a user for a value it will be available with the prompted value.

GetConfigParameter(string parameterName)

A method that returns the parameter object stored in the default configuration.

E.g. Any parameter that is not being utilized by the report being executed.

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 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()`.

Examples

The following are two examples of Custom Functions. The first is written in JavaScript, the second in C#.

JavaScript Example

Reverses characters in the input string.

```
return args[0].split("").reverse().join("");
```

C# Example

Provides a function that returns the line number of the report being written by creating and incrementing a Stored Value which exists only for the report execution.

```
// this function creates a Stored Value and increments the value by 1 each time the value is rendered on a report
int i = (int)sessionInfo.GetStoredValue("IncrementNumber", 0);

// increment the value by 1 and return
sessionInfo.SetStoredValue("IncrementNumber", ++i);
return i;
```

Default Custom Functions

Starting with version 2017.2, Exago BI ships with several built-in custom functions. 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 as on an upgrade, you can use the following setting to restore them to the configuration: (**Filter Settings**) Restore All Default Formula Functions

The following custom functions ship with Exago BI:

MonthName

Given a date value, returns the name of the month of the date. The month name is retrieved from the active language dictionary. So, for example, given the date "01/01/2017", MonthName will return "January" in an English-speaking environment, and "Enero" in a Spanish-speaking one.

QuarterName

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.

QuarterNumber

Given a date value, returns the fiscal quarter of the date, as "1", "2", "3", or "4". Otherwise, this function has the same behavior as QuarterName.

Custom Filter Functions

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.

Creating Filter Functions

To create a custom function, select 'Date Functions' in the Main Menu and click the Add button. This will open a Date Function tab.

Each Custom Date Filter Function has the following properties:

Name

A name for the filter function that will be displayed to the end users.

Description

NOTE. To support multi-language functionality, if the filter function's name or description prepended with '_wrFunctionId' 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.

- For the example function below you could create a language id 'Beginning_of_Month_wrFunctionId'. The string associated with this id would be displayed instead of the name. For more information see **Multi-Language Support**.

Filter Type

Determines the "data type" of filters that the filter function should be available for.

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. 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.

Reference

A semicolon-separated list of any dlls that need to be referenced by the Date Function. If the dlls are not accessible in the GAC then the dlls must be copied to the 'Bin' folder of Exago or the reference should point to their physical path.

NOTE. System.dll does not need to be listed as a reference as it is already available.

Program Code

The program code for your Date Function. The code must return a DateTime value. Press the green check mark to verify the code executes properly.

NOTE. Parameters may be referenced within custom functions by placing their name between @'s.

The screenshot shows a configuration window for a custom function named 'FirstDayOfCurrentMonth'. The fields are as follows:

- Name: FirstDayOfCurrentMonth
- Description: FirstDayOfCurrentMonth
- Filter Data Type: date
- List Order: 131
- Language: CSharp
- References: (empty field with a green checkmark icon)
- Namespaces: (empty field)

The code editor contains the following code:

```
1 return new DateTime(sessionInfo.Today.Year, sessionInfo.Today.Month, 1);|
```

Example

The following is an example of a Custom Function.

- **Name** – Beginning_of_Month
- **Language** – C#
- **Program Code** –

```
// retrieve the first day of the current month
DateTime now = DateTime.Now;
DateTime FirstDayInMonth = new DateTime(now.Year, now.Month, 1);
// return as date time
return FirstDayInMonth;
```

Custom Options

This chapter explains how to create Custom Options. Custom Options provide a modifiable menu for end users to set values that can be utilized by Custom Functions, Server Events, or the API.

- To add a new Option select 'Custom Options' in the Main Menu then click the add button.
- To edit an existing Option either double click it or select it and click the edit button.
- To delete an Option select it and click the delete button.

About Options

Custom Options allow for the definition of settings that users can be 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 our **multi-language** feature. Custom Options can store several types of data such as integer, boolean, text, etc. Each data type provides an appropriate UI element for the user to select a value.

Creating Options

To create a Custom Option, select 'Custom Options' in the Main Menu and click the 'Add' button. This will open a Custom Options tab.

Each Custom Option has the following properties:

Id

The unique Id of the 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.

NOTE. To support multi-language functionality, create an element in the language file(s) with an Id that matches the Option'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 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.
- **Bool** – Represents a Boolean value. A checkbox is displayed.
- **Text** – Represents text and displays a text box.
- **List** – Represents a choice from among multiple values. Click the 'Add' button to define choices.

Setting Options

After Custom Options are created, the Custom Options Menu will be available in the Report Designer of Standard and Crosstab Reports. In the Custom Options Menu, options can be set using the UI elements displayed above.

NOTE. The Custom Options Menu will only display if Custom Options exist.

Accessing Options

The .Net Api, Server Events, and Custom Functions can access Custom Options values through the SessionInfo.Report object by using the following method:

```
string GetCustomOptionValue(string id)
```

Description	Returns the value of the specified Custom Option as a string.
Remarks	For Bool options the value returned will be "true" or "false". 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. <pre>return sessionInfo.Report.GetCustomOptionValue(args[0].ToString());</pre>

Hidden Flags

The following options are inaccessible from the Admin Console, but may be toggled on or off by editing the field in the config file xml.

- `<showcrosstabwizard>` – Show or hide the CrossTab Wizard button in the Report Designer.
- `<allowhtmlinscheduledemails>` – Set to True to allow users to insert html tags within the body of scheduled emails.
- `<showbrowseroutofdatewarning>` (v2016.3.4+) – Set to False to prevent a popup error message from appearing if a user accesses Exago with an unsupported browser.
- `<webfarmssupport>` – Must be set to True if running in a web farm environment.
- `<expressviewdefaultformattheme>` – (v2017.1+) Select a default ExpressView theme.
- `<aliasallentities>` – (v2017.2+) Set to True to always use data object Ids as an alias in the generated SQL.
- `<safemode>` – (v2017.2+) Set to True to ensure that data object names and Ids are unique. If not, log an error.
- `<allowearlypagebreak>` – (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.
- `<loadassemblyinexternaldomain>` – (v2018.1+) Set to True to load Assembly Data Sources in an external domain, as opposed to the application domain.
- `<canjointransformobjectsindb>` – (v2018.1+) Set to False to cause vertical table transformations to be processed in the application instead of in the database. This will prevent the use of advanced joins with vertical tables.
- `<cacheconfig>` – (v2018.1+) Set to True to cause the configuration data to be cached in the session. This can increase performance when using extremely large config files.
Warning: Any in-session operations which modify the configuration will cause erroneous behavior. Such operations include: OnConfigLoadEnd server event, advanced joins, vertical table transformations.
- `<performancetesting>` - (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>` - (v2018.2+) Set to False to opt out of caching in the GetReportList method when using a Folder Management assembly.

Any fields which are not mentioned here are either not intended for external use or not fully implemented, and should be ignored.

Setting Up Monitoring

Monitoring is a new feature of Exago v2017.1 that allows you to track report management, execution, and performance statistics for the web application and schedulers. Monitoring data is stored in local sqlite database files, and can be reported on using Exago.

When you install the Exago web application, the monitoring system is automatically installed, but it is disabled by default. You must configure and enable it manually. Many actions in the application can be tracked:

- 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.

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

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:



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:\file\path\`" (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 `ExagoWeb`, 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 `ExagoWeb\Monitoring`.

Example

```
<appSettings>
<add key="Monitoring.DbPath" value="C:\Exago\ExagoWeb\Monitoring\" />
...
</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:

Note. Keys are formatted as "`Monitoring.Option`"; The "`Monitoring.`" prefix is omitted below.

- `CollectDeleteReportUsage`
- `CollectRenameReportUsage`
- `CollectDuplicateReportUsage`
- `CollectExecuteReportUsage`
- `CollectSaveReportUsage`
- `CollectSaveReportXmlUsage`
- `CollectDesignReportUsage`
- `CollectGaugeControlUsage`
- `CollectGoogleMapControlUsage`
- `CollectMapControlUsage`
- `CollectChartControlUsage`
- `CollectScheduleReportControlUsage`
- `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 application:

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 polling service

The monitoring system uses a Windows or Linux service that updates the core database with data from the web application and schedulers 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

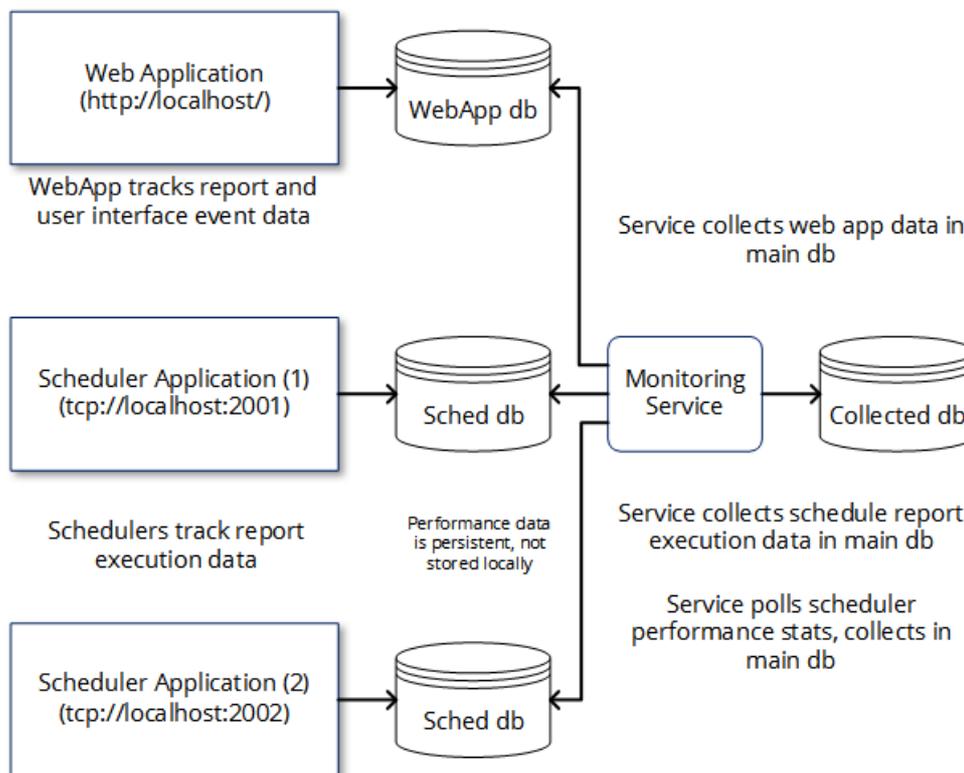
1. As an administrator, open the Services manager:
 1. Click **Start>Run**.
 2. Type `services.msc`.
 3. Press Enter.
2. Locate the service `Exago Monitoring Service vX.X.X.X`, where `vX.X.X.X` is your Exago version. Right-click the service and select **Properties**.
3. In the Properties window:
 1. From the Startup Type list, select **Automatic**.
 2. Click **Start** to start the service.
 3. Click **OK**.

Using the monitoring database with Exago

Before using monitoring data in reports, you need to add the core database file `MonitoringService/Monitoring.sqlite` as an Exago data source. See [Using SQLite Data Sources](#) for instructions.

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 application 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\Monitoring\Monitoring.sqlite - WebApp db

ExagoWeb\appSettings.config - Select which web app data to track

MonitoringService\Monitoring.sqlite - Main collected db

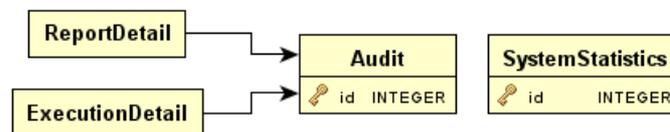
MonitoringService\Monitoring.exe.config - Set Extraction & Statistics intervals

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

Introduction to Integration

Exago is designed to be seamlessly integrated into the host application. Integration can entail either styling Exago' interface to match the host or making API calls such as report execution directly from the host application. To access the user interface, Exago can either be embedded in a div or iframe or users can be directed to a separate page.

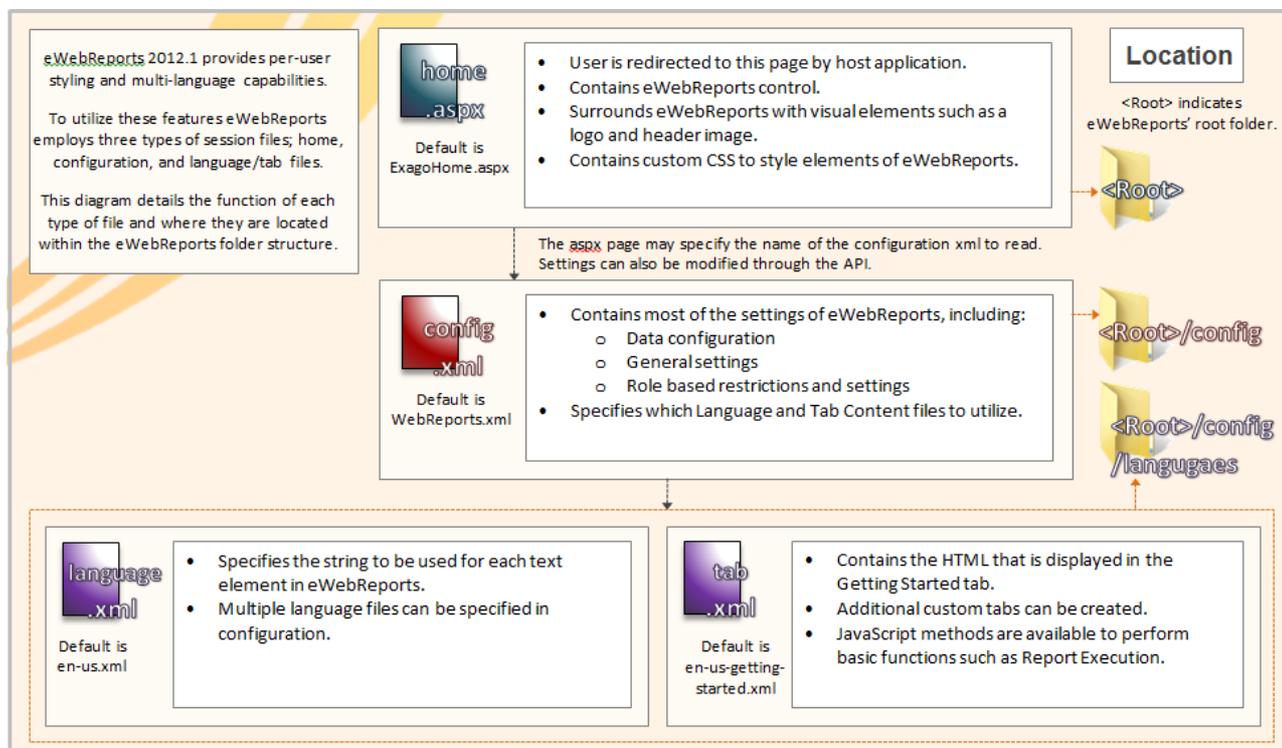
Whether you are exposing the provided interface or calling API methods it is important to:

- **Ensure users are verified through the host application:** Users should be signed in through the API to access Exago. To ensure that this happens, disable direct access to Exago by setting the parameter 'Allow direct access to Exago' to False in the **Main Settings**.
- **Assure the correct permissions and features are available to the user:** As the user is signed in, activate the correct role and set values for any necessary parameters to assure that the user can only access the data, features, folders and reports that he/she has permission to use. For more information see **Roles**.

To further integrate Exago you can:

- Re-style the user interface to match the aesthetic of your application. See **Styling**.
- Translate or modify any text that appears in the user interface. See **Multi-Language Support**.
- Customize the Getting Started Tab and/or create additional custom tabs. See **Customizing Getting Started Content**.
- Integrate the Exago installer into the host application's installer. See **Manual Application Installation**.

Integration utilizes several types of files. The diagram below details the role of these files:



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.aspx" 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** – Loads a configuration file other than that created by the Administration Console (ex. ConfigFile="NorthwindConfig.xml").
- **LanguageFile** - Specify which language file(s) to use in place of the 'Language File' parameter of **Main Settings** in the configuration file. (ex. LanguageFile ="es-mx, getting-started-custom").
- **ForceIECompatMode** – 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. (ex. ForceIECompatMode="true").
- **XUaCompat** – Setting that 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. (ex. XUaCompat="true").

Note. IE8 is no longer supported by Exago, as of version 2016.1.

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= "Config\Images\Custom\HTMLExecutIconLarge.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 browser'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).

```
img#wrCtrl1a8eb_DesignSplitter_DesignToolBarCtrl_SaveReportBtn.wrTbImg.wrDsgnTbImg | 18 x 18
Element ID
UI Element
```

Styling the Administration Console

Though we strongly recommend **against** exposing the administration console to end-users or clients, it can be stylized much like the Exago interface.

To style the administration console:

1. Make a copy of ExagoHome.aspx and give it a unique name (ex. CompanyAdmin.aspx)
2. At the top of this copy change the source from *WebReportsCtrl.ascx* to *WebAdminCtrl.ascx*:

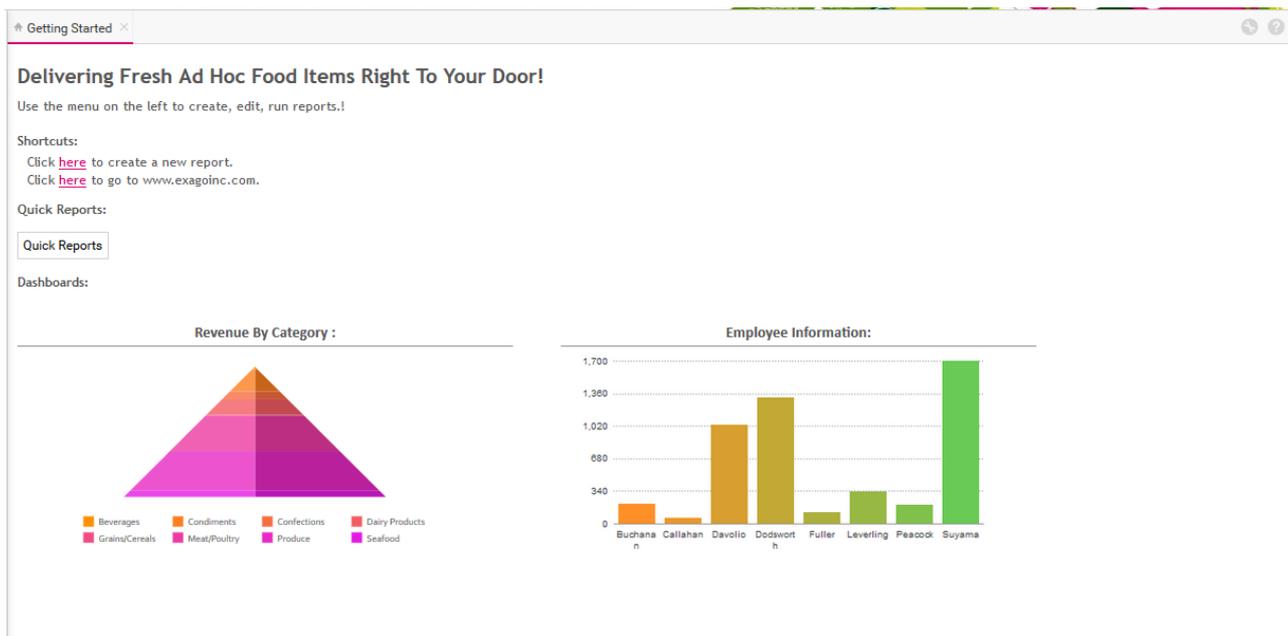
```
<%@ Page Language="C#" EnableViewState="false" %>
<%@ Register src="WebAdminCtrl.ascx" tagname="WebAdminCtrl" tagprefix="wr" %>
```

3. Modify css and images in the same manner described in the sections above.

Customizing Getting Started Content

The Getting Started tab is displayed as a user enters Exago. This tab can be customized by loading custom html. This is done by modifying the language element 'GettingStartedContent' in the file 'en-us-getting-started.xml'. To assist in customizing the Getting Started tab, Exago provides several JavaScript functions to open the New Report Wizard, execute reports, open other custom tabs and display reports as dashboards.

The example below demonstrates a custom tab with links to the New Report Wizard and Dashboards.



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**.

Creating Additional Custom Tabs

Additional custom tabs can be created by creating a language element with a unique name containing the html content. Custom tabs can be opened with the JavaScript function **wrAddTabbedContent** (see **Available JavaScript Functions**).

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>
```

Available JavaScript Functions

To assist with the creation of custom tab content, Exago provides a small number of JavaScript functions to allow custom html to call features of Exago.

`void wrStartNewReportWizard([string reportType])`

Opens the New Report wizard in a new tab. Optionally, specify which wizard to open.

Possible values for *reportType*:

- | | |
|--------------------|--|
| Description | <ul style="list-style-type: none"> • advanced • express • expressview • dashboard • chained |
|--------------------|--|

Example	Ex. <i>Click here<="" a="" create="" i="" new="" onclick="wrStartNewReportWizard();" report.<="" span>="" style="text-decoration: underline; cursor: pointer;" to=""></i>
----------------	--

`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	Ex. <i>Click here<="" a="" create="" duplicate="" i="" onclick="wrStartDuplicateReportDialog();" report.<="" span>="" style="text-decoration: underline; cursor: pointer;" this="" to=""></i>
----------------	--

`void wrExecuteReport(string reportFolder\reportName, string format)`

Description Executes the specified report in the specified format.

Example	Ex. <code><input type="button" class="Button" value="HTML" onclick="wrExecuteReport('Sales Reports\\Revenue by Category','html')></code>
----------------	--

`string wrGetSelectedReportName()`

Description Returns the name of the report that is selected in the Main Menu.

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 Content ID.

Remark The ContentID should match the element ID of the html you want to load.

Remark 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.

`data-onloadreportname= "ReportFolder\ReportName"`

Description Executes a report as HTML and loads it into a div or iframe.

Remark The report string should be formatted as "Report Folder\Report Name."

Remark NOTE. When using this function make sure the setting **Enable Debugging in Other settings** is **False**.

Example	Ex. <code><div class="Report" data-onloadreportname="Employee Reports\Number of Sales by Employee"></div></code>
----------------	--

`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 Exago viewer is not need set to False to load raw html.

Example	Ex. <code><div class="Report" data-onloadreportname="Employee Reports\Number of Sales by Employee" data-useviewer= "False"></div></code>
----------------	--

data-enablescrolling ="True/False"

Description	Specifies whether or not to show scroll bars.
Remark	Default value is True. This can helpful for certain reports that may not fit exactly within the startup content.
Example	Ex. <code><div class="Report" data-onloadreportname="Employee Reports\Number of Sales by Employee" data-enablescrolling= "False"></div></code>

data-reloadinterval="n"

Description	Reloads a report every <i>n</i> seconds.
Remark	This function is used in conjunction with data-onloadreportname .
Example	Ex. <code><div class="Report" data-onloadreportname="Employee Reports\Number of Sales by Employee" data-reloadinterval="2"></div></code>

data-allowexport="0/1"

Description	Specifies wether or not to show the re-export menu for the report.
Remark	The default value is 0 (does not show the menu). Set to 1 to have the re-export options display.
Example	Ex. <code><div class="Report" data-onloadreportname="Employee Reports\Number of Sales by Employee" data-reloadinterval="1"></div></code>

Custom Context Sensitive Help

Exago is installed with context sensitive help. When a user clicks the help button a tab appears displaying 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 webpage for the custom help.
2. Set the URL of the webpage in the Custom Help Source parameter in **Feature/UI Settings**. Ex
url=http://www.Customhelp.com/Exago;

NOTE. When a user clicks the help button Exago will populate a tab with the content received from the URL. To notify the host application the user's language the URL will be appended with the 'Language File' of **Main Settings** and a context parameter (listed below). Ex. <http://www.customhelp.com/Exago?helpKey= newreport&language=en-us>

Context Parameter	Details
tabexecute	The user has report output active.
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.
New 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.
New Report Wizard	
tabStandardName	The user has the Names tab of the New Advanced Report Wizard active.
tabStandardCategories	The user has the Categories tab of the New Advanced Report Wizard active.
tabStandardSorts	The user has the Sorts tab of the New Advanced Report Wizard active.
tabStandardFilters	The user has the Filters tab of the New Advanced Report Wizard active.
tabStandardLayout	The user has the Layout tab of the New Advanced Report Wizard active.
Report Designer	
tabDesign	The user is editing an Advanced or Crosstab report and has the design grid active.
dialogName	The user has the Rename Menu active.
dialogDescription	The user has the Description Menu active.

dialogCategories	The user has the Categories Menu active.
dialogSorts	The user has the Sorts Menu active.
dialogFilters	The user has the Filters Menu active.
dialogGeneralOptions	The user has the Options Menu active.
listItemReportHtmlOptionsGeneral	The user has the General section of the HTML Options active.
listItemReportHtmlOptionsFilters	The user has the Filter section of the HTML Options active.
listItemReportHtmlOptionsSorts	The user has the Sorts section of the HTML Options active.
dialogTemplate	The user has the Template Menu active.
dialogJoins	The user has the Advanced Menu active.
dialogJoinEdit	The user has the Report Join Menu active.
dialogFormulaEditor	The user has the Formula Editor active.
dialogLinkedReport	The user has the Linked Report Menu active.
tabCellFormatNumber	The user has the Number tab of the Cell Format Menu active.
tabCellFormatBoder	The user has the Border tab of the Cell Format Menu active.
tabCellFormatConditional	The user has the Conditional tab of the Cell Format Menu active.
dialogCrosstabDesign	The user has the Crosstab Menu active.
dialogGroup	The user has the Group Section Menu active.
dialogSectionShading	The user has the Section Shading Menu active.
tabChartType	The user has the Type tab of the Chart menu active.
tabChartLabels	The user has the Labels tab of the Chart menu active.
tabChartData	The user has the Data tab of the Chart menu active.
tabMapType	The user has the Type tab of the Map menu active.
tabMapLocations	The user has the Locations tab of the Map menu active.
tabMapData	The user has the Data tab of the Map menu active.
tabGaugeType	The user has the Appearance tab of the Gauge menu active.
tabGaugeData	The user has the Data tab of the Gauge menu active.
Dashboards	
tabDashboardDesigner	The user has the Dashboard designer active.
dialogDashboardUrlOptions	The user has the Insert Url menu active.
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 has the Report tab of the Insert Report menu active.
tabDashboardReportOptionsFilterPrompts	The user has the Filters tab of the Insert Report menu active.
tabDashboardReportOptionsParameterPrompts	The user has the Parameters tab of the Insert Report menu active.
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 has the Options menu of a Data Visualization active.
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.

NOTE. 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 shown.

Themes allow a user to quickly stylize reports or elements of reports such as maps and charts. Exago comes with several themes pre-installed. Additional custom themes can also be created.

Pre-installed themes are saved in the Themes folder of Exago. By default custom themes are saved in the Report Path, which is specified in **Main Settings**. Alternatively the host application can manage theme storage by implementing the GetTemplate, GetTemplateList, and SaveTemplate functions. See **Report and Folder Management** for more information.

NOTE. To support multi-language functionality, if the theme name concatenated with '_wrThemeld' 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. **Ex.** For the Basic theme that is installed with Exago, there exists a language id 'Basic_wrThemeld'. The string associated with this id is displayed. For more information see **Multi-Language Support**

Chart Themes

A user can quickly select colors for Charts by applying a chart theme.

To create custom Chart themes:

1. In folder specified in the Report Path of **Main Settings** create a text file containing a comma separated list of the css values of the desired colors. Save the file and change the extension to 'wrth'.

NOTE. The file name will be displayed to the end user. To translate the name of a custom theme, see the note above section.

Ex: The theme 'Cocktails In Miami.wrth' contains the list: Navy, #00ff00, Yellow, Orange, Red.

Crosstab Themes

A user can quickly style Crosstabs by applying a crosstab 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 as several Tabulation Data, Row Headers, Column Headers as well as sub-totals and grand totals.

NOTE. If a user adds more Tabulation Data, Row Headers or Column Headers than existed on the theme they will appear without styling. We recommend Crosstab Themes have 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.

2. In the Report Designer stylize each cell of the Crosstab as desired.
3. Move your cursor over the Crosstab. Notice a dropdown menu appears in the bottom left corner.
4. Hold Alt+Ctrl+Shift and click on the dropdown.

1	Product Info				
2				ProductID	
3				SupplierID	
4				UnitsIn Stock	
5	CategoryName	ProductName	Discontinued	ReorderLevel	UnitsOnOrder
6	Categories. Cate	Products. Produ	Products. Disco	Products.ReorderLevel	
7	goryName	ctName	ntinued		
8					

☰

-  Modify
-  Save as Theme
-  Delete

5. Click 'Save as Theme'.
6. Enter a name for the Theme. This name will be displayed to the end-users.

Express Report Themes

A user can quickly style Express Reports by applying an express report theme. Express report themes can specify background color, foreground color, section shading, borders, fonts and text size.

To create custom Express Report themes:

1. Create an Express Report with Headers, Footers and a Page Header/Footer and a Grand Total.

NOTE. If a user adds more Columns, Headers, or Footers than existed on the theme they will appear without styling. We recommend Express Report Themes utilize many Columns, Headers and Footers.

2. In the Layout tab stylize the report as desired.
3. On your keyboard, hold **Ctrl+Alt+Shift** and click on the save button .
4. 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.

To create custom Geochart themes:

1. In folder specified in the Report Path of **Main Settings** create a text file containing a comma separated list of the css values of the desired colors. Save the file and change the extension to 'wrtm'.

NOTE. The file name will be displayed to the end user. To translate the name of a custom theme, see the note above section.

Ex: The theme 'Cocktails In Miami.wrtm' contains the list: Navy, #00ff00, Yellow, Orange, Red.

Multi-Language Support

NOTE. The language elements discussed in this section 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**.

To help localize Exago, any text in the application can be translated or modified. This can be accomplished by creating xml files in the Language folder that map ID's 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 'Language File' 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="NewReportLb1">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>
```

NOTE. 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'. Without changing the IDs translate the strings as desired (see example above). Then set the 'Language File' parameter of **Main Settings** to specify the desired translation.

NOTE. If you are using the Exago Scheduler Service be sure to copy all custom language xml files to the '<scheduler_dir>/Languages' folder of the Scheduler Service.

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 parameter 'Language File' list the all of the files you want to load separated by comas 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' parameter to 'en-us, en-custom' and the Getting Started tab will reflect the change made in the custom file.

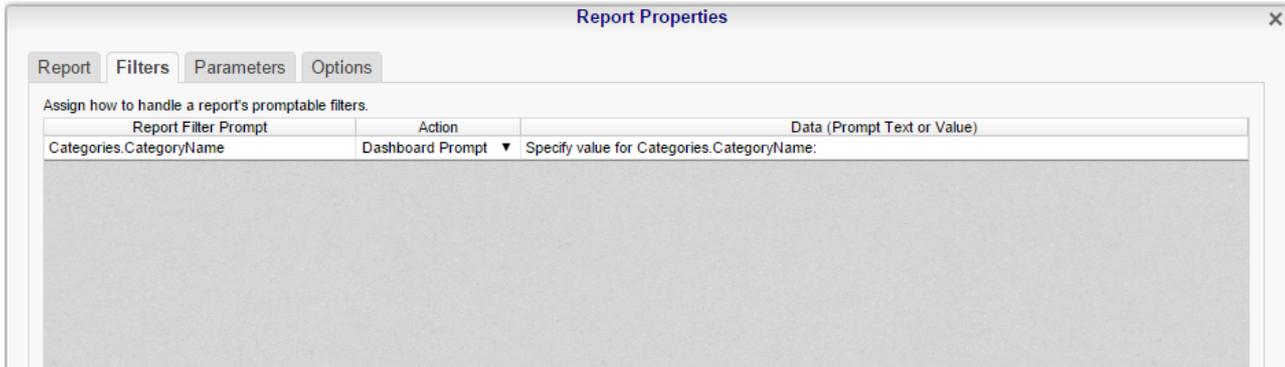
NOTE. Begin all language xml files with the line '<?xml version="1.0" encoding="utf-8" ?>'

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.



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

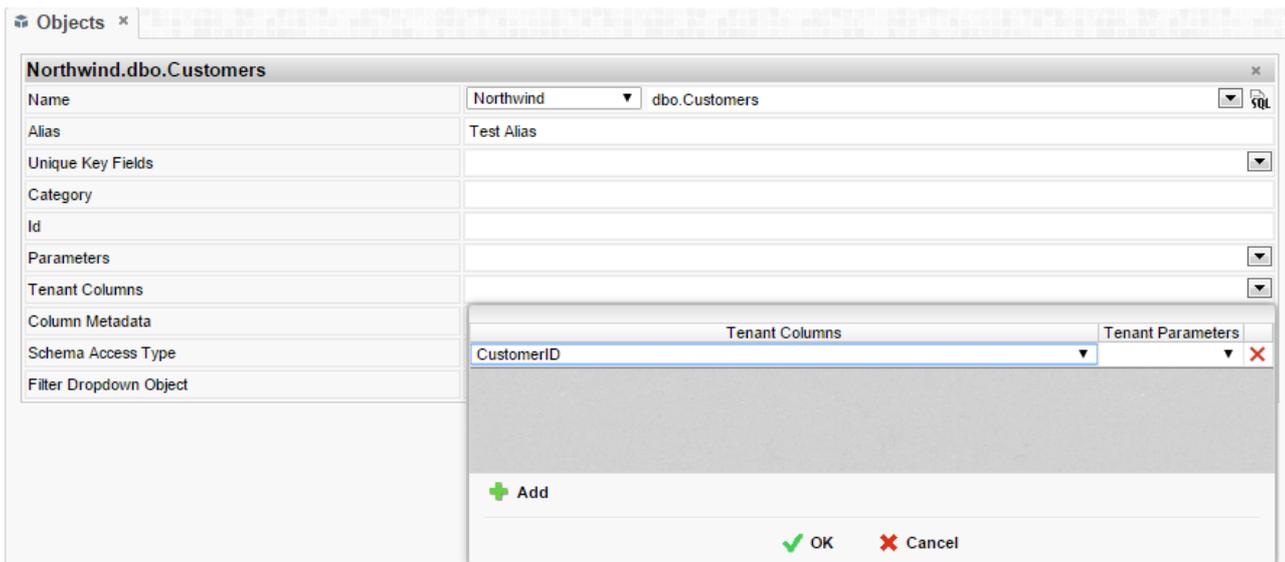
The most basic multi-tenant environment is when each table, view and stored procedure has one or more columns that indicate which user(s) has access to each row.

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.



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 (blank for default)' to any valid value.

- 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.
- When initializing Exago through the API, set the schema on the Data Source for the current user.

Northwind	
Name	Northwind
Type	mssql
Schema/Owner Name (blank for default)	example
Connection String	Server=<servername>;Database=<database>;uid=<user>;pwd=<password>;

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:

- Create a Data Source and corresponding Data Objects using any one of the Databases.
- When initializing Exago through the **API**, set the connection string on the Data Source to access the appropriate database for the current user.

Northwind	
Name	Northwind
Type	mssql
Schema/Owner Name (blank for default)	example
Connection String	Server=<servername>;Database=<database>;uid=<user>;pwd=<password>;

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:

- For each Data Object open the **Custom SQL** menu and create the desired SQL utilizing parameters to assure only appropriate information is available.

NOTE. Parameters should be surrounded by single quotes.

- When initializing Exago through the **API**, set the value of any parameters utilized in the SQL for the current user.

Custom SQL Object			
Object Name	Customers	Data Source	Northwind
Parameter		Parameter	+ Add
Test		Test	✓

```

1 Select *
2 From Customers
3 Where Tenant_Column = '@TenantParam@'

```

OK Cancel

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.

Overview

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
- External Interface
- Custom Options

Custom SQL

Exago supports a Custom SQL type Entity. 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 @StartDate@ and @EndDate@ 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 an 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 addition to traditional relational data sources Exago can consume data from *programmable* data sources such as .NET assemblies and SOAP-based web services. 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.

External Interface

The External Interface is a set of pre-defined event types that can be trapped and instrumented. External Interface events are similar to Server Events with two key differences: External Interface can be invoked via web service (whereas Server Events require code developed using the Administration Console code editor or via .NET assemblies). As a result External Interface methods do not have access to the global `SessionInfo` object.

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 Server Events

This section explains how to create Events Handlers that run custom code when certain actions occur in the Exago BI runtime.

- To add a new Event Handler select 'Server Events' in the Main Menu then click the add button.
- To edit an existing Event Handler either double click it or select it and click the edit button.
- To delete an Event Handler select it and click the delete button.

Event Handlers

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 within the Exago configuration file.

All existing Event Handlers are listed in the **Main Menu** under Server Events. All the Event Handlers you are adding or editing will be displayed in a **Tab** entitled Server Events.

Each Event Handler has the following properties:

- **Name** – Provides a unique identifier for each Event Handler
- **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 menu. See **Custom Code** for information on how to access the arguments for each Event. Click the green check mark to verify the code compiles.

Custom Code has three properties:

- **Language** – Code can be written in C#, VB.NET, or JavaScript (Windows only).
- **References** – A semicolon-separated list of any .NET Assembly dlls that need to be referenced by the Event Handler (JavaScript code cannot access .NET libraries).

NOTE. System.dll does not need to be listed as a reference as it is already available.

- **Code** – The code that will be executed by Exago when called.

C# Example

```
System.Data.DataTable dt = (System.Data.DataTable)args[0];
if (dt.Columns.Contains("Employees.BirthDate"))
    foreach (System.Data.DataRow row in dt.Rows)
        for (int i = 0; i < row.ItemArray.Length; i++)
            row["Employees.BirthDate"] = DBNull.Value;

return dt;
```

- **.NET Assembly Method** – 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 (for ExagoWeb, WebServiceApi, and all Schedulers, 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);
```

 See **.NET Assemblies** for info on how to access the arguments for each Event.
- **Global Event** – In this dropdown select an Event to indicate that the Event Handler should be called whenever this event occurs for **all** report execution. Leave Global Event set to 'None' to indicate the Event Handler is meant for a specific report. E.g. Selecting *OnReportExecuteStart* from this dropdown will cause the Event Handler to be called at the start of any Report Execution.
- **None** – The Event Handler will **not** be called automatically for all reports, but can be set to run for the execution of specific reports. See **Setting Event Handlers on Specific Reports** for more information.

Arguments

Server events can access the following information in order to inspect the session state, and utilize built-in methods:

- **sessionInfo** – The **sessionInfo** object provides access to elements of Exago's current session such as parameters, filters, and the current report.
- **args** – 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 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 needed).
- 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 BI dll is referenced by the assembly, then it must be recompiled to the current version whenever Exago BI is updated.

Adding Server Events to Specific Reports

Event Handlers can either be set to run during the execution of every report or to only be called when executing specific reports.

NOTE. When multiple Event Handlers are set to run for a single Event, all the Event Handlers are run using the same input values and then the first non-null return value is used by Exago. This means that the return value of Report-specific Event Handlers will take precedence over global Event Handlers.

Ex. Suppose there is a global Event Handler for *OnExecuteSqlStatementConstructed* that logs each reports SQL query and a report specific Handler that modifies the 'Where' clause of the SQL. When the specified report is run, both Handlers will be executed and return an SQL string. If non-null, the modified SQL from the report specific Event Handler will be utilized by Exago to query the database.

To set an Event Handler to be report specific:

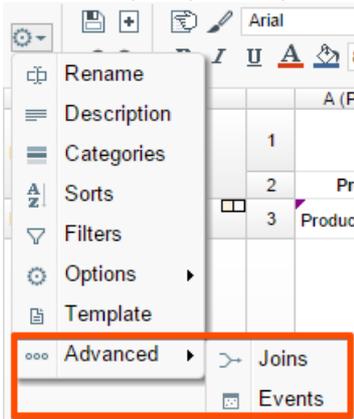
In the Administration Console:

1. Set the Event Handler's Global Event to None. Click Apply or Ok.

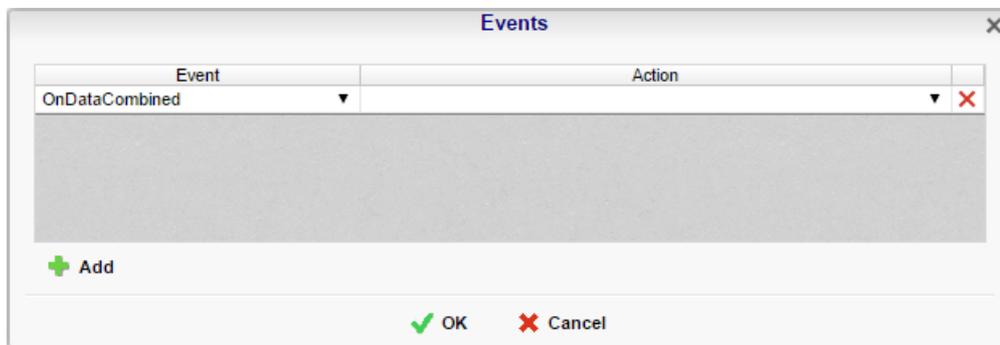
2. In the **Feature/UI Settings** set Show Events Window to True. Click Apply or Ok.

In the Reporting Application:

1. In the Main Menu select the desired report and double click or click the edit button.
2. Select the Report Options drop-down menu and hover over Advanced. Click Events. This will cause the Events Menu to appear.



3. In the Event Menu click the Add button.
4. From the Event dropdown select when the Event Handler should be called.
5. From the Action dropdown select which the desired Event Handler.
6. Click Okay and save the report.



Displaying User Messages

Some Server Events are designed to display messages to the user based on a return value. For the other server events a user message can be displayed by throwing the following exception method.

WrUserMessage(string messageOrId, wrUserMessageType Type)

Description Displays a message to the user.

Remarks
 wrUserMessageType can either be Text or Id.
Text – The user message will display the string message
Id – The user message will display the string associated with the Id in the Language Files.
 This requires a reference to WebReports.Api.Common

Example

```
//OnWebServiceExecuteEnd, inspect the returned value and throw a
//message if it matches any of the error messages.

object webServiceResult = args[0];

switch(webServiceResult.ToString())
{
    case "message1" : throw new WrUserMessage("Some Message to User", WrUserMessageType.Text);
}

return webServiceResult;
```

Note: This cannot be used for the Events **OnConfigLoadStart**, **OnConfigLoadEnd** or **OnExceptionThrown**.

SessionInfo

Custom Functions, Server Events, Action Events, and Assembly Data Sources can access the Exago session state through a "sessionInfo" variable. 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.

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 report's Data Object, sort, filter, and layout information.

SetupData

An object that contains all of the session's configuration settings including Functions, Parameters, Data Objects, Joins, Roles, etc.

CompanyId

Contains the value specified by the companyId **Parameter**.

UserId

Contains the value specified by the userId **Parameter**.

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 session'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.

Ex. If a parameter is prompting a user for a value it will be available with the prompted value.

GetConfigParameter (string parameterName)

A method that returns the parameter object stored in the default configuration.

Ex. Any parameter that is not being utilized by the report being executed.

WriteLog (string text)

A method that writes the specified text to the Exago's 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 Store 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 Store Value. Setting newValue to 'null' will delete the Stored Value.

Calling Functions

To call an existing function from within your extension, use the class `CellFormula` and return the method `CellFormula.Evaluate(null)`.

Example

```
CellFormula formula = CellFormula.CreateFormula(sessionInfo.PageInfo, formulaText, CellVariableCollectionFilter.De
return formula.Evaluate(null);
```

CellVariableCollectionFilter types:

DataField
 AggFunction
 CellReference
 LinkedReport
 Parameter
 WidgetCellReference
 All

Introduction to Action Events

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 Exago UI 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 usages.

Creating Event Handlers

Action event handlers are created using the Admin Console or by directly editing the `WebReports.xml` config file. They can also be added or modified on a per-session basis in a .NET configuration using the 'Api.SetupData.ActionEvents' server call.

- To create a new Event Handler expand 'Extensions' in the Main Menu, select 'Action Events', and click the add button.
- To edit an Event Handler either double click it or select it and click the edit button.
- To delete an Event Handler select it and click the delete button.

The Action Events tab will open and display the selected event or a New Action Event dialog:

The screenshot shows a dialog box titled "OnLoadChart" with a close button (X) in the top right corner. The dialog contains the following fields:

- Name:** OnLoadChart
- Function:** Custom Code (selected in a dropdown menu)
- Event Type:** Load (selected in a dropdown menu)
- Global Event Type:** None (selected in a dropdown menu)
- Assigned UI Item(s):** (empty text field)

At the bottom of the dialog, there are two buttons: "Apply" (with a blue arrow icon) and "OK" (with a green checkmark icon).

Each Event Handler has the following properties:

- **Name** – A unique identifier for each Event Handler.
- **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 menu. See **Writing Action Events** for information on how to access the arguments for each Event. Press the green check mark to verify that the code compiles.

Custom Code has four properties:

- **Language** – Code can be written in C#, VB.NET, or JavaScript (Windows only).
- **References** – A semicolon-separated list of any .NET Assembly dlls that need to be referenced by the Event Handler.

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.
- **.NET Assembly Method** – 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 (for ExagoWeb, WebServiceApi, and all Schedulers, 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 Assembly'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 Event Handlers the Assembly must include a reference to WebReportsApi.dll. For more information see **SessionInfo**.

Note: All methods used as Event Handlers must be static.

- **Event Type** – Select an option in this dropdown to create an event that will be executed when certain client-side actions are taken.
 - **None** – This event handler is a Global Event. You must specify a Global Event Type in the following dropdown.
 - **Load** – The event handler will execute when a report item is loaded in the Report Designer, Viewer, or (**v2016.2.5+**) upon Export. This type of handler is typically used to interpret and then apply alterations to report data, e.g. conditionally changing the colors on charts or maps. As of **v2016.2.5** Load events can affect Export formats (PDF, Excel, RTF, CSV).
 - **Click** – The event handler 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 on 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. See **Global Events**.

Note: Selecting a **Global Event Type** will cause Exago to ignore any selected Local **Event Type**.

- **Assigned UI Item(s)** – This field designates a comma-separated list of UI item IDs for items in the Exago interface. These elements can be intercepted and modified by assigning them in this field. For a list of compatible UI items, see **UI Elements**.

Note: 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 **SessionInfo**. The most relevant elements are the following:

Note: To access the sessionInfo from a .NET Assembly, you must include a reference to WebReportsApi.dll.

- **SetupData** – The Admin Console options and data.
- **UserId** and **CompanyId**
- **Report** – The current report object.
- **JavascriptAction** – This object is set when sessionInfo is called from an action event. It is primarily used to load the client-side Javascript:
 - **JavascriptAction.SetJsCode**(string JsCode) – Pass the client side code as a string. The action event must return the JavascriptAction object.

- **clientInfo** – A JavaScript object that is accessed from within the client-side script. Provides access to any specified client-side

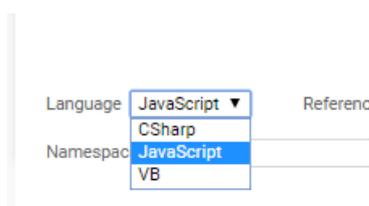
parameters and information about the item attached to the event handler. For a breakdown of the elements in `clientInfo`, see **ClientInfo**.

arguments array – 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 function `clientInfo.ServerCallback(eventName, args...)`.

JavaScript

Note: This is only available in Windows environments.

Both the server-side and client-side code for action events can be written in JavaScript. The client side code must still be passed to the `sessionInfo.JavascriptAction` object as a string. This can be done by calling `toString()` on a function, then concatenating the invocation operator, before passing it to the `JavascriptAction.JsCode`. This can be an easier way to write client scripts since the code is written natively, instead of as a string literal. However, note that writing server code in JS means that it cannot access any C#, .NET, or CLR libraries.



Selecting JavaScript from the Custom Code window

Example of writing client-side JS in the Custom Code window

```
// this function wraps the client-side code
function debug() {
    debugger;
}

/* any other server-side processing can be done in JS */

// call Function.toString() on the client-side function, then concatenate
// the "invoke" operator (), before passing it to the JavascriptAction
var jscode = "(" + debug.toString() + "());";
sessionInfo.JavascriptAction.SetJsCode(jscode);
return sessionInfo.JavascriptAction;
```

Note: The `clientInfo` object is only accessible from within the client code, not on the server. However, it could be passed to the server in a callback as JSON using:

```
clientInfo.ServerCallback("eventName", JSON.stringify(clientInfo));
```

This will require either two separate action events - one to send the object, and one to receive it - or one with some conditional logic to handle both cases.

Adding Action Events to a Report

To enable an end-user to add Action Events to items in a report, the user must have access to the Report Viewer and the Action Events toolbar option in the Report Designer. The options to enable these features are located in the following sections of the Admin Console:

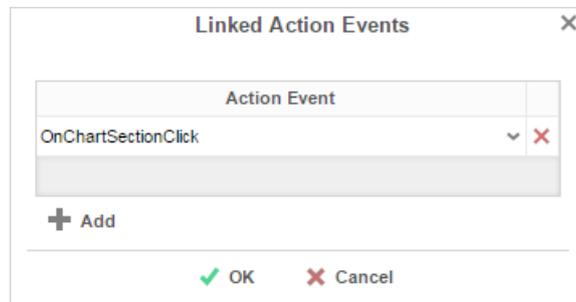
- **Main Settings**

Allow Execution in Viewer

- **Feature/UI Settings > Advanced Report Designer Settings**

Show Linked Action

After a Local Action Event has been created, the event will be available to add to a report. In the Report Designer, select the cell to which to add an event and click on the **Linked Action Event** button. The Linked Action Events Menu will open:



Press Add and select the event from the dropdown list. Press Delete to remove the selected event. Press OK when finished.

If the event is a Load event, you will have to save and re-open the report to see the changes applied in the Report Designer.

Global Action Events

Global Events are actions that can be attached 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.

Please note that a **subset** of global events, namely the ones which are used to handle report tree interaction, require a *true* or *false* return value in the client script. *True* indicates to Exago that we don't want to continue with the "normal" course of action, which we have replaced with our custom code. *False* indicates that we should continue with the normal action.

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*.

Also note that 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 true/false return value are labeled.

OnSaveReport

Description Fires when a user attempts to save an open report. This action event overrides 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 Passes the report object.

OnEditReport (v2016.3+)

Description Fires when a report is opened for editing.

Remarks Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to continue normal operation. Must be enclosed in a function.

OnSelectReport (v2016.3+)

Description Fires when a report item in the folders tree is selected.

Remarks Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to

continue normal operation. Must be enclosed in a function.

OnDeleteReport (v2016.3+)

Description	Fires when a report is deleted from within the folders tree.
Remarks	Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to continue normal operation. Must be enclosed in a function.

OnRenameReport (v2016.3+)

Description	Fires when a report is renamed from within the folders tree.
Remarks	Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to continue normal operation. Must be enclosed in a function.

OnExecuteReport (v2016.3+)

Description	Fires when a report is executed from within the folders tree.
Remarks	Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to continue normal operation. Must be enclosed in a function.

OnDoubleClickReport (v2016.3+)

Description	Fires when a report item in the folders tree is double-clicked.
Remarks	Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to continue normal operation. Must be enclosed in a function.

OnRightClickReport (v2016.3+)

Description	Fires when a report item in the folders tree is right-clicked.
Remarks	Passes the webReportsCtrl object, i.e. the application DOM, including the main UI window, folders tree, main menu, etc. Returns true or false to indicate whether to continue normal operation. Must be enclosed in a function.

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 (v2017.1.5+)

Description	Fires after an existing report is successfully saved from a Wizard, Designer, or Viewer. Works for all report types. Also fires when 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 for these cases.

Remarks

This can be useful for implementing 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.

Actionable UI Elements

The following report designer toolbar items can be attached a **"Click" action event handler**:

ID	Icon
ReportOptionsBtn	
SaveReportBtn	
DesignNewReportBtn	
UndoBtn	
RedoBtn	
FormatCellsBtn	
FormatPaintbrushBtn	
FontNameList	<input type="text" value="Arial"/>
BoldBtn	B
ItalicBtn	<i>I</i>
UnderlineBtn	<u>U</u>
UnderlineSelect	<u>U</u> 
ForegroundColorBtn	
BackgroundColorBtn	
FontSize	<input type="text" value="8"/>
AlignTopBtn	
AlignMiddleBtn	
AlignBottomBtn	
MergeCellsBtn	
SplitCellsBtn	
AlignLeftBtn	
AlignCenterBtn	
AlignRightBtn	
AlignJustifyBtn	
WrapTextBtn	
AutoSumBtn	Σ
InsertPictureBtn	
EditFormulaBtn	<i>fx</i>
SuppressDuplicatesBtn	
CrossTabWizardBtn	
LinkedReportBtn	
LinkedActionBtn	

The return value of the embedded javascript determines whether the result of clicking the button should take place. Return **true** to cause the button press to be halted, **false** to continue as normal.

Example

```
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 Reports UI.

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. E.g. 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 **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.

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 UI 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 New Report Wizard 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.

How to Inspect Session Data and Debug Extensions

Using a debugger to inspect session data is a good way to learn how to interact and extend application functionality. This guide explains how to inspect session state for events which allow you to insert custom event handlers. The Visual Studio debugger and Google Chrome javascript console are used in this demonstration, but other tools can be used as well.

Caution: These methods cause program halting, and are therefore not for use in a production environment.

Part 1 explains how to inspect server data for use with **Server Events** and **Custom Functions**.

Part 2 explains how to inspect client data for use with **Action Events**.

Server Data

You can inspect server data by calling out to an assembly with a *debugger* statement in a Server Event or Custom Function. By attaching the process to a debugger you can see the object variables currently in use.

First, build an assembly that has a public method call to launch the debugger. This can be accomplished in only a few lines of code. The method should pass in the *sessionInfo* object as an argument.

Note: The assembly needs a reference to the *WebReportsApi.dll* file in the {Exago}\bin folder.

Example

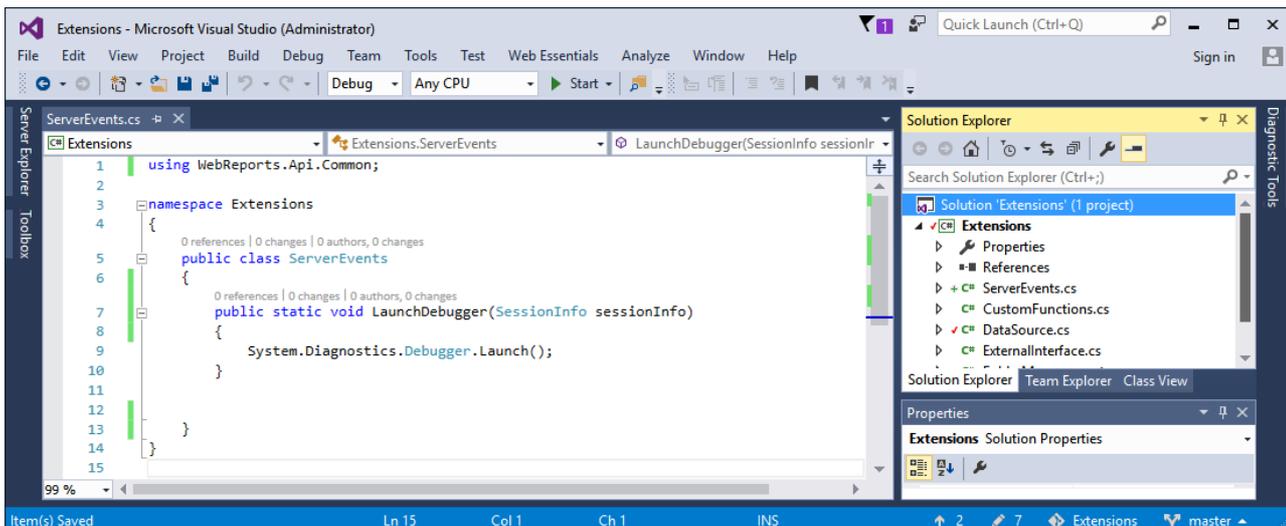
```
namespace Extensions
{
    public class ServerEvents
    {
        public static void LaunchDebugger(WebReports.Api.Common.SessionInfo sessionInfo)
        {
            System.Diagnostics.Debugger.Launch();
        }
    }
}
```

You could also use an object array argument to make it easier to pass in any other session variables you want to inspect.

Example

```
public static void LaunchDebugger(params object[] args)
```

Compile the program as a .dll assembly in Debug mode. Then copy the .dll to the {Exago}\bin folder. Keep Visual Studio open in the background so that the debugger is recognized.



Code to launch the Visual Studio debugger

Next, open the Exago Administration Console and add a new Server Event or Custom Function. For a Server Event, set the Global Event type to the application state that you want to inspect.

Inspect SessionInfo ✕

Name

Function

Global Event

Setting the event to fire at the start of a report execution

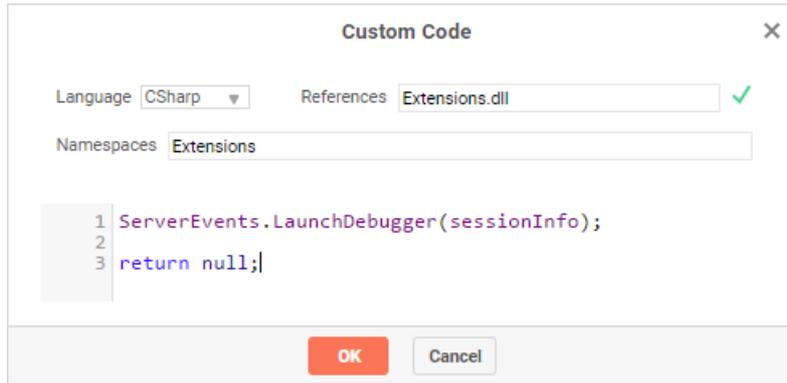
In the custom code window, add your .dll as a reference, and add the namespace if necessary. Then in the code field, call the debugger method, passing in the *sessionInfo* object and any other variables relevant to the session state.

Note: Make sure to use the correct return value for the Global Event.

Example

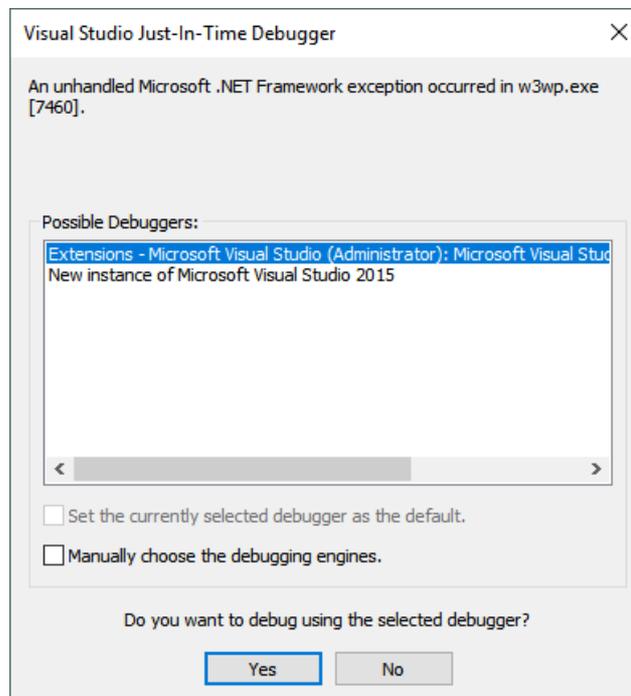
```
ServerEvents.LaunchDebugger(sessionInfo);
return null;
```

Click the green check mark to verify that the code is valid. Then click **OK** to save the event.



Calling the debugger method from the Server Event

Next, launch the Exago interface and cause the Server Event or Custom Function to fire. An exception will occur and Visual Studio will ask you to debug the process. Select the instance of Visual Studio with your assembly code and click **Yes**.



Visual Studio exception handler

Finally, if an Attach Security Warning appears, click **Attach**.



Attach the debugger to the Exago process

The Visual Studio debugging and diagnostics tools will launch for this program instance. There are many tools available, but for now focus on the **Autos** window. This shows all of the variables in use for the application at the moment. Any arguments passed into the debugger

method, including *sessionInfo*, are available to inspect. This is a great way to see how variables are used at different points in the application, and to learn about how the API is structured.

The screenshot shows the Visual Studio IDE with the following components:

- Code Editor:** Displays the file `ServerEvents.cs` with the following code:


```

1 using WebReports.Api.Common;
2
3 namespace Extensions
4 {
5     public class ServerEvents
6     {
7         public static void LaunchDebugger(SessionInfo sessionInfo)
8         {
9             System.Diagnostics.Debugger.Launch();
10        }
11    }
12 }
13
14 
```
- Autos Window:** Shows the state of the `sessionInfo` object. The table below represents the data shown in the window.

Name	Value	Type
sessionInfo	{WebReports.Api.Common.SessionInfo}	WebRep
CompanyId	"Alex"	string
CompositeReport	null	WebRep
CurQuarter	1	int
JavascriptAction	{WebReports.Api.Programmability.JavascriptAction}	WebRep
PageInfo	{WebReports.Api.Common.PageInfo}	WebRep
Report	{WebReports.Api.Reports.Report}	WebRep
ReportObject	{WebReports.Api.Reports.Report}	WebRep
ReportSchedulerService	'sessionInfo.ReportSchedulerService' threw an exception of type 'System.Exceptio	WebRep
SetupData	{WebReports.Api.Common.SetupData}	WebRep
Today	{2017-06-27 00:00:00}	System.I
UserId	'sessionInfo.UserId' threw an exception of type 'System.NullReferenceException'	string (S)
VisualizationReport	null	WebRep
Static members		
Non-Public members		

Inspecting sessionInfo data

Click the **Continue** or **Stop** buttons to exit the debugger when you are finished.

Client Data

You can inspect client data by inserting a debugger statement into the page's javascript using an Action Event. This lets you use your browser's javascript console to inspect the client objects, such as application DOM and report output data. Unlike Server Data, debugging client data requires no external software.

The javascript code can be implemented in an assembly, imported into the Exago home page, or written directly in the Administration Console. This example uses an assembly.

First, use the proper method formatting to add an Action Event which simply calls the javascript debugger. The encapsulating C# method must pass in the *sessionInfo* object, and return *sessionInfo.JavascriptAction* which contains the client-side javascript.

Example

```

using WebReports.Api.Common;
using WebReports.Api.Programmability;

namespace Extensions
{
    public class ActionEvents
    {
        public static JavascriptAction LaunchConsole(SessionInfo sessionInfo)
        {
            sessionInfo.JavascriptAction.SetJsCode("(function() { debugger; })()");
            return sessionInfo.JavascriptAction;
        }
    }
}

```

Note: Using an anonymous function wrapper is not required for all action events, but it is more broadly compatible, and thus recommended.

```
using WebReports.Api.Common;
using WebReports.Api.Programmability;

namespace Extensions
{
    public class ActionEvents
    {
        public static JavascriptAction LaunchConsole(SessionInfo sessionInfo)
        {
            sessionInfo.JavascriptAction.SetJsCode("(function() { debugger; })()");
            return sessionInfo.JavascriptAction;
        }
    }
}
```

The javascript code is the string argument of JavascriptAction.SetJsCode

If you are using an assembly, compile the program as a .dll, then copy it to the {Exago}\bin folder. If your javascript is in a separate .js file, copy that as well.

Next, open the Exago Administration Console and add a new Action Event. Set the Event Type to Click or Load if you want to attach the event to a specific occurrence in the Report Designer or on a Report. Set the Global Event Type if you want to attach the event to a general application action.

"Click" events can be attached to report items or buttons in the Report Designer

In the custom code window, add the assembly .dll as a reference. In the code field, return the Action Event method, passing in *sessionInfo* as an argument.

Example

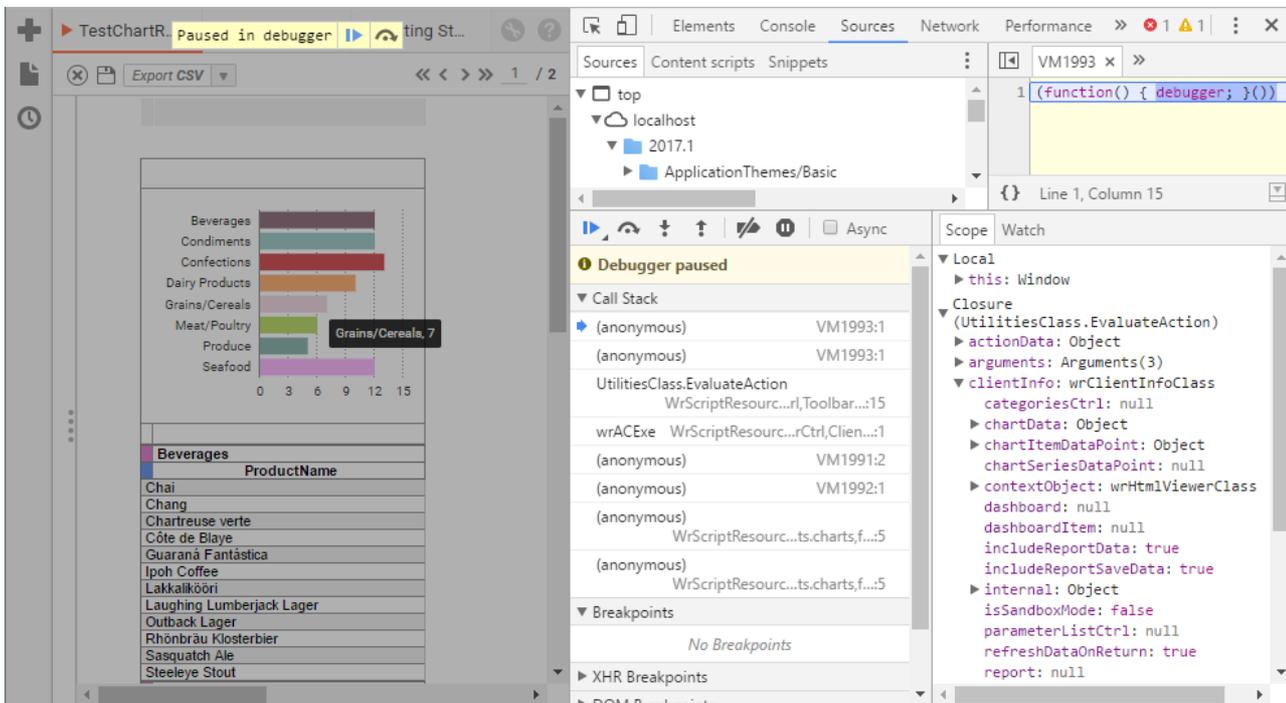
```
return ActionEvents.LaunchConsole(sessionInfo);
```

Click the green check mark to verify that the code is valid. Then click **OK** to save the event.

Calling an Action Event in an assembly

Next, launch the Exago UI. If you are using a *Local* action event, with a type Click or Load, then attach the event to an item in the report designer or on a report.

Press F12 to open the browser's Developer Tools. Finally, cause the Action Event to fire. The browser will load the object state in the debugger. Click the **Sources** tab and focus on the right-most pane. Expand the Scope pane to see the client object data. The **clientInfo** object contains the data most relevant to the application session. You can see the current state variables, and you can browse the structure of the client data to learn how to develop Action Events for your Exago environment.



Examining clientInfo variables

Press F8 to continue or F12 to close the debugger when you are done.

Introduction to the .NET API

The Exago interface consists of two 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 can interface with the **SOAP Web Service API** or the **REST Web Service API**, which offer subsets of the .NET API.

This guide will walk through the process of integrating Exago into a .NET-based web application. We will demonstrate how to use the API to connect with an existing installation of the Exago Application, and showcase how to do some basic report management tasks. For a full list of classes see the **Technical Guide**.

For this example, we will load an existing report, modify its run-time filters, and then execute the report in a browser window.

Referencing the Api

First reference the Web Reports API library. This is usually located in your **Installation Directory > bin** folder, and is called **WebReportsApi.dll**. This library contains all the namespaces necessary to utilize the Web Application's features.

Include the following namespaces for now:

- **WebReports.Api**: Contains the Api class which is the main interaction class between Exago and your application.
- **WebReports.Api.Reports**: Contains the Report class, for creating and managing reports.
- **WebReports.Api.Roles**: Contains classes for managing role security settings.

Creating an Api Object

An Api Object is the main class you need to create in order to interact with Exago. It contains the first and last points of entry for each instance of your application.

Create an Api Object using the following overloaded constructor:

```
Api myApi = new Api(AppPath);
```

AppPath is the physical or virtual location of your Exago installation.

NOTE: You can use the ConfigFile parameter to load an Exago Config file. If left out, it will default to WebReports.xml.

NOTE: Do not call the constructor method with empty parameters.

Loading a Report

Now let's load the report we want to modify.

```
Report myReport = (Report)myApi.ReportObjectFactory.LoadFromRepository(@"TestReport");
```

NOTE: You do not need to specify the file extension. The application will not allow reports of the same name but different types.

The `ReportObjectFactory` class is the collection class for report management methods. This contains methods for creating, loading, copying, saving, renaming, and deleting report files. It can be used to work within a repository or a temporary browser session, or both if necessary. In this case, we're simply loading a pre-existing report into the Api.

NOTE: Several methods reference an *Active* report by default. This is usually the last report accessed, but can be specified by the `Active` property of the `ReportObjectFactory`.

Reports called from within the `ReportObjectFactory` are `ReportObject` objects, which is a general class representing the various report types.

We are casting the `ReportObject` called from the factory to a `Report`, because we know it is a report. `Report` inherits several methods from `ReportObject`, but greatly expands the management capabilities.

Note: If we loaded a dashboard report or a chained report, we would cast it to `DashboardReport` or `ChainedReport` respectively.

See [Loading Reports in the .NET API](#) for more information.

Retrieving the Role Security

Let's see if the current user has access to the database object we want to filter for.

```
if (myApi.Roles.ActiveRole.Security.DataObjects.GetDataObject("Customers") != null)
{
    Permission Granted!;
}
else
    Permission Denied!;
```

The `Roles` object allows for accessing and modifying the elements found within the Admin Console. Changing or adding Role settings will only persist for the current session, and will not modify the config file.

We can also set some additional security and restrict the user from seeing specific data object rows:

```
DataObjectRow dataObjectRow = myApi.Roles.ActiveRole.Security.DataObjectRows.NewDataObjectRow();
dataObjectRow.ObjectName = "SSN";
dataObjectRow.FilterString = "false";
```

Modifying the Report

Now that we have loaded and declared our `Report` object, we can begin to modify its contents. Let's clear out the pre-existing runtime filters, and add one of our own:

```
foreach (Filter oldFilter in myReport.Filters)
{
    myReport.Filters.Remove(oldFilter);
}

Filter myFilter = myReport.Filters.NewFilter();
myFilter.DbName = "Customers.CompanyName";
myFilter.Operator = wrFilterOperator.NotEqualTo;
myFilter.Value = "Exago Inc.";
myFilter.AndOrWithNext = wrFilterAndOrWithNext.And;
```

Note that all these modifications will take place within the current session, and will not modify the report data unless you specifically overwrite the report file.

Closing the Session and Executing

That's enough for now, Let's execute the altered report. First, some administrative stuff.

Set the report execute type to HTML, so it runs in the browser:

```
myReport.ExportType = wrExportType.Html;
```

And hide the application tabs to avoid clutter:

```
myApi.ShowTabs = false;
```

Now save the changes we've made to the Api. This will load the report into a temporary space in the Api to prepare for execution.

```
myApi.ReportObjectFactory.SaveToApi(myReport);
```

And we're all set! Calling the following method will end the session, performing the specified action, and return the session URL. The session URL is an alphanumeric single-use identifier for the session. Append it to your Exago application URL.

Since we loaded a report into the Api, the action defaults to **Execute**.

```
string url = @"//MyDomainServer/Exago/" + myApi.GetUrlParamString(ExagoHome);
```

The ExagoHome parameter can be used to specify the Exago application's home page. It will default to ExagoHome.aspx.

Now redirect your browser to the generated URL and see the results!

Load Reports in the .NET API

The ReportObject Class

All Report type objects inherit from `ReportObject`. The class includes basic functions such as loading and saving the objects. Report type objects consist of *simple* report types such as express and standard, and *composite* types such as dashboard and chained.

Accessing Reports via the API

A Report is normally loaded via API in order to launch directly into the *execution* or *editing* of the report. In most cases, report loading is handled by the Exago runtime via the `LoadFromRepository` call. `LoadFromRepository` returns an object of type `ReportObject`. In order for the load to complete, a corresponding `SaveToApi` is done. `SaveToApi` will save the `ReportObject`, including any modifications back to the API for loading (for execution or editing). Both `LoadFromRepository` and `SaveToApi` can be accessed via the `Api.ReportObjectFactory` property.

Both `LoadFromRepository` and `SaveToApi` return and store a `ReportObject` respectively. Most modifications made to a report loaded at runtime (such as changing filter values) are performed on properties defined within the child types (such as `Report` or `DashboardReport`). Such modifications would require a cast of the `ReportObject` to the appropriate child type.

If no modifications are required, the sequence is simple:

```
//Load the report.  refers to the full folder path
//followed by the report name
ReportObject ro = api.ReportObjectFactory.LoadFromRepository();
//Save the report back to the API
api.ReportObjectFactory.SaveToApi(ro);
```

If modifications are desired, it is necessary to cast the value returned by `LoadFromRepository` to the appropriate base type. Supported base types include `WebReports.Api.Reports.Report` for standard, express, and crosstab reports, `WebReports.Api.Composite.Dashboard.DashboardReport` for dashboards, and `WebReports.Api.Composite.Chained.ChainedReport` for chained reports.

For example, you can modify Filters on a report to be executed as follows:

```
//Load the report.  refers to the full folder path
//followed by the report name
Report report = api.ReportObjectFactory.LoadFromRepository() as Report;
report.Filters[0].Value = ;
//Save the report back to the API
api.ReportObjectFactory.SaveToApi(report);
```

Note that the filter change will only apply to this execution of the report. The report design itself will not be modified.

Note also that all elements of the `Report` object (as opposed to the `ReportObject` object!) can be modified in this manner. It is also possible to build a report from scratch in the API. There are examples in production of clients that have programmatically built their own reports.

.NET API General Reference

This article contains a list of examples for using the .NET API's various features. The .NET API can be used for many purposes, and the most common is to create secured access points for end-users via browser sessions. However, you can also use the API to create and manage schedules, generate config files and reports programmatically, edit existing reports, and more.

Note: This guide is kept up to date for the most current version available. For a list of changes in the API, see the **Updating Guide**.

Getting Started

Exago BI .NET applications must include a reference to the file **WebReportsApi.dll**, found in the bin folder of the application directory. This contains all the classes and method groups necessary for the examples in this article.

Optional: If you are connecting to a .NET Assembly Data Source, you will need to include a reference to **WebReportsAsmi.dll**. A reference to **WebReports.dll** is unnecessary, except in some rare cases that are not discussed in this guide.

The following Exago BI namespaces are used in the examples in this guide:

- WebReports.Api
- WebReports.Api.Reports
- WebReports.Api.Common
- WebReports.Api.Data
- WebReports.Api.Roles
- WebReports.Api.Programmability
- WebReports.Api.Scheduler
- WebReports.Api.ReportMgmt
- WebReports.Api.Theme

As well as the following system namespaces:

- System.Collections.Generic
- System.Linq
- System.Xml

Contents

- **API Object**
- **Sorts and Filters**
- **Settings**
- **Role Permissions**
- **Advanced Configuration**
- **Scheduling**
- **Managing Files and Folders**

API Object

An API object must be instantiated at the start of an API application, as the access point for all API activity. API objects also encompass sessions which are used to encapsulate user-specific changes, such as security and permissions settings.

Note: Variable names, such as "api" and "report", are declared in their respective sections in this guide, then referenced throughout the remainder.

Constructors

Basic constructor: appPath is the file path or virtual path to Exago base install.

```
Api api = new Api("appPath");
```

Specify a config file (other than the default WebReports.xml)

```
Api api = new Api("appPath", "configFn");
```

Custom config file and its location in Azure storage (must match web.config)

```
Api api = new Api("appPath", "configFn", "azurePath");
```

Let the host application configure log4net

```
Api api = new Api("appPath", bool isLogCustomConfig, "configFn");
```

Write to the default log file (v2018.1+: [Temp]\WebReportsApiLog.txt)

```
Logger logger = Log.GetLogger();  
logger.Info("message"); // writes at info level
```

API Action

The location to direct the Exago interface when first loaded. Edit or Execute actions work on "active" report.

```
api.Action = wrApiAction.Home;
```

Hide tabbed UI

```
api.ShowTabs = false;
```

Active Report

Load a report from repository. Makes it the active report. "reportName" is the fully qualified path relative to the base report directory, without file extension. ReportObject is a generic class encompassing all report types.

```
ReportObject reportObject = api.ReportObjectFactory.LoadFromRepository("reportName");
```

Report is validated (checked for errors) unless overridden

```
ReportObject reportObject = api.ReportObjectFactory.LoadFromRepository("reportName", bool validate);
```

Manually validate

```
reportObject.Validate();
```

Check the errors list. See [list of error types](#).

```
foreach (ReportValidationError error in report.ValidationErrors)
{
    string.Format("Error: {0}\n\t{1}\n\t{2}\n",
        error.ReportErrorType.ToString(),
        error.Data1,
        error.Data2);
}
```

Get the report type

```
string reportType = reportObject.ReportType.ToString();
```

Cast the report object to higher level class

```
Report report = (Report)reportObject;
```

Validate the export type

```
bool IsReportAllowPdfExport = report.IsExportTypeAllowed(wrExportType.Pdf);
bool IsConfigAllowPdfExport = api.General.AllowPdfOutput;
```

Set the export type

```
report.ExportType = wrExportType.Pdf;
```

Launch Exago and Execute Report

To launch in a browser frame with specified Action, get the App URL to redirect the browser. Redirecting to the App URL closes the API to further changes. It should be the last thing done.

Get the App URL, with the default home page "ExagoHome.aspx"

```
string appUrl = api.GetUrlParamString();
```

Specify a custom Exago home page. "homePage" is the file name without the file extension.

```
string appUrl = api.GetUrlParamString("homePage");
```

Set ShowErrorDetail to true, for more detailed user error messages

```
string appUrl = api.GetUrlParamString(null, true);
// this is equivalent to:
string appUrl = api.GetUrlParamString() + "?ShowErrorDetail=true";
```

Redirect the browser to the App URL. This will close the API to any further changes.

```
Frame.Src = baseUrl + appUrl;
```

GetExecute

As an alternative to Api Action, reports can be executed and the data returned directly to the application. This will not close the API. Since this does not launch Exago, report interactivity is not supported, only bare HTML, JSON, and so on. Acts on the specified report, not the active report.

```
string reportHtml = report.GetExecuteHtml();
string reportJson = report.GetExecuteJson();
byte[] reportData = report.GetExecuteData();
string[] reportSql = report.GetExecuteSql();
```

Sorts and Filters

You can add sorts and filters to reports at runtime. After making changes, save back to the API for execution in the session.

```
api.ReportObjectFactory.SaveToApi(report);
```

Changes could be saved back to the report file on disk

```
api.ReportObjectFactory.SaveToRepository(report);
```

Save edited report as a new report

```
api.ReportObjectFactory.Copy(report, "newName", null);
```

Sorts

See the existing sorts in a report

```
foreach (Sort sort in report.Sorts)
{
    string.Format("{0}, {1}\n", sort.SortText, sort.Direction);
}
```

Add a new sort to a report

```
report.Sorts.Add(new Sort(api.PageInfo)
{
    // use object Alias name, and field Alias name if set
    SortText = objectName + '.' + fieldName,
    Direction = wrSortDirection.Ascending
});
```

Formula sort

```
SortText = string.Format("=Formula({{0}.{1}})", objectName, fieldName);
```

Sorts is an ordered list. Reorder the list to change the precedence of the sorts.

```
// move a sort from position 3 to position 1
for (int i = 3; i > 1; i--)
{
    Sort temp = report.Sorts[i - 1];
    report.Sorts[i - 1] = report.Sorts[i];
    report.Sorts[i] = temp;
}
```

Filters

See the existing filters in a report

```
string filterString = ""; // build a filter summary string for display
foreach (Filter f in report.Filters)
{
    filterString += string.Format("{0}{1} {2} {3}{4}",
        new string(' ', f.GroupStartCnt),
        f.Name,
        f.DisplayOperatorText,
        f.DisplayValue,
        f.GroupEndCnt > 0 ? new string(' ', f.GroupEndCnt) : ' ' + f.AndOrValue + ' ');
}
}
```

Add a new filter

```
// get the data field info (should first check that the report contains the entity)
EntityColumn field = report.Entities.GetEntity("objectName").GetColumn("fieldName");

report.Filters.Add(new Filter(api.PageInfo)
{
    Name      = field.FullName, // object and field Alias names
    DataType  = field.DataType,
    Operator  = wrFilterOperator.OneOf, // this filter type takes multiple values
    DataValues = new DataValueCollection()
    {
        new DataValue(api.PageInfo, field) { Value = "value1" },
        new DataValue(api.PageInfo, field) { Value = "value2" }
    }
});
```

Group Min/Max filters

```
string groupFilterString = ""; // build a filter summary string for display
int i = 0;
foreach (GroupFilter filter in report.GroupFilters)
{
    groupFilterString += string.Format("{0} {1} for each {2} {3}{4}",
        filter.DisplayOperatorText,
        filter.Name,
        filter.GroupName,
        filter.IgnoreOtherGroups ? "ignoring other groupings" : "",
        (i < report.GroupFilters.Count()) ? ", " : "" );
    i++;
}
}
```

Add a new group filter

```
// should check that the entity and sorts exist on the report
EntityColumn field = report.Entities.GetEntity("objectName").GetColumn("fieldName");
Sort sort = report.Sorts.GetSort("sortName");

report.GroupFilters.Add(new GroupFilter(api.PageInfo)
{
    Name      = field.FullName, // object and field Alias names
    Operator  = wrGroupFilterOperator.Maximum,
    GroupName = sort.SortText // or sort.Entities[0].Name, or "EntireDataSet"
});
```

See Top N filter

```
// build a filter summary string for display
// reports limited to one Top N filter, max one foreach group per filter
string topNFilterString = "";
if (report.TopNItems.Count() > 0)
{
    if (report.TopNItems[0].UseTopNItem)
    {
        TopNItem filter = report.TopNItems[0];
        topNFilterString = string.Format("{0} {1} {2}{3}",
            filter.Action == TopNAction.top ? "Top" : "Bottom",
            filter.Number,
            report.Cells.GetCellById(filter.CellId).DisplayText,
            (filter.ForEachGroup.Count() > 0) ? " for each " + filter.ForEachGroup[0] : "");
    }
}
```

Add Top N filter

```
// check that the cell exists
int cellId = report.Cells.GetCell(int row, int col).Id;

TopNItem topN = new TopNItem(api.PageInfo)
{
    Action      = TopNAction.top,
    Number      = int N, // the N in "Top N"
    CellId      = cellId,
    ForEachGroup = new List<string> { },
    UseTopNItem = true
};

// must be at zero-index position
if (report.TopNItems.Count() == 0)
    report.TopNItems.Add(topN);
else
    report.TopNItems[0] = topN;
```

Settings

Change the values of config settings and parameters dynamically, which often depend on user access rights. A dynamic config can be thought of as an extension of a Role, although Roles aren't necessarily required.

Change config setting

```
api.General.anySetting = newValue;
```

See [Config File and API Setting Reference](#) for the full list of config settings.

Parameters

Parameters are "buckets" for values that persist throughout a session, and are reachable by extensions. You can use them to set custom environment variables. **userId** and **companyId** are built-in parameters for storing user information.

Caution: The .NET namespace `System.Web.UI.WebControls` also contains a class called `Parameter`. You may wish to alias one or both classes to resolve name conflicts.

```
using wrParameter = WebReports.Api.Common.Parameter;
```

List config parameters

```
foreach (Parameter parameter in api.Parameters)
{
    string.Format("Name: {0}, Value: {1}\n", parameter.Id, parameter.Value);
}
```

Get a specific parameter by Id

```
Parameter parameter = api.Parameters.GetParameter("parameterId");
```

Add a new parameter

```
api.Parameters.Add(new Parameter(api.PageInfo)
{
    Id      = "foo", // no spaces
    DataType = (int)DataType.String,
    Value   = "bar"
});
```

Non-hidden parameters are usable on reports

```
parameter.IsHidden = false;
```

Prompting parameters will ask the user for a value when running a containing report

```
parameter.PromptText = "Enter a value";
```

Prompting parameters can display a selection list based on a data field or custom SQL

```
Entity entity = api.Entities.GetEntity("entityName");

parameter DropdownDataSourceId      = entity.DataSourceId;
parameter DropdownObjectType       = entity.ObjectType;
parameter DropdownDbName           = entity.DbName;
parameter DropdownValueField       = entity.GetColumn("colName").Name;
parameter DropdownDisplayValueField = entity.GetColumn("colName").Name;
```

Save to Disk

Creates .xml and encrypted .enc files in the Temp directory. If `isPermanent = true`, creates the files in the Config directory, overwriting the existing config files.

```
api.SaveData(bool isPermanent);
```

Role Permissions

Roles are a neat way to encapsulate a collection of permissions. Roles also allow for some more fine grained control over data and folder access than the base config settings. Only one role can be active at a time.

Get a specific role

```
Role role = api.Roles.GetRole("roleId");
```

Create new role

```
api.Roles.Add(new Role(api.PageInfo) { Id = "roleId", IsActive = true });
```

Edit role settings

```
role.General.AnySetting = "value";
```

Folder/Object/Row Security

```
role.Security.Folders.IncludeAll = true;
role.Security.Folders.Add(new Folder() { Name = "folderPath", ReadOnly = true });
role.Security.DataObjects.Add(new DataObject() { Name = "objectName" });
role.Security.DataObjectRows.Add(new DataObjectRow()
{
    ObjectName = "objectName",
    FilterString = "filterString"
});
```

Activate a role

```
role.Activate();
```

Advanced Configuration

You can dynamically change data sources, objects, joins, etc., in the API, or simply use these settings to programmatically generate a config file.

Data Sources

View data source

```
api.DataSources.GetDataSource("dataSourceName").DataConnStr;
```

Create a new data source

```
api.DataSources.Add(new DataSource(api.PageInfo)
{
    Name           = "dataSourceName",
    DbType         = Constants.DatabaseType.SqlServer,
    DataConnStr    = "connectionString"
});
```

Data Objects

View data objects

```
foreach (Entity entity in api.Entities)
{
    // three ways to identify an entity
    string.Format("Alias: {0}, Id: {1}, Database Name: {2}\n",
        entity.Name, // Alias
        entity.Id,   // Id
        entity.DbName // Database Name
    );
}
```

Get a specific data object

Note: This is the recommended way to get an entity by name, but there are other methods provided as well

```
Entity entity = api.Entities.GetEntity("entityAlias"); // returns null if it does not exist
// best to retrieve entities by Alias name, because Aliases are unique and required
// Ids are unique, but not required; Db names are required, but not unique (across sources)
```

View object fields

```
foreach (EntityColumn column in entity.Columns)
{
    string.Format("Alias: {0}, Database name: {1}\n",
        column.Name, // Alias (or actual if no alias)
        column.ActualName // Database name
    );
}
```

Create new data object

```
Entity entity = api.Entities.NewEntity();

entity.DataSourceId = api.DataSources.GetDataSource("dataSourceName").Id;
entity.ObjectType   = DataObjectType.Table;
entity.DbName       = "databaseName"; // required
entity.Name         = "aliasName";    // required, unique
entity.Id           = "idName";       // unique

// add key column
entity.KeyColumns.Add(new KeyColumn(entity.GetColumn("colName").ActualFullName));
```

SQL Object

```
entity.SqlStmt = "SELECT * FROM Employees";
```

Add tenanting to object

```
entity.Tenants.Add(new EntityTenant(api.PageInfo,
    entity.Name,
    entity.GetColumn("colName").ActualFullName, //get col by Alias but supply ActualFullName
    "parameterId" //tenant parameter
));
```

Filter dropdowns

```
entity.FilterObjectType = DataObjectType.Table; // Table, View, Function, Procedure, etc.
entity.FilterDbName = "FilterObjectName";
// or custom SQL:
entity.FilterObjectType = DataObjectType.SqlStmt;
entity.FilterSqlStmt = "SELECT etc...";
```

Joins

See all config joins

```
// build the join string for display
string joinString = "";

// all config joins; for report joins, use report.Joins
foreach (Join join in api.Joins.OrderByDescending(x => x.Weight)) // order by weight
{
    foreach (JoinColumn col in join.JoinColumns)
    {
        joinString += string.Format("{0} {1} {2}{3}\n",
            col.FromColumn.FullKeyName,
            join.JoinText,
            col.ToColumn.FullKeyName,
            join.RelationType == 1 ? "(s)" : "" // 1-to-Many
        );
    }
    if (join.Weight > 0) joinString += ("Weight: " + join.Weight + "\n");
}
```

Get specific join

```
Join join = api.Joins.GetItem("fromEntity", "toEntity", false);
```

Create a new join

Note: For creation of Advanced Joins, see [Advanced Joins](#).

```
Entity fromEntity = api.Entities.GetEntity("fromName");
Entity toEntity = api.Entities.GetEntity("toName");

Join newJoin = new Join(api.PageInfo)
{
    EntityFromName = fromEntity.Name,
    EntityToName    = toEntity.Name,
    Type            = (int)JoinType.Inner,
    RelationType    = 0, // 0: One-to-One, 1: One-to-Many
    Weight          = 0
};

// add the key columns (legacy version, pre-v2017.2)
newJoin.JoinColumns.FromColumns.Add(
    new KeyColumn(fromEntity.GetColumn("fromColName").ActualFullName));
newJoin.JoinColumns.ToColumns.Add(
    new KeyColumn(toEntity.GetColumn("toColName").ActualFullName));

// add the key columns (v2017.2+)
newJoin.JoinColumns.Add(new JoinColumn(
    new KeyColumn(fromEntity.Name, fromEntity.GetColumn("fromColName").ActualFullName),
    new KeyColumn(toEntity.Name, toEntity.GetColumn("toColName").ActualFullName)
));

// add to the config. for reports, use report.Joins.Add()
api.Joins.Add(newJoin);

// if there is an active report, recreate the joins
report.CreateJoins();
```

Custom Functions

Get a specific function

```
UdfFunction function = api.CustomFunctions.GetItem("functionName");
```

Create a new function

```
UdfFunction newFunction = new UdfFunction(api.PageInfo)
{
    Name          = "functionName",
    AvailableIn   = UdfFunctionAvailableType.Formula, // custom or filter function
    //MinArgs = 0, (deprecated in v2017.2)
    //MaxArgs = 0, (deprecated in v2017.2)
    ArgumentsJson = "[{'Name':'argName','Required':true,'Description':'desc'}]", // (v2017.2+)
    Language      = CodeLanguage.CSharp.ToString(),
    ProgramCode   = "Code()";
};

// add any references and namespaces
newFunction.Namespaces.Add("Program.Namespace");
newFunction.References.Add("Reference.dll");

// add to the config
api.CustomFunctions.Add(newFunction);
```

Server Events

Get a specific server event

```
ServerEvent serverEvent = api.ServerEvents.GetByName("eventName");
```

Create a new server event

```
ServerEvent newEvent = new ServerEvent(api.PageInfo)
{
    Name          = "eventName",
    EventType     = ServerEventType.OnReportExecuteStart, // global event, or "None"
};

// custom code
newEvent.ServerCode.CustomCode.Language      = CodeLanguage.CSharp;
newEvent.ServerCode.CustomCode.ProgramCode = "Code()";

// code from data source
newEvent.ServerCode.DataSourceId = api.DataSources.GetDataSource("eventsAssembly").Id;
newEvent.ServerCode.FunctionName = "functionName";

// add to config
api.ServerEvents.Add(newEvent);

// add to a report (must be in config)
report.ServerEvents.Add(new ReportServerEvent(api.PageInfo)
{
    EventType = ServerEventType.OnDataCombined,
    EventId   = api.ServerEvents.GetByName("eventName").Id
});
```

Scheduling

View schedule list

```

List<Exception> exceptions;

// build the job list string
string jobList = "";

foreach (List<JobInfo> schedule in api.ReportScheduler.GetJobList(out exceptions))
{
    foreach (JobInfo job in schedule.OrderBy(x => x.NextExecuteDate).ThenBy(x => x.Name))
    {
        jobList += string.Format("Job '{0}' for report '{1}' ",
            job.Name,
            api.ReportScheduler.GetReportScheduleInfoByJobId(job.JobId.ToString()).ReportBaseName
        );

        switch (job.Status)
        {
            case JobStatus.Completed:
                jobList += string.Format("ran on {0}, at host {1}.\n",
                    job.LastExecuteDate.ToString("MMM d hh:mm tt"),
                    api.ReportScheduler.GetHost(api.ReportScheduler.GetHostIdxForJob(job.JobId))
                );
                break;
            case JobStatus.Ready:
                jobList += string.Format("ready to run on {0}.\n",
                    job.NextExecuteDate.ToString("MMM d hh:mm tt")
                );
                break;
            case JobStatus.Deleted:
            case JobStatus.Removed:
            case JobStatus.Abanded:
            case JobStatus.UserAbort:
                jobList += string.Format("ended. Last run on {0}, at host {1}.\n",
                    job.LastExecuteDate.ToString("MMM d hh:mm tt"),
                    api.ReportScheduler.GetHost(api.ReportScheduler.GetHostIdxForJob(job.JobId))
                );
                break;
            default:
                jobList += string.Format("status unknown.\n");
                break;
        }
    }
}

```

Create an immediate schedule (basic options)

```

// run-once, immediately, save to disk
string jobId; // use to retrieve schedule info later for editing
int hostIdx; // assigned execution host id

ReportScheduleInfo newSchedule = new ReportScheduleInfoOnce()
{
    ScheduleName      = "Immediate Schedule",           // schedule name
    ReportName        = @"Report\Full\Path",           // report path
    ReportType        = wrReportType.Advanced,         // report type
    RangeStartDate    = DateTime.Today,                // start date
    ScheduleTime      = new TimeSpan(DateTime.Now.Ticks), // start time
    SendReportInEmail = false                           // email or save
};

// send to the scheduler; wrap in try/catch to handle exceptions
try {
    api.ReportScheduler.AddReport(
        new ReportSchedule(api.PageInfo) { ScheduleInfo = newSchedule }, out jobId, out hostIdx);
}
catch (Exception) { }

```

Recurring schedules (additional options)

Daily

```

ReportScheduleInfo newSchedule = new ReportScheduleInfoDaily()
{
    ... // include basic options
    // range of recurrence, every N days, or every weekday
    DailyPattern = ReportScheduleInfo.DailyPatternType.EveryNDays,
    EveryNDays   = 2, // N days

    // end date (optional):
    RangeEndDate   = DateTime.Parse("December 25 2017"), // end on a specific date
    RangeNOccurrences = 10,                               // or end after N occurrences

    // intraday recurrence (optional):
    RepeatEvery     = true, // enable intraday recurrence
    RepeatEveryHours = 4,   // repeat every N hours
    RepeatEveryMinutes = 0, // and N minutes
    RepeatEveryEndTime = new TimeSpan(DateTime.Parse("12:00 PM").Ticks) // optional end time
}

```

Weekly

```

ReportScheduleInfo newSchedule = new ReportScheduleInfoWeekly()
{
    ... // include basic options
    ... // optional end date, optional intraday recurrence
    EveryNWeeks = 1, // every N weeks
    // on these days:      Sun   Mon   Tues   Wed   Thurs   Fri   Sat
    IsDayOfWeek = new bool[] { false, true, false, false, true, false, false }
}

```

Monthly

```

ReportScheduleInfo newSchedule = new ReportScheduleInfoMonthly()
{
    ... // include basic options
    ... // optional end date, optional intraday recurrence
    // specific or relative day pattern
    MonthlyPattern = ReportScheduleInfo.MonthlyPatternType.SpecificDayOfMonth,

    // specific: day X of every N months
    SpecificDayOfMonth = 7, // day of the month
    SpecificEveryNMonths = 2, // every N months

    // relative: the Xth day-of-week of every N months
    RelativeWeekOfMonth = ReportScheduleInfo.WeekOfMonthType.First, // week of month
    RelativeDayOfWeek    = ReportScheduleInfo.DayOfWeekType.Weekday, // day of week
    RelativeEveryNMonths = 2 // every N months
}

```

Yearly

```

ReportScheduleInfo newSchedule = new ReportScheduleInfoYearly()
{
    ... // include basic options
    ... // optional end date, optional intraday recurrence
    // specific or relative day pattern
    YearlyPattern = ReportScheduleInfo.YearlyPatternType.SpecificDayOfYear,

    // specific: Month/Day of every year
    SpecificMonthOfYear = 3, // month of the year
    SpecificDayOfMonth  = 15, // day of the month

    // relative: the Xth day-of-week for month N
    RelativeWeekOfMonth = ReportScheduleInfo.WeekOfMonthType.Last, // week of month
    RelativeDayOfWeek    = ReportScheduleInfo.DayOfWeekType.Friday, // day of week
    RelativeMonthOfYear  = 3 // month N
}

```

Email job

```

newSchedule.SendReportInEmail = true;           // enable email
newSchedule.EmailSubject      = "Subject Text"; // email subject
newSchedule.EmailBody         = "Hello World!"; // email body
newSchedule.EmailToList.Add("email@company.com"); // to addresses
// newSchedule.EmailCcList.Add();           // cc addresses
// newSchedule.EmailBccList.Add();         // bcc addresses

```

Batch email

```

// batch addresses entity (must be in the batch report)
Entity batchAddresses = report.Entities.GetEntity("entityName");

newSchedule.IsBatchReport = true;           // enable batch
newSchedule.BatchEmailToList.Add("supervisor@example.com"); // summary recipients
// newSchedule.BatchEmailCcList.Add();     // summary cc recipients
newSchedule.BatchEntity = batchAddresses.Name; // email address object
newSchedule.BatchField = batchAddresses.GetColumn("Email").Name; // email address field
newSchedule.IncludeReportAttachment = true; // include the report

```

Access existing schedule by jobId

```
ReportScheduleInfo schedule = api.ReportScheduler.GetReportScheduleInfoByJobId("jobId");
```

Update an existing schedule

```
api.ReportScheduler.UpdateExistingSchedule(newSchedule, "jobIdToUpdate");
```

Delete an existing schedule

```
api.ReportScheduler.DeleteSchedulerJob("jobIdToDelete");
```

Managing Files and Folders

Initialize the manager class for the type of folder mgmt in use

```

// File System
ReportMgmtFileSystem manager = new ReportMgmtFileSystem(api.PageInfo);
// Database
ReportMgmtMethod manager = new ReportMgmtMethod(api.PageInfo);
// Cloud drive
ReportMgmtCloud manager = new ReportMgmtCloud(api.PageInfo);

```

View the full reports tree

```
XmlDocument tree = new XmlDocument() { InnerXml = manager.GetReportListXml() };
```

View themes list by type

```
List<string> evThemes = manager.GetThemeList(ReportTheme.ReportThemeType.ExpressView.ToString());
```

Get a specific theme (class depends on theme type)

```
ExpressViewTheme evTheme = (ExpressViewTheme)ReportTheme.GetTheme(
    api.PageInfo, ReportTheme.ReportThemeType.ExpressView, "themeName");
```

View list of templates

```
List<string> templates = manager.GetTemplateList();
```

Add a new folder

```
manager.AddFolder("parentFolder", "newFolderName");
```

Move or rename a folder

```
manager.RenameFolder("oldPath", "newPath");
```

Move or rename a report

```
manager.RenameReport("oldPath", "newPath");
```

Duplicate an existing report

```
api.ReportObjectFactory.Copy("reportName", "newName");
```

Save a new report to disk

```
api.ReportObjectFactory.Copy(report, "reportName", null);
```

Introduction to REST

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

To install REST, run the **Exago Installer**, then select the Exago Web Service API.



Choose a file path, IIS website, and a virtual path for the web service. The installer should automatically create an application in IIS for the web service.

Set the application pool in IIS. Then grant it Full Control permissions to the web service **Config** directory. See **Configuring IIS for Exago** for detailed instructions.

In the Web Service directory, edit **appSettings.config** and add the key *ExagoRest* with value *True*, to enable REST:

```
<add key="ExagoREST" value="True" />
```

Then, in the **Config** sub-directory, edit **WebReportsApi.xml** and set the `<apppath>` to the file path of your main Exago installation (the host application, not the web service). (See **Configuring Web Services** for more info about the config file).

Restart your web server and you should be ready to test your installation. See **Authentication** for options for password-protecting your web service.

See **Starting a REST Session** to get started using the REST API.

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 the **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**.

The 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 as **Advanced Rest Client** 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 username ("User ID") and "REST Key" found within the **Exago configuration file** currently being accessed.

Note. In versions prior to v2017.3, the "Password" field was utilized as the 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

```
QWxhZGRpbjpvOGVuIHNlc2FtZQ==
```

3. Append "Basic " to the front

```
Basic QWxhZGRpbjpvOGVuIHNlc2FtZQ==
```

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:6HZE5tCWjsjbJY+VXQg3UzX1K/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 wasn'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	Writeable	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 **Writeable** 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	Writeable	Description
DataType	enum	yes	Parameter Type

Properties which accept constant string values are indicated as having a "const" type. The description field in its JSON representation links to the list of possible values. For example:

Name	Type	Writeable	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.

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, 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==" ^
-X POST
```

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, "Show"
```

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':'Test\\TestReport'}}" ^
-H "Accept: application/json" ^
-H "Content-Type: application/json" ^
-H "Authorization: Basic Og==" ^
-X POST
```

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==" ^
-X POST
```

json.txt

```
"{'ReportSettings':{'ReportPath':'Test\\TestReport'}}"
```

List of REST Endpoints

The following REST endpoint paths are available. All calls require the following headers:

```
Content-Type: application/json
Accept: application/json
Authorization: {type} {authstring}
```

/rest/Sessions

- POST
- GET

/rest/Sessions/{sid}

- GET
- PATCH
- DELETE

/rest/Batch

(v2018.1.1+)

- POST

All following calls require URL parameter: sid={sid}

Example

```
GET /rest/Settings?sid={sid}
```

/rest/Settings

- GET
- PATCH

/rest/Parameters

- POST
- GET

/rest/Parameters/{Id}

- GET
- PATCH
- DELETE

/rest/Roles

- POST
- GET

/rest/Roles/{Id}

- GET
- PATCH

/rest/Roles/{Id}/Settings

- GET
- PATCH

/rest/Roles/{Id}/Folders

- GET
- PATCH

/rest/Roles/{Id}/Entities

- GET
- PATCH

/rest/Roles/{Id}/DataObjectRows

- GET
- PATCH

/rest/Folders/{Name}

- POST
- DELETE

/rest/Folders/Rename

- POST

/rest/Reports/List

- GET

/rest/Reports/Execute/{Type}

- POST

/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

/rest/Joins

- POST
- GET

/rest/Joins/{Id}

- GET
- PATCH

- DELETE

/rest/Functions

- POST
- GET

/rest/Functions/{Id}

- GET
- PATCH
- DELETE

/rest/ServerEvents

- GET

/rest/ServerEvents/{Id}

- GET
- DELETE

Executing Reports with the API

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. This only supports simple report types, Advanced, Express, and CrossTab Reports. 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.

Note: GetExecute was previously referred to as *Direct Execution*.

Note: GetExecute methods are not supported by **Remote Execution**.

Overview

	Launch Method	Supported Report Types	.NET Supported Output Types	SOAP 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 reports, ExpressViews	HTML, CSV, PDF, RTF, Excel, JSON	HTML, CSV, PDF, RTF, Excel	HTML, CSV, PDF, RTF, Excel, JSON

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.

NOTE. 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 UI. 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

First create an API session and load a report object.

```
// a .net api object is a specific session; methods act on that session
WebReports.Api.Api netApi = new WebReports.Api.Api(appPath);
Report myReport = (Report)netApi.ReportObjectFactory.LoadFromRepository(reportPath);
```

Set the desired output type.

```
myReport.ExportType = wrExportType.Html;
```

Then save the report object back to the API.

```
netApi.ReportObjectFactory.SaveToApi(myReport);
```

Set the API action to execute.

```
netApi.Action = wrApiAction.ExecuteReport;
```

Finally, call `GetUrlParamString` to get the session URL.

```
// note: this terminates the session
string url = netApi.GetUrlParamString(homePage);
```

REST

When using the REST 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.

To use REST, create the session and pass the session variables. Take note of the `sessionId` and `UrlParamString`.

POST /sessions

Payload:

```
{
  "Page": homePage,
  "ApiAction": "ExecuteReport",
  "ExportType": "Html",
  "ReportSettings":
  {
    "ReportPath": reportPath
  }
}
```

Response (*some params omitted*):

```
{
  "sid": sessionId,
  "AppUrl": urlParamString
}
```

SOAP

When using the SOAP API, the initialization call creates a session ID string, which is a required parameter in all subsequent method calls.

First create an API object and initialize an API session.

```
// a soap api object is not a specific session; it's an accessor for the methods
WebServiceApi.Api soapApi = new WebServiceApi.Api();
// initialize a session and save the id to memory
string sessionId = soapApi.InitializeApi();
```

Set the active report.

```
// all methods require the session id parameter
soapApi.ReportObject_Activate(sessionId, reportPath);
```

Set the API action to execute.

```
soapApi.SetAction(sessionId, (int)wrApiAction.ExecuteReport, null);
```

Finally, call `GetUrlParamString` to get the session URL.

```
string url = soapApi.GetUrlParamString(sessionId);
```

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, there is no need to call `GetUrlParamString`.

See the following sections for examples. Variable names and arguments are placeholders.

.NET

First create an API session and load a report object.

```
// a .net api object is a specific session; methods act on that session
WebReports.Api.Api netApi = new WebReports.Api.Api(appPath);
Report myReport = (Report)netApi.ReportObjectFactory.LoadFromRepository(reportPath);
```

Then call the appropriate `GetExecute` method for the desired data type.

```
string reportHtml = myReport.GetExecuteHtml();
```

REST

When using the REST 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.

First create an API session. Take note of the session ID.

POST /sessions

Response (*some params omitted*):

```
{
  "Id": sessionId
}
```

Then execute the selected report. Supported types are HTML, CSV, PDF, RTF, Excel, JSON.

POST /reports/execute/{type}?sid="sid"

Payload:

```
{
  "ReportPath": reportPath
}
```

Response (*some params omitted*):

```
{
  "ExecuteData": rawReportData
}
```

SOAP

When using the SOAP API, the initialization call creates a session ID string, which is a required parameter in all subsequent method calls.

First create an API object and initialize an API session.

```
// a soap api object is not a specific session; it's an accessor for the methods
WebServiceApi.Api soapApi = new WebServiceApi.Api();
// initialize a session and save the id to memory
string sessionId = soapApi.InitializeApi();
```

Set the active report.

```
// all methods require the session id parameter
soapApi.ReportObject_Activate(sessionId, reportPath);
```

Then call the appropriate Report_GetExecute method for the desired data type. Supported methods are Report_GetExecuteHtml and Report_GetExecuteData.

```
string reportHtml = soapApi.Report_GetExecuteHtml(sessionId);
```

JavaScript API

The Exago JavaScript (JS) API allows Exago functionality to be embedded directly into HTML div containers.

Divs can inherit code and styles from the host application. Since CSS cascades down to Exago, as well as up to the parent app, this allows you to maintain a single base of styles rather than separate ones for the host app and for Exago. And 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 functionality in the client browser. Besides the advantages of being able to embed in divs and interact programmatically, the API also allows for multiple calls to happen without needing to generate a new session for each one. As sessions are created only once per page load, this can increase the feeling of responsiveness in the host application.

Because the JS API runs on the client-side, it is not standalone. You are still required to generate session objects with either the **.NET** or **REST APIs**. 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 you cannot have multiple JS API objects active on the same page.

The JS API object provides a set of functions that are used to implement elements of Exago functionality.

Setup and Configuration

These steps describe how to configure your environment to use the JS API, as well as how to implement it in an application.

Create the Session

First you need to use the .NET or REST API to set up security and permissions for the session. Make all your configuration changes here, as these settings cannot be changed once the JS API object is loaded.

Set the WebAppBaseUrl property to the virtual directory where Exago BI is installed:

.NET

```
api.SetupData.WebAppBaseUrl = "http://server/Exago/";
```

REST

Do one of the following:

- PATCH **/Rest/Settings?sid={sid}**, Payload:

```
{ "WebReportsBaseUrl" : "http://server/Exago" }
```

- Add to the web service config file ({WebService}/Config/WebReportsApi.xml):

```
<webreportsbaseurl>http://server/Exago/</webreportsbaseurl>
```

The JS API has no concept of an *Active Report* or an *API Action*, so do not set these as they will have no effect. The action and report are specified by the individual JS function calls.

Note: A side-effect of this is that you cannot make per-session report changes in memory, since the JS API function can only act on saved reports. You will need to save any changes to disk instead.

When the session is ready, get the ApiKey. This encodes the session settings to pass to the JS API.

.NET

```
return api.GetApiKey();
```

REST

GET "**Rest/Sessions**{sid}", then get the `ApiKey` property from the response object:

```
{
  ...
  "ApiKey": "encodedAlphanumericApiKey"
}
```

Note: This is NOT the `UrlParamString / AppUrl`.

JS API Object

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.

Using the `ApiKey`, initialize a JS API object.

```
var api = new ExagoApi(ExagoBaseUrl, ApiKey, onLoadCallback, [showErrorDetail]);
```

- `ExagoBaseUrl` - URL to the installation of Exago BI
- `ApiKey` - key generated when the session was created
- `onLoadCallback` - function to execute once the JS API has been fully loaded and is ready for use
- Optional: `showErrorDetail` - set to `True` to see more detailed application error messages. (Default: `False`).

Note: `ApiKeys` are one-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 the `onLoadCallback` 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 User Interface in a div.

Parameter	Description
container	Div container to place the full UI 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 actions such as executing reports or creating new reports.

ExecuteReport(container, exportType, reportPath, [udf], [updateCallback], [errorCallback])

Execute a report or dashboard to a specific output type in a defined container.

Parameter	Description
container	Div container to place the executed report into
exportType	html pdf csv excel rtf
reportPath	Relative path to report to execute Example: "MyFolder\MyReport"
udf	Optional: Report UDF information for use with folder management
	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 'container': The same container HTMLElement that was passed in to the call
- Parameter 'updateType': The type of update as a string, either

updateCallback (container, updateType)

The parameter 'updateType' will assume one of these string values:

- "executionstart": The report has been loaded and is starting execution, or
- "initialcontentload": 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.

Optional: Callback to execute in the event an execution blocking error occurs

- Parameter 'container': The same container HTMLElement that was passed in to the call
- Parameter 'errorMessage': The error text

Return value: see errorCallback return values below.

ExecuteStaticReport(exportType, reportPath, udf, successCallback, [errorCallback])

Execute a report, and return its output to the successCallback function. Report is not interactive.

Parameter	Description
exportType	html pdf csv excel rtf json
reportPath	Relative path to report to execute Example: "MyFolder\MyReport"
udf	Report UDF information for use with folder management
successCallback (executionData)	<p>Callback to execute when execution request returns</p> <ul style="list-style-type: none"> • Parameter 'executionData': a string containing the executed report's data is passed as a parameter, whose formatting depends on the specified exportType. If the export type is "html", "csv", or "json", the returned string will contain all of the data in the corresponding format. If the export type is "pdf", "excel", or "rtf", the returned string will be partial URL to the file, e.g. "ExecuteExport.aspx?eid=...". See the below example for prefixing the returned partial URL with the base address.
errorCallback (errorMessage)	<p>Optional: Callback to execute in the event an error occurs.</p> <ul style="list-style-type: none"> • Parameter 'errorMessage': The error text.

```
const container = ...
const exportType = ...
const reportPath = ...

api.ExecuteStaticReport(exportType, reportPath, null,
  (executionData) =>
  {
    if (exportType == "excel" || exportType == "rtf" || exportType == "pdf")
      container.innerHTML = "<lt; iframe src=" + api.GetBaseAddress() + executionData + "'></ iframe>";
    else
      container.innerHTML = executionData;
  },
  (errorMessage) =>
  {
    container.innerHTML = errorMessage;
  });
```

ScheduleReportWizard(container, reportPath, [udf], [errorCallback])

Open the schedule report wizard for a report.

Parameter	Description
container	Div container to place the scheduled report wizard into

<code>reportPath</code>	Relative path to report to schedule Example: "MyFolder\\MyReport"
<code>udf</code>	Optional: Report UDF information for use with folder management
	Optional: Callback to execute in the event an error occurs, such as the scheduler being disabled
<code>errorCallback (container, errorMessage) => string</code>	<ul style="list-style-type: none"> Parameter 'container': The same container HTMLInputElement that was passed in to the call Parameter 'errorMessage': The error text
	Return value: see errorCallback return values below.

ScheduleReportManager(container, [errorCallback])

Open the schedule report manager.

Parameter	Description
<code>container</code>	Div container to place the scheduled report manager into
	Optional: Callback to execute in the event an error occurs, such as the scheduler being disabled
<code>errorCallback (container, errorMessage) => string</code>	<ul style="list-style-type: none"> Parameter 'container': The same container HTMLInputElement that was passed in to the call Parameter 'errorMessage': The error text
	Return value: see errorCallback return values below.

LoadReportTree(successCallback, [errorCallback])

Load the report tree as JSON, returned to the successCallback method.

Parameter	Description
<code>successCallback (reportTree)</code>	Callback to execute once the report tree has been loaded. <ul style="list-style-type: none"> Parameter 'reportTree': A JSON object representing the report tree.
	Optional: Callback to execute in the event an error occurs
<code>errorCallback (errorMessage)</code>	<ul style="list-style-type: none"> Parameter 'errorMessage': The error text

EditReport(container, reportPath, [udf], [errorCallback])

Load the report designer for a report.

Parameter	Description
<code>container</code>	Div container to place the report designer into
<code>reportPath</code>	Relative path to report to edit Example: "MyFolder\\MyReport"
<code>udf</code>	Optional: Report UDF information for use with folder management
	Optional: Callback to execute if the report fails to load
<code>errorCallback (container, errorMessage) => string</code>	Parameter 'container': The same container HTMLInputElement that was passed in to the call
	Parameter 'errorMessage': The error text
	Return value: see errorCallback return values below.

NewReport(container, reportType)

Load the report designer for a new report.

Parameter	Description
<code>container</code>	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.

Parameter	Description
container	Div container to dispose

IsAllowedReportType(reportType)

Returns whether or not a specified report type is allowed for the session.

Parameter	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", "Examples\\ClientReport");
}
```

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, ie. `ExecuteReport(...)`, the container will auto-dispose and execute the *OnDisposeContainer* callback if one is defined.

Example:

```
api.OnDisposeContainer = function(container) {
  container.parentElement.removeChild(container);
};
```

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 dialog's dismiss button will close the dialog and call 'DisposeContainerContent'. The container's content will be closed, the inner HTML will be cleared, and the OnDisposeContainer callback will be called.

Cookieless Sessions

Cookieless sessions are available in versions 2018.1 and above.

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 in application tasks. 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, and in versions of Exago BI prior to 2018.1, the session ID is stored as a browser cookie. However, when embedding using the JavaScript API in a cross-origin environment (where the Exago BI application and host application are on separate servers with different domains), cookies are considered a potential attack vector for cross-site scripting (see [IBM: Prevent a cross-site scripting attack](#) for more information). Therefore, most modern web browsers prevent embedded content from separate origins from using cookies.

If you are embedding Exago BI using the REST API and JavaScript API, and your host application and Exago BI are on separate domains, you will need to use cookieless sessions.

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.

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.

Cross-Origin Resource Sharing

If the Exago BI application is being accessed over a network, the Exago web server must be configured to allow cross-origin requests to the client 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.

To permit content to be accessed by whitelisted resources, you can configure the Exago web server to send a `Access-Control-Allow-Origin` header. For example:

```
Access-Control-Allow-Origin: "hostapp.server.com"
```

In IIS, this is either set in the **HTTP Response Headers** or **added manually to the web.config file**.

In **Apache**, following line is added to all relevant `.conf` files:

```
Header set Access-Control-Allow-Origin "hostapp.server.com"
```

See [Cross-Origin Resource Sharing \(CORS\) - HTTP | MDN \(Mozilla\)](#) for more information.

Assembly Data Sources

Exago BI supports the ability to retrieve data from .NET assemblies. Assembly Data Sources (ADS) are custom programmatic .NET Framework binaries for transforming and exposing arbitrary data in a format that can be read by Exago BI.

ADS can be used as data connectors and drivers for unconventional data sources. They can also be used as ETL (Extract, Transform, Load) layers for processing, combining, and caching data before it reaches the reporting engine. This article describes how to write custom .NET ADS for Exago BI.

ADS must be written for, and compiled with **supported versions of the .NET Framework** (which, at the time of this writing, is version **4.5** or higher). .NET Core and .NET Standard are not supported. The ADS must be compiled as a class library (.dll).

Interface

ADS must be coded to fit the following interface.

The entry point or points for Exago BI are public static methods, which accept at least an integer Call Type parameter and return an ADO.NET `System.Data.DataSet` object. The Call Type parameter name must be defined in the **Programmable Object Settings** of the **Admin Console**.

Parameter	Value
Programmable Object Settings ×	
Call Type Parameter Name	CallType
Column Parameter Name	Columns
Filter Parameter Name	Filter
Full Filter Parameter Name	FullFilter
Sort Parameter Name	Sort
Data Category Parameter Name	Category
Data Object ID Parameter Name	Id

Parameter names defined in the Admin Console

Note: Although the return type is a DataSet, each ADS method is expected to return only a single data table. This is usually done by merging in a `System.Data.DataTable` object which has a schema and has been populated with data. If a DataSet with multiple tables is returned, only the first one will be recognized.

Entry methods can optionally accept some or all other parameters described in the **Programmable Object Settings**, in order to have additional context for the method call. All the parameters in use must have defined names in the **Admin Console**.

In Exago BI v2016.2+, entry methods may also access a `WebReports.Api.Common.SessionInfo` object. This contains the active session's configuration settings and references to the active API objects. It can be used to identify properties as well as mutate session state. An assembly reference to `bin\WebReportsApi.dll` is required for the `SessionInfo` class definition.

Method Examples

Considering the following parameter names defined in the configuration:

- Call Type Parameter Name: `CallType`
- Column Parameter Name: `Columns`
- Filter Parameter Name: `Filter`

The following are valid entry method signatures:

```
public static DataSet Method(int CallType)
```

```
public static DataSet Method(int CallType, string Columns, string Filter)
```

```
public static DataSet Method(int CallType, string Columns = "")
```

```
public static DataSet Method(SessionInfo Session, int CallType)
```

```
public static object Method(SessionInfo Session, object CallType)
```

There can be more than one entry method. Each public static method in the class will appear as a virtual table when adding the ADS data model to the Admin Console.

Alternatively, one method can support multiple tables by incorporating the Object ID parameter, which passes the object ID as defined in the configuration. See **Handling Multiple Data Tables** for details.

Additional Notes

1. Methods cannot be overloaded, as it will be ambiguous which one Exago BI should use.
2. For security reasons, only the entry methods in the class should be marked as public, so that any other utility or misc methods are not erroneously visible in the Admin Console.

Class Definition

The entry methods must reside within a public class (*non-static*) within a defined namespace. The class must be at the top level of the namespace, not within another class. Its default constructor must be public (defining the constructor is optional).

Example:

```
namespace AssemblyDataSource {
    public class Tables {
        public Tables() { ... } /* optional */
        public static DataSet EntryMethod(...) {...}
    }
}
```

Note: Due to a bug, in versions prior to v2017.3.4, an assembly cannot contain multiple data source classes. This has since been addressed.

Implementation

Each ADS query is a discrete atomic operation. While you may choose to have Exago BI cache the assembly in memory, the application will not maintain a connection with it between queries. Unless you store application state on disk, queries will have no knowledge of each other.

ADS method calls are synchronous, meaning that the main application thread will wait until the method returns before continuing.

Exago BI queries ADS for three distinct reasons:

- **Schema:** The column/field names and data types. This is called whenever the schema needs to be retrieved (such as expanding a category's fields in the **ExpressView** data pane) and the **Schema Access Type** is set to *Datasource*. The frequency of schema calls can be significantly reduced by specifying a static schema for the ADS in the **Column Metadata**.
- **Data:** The data values for the table. This is called whenever a report is executed.
- **Filter values:** Data values for a single column, possibly filtered by an input string. This is called when values need to be populated for a filter dropdown list.

Each type of query requires a different amount of information to be returned. The **Programmable Data Object** parameters are passed alongside each query to indicate the information that Exago BI is expecting. With this in mind, you can write ADS with certain optimizations to improve performance.

Note: If performance is a strict concern, consider that databases will almost always be more performant for large data sets than ADS.

See **Optimizing for Query Types** for more details.

Data Format

Exago BI expects an ADO.NET DataSet object to be returned from the ADS method. A DataSet is an in-memory representation of a

relational data model.

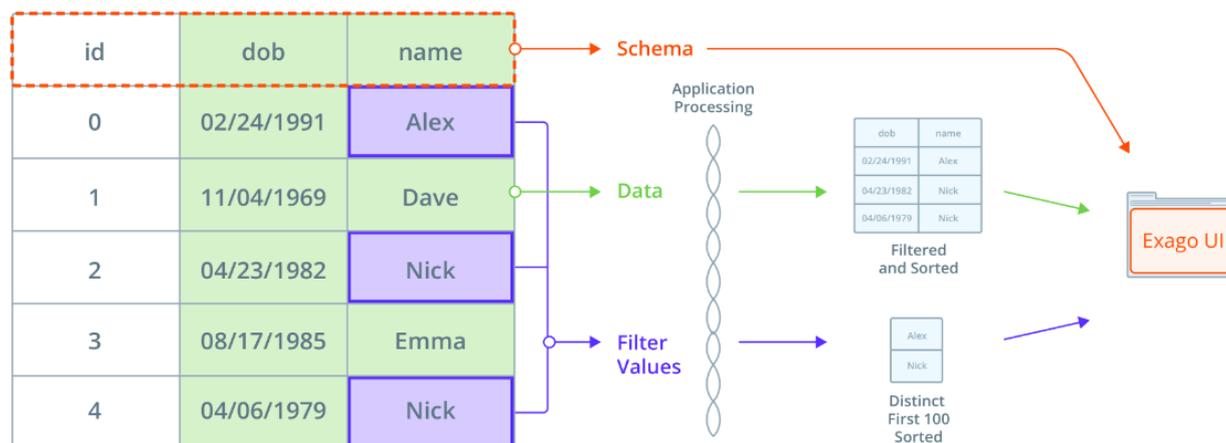
To return no data, for any reason, construct and return a DataSet with the schema and no rows. Sending null or an empty schema will cause an exception to be thrown.

Exago BI reads only the first DataTable in a set, at index `DataSet.Tables[0]`, regardless of the number of tables comprising the set. To return multiple tables from an ADS, return a different DataSet depending on an input parameter or method name. See **Handling Multiple Data Tables** for more information.

See [DataSets](#), [DataTables](#), and [DataViews](#) for the ADO.NET DataSet documentation.

Optimizing for Query Types

Avoid retrieving the full data set whenever possible. This is usually the most computationally intensive query task. A number of Programmable Object Parameters are passed to the ADS in order to determine the amount of processing required for each query.



The ADS query types, what data they expect, and how the data is processed by Exago

Using the Call Type Parameter

Call Type is an integer parameter, with value 0 for a schema query, 1 for a data set query, and 2 for a filter values query.

Schema queries are expected to be quick, and do not require the actual data table. Returning a static schema is the most performant way to handle this. Do not make the application wait to populate the full table.

Warning: It is highly recommended to set the **Schema Access Type** for the ADS to *Metadata* in order to eliminate schema queries to the data source. This will significantly improve the responsiveness of the user interface.

Filter values query a single data column, with a text filter applied to limit the number of values returned. Therefore this call type works best with the Column and Filter parameters.

Column, Filter, and Full Filter Parameters

For data queries (Call Type 1), it is not strictly necessary to implement these parameters to filter the data in the ADS, because Exago BI will filter the data regardless once it is returned to the application.

Suppressing Application Filtering & Sorting:

Beginning with *v2018.1*, you have the option to suppress the default application filtering and sorting of data returned for data queries. You can choose to filter and sort in the ADS code instead, which allows for more control over the manner in which these processes are done.

To suppress application filtering and sorting for selected objects, set the **Suppress Sort and Filter (Objects)** setting in the **Admin Console** to *True* for each object.

Note: Suppression will only occur for reports where the object is joined exclusively to zero or more programmable objects which also have suppression enabled. In-memory processing such as advanced joins, special Cartesian processing, and sort and filter formulas will override this setting and force application processing to occur.

Warning: Exago BI will not filter the dropdown values for filter values queries (Call Type 2) by default. It is highly recommended to apply filtering for these queries in the ADS, to avoid inaccurate and potentially insecure behavior for the filter dropdown lists. Therefore, these parameters should not be seen as optional for all but the most trivial applications.

The following table shows the parameter values for either data or filter values queries.

	Data	Filter values
Call Type	1	2
Column	List of requested column database names "col1,col2,col3"	Requested column database name "col2"
Filter	Data object filter SQL "(col1 = 'value') AND (tenantId = 'id')"	Filter dropdown value "value"
Full Filter	Report filter SQL "([obj].[col1] = 'value') AND ([obj].[tenantId] = 'id')"	Tenant and row-level filter SQL "([obj].[tenantId] = 'id')"
Sort	Sort SQL "col3 asc, col1 desc"	Empty string
Expected Return	DataSet with requested columns, optionally filtered	DataSet with single column, filtered

The Column parameter will give the list of fields necessary for a report execution (data query) or the field requested by a filter dropdown (filter values query). It may be more performant to build the DataSet with only the requested columns.

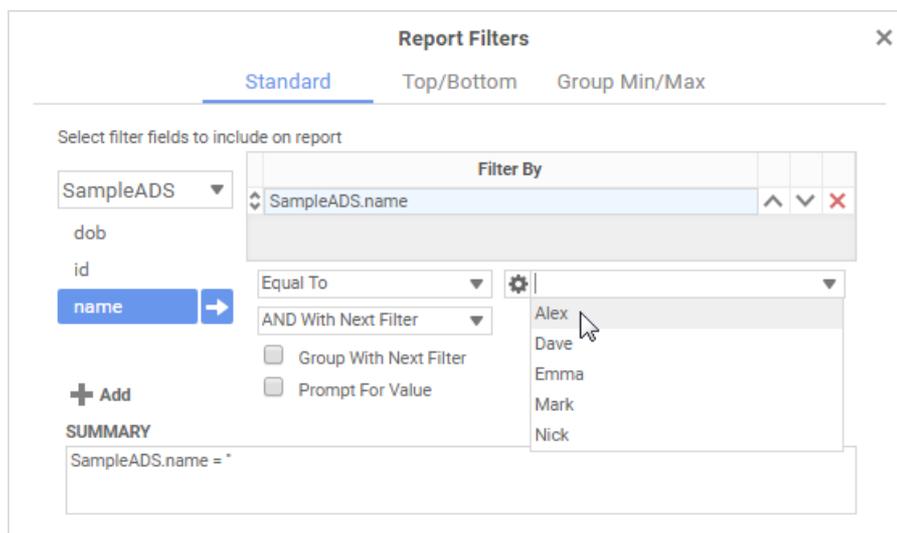
Filtering for data queries is only required if the **Suppress Sort and Filter** setting (v2018.1+) is *True*. Otherwise Exago BI automatically filters the data set in memory for data queries.

Note: In Exago BI v2018.2+, users will be able to filter by formulas, using the application's internal formula engine. It is impossible to handle formula filters in an ADS code.

You must apply filtering for filter value queries.

What is a filter values query?

Several activities in the application involve querying a table for a list of values from a single field, possibly filtered by a user input string. This occurs when users interact with filter dropdowns, so that they can view and select valid field data when creating report filters.



Example of interaction with a filter dropdown

Note: Report viewer and dashboard interactive filters are populated by the data query for the report execution, not by a separate filter values query.

You can prevent this type of query entirely by setting the **Read Database for Filter Values (Filter Settings)** setting in the **Admin Console** to *False*.

Note: It is not possible to implement Filter Dependencies in an ADS.

Typing into a filter dropdown field causes it to return values filtered by the input text. This filtering must be done manually in the ADS for two reasons:

1. Accuracy: Dropdown lists are not filtered by Exago BI, so typing to filter will be ineffective if not implemented manually.
2. Security: Row and column tenancy filters must be applied to the dropdown list so that users cannot view data values without access rights.

The Column, Filter, and Full Filter parameters must be combined together to generate the proper filtering. With Call Type 2, the Column parameter contains the name of the requested data field, and the Filter parameter contains the input text. These can be combined to generate the necessary SQL WHERE clause.

Example:

```
string SQL = string.Format("(CONVERT({0}, System.String) LIKE '%{1}%'", Columns, Filter);
// Note: DateTime columns should use a different comparison
```

The Full Filter parameter already contains the necessary tenancy SQL specified in the data object and active role settings, so this can be easily added to the input filter SQL to create the full clause. Then apply it to the data set and return only the resulting rows.

```
SQL += " AND " + FullFilter;
DataRow[] DropdownRows = FullTable.Select(SQL);
// Note: In a proper implementation be sure to sanitize the SQL before passing it to the db
```

Additional Considerations

DateTime comparisons should be handled in a different manner than with a LIKE operator.

Only distinct values are shown in the dropdown, so there is no need to use a DISTINCT clause in the SQL statement.

A maximum of 100 values are shown in the list. You can choose to implement a TOP 100 clause, if it would improve performance.

As of Exago BI v2017.3+, for database sources, end-users have the option of selecting, via a menu, whether their filter dropdown input should apply a **Starts With or Contains** filter to the data field. This behavior cannot currently be implemented in an ADS, because the value of this setting is not passed in a parameter.

As of v2017.3+, values are queried incrementally as the user scrolls the dropdown list, for supported sources. This is also not currently supported by ADS.

Sort Parameter

It is only required to sort in an ADS if the **Suppress Sort and Filter** setting (v2018.1+) is set to *True*. Otherwise Exago BI automatically sorts the data set in memory for data queries.

Note: Formula sorts use the application's internal formula engine. It is impossible to handle formula sorts in an ADS code.

Note: The sort parameter is irrelevant for filter values queries. Filter values are always sorted in ascending order.

Handling Multiple Data Tables

There are two ways which an ADS can allow for multiple data tables to be returned as separate objects in the Exago BI data model. The first involves using multiple entry methods. The second involves specifying table names in the metadata.

Multiple Methods

Separate tables can be implemented as separate entry methods within the ADS. This has several advantages. Each public method from a source will appear as a "table" in the **Admin Console** when adding objects to the data model. Method names can describe the tables in a self-documenting manner.

Example:

```
/* Public methods seen by Exago BI */
public static DataSet EmployeeTable(int call) { return GetTableData("Employee", call); }
public static DataSet ProductsTable(int call) { return GetTableData("Products", call); }
/* Common data retrieval method */
private static DataSet GetTableData(string tableName, int call) { ... }
```

However, this does not allow retrieval of previously undefined tables without editing the ADS and adding additional methods in support. So for a database where objects are likely to be added, programmatic maintenance of the ADS will be required, which is not ideal.

Single Method with Parameters

The Data Category and Data Object ID parameters correspond to the user-input metadata fields *Category* and *ID* (respectively) in each **data object's properties** in the **Admin Console**. The ID field, in particular, can accept arbitrary input without affecting the user interface. So table names or identifiers can be defined in the ID field. Thus, one entry method can support multiple tables based on a conditional string from the metadata. This allows an ADS to support a more dynamic data structure, since any subsequent tables can be added simply by editing the metadata.

Example:

```
public static DataSet GetTableData(string objectId, int call)
{
    switch (objectId)
    {
        case "Employee": { ... }
        case "Products": { ... }
        default: { throw ... }
    }
}
```

However, adding metadata elements to the config can be error-prone, so ensure that admins always have the up-to-date mapping of IDs to tables.

Configuration

Once written, the following steps are taken for Exago BI to access the ADS.

1. Ensure that the assembly is in a location accessible by the IIS application pool user (and the Exago Scheduler service, if in use), with read and execute permissions.
2. Add the ADS to the Admin Console data sources (with `Type=Assembly`) using the following connection string format:

```
assembly=\path\to\assembly.dll;class=Namespace.Class
```

3. Click the **connection verification** ⚡ icon to verify that the assembly and class are accessible.
4. Define the method parameter names in the **Programmable Object Settings**.
5. Add the entry methods to the **Admin Console** data objects (the Automatic Data Discovery tool does not support ADS), then add any relevant joins and metadata aliasing.

Maintenance

There are several notes to keep in mind when updating an ADS and/or updating Exago BI.

System Locks

The web server maintains a lock on the ADS file as long as it is running. Additionally, if using any classes in the `WebReports.Api.Scheduler` namespace, or if there are any scheduled or **cached** reports that use the ADS, the scheduler services will lock it as well.

When updating the ADS, the web server and schedulers will need to be disabled while the file is replaced. So it is generally not appropriate to update an ADS in a live environment.

WebReportsApi.dll Version

If you are using `SessionInfo` or any other Exago .NET API classes, the ADS must be recompiled with the new version of the `WebReportsApi.dll` assembly whenever Exago BI is updated.

Note: Utility libraries, such as `SQLUtils.dll` are updated less frequently than the main web application binaries. Even if only referencing utilities, verify nevertheless that the ADS works without needing recompilation.

Example

The following example is provided for reference. While trivial, the example is fully working and demonstrates a typical ADS framework. It can be freely used, extended, and redistributed for any custom implementation.

[Download the example.](#)

Application 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 application's Temp path (specified in the base configuration). 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.

Use the following administrative setting to set the log level or disable logging:

(**Other Settings**) Write Log File

- **None** – Turns logging off.
- **Error** – Logs application errors.
- **Warn** – Logs application warnings, which may be indicative of problems in the configuration, as well as all **Error** messages.
- **Info** – 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

Start time, userId, companyId, full report name, filter summary.

Execution end

End time, runtime, userId, companyId, full report name.

- **Debug** – 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.

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.

Note: A custom log configuration file will override the application's configuration settings.

To use a custom log configuration, create a file called `log4net.config` in the Config directory of the Exago web application. The following shows a sample config file:

```
<?xml version="1.0" encoding="UTF-8"?>
<log4net>
  <appender name="RollingFileAppender" type="log4net.Appender.RollingFileAppender">
    <file value="C:\Exago\Temp\WebReportsLog.txt" />
    <encoding value="utf-8" />
    <appendToFile value="true" />
    <rollingStyle value="Size" />
    <maxSizeRollBackups value="10" />
    <maximumFileSize value="1MB" />
    <staticLogFileName value="true" />
    <lockingModel type="log4net.Appender.FileAppender+ExclusiveLock" />
    <layout type="log4net.Layout.PatternLayout">
      <conversionPattern value="%date %-5level [%property{SessionId}] %message%newline" />
    </layout>
  </appender>
  <!-- Setup the root category, add the appenders and set the default level -->
  <root>
    <level value="INFO" />
    <appender-ref ref="RollingFileAppender" />
  </root>
</log4net>
```

For more information and extensibility, see: [Apache log4net](#). See the following examples for some simple modifications:

Change Logfile Location

```
<file value="Path\To\Log.txt" />
```

Specifies the directory and filename for the log file.

Change Logging Level

```
<level value="INFO" />
```

Specifies the Exago logging level: ERROR, INFO, or DEBUG.

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.

Changing the 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`.

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.