# Technical Guide
## Version 2016.1

| Exago, Inc.<br>Two Enterprise Drive<br>Shelton, CT 06484 USA | |
|---|---|
| Phone | 203.225.0876 |
| Fax | 203.926.9505 |
| Email | **exagoinc.com** |
| Support & development | |
| Phone | 845.481.5221 |
| Fax | 845.255.0209 |
| Email | **support@exagoinc.com** |
| Web | **exagosupport.com** |
| Blog | **exagoinc.com/blog.html** |

# Table of Contents

# Technical Overview

Exago is an ASP .NET web application that utilizes C#, JavaScript and AJAX. Exago consists of four main components.

1.  **.NET API**: Controls all server-side processing and the user interface. The .NET API can also be called by your host application to integrate Exago. For more information see **.NET API** and **Integration**.

2.  **User Interface**: ASP.NET pages that are converted to HTML at runtime. To access the UI see **Accessing the Administration Console**.

3.  **REST and Web Service API**: Gives non .NET host applications access for integration purposes. For more information see **REST API**, **Web Services API**, and **Integration**.

4.  **Report Scheduler Service**: Windows service used that handles processing scheduled reports. The Scheduler can also be used for load balancing. For more information see **Scheduler Installation** and **Load Balancing Execution**.

## Architecture

The diagram below details the architecture of Exago:
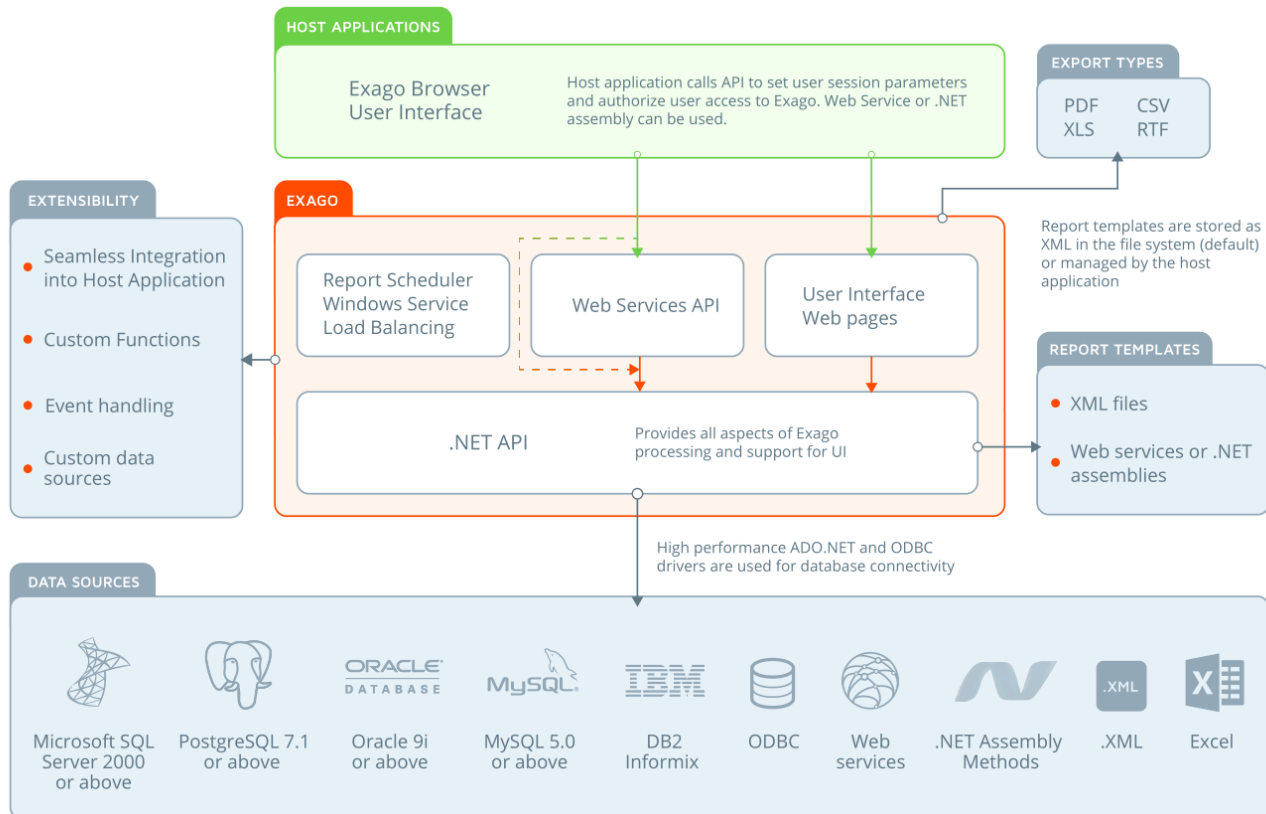


**Host Application**

The Host Application uses the REST, Web Service, or .NET API to set user permissions and embed the User Interface.

## Exago

Exago uses the .NET API to process reports and support the User Interface. High performance ADO.NET drivers are used to for database connectivity.

## Report Templates

Report templates are stored as XML in the file system by default. Alternatively the Host Application can use a Web Service to manage report template storage.

## Data Sources

Exago can retrieve data from tables, views, stored procedures, Web Services, .NET Assemblies or custom SQL. Data can be joined across data sources to provide additional flexibility.

## Extensibility

Exago provides several features that allow the Host Application to dynamically extend its capabilities.

# Installation

The following chapter details the system requirements and walks through the installation of Exago.

> NOTE. Please be sure to disable your antivirus software (and check to make sure it's stopped for services, and, not running in your task list) before installing. Antivirus software may lock up the installation or cause it to fail.

To begin download the installer from our **support site**. Make sure your antivirus software is disabled and run the installer as Administrator. There are three components of Exago that can be installed, but only the Web Application is required.

## Web Application

This component consists of the User Interface and the .NET assembly WebReportsApi.dll which can be used directly by .NET host applications. See **Web Application Installation**.

## Web Service API (Optional)

This component provides a platform independent means of communication with Exago at runtime. As well as being platform independent the Web Service API provides cross domain accessibility and application isolation. See **Web Service Installation**.

## Scheduler Service (Optional)

This component uses .NET Remoting to communicate with Exago. This service can be used to load balance report processing. Additionally, it can be used to schedule and email reports. The Exago Scheduling Service can be installed on any server that can communicate with the Exago web application via an HTTP URL/Port. See **Scheduler Service Installation**.

# System Requirements

The following components are required to install and run Exago:

**Windows:**

Windows Server 2003 or greater / Windows XP / Windows Vista / Windows 7 / 8 / 10
Internet Information Services v5.1 or greater
Microsoft .NET Framework version 4.5 or greater

> NOTE. IIS should be installed prior to the .NET Framework. If IIS is installed after the .NET Framework, then the .NET Framework must be reinstalled or repaired via Add/Remove Programs >.NET Framework 4.0 > Change/Remove, then choose the Repair option.

**Linux:**

Red Hat Enterprise Linux 7+ / SLES 12+ / CentOS 7+ / Fedora 21+ / Debian 8+ / Ubuntu 14+
Apache HTTP Server 2.4+
Mono 3.12+ (4.0+ recommended)
mod-mono 3.2.8+
mono-basic (optional VB.NET support)
any basic font or font package

**Data Sources (one or more):**

Microsoft SQL Server 2000 or greater
Oracle 9i or greater
MySQL 5.0 or greater
PostgreSQL 7.1 or greater
Web Services
.NET Assemblies

**Data Adapters (one or more):**

Oracle – ODAC11 from oracle.com
MySQL – Connector/Net from mysql.com
PostgreSQL – dotConnect for PostgreSQL Express from devart.com

# Web Application Installation

## Installing the Web Application

For instructions on installing Exago on Linux, see **Installing Exago on Linux**.

Use the following steps to install the web application on Windows:

- Download the installation wizard from our **support site**.

- Make sure your antivirus software is disabled and run the installation Wizard as Administrator.



- Click the Web Application button.
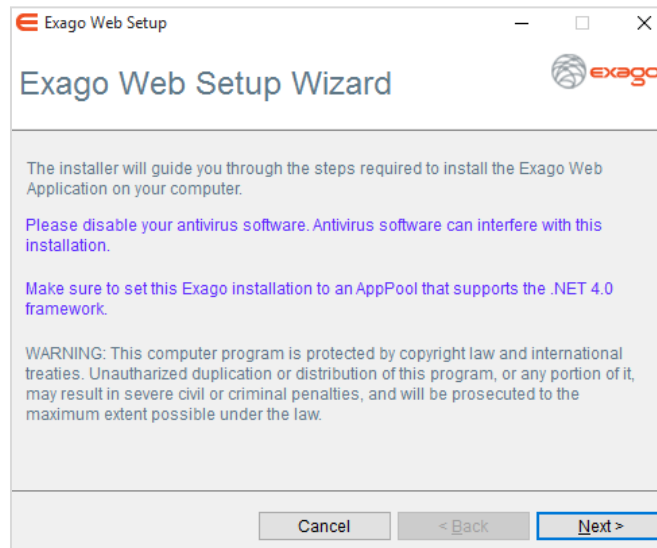
- Click **Next** to bring up the 'Select Installation Location' menu.
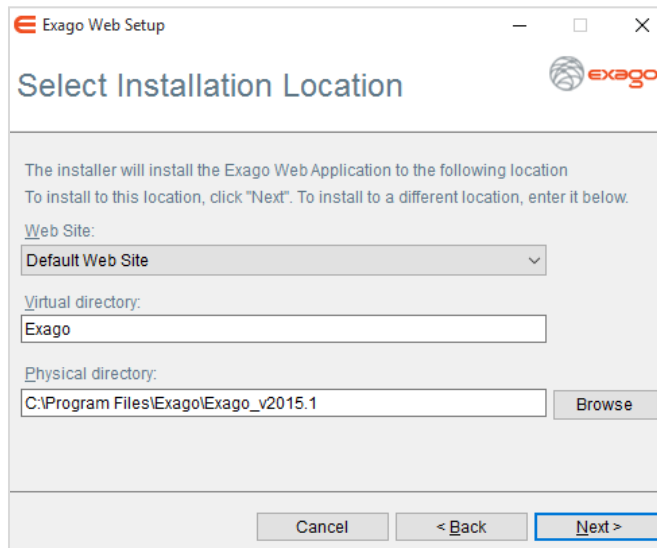


- In this menu specify the web site, virtual directory and physical directory where you want Exago installed. Click **Next**.
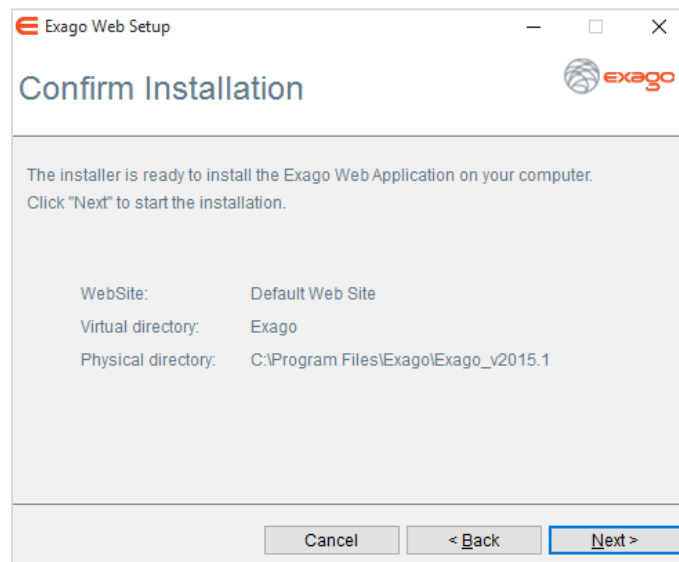
- Confirm your location selections by clicking **Next**.
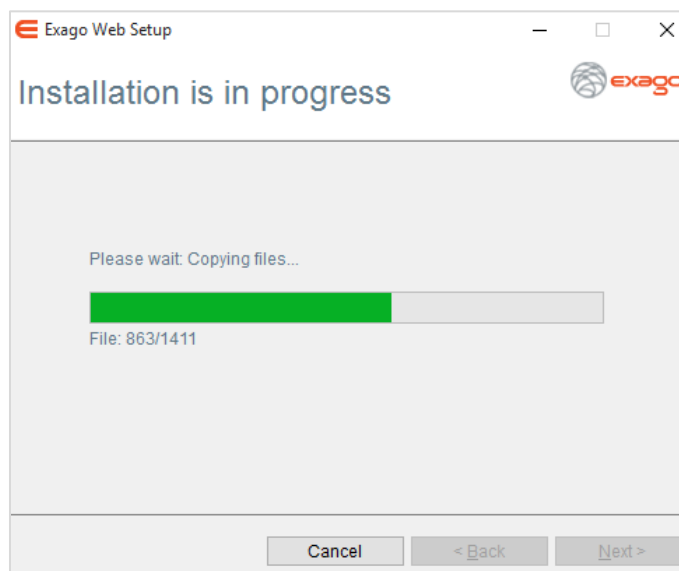


- Monitor the installation and click **Finish** when it is complete.

## Configuring Exago

After the installation is complete, configure Exago using the following steps:

- 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.

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

- Give the Report Folder read and write privileges for the ASP.NET user.

- Specify the location of the Report Folder in the 'Report Path' setting of the Administration Console. See **Accessing the Administration Console** and **Main Settings** of the General Section.

    Below are three examples of report paths to the folder \ReportsRepository:

    1. C:\Program Files\Exago\ ReportsRepository – Folder is on a file system.

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

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

- Verify that the user running under the Exago instance within IIS has read/write privileges on the folders below:

    o The folder specified in the Report Path of the **Main settings** of Administration Console.

    o The Config sub-folder of the Exago installation.

    o The folder specified in the Temp Path. By default this is a sub-folder of Exago called 'Temp'. However this can be changed in the **Main Settings** of the Administration Console.

# Web Service Installation

## Installing the Web Services API

For instructions on installing the Web Services API on Linux, see **Installing Exago on Linux**.

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

- Download the installation wizard from our **support site**.

- Make sure your antivirus software is disabled and run the installation wizard as an Administrator.



- Click the Web Service button.
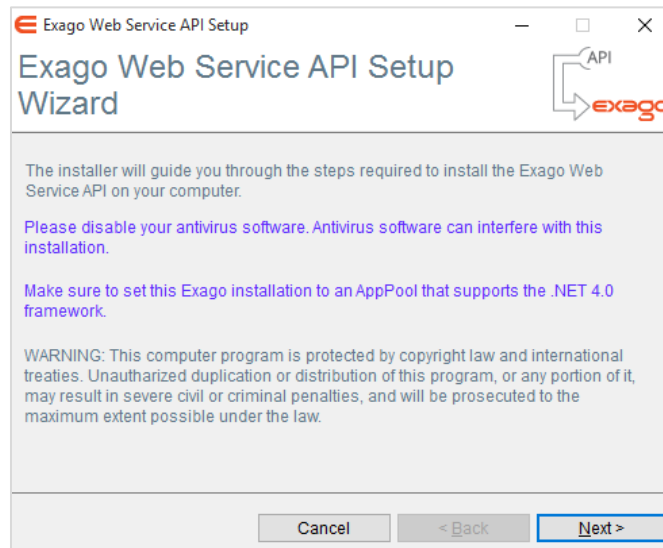
- Click **Next** to bring up the 'Select Installation Location' menu.
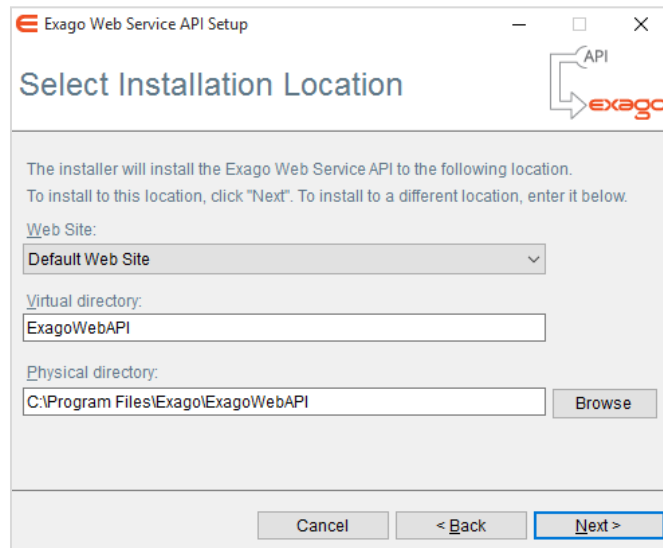


- In this menu specify the web site, virtual directory and physical directory where you want Exago installed. Click **Next**.

  NOTE. The Web Service API must be installed on the **same server and web site** as the Exago Application.
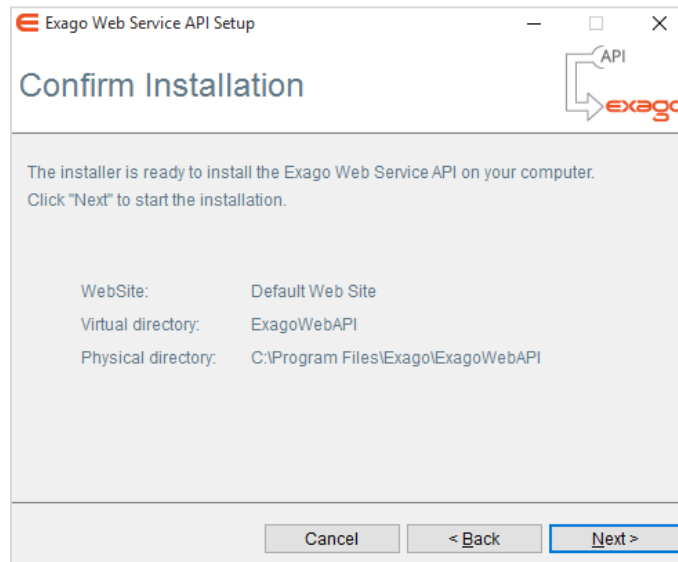
- Confirm your location selections by clicking **Next**.
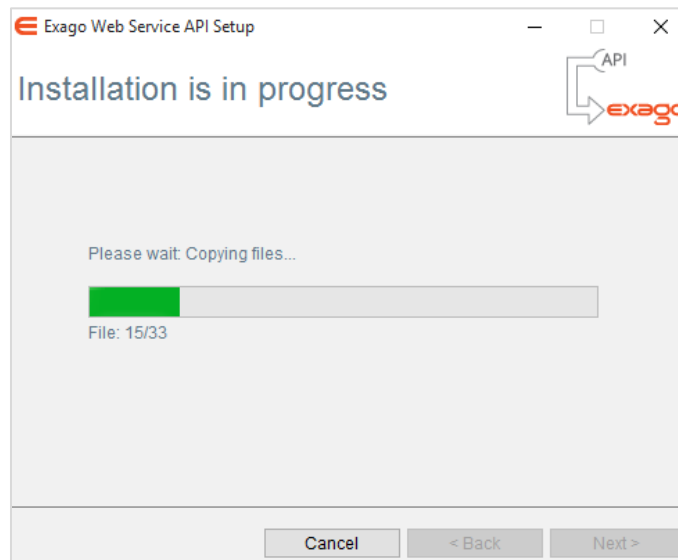


- Monitor the installation and click **Finish** when it is complete.

## 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** – IIS virtual directory of the Exago web application. For example, entering '/Exago' will cause the Web Service API to look for the Exago virtual directory.

- **throwexceptiononerror** – set to true if you want to catch exceptions in your application thrown by Exago.

- **writelog** – set to true to write a log file (WebReportsApiLog.txt in the Config sub-folder) of any exceptions thrown. Write permissions for the Config sub-folder must be given to the ASP.NET user.

# Scheduler Service Installation

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.

## Installing the Scheduler Service

For instructions on installing the Scheduler Service on Linux, see **Installing Exago on Linux**.

Use the following steps to install the Scheduler Service on Windows:

- Download the installation wizard from our **support site**.

- Make sure your antivirus is software disabled and run the installation wizard as an Administrator.



- Click the Scheduler button.
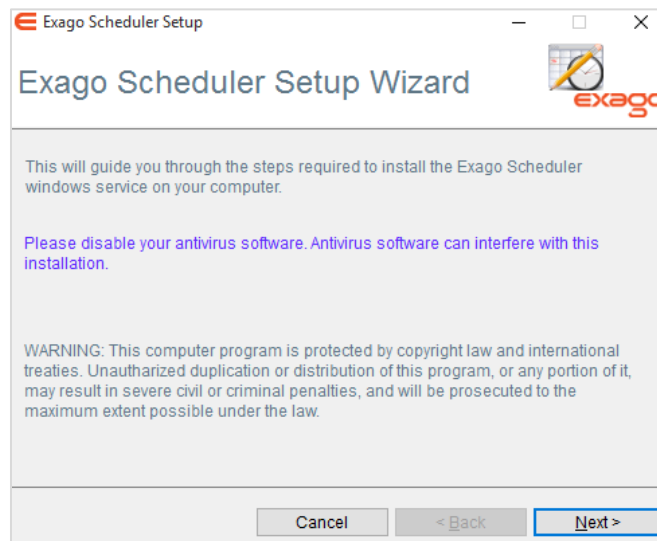
- Click **Next** to bring up the 'Select Installation Location' menu.
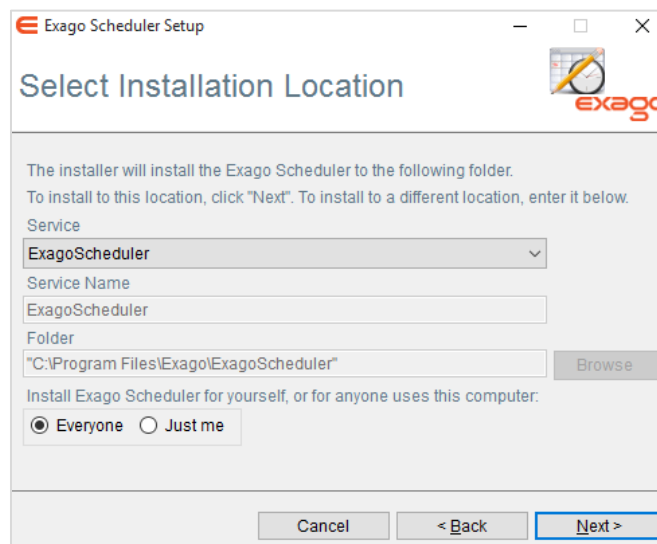


- Specify if you want to create a new service or if you want to update an existing one.

- 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**.
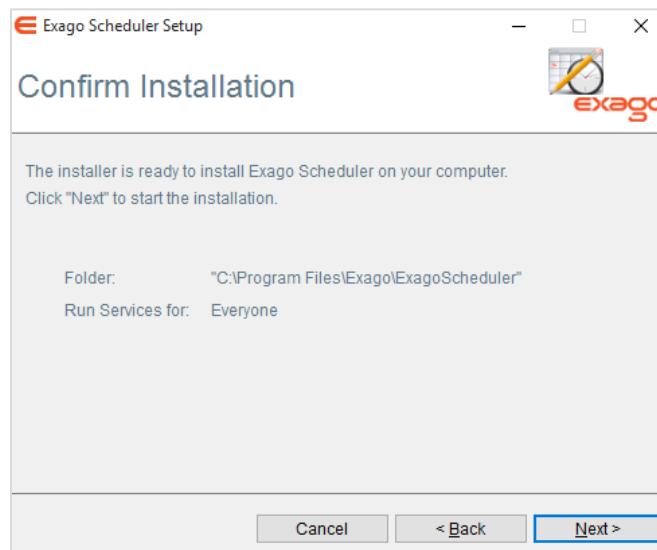
- Confirm your location selections by clicking **Next**
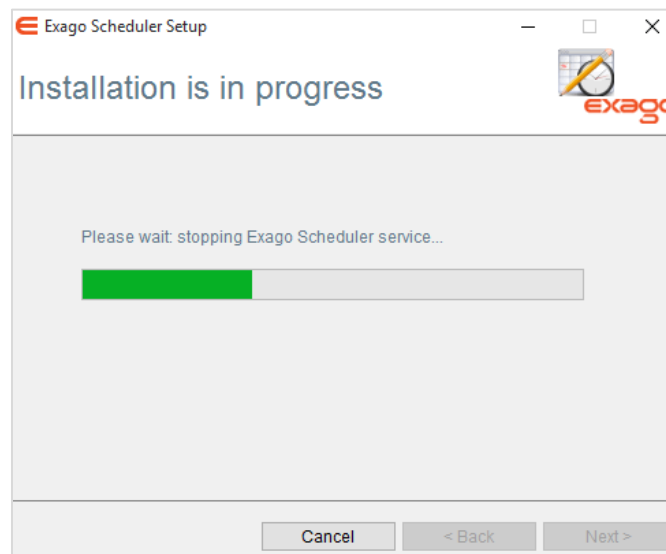


- Monitor the installation and click **Finish** when it is complete.

## Configuring Scheduler Services

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

Set the following items:

NOTE. Settings that can be true or false are case sensitive and should use lower case. Ex. encrypt_schedule_files will cause an error for True.

- **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 id 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** of the Administration Console.

- **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 Administration Console.

- **working_directory** – The directory where scheduled documents and temporary files are written. The default setting '[INSTALLDIR] working' will create a 'working' folder in the location the scheduler was installed.

- **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.'

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

- **sleep_time** - The time interval 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** – Set to 'on' in order to log events to ExagoScheduler.log in the working directory.

- **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 Administration Console. 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 Report Viewer capabilities when using remote execution. It is not recommended setting this value any lower than 60 minutes.

- **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.

## Starting and Changing Scheduler Services
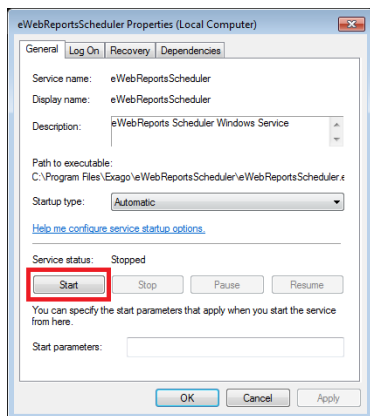
The Windows Service will have to be manually started for new installations of the Scheduler. Starting the service will create the working directory as set in 'working_directory' described above.

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.

# Installing Exago on Linux

The Exago Linux Setup Application can be used to install the Web Application, Web Service API, and Scheduler Service. The application can also install a compatible version of Mono. Use the following steps to install Exago on Linux.

- Download the Linux Setup Application from our **support site**.

- Decompress the download into a temporary location.

- Run `installExago.sh` as root.

  NOTE. SELinux is not supported by default. The installer will exit if SELinux is running.

  o The installer will attempt to detect certain system information such as OS and Apache versions. If it cannot detect something it will prompt for the information.

  o Specify an install path when prompted. If you do not specify a path the installer will default to '`/opt/Exago`'.

  o You may elect to install Mono when prompted. If you already have Mono installed this option will have no effect. You can also run `installMono.sh` at a later time.

  o You will be prompted to select which components to install. Select up to three components by typing their respective numbers delineated by a space. If you are not sure whether you will need a certain component, you can install it at a later time by running `installExago.sh` again.

    1. Web Application

    2. Web Service API

    3. Scheduler
       NOTE. The Scheduler will not run as a Linux service by default. It must be added to your init script.

  o If you elect to install the Web Service API and/or the Scheduler, the installer will create the subdirectories '`WebServiceApi`' and/or '`Scheduler`' in your previously specified install path.

  o The installer will detect your Apache installation and generate a default configuration file called `exago.conf` in your Apache site path. You can also run `configApache.sh` at a later time.

  o The installer will automatically set read/write permissions for the current Apache user on the install paths, Config directories and Temp directories.

- You must manually create and set the permissions for a report directory. For instructions on how to configure a report directory see **Configuring Exago**.

- For instructions on configuring the Web Service API, see **Configuring Web Services API**.

- For instructions on configuring the Scheduler, see **Configuring Scheduler Services**.

# Installation Manifest

When installing the Exago Host Application, the installer will create a new manifest file on your system called 'ExagoMainfest.txt'. **It is very important that you do not delete this file.**

Some features in the Exago application enable you to create your own files located in the Exago' installation folder outside of the Config folder. The manifest file ensures that future installs of Exago do not wipe out the local files that you have created.

During subsequent upgrades to the Host Application, the installer will read the manifest file to determine which files to over-write. If the manifest file *exists but the installer cannot access it*, the installation will fail and give notification.

If, however, the manifest file *does not exist,* **any files in the Exago tree outside of the *Config* folder will be deleted**. Additionally, any custom .aspx pages or image files that live outside the *Config* folder (such as **per-user styling**) will be erased.

# Administration Console

The following chapter details how to configure data, set permissions and enable/disable features through the Administration Console.

## About

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

- Establish how to connect to data. Determine what data should be exposed to users. See **Data**.

- Modify global settings of Exago to enable/disable features. See **General**.

- Create and modify security Roles for individuals or groups of users. See **Roles**.

- Create and modify custom functions to make calculations on reports. See **Functions**.

- Create and modify custom code that is run when reports execute. See **Server Events**.

- Create and modify custom options that can be set on reports. See **Custom Options**.

The Administration Console creates three configuration files: an XML file called WebReports.xml, a backup of the XML file called webreport.xml.backup, and an encrypted XML file called WebReports.xml.enc. These files are created and saved in the Config folder where Exago was installed.

### Important Security Notes

- Each time you save the Administration Console settings a backup copy of WebReports.xml is created. Store this xml copy in a secure place and delete the WebReports.xml and the WebReports.xml.backup from the Config directory.

- Before deploying Exago into a production evnrionment be sure to set a value for the Temp Path in Main Settings to a location that resides behind 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** page.

- 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.
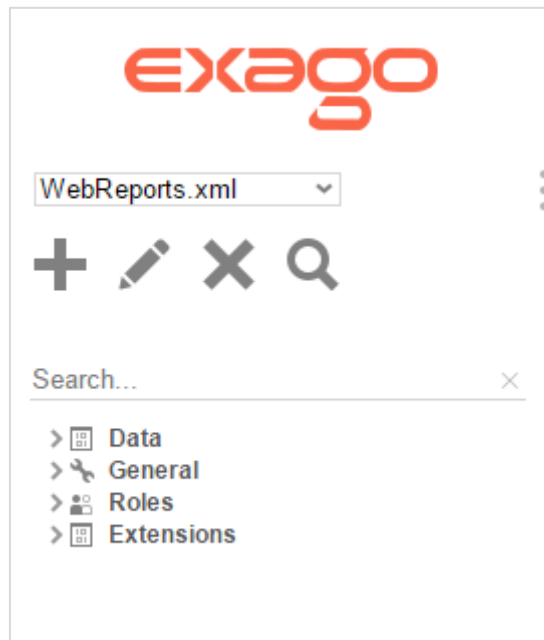
# Accessing the Administration Console

Once Exago is **installed** navigate your browser to **http://'Your Server'/Exago/Admin.aspx**. 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.
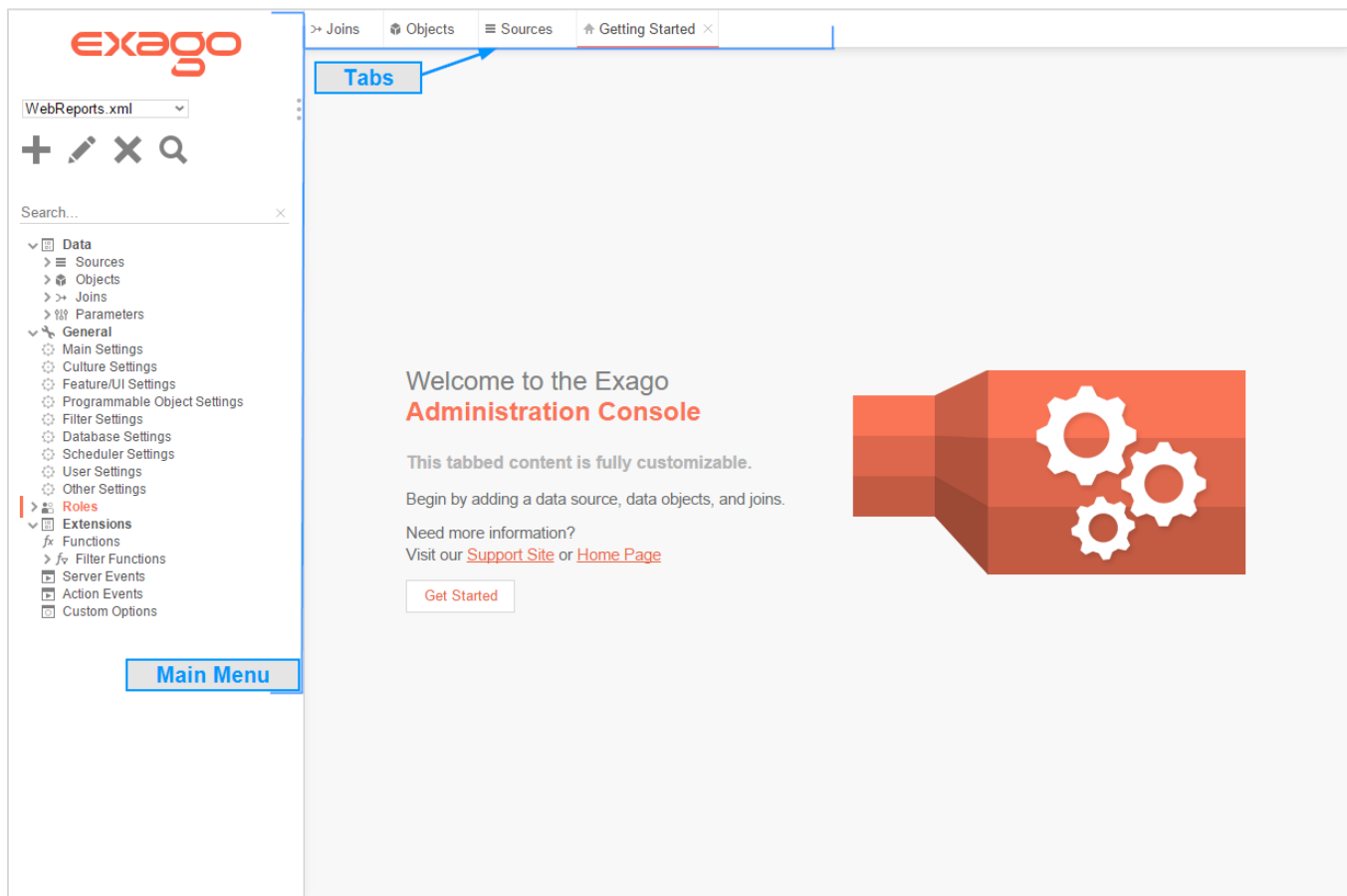
## Main Menu

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 circles (⬢) to hide the main menu.

## Tabs

The right section of Exago is made up of tabs containing the 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.
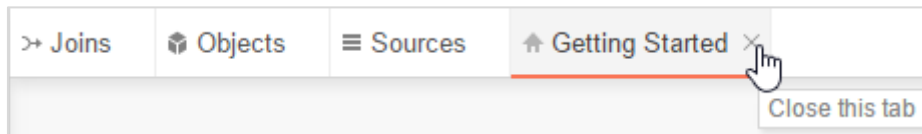
Tabs can also be rearranged by clicking and dragging them as desired.
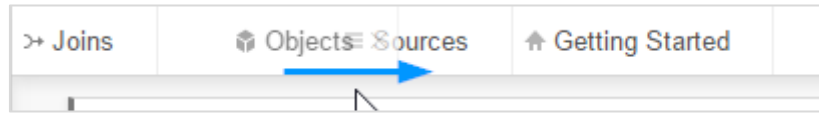


## Supported Browsers

The Administration console can be accessed by the following browsers:

- Firefox 3+
- Internet Explorer 8+
- Google Chrome
- Safari

NOTE. In Internet Explorer's Compatibility Mode some items may not display correctly.

# Data

This chapter discusses how to determine which data should be made available to users to create reports. Using Data Sources, Data Objects, Parameters and Joins you can create user friendly names and control what data users may access.
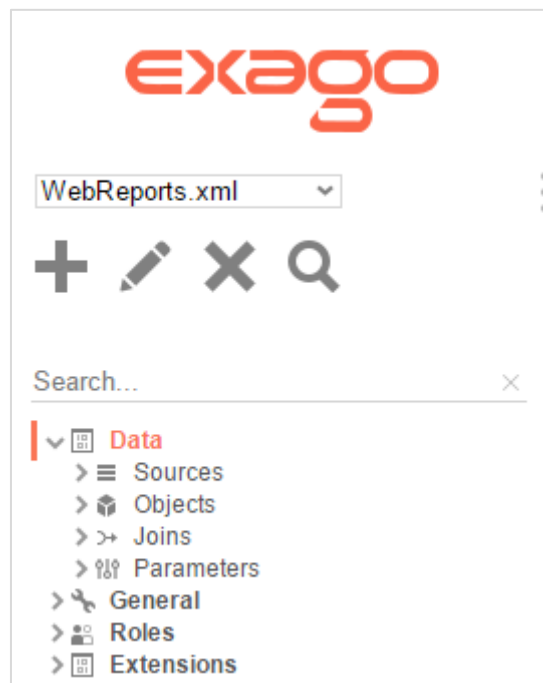
All existing Data elements are listed in the **Main Menu** under Data. Whenever a data element is added or modified it will be displayed in a **Tab** based on its type. For example, all Data Objects you select to edit will appear in an Objects tab.

- To add a new element select the type (Sources, Object, Parameter or Join) in the Main Menu under Data, then click the add button ( ✚ ).

- To edit an element either double click it or select the desired element and click the edit button ( 🖊 ).

- To delete an element select it and click the delete button ( ✖ ).

- To save changes click the Ok button ( ✔ ) or press the 'Apply' button ( ❯ ).

## 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 a **Tab** entitled Data Sources.

- 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 Ok button ( ✔ ) or press the '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:

    o mssql – Microsoft SQL Server.
    o mysql – MySQL.
    o oracle – Oracle.
    o postgres – PostgreSQL.
    o db2 – IBM db2.
    o informix – IBM Informix.

- o websvc – Web Service. For more information see **Web Services**.
- o assembly - .NET Assembly dll. For more information see **.NET Assemblies**.
- o file – XML or Excel file. For more information see **Excel and XML Files**.
- o msolap – OLAP. For more information **OLAP and MDX Queries.**
- o odbc – ODBC Driver. For more information see ODBC drivers.

- **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:

  - o mssql, oracle, postgres, mysql and olap – Please refer to ConnectionStrings.com for database connection strings.
  - o websvc – Can take up to four parameters but only requires url.
    - ▪ url – The url of the web service.
    - ▪ Authentication (optional) – Set to 'basic' to utilize basic authentication through IIS. This will send the userid and password as clear text (unless https is used).
    - ▪ uid (optional) – User id is passed to the web service.
    - ▪ pwd (optional) – Password is passed to the web service
  - o assembly – Requires two parameters.
    - ▪ assembly – The full path of the assembly name.
    - ▪ class – The class name in the assembly where the static methods will be obtained.
  - o file – Requires the physical path to the excel or xml file and the file type. Ex. File=C:\example.xls;Type=excel;

Click the green check mark to verify the connection succeeds ( ✓ ).

## Data Source Drivers

Below is a lists of 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 - **http://www.oracle.com/technetwork/database/windows/downloads/index-090165.html**

- **MySQL/MariaDB** – dcmysqlfree.exe – Devart Connector - **http://www.devart.com/dotconnect/mysql/download.html**

- **PostgreSQL** – dcpostgresqlfree.exe – Devart Connector - **http://www.devart.com/dotconnect/postgresql/download.html**

- **DB2/Informix** – ibm_data_server_driver_package_win64_v10.5.exe or newer – IBM Data Server Driver Package –

**https://www14.software.ibm.com/webapp/iwm/web/reg/download.do?source=swg-idsdpds&lang=en_US&S_PKG=win_64&cp=UTF-8&dlmethod=http**

## 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 a 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 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 green check mark 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.

### .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.

```csharp
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
        }
    }
}
```

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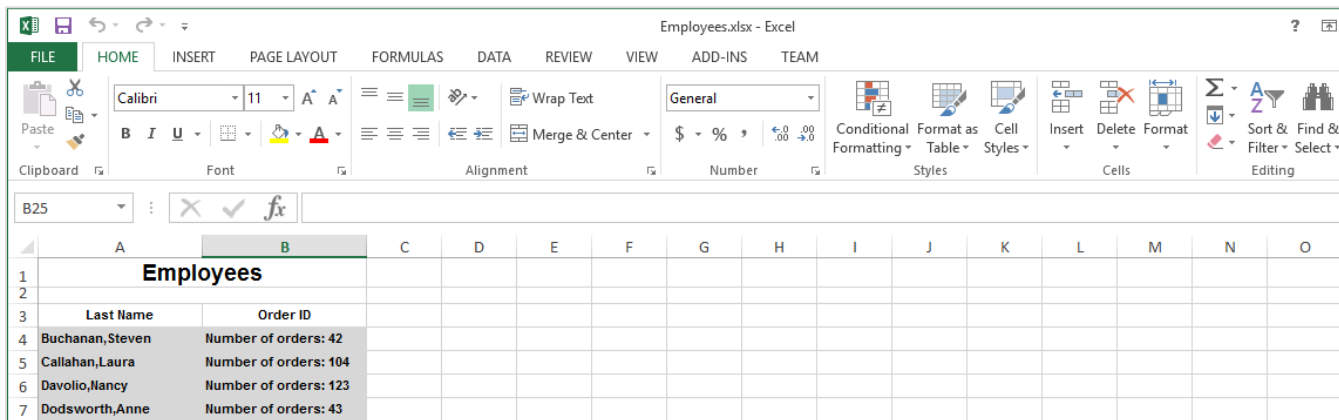
- Methods do not need to be static
- Methods must return a serialized XML string. The returned XML must follow the structure used by the C# method DataSet.GetXML. An example of XML format can be found in the following section.

## 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.

### 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.

## 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
<ExagoData>
  <xs:schema id="ExagoData" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <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:element>
  </xs:schema>

  <Call>
    <CallID>890</CallID>
    <StaffID>134</StaffID>
    <VehicleUsed>12</VehicleUsed>
</ Call >
```

```xml
< 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 regular data base type object with the following exceptions.

- OLAP Data Objects will always be MDX Queries written in the **Custom SQL Object** menu.

- 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 Delimiter(s). Leaving this setting blank will not surround tables, views or other objects with any delimiter. Setting a delimiter will prevent issues if you have spaces in your object names.

## 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 you are adding or editing 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.

- **Type** – the type of parameter being used.

- **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.

- **Hidden** – check hidden 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).

    o **Stored Procedure Parameters** – a list of preexisting Exago parameters to be used as variables for a selected stored procedure.

    o **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.

- o **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.

- o **Display Type** – the display value data type.

## 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 the Data Objects are adding or editing will be displayed in a **Tab** entitled Objects.

- 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 Ok button ( ✔ ) or press the 'Apply' button ( ❯ ).

Each Data Object has the following properties:

- **Name** – Select the Data Object's Source from the first dropdown. In the second dropdown select a Data Object.

  NOTE. This will display all the of the Source's tables, views, methods, stored procedures and functions.

  - o To add custom SQL click the 'Add Custom SQL' button (🔲) next to the Data Object dropdown. For more details see **Custom SQL Objects**.

    NOTE. The name of tables or views should not contain commas.

- **Alias** – the user friendly name for the Data Object. The alias will be displayed to end-users.

  - o NOTE. Alias should not contain the characters '@', '{', '[', or ','.

- **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 **Using Data Object Ids**.

- **Parameters** – parameters that are passed to stored procedures, table functions, Web Services or .NET assembly methods. Clicking in the dropdown will bring up a menu. Click the add button (**+**) and select the parameter from the dropdown list. For more information see **Parameters**, **Stored Procedures** and **Web Services & .NET Assemblies.**

   - o   NOTE. 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.

   - o   This setting is used to filter data when multiple users' information is held within the same table or view and a column(s) holds information identifying each user. Exago will only retrieve the rows where the column value(s) matches the corresponding parameter(s).

- **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:

   - o   **Default –** Follow the global Schema Access Type setting in **Other Settings**.

   - o   **Datasource –** Queries the Data Source for the schema.

   - o   **Metadata –** Reads the schema from the stored metadata.

   NOTE. For more information see **Note on Retrieving Data Object Schemas**.

- **Filter Dropdown Object –** Specify an alternative Data Object to be queried when a user clicks the value dropdown 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.

## Stored Procedures

Stored Procedures offer the ability to use high level code to modify the data set before it is sent to Exago. It is important to note that 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 Object 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.

```sql
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)

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 @sql = 'select ' + @columnStr + ' from vw_webrpt_person where ' +
@filterStr + ' order by ' + @sortStr
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 + ' and ' + @fullFilterStr + ' order by ' +
@columnInfo
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 Data Object menu under Functions. Exago handles table value functions similar to views and tables except it will pass any parameters set in the **Data Object Tab** or in the **Programmable Object Settings**.

## 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 Data Object click the 'Custom SQL' button (▦) and a dialog 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 ( Test ✓ ) to verify that the SQL statement is correct.

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



**Data Object Macros**

'Macros' can be embedded in Custom SQL Data 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 | Will include the trueCondition if a user is executing a report. Will include the falseCondition otherwise. |
|---|---|
| Example | select * from vw_webrpt_optionee IfExecuteMode("where [State] = 'CT'","") |

IfExistReportDataObject' (**string** dataObjectName, **string** trueCondition, **string** falseCondition)

| Description | Will include the trueCondition if dataObjectName exists inside the full Exago SQL statement to the data source. Will include the falseCondition otherwise. |
|---|---|
| Example | select * from vw_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.

The following properties of each column can be modified:

- **Column Alias –** The name of the Data Field that the end-users see.

- **Data Type –** The type of data Exago should treat the Data Field as (ex. DateTime).

  o Valid values for Data Type include: String, Date, Datetime, Time, Int, Decimal, Image, Float, Boolean, and Guid.

- **Filterable –** If set to False the Data Filed will not be listed in the Filters menu.

- **Visible –** If set to False the Data Field will not be listed for users.


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 dropdowns to set the desired properties.

Click the 'Read Schema' button ( Read Schema ) to quickly create column metadata for each column in the Data Object.

To remove Column Metadata for a column select it in the right panel and click the delete button ( ✖ )

To save changes to Colum Metadata, click the 'Ok' button ( ✔ ). Click the 'Cancel' button to discard changes.

## 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. Then Exago can 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 retitled.

For Exago to retrieve schema information from Metadata:

1. In **Other Settings**, set 'Schema Access Type' to 'Metadata'. This will force Exago to get all schema information from 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**.

a. Click the 'Read Schema' button ( ⊞ Read Schema ). A message will appear asking you to confirm you want to continue. Click Ok.

b. Click Ok to close the Column Metadata Menu.

c. Press Apply or Ok 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, 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 Data 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. (see example below)

- For each Data Object:

    o Select 'Object' in the **Main Menu** and click the 'add' button

    o Select the single Service/Assembly/Procedure

    o Provide an Alias and an Id for the Object

    o Select the key columns

    o Click Apply or Ok to save the Object.

Ex. This stored procedure uses the Object Id Parameter (@objectID) to return different data/schema information for different Object Ids.

```sql
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
```

### 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

Joins specify to Exago the relationship between Data Objects.

All existing Joins are listed in the **Main Menu** under Data. All the joins you are adding or editing will be displayed in a **Tab** entitled Joins.

- To add a new join click 'Joins' in the Main Menu then click the add button ( ).

  NOTE. Joins can be added quickly using **Automatic Database Discovery**.

- To edit a join either double click it or select it and click the edit button ( ).

- To delete a join select it and click the delete button ( ).

- To save changes and new join click the Ok button ( ).

Each join has the following properties:

- **From Object** –the first Data Objects you would like to join.

- **To Object** – the other Data Object you would like to join.

o NOTE. The order of the Data Objects is important if you have a one-to-many relation type or a Left/Right Outer Join type. See below for details.

- **Join Column(s)** – specify the field(s) of each Data Object that must match to join an entity in the From Object to an entity(s) in the To Object.

- **Join Type** – specify whether rows from either Data Object that do not have a match should or should not be included.

  o Inner: only includes rows of the From Object that have a match in the To Object and vice versa.

  o Left Outer: includes rows of the From Object that do not have a match in the To Object but not vice versa.

  o Right Outer: includes rows of the To Object that do not have a match in the From Object but not vice versa.

  o Full Outer: includes rows in either Data Object that do not have a match.

- **Relationship Type** – specify if the join type is one-to-one or one-to-many.

  o One-to-One: Each row in the From Object can join to at most one row from the To Object.

  o One-to-Many: Each row in the From Object can join to any number of rows from the To Object.

- **Weight** – give a join weight in order to set its precedence when multiple join paths exist between Data Objects. The path with the higher weight will be utilized.

  o Ex. A report contains three Data Objects 'Students', 'Professors' and 'Comp Sci 101.' Students is joined to 'Professors' and 'Comp Sci 101.' Additionally 'Professors' is joined to 'Comp Sci 101.' There are two available join paths between 'Students' and 'Comp Sci 101.' Adding weight to a join will clarify which of the two paths Exago should use.
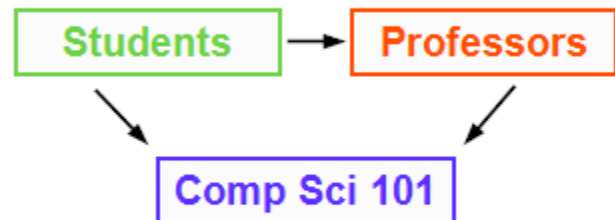


## Modifying Joins

Although joins are created in the Administration Console they are saved within each individual report. For Join changes in the Administration Console to take effect edit the report and use the 'Recreate' button in the Advanced Options menu. For instructions on how to access the Advanced Options please see the User Guide.

IMPORTANT. It is important to make sure that all of the joins are set to your desired specifications in the Administration Console before you begin building numerous reports.

### Note About Cross Source Joins

Data Objects from different Data Sources can be joined in Exago. Because the Data Objects come from distinct databases they must be joined through code by Exago. Though Exago strives for efficiency **this process may be memory intensive** for large data sets.

## 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 database type sources: 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 comepleted individually in a new tab entitled 'Incomplete Objects'.

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

Q Northwind ×

☑ Select All Complete Objects    ☑ Select All Objects    ☐ Clear Selections

❯ Add Objects    ❯ Add Objects and Joins    ☐ Preview Only

∨ Northwind
  ∨ Tables
    🔑 ☐ dbo.Categories
    ⤐ ☐ dbo.CustomerCustomerDemo
    🔑 ☐ dbo.CustomerDemographics
    🔑 ☐ dbo.Customers
    ⤐ ☐ dbo.Employees
    ⤐ ☐ dbo.EmployeeTerritories
    ⤐ ☐ dbo.OrderDetails
    ⤐ ☐ dbo.Orders
    ⤐ ☐ dbo.Products
    🔑 ☐ dbo.Region
    🔑 ☐ dbo.Shippers
    🔑 ☐ dbo.Suppliers
    ⤐ ☐ dbo.Territories
  ❯ Views
  ❯ Functions
  ❯ Procedures

Total Complete Objects added: 0

Total Incomplete Objects found: 0

Total Objects skipped: 0

Total Joins added: 0

Total Joins skipped: 0

✕ Close

# General

This chapter details the available settings to enable, disable and modify various features of Exago. Settings made in General will be set for all users unless specifically overwritten in **Roles**.
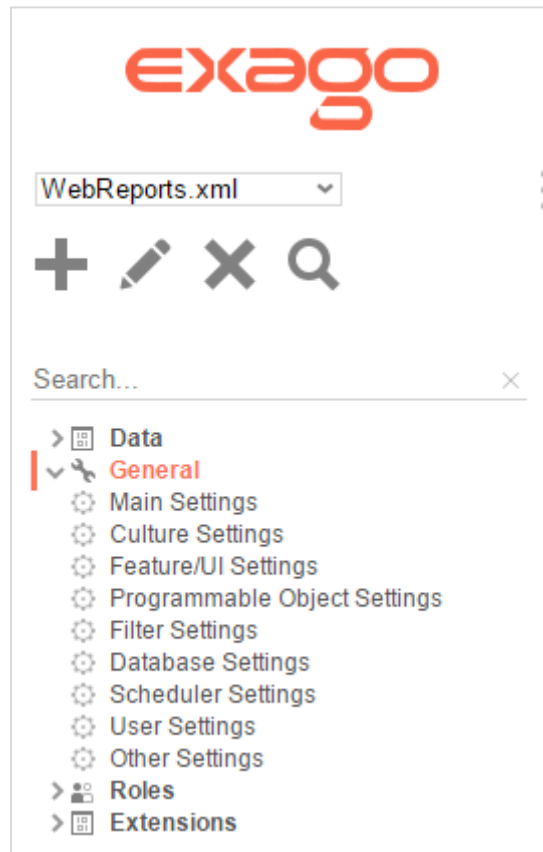
To edit any of the settings double click the category or select it and click the edit button. (  )

## Main Settings

The main settings are the basic options for Exago.
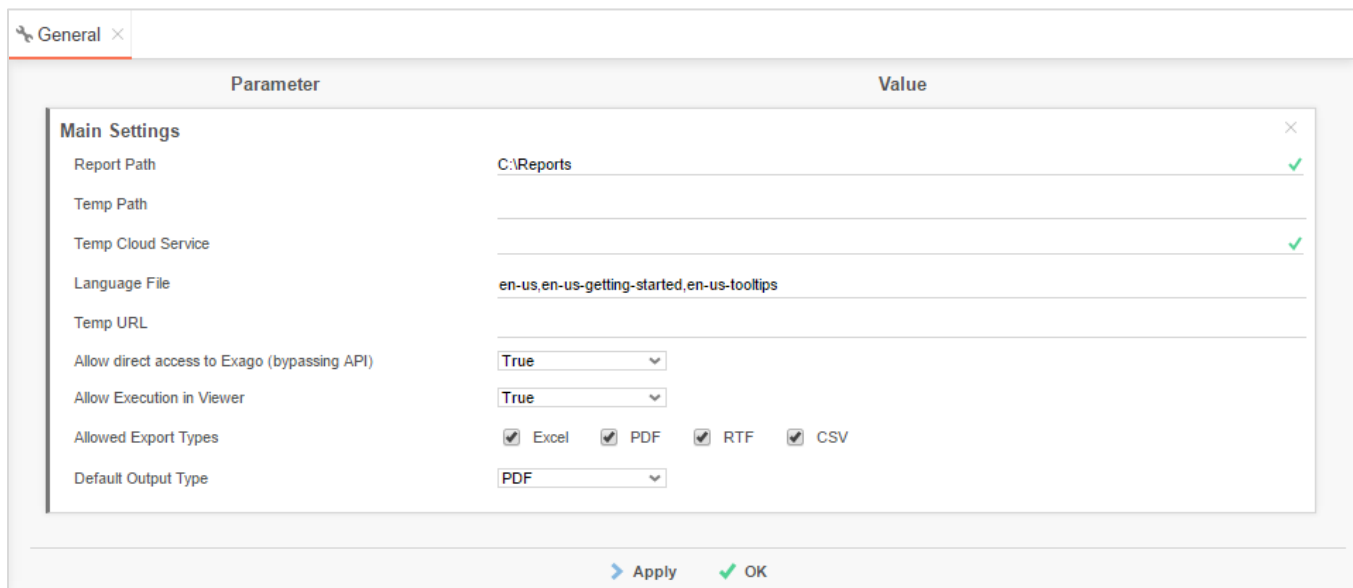


The following settings are available:

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

  o **Virtual Path**

  o **Absolute Path –** used to provide increased security (ex. C:\Reports)

  o **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:

  o **Blank –** All temp files and images will be saved to /Temp.

  o **Virtual Path**

  o **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 reports that are run in the Report Viewer. Temp URL can override just the scheme (i.e. https) or the full URL.

- **Allow direct access to Exago –** A boolean setting:

  o **True –** allows users direct access to Exago with no security.

  o **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.'

  IMPORTANT. We **highly recommend setting this to False** before deploying Exago in a production environment.

- **Allow Execution in Viewer –** Enables or disables running reports in the Report Viewer.

- **Allowed Export Types –** The available formats for exporting all reports. Check the box of the formats that should be available.

- **Default Output Type –** The 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.

## 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.

- **Numeric Decimal Places –** Default number of decimal places for numeric 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.

- **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.

## Features/UI Settings

The Features/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.

&#x2692; General   ×

| Parameter | Value |
|---|---|

**Available Report Types**    ×

| | |
|---|---|
| Allow Creation/Editing of Express Reports | True |
| Allow Creation/Editing of Standard Reports | True |
| Allow Creation/Editing of Crosstabs | True |
| Allow Creation/Editing of Dashboards | True |
| Allow Creation/Editing of Chained Reports | True |

**Express Report Designer Settings**

| | |
|---|---|
| Show Styling Toolbar | True |
| Show Themes | True |
| Show Grouping | True |
| Show Formula Button | True |

**Standard Report Designer Settings**

| | |
|---|---|
| Show Chart Wizard | True |
| Chart Colors | #f54c45,#ddcc1c,#a8d060,#1297aa,#7c449d,#5b7fd5 |
| Maximum Number of Chart Data Points | 300 |
| Default Chart Font | Arial |
| Show Map Wizard | True |
| Map Colors | Lightblue,Navy |
| Show Gauge Wizard | True |
| Gauge Colors | #ff0000, #ffff00, #00ff00 |
| Show Document Template | True |
| Show Document Template Upload Button | True |
| Show Linked Report | True |
| Show Linked Action | False |
| Show Insert Image | True |
| Show Joins Window | True |
| Show Advanced Joins | True |
| Show Events Window | False |
| Show Linked Reports in New Tab | False |

| | |
|---|---|
| Show Group Headers Formula Button | True |

**Dashboard Report Designer Settings**

| | |
|---|---|
| Prompt user for Parameters/Filters on Execution | False |
| Show URL Item Button | True |
| Allow Creation/Editing of Dashboard Visualizations | True |
| Show Data Fields Search Box | True |
| Use Sample Data for Dashboard Visualization Design | True |
| Visualization Database Row Limit | 0 |

**Common Settings**

| | |
|---|---|
| Default Designer Font | Arial |
| Default Designer Font Size | 8 |
| Show Help Button | True |
| Custom Help Source | |
| Show Exports in Tab | True |
| Show IE Download Button | False |
| Show Join Fields | True |
| Show Grid Lines in Report Viewer | True |
| Show Enhanced Tooltips | True |
| Save on Report Execution | True |
| Save on Finish Press | True |
| Enable Right-Click Menus | True |
| Enable Reports Tree Drag And Drop | True |
| Show Report Upload/Download Options | False |
| Allow interactivity in report viewer | True |
| Show Toolbar in Report Viewer | Auto |
| Default interactive report viewer dock is open | False |
| Interactive report viewer default dock placement | Left |
| Allow save to report design for from the report viewer | True |

> Apply    ✔ OK

## Available Report Types

These settings enable/disable report types.

- **Show Express Reports –** Displays/Hides the Express Report Wizard.

- **Show Standard Reports –** Displays/Hides the Standard Report Wizard and Report Designer.

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

- **Show Crosstab Reports** – Displays/Hides the Crosstab Report Wizard and Insert Crosstab button in the Report Designer.

- **Show Dashboard Reports** – Displays/Hides the Create New Dashboard button.

## Express Report Designer Settings

These settings only apply to the Express Report Wizard.
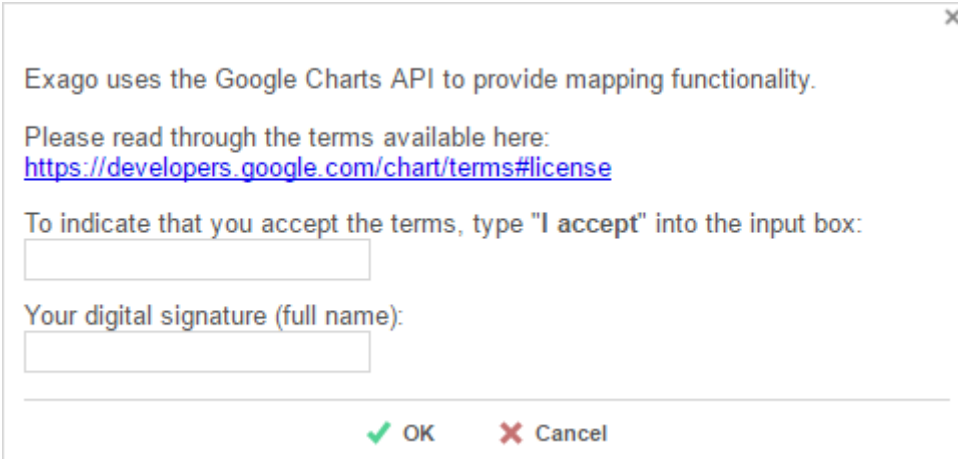
- **Show Styling Toolbar** – Displays/Hides the styling tools in the Layout tab of the Express Report Wizard.

- **Show Themes** – Displays/Hides the Theme dropdown 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.

## Standard Report Designer Settings

These settings only apply to the Report Designer.

- **Show Chart Wizard –** Displays/Hides the Insert Chart button in the Report Designer. Set to False to disable users from creating or editing charts.

- **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 Map Wizard –** Displays/Hides the Insert Map button in the Report Designer. Set to False to disable users from creating or editing maps.

  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.

  

- **Show Gauge Wizard –** Displays/Hides the Insert Gauge button in the Report Designer. Set to False to disable users from creating or editing gauges.

- **Chart Colors –** Lists the values used for default chart colors. Hexadecimal values should be separated by commas (or semicolons).

- **Map Colors –** List the values used for default map colors. Hexadecimal values or css color names should be separated by commas (or semicolons).

- **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 –** Displays/Hides 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 Reports Button –** Displays/Hides the Linked Report button in the Report Designer. Set to False to disable users from creating Linked Reports.

- **Show Insert Image Button –** Displays/Hides the Insert Image button in the Report Designer. Set to False to disable users from inserting images.

- **Show Advanced Window –** Displays/Hides the Joins Menu under Advanced. Set to False to disable users from modifying joins.

- **Show Advanced Joins –** Displays/Hides additional options in the Joins Menu. Set to True to enable advanced users to apply Event Handlers for the report. See **Server Events** for more information.

- **Show Events Window –** Displays/Hides the Events Menu under Advanced. Set to True to enable advanced users to create, delete, and modify joins.

- **Show Linked Reports in New Tab –** 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.

- **Show Group Headers Formula Button –** Displays/Hides the Formula Editor Button in the Group Header/Footer Menu. Set to False to disable users from grouping on formulas.

## 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.

- **Show Data Fields Search Box** – Display/Hide the search bar above the Data Fields of the Dashboard Designer.

  NOTE. The search will require the schema of all available Data Objects. For best performance, only set this to True if using Column Metadata for Schema Access type. See **Retrieving Data Object Schemas** for more details.

- **Visualization Database Row Limit** – 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.

- **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 queiry the Data Source for each change made while editing Dashboard Visualizations.
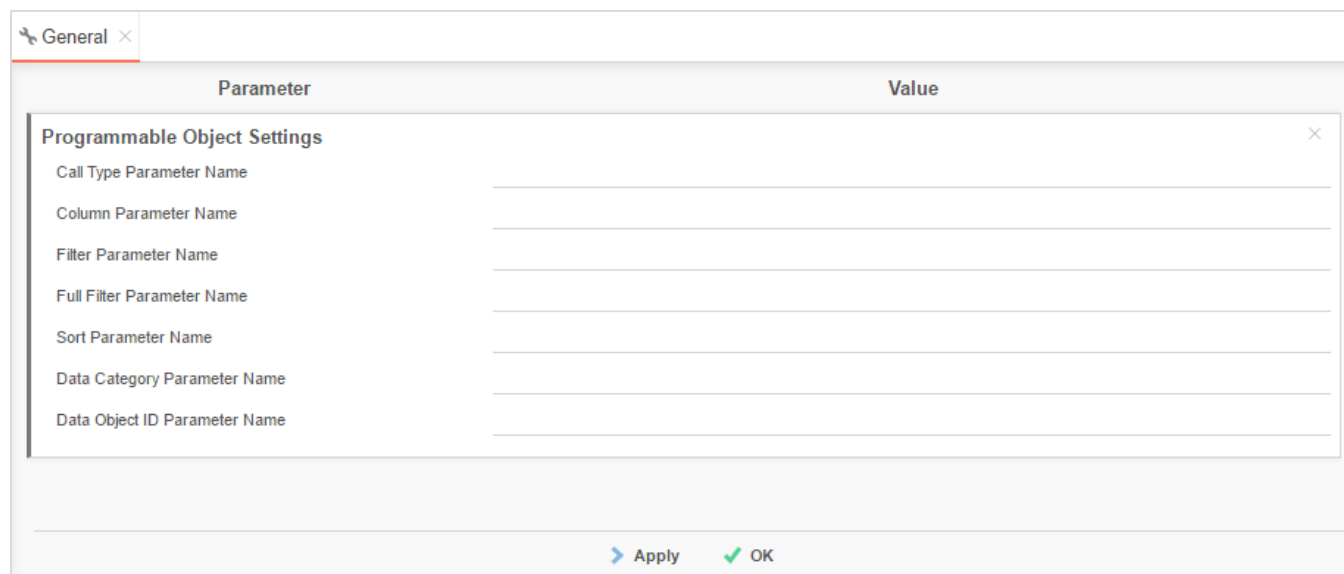
## Common Settings

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

- **Default Designer Font Size –** Specifies a default font size for reports created in the Standard Report Wizard, Express Report Wizard, Standard Report Designer, and Dashboard Designer. This setting can be overridden on a per-Report basis. Does not apply to CrossTabs.

- **Show Help Button –** Displays/Hides 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 –** Displays/Hides 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.

- **Enable Right-Click Menus –** Set to False to disable right click menus Exago.

- **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.

    o **Auto** - Exago will detect if the report only displays a single page of content from the Report Footer Section. If so the Toolbar will be hidden, otherwise it will show.

    o **Show** – The toolbar will always show.

    o **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.

## 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 which will increase performance and reduce server resources. For more information on these types of Data Objects see **Web Services & .NET Assemblies**.



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** – returns 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 -** Name of the column being requested by the user. Only this column needs to be returned to Exago.

- **Filter Parameter Name –**

  - **CallType = 1:** The filter string specific to the Data Object being called passed as standard SQL.

  - **CallType = 2:** The current value of the filter whose dropdown is being requested.

- **Full Filter Parameter Name –**

  - **CallType = 1:** The filter string for the entire report passed as standard SQL.

  - **CallType = 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**.

## Filter Settings

The Filter Settings provide control over what filter options are exposed to users and how dropdowns in filters behave.

Names for the following Parameters can be set:

- **Show Group (Min/Max) Filters** – Displays/Hides the Min/Max Filter menu. Set to False to disable users from using Min/Max filters.

- **Allow New Filters at Execution** – Controls the creation of new filters when a user is prompted for a filter value at the time of report execution. Set to False to disable new filters from being created at execution.

- **Read Database for Filter Values** – Enables/Disables filter dropdowns to contain values from the database. Set to false only if retrieving values for the dropdown will take more than a couple of seconds.

- **Allow Filter Dependencies** – Causes filter dropdowns to retrieve values dependent on the filters above them in the menu. Set to True to enable.

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

    o NOTE. Dropdowns 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 –** Species the type of filter execution window to new reports should use by default.

    o **Standard –** New reports display the standard filter execution window.

    o **Simple with Operator –** New reports display a simplified filter execution window that only allows the operator and value to be changed.

- o **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.

## 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.

| Parameter | Value |
|---|---|
| **General** ✕ | |
| **Database Settings** | ✕ |
| Database Timeout | 600 |
| Database Row Limit | 0 |
| Disable Non-Joined Data Objects | True |
| Enable Special Cartesian Processing | True |

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 a query. This only applies to Tables, Views and Functions. Set to '0' to return all rows.

- **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 on-to-many Joins will cause special processing to avoid data repeating on the report. Set to False to disable this behavior.

### 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 **http://msdn.microsoft.com/en-us/library/12kxkt25(v=vs.80).aspx** 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. Ex. "@database@" will be replaces 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 the Scheduler Service**.

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.

- **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.

- **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** – Displays/Hides options in the Recurrence tab to have a schedule repeat throughout the day it is scheduled.

- **Show Schedule Intraday Recurrence Option –** Displays/Hides options in the New Schedule Wizard to have the schedule repeat throughout the day it is run. Set to False to disable users having schedules repeate during its execution day.

- **Scheduler Manager User View Level –** Controls what information each user can see in the Schedule Manager. These levels utilize the Parameters companyId and userId. There are three possible values:

  o **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.

  o **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.

  o **All Users in All Companies:** User can view and delete report schedules for all companies (administrator).

- **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 an External Repository**.

- **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.

- **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 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 dropdowns, 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.

- **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:

  o **A: requires an upper case letter for each 'A'.**
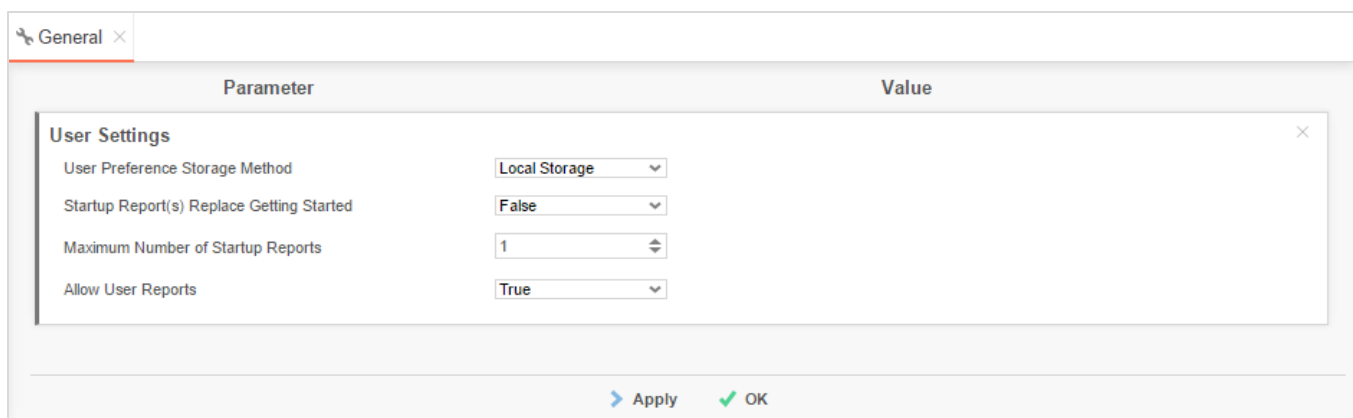
- o **a: requires a lower case letter for each 'a'.**

- o **n: requires a numeric character for each 'n'**

- o **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.

## User Settings

The User Settings give administrators choices about how to store and utilize users' preferences such as which Dashboards and/or Reports to execute when they enter Exago.

| Parameter | Value |
| --- | --- |
| **User Settings** | |
| User Preference Storage Method | Local Storage |
| Startup Report(s) Replace Getting Started | False |
| Maximum Number of Startup Reports | 1 |
| Allow User Reports | True |
| | > Apply    ✓ OK |

The following User Settings are available:

- **User Preference Storage Method –** How to store User Preferences such as which Dashboards and/or Reports to execute when a user enters Exago. There are two possible values:

  - o **None** – Users will not have the ability modify User Preference features. The User Preferneces button will be hidden.

  - o **Cookie –** User Preferences are stored in the browsers cookie. This is the default behavior as it does not require any additional setup. However a user's preferences will not be carried over to other machines or browsers and will be lost if the user deletes their browser's cookies.

    NOTE. A user is identified by the values of the Paramters userId and companyId.

  - o **External Interface –** User Preferences are stored and retrieved via the **External Interface**. This requires the host application to implement the methods **SetUserPreference** and **GetUserPreferences** in the External Interface but User Preferences will be preserved for a user across browsers and machines.

- o **Server Events –** User Preferences are stored and retrieved via **Server Events**. This requires the global events **onGetUserPreferences** and **onSetUserPreferences** must be implemented.

- **Startup Report(s) Replace Getting Started –** Set to False to display both the **Getting Started Content** and any Dashboard and/or Reports that were set to run at startup. Set to True to hide the Getting Started Content if any Dashboards and/or Reports execute when a user enters Exago.

- **Maximum Number of Startup Reports –** Sets the number of Dashboards and/or Reports that can be executed when a user enters Exago.

- **Allow User Reports –** Set to False to prevent users from saving changes made in the Interactive Report Viewer as User Reports. Set to True to allow changes in the Interactive Report Viewer to be stored as User Reports and applied to future report executions.

## Other

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.

- **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 Report Viewer** – Controls when report data is sent to the client. Set to true to send data as each page is requested (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** – Set to True to write a log file for debugging purposes. For more information see **Read the Log File**.

- **Enable Debugging** – Set to True to enable debugging. For more information see **Manually Creating a Debug Package**.

- **Maximum Age for Temp Files** – Maximum number of minutes a temp file can exist before Exago's automatic cleanup of temp files will remove it. It is important to understand that setting the maximum age too low may cause an error as users might spend some time viewing a report which uses AJAX to read temp paging files. The default value is 1440 minutes (1day). 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.

- **Schema Access Type** – Specifies whether to query the Data Source for an Object's schema or to read it from Column Metadata. See **Note on Retrieving Data Object Schemas** for more information.

- **Allow Multiple Sessions** – Allows multiple sessions of Exago per user. Set to True to enable this behavior.

- **'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.

- **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** – User Id to gain Access to the Administration Console. Leave blank to permit unverified access to the Administration Console.

- **Password** – Used in conjunction with User ID to gain access to the Administration Console.

- **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 me 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.

# Roles

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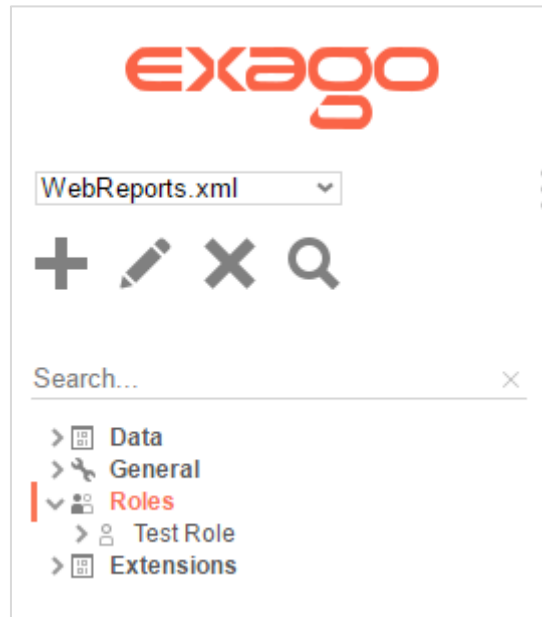


- To add a new role select 'Roles' in the Main Menu then click the Add button (  ).

- 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 the users access. For more information see chapters **Integration** and **API**.

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

**Main** – controls the broad properties of the Role.
**General** – overrides General Settings.
**Folders** – controls which report folders a role can see and edit.
**Objects** – controls which Data Objects a role can access.
**Filters** – provides row level filters on Data Objects.
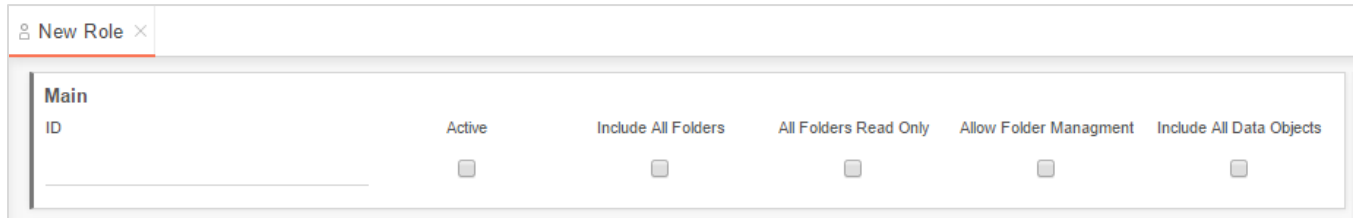
## Creating Roles

To create a Role click 'Roles in the Main Menu and click the Add button (✛). This will open the Main Section.

## Main Settings

The main settings control the broad properties of the Role.



The Main settings control:

**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**.

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.asmx'. 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 visit **http://msdn.microsoft.com/en-us/library/8kb3ddd4%28v=vs.71%29.aspx.**

- **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 Grid Lines in Report Viewer –** Sets the default value option when running a report to 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 dropdown 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 Standard Reports –** Displays/Hides the Standard Report Wizard and Report Designer.

  NOTE. If 'Show Standard Reports' is False then attempts to edit Standard 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 dropdowns to contain values from the database. Set to false only if retrieving values for the dropdown 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:

  o **Current User:** Can only view and delete report jobs that have been created by that user.

  o **All Users in Current Company:** User can only view and delete report schedules for their company.

  o **All Users in All Companies:** User can view and delete report schedules for all companies (administrator).

## Folder Access

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



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 Add ( ✚ Add ).
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 (✖).

## Object Access

The Objects Access controls which Data Objects are accessible to the Role. A report can only be run if the Role has access to all the Data Objects on the report.
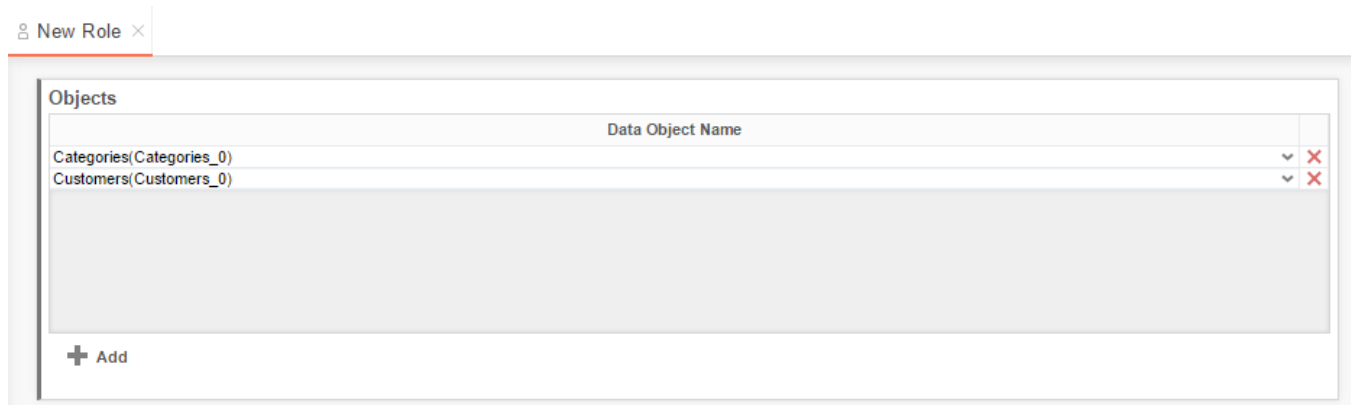
NOTE. If **Include All Data Objects** is checked this list will deny access to the Data Objects added, if unchecked the list will allow access to the Data Objects added.

To add a Data Object click Add ( + Add ).
Click in the Data Object Name column and select the Object you want to add.
To delete an Object click the delete button (✕).

## Filters Access

The Filter Access provides a means to filter a Data Object by Role.
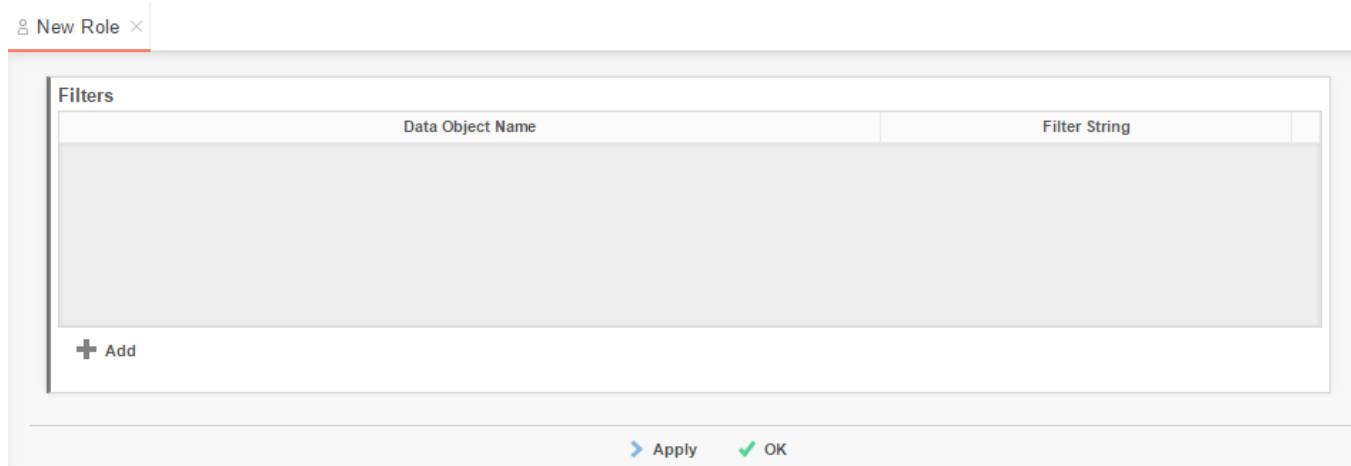


To add a Data Object click Add ( + Add ).
Click in the Data Object Name column and select the Object you want to add.
Enter the filter string in the Filter String Column. The filter string should be Standard SQL. This string will be added to the Where clause.
To delete a Data Object click the delete button (✕).

# Extensions

This chapter explains how to utilize several of Exagos Extensions, which permit the host application to extend the capabilities and behaviors of Exago.

There are several types of Extensions offered:

- **Functions** – Custom code that can be used as formulas inside or reports and dashboards.

- **Filter Functions** – Custom code to provide filters with dynamic values that are updated each time the report is executed.

- **Server Events** – Events in the Exago runtime where custom code handlers can determine the application's behavior.

- **Custom Options** – Place holders for values that a user can specify while editing a report. These values can then be utilized by other extensions such as custom functions or server events.
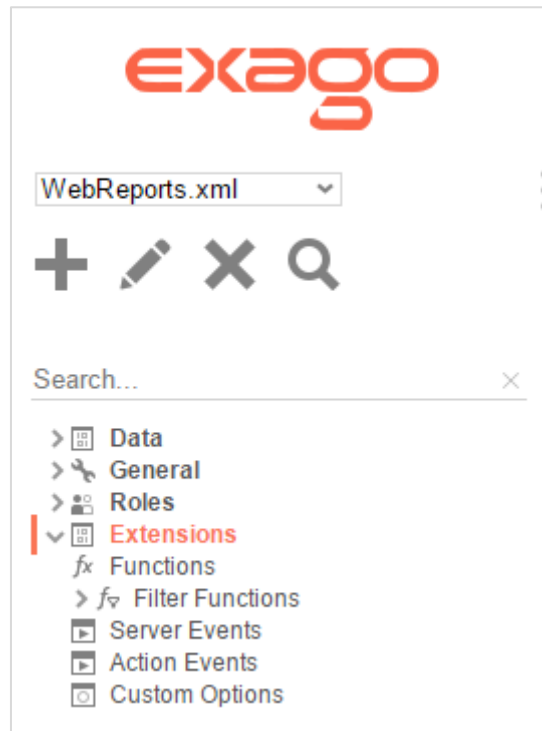


## Functions

Exago comes with a large number of predefined functions that can be used to make formulas in the Formula Editor. As an administrator you may create additional custom functions using high level coding languages. Custom functions will be accessible to users in the Formula Editor or by typing their name into a cell of a report. Functions can be added to a preexisting function category or a function can be put into a new custom category.

Functions can be written in C#, JavaScript or VB. Net. Functions can take as few or many arguments as inputs, provided that the max number of arguments is greater than or equal to the minimum number of arguments.

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 Functions

To create a custom function, select 'Functions' in the Main Menu 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, if the description matches the id of any element in the language files then the string of that language element will be used instead of the description. For more information see **Multi-Language Support**.

- **Minimum Number of Arguments –** The minimum number of values that an end user must enter in the function separated by commas.

- **Maximum Number of Arguments -** The maximum number of values that an end user may enter in the function separated by commas.

  > NOTE. Arguments are passed to your code as an array of generic objects so there can be as many arguments as desired. The argument array is accessed by args[ ].Arguments are passed into the function as objects.

- **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 tells Exago to create a new Category with the specified name.

  > NOTE. To support multi-language functionality, if the custom category matches the id of any element in the language files then the string of that language element will be used instead of the description. For more information see **Multi-Language Support**.

- **Language –** The high-level language of the code for the function. May be C#, JavaScript or VB.Net.

- **Reference –** A semicolon-separated list of any dlls that need to be referenced by the Custom 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 Custom Function. 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.

## Exago Session Info

Custom Functions 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.

NOTE. sessionInfo can also be accessed by Server Events and Action Events. For more information see **Server Events** and **Action Events**.

The second **example** in the next section 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.

The following properties are available:

- **PageInfo** – this is the parent of all information in the current session. This includes the active Report and SetupData objects.

    NOTE. Since the Report and SetupData objects are accessed frequently, direct pointers are include 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** – contains the value specified by the companyId **Parameter**.

- **UserId** – contains the value specified by the userId **Parameter**.

The following methods are available:

- **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 specigied Parameter Object. GetParameter first looks in the Report Parameter collection, parameters beign utilized by the report, and then in the Config Parameter collection, 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 Exago Functions

Cases may arise where you want to call an existing function within your Custom Function. Using the class CellFormula and returninging the method CellFormula.Evaluate(). An example of this is provided at the end of the **Example** section below.

## Example

The following are two examples of Custom Functions.

**Name** – ReverseString
**Description** – Reverses characters in the input string
- **Minimum Number of Arguments** – 1
- **Maximum Number of Arguments –** 1
- **Language –** C#
- **Category –** Other
- **Program Code –**

```csharp
string inputString = args[0].ToString();
char[] inputChars = inputString.ToCharArray();
System.Text.StringBuilder reverseStringSb = new System.Text.StringBuilder("");

for (int i = inputChars.Length - 1; i &gt;= 0; i--)
{
    reverseStringSb.Append(inputChars[i]);
}

return reverseStringSb.ToString();
```

**Name** – LineNumber
**Description** – Displays the number of the line of the report.
- **Minimum Number of Arguments** – 0
- **Maximum Number of Arguments –** 0
- **Language –** C#
- **Category –** Other
- **Program Code –**

```csharp
// 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;
```

**Name** – BarCode
**Description** – Transforms the input into an image of a barcode.
- **Minimum Number of Arguments** – 1
- **Maximum Number of Arguments –** 1
- **Language –** C#
- **Category –** Other
- **References –** WebReports.Api.Reports;(A dll to create barcodes)
- **Program Code –**

```csharp
//Get argumentstring
inputField = args[0].ToString();
// Call custom code to get image as a bit string
filenamestring imageFilename = null;
// create formula text
string formulaText = String.Format("=LoadImage(\"{0}\")", imageFilename);
// create embedded formula for LoadImage function
CellFormula formula = CellFormula.CreateFormula(sessionInfo.PageInfo, formulaText,
CellVariableCollectionFilter.DataField);
// evaulate and return result
return formula.Evaluate(null);
```

## Filter Functions

This chapter explains how to create 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.
    > **Ex**. For the **example** function below you could create a language id 'Begining_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** – The data type of filters the fitler function should be available for.

- **List Order** – The order the filter function will appear amongst 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 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.

## Example

The following is an example of a Custom Function.

- **Name** – Begining_of_Month
- **Language –** C#
- **Program Code –**

```csharp
// 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;
```

## Server Events

In an effort to meet the reporting needs of unique and varied environments, Exago offers numerous **extensibility features**. As part of this effort Exago makes available certain events during the report execution process. When these events occur, an Event Handler consisting of a .Net Assembly method or custom code snippet can be executed to make impactful changes on the report execution process.

This chapter explains how to create Events Handlers that run custom code when reports are executed.

- 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 report execution process. This code can either come from a .Net Assembly method or within Exago configuration.

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.

  o **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. Press the green check mark to verify the code executes properly ( ✓ ).

  Custom Code has three properties:

  ▪ **Language** – Code can be written in C#, Javascript or VB. Net.

  ▪ **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.

  ▪ **Code** – The code that will be executed by Exago when called.
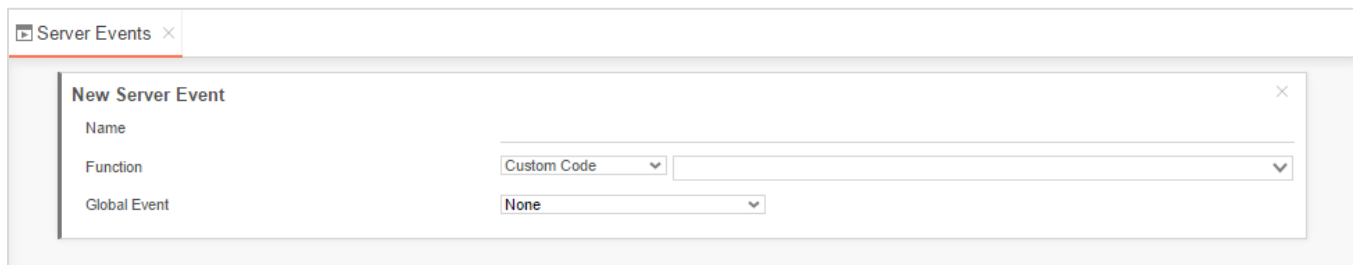
o **.Net Assembly Method** – To utilize a .Net Assembly method first create a .Net Assembly **Data Source**. Select the desired assembly from the first Function dropdown. Clicking on the second dropdown will open a list of available methods.
See **.Net Assemblies** for information on how to access the arguments for each Event.

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 **Session Info**.

NOTE. All methods used as Event Handlers must be static.



- **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.

  o **Specified Event** – The Event Handler will be called when the specified Event happens during the execution of **all** reports.

    Ex. Selecting OnReportExecuteStart from this dropdown will cause the Event Handler to be called whenever any Report Execution begins.

  o **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.

## Custom Code

Event Handler custom code can be saved directly in Exago via the Administration Console. There are two objects that custom code can utilize to access information relevant to an Event.

- **sessionInfo** – Without any special references, all custom code can make use of a **sessionInfo** object that provides access to elements of Exago current session such as parameters, filters, the logger, etc.

- **arguments array** – Custom code can also access an array of input values called args[]. For each Event the content of the args array will be different. The content of this array is detailed in **Full Description of Events**.

## .Net Assemblies

Event Handlers can also reside in .Net Assemblies. The following are important details for using .Net Assemblies as Event Handlers.

- 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.

- 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. For more information see **Available Events**.

## Setting Event Handlers on 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 OnExecuteSqlStatmentConstructed 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.
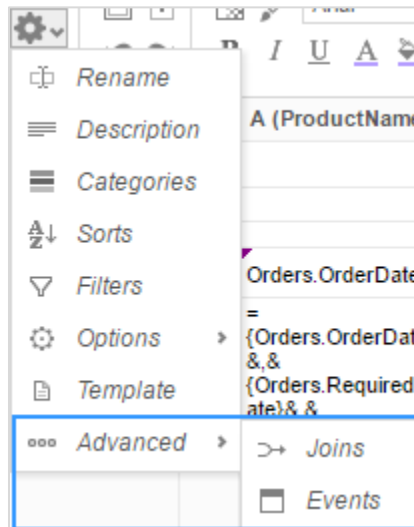
3. In the Event Menu click the Add button ( ➕ Add ).

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.
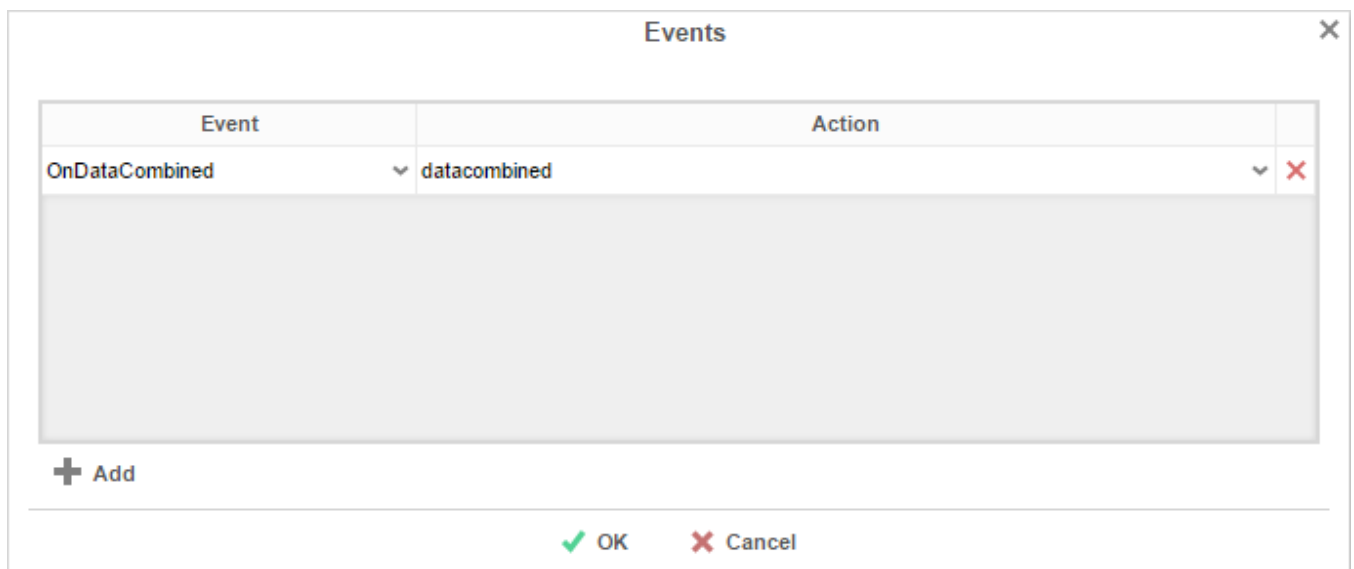


## Displaying User Messages from Server Events

Some Server Events are designed to displays messages to the user based return value. However for the other server events* a user message can be displayed by throwing the following excetion method.

WrUserMessage(string messageOrId, wrUserMessageType Type)

| | |
|---|---|
| **Description** | Displays a message to the user. |
| **Remarks** | wrUserMessageType can either be Text or Id.<br> **Text** – The user message will display the string messageOrId<br> **Id** – The user message will display the string associated with the Id in the Language Files.<br><br>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);      // add any other messages }  return webServiceResult;``` |

NOTE. This cannot be used for the Events **OnConfigLoadStart** or **OnConfigLoadEnd**.

## Quick List of Events

The following Events can be assigned Event Handlers for runtime invocation.

**OnDataCombine** – occurs when data is combined and initially processed; expects a Data Table to be returned.

**OnReportExecuteStart** – occurs when report execution begins; expects a string to be returned to indicate if execution should proceed.

**OnReportExecuteEnd** – occurs when a report execution finishes; any return value will be ignored.

**OnWebServiceExecuteEnd** – occurs when a web service data source returns data; expects an xml string to be returned.

**OnExecuteSqlStatmentConstructed** – occurs before the data source is queried for report execution; expects an SQL string to be returned.

**OnFilterSqlStatmentConstructed** – occurs before the data source is queried to populate the filter dropdown; expects an SQL string to be returned.

**OnOkFiltersDialog** – occurs when Ok is clicked on the Filter Execution Window; expects a string to be returned to indicate if execution should proceed.

**OnOkParametersDialog** – occurs when Ok is clicked on the Parameter Execution Window; expects a string to be returned to indicate if execution should proceed.

**OnScheduledReportExecuteSuccess** – occurs when a scheduled report is executed; expects a Boolean to be returned to indicate if the report should be sent as scheduled or intercepted.

**OnRenameFolderStart** – occurs when a user attempts to rename a folder; expects a string to be returned to indicate if execution should proceed.

**OnRenameFolderEnd** – occurs when a folder has been renamed; any return value will be ignored.

**OnConfigLoadStart** – occurs when the configuration of Exago is initially loaded; expects a void return.

**OnConfigLoadEnd** – occurs after the last Api changes have been made to of Exago' configuration; expects a void return.

**OnDataFieldsRetrieved** – occurs after Data Fields are retrieved from specific Data Objects; expects a Data Table to be returned to indicate how to display the Data Fields.

**OnGetUserPreferences** – called to retrieve user preferences when entering the application and editing/executing reports.

**OnSetUserPreferences** – called to save user preferences when a user specifies startup reports or saves Interactive Report Viewer changes as a user report.

**OnLoadReportParameters** – passes a list of Parameter elements that can be reordered or modified before they are sent to the client for display.


NOTE. For the following descriptions the data type WebReports.Api.Reports.SessionInfo is refered to as SessionInfo. The class System.Data.DataTable is referred to as DataTable.

## Full Description of Events

### OnDataCombined

The OnDataCombined Event allows the inspection and/or modification of the raw data set after retrieval from the Data Sources and initial combining within Exago. A common use of this event is to modify or blank sensitive data fields in a Report depending on the authorizations available to the user executing the report.

### Signature

For custom code the args array is structured as follows:

> args[] contains a single DataTable of the combined data in position zero.

For .Net Assemblies the method signature is as follows:

> DataTable **EventHandlerName**(SessionInfo sessionInfo, DataTable combinedData)

### Expected Return

The OnDataCombined Event expects a DataTable to be returned. The schema of the DataTable must match that of combinedData.

### Notes

In the DataTable, if a Data Object has an **Id** then that will be used as the column names, otherwise the database name will be used. Data Fields will always use their database names despite any **Column Metadata**.

### Example

The following example checks a Parameter called AllowViewSSN and then censors the columns named SocialSecurityNumber.

```
System.Data.DataTable dt = (System.Data.DataTable) args[0];
```

```
if (sessionInfo.GetConfigParameter("AllowViewSSN") == "true" &&
dt.Columns.Contains("Employees.SocialSecurityNumber"))
{
    //change the value of SSN to blank
    foreach (System.Data.DataRow row in dt.Rows)
    {
        for (int i = 0; i < row.ItemArray.Length; i++)
        {
            row["Employees.SocialSecurityNumber"] = "xxx-xx-xxxx";
        }
    }
}
return dt;
```

NOTE. This assumes the column SocialSecurityNumber is saved as a string. If trying to set a date or date time field to blank use System.DBNull.Value.

The following example filters the data based on a calculated age value.

```
// get field name and age from parameters to compare against
string fieldName = sessionInfo.GetParameter("fieldName").Value;
int age = int.Parse(sessionInfo.GetParameter("age").Value);

// log parameters
sessionInfo.WriteLog("FilterByAge fieldName: " + fieldName);
sessionInfo.WriteLog("FilterByAge age value: " + age.ToString());

// get DataTable view and filter
System.Data.DataTable dt = (System.Data.DataTable)args[0];
System.Data.DataView dv = dt.DefaultView;

foreach(System.Data.DataRowView drv in dv)
{
    if (drv[fieldName] == System.DBNull.Value || (int)((System.DateTime.Today -
(System.DateTime)drv[fieldName]).Days / 365) < age)
        drv.Delete();
}

// return filtered DataTable
return dv.ToTable();
```

## OnReportExecuteStart

The OnReportExecuteStart Event occurs at the beginning of the Report Execution process. This Event could be used to check properties of a report and log or stop execution.

## Signature

For custom code the args array is structured as follows:

`args[]` is empty.

For .Net Assemblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo)

## Expected Return

The OnReportExecuteStart Event expects a string to be returned. Based on the return string there are three possible results.

- **Null / Whitespace** – If the string is null or whitespace then the report execution will continue as expected.

- **LanguageId** – If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user and the report execution will terminate. For more information see **Multi-Language Support**.

- **Other –** If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user and the report execution will terminate.

**Notes**

The report being executed can be accessed through the sessionInfo object by using sessionInfo.Report.

**Example**

The following example shows how each report execution can be written to a log file.

```
//Writes the current time, companyId, userId and report name to a specified log file.
File.WriteAllText("C:\ReportExecutionLogFile", String.Format("{0}, {1}, {2}, {3}",
DateTime.Now.ToString(), sessionInfo.CompanyId, sessionInfo.UserId, sessionInfo.Report.Name));
//returns null to proceed with execution
return null;
```

**OnReportExecuteEnd**

The OnReportExecuteEnd Event occurs at the end of the Report Execution process. This Event could be used to track which report executions return data.

**Signature**

For custom code the args array is structured as follows:

args[] contains a single Boolean indicating if Data qualified (True), or not (False).

For .Net Assemblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo, bool DataQualified)

**Expected Return**

Anything can be returned to the OnReportExecuteEnd Event. Any return value will be ignored.

**OnWebServiceExecuteEnd**

The OnWebServiceExecuteEnd Event occurs when data is returned from a Web Service Data Source. This Event could be used to decompress or decrypt data being returned from a Web Service Data Source.

**Signature**

For custom code the args array is structured as follows:

Args[] contains a single string of the data coming from the Web Service in position zero.

For .Net Assemblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo, string webServiceXml)

**Expected Return**

The OnWebServiceExecuteEnd Event expects a string to be returned.

NOTE. This Event is only occurs when the callType Parameter has the value 1.

**Example**

The following example shows how information from a web service could be decompressed.

```
byte[] compressedBuffer = Convert.FromBase64String((string)args[0]);

using (System.IO.MemoryStream stream = new System.IO.MemoryStream())
{
        int uncompressedLength = BitConverter.ToInt32(compressedBuffer, 0);
        stream.Write(compressedBuffer, 4, compressedBuffer.Length - 4);
        byte[] uncompressedBuffer = new byte[uncompressedLength];

        stream.Position = 0;
        using (System.IO.Compression.GZipStream compress = new
System.IO.Compression.GZipStream(stream, System.IO.Compression.CompressionMode.Decompress))
        {
                compress.Read(uncompressedBuffer, 0, uncompressedBuffer.Length);
                compress.Close();
                return System.Text.Encoding.UTF8.GetString(uncompressedBuffer);
        }
}
```

### OnExecuteSqlStatementConstructed

The OnExecuteSqlStatementConstructed Event occurs just before SQL is sent to the Data Source to retrieve data for report execution. This Event could be used to inspect, log or modify the SQL that is being used for report execution.

**Signature**

For custom code the args array is structured as follows:

args[] contains a string representing the execution SQL in position zero.

For .Net Assemblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo, sting exectuionSql, SqlObject sqlObject)

**Expected Return**

The OnExecuteSqlStatementConstructed Event expects a string to be returned.

**Example**

The following example shows how report execution SQL can be written to a specified log file.

```
//Writes the current time, companyId, userId and report name to a specified log file.
File.WriteAllText("C:\ReportSqlLogFile", String.Format("{0}, {1}, {2}, {3}",
DateTime.Now.ToString(), sessionInfo.CompanyId, sessionInfo.UserId, args[0]));
//returns null to proceed with execution
return args[0];
```

**OnFilterSqlStatmentConstructed**

The OnFilterSqlStatementConstructed Event occurs just before SQL is sent to the Data Source to retrieve data to populate the filter dropdown menu of Exago. This Event could be used to inspect, log or modify the SQL that is being used to populate the filter dropdown menu.

**Signature**

For custom code the args array is structured as follows:

args[] contains a string representing the filter SQL in position zero.

For .Net Assmblies the method signature is as follows:

string **EventHandlerName**(SessionInfo sessionInfo, sting filtersSql, SqlObject sqlObject)

**Expected Return**

The OnFilterSqlStatementConstructed Event expects a string to be returned.

**Note**

This Event will provide the SQL for the Filter Dropdown Object if that feature is being utilized. See **Data Objects** for more information on Filter Dropdown Objects

**Example**

The following example shows how the filter dropdown SQL can be modified to provide the top 200 results instead of the top 100.

```
//this code example assumes SQL Server as a Data Source
string sql = args[0].ToString();
string newSqul = sql.Replace("Top 100" , "Top 200");
return newSql;
```

**OnOkFiltersDialog**

The OnOkFiltersDialog Event occurs when a user clicks on the Ok button in the Filter Execution Window. This window only displays if prompt for value was checked for a filter. This Event could be used to see what filters are being used on the report and/or assure that a filter exists.

**Signature**

For custom code the args array is structured as follows:

args[] is empty.

For .Net Assemblies the method signature is as follows:

> string **EventHandlerName**(SessionInfo sessionInfo)

**Expected Return**

The OnOkFiltersDialog Event expects a string to be returned. Based on the returned string there are three possible results.

- **Null / Whitespace** – If the string is null or whitespace then the report execution will continue as expected.

- **LanguageId** – If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user and the report execution will terminate. For more information see **Multi-Language Support**.

- **Other –** If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user and the report execution will terminate.

**Notes**

The filters of the report being executed can be accessed through the sessionInfo object by using sessionInfo.ReportExecFilters.

**Example**

The following example provides C# code that will prevent the Filter Execution Window from closing if there are no filters specified. This and similar checks can help prevent users from executing Reports that result in unnecessarily-large queries going against the Data Source(s)."

```
string hasFilters = null;

if(sessionInfo.Report.Filters.Count() > 0)
{
   hasFilters = "Please add Filters to the Report.";
}

return hasFilters;
```

**OnOkParametersDialog**

The OnOkParametersDialog Event occurs when a user clicks on the Ok button of the Parameter Prompt Window. The window will only displays if the report has a non-hidden parameter with a prompt text. This Event could be used to see what values the user is setting for each prompting parameter.

**Signature**

For custom code the args array is structured as follows:

> args[] is empty.

For .Net Assmblies the method signature is as follows:

> string **EventHandlerName**(SessionInfo sessionInfo)

**Expected Return**

The OnOkParametersDialog Event expects a string to be returned. Based on the returned string there are three possible results.

- **Null / Whitespace** – If the string is null or whitespace then the report execution will continue as expected.

- **LanguageId** – If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user. For more information see **Multi-Language Support**.

- **Other –** If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user.

**Notes**

This Event cannot override the value of Parameters for the report execution.

The Parameters of the report being executed can be accessed through the sessionInfo object by using sessionInfo.Report.

**Example**

The following example provides C# code that will prevent the Parameters Execution Window from closing if a specified parameter is blank. The user will be prompted with a message from the language file.

```
//assumes the language file has an element with the id "PleaseEnterParam"
return (String.IsNullOrWhiteSpace(sessionInfo.GetReportParameter("promptName").Value) ?
"PleaseEnterParam" : null);
```

**OnScheduledReportExecuteSuccess**

The OnScheduledReportExecuteSuccess Event occurs when scheduled report execution is finished. This event can be used to create an audit log of scheduled reports or check values on the report and determine if they should be sent as scheduled or interrupted.

**Signature**

For custom code the args array is structured as follows:

> args[] is empty.

For .Net Assemblies the method signature is as follows:

> bool **EventHandlerName**(SessionInfo sessionInfo)

**Expected Return**

The OnScheduledReportExecuteSuccess Event expects a Boolean to be returned. Returning True will prevent the scheduled report from being sent. Returning False will allow the report schedule to proceed with processing.

NOTE. This server event is called for Remote Execution of reports. However, the return value will be ignored as there is no email to be prevented.

## OnConfigLoadStart

The OnConfigLoadStart Event occurs after the configuration file is loaded.This may happen in the Api when the api object is initialized or in Exago when entering the application directly. This event can be used to change any configuration information on-the-fly via the SessionInfo object, such as decrypting database connection strings.

### Signature

For custom code the args array is structured as follows:

args[] is empty.

For .Net Assmblies the method signature is as follows:

void **EventHandlerName**(SessionInfo sessionInfo)

### Expected Return

The OnConfigLoadStart Event has a void return value.

## OnConfigLoadEnd

The OnConfigLoadEnd Event occurs after all Api changes are made and the host application container is redirected to Exago. If entering Exago directly this event is occurs immediately after OnConfigLoadStart. If the Api is being used but the host application does not redirect to Exago (such as using the direct Report.GetExecuteData method) the event can manually be called using the public method Api.SetupData.FireOnConfigLoadEndEvent().

Similar to the OnConfigLoadStart event, this event can also be used to change configuration information on-the-fly via the sessionInfo object. However making these changes after the Api calls can provide extra convience. For example if the host application is using the Web Service Api it can set a single parameter value using the WebService and then based on that parameter make further configuration changes within this event. This provides better performance, security and a reduction of http requests.

### Signature

For custom code the args array is structured as follows:

args[] is empty.

For .Net Assemblies the method signature is as follows:

void **EventHandlerName**(SessionInfo sessionInfo)

### Expected Return

The OnConfigLoadEnd Event has a void return value.

## OnRenameFolderStart

The OnRenameFolderStart Event occurs when a user attempts to rename a folder. This event happens before the folder is renamed permitting you to stop the renaming if desired.

**Signature**

For custom code the args array is structured as follows:

> args[] is contains two strings, the first represents the fully qualified current folder name, the second is the new folder name.

For .Net Assemblies the method signature is as follows:

> string **EventHandlerName**(SessionInfo sessionInfo, string currentFolderName, string newFolderName)

**Expected Return**

The OnRenameFolderStart Event expects a string to be returned. Based on the returned string there are three possible results.

- **Null / Whitespace** – If the string is null or whitespace then the report execution will continue as expected.

- **LanguageId** – If the string matches the id of any element in the language files then the string of that language element will be displayed as a message to the user. For more information see **Multi-Language Support**.

- **Other –** If the string does not match the id of any element in the language files then the returned value will be displayed as a message to the user.

**OnRenameFolderEnd**

The OnRenameFolderEnd Event occurs after user has renamed a folder.

**Signature**

For custom code the args array is structured as follows:

> args[] is contains two strings, the first represents the fully qualified old folder name, the second is the new folder name.

For .Net Assemblies the method signature is as follows:

> string **EventHandlerName**(SessionInfo sessionInfo, string currentFolderName, string newFolderName)

**Expected Return**

Anything can be returned to the OnRenameFolderEnd Event. Any return value will be ignored.

**OnDataFieldsRetrieved**

The OnDataFieldsRetrieved Event occurs after Data Fields are retrieved for a specific Data Object. This event is commonly used to change the order Data Fields are displayed in the Data Menu of the Report Designer.

**Signature**

For custom code the args array is structured as follows:

> args[] is contains three objects, the first a System.Data.DataTable containing the names and metadata of the Data Fields, the second the Data Object as a WebReports.Api.Reports.Entity object, the third a reference to a WebReports.Api.Data.DataObjectBase object which calls the event.

For .Net Assemblies the method signature is as follows:

> string **EventHandlerName**(SessionInfo sessionInfo, DataTable originalDataFields, Entity dataObject, DataObjectBase eventCaller)

**Expected Return**

Expects a System.Data.DataTable return value, which represents the modified data.

**Notes**

The DataTable being being passed to the event in the first argument has already applied Column Metadata.

**Example**

The following example shows how the fields could be displayed in reverse alphabetic order.

```
//this code example makes referens to System.Data.dll and System.xml.dll and uses the namespaces
Syste.Data, WebReports.Api.Reports and WebReports.Api.Data

DataTable dt = args[0] as DataTable;
DataView dv = dt.DefaultView;
dv.Sort = "column_name desc";
return dv.ToTable();
```

**OnGetUserPreferences**

The OnGetUserPreferences Event is used to retrieve user preferences when entering the application and when editing/executing reports.

**Signature**

For custom code the args array is structured as follows:

> args[] is contains two objects, the first a string with the user preference's id and the second a string with the user preference's value.

For .Net Assemblies the method signature is as follows:

> string **EventHandlerName**(SessionInfo sessionInfo, string id, string value)

**Expected Return**

Expects a string return value, which represents the user preference's value.

**Notes**

The event will only be called if the 'User Preference Storage Method' is set to Server Events in the **User Settings**.

### Example

The following example shows how the event can retrieve the user prefence's value from a database.

```
//this code retrieves user preferences from a database. This assumes two things:
// 1. A global variable exists called reportTableName which represents where the user preferencs
are stored
// 2. A method called ExecuteSQLCmd exists to execute sql statements

string stmt = String.Format("Select upValue From {0} Where id = @id",  reportTableName);
List<SqlParameter> sqlParams = new List<SqlParameter>();
sqlParams.Add(new SqlParameter("@id", id));
Object queryResult;
queryResult = ExecuteSQLCmd(stmt, sqlParams);
return queryResult == null ? null : queryResult.ToString();
```

### OnSetUserPreferences

The OnSetUserPreferences Event is used to store user preferences when setting startup reports or saving Interactive Report Viewer changes as user reports.

### Signature

For custom code the args array is structured as follows:

> args[] is contains two objects, the first a string with the user preference's id and the second a string with the user preference's value.

For .Net Assemblies the method signature is as follows:

> void **EventHandlerName**(SessionInfo sessionInfo, string id, string value)

### Expected Return

The event has a void return value.

### Notes

The event will only be called if the 'User Preference Storage Method' is set to Server Events in the **User Settings**.

### Example

The following example shows how the event can retrieve the user preference's value from a database.

```
//this code retrieves user preferences from a database. This assumes two things:
// 1. A global variable exists called reportTableName which represents where the user preferencs
are stored
// 2. A method called ExecuteSQLCmd exists to execute sql statements

string stmt = String.Format("Select upValue From {0} Where id = @id",  reportTableName);
List<SqlParameter> sqlParams = new List<SqlParameter>();
sqlParams.Add(new SqlParameter("@id", id));
Object queryResult;
```

```
queryResult = ExecuteSQLCmd(stmt, sqlParams);
return queryResult == null ? null : queryResult.ToString();
```

## OnLoadReportParameters

The OnLoadReportParameters event passes a list of Parameter elements that can be reordered or modified before they are sent to the client for display. Called when report parameters are loaded, but before any processing has occurred.

### Signature

For custom code the args array is structured as follows:

args[] is contains one object, a list of Parameter elements.

For .Net Assemblies the method signature is as follows:

void **EventHandlerName**(SessionInfo sessionInfo, List<Parameter> parameters)

### Expected Return

The event has a void return value.

### Example

The following example sorts Parameters based on their dependencies on each other in custom SQL.

```
List<Parameter> parameters = args[0] as List<Parameter>;
List<string> sortedParameters = new List<string>();

string paramPattern = @"@\w+@";

foreach(Parameter p in parameters) {
    int minPosition = 0;
    foreach (Match m in Regex.Matches(p.DropdownSqlStmt, paramPattern))
    {
        string matchedParamName = m.Value.Replace("@","");
        int matchIndex = sortedParameters.IndexOf(matchedParamName);
        Logger.Instance().Info("Found instance of " + matchedParamName + " (at index " +
matchIndex + ") in parameter " + p.Id);
        if(matchIndex >= minPosition)
            minPosition = matchIndex+1; //Needs to go after this since it is dependent upon it
    }
    sortedParameters.Insert(minPosition, p.Id);
}

parameters.Sort((a,b) => {

    int a_order = sortedParameters.IndexOf(a.Id);
    int b_order = sortedParameters.IndexOf(b.Id);

    if(a_order >= 0 && b_order >= 0)
       return (a_order - b_order);

    return 0;
});
```

```
for(int i=1; i<parameters.Count; i++) {
    parameters[i].IsEnabled = false;
}

return null;
```

## Action Events

Action Events are a framework for adding custom extensions to Exago. They are a simple and relatively straightforward way to customize the way the application responds to user input.

At their most basic, action events are custom code which activate when a certain condition in the Exago application is met. They can be used for a variety of purposes, including adding interactivity to reports and modifying the behavior of certain items in the Exago interface. Events can use only client-side scripts or a combination of client and server-side interaction.

Action Events can be grouped into two general categories: Local and Global events.

- **Local** events have two sub-categories:

    o Handlers attached to items in reports and set to fire automatically, or when the item is interacted with in the Report Viewer.

    o 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.
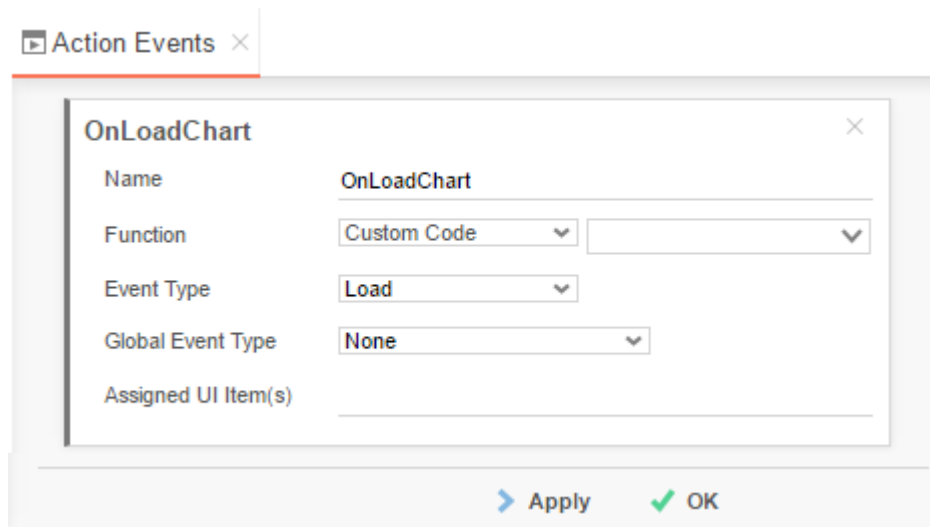
This chapter explains how to create Local and Global action events, describes the ways in which action events can interact with the Exago application, and lays out examples for common usages.

### 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:
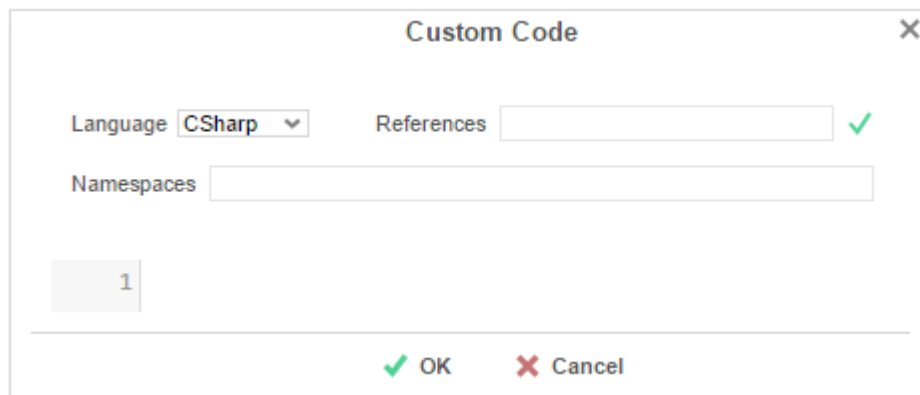
Each Event Handler has the following properties:

- **Name** – A unique identifier for each Event Handler.

- **Function** – Can either be Custom Code or a .Net Assembly method.

    o **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 the code executes properly ( ✔ ).



Custom Code has four properties:

- **Language** – Code can be written in C#, Javascript or VB.NET.

- **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.

- o **.NET Assembly Method** – To utilize a .NET Assembly method first 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. See **Writing Action Events** for information on how to access the arguments for each Event.

  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 **Exago Session Info**.

  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.

  - o **None** – This event handler is a Global Event.

  - o **Load –** The event handler will execute when a report item is loaded in the Report Designer or Viewer.

  - o **Click –** The event handler will execute when a user clicks on an item in a report or in the Exago UI.

- **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 **Description of 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 **List of 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.

## Adding Local Events to a Report Item

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 | True

- **Feature/UI Settings** > **Standard Report Designer Settings**

  Show Linked Action | True

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** ($\mathscr{F}$) button. The Linked Action Events Menu will open:
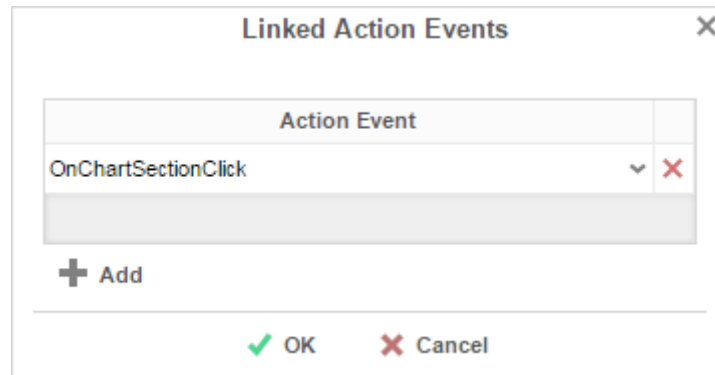


Press Add ( ✚ **Add** ) and select the event from the dropdown list. Press Delete (✖) to remove the selected event. Press OK ( ✔ **OK** ) when finished.

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.

## 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 **Exago Session Info**. 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** parameters.

  - **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)` – Sets the javascript string.

    NOTE. An action event must return the **JavascriptAction** object.

- **clientInfo** – A Javascript object that is called 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 **List of ClientInfo Elements**.

**arguments array** – Action events can also access an array of input values called args[]. These parameters can be set manually using a **ServerCallback**, but a few Global Events pass some preset parameters related to their specific action.

**Simple Example – Chart Formatting**

A simple action event may look something like the following. This example is attached to a chart object in a report. The code looks at the chart data to determine the highest and lowest values, then sets custom colors and labels for the respective items.

This event requires no special references or namespaces. This event does not make any server callbacks, and exclusively uses client-side scripts.

```
// First let's write the javascript.
string actionString = @"
        // Create variables for the lowest and highest values of the chart.
        var lowValueIdx = 0;
        var highValueIdx = 0;

        // Since we're attaching this action to a chart, the chartData object is set.
        // Cycle through the chart data items.
        for (var i = 1; i < clientInfo.chartData.data.length; i++)
        {
        // Determine the lowest and highest values.
        if (clientInfo.chartData.data[i].value < clientInfo.chartData.data[lowValueIdx].value)
                lowValueIdx = i;
        if (clientInfo.chartData.data[i].value > clientInfo.chartData.data[highValueIdx].value)
                highValueIdx = i;
        }

        // Set the colors and labels accordingly.
        clientInfo.chartData.data[lowValueIdx].color = "#FF0000";
        clientInfo.chartData.data[lowValueIdx].label += " (lowest)";
        clientInfo.chartData.data[highValueIdx].color = "#00FF00";
        clientInfo.chartData.data[highValueIdx].label += " (highest)";
";

// Finally, set the client script to the code we just wrote and return JavascriptAction.
sessionInfo.JavascriptAction.SetJsCode( actionString );
return sessionInfo.JavascriptAction;
```

**Complex Example – Save Handling**

Next, let's take a look at a more complex example. This example uses a global event to intercept the standard save report functionality and prompt the user for some additional information, which is then saved alongside the report. This example makes multiple server callbacks; in fact, the action event itself is called multiple times with different arguments. Additionally, the event is stored within a .NET Assembly.

NOTE. This example could be written in a variety of ways. The entire code could be saved within the exago config file, for example. Or it could be split into multiple action events, each handling a separate case. The format is flexible and completely up to you.

**SaveReport.js**

This is the client-side javascript responsible for creating and handling a Dialog prompt for the user.

```
// Called upon clicking the save report button
function ShowSaveReportWindow(clientInfo, reportId, reportName, description)
{
        // Can create raw Html or an Html DOM element. LoadHtmlDialog will handle both cases.
```

```
        var html = "<div style='padding:15px'>";
        html += "<br /><br /><span>Fill in the fields below, then click OK to continue.</span>";
        html += "<br /><br /><br />";
        html += "<span>Report Folder \\ Name</span> <input id='reportName' type='text' />";
        html += "<br /><br />";
        html += "<span>Description</span> <input id='description' type='text' />";
        html += "<br /><br />";
        html += "<span>Other Info</span> <input id='other' type='text' />";
        html += "<br /><br /><br />";
        html += "</div>";

        // Create options argument; renderOnly tells Exago to keep visibility hidden, so that we
        can put html in DOM and set values as shown below.
        // Alternately, we could set values in our HTML above, and not set renderOnly to true.
        var options = {
                height: 300, width: 400, titleText: "Save Report", okCallback: function () {
                parent.OnOkSaveReport(clientInfo); },
                cancelCallback: function () { parent.OnCancelSaveReport(clientInfo); },
                renderOnly: true, showButtons: true
        };

        clientInfo.LoadHtmlDialog(html, options);

        // Set any values (for convenience).
        clientInfo.GetDialogElementById("reportName").value = reportName;
        clientInfo.GetDialogElementById("description").value = description;

        // Since we set renderOnly to true, we now need to show the dialog.
        clientInfo.ShowDialog();
}

// Called upon click of the OK button in the dialog.
function OnOkSaveReport(clientInfo)
{
        // Validate the data.
        var reportName = clientInfo.GetDialogElementById("reportName").value;
        var description = clientInfo.GetDialogElementById("description").value;
        var other = clientInfo.GetDialogElementById("other").value;

        // Additional (non-report) data can be saved either by ajax call in host application, or
        by passing data to server via server callback.
        // This demonstrates passing complex data via Exago interface:
        var oth = { x: 'xData', y: 7 };
        var arr = ['a', 'b', 'z'];
        // Objects are automatically converted to json; We'll need to deserialize on server side.
        var obj = { reportName: reportName, description: description, other: oth, array: arr };

        // Now call back to the server to save the data.
        clientInfo.ServerCallback("SaveReport", "save", obj);

        // We can close the dialog right away or when save is completed after server callback.
        clientInfo.CloseDialog();
}

// Called upon click of the Cancel button in the dialog.
function OnCancelSaveReport(clientInfo)
{
        clientInfo.CloseDialog();
}
```

## ActionEvent.dll

This is the server-side code which tells Exago how to handle the user input. It's stored in an assembly.

```csharp
public class SaveReportDataOther
{
        public string x;
        public int y;
}

public class SaveReportData
{
        public string reportName;
        public string description;
        public SaveReportDataOther other;
        public string[] array;
}

public class ActionEvent
{
        public static object SaveReport(SessionInfo sessionInfo, string actionType, string jsonData)
        {
                JavascriptAction jsAction = sessionInfo.JavascriptAction;
                switch (actionType)
                {
                case null:
                // Initialization. Happens when Exago is first loaded in order to set the
                javascript click event.
                        jsAction.SetJsCode("clientInfo.refreshDataOnReturn = false;
                        clientInfo.ServerCallback(\"SaveReport\", \"show\");");
                        break;
                case "show":
                // Happens when the save button is clicked. We're calling back to server since we
                don't have all the following session values.
                        jsAction.SetJsCode(String.Format("parent.ShowSaveReportWindow(clientInfo,\"
                        {0}\",\"{1}\",\"{2}\");", sessionInfo.Report.Id,
                        jsAction.EncodeString(sessionInfo.Report.Name),
                        jsAction.EncodeString(sessionInfo.Report.Description)));
                        break;
                case "save":
                // Called when the OK button is clicked in our custom dialog. Exago has not saved
                the report, but makes reportXml available to save as shown below.
                // This describes how we can create complex data that gets mapped to a custom
                class definition.
                        SaveReportData data = Json.Deserialize(jsonData, typeof(SaveReportData)) as
                        SaveReportData;
                // Get the report xml to save.
                        string reportXml = sessionInfo.ReportObject.GetXml();
                // Save the report. Report name and ID are embedded within reportXml, although we
                can also pass them in seperate fields or within our combined jsonData.
                // We can alert the user of something here.
                        sessionInfo.JavascriptAction.SetJsCode("clientInfo.Alert(\"Report has been
                        saved\");");
                        break;
                default:
                        throw new Exception(String.Format("Invalid actionType: {0}", actionType));
                }
                return sessionInfo.JavascriptAction;
        }
}
```

}

NOTE. For an additional action event example, see **Responsive Dashboards**.

## Description of Global 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.

Following is the full list of global events that can have an action event attached. This list applies to the current version of Exago, but it is not final. Additional event triggers will be added in future releases.

### OnSaveReport

| Description | Fires when an open report is saved. |
| --- | --- |
| Remarks | Passes the report object. |

### 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** | Called 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. |

## List of ClientInfo Elements

Following are brief descriptions of 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** | Likely unnecessary in most cases. A pre-written action event provided to you by a support analyst may make use of this. |

`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 **List of UI Elements**. |

**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, 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. |
|---|---|

## List of UI Elements

This is an incomplete list of items in the Exago UI that can be captured by "Click"-type Local Action Events. If an item is not on this list, you can use a Web Inspector to determine the ID of an UI item.

NOTE. You can capture clicks on arbitrary UI items as long as the item has an associated ID field.

For example, in Chrome press **F12** to open the developer tools, press **Ctrl-Shift-C** to enable the Inspect Element tool, and hover over the item for which you wish to find the ID:
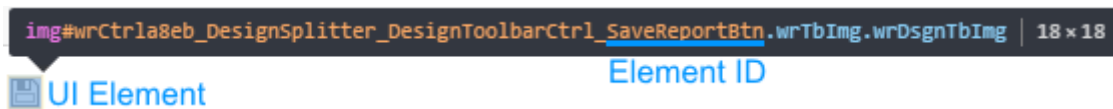


The Element ID is usually going to be the text string situated in between the rightmost underscore mark (_) and the leftmost period mark (.).

To attach an action event to UI elements, key the element IDs into the **Assigned UI Item(s)** text field, separated by a comma:



NOTE. UI items do not yet support "Load"-type action events.

| *Toolbar* | MergeCellsBtn |
|---|---|
| ReportOptionsBtn | SplitCellsBtn |
| SaveReportBtn | WrapTextBtn |
| DesignNewReportBtn | AutoSumBtn |
| UndoBtn | SuppressDuplicatesBtn |

| | |
|---|---|
| RedoBtn | InsertFileInput |
| FormatCellsBtn | CrossTabWizardBtn |
| FormatPaintbrushBtn | EditFormulaBtn |
| BoldBtn | LinkedReportBtn |
| ItalicBtn | LinkedActionBtn |
| UnderlineBtn | ExportBtn |
| DropDownImage | ExecuteBtn |
| FontNameList_DropDownContainer | |
| FontNameList_DropDownText | *Main Menu* |
| FontNameList_DropDownSelect | NewReportBwn |
| FontSize_NumericEditContainer | ManageFoldersBtn |
| FontSize_NumericEditTB | ExecuteInteractiveBtn |
| FontSize_NumericEditUp | DuplicateReportBn |
| FontSize_NumericEditDown | DeleteReportBtn |
| AlignTopBtn | ExportBtn |
| AlignLeftBtn | ScheduleReportBtn |
| AlignMiddleBtn | |
| AlignCenterBtn | *Tab Bar* |
| AlignBottomBtn | UserPreferencesBtn |
| AlignRightBtn | HelpBtn |
| AlignJustifyBtn | |

## 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 Option

Custom Options enable you to define 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.

- o **Int** – Represents a whole number.
- o **Decimal** – Represents a decimal.
- o **Bool** – Represents a Boolean value. A checkbox is displayed.
- o **Text** – Represents text and displays a text box.
- o **List** – Represents a choice from among multiple values. Click the add button (+) to define choices.
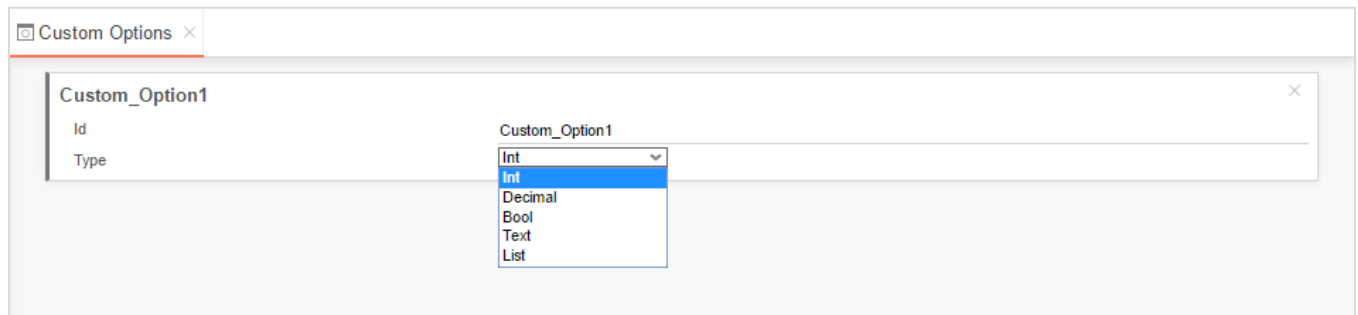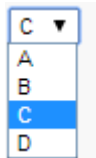
## Setting Options

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.
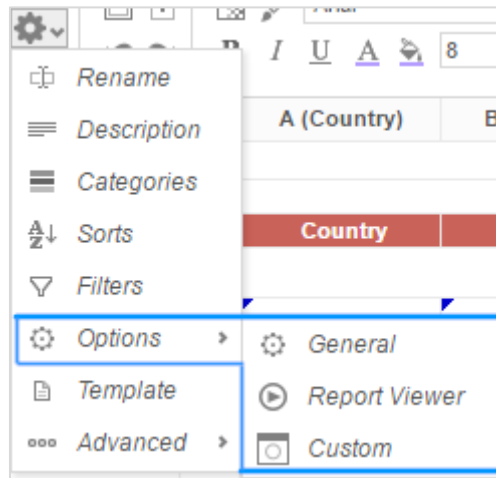
## Accessing Options

The .Net Api, Server Events and Custom Functions can access Custom Options values through the SessionInfo 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".<br>For List Options, the chosen Id is returned.<br>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.<br><br>return sessionInfo.GetReportCustomOptionValue(args[0].ToString()); |

# Integration

The following chapter details how to integrate Exago into your host application.

This chapter will assume that you have already used the **Administration Console** to establish the desired data structure, general settings and roles.

## About

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**.

## Styling

Visually modifying and rebranding the user interface is a simple but effective step toward integrating Exago into the host application. For styling purposes Exago can be thought of as a control that sits within a div on an .aspx page. Aesthetic changes can be made for single users or groups of users by directing each user/group to different custom .aspx pages.

To visually integrate Exago make a copy of the.aspx example below 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 Properties

Within each .aspx page 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"`).

    NOTE. If entering Exago through the Api this parameter is ignored.

- **Language File -** 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, gettingstartedcustom"`).

- **ForceIECompatMode –** Setting to True will force certain JavaScript functions to working in 'compatibility' mode. Currently this property only needs to be set 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-comptaible 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"`).

## Changing CSS

All of the CSS used by Exago can be modified at the bottom of the .aspx page. This means that every individual element or class of objects can be modified. To do make changes, add `<style type="text/css"></style>` to your .aspx page in the line above `</body>`. Between these style tags place the desired modifications to the CSS.

The table below details the recommend CSS classes for styling.

| Class | Feature | Property | Example |
|-------|---------|----------|---------|
| **Text Elements** | | | |
| .wrMain | Modifies text throughout Exago. | color | `.wrMain {color:Red;}` |
| .wrInputText | Modifies the text of input boxes and dropdowns. | color | `.wrInputText {color:Blue;}` |
| .wrTree | Modifies the text of tree controls such as the reports in the Main Menu or the Data Fields in the Report Designer. | color | `.wrTree {color:Green;}` |
| .wrTreeItemSelected | Modifies the selected item in a tree control. | color | `.wrTreeItemSelected {color:yellow;}` |
| .wrDynamicTabItem | Modifies the styling of tabs that can be deleted and moved. | color | `.wrDynamicTabItem {color:Orange;}` |
| .wrDynamicTabItemSelected | Modifies the styling of a selected dynamic tab. | color | `.wrDynamicTabItemSelected {color:Green;}` |
| .wrStaticTabItem | Modifies the styling of tabs that can not be deleted and moved. | color | `.wrStaticTabItem {color:Orange;}` |
| .wrStaticTabItemSelected | Modifies the styling of a selected | color | `.wrStaticTabItemSelected` |

| | | | |
|---|---|---|---|
| | static tab. | | `{color:`<span style="color:green">`Green`</span>`;}` |
| `.wrDialogTitle` | Modifies the text of the title of dialog menus | color | `.wrDialogTitle {color:`<span style="color:orange">`Orange`</span>`;}` |
| **Background Elements** | | | |
| `.wrMainLeftPane` | Modifies the background of the Main Menu | background-color | `.wrMainLeftPane {background-color `<span style="color:blue">`Blue`</span>`;}` |
| `.wrTabContent` | Modifies the background of all Tabs | background-color | `.wrTabContent {background-color:`<span style="color:blue">`Blue`</span>`;}` |
| `.wrTabContentWizard` | Modifies the background of all Wizards (ex. the New Report Wizard) | background-color | `.wrTabContentWizard {background-color:`<span style="color:blue">`Blue`</span>`;}` |
| `.wrDialogShadow` | Modifies the background of all dialog menus (ex. the Filters Menu) | background | `.wrDialogShadow {background: -webkit-gradient(linear, left top, left bottom, from(white), to(Blue));}` |
| `.wrPopupMenu` | Modifies the background of all popup menus (ex. the Folder Management Menu) | background | `.wrPopupMenu {background: -webkit-gradient(linear, left top, left bottom, from(white), to(blue));}` |
| `.wrDesignLeftPane` | Modifies the background of the Data Field Menu in the Report Designer | background-color | `.wrDesignLeftPane {background-color:`<span style="color:blue">`Blue`</span>`;}` |
| `.wrDsgnTbContainer` | Modifies the background of the Report Designer | background-color | `.wrDsgnTbContainer {background-color:`<span style="color:blue">`Blue`</span>`;}` |
| `.wrTabItem` | Modifies the background unselected tabs | background-color | `.wrTabItem {background-color:`<span style="color:blue">`Blue`</span>`;}` |
| `.wrDsgnTbSection` | Modifies the gradient behind the buttons on the Report Designer | background | `.wrDsgnTbSection {background: -webkit-gradient(linear, left top, left bottom, from(white), to(Blue));}` |
| **Selected Elements** | | | |
| `.wrTabItemSelected` | Modifies the selected tab | background | `.wrTabItemSelected {background: -webkit-gradient(linear, left top, left bottom, from(white), to(Blue));}` |
| `.wrTreeItemSelected` | Modifies the selected item in a tree control | background-color | `.wrTreeItemSelected {background-color:`<span style="color:purple">`Purple`</span>`; }` |
| `.wrPopupMenuItemHover, wrPopupMenuItem:hover` | Modifies the selected item popup menu (ex. the Report Folder Management) | background-color | `.wrPopupMenuItemHover, .wrPopupMenuItem:hover {background-color:`<span style="color:purple">`Purple`</span>`; }` |
| `.wrTbImgHover, .wrMainTbImgHover, .wrTbImg:hover` | Modifies the background of tool bar images when they are hovered over. | background-color | `.wrTbImgHover, .wrMainTbImgHover, .wrTbImg:hover {background-color:`<span style="color:orange">`Orange`</span>`;}` |
| **Other Elements** | | | |
| `.wrImageButton, .wrButton1` | Modifies the buttons (ex. Ok, Cancel) | background | `.wrImageButton, .wrButton1 {background: -webkit-gradient(linear, left top, left bottom, from(white), to(Blue));}` |
| `.wrDialogDragBar` | Modifies the bar atop all dialog menus (ex. the Filters Menu) | background | `.wrDialogDragBar {background: -` |

| | | | webkit-gradient(linear, left top, left bottom, from(white), to(Blue));} |
|---|---|---|---|
| .wrMainReportDescription Container | Modifies the report description in the Main Menu | border | .wrMainReportDescriptionContainer {border: solid 1px blue;} |

A comment within the ExagoHome.aspx highlights how to make these changes as well as explaining how to quickly remove the Exago logo from the application.

```
<body>
    <form id="form1" runat="server">
        <div>
            <div id="WebReportsContainer">
                <wr:WebReportsCtrl ID="WebReportsCtrl" runat="server" />
            </div>
        </div>
    </form>
    <style type="text/css">
        /* Styles can be overidden here like this: .wrDynamicTabItemSelected { color:red; }
        Please duplicate this file (ExagoHome.aspx) to your own aspx page, since this file will be recreated during next install
        See technical guide for additional information */

        /* Uncomment the following styles to remove the company logo from the left-hand sidebar */
        /*
        .wrMainCompanyLogo {
            display: none;
        }
        .wrMainReportsContainer {
            top: 111px;
        }
        .wrMainTbContainer {
            margin-top: 10px;
        }
        */
    </style>
```

The code below demonstrates an example of custom CSS styling.

**Code:**

```
<style type="text/css">
    .wrMain { color:Red; }
    .wrInputText { color:Blue; }
    .wrTree { color:Green; }
    .wrTreeItemSelected { background-color:Purple; }
    .wrDynamicTabItemSelected { color: Pink; }
    .wrDialogTitle { color: Orange; }
    .grRh { color: mediumaquamarine; }
    .grNum { color: sandybrown; }
    .wrGridTbl thead th { color: cadetblue; }
    .wrTabText { color: dodgerblue; }
</style>
```

**Before:**

**After:**

## Changing Icon Images

To further Exago's integration capabilities, any icon in Exago can be changed on a per-company or per-user basis.

To change the icons of Exago:

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.

    Ex. `<element id="ExportTypeMenuHtml" image="Config\Images\Custom\HTMLExecutIconLarge.png"></element>`

### Hovering Images

For icons that have hover effects (ex. the next page button on report output) there is a special naming convention.

To change cusom 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 pressin 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).
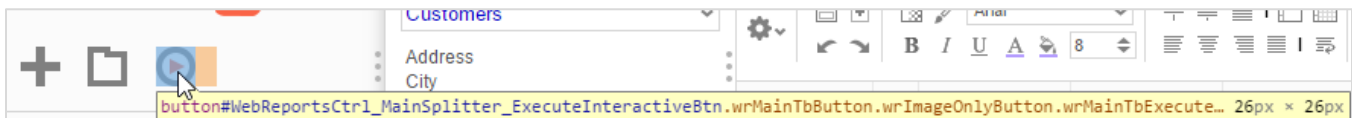


## Styling the Administration Console

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 (see example below.

   ```
   <%@ Page Language="C#" EnableViewState="false" %>
   <%@ Register src="WebAdminCtrl.ascx" tagname="WebAdminCtrl" tagprefix="wr" %>
   ```

3. Modify surrounding styles and css in the same manner described in the sections above.

# Multi-Language Support
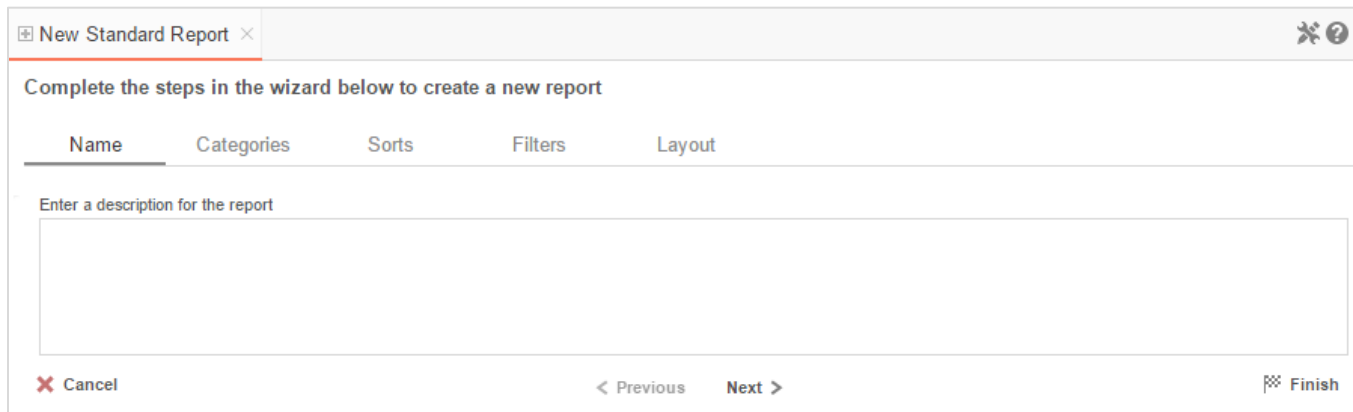
NOTE. The language elements discussed in this section do not include those created by users or administrators such as reports, folders, express report/crosstab themes or Data Field names. To modify Data Field names please see **Column MetaData**. To modify theme names please see **Express Report and 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="NewReportLbl">Complete the steps in the wizard below to create a new
        report</element>
</NewReport>
```



Es-mx.xml:

```
<NewReport>
        <element id="NewReportLbl">Complete los pasos en el asistente para crear un nuevo
        informe</element>
</NewReport>
```



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
<?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.
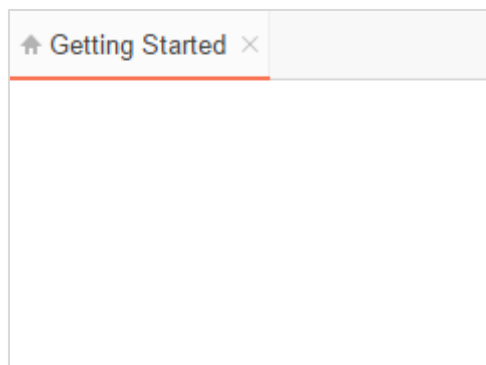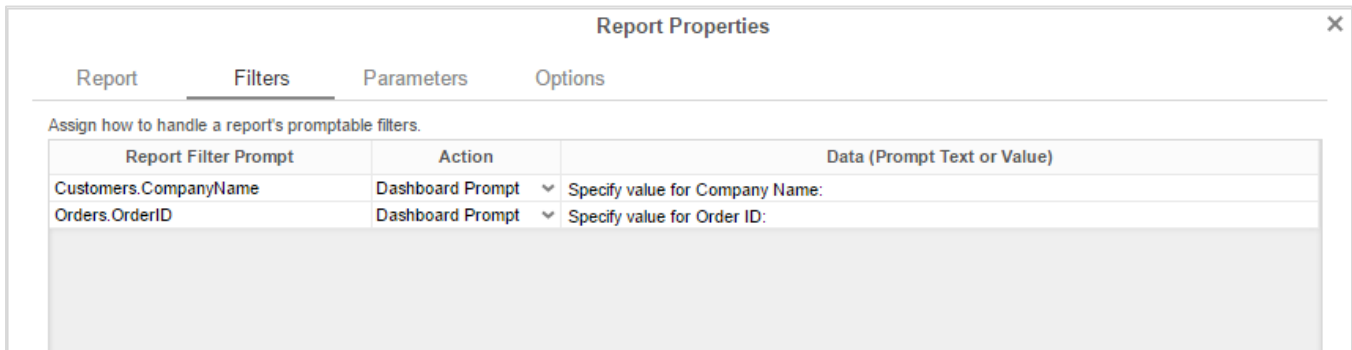
NOTE. Begin all language xml files with the line '`<?xml version="1.0" encoding="utf-8" ?>`'

## Text of Prompting Filters and Parameters on Dashboards

When adding a Report to a Dashbaord 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.



# 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, run 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.
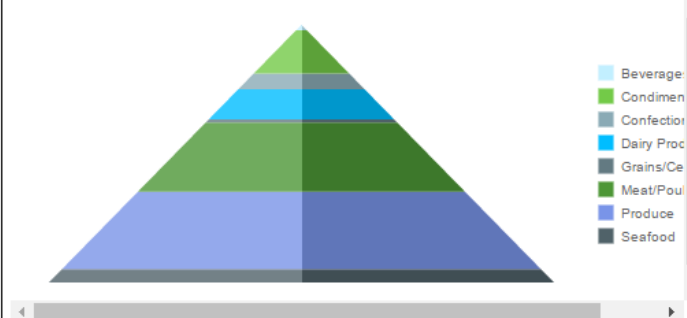
NOTE. It is recommended to make custom tabs in a separate language file to make it easy to change tabs by user or groups of users. See **Modifying Select Language Elements**.

## Creating Additional Custom Tabs

Additional custom tabs can be created by creating two language elements with unique names. One element specifies the title of the custom tab and the second contains the html content. Custom tabs can be opened with the JavaScript function wrAddTabbedContent (see **Available JavaScript Functions**).

The example below demonstrates a custom tab that has buttons to launch reports.

```
<element id="QuickReportsTabName">Quick Reports</element>
<element id="QuickReportsTab">
        <style type="text/css">
        .Button
        {
                height:20px;
                width: 60px;
                color: black;
                font-size:8pt;
```

```
        margin-right:5px;
    }
    .divProductDescription
    {
        margin-bottom:3px;
    }
    </style>
    <p style="font-family:Arial; font-size:12pt; font-weight:bold; text-decoration:underline;
text-align:center; margin-bottom:10px;">Click the format below the report you want to run. </p>
    <br />
    <div class="divProductDescription">
        <b>Revenue by Category (with drilldown)</b> - Complete list of revenue generated
by each category of products.
    </div>
    <div class="divProductButtons">
        <input type="button" class="Button" value="HTML" onclick="wrExecuteReport('Sales
Reports\\Revenue by Category','html')" />
        <input type="button" class="Button" value="EXCEL" onclick="wrExecuteReport('Sales
Reports\\Revenue by Category','excel')" />
        <input type="button" class="Button" value="PDF" onclick="wrExecuteReport('Sales
Reports\\Revenue by Category','pdf')" />
        <input type="button" class="Button" value="RTF" onclick="wrExecuteReport('Sales
Reports\\Revenue by Category','rtf')" />
        <input type="button" class="Button" value="CSV" onclick="wrExecuteReport('Sales
Reports\\Revenue by Category','csv')" />
    </div>
</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()

| Description | Opens the New Report wizard in a new tab. |
|---|---|
| Example | Ex. *Click <span style="text-decoration:underline; cursor:pointer;" onclick="wrStartNewReportWizard();">here</span> to create a new report.* |

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. *Click <span style="text-decoration:underline; cursor:pointer;" onclick=" wrStartDuplicateReportDialog();">here</span> to create a duplicate this report.* |

### void wrExecuteReport(string reportFolder\\reportName, string format)

| | |
|---|---|
| **Description** | Executes the specified report in the specified format. |
| **Example** | Ex. **<input type="button" class="Button" value="HTML" onclick="wrExecuteReport('Sales Reports\\Revenue by Category','html')** |

### string wrGetSelectedReportName()

| | |
|---|---|
| **Description** | Returns the name of the report that is selected in the 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. The TabName should make the element ID of the name you want the tab to display. |

### 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. NOTE. When using this function make sure the setting **Enable Debugging** in **Other settings** is **False**. |
| **Example** | Ex. **<div class="Report" data-onloadreportname="Employee Reports\\Number of Sales by Employee"></div>** |

### data-useviewer ="True/False"

| | |
|---|---|
| **Description** | Specifies to load a report as raw html or utilize Exago dynamic report viewer. |
| **Remark** | Default value is True. In cases where the dynamic capabilities of the Exago viewer is not need set to False to load raw html. |
| **Example** | Ex. *<div class="Report"  data-onloadreportname="Employee Reports\\Number of Sales by Employee" data-useviewer= "False"></div>* |

### 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. *<div class="Report" data-onloadreportname="Employee Reports\\Number of Sales by Employee" data-enablescrolling="False"></div>* |

data-reloadinterval="n"

| Description | Reloads a report every *n* seconds. |
|---|---|
| Remark | This function is used in conjunction with **data-onloadreportname**. |
| Example | Ex. *<div class="Report" data-onloadreportname="Employee Reports\\Number of Sales by Employee" data-reloadinterval="2"></div>* |

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. *<div class="Report" data-onloadreportname="Employee Reports\\Number of Sales by Employee" data-reloadinterval="1"></div>* |

# Themes: Charts, Crosstabs, Express Reports & Maps

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 '_wrThemeId' 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_wrThemeId'. 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 Colum 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.
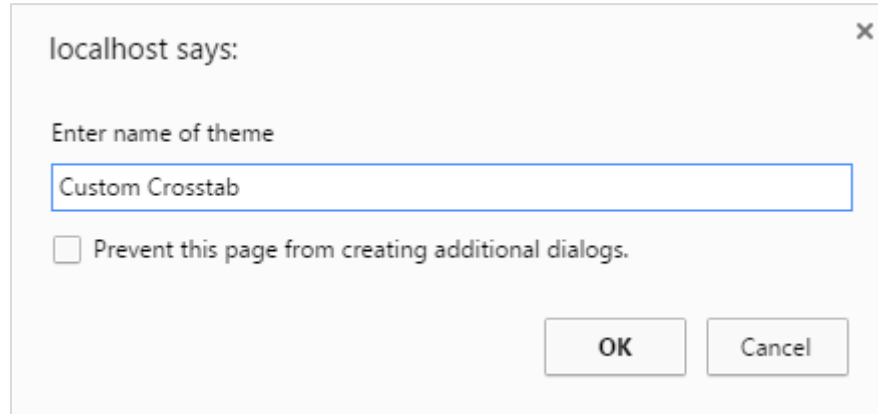


5. Click 'Save as Theme'.

6. Enter a name for the Theme. This name will be displayed to the end-users.
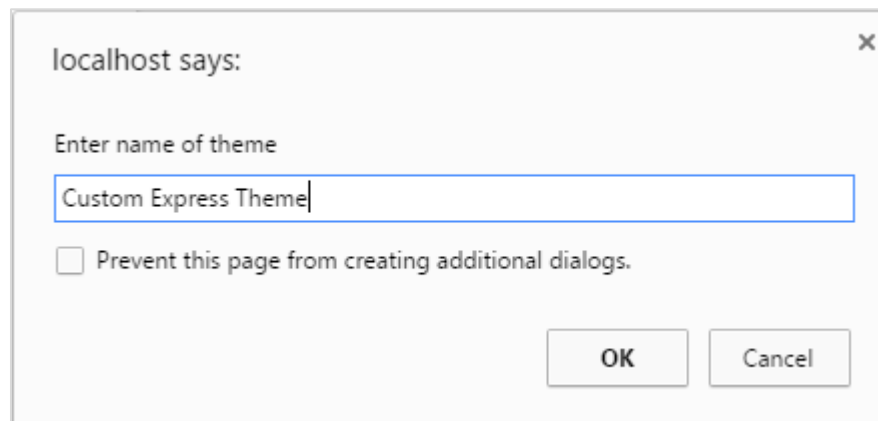


## Express Report Themes

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. Hold Alt+Ctrl+Shift and click on the save button (💾).

4. Enter a name for the theme. This name will be displayed to the end-users.



## Map Themes

A user can quickly select colors for Maps by applying a map theme.

To create custom Map 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.

# Using Exago within a WinForm

To embed Exago within a WinForm application the following properties should be set within the WebReportsCtrl line of the .aspx page that contains Exago (default is Exagohome.aspx).

- WinFormsApp – Set to True to ensure proper functionality within Exago.

- BrowserEmulation – Forces Exago to emulate the behavior of a specific browser. Valid Values are as follows: IE7, IE8, IE9, Firefox, Chrome, and Safari.

The example below shows these properties being set to force emulation of IE9 and make Exago aware that it is running within a WinForms application.

```
<wr:WebReportsCtrl ID="WebReportsCtrl" runat="server" BrowserEmulation="IE9" WinFormsApp="true" />
```

NOTE. The Host application can disable right clicking within Exago by setting the property IsWebBrowserContextMenuEnabled on the browser control to False.

# Cloud Environment Integration

By default, Exago stores all files on the server where it is installed, however, Exago can be deployed in cloud environments. This can be accomplished through either direct support of **Microsoft Azure** or by building a .**NET Assembly or Web Service** to handle the storage and retrieval of reports, folders, themes, document templates, images and temporary files.

## Azure Cloud Support

To deploy Exago within an Azure environment the storage of 3 file-groups must be handled.

- **Configuration files**
- **Reports, folders, themes, and document templates**
- **Temp files and and images**

## Configuration File Storage

Configuration files are the xml and encrypted xml that is created by the Admnistration Console. Instead of being stored in the default ./Config folder, these files can be stored in Azure.

This can be accomplished in one of two ways:

- Within the root directory of Exago modify the file 'appSettings.config' and add the Azure connection information as shown in the example below.
  Ex:    `<appSettings>`
  `<add key="ExagoConfigPath" value="pathtype=azure;credentials='My Azure Credentials Connection String';storagekey=config"/>`
  `</appSettings>`

  Then when the host application instantiates the Api use the api constructor that includes the Azure Path using the same connection string as above.

  - For the .Net Api use **Constructor**(string appVirtualPath, string configFile, string azurePath)

- Within the host application add the xml below to either the web.config or app.config file to specify the Azure connection information.
  Ex:    `<appSettings>`
  `<add key="ExagoConfigPath" value="pathtype=azure;credentials='My Azure Credentials Connection String';storagekey=config"/>`
  `</appSettings>`

NOTE. For details on the format of Azure connection strings please see the section **Report Storage** below.

## Report Storage

Exago has direct support for the storage and retrieval of reports, folders, themes, and templates using Microsoft Azure. To enable this set the Report path of **Main Settings** in the Administration Console to an Azure path that follows the format below.

`pathtype=azure;credentials='Credentials String';storagekey='';usefilestorage=false`

- **Pathtype**: Indicates the type of report repository. Currently, the only valid value is 'Azure'.
- **Credentials**: Indicates the credentials to the Azure account.
- **Storagekey**: (optional) Defaults to 'wrreports'. This is prefix for a blob container or fileshare used to store report files and can be used to allow different sets of report storage based on the end-user client. For example: If the value for storagekey = 'user1' all reports will be stored in the container or fileshare 'user1-reports', templates will be stored in 'myreports-templates', and themes will be stored in 'user1-themes'.
- **Usefilestorage**: (optional) Defaults to 'false' which uses Azure blob storage. If set to 'true', Exago will use Azure file storage.
  - Templates and themes always use blob storage.
  - Templates are automatically stored in blobs when using the template upload button.
  - Themes must be uploaded manually via an external application.

## Temporary Files Storage

For Exago to store temp files and images in Azure set an azure path with the following format in the 'Temp Cloud Service' **Main Settings** of the Administration Console.

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

Temp files can only be saved to a blob container.

## .Net Assembly/Web Service Cloud Support

To integrate Exago into a non-azure cloud environment two things are required.

- Report/Folder Management must be used to store and retrieve reports, folders, themes and tempaltes. See Report and Folder Management for more information

- A .Net Assembly or Web Service must be implemented to handle the storage and retrieval of Temp files and Images. See below for more details.

To handle temp file storage create and specify a .Net Assembly or Web Service in the Temp Cloud Service of the **Main Settings** in the Administration Console.

> NOTE. .NET Assembly format should be 'assembly = AssemblyFullPath.dll;class-Namespace.ClassName'. Web Service should be formatted as 'url=http://WebServiceUrl.asmx'.

The .Net Assembly/Web Service must have the following functions:

**void** SetValue (**string** companyId, **string** userId, **string** key, **byte[]** value)

| | |
|---|---|
| **Description** | Provides the byte content of the temp file to be saved. |
| **Remark** | The key is the name of the file being stored. |

**byte[]** GetValue (**string** companyId, **string** userId, **string** key)

| | |
|---|---|
| **Description** | Returns the byte content of the temp file. |
| **Remark** | The key is the name of the file being retrieved. |

**void** CleanUp (**string** companyId, **string** userId, **int** maxFileAge)

| | |
|---|---|
| **Description** | Optional function to delete old temp files. |

### Example

```
using System;
using System.IO;
namespace Exago.Services
{
    public class TempStorage
    {
```

```csharp
public static void SetValue(string companyId, string userId,
string key, byte[] value)
{
        File.WriteAllBytes(@"c:\Exago\AssemblyDataSource\Temp\" +
        key, value);
}
public static byte[] GetValue(string companyId, string userId,
string key)
{
        return
        File.ReadAllBytes(@"c:\Exago\AssemblyDataSource\Temp\" +
        key);
}

public static void Cleanup(string companyId, string userId, int
maxFileAge)
{
        try
        {
                DateTime expiredTime =
                DateTime.Now.AddMinutes(maxFileAge * -1);
                DirectoryInfo dirInfo = new
                DirectoryInfo(@"c:\Exago\AssemblyDataSource\Temp");
                FileInfo[] files = dirInfo.GetFiles();
                foreach (FileInfo file in files)
                {
                        if (file.LastWriteTime < expiredTime)
                        {
                                try { file.Delete(); }
                                catch { /* not critical */ }
                        }
                }
        }
        catch { /* not critical */ }
}
    }
}
```

# 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.
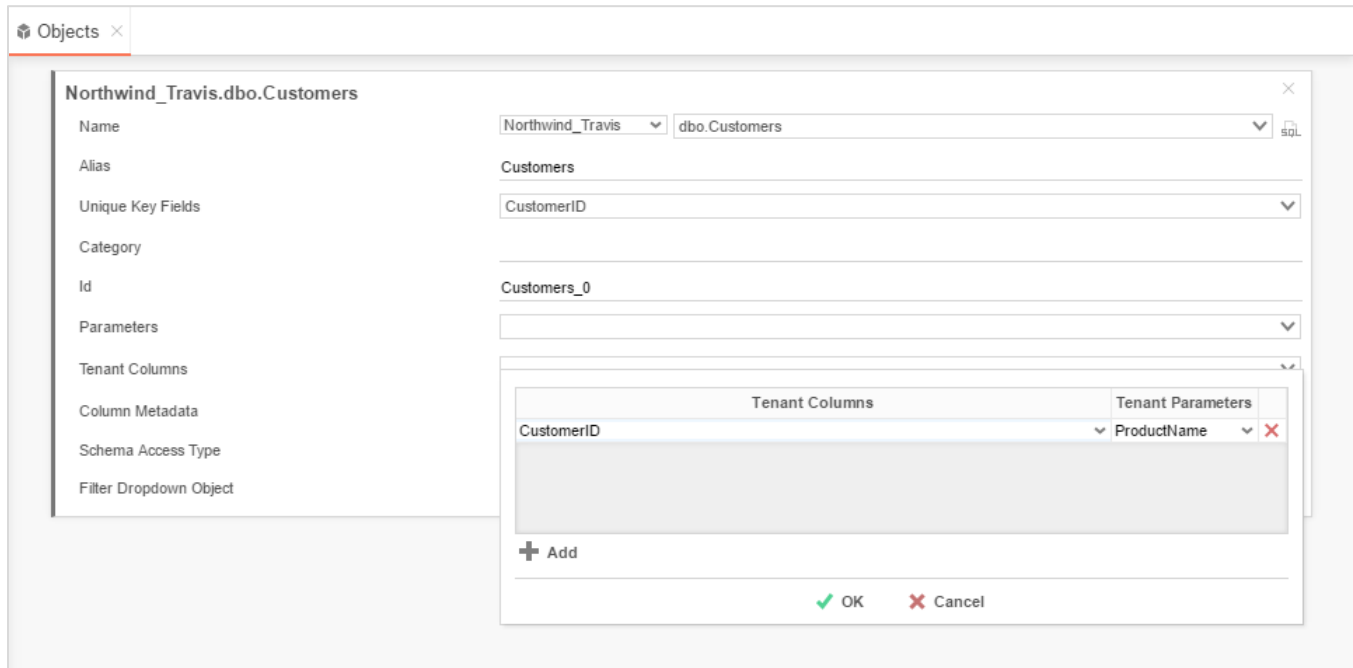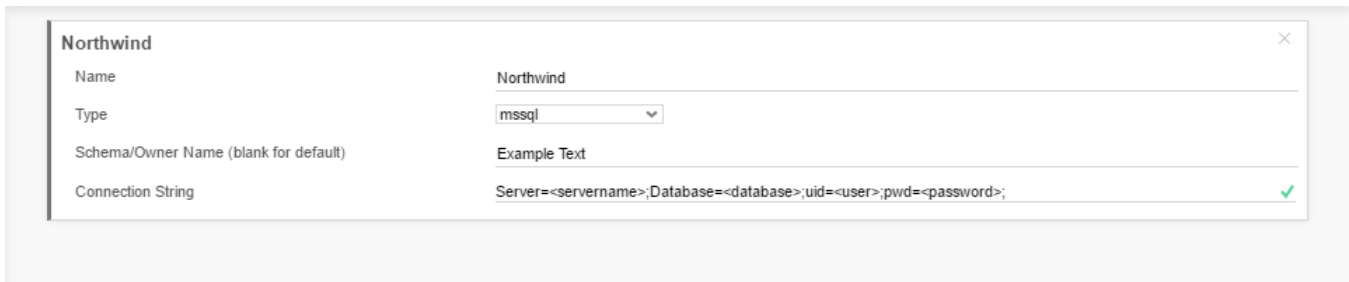


## Schema Based Tenancy

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.

2. For each table/view/stored procedure create a Data Object. In the Name dropdown select the object that utilizes the schema value used in step 1. This will tell Exago that for this Data Object it should retrieve the schema from the Data Source.

3. When initializing Exago through the **Api**, set the schema on the Data Source for the current user.
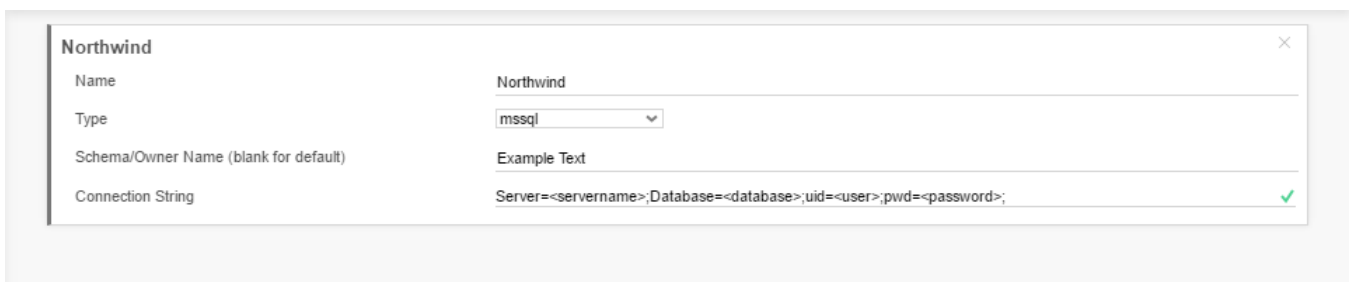
## Database Based Tenancy

Another way to assure that each user can only access their data is to provide a separate database for each user. In this situation each database should have the same tables, views and stored procedures.

To support database based tenancy in Exago:

1. Create a Data Source and corresponding Data Objects using any one of the Databases.

2. When initializing Exago through the **Api**, set the connection string on the Data Source to access the appropriate database for the current user



## Custom SQL Based Tenancy

Multi-Tenant security can also be assured by using **Custom SQL** for all Data Objects. Exago can pass parameter values into each SQL statement to filter data based on user.

To set Custom SQL based tenancy in Exago:

1. For each Data Object open the **Custom SQL** menu and create the desired SQL utilizing parameters to assure only appropriate information is available.

   NOTE. Parameters should be surrounded by single quotes.

2. When initializing Exago through the **Api**, set the value of any parameters utilized in the SQL for the current user.
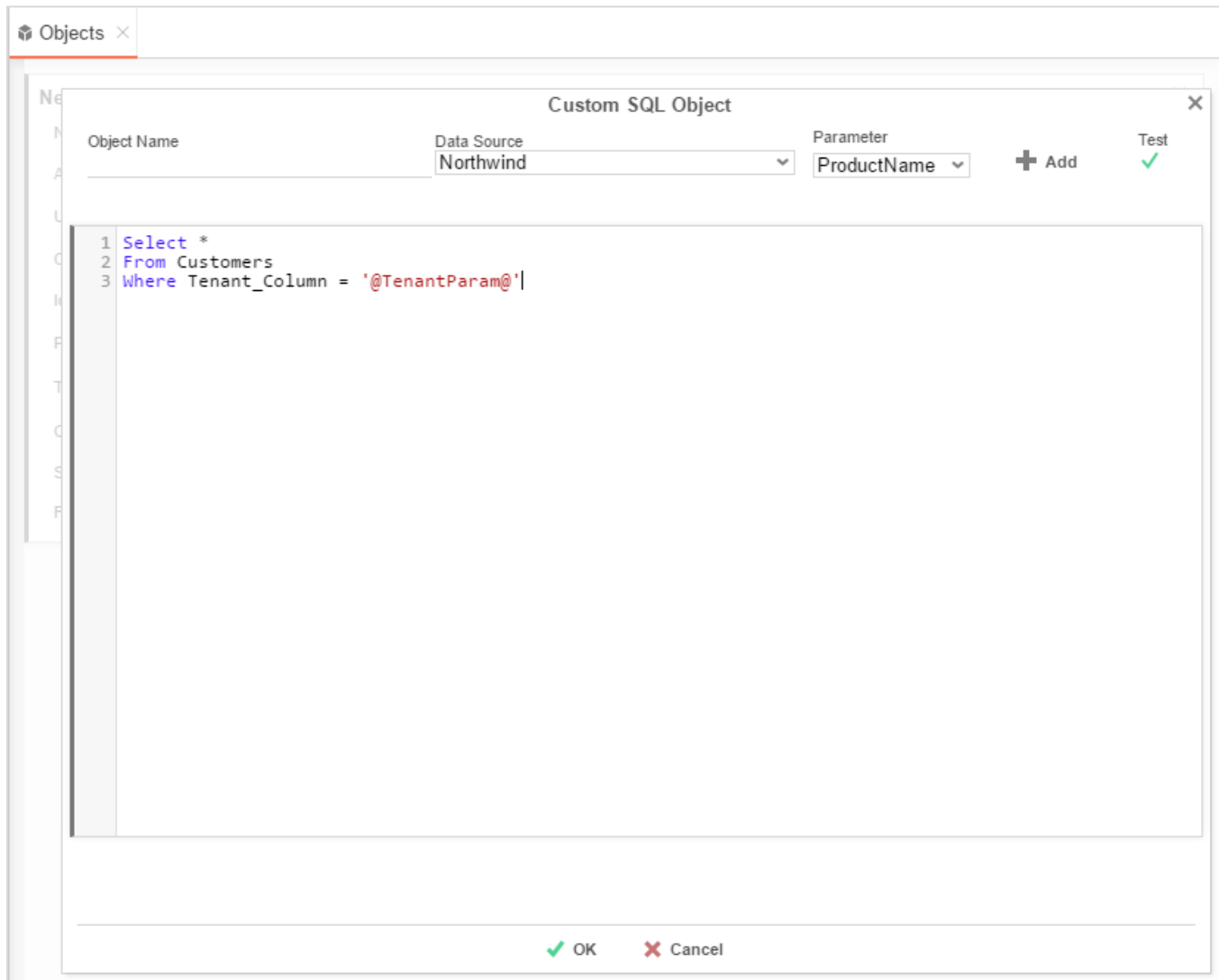
# Manual Application Installation

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

This section will show how to integrate the installation of:

> **Exago and Exago Web Service Api**
> **Exago Scheduler Service**

NOTE. Due to significant differences in IIS before and after version 7, some sections will provide separate explanations for versions prior to IIS 7 and after IIS 7.

## Exago and Exago Web Service Api Installer Integration

### Summary

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

1. Copy the Exago and/or the Exago Web Service Api files to installation folders.

2. Create IIS Virtual Directory to point to Exago/Exago Web Service Api.

3. Configure IIS as required for Exago/Exago Web Service Api setup.

4. Modify the system registry (optional).

NOTE. The installation of Exago Web Service Api is only used for clients who wish to develop using the Web Service Api instead of the .NET Assembly.

## Directory Structure

The directory structure should be preserved as follows:

**Exago:**
- [Exago Physical Directory]
    - /Bin
    - /Config
    - /Images
    - /Temp

**Exago Web Service API:**
- [ExagoApi Physical Directory]
    - /Bin
    - /Config

## File Installation

The host installer should create a copy of all the files that are initially created by the Exago/Exago Web Service Api Installer.

NOTE. **(optional)**

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

**Exago:**
- /Config/
    - WebReports.xml and/or WebReports.xml.enc

**Exago Web Service Api:**
- /Config
    - WebReportsApi.xml

## IIS Configuration

The method of creating new web applications and services differs depending on what version of IIS the server is using. Microsoft made significant changes to IIS versions 7+ which simplified creating new Web Sites, Virtual Directories, etc.

NOTE. Verify that the Virtual Directory does not exist before attempting to create the new one.

**IIS Version 5.0-6.0**

**Create Virtual Directory**

A virtual directory requires the following input:

- **siteName** – Name of the IIS Web Site where it will be installed. (ex. 'Default Web Site')
- **vDirName** – Name of Virtual Directory for the installation (ex. 'Exago' or 'ExagoApi')
- **physicalPath** – Physical installation path. (ex. 'C:\Program Files\Exago\Exago')

The following C# code provides an example of how to set these properties.

```csharp
public void CreateVDir(string siteName, string vDirName, string physicalPath)
{
        System.DirectoryServices.DirectoryEntry oDE;
        System.DirectoryServices.DirectoryEntries oDC;
        System.DirectoryServices.DirectoryEntry oVirDir;

        oDE = new DirectoryEntry(siteName + "/Root");

        //Get Default Web Site
        oDC = oDE.Children;

        // Delete before it re-create
        bool isVDirExists = true;
        try
        {
                DirectoryEntry dirEnt = oDC.Find(vDirName,  oDE.SchemaClassName.ToString());

                if (dirEnt != null)
                {
                        //Changed to Update virtual directory physical path.
                        //If virtual directory already exist do not delete and
                        //recreate.
                        dirEnt.Properties["Path"].Value = physicalPath;
                        dirEnt.CommitChanges();
                }
        }
        catch (DirectoryNotFoundException)
        {
                isVDirExists = false;
        }
        catch (COMException comEx)
        {
                if (comEx.Message == "Exception from HRESULT: 0x80005008")
                        return;
                else
                        throw;
        }

        if (isVDirExists)
```

```
                        return;

                //Add row
                oVirDir = oDC.Add(vDirName, oDE.SchemaClassName.ToString());

                //Commit changes for Schema class File
                oVirDir.CommitChanges();


                //Create physical path if it does not exists
                if (!Directory.Exists(physicalPath))
                {
                        Directory.CreateDirectory(physicalPath);
                }

                //Set virtual directory to physical path
                oVirDir.Properties["Path"].Value = physicalPath;

                //Set read access
                oVirDir.Properties["AccessRead"][0] = true;

                //Create Application for IIS Application (as for ASP.NET)
                oVirDir.Invoke("AppCreate", true);
                oVirDir.Properties["AppFriendlyName"][0] = vDirName.Substring(vDirName.LastIndexOf('/') +
                1);
                oVirDir.Properties["DefaultDoc"][0] = "Home.aspx";
                oVirDir.Properties["EnableDefaultDoc"][0] = false;
                oVirDir.Properties["AppIsolated"][0] = 2;

                //Save all the changes
                oVirDir.CommitChanges();
}
```

## Configure Framework

All Exago components require .NET Framework 4.0. Thus, IIS needs to be set to an app pool that also uses .NET Framework 4.0. The host installer should verify that this Framework is currently installed on the web server.

The following C# code provides an example of how to check and set the proper Framework.

```
public static void SetFramework(string webSitePath)
{
        try
        {
                string frameworkPath = Environment.GetEnvironmentVariable("WINDIR") +
                @"\microsoft.net\framework";

                // Check to see if the system has the 64 bit version of .NET
                if (Directory.Exists(frameworkPath + "64"))
                {
                 frameworkPath += "64";
                }

                // Set the .NET Framework to .NET 4.0
                string strExe = frameworkPath + @"\v4.0.30319\aspnet_regiis.exe";
                if (File.Exists(strExe))
                {
```

```
                    ProcessStartInfo pi = new ProcessStartInfo();
                    pi.FileName = strExe;
                    pi.Arguments = "-s " + webSitePath.Replace(@"IIS://localhost/", "");
                    pi.UseShellExecute = false;
                    pi.CreateNoWindow = true;
                    Process proc = Process.Start(pi);
                    proc.WaitForExit();
            }
        }
        catch
        {
            throw;
        }
    }
}
```

## IIS Version 7+

The following is a C# code sample of how to create a new IIS installation of Exago/Exago Web Service API, using *Microsoft.Web.Administration.dll*. The code requires the following input:

- **siteName** – Name of the IIS Web Site where it will be installed. (ex. 'Default Web Site')

- **vDirName** – Name of Virtual Directory for the installation (ex. 'Exago' or 'ExagoApi')

- **physicalPath** – Physical installation path. (ex. 'C:\Program Files\Exago\Exago')

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

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

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

                    app.ApplicationPoolName = "DefaultAppPool";
            }

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

```
        {
                throw;
        }
}
```

## Exago Scheduler Installer Integration

### Summary

The installer integration of the Exago Scheduler has six steps:

1.  Check to see if the Exago Scheduler is running as a Windows Service (if so stop this service).

2.  Copy the Exago Scheduler files to installation folders.

3.  Modify the system registry (optional).

4.  Modify the security settings on the Exago Scheduler directory.

5.  Create a new Windows Service for the Exago Scheduler

6.  Enable the Exago Scheduler service.

### File Installation

Before running the installation, the Windows Services should be checked to see if the Exago Scheduler is currently installed and/or running as a service. If the Exago Scheduler is currently installed and/or running as a service it should be shut down. The host installer should then create a copy of all the files that are initially created by the Exago Scheduler Installer.

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

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

```csharp
ServiceState serviceSt = WindowsServiceInstaller.GetServiceStatus("ExagoScheduler");

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

        for (int ProgCtr = 0; ProgCtr <= 120; ProgCtr++)
        {
                Thread.Sleep(1000);
                serviceSt = WindowsServiceInstaller.GetServiceStatus("ExagoScheduler");

                if (serviceSt == ServiceState.Stop)
                        break;

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

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

```
            }
}
```

## Directory Security Settings

The Exago Scheduler service will require changes to the security settings of the installation directory to enable Windows to run the program scheduler.exe as a Windows Service.

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

- **dirName** – Physical path to Exago Scheduler (ex. 'c:\Program Files\Exago\ExagoScheduler\')

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

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

            DirectoryInfo dirInfo = new DirectoryInfo(dirName);

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

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

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

## Windows Service Creation

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

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

```csharp
serviceSt = WindowsServiceInstaller.GetServiceStatus("ExagoScheduler");
// is Exago Scheduler already installed as a service
if (serviceSt == ServiceState.NotFound || serviceSt == ServiceState.Unknown)
{
        // install Exago as a new Windows Service
        WindowsServiceInstaller.Install("ExagoScheduler","ExagoScheduler",
                                filePath + "ExagoScheduler.exe");


        for (int timeCtr = 0; timeCtr <= 120; timeCtr++)
        {
                serviceSt = WindowsServiceInstaller.GetServiceStatus("ExagoScheduler");

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

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

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

        RegistryKey key = Registry.LocalMachine.OpenSubKey("SYSTEM\\CurrentControlSet\\Services\\" +
                "ExagoScheduler", true);

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

                for (int timeCtr = 0; timeCtr <= 120; timeCtr++)
                {
                        serviceSt = WindowsServiceInstaller.GetServiceStatus("ExagoScheduler");

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

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

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

## Optional Setup Information

Registry keys may be added to better enable reinstallation functionality (ex. pre-selecting values such as installation path, virtual directory name, etc.). These keys are optional and are not required for installer integration.

## Creating a Registry

A new registry item will need to be created in the path HKEY_LOCAL_MACHINE/SOFTWARE. Below are examples of such paths for the application, the Api and the scheduler.

> **Exago:**
> - HKEY_LOCAL_MACHINE
>   - o SOFTWARE
>     - ▪ Exago
>       - • Exago
>         - o Default Web Site/WebReports
>
> **Exago Web Service Api:**
> - HKEY_LOCAL_MACHINE
>   - o SOFTWARE
>     - ▪ Exago
>       - • ExagoAPI
>         - o Default Web Site/WebReportsAPI
>
> **Exago Web Service Api:**
> - HKEY_LOCAL_MACHINE
>   - o SOFTWARE
>     - ▪ Exago
>       - • ExagoScheduler

## Values in a Registry

The following values can be added to the appropriate registry folders:

- **CreateDate –** Initial Installation date (ex. 6/17/2012 12:35:60)
- **DisplayName –** Has two possible values
  - o **Exago/Exago Web Service Api** – Set to the Installation Web Site followed by the Virtual Directory Name (ex. Default Web Site/Exago)**.**
  - o **Exago Scheduler –** Set to the directory name where the Exago Scheduler was installed (ex. Exago).
- **Location –** Physical installation path (ex. c:\Program Files\Exago\Exago).
- **UpdateDate –** Initially set to the installation date. Should be updated whenever Exago is reinstalled.
- **Version –** Set to the version of Exago being installed (ex. 2012.1.1). This value can be found by pressing 'Ctrl + Shift+ V' in Exago.

## Example of Registry

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

- **application** – Set to 'ExagoScheduler', 'Exago' or 'ExagoApi'.

- **path** – Set to the installation path.

- **website** – Set to the IIS Web Site where Exago is installed. Leave blank for the Exago Scheduler.

- **vdir** – Set to the virtual directory that Exago is set up as. Leave blank for the Exago Scheduler.

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

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

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

# Extensibility

The following chapter details features of Exago that can be enhanced or extended by the host application to provide additional functionality.

## Load Balancing 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 in the order they are specified.

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:

    o The Scheduler version must match the Exago Application version.

    o The Scheduler's language files and the Exago Application's language files must match.

    o Any custom assemblies must be present in the Scheduler directory.

- Configure the Exago Scheduler. For detailed instructions see: **Configuring Scheduler Services**.

    o 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:**

1. Using the Admin Console, open the **Scheduler Settings** menu.

    o Set 'Enable Remote Report Execution' to True in the **Report Scheduling Settings**.

    o 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.

# Multiple Data Models

In some cases a user may want the same Data Objects to be joined together differently.. To accomplish this, Data Objects and Joins can be placed into Categories to create multiple data models. When an end user selects a Data Object from a Category it indicates which joins to use.

The following steps detail how to create multiple data models.

1. In the Administration Console open **Other Settings** and set 'Limit Report to One Category' to True.
2. Open the configuration file (WebReports.xml) in the Config folder.

3. In the `<webreports>` section, begin by creating a `<category>` for each data model.

   NOTE. Each xml tag must be closed (ex. `<category>` must be closed with `</category>`).

4. For each data model:

5. Specify an ID with the `<category_id>` tag. The ID should be a unique identifier for the data model and will be utilized by the Data Objects and Joins.

6. Give the model a name that will be displayed to the end user using the `<category_name>` tag.

   NOTE. The `<category_name>` tag - acts as a 'folder' to group Data Objects. Sub-'folders' can be created by entering the category name followed by a backslash then the sub-category name. Ex. 'Sales\Clients'.

**Example:**

```
<category>
  <category_name>Exago University\Advisors</category_name>
  <category_id>advisorModel</category_id>
</category>
```

- For each Data Object (`<entity>` tag ):

    o With the `<category>` tag, create a comma separated list of IDs for each data model in which you want the Object to be available. In the example below two data models are specified by their IDs (advisorModel & classesModel).

**Example:**

```
<entity>
  <entity_name>Professors</entity_name>
  <db_name>Professor</db_name>
  <category> advisorModel,classesModel</category>
  <datasource_id>7</datasource_id>
  <object_type>xmltable</object_type>
  <key>
    <col_name>ID</col_name>
  </key>
```

- For each Join (`<join>` tag):

    o With the `<category>` tag, create a comma separated list of IDs for each data model in which you want the Join to be utilized. In the example below a Join between two Data Objects is being set to one data model (advisorModel).

**Example:**

```
<join>
 <entity_from_name>Professor</entity_from_name>
 <entity_to_name>Student</entity_to_name>
 <join_type>rightouter</join_type>
 <relation_type>1M</relation_type>
 <weight>0</weight>
 <category>advisorModel</category>
 <joincol>
    <col_from_name>ID</col_from_name>
    <col_to_name>Advisor</col_to_name>
 </joincol>
</join>
```

## Example

The following configuration example demonstrates how three Data Objects are made available in two different relational models. In the `advisorModel` model Students are joined directly to Professors, while in the `classesModel` model Students are joined to Professors indirectly through Classes.

**Models:**

```
<category>
    <category_name>Exago University\Advisors</category_name>
    <category_id>advisorModel</category_id>
 </category>
 <category>
    <category_name>Exago University\Classes</category_name>
    <category_id>classesModel</category_id>
 </category>
```

**Data Objects:**

```
 <entity>
   <entity_name>Classes</entity_name>
   <db_name>Class</db_name>
   <category>advisorModel,classesModel</category>
   <datasource_id>7</datasource_id>
   <object_type>xmltable</object_type>
   <key>
     <col_name>ID</col_name>
   </key>
 </entity>
 <entity>
   <entity_name>Students</entity_name>
   <db_name>Student</db_name>
   <category> advisorModel,classesModel </category>
```

```
      <datasource_id>7</datasource_id>
      <object_type>xmltable</object_type>
      <key>
        <col_name>ID</col_name>
      </key>
    </entity>
    <entity>
      <entity_name>Professors</entity_name>
      <db_name>Professor</db_name>
      <category>advisorModel,classesModel</category>
      <datasource_id>7</datasource_id>
      <object_type>xmltable</object_type>
      <key>
        <col_name>ID</col_name>
      </key>
    </entity>
```

**Joins:**

NOTE. The Professors => Classes join is utilized by both Data Models because no <category> is set.

```
<join>
   <entity_from_name>Professor</entity_from_name>
   <entity_to_name>Student</entity_to_name>
   <join_type>rightouter</join_type>
   <relation_type>1M</relation_type>
   <weight>0</weight>
   <category>advisorModel</category>
   <joincol>
      <col_from_name>ID</col_from_name>
      <col_to_name>Advisor</col_to_name>
   </joincol>
</join>
<join>
   <entity_from_name>Professor</entity_from_name>
   <entity_to_name>Class</entity_to_name>
   <join_type>inner</join_type>
   <relation_type>1M</relation_type>
   <weight>0</weight>
   <joincol>
      <col_from_name>ID</col_from_name>
      <col_to_name>Professor</col_to_name>
   </joincol>
</join>
<join>
   <entity_from_name>Student</entity_from_name>
   <entity_to_name>Class</entity_to_name>
   <join_type>inner</join_type>
   <relation_type>1M</relation_type>
   <weight>0</weight>
   <category>classesModel</category>
   <joincol>
      <col_from_name>Enrolled in</col_from_name>
      <col_to_name>Title</col_to_name>
   </joincol>
</join>
```

# External Interface

There are certain features of Exago that the host application may want to control directly. In some cases Exago provides the ability for the host application to do this by calling out to a specified Web Service or .NET Assembly with specific methods.

To utilize the External Interface:

1. Create a Web Service or .Net Assembly that contain the functions described below.

2. Specify the Web Service or .NET Assembly in the External Interface property of **Other Settings**.

    NOTE. A different external interface can be specified within the Scheduling Service configuration. For more details see **Configuring Scheduler Settings.**

    NOTE. The Web Service should be formatted as 'url=http://WebServiceUrl.asmx'. The .NET Assembly should be formatted as 'assembly = AssemblyFullPath.dll;class=Namespace.ClassName'. For a .NET Assembly all methods should be static.

The functions below will use the **parameters** 'companyId', and 'userId' which should be set through the Api as users enter Exago.

## Report Execution Start Event

To enable the host to track report executions, Exago and the Exago Scheduling Service will fire an event at the start of each report execution. The following method will be used.

void ReportExecuteStart(string companyId, string userId, string reportName)

| | |
|---|---|
| **Description** | Used to track report execution by user. |
| **Remark** | Should not return any value. |

## User Preference Management

By default Exago will store User Preferences such as which Dashboard Reports to execute on startup in a browser's cookie. While convenient this means if a user switches browsers or machines their preferences will be lost. Instead the host application can manage how these User Preferences are stored using the External interface.

To handle the storage of User Preferences:

1. In the **User Settings**, set User Preference Storage Method to "External Interface"
2. Implement the following methods:

void SetUserPreference(string companyId, string userId, string id, string value)

| | |
|---|---|
| **Description** | Used to set a particular user preference value. The id is a unique identifier for the user preference, and the value is the user preference value (may be null). |

| | |
|---|---|
| **Remark** | Should not return any value. |

string GetUserPreference(string companyId, string userId, string id)

| | |
|---|---|
| **Description** | Used to retrieve the value parameter of most recent SetUserPrefernce call for the companyId and userId. |
| **Remark** | Returns a string |

## Handling Time Zones

A server in one time zone may be utilized by users around the globe. This presents problems when handling functions that run on the server such as Now( ). There are two ways to handle such a situation: Use the **Culture Setting**, Server Time Zone Offset, or use the external interface functions below.

NOTE. For these functions to be called the Culture setting Sever Time Zone Offest must be blank.

DateTime ConvertToServerDateTime(string companyId, string userId, DateTime clientDateTime)

| | |
|---|---|
| **Description** | Used to adjust clients time to server's time zone. |
| **Remark** | Returns a DateTime. |

DateTime ConvertToClientDateTime(string companyId, string userId, DateTime serverDateTime)

| | |
|---|---|
| **Description** | Used to adjust server time to client's time zone. |
| **Remark** | Returns a DateTime. |

## Email List for Report Scheduling

Through the external interface, the Exago Scheduling Service can retrieve email distribution groups from the host application. This prevents having to maintain separate lists of email addresses within Exago.

When a report is scheduled, a call out is made to the host application to get the list of email addresses and distribution groups for the user to select from. This is done with the following method.

string GetEmailListXml(string companyId, string userId)

| | |
|---|---|
| **Description** | Returns a string listing folders and report names in xml format (see example). |
| **Remark** | Leave the tag `<email>` blank for an entry to indicate it is a distribution group. |

| | |
|---|---|
| **Example** | ```xml<br><emailAddressList><br>  <item><br>    <name>John Smith</name><br>    <email>jsmith@mycompanydomain.com</email><br>  </item><br>  <item><br>    <name>Sales Group</name><br>    <email></email><br>  </item><br></emailAddressList><br>``` |

If a scheduled report uses a distribution list then the following method will be called at the time the report is executed.

**string** GetEmailDistributionListXml(**string** companyId, **string** userId, **string** listName)

| | |
|---|---|
| **Description** | Returns a string listing folders and report names in xml format (see example). |
| **Remark** | Do not leave the `<email>` tag blank. The name item does not need to be returned for this method. |
| **Example** | ```xml<br><emailAddressList><br>  <item><br>    <email>jsmith@mycompanydomain.com</email><br>  </item><br>  <item><br>    <email>ajones@mycompanydomain.com</email><br>  </item><br></emailAddressList><br>``` |

## Custom Scheduler Recipient Window

To utilize the Custom Scheduler Recipient Window feature the following function may exist in the External Interface. See **Custom Scheduler Recipient Window** for more information.

**string** GetEmailList(**string** controlData)

| | |
|---|---|
| **Description** | Sends the external interface the Control Data previously provided by host application when a user clicks OK in the Custom Scheduler Recipient window. |
| **Remark** | Returns a string of email addresses separated by commas or semi colons. |

## Scheduler Repository Notification

When '**Email Scheduled Reports**' is set to False in the Administration Console the following method will call the External Interface to let the host application know when a scheduled report has been saved in the Scheduler Repository.

See **Saving Scheduled Reports to External Repository** for more information.

void ScheduledReportExecutionComplete(string companyId, string userId, string reportName, string exportFileName, int statusCode, string statusMsg)

| Description | Sends the external interface a notification that a scheduled report has been saved to the Scheduler Repository. |
|---|---|
| Remark | statusCode is 0 if the execution was successful, 1 if an error occurred or no data qualified.<br><br>statusMsg details the result of the execution (eg. "Report has successfully executed", or "There were errors in the report layout; please edit or contact your administrator").<br><br>Return value is void. |

# Custom Scheduler Recipient Window

When the functions GetEmailListXml and GetEmailDistributionListXml exist in the **External Interface** the To and Cc buttons on the Schedule Report Wizard become clickable and open a dialog for users to select email addresses or groups. This dialog can be replaced with a custom window created by the host application.

To utilize a Custom Scheduler Recipient Window:

1. Set a URL, height and width in the Custom Scheduler Recipient Window parameter in the **Scheduler Settings**. Ex url=www.CustomScheduler.com;height=100;width=300;

   NOTE. Height and Width are numbers that represent the dimensions of the window in pixels.

2. In the custom window utilize the following JavaScript functions:

   wrGetScheduleRecipientWindowEmailAddressData ()

   | Description | Use this function to retrieve any existing email address data the user has entered into the Schedule Report Wizard. |
   |---|---|

   wrSetScheduleRecipientWindowEmailAddressData (string displayData, string controlData)

   | Description | Call this function when the user clicks OK to tell Exago the email address data. |
   |---|---|
   | Remark | The displayData will appear in the To or Cc box of the Recipients window.<br><br>The controlData will be passed back to the Host application when the Scheduled report is run and sent out. |

   wrCancelScheduleRecipientWindow ()

   | Description | Call this function to close the custom window. |
   |---|---|

3. Create the function GetEmailList(string controlData) in the **External Interface** to convert the control data into the actual email addresses when the scheduled report has been run and is ready to be sent.

# Custom Filter Execution Window

When a report is executed, a filter execution dialog will appear if any of the filters on the report are set to 'Prompt for Value'. This dialog can be replaced with a custom window created by the host application. The custom window can be either a control saved within Exago or a separate webpage outside of Exago

To create Custom Filter Execution Window as a control within Exago:

1. Create an ascx file in the installation directory of Exago. Ex CustomFilterWindow.ascx

2. Set the control, height and width in the Custom Filter Execution Window parameter in the **Filter Settings**. Ex control=CustomFilterWindow.ascx;height=100;width=300;

   NOTE. Height and Width are numbers that represent the dimensions of the window in pixels. These settings are optional. If omitted, the dialog is sized to the value in the wrDialogMasterContainerCentered css class, which is currently 70%.

3. In the control use the JavaScript functions described below to show the custom filter window and create or modify filters before report execution begins.

To create Custom Filter Execution Window as a web page:

1. Set a URL, height and width in the Custom Filter Execution Window parameter in the **Filter Settings**. Ex url=www.CustomFilterExecution.com;height=100;width=300;

   NOTE. Height and Width are numbers that represent the dimensions of the window in pixels.

   NOTE. 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. www.CustomFilterExecution.com?language=en-us

2. In the custom webpage use the JavaScript functions described below to show the custom filter window and create or modify filters before report execution begins.

   NOTE. all the JavaScript functions must begin with 'parent.' as the page is placed inside an iFrame by Exago.

## Available JavaScript Functions

The following JavaScript functions are available for the Custom Filter Execution Window.

object[] wrGetFilterWindowData()

| Description | Gets the report's existing filters created in Exago as an array. |
|---|---|
| Remark | Returns an array of filter objects. For more information on the filter objects see wrReportFilter(). |

object[] wrGetFilterWindowDataObjects()

| Description | Gets the Data Categories of the report and their associated Data Fields. |
|---|---|
| Remark | Returns an array of representing the available Data Categories.<br>Each Data Category has a string providing its Name and a sub array representing the Data Fields.<br>Each Data field has a string providing its Name and an integer representing its Data Type. This integer uses the Data Type Constant described below.<br><br>DataType Constants:<br>0 - String<br>1 – Date<br>2 – Integer<br>3 - Bit<br>4 - Numeric<br>5 - Float<br>6 - Decimal<br>7 - Guid<br>8 - DateTime<br>9 – Time - not currently used<br>10 - Image<br><br>NOTE. All the Categories names being passed are the Alias for each Data Object. Similarly the Data Fields will return the name specified in column metadata if provided. |

## string wrGetActiveReportName()

| Description | Returns the name of the report being executed. |
|---|---|
| Remark | The returned string includes the folder path of the report separated by slashes. |

## bol wrShowFilterWindow()

| Description | Displays the custom filter execution window. |
|---|---|

## void wrReportFilter()

| Description | Creates a Filter object that can be added to the Filters array returned by wrSetFilterWindowData(). |
|---|---|
| Remark | Filter Objects have the following properties:<br><br>Name – The name of the data field being used.<br>Operator – Operator for filter. Uses enumeration `wrFilterOperator`<br>Values – Value(s) of filter<br>AndFlag – Boolean to set And/Or with next filter.<br>GroupWithNext – Boolean to group with next filter.<br>GroupStartCount – The number of opening parentheses that were manually added to the filter using ctrl + [.<br>GroupEndCount – The number of closing parentheses that were manually added to the filter using ctrl + ].<br>DataType – the type of data being filtered. Uses constants DataType (see below.<br><br>DataType Constants:<br>0 - String |

| | 1 – Date<br>2 – Integer<br>3 - Bit<br>4 - Numeric<br>5 - Float<br>6 - Decimal<br>7 - Guid<br>8 - DateTime<br>9 – Time - not currently used<br>10 - Image |
|---|---|

## bol wrSetFilterWindowData(object[] filters)

| Description | Sets the filters for the report, closes the custom filter execution window, and then begins report execution. |
|---|---|
| Remark | Returns a Boolean to indicate success.<br><br>This or wrCancelFilterWindow() should be the last function called by the custom filter execution window. |

## bol wrCancelFilterWindow()

| Description | Closes the custom filter execution window without changing the report's filters. |
|---|---|
| Remark | Returns a Boolean to indicate success.<br><br>This or wrSetFilterWindowData() should be the last function called by the custom filter execution window. |

## Example Custom Filter Execution Control

```
<%@ Control Language="C#" ClassName="MyCustomFilterDialog" EnableTheming="false" %>
<span>Hello Custom Filter Dialog</span>
<input type="button" value="Ok" onclick="OnOk();" />
<input type="button" value="Cancel" onclick="OnCancel();" />
<script type="text/javascript">
      OnOk = function()
      {
            // create array of wrReportFilter objects to send back to parent
            var filters = new Array();
            var filter = new wrReportFilter();
            filter.Name = "Employee.First Name";
            filter.Operator = wrFilterOperator.OneOf;
            filter.Values.push("Travis");
            filter.Values.push("Stew");
            filters.push(filter);
            wrSetFilterWindowData(filters);  // also continues execution
      }
      OnCancel = function()
      {
            wrCancelFilterWindow();
      }
      // initialize custom window with values from the parent
      var filters = wrGetFilterWindowData();
      var dataObjects = wrGetFilterWindowDataObjects();
```

```
        wrShowFilterWindow();
</script>
```

## Example Custom Filter Execution WebPage

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server"></script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
        <script type="text/javascript">
            window.onload = function() { Initialize(); };
            function Initialize()
            {
                    // initialize custom window with values from the parent
                    var filters = parent.wrGetFilterWindowData();
                    var dataObjects = parent.wrGetFilterWindowDataObjects();
                    parent.wrShowFilterWindow();
            }
            function OnOk()
            {
                    // create array of wrReportFilter objects to send back to parent
                    var filters = new Array();
                    var filter = new parent.wrReportFilter();
                    filter.Name = "Employee.First Name";
                    filter.Operator = parent.wrFilterOperator.OneOf;
                    filter.Values.push("Travis");
                    filter.Values.push("Stew");
                    filters.push(filter);
                    parent.wrSetFilterWindowData(filters);  // also continues execution
            }
            function OnCancel()
            {
                    parent.wrCancelFilterWindow();
            }
        </script>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <input type="button" value="Ok" onclick="OnOk();" />
        <input type="button" value="Cancel" onclick="OnCancel();" />
    </div>
    </form>
</body>
</html>
```

# Saving Scheduled Reports to External Repository

When using the Exago Scheduling Service you may specify for reports to be saved to a repository instead of having them emailed as attachments. When a Scheduled report is run and saved a callout to the **External Interface** will be made to notify the host application. This will allow the host application to notify the appropriate users their report is available.

To utilize the Repository:

1. Set '**Email Scheduled Reports**' in the **Scheduler Settings** to False**.**

2. In the Exago Scheduling Service installation open the file ExagoScheduler.xml.

3. Set the parameter "<report_path>" to specify the repository you want to use.

4. Create the function ScheduledReportExecutionComplete(string companyId, string userId, string reportName, string exportFileName, int statusCode, string statusMsg) in the **External Interface** to notify the host application the report execution is complete.

# 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 Viewer 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. |
| dialogCrosstabOptions | The user has the Options menu open in the Layout Tab of the Crosstab Wizard. |
| **New Report Wizard** | |
| tabStandardName | The user has the Names tab of the New Standard Report Wizard active. |
| tabStandardCategories | The user has the Categories tab of the New Standard Report Wizard active. |
| tabStandardSorts | The user has the Sorts tab of the New Standard Report Wizard active. |
| tabStandardFilters | The user has the Filters tab of the New Standard Report Wizard active. |
| tabStandardLayout | The user has the Layout tab of the New Standard Report Wizard active. |
| **Chained Report Wizard** | |
| tabChainedName | The user has the Names tab of the New Chained Report Wizard active. |
| tabChainedReports | The user has the Reports tab of the New Chained Report Wizard active. |
| tabChainedOptions | The user has the Options tab of the New Chained Report Wizard active. |
| **Report Designer** | |

| | |
|---|---|
| tabDesign | The user is editing a standard 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 General Options Menu active. |
| dialogHtmlOptions | The user has the Report Viewer Options menu active. |
| listItemReportHtmlOptionsGeneral | The user has the General section of the Report Viewer Options active. |
| listItemReportHtmlOptionsFilters | The user has the Filter section of the Report Viewer Options active. |
| listItemReportHtmlOptionsSorts | The user has the Sorts section of the Report Viewer Options active. |
| dialogTemplates | 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. |
| dialogChartBenchmarkLine | The user has the Benchmark Lines menu in the Appearance tab active. |
| tabChartType | The user has the Type tab of the Chart menu active. |
| tabChartDataFormat | The user has the Data Format tab of the Chart menu active. |
| tabChartAppearance | The user has the Appearance tab of the Chart menu active. |
| tabChartData | The user has the Data tab of the Chart menu active. |
| tabChartSizeAndPreview | The user has the Size and Preview 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.

# Report Templates Setup

Exago can map data onto PDF, RTF and Excel templates. To utilize this feature the templates must be properly set up in order to accept data from Exago. After being configured (see below**) templates should be saved in the report path** and these templates will be detected automatically by Exago.

NOTE. Configuring templates varies slightly by format.

## PDF Templates

On the PDF Template file create Form Fields where you want to map data. Remember that the name of the field will be displayed to users in Exago.

For items that repeat (those that will be mapped to cells in a 'detail section') give each form field the same name followed by a period and a number starting with 0. (ex. item.0, item.1, item.2, etc.)

Check the Multiline property on any PDF field where data may need to wrap to fit inside the field.

NOTE. Although you can use any program you would like to create and edit PDF templates we recommend Adobe Acrobat Professional or **http://www.pdfescape.com**.

### Check Boxes in PDF Templates.

Checkboxes are not currently supported in PDF templates. However the steps below detail how to have Exago populate a text field with a check mark.

1. For the PDF text field you want to contain checks set the font to wingdings. Do not put a border on the field. Save the template and place it in the report path.
2. In a cell on the report use an IF function whose results are char(254) for a checked box and char(111) for an unchecked box [ex. =if({Employees.Title} = 'Sales Representative' , char(254), char(111))]
3. In the Template menu assign the cell to the pdf field.

## RTF Templates

For RTF Template files create Bookmarks where you want to map data. Bookmark names do not display on the document, so we suggest typing the bookmark name in the document where the field will go, then select the text and make it a bookmark. The typed text within the bookmark will be replaced by mapped data when the report is executed.

There are two ways to display content that repeats on an RTF Template (those that will be mapped to cells in a 'detail section').

- If there will be a limited number of repetitions. Give each bookmark the same name followed by an underscore and a number starting with 0. Ex. item_0, item_1, item_2, etc.

- For content that may need to repeat an indefinite number of times. Create a single line of content and create a bookmark with the name structure RepeatForEach_bookmarkname.

## Dynamic content with RTF Templates

RTF Templates may also use Bookmarks to dynamically hide/display text or entire paragraphs.

To do this:

1. Select the text/paragraph you want to display/hide.
2. Make a bookmark using the naming convention KeepIF_name
3. In Exago make a formula that as that returns 1 if the text should be displayed and 0 if it shouldn't (ex. '=if({Products.ProductName} = 'Chai', 1, 0)' )
4. In the Document Template menu set the cell with the if condition to the bookmark.
5. Run the report as RTF.

# Excel Templates

The first worksheet of the Excel template should be left blank (except for the first row) as this is where Exago will populate the data. In the top row of this sheet place the name of the column that will be seen by the end user. All the other worksheets in the template will remain unchanged by Exago.

## Referencing Data in Excel Templates

When using an Excel Template there are two ways for Charts or Pivot Tables to reference the data populated by Exago: Named Ranges or referencing specific rows.

### Named Ranges

Excel has a concept of a Named Range which can be used by Charts or Pivot Tables to refer to a range of cells.
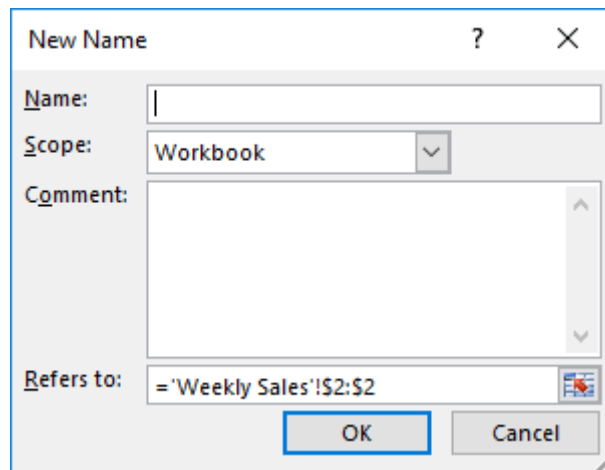
When creating the Template utilize Named Ranges by:

1. In the formula tab open the Name Manager
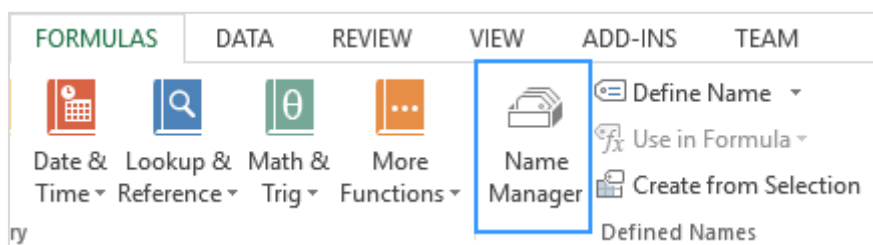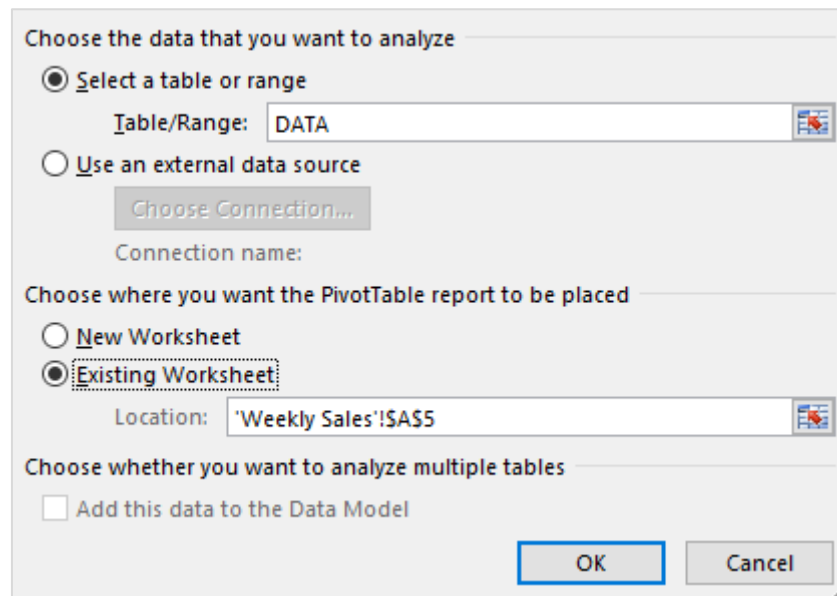


2. Add a new Named Range whose name matches the name of the first Worksheet. In the "Refers To" property select the upper left and upper right boundaries of the desired range. (Ex. If you want all the data from columns A – J, select cells 'Sheet1!$A$1:$J$1'.)

   NOTE. When the report is executed Exago will modify this range to include all of the rows in these columns that include data. (In the previous example if the report had 100 rows the range would be updated to 'Sheet1!$A$1:$J$100'.)

3. Set the Chart or Pivot Table to use the Named Range as its Data Source.

**Row Selection**

Instead of using Name Ranges each Chart or Pivot Table can be set to reference the first two rows on the first worksheet. (Ex. For a template with 5 columns the reference would be '=Sheet1!$A$1:$E$2'.) When the data is populated by Exago rows are inserted in a fashion that these references will automatically expand to incorporate each row of data.

# Report and Folder Storage/Management

By default, Exago stores the reports in a file system folder. The location of this folder is specified in the 'Report Path' property set in the Administration Console. Alternatively, report, template and folder storage, and retrieval can be handled by building a Web Service or .NET Assembly. This would allow for reports, folders and templates to be stored in a database. To do this, specify the Web Service or .NET Assembly in the Report Path of **Main Settings**. The Web Service or .NET Assembly should contain all of the methods specified in **List of Methods**.

## User Defined Fields

Exago has optional support for a User Defined Field (UDF) in a report. UDFs allow for storing custom attributes within reports. You can use UDFs in a custom folder management solution in order to associate an ID with an individual report.

NOTE. UDFs are not required for folder management. They are provided for convenience.

Certain report and folder management methods are overloaded to support UDFs. These methods are associated with report retrieval. If a UDF is associated with a report, Exago will pass it as an argument to these methods.

If a UDF is set it is stored in an element in the report XML called `<linked_udf>`. Report storage methods do not overload a UDF argument since they include the report XML as an argument.

## List of Methods

NOTE. The methods will use the **parameters** 'companyId', and 'userId' which should be set through the Api as users enter Exago from the host application.

NOTE. If using a .Net Assembly, the folder management code can use alternative method signatures to be passed Exago's SessionInfo object for additional flexibility. See **Accessing SessionInfo in Folder Management** for more information.

string GetReportListXml(string companyId, string userId)

| Description | Returns a string listing folders and report names in xml format (see example). |
|---|---|
| Remark | For reports set the flag `<leaf_flag>` to True. For folders set this flag to False. <br><br> If an error occurs return null and a generic error will be displayed to the user. |
| Example | Returns: <br><br> ```<entity>     <name>Travis' Reports</name>     <leaf_flag>false</leaf_flag>     <readonly_flag>false</readonly_flag>     <entity>       <name>Sales Report</name>       <leaf_flag>true</leaf_flag>       <readonly_flag>false</readonly_flag>     </entity>     <entity>       <name>Employee Reports</name>       <leaf_flag>false</leaf_flag>       <readonly_flag>false</readonly_flag>       <entity>         <name>Employee Benefits Report</name>         <leaf_flag>true</leaf_flag>         <readonly_flag>false</readonly_flag>       </entity>     </entity>   </entity>``` |

string GetReportXml(string companyId, string userId, string reportName, [string udfData = null])

| Description | Returns a string containing the report in xml format. |
|---|---|
| **Remark** | If an error occurs return null and a generic error will be displayed to the user. |

string GetReportType(string reportName, string udfData)

| Description | Returns the type of report as a wrReportType enum. |
|---|---|
| **Remark** | Any invalid type returns null. This will cause any report management methods to fail. |

string SaveReport(string companyId, string userId, string reportName, string reportXml)

| Description | Saves a report (reportName is fully qualified) |
|---|---|
| **Remark** | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

string DuplicateReport(string companyId, string userId, string reportName, string reportXml)

| Description | Duplicates a report (reportName is fully qualified). If this method is not provided, SaveReport will be called. |
|---|---|
| **Remark** | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string matches an id of any element in the language files then the string of that language element will be displayed to the user instead of the returned value. For more information see **Multi-Language Support.** |

string DeleteReport(string companyId, string userId, string reportName, [string udfData = null])

| Description | Deletes a report (reportName is fully qualified) |
|---|---|
| **Remark** | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

string RenameReport(string companyId, string userId, string reportName, string newReportName, [string udfData = null])

| Description | Renames a report (reportName & newReportName are fully qualified). If this method is not provided, DeleteReport and SaveReport will be called. |
|---|---|
| Remark | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

string AddFolder(string companyId, string userId, string folderName)

| Description | Adds a report folder (folderName is fully qualified). Folder should not be named to one that already exists. |
|---|---|
| Remark | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

string DeleteFolder(string companyId, string userId, string folderName)

| Description | Deletes a report folder (folderName is fully qualified). |
|---|---|
| Remark | Exago' default report template management will not allow a folder to be deleted that contains any reports.<br><br>Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

string RenameFolder(string companyId, string userId, string oldName, string newName)

| Description | Renames a report folder (folder names are fully qualified). Folder should not be moved to a location where a Folder with a matching name already exists. |
|---|---|
| Remark | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

string MoveFolder(string companyId, string userId, string oldName, string newName)

| Description | Moves a report folder (folder names are fully qualified). Folder should not be renamed to one that already exists. |
|---|---|

| Remark | Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |
|---|---|

## bool ExistFolder(string companyId, string userId, string folderName)

| Description | Determines if a folder exists. |
|---|---|
| Remark | Returns true or false. |

## List<string> GetTemplateList(string companyId, string userId)

| Description | Returns a list of strings or a string array containing the available templates. |
|---|---|
| Remark | If an error occurs return null and a generic error will be displayed to the user. |

## byte[] GetTemplate(string companyId, string userId, string templateName)

| Description | Returns a byte array containing the template. |
|---|---|
| Remark | If an error occurs return null and a generic error will be displayed to the user. |

## string SaveTemplate(string companyId, string userId, string templateName, byte[] templateData)

| Description | Saves a template |
|---|---|
| Remark | Returns an error message if one occurs, else return null.<br><br>**Note:** To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

## List<string> GetThemeList(string companyId, string userId, string themeType)

| Description | Returns a list of strings or a string array containing the available themes. |
|---|---|
| Remark | Valid values of themeType: "CrossTab", "Express".<br><br>If an error occurs return null and a generic error will be displayed to the user. |

## bool ExistTheme(string companyId, string userId, string themeType, string themeName)

| Description | Determines if a theme exists. |
|---|---|
| Remark | Valid values of themeType: "CrossTab", "Express".<br><br>Returns true or false. |

string GetThemeXml(string companyId, string userId, string themeType, string themeName)

| Description | Returns a string representing the theme in xml format. |
|---|---|
| Remark | Valid values of themeType: "CrossTab", "Express".<br><br>If an error occurs return null and a generic error will be displayed to the user. |

string SaveTheme(string companyId, string userId, string themeType, string themeName, string themeXml)

| Description | Saves a theme. |
|---|---|
| Remark | Valid values of themeType: "CrossTab", "Express".<br><br>Returns an error message if one occurs, else return null.<br><br>NOTE. To support multi-language functionality, if the returned string 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 returned value. For more information see **Multi-Language Support.** |

## Accessing SessionInfo in Folder Management

This section only applies to Folder Management using a .Net Assembly.

After adding a reference to WebReportsApi.dll you may gain additional flexibility by replacing companyId and userId with Exago' sessionInfo object in the methods listed above. The sessionInfo object grants access to all of the parameters, configuration, and report information of the Exago session. See **Exago SessionInfo** for more information.

To utilize the sessionInfoObject **replace** the companyId and userId parameters in the method signatures above with sessionInfo sessionInfo (see example below).

NOTE. To use the sessionInfoObject all the methods must be static.

Utilizing the sessionInfo object will allow the folder management code to access much more information about the user and/or Exago. For example the capability to access the sessionInfo object could be used to pass additional parameters to your folder management code such as the preferred language of the user or from which part of the host application they entered Exago.

NOTE. Passing the sessionInfo object will lock the folder management Assembly. In order to unlock the assembly, either IIS will need to be restarted, or the application pool running Exago will need to be recycled.

The following is an example of the method signature for GetReportXml to utilize the sessionInfo object.

<span style="color:blue">string</span> GetReportXml(<span style="color:blue">SessionInfo</span> sessionInfo, <span style="color:blue">string</span> reportName)

| Description | Returns a string containing the report in xml format. |
|---|---|
| **Remark** | If an error occurs return null and a generic error will be displayed to the user.<br><br>companyId and userId can still be retrived using the calls sessionInfo.SetupData.Parameters.GetValue('comapnyId') and sessionInfo.SetupData.Parameters.GetValue('userId') respectively.<br><br>The sessionInfo object must be the first parameter in the method. |

# 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 called 'WebReportsLog.txt' to the application's Temp path (specified in 'WebReports.xml'). 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.

There are three configurable verbosity levels for the logger. By default, Exago runs at the **Info** level.

- **Error** – Only logs error messages in the application.

- **Info** – Logs SQL statements, number of rows returned from each statement, and report execution information, as well as all **Error** messages.

  NOTE. Report execution information includes the following:

  On execution start: Start time, userId, companyId, full report name, filter summary.

  On 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 app, as well as all **Info** and **Error** messages.

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, as well as to customize and extend logging capability.

To create a custom config file, create a file called '`log4net.config`' in the Config directory of the Exago application. The following shows a sample config file:

```
<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>
```

## Custom Logging

Exago uses Apache Log4Net to handle custom logging. For more information and extensibility features, see: **Apache log4net**. See the following examples for some simple modifications:

### Changing Logfile Location

```
<file value="Path\To\Log.txt" />
```

Specifies the directory and filename for the log file.

### Changing Logging Level

```
<level value="INFO" />
```

Specifies the Exago logging level: `ERROR`, `INFO`, or `DEBUG`.

### Unlocking 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.

# Responsive Dashboards

Automatic scaling for running dashboards is implemented by use of an Action Event which is bundled but disabled by default. The event is fully customizable in case one would wish to alter the method of scaling.

This implementation makes use of the **OnDashboardResize Action Event**, which fires whenever an active dashboard has its size changed. This could occur on a change of the dashboard container or a change of the browser window.

> NOTE. Reports included on a dashboard are not responsive, and will not resize to fit a screen. However, all other elements of a dashboard, including visualizations, images, text fields, and filters, are responsive.

**Event Type**

OnDashboardResize

**Global Event Type**

Load

**References**

WebReportsApi.dll

**Namespaces**

WebReports.Api

**Custom Code**

```
sessionInfo.JavascriptAction.SetJsCode(@"
var dashboardItemsList = clientInfo.dashboard.dashboardItemsList;

if (clientInfo.dashboard.container.clientWidth < 700)
{
        var currentBottom = 0;
        for (var i=0; i<dashboardItemsList.length; i++)
        {
                var currentItem = dashboardItemsList[i];
                var canvasItem = currentItem.canvasItem;
                var dashboardElement = currentItem.Element;
                var designHeight = currentItem.DesignHeight;
                var designWidth = currentItem.DesignWidth;
                var scalingFactor = 1;
                var newHeight = designHeight;
                var newWidth = designWidth;

                if (designWidth > clientInfo.dashboard.container.clientWidth)
                {
                        var ratio = designHeight / designWidth;
                        newWidth = Math.floor(clientInfo.dashboard.container.clientWidth - 5);
                        if (currentItem.Type != 'Report') newHeight = Math.floor(newWidth * ratio);
                        scalingFactor = newWidth / designWidth;
                }
        canvasItem.SetBounds({left:0, top: currentBottom, width: newWidth, height: newHeight},
true);

                if(currentItem.Type == 'Text')
                {
                        if (scalingFactor != 1)
                        {
                                dashboardElement.style.transform = 'scale(' + scalingFactor + ')';
```

```
                        dashboardElement.style.transformOrigin = 'top';
                }
                else
                {
                        dashboardElement.style.transform = '';
                        dashboardElement.style.transformOrigin = '';
                }
        }

        currentBottom += newHeight + 5;
    }
}
else
{
    for (var i=0; i<dashboardItemsList.length; i++)
    {
            var currentItem = dashboardItemsList[i];
            var canvasItem = currentItem.canvasItem;
            var dashboardElement = currentItem.Element;
            var X = currentItem.DesignX;
            var Y = currentItem.DesignY;
            var Width = currentItem.DesignWidth;
            var Height = currentItem.DesignHeight;
            canvasItem.SetBounds({left:X, top: Y, width: Width, height: Height}, true);
            if (currentItem.Type == 'Text')
            {
                    dashboardElement.style.transform = '';
                    dashboardElement.style.transformOrigin = '';
            }
    }
}
");
```

return sessionInfo.JavascriptAction;

# Scheduler Queue

Scheduled reports can be controlled by an external custom queue whose code you administer via a .NET Assembly or an XML Soap Web Service. This may be desirable in a configuration which uses a large number of scheduler services, where customized load balancing is desired.

This queue must be referenced in each scheduler service in the eWebReportsScehduler.xml configuration file:

<**queue_service**>Assembly=Path\To\Assembly.dll;class=Scheduler.SchedulerQueue</**queue_servic e**>

And in the each of the Exago web application instances' config files (or via the API at runtime):

<**schedulerqueueservice**>Assembly=Path\To\Assembly.dll;class=Scheduler.SchedulerQueue</**sche dulerqueueservice**>

New helper classes have been added to the .NET Api: **QueueApi** and **QueueApiJob**. These are contained within the WebReports.Api.Scheduler namespace.

The following methods are required to be in your queue interface:

```
public static void Start(string serviceName)
```

Called from scheduler services to indicate when a specific service starts.

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

Called from the Exago UI to populate the jobs in the scheduler manager.

```
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.

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

Called from the Exago UI schedule manager to get the full job XML data for a specific 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.

# Exago API

The following chapter details the Application Programming Interface (API) offered by Exago.

## About

The Exago application consists of two basic parts: the user interface (and all of its support code), and the Api. The user interface is built entirely on top of the **.NET Api**. This means .NET Applications can interface directly with Exago. Non-.NET applications can interface through the **Web Service Api** which offers a subset of the .NET Api.

NOTE. Non Windows IIS applications can interface with the Exago Web Service Api as long as a Windows IIS Server is setup to run Exago and the Web Service Api.

## .NET API

To use the .NET Api the host application must include a reference to the assembly WebReportsApi.dll in its project.

NOTE. A configuration using an **Assembly Data Source** must include a reference to the assembly WebReportsAsmi.dll.

### Quick List of Name Spaces and Classes

Below the Name Spaces employed by Exago are listed. The name spaces utilized to integrate Exago display their classes below.

**WebReports.Api** – main namespace; contains Api class used in application integration.
    Api
**WebReports.Api.Charts** – Chart creation and processing classes.
**WebReports.Api.Common** – Common classes used by classes in other namespaces.
    ReportObjectFactory
    ReportObject
**WebReports.Api.Custom** – Classes used for custom work.
**WebReports.Api.Composite.Dashboard** – Classes used for Dashboard Reports.
    DashboardReport
**WebReports.Api.Data** – Data source and access classes.
    DataSource
    DataSourceCollection
**WebReports.Api.Export** – Report execution export classes.
**WebReports.Api.Reports** – All classes used in report creation.
    Filter
    Report
    ReportFilterCollection
    ReportSortCollection
    Sort
**WebReports.Api.ExecuteData** – Classes used for report execution processing.
**WebReports.Api.Roles** – Role creation and processing classes.
    DataObject

DataObjectCollection
DataObjectRow
DataObjectRowCollection
Folder
FolderCollection
General
Parameter
ParameterCollection
Role
RoleCollection
Security

**WebReports.Api.ReportMgmtBase –** Base class used for report and folder management.

## WebReports.Api

### Api Class

The Api class is the main interaction class between Exago and the host application. All API session parameters are accessed through this class. **An Api object should be the first thing that is created to interact with Exago.**

An Api object has the following properties:

- **Action** – Value that may indicate to execute a report or open Exago directly to the Report Design Grid or New Report Wizard. For the values 'EditReport', 'NewReport', 'NewCrossTabReport, and 'NewExpressReport' the main menu will be disabled.

    o Uses the enumeration wrApiAction(Default, Home, ExecuteReport, EditReport, NewReport, NewCrossTabReport, NewExpressReport, NewDashboardReport, ScheduleReport, ScheduleReportManager).

    o NOTE. If you have a Report object loaded then the value of Default will execute the report directly. Otherwise it will open the home page.

- **AppVirtualPath** – IIS virtual directory of Exago' location. This should be set to an absolute path (i.e. /ExagoWebSite/Exago).

- **DataSources** – DataSources collection. See **DataSourceCollection Class**.

- **Parameters** – Parameters collection. See **ParameterCollection Class**.

- **ReportObjectFactory –** Used to manage all report objects within the application. See **ReportObjectFactory Class**.

- **ReportScheduler –** Scheduler object. See **Report Scheduler Class**.

- **Roles** – Roles collection. See **RoleCollection Class**.

- **ShowTabs** – Boolean value. Set to False to hide the tabs and help button of Exago.

- **DefaultReportName** – String value used in conjunction with api.Action.

    o **When api.Action is set to NewReport, NewCrossTabReport or NewExpressReport:** The DefaultReportName provides the full path name for the report. The Info tab of the new report wizard will be hidden and the report designer will not display menus to rename the report or change its description.

    o **When api.Action is set to EditReport:** If DefaultReportName is any non-empty value the report designer will not display menus to rename the report or change its description.

An Api object has the following methods:

Constructor( )

| Remark | Do not call this method from the .NET Api. |
| --- | --- |

## Constructor(string appVirtualPath)

| Description | Initializes an Api object and sets the AppVirtualPath. |
| --- | --- |
| Remarks | Return value is void. |

## Constructor(string appVirtualPath, string configFile)

| Description | Initializes an Api object, sets the AppVirtualPath and loads the specified configuration. |
| --- | --- |
| Remarks | Can be used to load configuration other than WebReport.xml. Return value is void. |

## Constructor(string appVirtualPath, string configFile, string azurePath)

| Description | Initializes an Api object, sets the AppVirtualPath and loads the specified configuration from Azure. |
| --- | --- |
| Remarks | The specified Azure path must match that is app.config in the Exago installation. See **Azure Cloud Support** for more information. Return value is void. |

## GetUrlParamString( )

| Description | Calls GetUrlParamString("ExagoHome") |
| --- | --- |
| Remarks | Return value is void. |

## GetUrlParamString(string webPageName)

| Description | Returns the URL parameter string used to redirect browser or frame to Exago. Append this string to your Exago URL. |
| --- | --- |

## GetUrlParamString(string webPageName, boolean showErrorDetail)

| Description | Returns the URL parameter string used to redirect browser or frame to Exago. Append this string to your Exago URL. Set showErrorDetail to True to display detailed error messages. |
| --- | --- |

## WebReports.Api.Data

### DataSource Class

The DataSource class is used to set or override the data connection string of a pre-existing data source at runtime.

A DataSource object has the following properties:

- **Name** – Name of the data source.

- **DataConnStr** – Value of data connection string.

A DataSource object has no available methods.

### DataSourceCollection Class

This **collection should not be instantiated**; there is a single DataSourceCollection object that is accessed through the DataSources property of the Api object.

The DataSources property of an Api object has one available method:

GetDataSource(string dataSourceName)

| | |
|---|---|
| **Description** | Returns a DataSourceObject. Returns Null if the object is not found. |

## WebReports.Api.Common

### ReportObjectFactory Class

The ReportObjectFactory class is the entry point to application report object manipulation. This factory manages access to reports via API, updating that report's schedules when required (rename, delete), and creation of new reports. This class logically sits on top of ReportMgmtBase and ReportScheduler for higher level report management.

The ReportObjectFactory has the following properties:

- Active – The active report object. The active report object is whichever report was most recently created/loaded/deleted/etc.

The ReportObjectFactory has the following methods:

ReportObject Create(wrReportType reportType)

| Description | Create a new report object, it has yet to be saved into the report repository |
|---|---|
| Remarks | The created report object is made the active report object on return.<br><br>Both the Report class and the DashboardReport class inherit from ReportObject, a cast to the appropriate child class is required for more specific access to the report. |

ReportObject LoadFromRepository(string name, [string udfData], [bool setActive = true])

| Description | Load an existing report object from the report repository. |
|---|---|
| Remarks | The loaded report object is made the active report object on return.<br><br>Both the Report class and the DashboardReport class inherit from ReportObject, a cast to the appropriate child class is required for more specific access to the report. |

void Delete(string name, [string udfData = null])

| Description | Delete the provided report from the report repository. |
|---|---|
| Remarks | The deleted report object is made the active report object on return. |

void Delete(ReportObject reportObject, [bool deleteScheduledReports = true], [string udfData = null])

| Description | Delete the provided report object the report repository. |
|---|---|
| Remarks | The deleted report object is made the active report object on return. |

void Delete([bool deleteScheduledReports = true], [string udfData = null])

| Description | Delete the currently active report object from the report repository. |
|---|---|

void Rename(string name, string newName, [string udfData = null])

| Description | Rename the provided report in the report repository. |
|---|---|
| Remarks | The renamed report object is made the active report object on return. |

void Rename(ReportObject reportObject, string newName, [string udfData = null])

| Description | Rename the provided report object in the report repository. |
|---|---|
| Remarks | The renamed report object is made the active report object on return. |

void Copy(string name, string newName, [string udfData = null])

| Description | Copy the provided report in the report repository to another location in the report repository. |
|---|---|

void Copy(ReportObject reportObject, string newName, [string udfData = null])

| Description | Copy the provided report object to another location in the report repository. |
|---|---|

void SaveToRepository(ReportObject reportObject)

| Description | Save the provided report object to the report repository. If it already exists it will be overwritten. |
|---|---|

void SaveToApi(ReportObject reportObject)

| Description | Save the provided report object to an API area which can be accessed by the application once it's given control via Api.GetUrlParamString |
|---|---|

## ReportObject Class

The ReportObject class is an abstract class that all report objects derive from. It contains all properties and methods that are common for any type of report within the application.

The ReportObject has the following properties:

- **ExportType** – This value indicates which format to export the report.

    o Uses the enumeration wrExportType:

        ▪ Html (Default Report Viewer), Excel, Pdf, Rtf, Csv

- **IsEditAllowed** – Boolean value. If False the report object cannot be edited because the active role does not have access to one or more elements defined in the report object.

- **IsExecuteAllowed -** Boolean value. If False the report cannot be executed because the active role does not permit access to one or more Data objects on the report.

## WebReports.Api.Composite.Dashboards

### DashboardReport Class

The DashboardReport class allows dashboard reports to be executed and manipulated from the host application. This class does not need to be instantiated, it should be retrieved using methods defined in ReportObjectFactory. The DashboardReport class is derived from the ReportObject abstract class.

A DashboardReport object has the following properties:

ReportItems – A list of ReportItem objects, each representing a report contained within the dashboard. To find the index of a particular report on a dashboard:

- Enter the dashboard designer.
- Press Ctrl+Shift+I.
- Click on the desired report. The index will appear in the reports title bar.

### ReportItem Class

The ReportItem class represents a report that is contained within a composite report.

A ReportItem object has the following properties:

Report – The report that this ReportItem represents (fully qualified name).

The ReportItem object has the following methods:

void SetFilterValue(string filterName, wrFilterOperator filterOperator, List<string> filterValues)

| Description | Set the value for a promptable filter that exists on this report |
|---|---|
| Remarks | The number of entries in filterValues depends on the filter operator. |

void SetParameterValue(string parameterName, string parameterValue)

| Description | Set the value for a promptable parameter that exists on this report |
|---|---|

## WebReports.Api.Composite.Chained

## ChainedReport Class

The ChainedReport class allows Chained Reports to be executed and manipulated from the host application. This class does not need to be instantiated, it should be retrieved using methods defined in ReportObjectFactory. The ChainedReport class is derived from the ReportObject abstract class.

A ChainedReport object has the following properties:

ReportItems – A list of ReportItem objects, each representing a report contained within the chained report.

## ReportItem Class

See **ReportItem**.

# WebReports.Api.Reports

## Filter Class

The Filter class is used to modify filters at runtime. New Filter objects should be created by the NewFilter method of ReportFilterCollection.

A Filter object has the following properties:

- **AndOrWithNext** – Value indicates to use an 'and' or 'or' with the next filter added.

    o   Uses the enumeration wrFilterAndOrWithNext (And, Or).

- **DbName** - The fully qualified database (not mnemonic) name of the filter (i.e. 'vw_optionee.Last Name').

- **GroupWithNext** – Boolean indicating if the filter should be grouped with the next filter.

- **Operator** – The comparison operator.

    o   Uses the enumeration wrFilterOperator (EqualTo, NotEqualTo, LessThan, GreaterThan, LessThanOrEqualTo, GreaterThanOrEqualTo, StartsWith, EndsWith, Contains, Between, NotBetween, OneOf, NotOneOf).

- **Prompt** – Boolean indicating whether to prompt user for the value of this filter at time of execution.

- **Value** – value of the filter if it uses an Operator that only takes a single value. Dates must be in the following format YYYY-MM-DD.

- **DataValues** – values of the filter using an Operator that takes multiple values, such as One Of or Between.

A Filter object has no available methods.

## Report Class

The Report class allows standard and express reports to be executed directly from the host application. This class does not need to be instantiated, it should be retrieved using methods defined in ReportObjectFactory. The Report class is derived from the ReportObject abstract class.

A Report object has the following properties:

- **Filters** – Filters collection. See **ReportFilterCollection** class below.

- **Sorts –** Sorts collection. See **ReportSortCollection** class below.

- **ShowStatus –** Boolean value. This value indicates whether to show status window during execution. Default is True.

A Report Object has the following methods:

GetExecuteHtml( )

| | |
|---|---|
| **Description** | Executes the report and returns HTML (Default Report Viewer). |
| **Remarks** | The raw HTML can be used to populate a container in the host application. Does not include Exago paging Report Viewer. |

GetExecuteData( )

| | |
|---|---|
| **Description** | Executes the report and returns data as a byte array. |
| **Remarks** | Any export type can be executed in this way; use the ExportType property prior to calling this method to set the export type. |

GetExecuteSql( )

| | |
|---|---|
| **Description** | Returns all SQL statements that would be generated as a result of executing the report. |
| **Remarks** | There may be more than one SQL statement generated if the report uses more than one SQL data source. |

## ReportFilterCollection Class

This **collection should not be instantiated**; there is a single ReportFilterCollection object that is accessed through the Filters property of the Report object.

The Filters property of a Report object has one available method:

NewFilter( )

| | |
|---|---|
| **Description** | Returns a new Filter object and adds it to the collection. |
| **Remarks** | The returned Filter object needs to have all of its properties filled or an error will occur. |

## ReportSortCollection Class

This **collection should not be instantiated**; there is a single ReportSortCollection object that is accessed through the Sorts property of the Report object.

The Sorts property of a Report object has one available method:

NewSort( )

| | |
|---|---|
| **Description** | Returns a new Sort object and adds it to the collection. |
| **Remarks** | The returned Sort object needs to have all of its properties filled or an error will occur. |

## Sort Class

The Sort class is used to modify sorts at runtime. New Sort objects should be created by the NewSort method of ReportSortCollection.

A Sort object has the following properties:

- **DbName –** The fully qualified database (not mnemonic) name of the sort (i.e. 'vw_optionee.Last Name').

- **Direction –** Direction of the sort.

    o   Uses the enumeration wrSortDirection (Ascending, Descending)

A Sort object has no available methods.

## WebReports.Api.Roles

### DataObject Class

The DataObject class can allow or deny access to specific Data Objects for a particular user session.

A DataObject object has the following property:

- **Name** – The name (non-mnemonic) of the Data Object to include or exclude.

  - A DataObject in the DataObjectCollection will be excluded if the property IncludeAll is True and included if it is False.

A DataObject object has no available methods.

### DataObjectCollection Class

This **collection should not be instantiated**; there is a single DataObjectCollection object that is accessed through the DataObjects property of the Security object.

The DataObjectCollection has the following property:

- **IncludeAll** – Boolean indicating whether to include all of the Data Objects (default) or none of the Data Objects.

The DataObjects property of a Security object has the following method:

GetDataObject (string dataObjectName)

| | |
|---|---|
| **Description** | Returns the DataObject object or null if not found. |

NewDataObject ( )

| | |
|---|---|
| **Description** | Returns a new DataObject object and adds it to the collection. |
| **Remarks** | The returned DataObject object needs to have all of its properties filled or an error will occur. |

### DataObjectRow Class

The DataObjectRow class can set Row Level filters to Data Objects for a particular user session.

A DataObjectRow object has the following properties:

- **ObjectName** – The name (non-mnemonic) of the Data Object.

- **FilterString** – The filter string for the Data Object. The filter string will be placed into the SQL WHERE clause.

A DataObjectRow object has no available methods.

## DataObjectRowCollection Class

This **collection should not be instantiated**; there is a single DataObjectRowCollection object that is accessed through the DataObjectRoles property of the Security object.

The DataObjectRoles property of a Security object has the following method:

GetDataObject (string dataObjectRowName)

| | |
|---|---|
| **Description** | Returns the DataObjectRow object or null if not found. |

NewDataObjectRow ( )

| | |
|---|---|
| **Description** | Returns a new DataObjectRow object and adds it to the collection. |
| **Remarks** | The returned DataObjectRow object needs to have all of its properties filled or an error will occur. |

## Folder Class

The Folder class is used to allow or deny access to folders or sets folders as execute-only for a particular user session.

A Folder object has the following properties:

- **Name** – The name (non-mnemonic) of the folder to include/exclude.

  o The folder in the FolderCollection will be excluded if the property IncludeAll is True and included if it is False.

- **ReadOnly** – Boolean indicating whether a folder is read only. Default is False.

- **Propagate** – Not used: Parameters set for a folder are always propagated down to all of its subfolders unless parameters for specific child folder are set.

A Folder object has no available methods.

## FolderCollection Class

This **collection should not be instantiated**; there is a single FolderCollection object that is accessed through the Folders property of the Security object.

A FolderCollection object has the following property:

- **IncludeAll** – Boolean indicating whether to include all of the folders (default) or none.

- **ReadOnly** – Global read-only for all of the folders in the collection. Individual Folder objects can be set with a different ReadOnly property.

- **AllowManagement** – Boolean indicating whether or not to allow users to manage folders. Set to False to hide the Manage Folder Icon.

The Folders property of a Security object has the following method:

GetFolder (string folderName)

| | |
|---|---|
| **Description** | Returns the Folder object or null if not found. |

NewFolder ( )

| | |
|---|---|
| **Description** | Returns a new Folder object and adds it to the collection. |
| **Remarks** | The returned Folder object needs to have all of its properties filled or an error will occur. |

## General Class

The General class is utilized to overwrite the **General Settings** of the Administration Console. This **collection should not be instantiated**; there is a single General object that is accessed through the General property of the Role object.

The General property of the Role object has the following properties:

- **DbTimeout –** The amount of time (in seconds) to allow the database to execute a query before returning to Exago.

- **DateFormat –** Used to format dates on a report output.

- **CurrencySymbol –** The symbol prepended to currency numbers on a report output.

- **SeparatorSymbol –** The symbol used to separate 3 digits of number on a report output.

- **ReadFilterValues –** Boolean value that indicates whether to show a list of data values associated with a specific filter in the Report Filters window. In certain cases, allowing this can result in a lengthy delay of showing filter values, however, this depends on the amount of data, the complexity of data object, etc. If the delay is unacceptable, setting this value to 'False' will disable the feature.

- **ShowGrid –** Boolean value that indicates whether to show the grid in the Report Viewer. Also sets the 'Show Grid Lines' default to Report Viewer Report Options.

- **ReportVirtualPath –** IIS virtual path for the location of the report path.

The General property of the Role object does not have any available methods.

## Parameter Class

The Parameter class is used to create and modify **Parameters**.

A Parameter object has the following properties:

- **Id** – Name of the parameter.

  NOTE. Parameter names ARE case sensitive.

- **Value –** The value being stored in the parameter.

A Parameter object has the following available methods:

Constructor (<span style="color:blue">string</span> paramId, <span style="color:blue">string</span> paramValue )

| Description | Instantiates a Parameter object with the specified Id and Value. |
|---|---|

## ParameterCollection Class

This **collection should not be instantiated**; there is a single ParameterCollection object that is accessed through the Parameters property of the Api object.

The Parameter property of an Api object has the following method:

GetParameter(<span style="color:blue">string</span> parameterId)

| Description | Returns the Parameter object or null if not found. |
|---|---|

## Role Class

The Role class contains all of the information concerning General and Security parameters. A Role can be created at runtime and used for a single session or loaded from the roles that have been created through the Administration Console. For more information see **Roles**.

This **collection should not be instantiated**; there is a single RoleCollection object that is accessed through the Role property of the Api object.

A Role object may have the following properties:

- **General –** Access to all of the General Parameters. See General Class.

- **Security –** Access to all of the Security Parameters. See Security Class.

A Role object has one available method:

Activate( )

| Description | Makes this role active. |
|---|---|

## RoleCollection Class

This **collection should not be instantiated**; there is a single RoleCollection object that is accessed through the Roles property of the Api object.

GetRole(<span style="color:blue">string</span> roleId)

| Description | Returns the Role object or null if not found. |
|---|---|

NewRole( )

| | |
|---|---|
| **Description** | Returns a new Role object and adds it to the collection. |
| **Remarks** | The returned Role object needs to have all of its properties filled or an error will occur. |

## Security Class

The Security class contains all of the security parameters for a user session.

This **collection should not be instantiated**; there is a single Security object that is accessed through the Security property of the Role object.

The Security object has the following properties:

- **Folders** – Controls access to all of the FolderCollection parameters. See **FolderCollection** class.

- **DataObjects** – Controls access to all of the DataObjectCollection parameters. See **DataObjectCollection** class.

- **DataObjectRows** – Controls access to all of the DataObjectRowCollection parameters. See **DataObjectRowCollection** class.

There are no available methods for a Security object.

## WebReports.Api.Scheduler

### ReportScheduler Class

The ReportScheduler class can be used to schedule reports to run on a regular basis. Output can be emailed or stored in a repository. The output destination (email or storage) is normally set on a global basis. This API allows you to override the global setting for individual report schedules if desired.

A ReportScheduler object uses the following enumerations:

- **ReportScheduleInfo.WeekOfMonthType –** weeks of the month.

  - Uses the enumeration WeekOfMonthType (First, Second, Third, Fourth, Last)

- **ReportScheduleInfo.DayOfWeekType –** days of the week.

Uses the enumeration DayOfWeekType (Day, Weekday, Weekendday, Sunday, Monday, Tuesday, The API is designed to mirror the capabilities of the SchedulerWizard in the Exago main interface. There are a few concepts that will be helpful to understand in using the API. In general each API call requires the following information:

- Schedule name: A "handle" to refer to this schedule.

- Recurrence Information: usually a Start Date and Time, recurrence pattern and end condition. The end condition may be "No end condition" which indicates that the schedule should execute indefinitely according to the specified recurrence pattern. In certain instances, the recurrence information uses static Enumerations from the ReportScheduleInfo class.

- Email information: Includes To List, CC List, BCC List, Subject and Body. A new class ScheduleEmailInfo has been created to easily pass this information.

NOTE. a small number of Scheduler API calls don't follow the above pattern. For example, there are CreateOnceSchedule and CreateImmediateSchedule calls that don't use any recurrence information.

For each schedule type that uses a recurrence pattern the following rules apply:

- The start time can be passed in one of two ways:

  - As the Time Component of the startDate DateTime

  - As a separate TimeSpan schedTime

    The TimeSpan will always take precedence if not null. If the TimeSpan is null, the scheduler will use the Time element of startDate

- The end condition can be set in one of three ways

  - No End Condition: Report executions will continue indefinitely

  - End by number of occurrences: Executions will cease after N occurrences, where N is a passed parameter

  - End by Date: Executions will cease after a certain date where the date is a passed parameter.

Each type of call is overloaded to reflect the desired end condition. For example, there are three possible ways to create an "Every Weekday" schedule:

void CreateEveryWeekdaySchedule(string name, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo) //No end condition

void CreateEveryWeekdaySchedule(string name, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo) //End by number of occurrences

void CreateEveryWeekdaySchedule(string name, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo) //End by date

As noted above, report output can be sent through email or stored to a repository, and the choice can be made with the schedule. Passing a ScheduleEmailInfo object to the appropriate argument will tell the Exago Scheduler to send output through email based on the passed information. Passing null for the argument will tell the scheduler to archive the output for that schedule.

> NOTE. Archiving requires a Report Path to be set in the Scheduler Configuration XML. The Report Path tells the scheduler where to store the output.

A ReportScheduler object has the following methods:

void CreateImmediateSchedule(string name, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run immediately. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateOnceScheduleByDateTime(DateTime schedDateTime, string name, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedule a report to be run every weekday.<br><br>Report will be executed indefinitely. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateEveryWeekdaySchedule(string name, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run every weekday. Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
|---|---|
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateEveryWeekdaySchedule(string name, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run every weekday. Report will be executed until the specified rangeEndDate. |
|---|---|
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateEveryNDaySchedule(string name, int everyNDays, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a daily interval. Report will be executed indefinitely. |
|---|---|
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` everyNDays indicates the interval at which the schedule is run (e.g. every 10 days). If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateEveryNDaySchedule (string name, int everyNDays, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a daily interval. Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
|---|---|
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using |

| | api.ReportObjectFactory.LoadFromRepository()<br><br>everyNDays indicates the interval at which the schedule is run (e.g. every 10 days).<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |
|---|---|

void CreateEveryNDaySchedule(string name, int everyNDays, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a daily interval.<br><br>Report will be executed until the specified rangeEndDate. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using api.ReportObjectFactory.LoadFromRepository()<br><br>everyNDays indicates the interval at which the schedule is run (e.g. every 10 days).<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateWeeklySchedule(string name, int everyNWeeks, List<DayOfWeek> days, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a weekly interval.<br><br>Report will be executed indefinitely. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using api.ReportObjectFactory.LoadFromRepository()<br><br>everyNWeeks indicates the interval at which the schedule is run (e.g. every 2 weeks).<br><br>days indicates a list of the days each week to run the schedule.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateWeeklySchedule(string name, int everyNWeeks, List<DayOfWeek> days, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a weekly interval.<br><br>Report will be executed until the specified rangeEndDate. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using api.ReportObjectFactory.LoadFromRepository() |

| | everyNWeeks indicates the interval at which the schedule is run (e.g. every 2 weeks).<br><br>days indicates a list of the days each week to run the schedule.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateWeeklySchedule(string name, int everyNWeeks, List<DayOfWeek> days, DateTime rangeStartDate, int rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| **Description** | Schedules a report to be run on a weekly interval.<br><br>Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
| --- | --- |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>everyNWeeks indicates the interval at which the schedule is run (e.g. every 2 weeks).<br><br>days indicates a list of the days each week to run the schedule.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateMonthlyScheduleByNumericDay(string name, int everyNMonths, int numericDay, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| **Description** | Schedules a report to be run on a specific day each month.<br><br>Report will be executed indefinitely. |
| --- | --- |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>everyNMonths indicates the interval at which the schedule is run (e.g. every 2 months).<br><br>numericDay indicates the day of each month to run the schedule (e.g. 17).<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateMonthlyScheduleByNumericDay(string name, int everyNMonths, int numericDay, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| **Description** | Schedules a report to be run on a specific day each month.<br><br>Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
| --- | --- |

| Remarks | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>everyNMonths indicates the interval at which the schedule is run (e.g. every 2 months).<br><br>numericDay indicates the day of each month to run the schedule (e.g. 17).<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |
|---|---|

void CreateMonthlyScheduleByNumericDay(string name, int everyNMonths, int numericDay, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a specific day each month.<br><br>Report will be executed the specified rangeEndDate. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>everyNMonths indicates the interval at which the schedule is run (e.g. every 2 months).<br><br>numericDay indicates the day of each month to run the schedule (e.g. 17).<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateMonthlyScheduleByDescriptionDay(string name, int everyNMonths, WeekOfMonthType ordinal, DayOfWeekType dayOfWeek, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a described day each month.<br><br>Report will be executed indefinitely. |
|---|---|
| Remarks | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>everyNMonths indicates the interval at which the schedule is run (e.g. every 2 months).<br><br>ordinal used in context with dayOfWeek describe when during each month to run the scheduled report.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>ordinal and dayOfWeek are enumerations detailed at the beginning of this section |

| | See **SchedulerEmailInfo** class for information on emailInfo. |
|---|---|

void CreateMonthlyScheduleByDescriptionDay(string name, int everyNMonths, WeekOfMonthType ordinal, DayOfWeekType dayOfWeek, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a described day each month. |
|---|---|
| | Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` |
| | everyNMonths indicates the interval at which the schedule is run (e.g. every 2 months). |
| | ordinal used in context with dayOfWeek describe when during each month to run the scheduled report. |
| | If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. |
| | ordinal and dayOfWeek are enumerations detailed at the beginning of this section |
| | See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateMonthlyScheduleByDescriptionDay(string name, int everyNMonths, WeekOfMonthType ordinal, DayOfWeekType dayOfWeek, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| Description | Schedules a report to be run on a described day each month. |
|---|---|
| | Report will be executed the specified rangeEndDate. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` |
| | everyNMonths indicates the interval at which the schedule is run (e.g. every 2 months). |
| | ordinal used in context with dayOfWeek describe when during each month to run the scheduled report. |
| | If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. |
| | ordinal and dayOfWeek are enumerations detailed at the beginning of this section |
| | See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateYearlyScheduleByNumericDay(string name, int numericMonth, int numericDay, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| | |
|---|---|
| **Description** | Schedules a report to be run on a specific month and day each year. |
| | Report will be executed indefinitely. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` |
| | numericMonth indicates the month of each year to run the schedule (e.g. 3). |
| | numericDay indicates the day of each month to run the schedule (e.g. 17). |
| | If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. |
| | See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateYearlyScheduleByNumericDay(string name, int numericMonth, int numericDas, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| | |
|---|---|
| **Description** | Schedules a report to be run on a specific day each month. |
| | Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` |
| | numericMonth indicates the month of each year to run the schedule (e.g. 3). |
| | numericDay indicates the day of each month to run the schedule (e.g. 17). |
| | If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. |
| | See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateYearlyScheduleByNumericDay(string name, int numericMonth, int numericDay, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| | |
|---|---|
| **Description** | Schedules a report to be run on a specific day each month. |
| | Report will be executed the specified rangeEndDate. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()` |
| | numericMonth indicates the month of each year to run the schedule (e.g. 3). |
| | numericDay indicates the day of each month to run the schedule (e.g. 17). |
| | If schedTime is null the time component of schedDateTime will be used to determine the time the report is run. |
| | See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateYearlyScheduleByDescriptionDay(string name, int numericMonth, WeekOfMonthType ordinal, DayOfWeekType dayOfWeek, DateTime rangeStartDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| | |
|---|---|
| **Description** | Schedules a report to be run on a described day each month.<br><br>Report will be executed indefinitely. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>numericMonth indicates the month of each year to run the schedule (e.g. 3).<br><br>ordinal used in context with dayOfWeek describe when during each month to run the scheduled report.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>ordinal and dayOfWeek are enumerations detailed at the beginning of this section.<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateYearlyScheduleByDescriptionDay(string name, int numericMonth, WeekOfMonthType ordinal, DayOfWeekType dayOfWeek, DateTime rangeStartDate, int rangeEndAfterNOccurrences, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| | |
|---|---|
| **Description** | Schedules a report to be run on a described day each month.<br><br>Report will be executed the number of times specified in rangeEndAfterNOccurrences. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using `api.ReportObjectFactory.LoadFromRepository()`<br><br>numericMonth indicates the month of each year to run the schedule (e.g. 3).<br><br>ordinal used in context with dayOfWeek describe when during each month to run the scheduled report.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>ordinal and dayOfWeek are enumerations detailed at the beginning of this section See **SchedulerEmailInfo** class for information on emailInfo. |

void CreateYearlyScheduleByDescriptionDay(string name, int numericMonth, WeekOfMonthType ordinal, DayOfWeekType dayOfWeek, DateTime rangeStartDate, DateTime rangeEndDate, TimeSpan schedTime, SchedulerEmailInfo emailInfo)

| | |
|---|---|
| **Description** | Schedules a report to be run on a described day each month.<br><br>Report will be executed the specified rangeEndDate. |
| **Remarks** | Before calling this method be sure to activate the report you want to schedule using |

| | api.ReportObjectFactory.LoadFromRepository()<br><br>numericMonth indicates the month of each year to run the schedule (e.g. 3).<br><br>ordinal used in context with dayOfWeek describe when during each month to run the scheduled report.<br><br>If schedTime is null the time component of schedDateTime will be used to determine the time the report is run.<br><br>ordinal and dayOfWeek are enumerations detailed at the beginning of this section<br><br>See **SchedulerEmailInfo** class for information on emailInfo. |
|---|---|

## SchedulerEmailInfo Class

The SchedulerEmailInfo class is utilized used by methods of ReportScheduler objects.

A SchedulerEmailInfo object has the following property:

- **toAddrs –** list of email addresses and/or distribution lists for the 'To' field of the email.

- **ccAddrs –** list of email addresses and/or distribution lists for the 'cc' field of the email.

- **bcAddrs –** list of email addresses and/or distribution lists for the 'bc' field of the email.

  NOTE. If toAddrs, ccAddrs and bcAddrs are all null, Exago will attempt to archive the report to the **Scheduler Repository**.

- **Subject –** The subject of the email.

- **body –** The body text of the email.

## Other Notes

### Using MySQL through the .NET Api

For Exago .NET Api to connect to a MySQL database add the following to the host application's web.config file.

```
<system.data>
  <DbProviderFactories>
    <remove invariant="MySql.Data.MySqlClient" />
    <add name="MySQL Data Provider" invariant="MySql.Data.MySqlClient" description=".NET
Framework Data     Provider    for MySQL"
type="MySql.Data.MySqlClient.MySqlClientFactory, MySql.Data, Version=6.3.6.0,
      Culture=neutral,    PublicKeyToken=c5687fc88969c44d" />
  </DbProviderFactories>
</system.data>
```

Additionally if the host application does not already have a MySQL ADO.NET Data Adapter copy the file 'Exago/bin/MySql.Data.dll' to the host application's bin folder.

## Examples

To access the Exago Api, add a reference to the assembly WebReportsApi.dll to your project.

In all of the examples below the return value should be checked for validity. The examples below have omitted validations for clarity.

### Create Api object

```
// create WebReports API object passing Exago virtual path;
Api api = new Api("ExagoServer/Exago");
```

### Adding/modifying a Role

```
// create a new runtime Role (is automatically made active)
Role role = api.Roles.NewRole();
// -- OR --
// accessing a pre-created Role and making it active
Role role = api.Roles.GetRole("Admin");
role.Activate();
```

### Adding Folder security to role

```
// start with privileges to all folders for this user session (this is the default)
role.Security.Folders.IncludeAll = true;

// disallow access to folder 'Stew's Reports' (and any subfolders)
Folder folder = role.Security.Folders.NewFolder();
folder.Name = "Stew's Reports";

// make folder 'Summary Reports' (and any subfolders) read only
Folder folder = role.Security.Folders.NewFolder();
folder.Name = "Summary Reports";
folder.ReadOnly = true;
```

### Adding Data Object security to role

```
// start with privileges to all data objects (this is the default)
role.Security.DataObjects.IncludeAll = true;

// disallow access to data object 'vw_cancellation'
DataObject dataObject = role.Security.DataObjects.NewDataObject();
dataObject.Name = "vw_cancellation";
```

### Adding Data Object Row security to role

```
// don't allow this user to view rows from the 'vw_grant' object with a
// 'Grant Date' value of '2000-01-01'
DataObjectRow dataObjectRow = role.Security.DataObjectRows.NewDataObjectRow();
dataObjectRow.ObjectName = "vw_grant";
dataObjectRow.FilterString = @"""Grant Date"" <> '2000-01-01'";
```

### Setting up several general user session parameters for role (overrides individual global general parameters)

```
// set global date format for this user
role.General.DateFormat = "dd/MM/yyyy";
```

```
// set currency symbol for this user
role.General.CurrencySymbol = "kr";
```

## Modifying the data connection string of a specific data source

```
// set data connection string for a specific datasource
DataSource dataSource = api.DataSources.GetDataSource("MyDb");
dataSource.DataConnStr = "Server=SVR;Database=db1;uid=sa;pwd=dba;";
```

## Modifying a parameter value

```
// modify a parameter value
Parameter parameter = api.Parameters.GetParameter("asOfDate");
parameter.Value = "2007-06-01";
```

## Setting a data column alias

```
// set column alias
api.Entities.GetEntity("vw_webrpt_optionee").ColumnMetadatas.SetColumnAlias("Hire Date", "Date
of Hire");
```

## Starting Exago - At this point if you want to run the Exago applications, do the following:

```
// setup URL
string url = "http://MyDomainServer/Exago/" + api.GetUrlParamString();
Response.Redirect(url);

// or you can redirect any control that can be set to a URL
this.ReportIFrame.Attributes["src"] = url;
```

## Executing a report directly from the host application – You can combine setting user session information as above with report execution. To do that, just omit the redirect above and do the following:

```
// load a specific report and return Report object (make sure to check return value)
Report report = (Report) api.ReportObjectFactory.LoadFromRepository(@"Stew Meyers' Reports\My
Report");


// add a sort
Sort sort = report.Sorts.NewSort();
sort.DbName = "vw_optionee.First Name";
sort.Direction = wrSortDirection.Ascending;
report.Sorts.Add(sort);

// add a filter
Filter filter = report.Filters.NewFilter();
filter.DbName = "vw_grant.Grant Date";
filter.Operator = wrFilterOperator.LessThan;  // default is EqualTo
filter.Value = "20070501";  // filter dates are entered in YYYYMMDD sequence
filter.AndOrWithNext = wrFilterAndOrWithNext.And;  // default is And
filter.GroupWithNext = false;  // default is false
filter.Prompt = true; // default is false
```

```csharp
// Set the Execute behavior to run in the Report Viewer (default is export to PDF)
report.ExportType = wrExportType.Html; // "Html" refers to the Report Viewer

// should Report Viewer be opened in new browser window
report.OpenNewWindow = false;  // default is false
report.ShowStatus = false;  // default is true

// saves a temporary version of the report to be used for execution
api.ReportObjectFactory.SaveToApi(report);
```

## Start report execution

```csharp
// setup URL
string url = "http://MyDomainServer/Exago/" + api.GetUrlParamString();
Response.Redirect(url);

// or you can redirect any control that can be set to a URL
this.ReportIFrame.Attributes["src"] = url;
```

## Executing a dashboard report directly from the host application:

```csharp
api.Action = wrApiAction.ExecuteReport;
        DashboardReport report = Api.ReportObjectFactory.LoadFromRepository(@"Reports\My
Dashboard") as DashboardReport;
                report.ReportItems[0].SetParameterValue("productname", "Parm1");
                report.ReportItems[0].SetFilterValue("Employees.EmployeeID",
wrFilterOperator.EqualTo, new List<string>() { "3" });
                report.ReportItems[0].SetFilterValue("Orders.OrderDate",
wrFilterOperator.GreaterThanOrEqualTo, new List<string>() { "1996-07-04 01:00:00" });

                report.ReportItems[1].SetParameterValue("productname", "Parm2");
                report.ReportItems[1].SetFilterValue("Employees.EmployeeID",
wrFilterOperator.EqualTo, new List<string>() { "5" });
                report.ReportItems[1].SetFilterValue("Orders.OrderDate",
wrFilterOperator.GreaterThanOrEqualTo, new List<string>() { "1996-07-04 01:00:00" });

                api.ReportObjectFactory.SaveToApi(report);

                string url = @"[Exago Install Path] /" + api.GetUrlParamString("Home");
        this.ReportIFrame.Attributes["src"] = url;
```

## Scheduling a report with a filter and emailing results:

```csharp
//load in the report
        Report report = api.ReportObjectFactory.LoadFromRepository(@"Reports\Employee
Performance") as Report;

//specify the export type of the scheduled report
        report.ExportType = wrExportType.Pdf;

//set email information
        List<string> toList = new List<string>();
        toList.Add("emailaddress@example.com");
        SchedulerEmailInfo info = new SchedulerEmailInfo(toList);
```

```
//set a filter on the report
        Filter filter = report.Filters.NewFilter();
        filter.DbName = "Employees.FirstName";
        filter.Operator = wrFilterOperator.EqualTo;
        filter.Value = "Janet";
        filter.Prompt = true;


//create a schedule
        api.ReportScheduler.CreateImmediateSchedule("Schedule Test", info);


//save the report to the api
        api.ReportObjectFactory.SaveToApi(report);
```

# SOAP Web Service API

The SOAP Web Service Api provides a way for non-.NET applications to interface with Exago. The functionality provided by the Web Service is a subset of the .NET Api, and includes basic methods to launch Exago and execute reports directly from the host application.

The Web Service must be installed on a Microsoft Windows Server running IIS and be able to access the Exago application directory directly or through an IIS virtual directory. For more information see **Web Service Installation**.

## Quick List of Web Service Methods

**Main:**
GetUrlParamString
GetUrlParamString2
InitializeApi
InitializeApi2
SetAction
SetAction2
SetDefaultReportName
SetGeneralProperty
SetGeneralProperties

**Data:**
DataObject_Add
DataObject_Add2
DataObject_SetColumnAlias
DataSource_AddXmlType
DataSource_Modify
Join_Add

**Folders:**
Folder_Add
Folder_Delete
Folder_Exist
Folder_Rename

**Parameters:**
Parameter_Add
Parameter_Modify

**ReportObjects:**
ReportObject_Activate
ReportObject_Delete
ReportObject_Duplicate

**Dashboards:**
Dashboard_SetReportParameterValue
Dashboard_SetReportFilterValue

**Reports:**
Report_AddFilter
Report_AddFilterValue
Report_AddSort
Report_GetExecuteData
Report_GetExecuteHtml
Report_GetReportListXml
Report_GetReportXml
Report_SetFilterValue
Report_SetParams
Report_TestExecute

**Roles:**
Role_GetRoles
Role_Activate
Role_Add
Role_AddDataObject
Role_AddDataObjectRow
Role_AddFolder
Role_SetCurrencySymbol
Role_SetDateFormat
Role_SetDbTimeout
Role_SetDecimalSymbol
Role_SetLanguageFile
Role_SetReadFilterValues
Role_SetReportVirtualPath
Role_SetScheduleManagerVie
wLevel
Role_SetSeparatorSymbol
Role_SetShowGrid
Role_SetShowScheduleRepor
ts
Role_SetShowScheduleRepor
tsEmail

Role_SetShowScheduleReportsManager

**Schelduer:**
Report_CreateImmediateSchedule
Report_CreateImmediateScheduleForArchiving
Report_CreateOnceScheduleByDateTime
Report_CreateOnceScheduleByDateTimeForArchiving
Report_CreateEveryWeekdaySchedule
Report_CreateEveryWeekdayScheduleForArchiving
Report_CreateEveryNDaySchedule
Report_CreateEveryNDayScheduleForArchiving
Report_CreateWeeklySchedule
Report_CreateWeeklyScheduleForArchiving
Report_CreateMonthlyScheduleByNumericDay
Report_CreateMonthlyScheduleByNumericDayForArchiving
Report_CreateMonthlyScheduleByWeekAndDay
Report_CreateMonthlyScheduleByWeekAndDayForArchiving
Report_CreateYearlyScheduleByNumericDay
Report_CreateYearlyScheduleByNumericDayForArchiving
Report_CreateYearlyScheduleByWeekAndDay
Report_CreateYearlyScheduleByWeekAndDayForArchiving

## Full Description of Web Service Methods

This section provides detailed information on the available web service api methods.

**Types of Web Service methods:**

- **Main Methods**

- **Data Methods**

- **Folder Methods**

- **Parameter Methods**

- **ReportObject Methods**

- **Dashboard Methods**

- **Report Methods**

- **Role Methods**

- **Scheduler Methods**

## Main Methods

This section lists the main web service methods used to access Exago.

### void GetUrlParamString(string apiId)

| | |
|---|---|
| **Description** | Returns the URL parameter string. Points to ExagoHome.aspx. |
| **Remarks** | **This is always the last method called.**<br><br>Appended the returned URL to your Exago application URL and redirect the user. |

### void GetUrlParamString2(string apiId, string webPageName, boolean showErrorDetail)

| | |
|---|---|
| **Description** | Returns the URL parameter string. Points to the specified home page. Set showErrorDetail to True to display detailed error messages. |
| **Remarks** | **This is always the last method called.**<br><br>Appends the returned URL to your Exago application URL and redirects the user. |

### string InitalizeApi( )

| | |
|---|---|
| **Description** | Returns an apiId as a string that is used in all subsequent calls. |
| **Remarks** | **This is always the first method called.** |

### string InitializeApi2(string configFn)

| | |
|---|---|
| **Description** | Returns an apiId as a string that is used in all subsequent calls. |
| **Remarks** | Can be used instead of InitializeApi to specify a configuration file other than WebReports.xml |

### bool SetAction(string apiId, int action, string defaultFolderName)

| | |
|---|---|
| **Description** | Set the Action property of the Api object. The action dictates the behavior of Exago when you call **GetUrlParamString**.<br>Returns Boolean indicating success/failure. |
| **Remarks** | Valid values for action are:<br>0: Default – Executes a report on **ReportObject_Activate**, otherwise opens the home page.<br>1: Home – opens the home page.<br>2: ExecuteReport – Executes the active report.<br>3: EditReport – opens the<br>4: NewReport – opens the new report wizard directly.<br>5: NewCrossTabReport – opens the new crosstab report wizard directly.<br>6: NewExpressReport – opens the new express report wizard directly.<br>7: NewDashboardReport – opens a new dashboard designer directly.<br>8: Schedule Report – opens the new schedule report wizard directly.<br>9: ScheduleReportManager – opens the new schedule report wizard directly. |

## bool **SetAction2**(string apiId, int action, string defaultFolderName, Boolean showTabs)

| | |
|---|---|
| **Description** | Set the Action property of the Api object. The action dictates the behavior of Exago when you call **GetUrlParamString**.<br>Returns Boolean indicating success/failure. |
| **Remarks** | Valid values for action are:<br>0: Default – Executes a report on **ReportObject_Activate**, otherwise opens the home page.<br>1: Home – opens the home page.<br>2: ExecuteReport – Executes the active report.<br>3: EditReport – opens the<br>4: NewReport – opens the new standard report wizard directly.<br>5: NewCrossTabReport – opens the new crosstab report wizard directly.<br>6: NewExpressReport – opens the new express report wizard directly.<br>7: NewDashboardReport – opens a new dashboard designer directly.<br>8: Schedule Report – opens the new schedule report wizard directly.<br>9: ScheduleReportManager – opens the new schedule report wizard directly. |

## bool **SetDefaultReportName**(string apiId, string defaultReportName)

| | |
|---|---|
| **Description** | Set the DefaultReportName property of the Api object. The DefaultReportName is used in conjunction with the Action property of the Api to modify the behavior of Exago when you call **GetUrlParamString**.<br>Returns Boolean indicating success/failure. |
| **Remarks** | The Default report name is a string providing the fully qualified path of the report.<br><br>This function's effect will change based on the set value of the Action.<br><br>When the Action is set to NewReport, NewCrossTabReport or NewExpressReport: The DefaultReportName provides the full path name for the report. The Info tab of the new report wizard will be hidden and the report designer will not display menus to rename the report or change its description.<br><br>When the Action is set to EditReport: If DefaultReportName is any non-empty value the report designer will not display menus to rename the report or change its description. |

## bool **SetGeneralProperty**(string apiId, string propertyName, string propertyValue)

| | |
|---|---|
| **Description** | Modify any of the **General Settings** in the Administration Console for the session. |
| **Remarks** | The propertyName must match the name used in the configuration file **WebReports.xml** for the setting you want to modify. Ex. 'showexpressreports' controls the **Feature/UI Setting** 'Show Express Reports'.<br><br>The propertyValue type will depend on the setting using the following rules based on how the property is shown in the Administration Console:<br> 1. If the setting is True/False then use a boolean.<br> 2. If the setting is enterable text (ex. chart colors) use a string.<br> 3. If the setting is a number use an int.<br> 4. If the setting is a dropdown of predefined values use the enumeration specified below. |

| | DefaultOutputType:<br> 0. Html (Default Report Viewer)<br> 1. Excel<br> 2. Pdf<br> 3. Rtf<br> 4. Csv<br> 6. Default<br><br>DateTimeTreatedAs:<br> 0. Date<br> 1. Time (Time filters are not supported.)<br> 2. DateTime<br><br>ScheduleManagerViewLevel:<br> 0. Current User at Current Company<br> 1. All Users at Current Company<br> 2. All Users at All Companies<br><br>UserPreferenceStorage:<br> 0. Cookie<br> 1. ExternalInterface:<br> 2. None<br><br>ExcelExportTarget:<br> 0. v2003<br> 1. v2007<br> 2. v2010<br><br>DefaultFilterExecutionWindow<br><br>SchemaAccessType:<br>Default<br>Datasource<br>Metadata |
|---|---|

## bool SetGeneralProperties(string apiId, string[] propertyName, string[] propertyValue)

| | |
|---|---|
| **Description** | Allows multiple SetGeneralProperty calls to be grouped together to avoid making many web service calls. |
| **Remarks** | The length the propertyName array and the propertyValue array must be equal.<br><br>See remarks above in the SetGeneralProperty method. |

## Data Methods

This section lists the web service methods used to create, modify or delete Data Objects, Data Sources and Joins.

## bool DataObject_Add(string apiId, string dataSourceName, int objectType, string, objectName, string mnemonicName, string keyName, string categoryName, string sqlStmt, string parmaterIds, string tenants)

| | |
|---|---|
| **Description** | Adds a Data Object. Returns Boolean indicating success/failure. |

| Remarks | Valid objectType values are:<br>0: database table<br>1: database view<br>2: database function<br>3: database stored procedure<br>4: database SQL statement<br>5: web service method<br><br>parameterIds is a comma delimited list whose values will be passed to the data object.<br><br>tenants is a comma delimited list of columns and parameters. Ex. 'db_col1,paramId1,db_col,paramId2' |
|---|---|

bool **DataObject_Add2**(string apiId, string dataSourceName, int objectType, string, objectName, string, objectId, string mnemonicName, string keyName, string categoryName, string sqlStmt, string parmaterIds, string tenants)

| Description | Adds a Data Object. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | Unlike DataObject_Add this function includes an objectId. This allows for multiple Data Objects with the same name. The objectID should be a unique value. |

bool **DataObject_Add3**(string apiId, string dataSourceName, int objectType, string, objectName, string, objectId, string schemaName, string mnemonicName, string keyName, string categoryName, string sqlStmt, string parmaterIds, string tenants)

| Description | Adds a Data Object. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | Unlike DataObject_Add this function includes an objectId and schemaName.<br>ObjectId allows for multiple Data Objects with the same name and should be a unique value.<br>SchemaName sets the database schema of the object. |

bool **DataObject_SetColumnAlias**(string apiId, string objectName, string columnName, string alias)

| Description | Sets the alias of a specific data column. Returns Boolean indicating success/failure. |
|---|---|

bool **DataSource_AddXmlType**(string apiId, string xml, string categoryNames)

| Description | Loads Xml into Exago as a data source. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | Xml can be Excel worksheet type or compatible with .NET DataSet.<br><br>The Data Object can appear in multiple categories using a comma delimiter. |

bool **DataSource_Modify**(string apiId, string dataSourceName, string dataConnStr)

| Description | Modifies the connection string of a Data Source. Returns Boolean indicating success/failure. |
|---|---|

## bool **Join_Add**(string apiId, string dataObjectFromName, string columnFromName, string dataObjectToName, string columnToName, int joinType int relationType, int weight)

| Description | Adds a Data Object Join. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | Valid relationType values are:<br>0:one-to-one<br>1:one-to-many<br><br>Valid joinType values are:<br><br>0:inner<br>1: left outer<br>2: right outer<br>3: full outer |

## Folder Methods

This section lists the web service methods used to create, modify or delete Folders.

## bool **Folder_Add**(string apiId, string parentName, string name)

| Description | Adds a report folder. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | parentName is relative to the Report Path and should not contain slashes.<br><br>Method will fail if a parent folder named parentName does not exist. |

## bool **Folder_Delete**(string apiId, string folderName)

| Description | Deletes a report folder. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | folderName is relative to the Report Path.<br><br>Method will fail if the report is not empty. |

## bool **Folder_Exist**(string apiId, string folderName)

| Description | Checks if a report folder exists. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | folderName is relative to the Report Path. |

## bool **Folder_Rename**(string apiId, string oldName, string newName)

| Description | Renames a report folder exists. Returns Boolean indicating success/failure. |
|---|---|
| **Remarks** | Both folder names are relative to the Report Path. |

## Parameter Methods

This section lists the web service methods used to create, modify or delete Parameters.

bool **Parameter_Add**(string apiId, string parameterId, string parameterValue, int dataType, bool isHidden, string promptText)

| Description | Adds a parameter. Returns Boolean indicating success/failure. |
|---|---|
| **Remarks** | Valid dataType values are:<br>0: string<br>1: date<br>2: integer<br>5: decimal |

bool **Parameter_Modify**(string apiId, string parameterId, string parameterValue)

| Description | Modifies a parameter value. Returns Boolean indicating success/failure. |
|---|---|

bool **Parameter_ModifyMultiple**(string apiId, string[] parameterIds, string[] parameterValues)

| Description | Modifies multiple parameter values. Returns Boolean indicating success/failure. |
|---|---|
| **Remarks** | The length of the parameterIds and parameterValues arrays must be the same. |

## ReportObject Methods

This section lists the web service methods used to create, modify or delete Report objects. A Report object is any type of report supported by the application (currently Report_ or Dashboard_).

bool **ReportObject _Activate**(string apiId, string reportName)

| Description | Activates an existing report. Returns Boolean indicating success/failure. |
|---|---|
| **Remarks** | Use backslashes to delineate subfolders. |

NOTE. Before calling any report or dashboard method call ReportObject_Activate to specify which Report object to modify.

bool **ReportObject _Delete**(string apiId, string reportName)

| | |
|---|---|
| **Description** | Deletes an existing report. Returns Boolean indicating success/failure. |
| **Remarks** | Use backslashes to delineate subfolders. |

bool **ReportObject _Duplicate**(string apiId, string srcReportName, string destReportName)

| | |
|---|---|
| **Description** | Creates a duplicate copy of an existing report (srcReportName) and provides a new name (destReportName). Returns Boolean indicating success/failure. |
| **Remarks** | Use backslashes to delineate subfolders. |

## Dashboard Methods

bool **Dashboard_SetReportFilterValue**(string apiId, int reportIndex, string filterName, wrFilterOperator filterOperator, List<string> filterValues)

| | |
|---|---|
| **Description** | Sets the dashboard value for a promptable filter that exists on the specified report contained within the dashboard |
| **Remarks** | To find the reportIndex of a particular report on a dashboard: Enter the dashboard designer. Press Ctrl+Shift+I. Click on the desired report. The index will appear in the reports title bar. The number of items in filterValues depends on the filter operator. |

bool **Dashboard_SetReportParameterValue**(string apiId, int reportIndex, string parameterName, string parameterValue)

| | |
|---|---|
| **Description** | Sets the dashboard value for a promptable parameter that exists on the specified report contained within the dashboard |
| **Remarks** | To find the reportIndex of a particular report on a dashboard: Enter the dashboard designer. Press Ctrl+Shift+I. Click on the desired report. The index will appear in the reports title bar. |

## Report Methods

bool **Report_AddFilter**(string apiId, string filterName, int filterOperator, string filterValue, int andOrWithNext, bool groupWithNext, bool promptForValue)

| | |
|---|---|
| **Description** | Adds a filter to a report. Returns Boolean indicating success/failure. |

| Remarks | Valid filterOperator values are:<br>0: equal to<br>1: less than<br>2: less than or equal to<br>3: greater than<br>4: greater than or equal to<br>5: not equal to<br>6: starts with<br>7: not starts with<br>8: ends with<br>9: not ends with<br>10: contains<br>11: not contains<br>12: between<br>13: not between<br>14: one of<br>15: not one of<br><br>filterValue can contain multiple values. Delineate values with '\|~\|' (pipe tilde pipe).<br><br>Valid andOrWithNext values are:<br>0: and<br>1:or<br><br>Dates must be in the following format YYYY-MM-DD. |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## bool **Report_AddFilterValue**(string apiId, int index, string value)

| Description | Adds a value to a filter that accepts multiple values (ex 'one of' filters).<br>Returns Boolean indicating success/failure. |
|-------------|----------------------------------------------------------------------------------------------------------------------------|
| Remarks | Index indicates which filter to add the value to.<br>This method can only be used on filters with the following operators: 'one of', 'not one of'. |

## bool **Report_AddSort**(string apiId, string sortName, int sortDirection)

| Description | Adds a sort to a report. Returns Boolean indicating success/failure. |
|-------------|---------------------------------------------------------------------|
| Remarks | Valid sortDirection values are:<br>0: ascending<br>1: descending |

## bool **Report_RemoveSort**(string apiId, string sortName)

| Description | Removes a sort from a report. Returns Boolean indicating success/failure. |
|-------------|--------------------------------------------------------------------------|

## bool **Report_SetSorts**(string apiId, string[] sortName, int[] sortDirection)

| Description | Replaces any existing sorts of a report with the new sorts specified. Returns Boolean indicating success/failure. |
|-------------|------------------------------------------------------------------------------------------------------------------|

| Remarks | Valid sortDirection values are:<br>0: ascending<br>1: descending<br><br>If the lengths of the sortName and sortDirection arrays are not equal the following behavior will occur:<br> sortNames without a corresponding sortDirection will default to ascending.<br> sortDirections without a corresponding sortName will be ignored. |
|---|---|

## byte[] Report_GetExecuteData(string apiId)

| Description | Executes a report directly and returns data as a byte array. |
|---|---|
| Remarks | Any export type can be used with this method. Use Report_setParams method to set the export type prior to this call. |

## string Report_GetExecuteHtml(string apiId)

| Description | Executes a report directly and returns HTML as a string. |
|---|---|
| Remarks | This can be used to populate a container in the host application.<br><br>HTML will not contain Report Viewer. |

## string Report_GetReportListXml(string apiId)

| Description | Returns the hierarchical structure of reports and folders as an Xml string. |
|---|---|
| Remarks | Returned list adheres to the active Role if set.<br><br>See **Report and Folder Storage/Management** for an example of the Xml output. |

## string Report_GetReportXml(string apiId)

| Description | Returns the hierarchical structure of the active report an Xml string. |
|---|---|

## bool Report_SetFilterValue(string apiId, int index, int subIndex, string value)

| Description | Sets the value of a filter. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | subIndex is used for filters with multiple values such as 'one of' or 'between' filters.<br><br>Set subIndex to -1 for single value operators.<br><br>Dates must be in the following format YYYY-MM-DD. |

## bool Report_SetParams(string apiId, int exportType, bool openNewWindow, bool showStatus)

| Description | Sets report execution parameters. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | Valid exportType values are:<br>0: html (Dafault Report Viewer)<br>1: excel<br>2: pdf<br>3: rtf<br>4: csv |

## Role Methods

This section lists the web service methods used to create, modify or delete Roles.

### string **Role_GetRoles**(string apiId)

| Description | Returns the list of existing Roles as an Xml string. |
|---|---|

### bool **Role_Activate**(string apiId, string roleId)

| Description | Activates a pre-created role. Returns Boolean indicating success/failure. |
|---|---|

NOTE. Before calling any of the following methods call Role_Activate to specify which role to modify.

### bool **Role_Add**(string apiId, bool includeAllFolders, bool foldersReadOnly, bool allowFolderManagement, bool includeAllDataObjects)

| Description | Creates a new temporary run-time role. Returns Boolean indicating success/failure. |
|---|---|

### bool **Role_AddDataObject**(string apiId, string objectName)

| Description | Adds a Data Object to the role. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | If includeAllDataObjects is True this method will exclude the Data Object and vice versa.<br><br>objectName is the database value not the mnemonic. |

### bool **Role_AddDataObjectRow**(string apiId, string objectName, string filterString)

| Description | Adds a Data Object row to the role. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | objectName is the database value not the mnemonic.<br><br>filterString should be standard SQL to go into the WHERE clause. |

bool **Role_AddFolder**(string apiId, string folderName, bool readOnly)

| Description | Adds a Report Folder to the role. Returns Boolean indicating success/failure. |
|---|---|
| Remarks | If includeAllFolders is True this method will exclude the Folder and vice versa. |

bool **Role_ SetCurrencySymbol** (string apiId, string currencySymbol)

| Description | Overrides global currency symbol. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetDateFormat** (string apiId, string dateFormat)

| Description | Overrides global date format. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetDbTimeout** (string apiId, int dbTimeout)

| Description | Overrides maximum seconds the database is allowed to execute a query before timing out. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetDecimalSymbol**(string apiId, string decimalSymbol)

| Description | Overrides global decimal symbol. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetLanguageFile**(string apiId, string languageFile)

| Description | Overrides global Language File. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetReadFilterValues**(string apiId, bool readFilterValues)

| Description | Overrides whether to allow users to see database values in filter dropdowns. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetReportVirutalPath** (string apiId, string reportPath)

| Description | Overrides report virtual path. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetScheduleManagerViewLevel** (string apiId, int scheduleManagerViewLevel)

| Description | Sets the level of view privilege for the user session Returns Boolean indicating success/failure. |
|---|---|
| Remarks | Valid values for scheduleManagerViewLevel are:<br>0: Current users (requires parameter userId be set)<br>1: Current Company (requires parameter companyId be set)<br>2: All |

bool **Role_ SetSeparatorSymbol** (string apiId, string separatorSymbol)

| Description | Overrides global numeric separator symbol. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_SetServerTimeZoneOffset**(string apiId, decimal serverTimeZoneOffset)

| Description | Overrides global Server Time Zone Offset. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetShowGrid** (string apiId, bool showGrid)

| Description | Overrides global numeric separator symbol. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetShowScheduleReports** (string apiId, bool showScheduleReports)

| Description | Overrides whether to show the schedule report option. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetShowScheduleReportsEmail** (string apiId, bool showScheduleReportsEmail)

| Description | Overrides whether to show the schedule reports instant email option. Returns Boolean indicating success/failure. |
|---|---|

bool **Role_ SetShowScheduleReportsManager**(string apiId, bool showScheduleReportsManager)

| Description | Overrides whether to show the schedule reports management option. Returns Boolean indicating success/failure. |
|---|---|

## Scheduler Methods

This section lists the web service methods used to create Schedules for Reports to be emailed or Archived.

Before calling any of the following methods, call Report_Activate to specify which report to schedule and Report_SetParams to set an export format.

NOTE. There are two methods for type of schedule: a regular method and a 'ForArchiving' method. The regular method will email the report while the ForArchiving method will save the report to the Scheduler Repository. For more information on archiving schedules see **Saving Scheduled Reports to External Repository**.

NOTE. Dates must be in the following format YYYY-MM-DD. Times must be in the following format HH:MM[:SS] (24-hour format).

bool **Report_CreateImmediateSchedule**(string apiId, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| Description | Schedules a report to run and emailed immediately. |
|---|---|

| | |
|---|---|
| | Returns Boolean indicating success/failure. |
| **Remarks** | name: The name of the schedule as it appears in the Schedule Manager |
| | toAddrArray: The array of email addresses and/or distribution lists for the 'To' field of the email. If none of To, CC or BCC are set, Exago will attempt to archive scheduled reports. |
| | ccAddrArray: The array of email addresses and/or distribution lists for the 'CC' field of the email. If none of To, CC or BCC are set, Exago will attempt to archive scheduled reports. |
| | bccAddrArray: The array of email addresses and/or distribution lists for the 'BCC' field of the email. If none of To, CC or BCC are set, Exago will attempt to archive scheduled reports.</param> |
| | subject: The subject line of the email |
| | body: The body text of the email |

## bool **Report_CreateImmediateScheduleForArchiving**(string apiId, string name)

| | |
|---|---|
| **Description** | Schedules a report to run and archived immediately. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | name: The name of the schedule as it appears in the Schedule Manager |

## bool **Report_CreateOnceScheduleByDateTime**(string apiId, string dateStr, string timeStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed at a specific date and time.. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | dateStr: The date to run the schedule. If the timeStr parameter is null, the scheduler will use the time value of this parameter |
| | timeStr: The time to run the schedule. If null, the scheduler will use the time value of the dateStr parameter |
| | NOTE. See remarks in **Report_CreateImmediateSchedule** toAddrArray, ccAddrArray & bccAddrArray |

## bool **Report_CreateOnceScheduleByDateTimeForArchiving**(string apiId, string dateStr, string timeStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived at a specific date and time.. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | dateStr: The date to run the schedule. If the timeStr parameter is null, the scheduler |

| | will use the time value of this parameter |
|---|---|
| | timeStr: The time to run the schedule. If null, the scheduler will use the time value of the dateStr parameter |

bool **Report_CreateEveryWeekdaySchedule**(string apiId, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed every weekday. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | startDateStr: The date to begin running the schedule. |
| | timeStr: The time to run the schedule. If null, the scheduler will use the time value of the startDateStr parameter. |
| | Three parameters are used to determine when to end a recurring schedule: bool NoEndDate, int endOccurrences, string endDateStr. These parameters adhere to the following logic. |
| | If noEndDate is true, the report will run indefinitely. Else if endOccurrences is greater than zero, the report will execute that many times. Else the schedule will execute until the date represented in endDateStr. |
| | NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool **Report_CreateEveryWeekdayScheduleForArchiving**(string apiId, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived every weekday. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | startDateStr: The date to begin running the schedule. |
| | timeStr: The time to run the schedule. If null, the scheduler will use the time value of the startDateStr parameter. |
| | NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of noEndDate, endOccurrences & endDateStr. |

bool **Report_CreateEveryNDaySchedule**(string apiId, int everyNDays, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed every N days. |
| | Returns Boolean indicating success/failure. |

| | |
|---|---|
| **Remarks** | everyNDays: Indicates the interval at which to run the schedule (e.g. every 10 days).<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr.<br><br>NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool **Report_CreateEveryNDayScheduleForArchiving**(string apiId, int everyNDays, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived every N days.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | everyNDays: Indicates the interval at which to run the schedule (e.g. every 10 days).<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |

bool **Report_CreateWeeklySchedule**(string apiId, int everyNWeeks, int[] dayNums, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed on a weekly interval.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | everyNWeeks: Indicates the interval at which to run the schedule (e.g. every 2 weeks).<br><br>dayNums: Days on which the schedule is to be run. Valid values are:<br>    1: Sunday<br>    2: Monday<br>    3: Tuesday<br>    4: Wednesday<br>    5: Thursday<br>    6: Friday<br>    7: Saturday<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr.<br><br>NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool **Report_CreateWeeklyScheduleForArchiving**(string apiId, int everyNWeeks, int[] dayNums, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived on a weekly interval. |

| | |
|---|---|
| | Returns Boolean indicating success/failure. |
| **Remarks** | everyNWeeks: Indicates the interval at which to run the schedule (e.g. every 2 weeks).<br><br>dayNums: Days on which the schedule is to be run. Valid values are:<br>    1: Sunday<br>    2: Monday<br>    3: Tuesday<br>    4: Wednesday<br>    5: Thursday<br>    6: Friday<br>    7: Saturday<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |

bool **Report_CreateMonthlyScheduleByNumericDay**(string apiId, int everyNMonths, int numericDay, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed on a specific day each month.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | everyNMonths: Indicates the interval at which to run the schedule (e.g. every 2 months).<br><br>numericDay: The numeric day of each month (e.g. 17) on which to run the schedule<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr.<br><br>NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool **Report_CreateMonthlyScheduleByNumericDayForArchiving**(string apiId, int everyNMonths, int numericDay, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived on a specific day each month.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | everyNMonths: Indicates the interval at which to run the schedule (e.g. every 2 months).<br><br>numericDay: The numeric day of each month (e.g. 17) on which to run the schedule<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |

bool **Report_CreateMonthlyScheduleByWeekAndDay**(string apiId, int everyNMonths, int weekOfMonthNum, int dayOfWeekNum, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed on a "described" day each month, consisting of the week and the day.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | everyNMonths: Indicates the interval at which to run the schedule (e.g. every 2 months).<br><br>weekOfMonthNum: The 'described' week of each month (e.g. 'Third') on which to run the schedule. Used in conjunction with dayOfWeek. Valid values are:<br>    1: First<br>    2: Second<br>    3: Third<br>    4: Fourth<br>    5: Last<br><br>dayOfWeekNum: The 'described' day of each week (e.g. 'Weekday') on which to run the schedule. Valid values are:<br>    1: Sunday<br>    2: Monday<br>    3: Tuesday<br>    4: Wednesday<br>    5: Thursday<br>    6: Friday<br>    7: Saturday<br>    8: Day<br>    9: Weekday<br>    10: Weekend Day<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr.<br><br>NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool **Report_CreateMonthlyScheduleByWeekAndDayForArchiving**(string apiId, int everyNMonths, int weekOfMonthNum, int dayOfWeekNum, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived on a "described" day each month, consisting of the week and the day.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | everyNMonths: Indicates the interval at which to run the schedule (e.g. every 2 months).<br><br>weekOfMonthNum: The 'described' week of each month (e.g. 'Third') on which to run the schedule. Used in conjunction with dayOfWeek. Valid values are:<br>    1: First<br>    2: Second<br>    3: Third |

| | 4: Fourth |
|---|---|
| | 5: Last |
| | |
| | dayOfWeekNum: The 'described' day of each week (e.g. 'Weekday') on which to run the schedule. Valid values are: |
| |     1: Sunday |
| |     2: Monday |
| |     3: Tuesday |
| |     4: Wednesday |
| |     5: Thursday |
| |     6: Friday |
| |     7: Saturday |
| |     8: Day |
| |     9: Weekday |
| |     10: Weekend Day |
| | |
| | NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |

bool **Report_CreateYearlyScheduleByNumericDay**(string apiId, int numericMonth, int numericDay, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed on a specific day each year. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | everyNMonths: Indicates the interval at which to run the schedule (e.g. every 2 months). |
| | numericMonth: The numeric Month of each year (e.g. 3) on which to run the schedule |
| | numericDay: The numeric day of each month (e.g. 17) on which to run the schedule |
| | NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |
| | NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool **Report_CreateYearlyScheduleByNumericDayForArchiving**(string apiId, int numericMonth, int numericDay, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived on a specific day each year. |
| | Returns Boolean indicating success/failure. |
| **Remarks** | everyNMonths: Indicates the interval at which to run the schedule (e.g. every 2 months). |
| | numericMonth: The numeric Month of each year (e.g. 3) on which to run the schedule |
| | numericDay: The numeric day of each month (e.g. 17) on which to run the schedule |

| | |
|---|---|
| | NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |

bool **Report_CreateYearlyScheduleByWeekAndDay**(string apiId, int numericMonth, int weekOfMonthNum, int dayOfWeekNum, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name, string[] toAddrArray, string[] ccAddrArray, string[] bccAddrArray, string subject, string body)

| | |
|---|---|
| **Description** | Schedules a report to be run and emailed on a specific day each year.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | numericMonth: The numeric Month of each year (e.g. 3 = March) on which to run the schedule<br><br>weekOfMonthNum: The 'described' week of each month (e.g. 'Third') on which to run the schedule. Used in conjunction with dayOfWeek. Valid values are:<br>    1: First<br>    2: Second<br>    3: Third<br>    4: Fourth<br>    5: Last<br><br>dayOfWeekNum: The 'described' day of each week (e.g. 'Weekday') on which to run the schedule. Valid values are:<br>    1: Sunday<br>    2: Monday<br>    3: Tuesday<br>    4: Wednesday<br>    5: Thursday<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr.<br><br>NOTE. See remarks in **Report_CreateImmediateSchedule** for a description of toAddrArray, ccAddrArray & bccAddrArray. |

bool Report_CreateYearlyScheduleByWeekAndDayForArchiving(string apiId, int numericMonth, int weekOfMonthNum, int dayOfWeekNum, string startDateStr, string timeStr, bool noEndDate, int endOccurrences, string endDateStr, string name)

| | |
|---|---|
| **Description** | Schedules a report to be run and archived on a specific day each year.<br><br>Returns Boolean indicating success/failure. |
| **Remarks** | numericMonth: The numeric Month of each year (e.g. 3 = March) on which to run the schedule<br><br>weekOfMonthNum: The 'described' week of each month (e.g. 'Third') on which to run the schedule. Used in conjunction with dayOfWeek. Valid values are:<br>    1: First<br>    2: Second<br>    3: Third<br>    4: Fourth |

| | 5: Last<br><br>dayOfWeekNum: The 'described' day of each week (e.g. 'Weekday') on which to run the schedule. Valid values are:<br>    1: Sunday<br>    2: Monday<br>    3: Tuesday<br>    4: Wednesday<br>    5: Thursday<br><br>NOTE. See remarks in **Report_CreateEveryWeekdaySchedule** for a description of startDateStr, timeStr, noEndDate, endOccurrences & endDateStr. |
|---|---|

## Examples C#

The following examples demonstrate the capabilities of the Web Service Api using C#.

It is important that the first call instantiate an Api object. After making all the desired changes the final call should be GetUrlParamString. Then redirect the user to Exago' Url concatenated with the string GetUrlParamString returns.

In all of the examples below the return value should be checked for validity. The examples below have omitted these checks for clarity.

### Create Api object and initialize

```
// create an instance of the web service
//(web service needs to have been discovered in your application)
ExagoWebService.Api api = new ExagoWebService.Api();

// initialize API; returns an ID which is used in subsequent calls
string apiId = api.InitializeApi();
```

### Adding/modifying a Role

```
// create a new runtime Role (is automatically made active)
api.Role_Add(apiId, true, false, true, true);
// -- OR --
// accessing a pre-created Role and making it active
api.Role_Activate(apiId, "Admin");
```

### Adding Folder security to role

```
// disallow access to folder 'Stew's Reports' (and any subfolders)
api.Role_AddFolder(apiId, "Stew's Reports", false);

// make folder 'Summary Reports' (and any subfolders) read only
api.Role_AddFolder(apiId, "Summary Reports", true);
```

### Adding Data Object security to role

```
// disallow access to data object 'vw_cancellation'
api.Role_AddDataObject(apiId, "vw_cancellation");
```

### Adding Data Object Row security to role

```
// don't allow this user to view rows from the 'vw_grant' object with a
```

```
// 'Grant Date' value of '2000-01-01'
api.Role_AddDataObjectRow(apiId, "vw_grant", @"""Grant Date"" <> '2000-01-01'");
```

**Setting up several general user session parameters for role (overrides individual global general parameters)**

```
// set global date format for this user
api.Role_SetDateFormat(apiId, "dd/MM/yyyy");

// set currency symbol for this user
api.Role_SetCurrencySymbol(apiId, "kr");
```

**Modifying the data connection string of a specific data source**

```
// set data connection string for a specific datasource
api.DataSource_Modify(apiId, "MyDb", "Server=SVR;Database=db1;uid=sa;pwd=dba;");
```

**Modifying a parameter value**

```
// modify a parameter value
api.Parameter_Modify(apiId, "asOfDate", "2007-06-01");
```

**Adding a data object**

```
api.DataObject_Add(apiId, "eowin", 0, "optionee", "Optionee Dynamic", "OPT_NUM", "Dynamic",
null, null, null));
```

**Setting data column alias**

```
api.DataObject_SetColumnAlias(apiId, "vw_webrpt_optionee", "Hire Date", "Date of Hire"));
```

**Adding a data object join**

```
api.Join_Add(apiId, "optionee", "OPT_NUM", "fn_webrpt_grant", "Optionee Number", 1, 10));
```

**Starting Exago - At this point if you want to run the Exago applications, do the following:**

```
// setup URL
string url = "http://MyDomainServer/Exago/" + api.GetUrlParamString(apiId);
Response.Redirect(url);

// or you can redirect any control that can be set to a URL
this.ReportIFrame.Attributes["src"] = url;
```

**Executing a report directly from the host application – You can combine setting user session information as above with report execution. To do that, just omit the redirect above and do the following:**

```
// activate specific report
api.ReportObject_Activate(apiId, @"Stew Meyers' Reports\My Report")

// add a sort
api.Report_AddSort(apiId, "vw_optionee.First Name", 0);

// add a filter
api.Report_AddFilter(apiId, "vw_grant.Grant Date", 1, "20070501", 0, false, true);
```

```
// set other execution params
api.Report_SetParams(apiId, 0, false, false);
```

## Start report execution

```
// setup URL
string url = "http://MyServer/Exago/" + api.GetUrlParamString(apiId);
Response.Redirect(url);

// or you can redirect any control that can be set to a URL
this.ReportIFrame.Attributes["src"] = url;
```

## Scheduler Examples

```
Api api = new Api(apiPath, "AdventureWorks.XML");
api.Report.Load(@"DevReports\Adventure Works\Product Locations and Inventory");
api.Report.ExportType = wrExportType.Pdf;

ReportScheduler scheduler = api.ReportScheduler;

List<string> toAddrs = new List<string>();
toAddrs.Add("foo@bar.com");
List<string> ccAddrs = new List<string>();
ccAddrs.Add("foo@bar.com");
List<string> bccAddrs = new List<string>();
bccAddrs.Add("foo@bar.com");

DateTime dt = new DateTime(2013, 5, 16, 11, 00, 0);
DateTime dt2 = new DateTime(2013, 6, 15, 10, 20, 0);
TimeSpan ts = new TimeSpan(17, 20, 25);

scheduler.CreateOnceScheduleByDateTime(dt, "Once by datetime", toAddrs);

scheduler.CreateDailySchedule(true, 3, dt, true, 0, dt, "Daily No End Date");
scheduler.CreateDailySchedule(false, 2, dt, false, 5, dt2, "Daily two occurrences");
scheduler.CreateDailySchedule(false, 2, dt, false, 0, dt2, string.Format("Daily end by {0}_{1}",
dt2.Month, dt2.Day), toAddrs, ccAddrs);

List<DayOfWeek> days = new List<DayOfWeek>();
days.Add(DayOfWeek.Wednesday);
days.Add(DayOfWeek.Wednesday);
days.Add(DayOfWeek.Sunday);
scheduler.CreateWeeklySchedule(2, days, dt, true, 0, null, "Weekly no end date");
scheduler.CreateWeeklySchedule(2, days, dt, false, 2, null, "Weekly 2 occurrences");
scheduler.CreateWeeklySchedule(2, days, dt, false, 0, dt2, "Weekly end by date", toAddrs,
ccAddrs, bccAddrs, ts);
```

## Examples PHP

The following examples demonstrate the capabilities of the Web Service Api using PHP.

It is important that the first call instantiate an Api object. After making all the desired changes the final call should be GetUrlParamString. Then redirect the user to Exago' Url concatenated with the string GetUrlParamString returns.

In all of the examples below the return value should be checked for validity. The examples below have omitted these checks for clarity.

## Create Api object and initialize

```
$client = new SoapClient('http://MyServer/ExagoApi/Api.asmx?wsdl');

$r = $client->InitializeApi();
$apiId = $r->InitializeApiResult;
```

## Activate a role

```
$r = $client->Role_Activate(array('apiId' => $apiId, 'roleId'=>'Admin'));
$result = $r->Role_ActivateResult;
```

## Activate a report

```
$r = $client->ReportObject_Activate(array('apiId' => $apiId, 'reportName'=>'Stew Meyers'
Reports\\My Report'));
```

## Get URL

```
$url = "http://MyServer/Exago/" . $client->GetUrlParamString(array('apiId' => $apiId);
```

# REST Web Service API

Exago provides a RESTful API Web Service useful for language-agnostic application support. If enabled, an admin can access and modify elements of the Exago Application using standard HTTP methods in either XML or JSON representation.

The REST API is installed to the Exago Web Service directory. For information on how to install the Exago Web Service, see **Web Service Installation**.

In order to enable the REST API, the following key must be added to the Exago Web Service's AppSettings.config file:

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

Once enabled, the REST API can be accessed through the "/REST/*" URL from the Web Service virtual directory.

The format of the data sent and returned can be either XML or JSON. The format of the content being sent is defined through the Content-Type header of the request; the format of the data returned is defined through the Accept header of the request:

> Content-Type: application/(json/xml)
> Accept: application/(json/xml)

For any request, parameters can be appended to the URL in the following form:
"/Path/To/Resource/?param1=value1&param2=value2"

A session id parameter "sid" is required for most resources. Each resource may additionally define parameters unique to that URL pattern.


## Authorization

The REST API can be accessed in either an authorized or unauthorized state. Certain URLs and/or HTTP Methods and/or properties within a resource may only be available or may behave differently depending on whether the request is authorized or not.

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 and password found within the configuration file currently being accessed.

**Basic Authorization**
When using basic authorization, the authorization header is constructed as follows:

1. The username and password are combined into a string "username:password".
2. The resulting string literal is encoded using Base64.

3. "Basic" and a space are placed before the encoded string.

For example, if the username is "Brian" and the password is "open sesame" then the authorization header would be:

Authorization: Basic QWxhZGRpbjpvcGVuIHN1c2FtZQ==

NOTE. The password 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 lacking a username and password 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 password using the HMAC-SHA256 algorithm.
2. The resulting signature is then encoded using Base64.
3. The username and a colon is put before the encoded signature.
4. "ExagoKey" and a space are placed before the username:encoded string literal.

For example, if the username is "Brian" and the password is "open sesame" then depending on the request the authorization header might be something like:

Authorization: ExagoKey Brian:6HZE5tCWjsjbJY+VXQg3UzXlK/jeoGhbm25YDXiHWdE=

Using ExagoKey does not send the password with each request, making it more secure than Basic Authorization. However 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/reporting/api/Sessions?config=myconfig" the absolute request path would be "/reporting/api/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 SID, or a string of zero length if no SID 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).

## List of Resources

| URL | GET | POST | PUT | PATCH | DELETE |
|---|:---:|:---:|:---:|:---:|:---:|
| **/Sessions** | | ✓ | | | |
| **/Sessions/{sid}** | ✓ | | ✓ | ✓ | ✓ |
| **/DataSources** | ✓ | ✓ | | | |
| **/DataSources/{id}** | ✓ | | ✓ | ✓ | ✓ |
| **/Joins** | ✓ | ✓ | | | |
| **/Joins/{id}** | ✓ | | | | |
| **/Roles** | ✓ | ✓ | | | |
| **/Roles/{id}** | ✓ | | ✓ | ✓ | ✓ |
| **/Roles/{id}/Settings** | ✓ | | ✓ | ✓ | |
| **/Roles/{id}/Entities** | ✓ | | ✓ | ✓ | |
| **/Roles/{id}/Folders** | ✓ | | ✓ | ✓ | |
| **/Roles/{id}/DataObjectRows** | ✓ | | ✓ | ✓ | |
| **/Settings** | ✓ | | ✓ | ✓ | |
| **/Parameters** | ✓ | ✓ | | | |
| **/Parameters/{id}** | ✓ | | ✓ | ✓ | ✓ |
| **/Entities** | ✓ | ✓ | | | |
| **/Entities/{id}** | ✓ | | ✓ | ✓ | ✓ |
| **/Entities/{id}/Fields** | ✓ | | | | |
| **/Entities/{id}/Fields/{fid}** | ✓ | | ✓ | ✓ | |
| **/Functions** | ✓ | ✓ | | | |
| **/Functions/{id}** | ✓ | | ✓ | ✓ | ✓ |
| **/ServerEvents** | ✓ | ✓ | | | |
| **/ServerEvents/{id}** | ✓ | | ✓ | ✓ | ✓ |
| **/Folders/{id}** | ✓ | ✓ | | | ✓ |
| **/Reports/List** | ✓ | | | | |
| **/Reports/Execute/{id}** | | ✓ | | | |

### Sessions

This resource provides access to session information. A session is the first object created in order to interface with the REST Api. "POST /Sessions" should be the first service called by the user. If successful, it will return a Session ID parameter that is the resource path for the REST services.

**POST /Sessions**
Creates a new session within the Exago application. This is the only URL that does not require a session ID in some form.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *config* | Optionally specify the base configuration to load for this session. |

| Input Data | **SessionResource** |
|---|---|
| Output Data | **SessionResource** |

### GET /Sessions/{sid}

Retrieves the session information for a previously created session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Path | {sid} | The session id to retrieve. |
| Output Data | **SessionResource** | |

### PUT /Sessions/{sid}

Updates a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Path | {sid} | The session id to update. |
| Input Data | **SessionResource** | |
| Output Data | **SessionResource** | |

### PATCH /Sessions/{sid}

Updates specified properties of a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Path | {sid} | The session id to update. |
| Input Data | **SessionResource** | |
| Output Data | **SessionResource** | |

### DELETE /Sessions/{sid}

Deletes a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Path | {sid} | The session id to delete. |

### DataSources

This resource provides access to the data sources.

### GET /DataSources

Retrieves the data sources in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| | *entity* | Only return joins that join the entity id. |
| Output Data | **DataSourceListItemResource** | A list of **DataSourceResource** objects. |

## GET /DataSources/{id}

Retrieve a single data source in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The data source id to retrieve. |
| Output Data | **DataSourceResource** | |

## POST /DataSources

Creates a new data source.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **DataSourceResource** | |
| Output Data | **DataSourceResource** | |

## PUT /DataSources/{id}

Update a single data source in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The data source id to update. |
| Input Data | **DataSourceResource** | |

## PATCH /DataSources/{id}

Update specified properties for single data source in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |

| Parameters | *sid* | Required session id. |
|---|---|---|
| Path | {id} | The data source id to update. |
| Input Data | **DataSourceResource** | |

## DELETE /DataSources/{id}
Delete a single data source in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The data source id to delete. |

## Joins

This resource provides access to join information.

### GET /Joins
Retrieves the joins in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| | *entity* | Only return joins that join the entity id. |
| Output Data | **JoinListItemResource** | A list of **JoinResource** objects. |

### GET /Joins/{id}
Retrieves a single join in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The join id to retrieve. |
| Output Data | **JoinResource** | |

### POST /Joins
Creates a new join.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **JoinResource** | |
| Output Data | **JoinResource** | |

### PUT /Joins/{id}
Update a single join in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The join id to update. |
| Input Data | **JoinResource** | |

### PATCH /Joins/{id}
Update specified properties for single join in a session. If a property is not specified, it will not be changed.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The join id to update. |
| Input Data | **JoinResource** | |
| Output Data | **JoinResource** | |

### DELETE /Joins/{id}
Delete a single join in a session.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The join id to delete. |

## Roles

This resource provides access to role information.

### GET /Roles
Retrieves the roles in a session.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Output data | **RoleListItemResource** | List of **RoleResource** objects. |

## GET /Roles/{id}
Retrieves a single role in a session.

| Info | | Description |
|------|------|-------------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to retrieve. |
| Output data | **RoleResource** | |

## POST /Roles
Creates a new role.

| Info | | Description |
|------|------|-------------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **RoleResource** | |
| Output data | **RoleResource** | |

## PUT /Roles/{id}
Update a single role in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|------|------|-------------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input Data | **RoleResource** | |

## PATCH /Roles/{id}
Update specified properties for single role in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|------|------|-------------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input Data | **RoleResource** | |

## DELETE /Roles/{id}
Delete a role in a session.

| Info | | Description |
|------|------|-------------|
| Authorization | Required | |

| Parameters | *sid* | Required session id. |
|---|---|---|
| Path | {id} | The role id to delete. |

## GET /Roles/{id}/Settings
Retrieves the settings for a single role in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to retrieve. |
| Output data | **RoleSettingsResource** | |

## PUT /Roles/{id}/Settings
Updates the settings for a single role in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleSettingsResource** | |

## PATCH /Roles/{id}/Settings
Updates specified properties for the settings for a single role in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleSettingsResource** | |

## GET /Roles/{id}/Entities
Retrieves the entity security for a single role in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to retrieve. |
| Output data | **RoleEntityResource** | |

## PUT /Roles/{id}/Entities

Updates the entity security for a single role in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleEntityResource** | |

### PATCH /Roles/{id}/Entities
Updates specified properties for the entity security for a single role in a session. If a property is not specified, it will not be changed.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleEntityResource** | |

### GET /Roles/{id}/Folders
Retrieves the folder security for a single role in a session.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to retrieve. |
| Output data | **RoleFoldersResource** | |

### PUT /Roles/{id}/Folders
Updates the folder security for a single role in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleFoldersResource** | |

### PATCH /Roles/{id}/Folders
Updates specified properties for the folder security for a single role in a session. If a property is not specified, it will not be changed.

| Info | | Description |
| --- | --- | --- |

| | | |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleFoldersResource** | |

### GET /Roles/{id}/DataObjectRows
Retrieves the data object row security for a single role in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to retrieve. |
| Output data | **RoleDataObjectRowResource** | |

### PUT /Roles/{id}/DataObjectRows
Updates the data object row security for a single role in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleDataObjectRowResource** | |

### PATCH /Roles/{id}/DataObjectRows
Updates specified properties for the data object row security for a single role in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The role id to update. |
| Input data | **RoleDataObjectRowResource** | |

## Settings

### GET /Settings
Retrieves the general settings of the configuration.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |

| Parameters | *sid* | Required session id. |
|---|---|---|
| Output data | **SettingsResource** | |

## PUT /Settings
Updates the general properties of the configuration in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input data | **SettingsResource** | |

## PATCH /Settings
Updates the general properties of the configuration. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input data | **SettingsResource** | |

## Parameters

This resource provides access to all the parameters within the application.

## GET /Parameters
Retrieves the parameters in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Output Data | **ParameterListItemResource** | A list of **ParameterResource** objects. |

## POST /Parameters
Creates a new parameter.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **ParameterResource** | |

## GET /Parameters/{id}

---

Retrieves a single parameter in a session.

| Info | | Description |
| --- | --- | --- |
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Path | {id} | The parameter id to retrieve. |
| Output Data | **ParameterResource** | |

### PUT /Parameters/{id}
Update a single parameter in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The parameter id to update. |
| Input Data | **ParameterResource** | |

### PATCH /Parameters/{id}
Update specified properties for single parameter in a session. If a property is not specified, it will not be changed.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The parameter id to update. |
| Input Data | **ParameterResource** | |

### DELETE /Parameters/{id}
Delete a single parameter in a session.

| Info | | Description |
| --- | --- | --- |
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The parameter id to delete. |

## Entities

This resource provides access to all the entities within the application.

### GET /Entities
Retrieves the entities in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Output Data | **EntityListItemResource** | A list of **EntityResource** objects. |

## POST /Entities
Creates a new entity.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **EntityResource** | |

## GET /Entities/{id}
Retrieves a single entity in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to retrieve. |
| Output Data | **EntityResource** | |

## PUT /Entities/{id}
Updates a single entity in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to update. |
| Input data | **EntityResource** | |

## PATCH /Entities/{id}
Updates specified properties for a single entity in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to update. |

| Input data | **EntityResource** |
|---|---|

## DELETE /Entities/{id}
Delete a single entity in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to delete. |

## GET /Entities/{id}/Fields
Retrieves the fields for a single entity in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to retrieve. |
| Output Data | **EntityFieldListItemResource** | A list of **EntityFieldResource** objects. |

## GET /Entities/{id}/Fields/{fid}
Retrieves a single field for a single entity in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to retrieve. |
| | {fid} | The field id to retrieve for the entity. |
| Output Data | **EntityFieldResource** | |

## PUT /Entities/{id}/Fields/{fid}
Updates a single field for a single entity in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to retrieve. |
| | {fid} | The field id to retrieve for the entity. |
| Input Data | **EntityFieldResource** | |

## PATCH /Entities/{id}/Fields/{fid}

Updates specified properties for a single field for a single entity in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The entity id to retrieve. |
| | {fid} | The field id to retrieve for the entity. |
| Input Data | **EntityFieldResource** | |

## Functions

This resource provides access to all the functions within the application.

### GET /Functions
Retrieves the functions in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| | *filter* | Optionally allows the user to filter by types of functions. Accepted values:<br>• custom: only custom functions |
| Output Data | **FunctionListItemResource** | A list of **FunctionResource** objects. |

### POST /Functions
Creates a new function.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **FunctionResource** | |

### GET /Functions/{id}
Retrieves a single function in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Path | {id} | The function id to retrieve. |
| Output Data | **FunctionResource** | |

## PUT /Functions/{id}

Updates a single function in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The function id to update. |
| Input Data | **FunctionResource** | |

## PATCH /Functions/{id}

Updates specified properties for a single function in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The function id to update. |
| Input Data | **FunctionResource** | |

## DELETE /Functions/{id}

Delete a single function in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The function id to delete. |

## ServerEvents

### GET /ServerEvents

Retrieves the server events in a session.

| Info | | Description |
|---|---|---|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| | *filter* | Optionally allows the user to filter by types of server events. Accepted values: |
| | | • custom: only custom server events |

| Output Data | **ServerEventListItemResource** | A list of **ServerEventResource** objects. |

## POST /ServerEvents
Creates a new server event.

| Info | | Description |
|------|------|------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Input Data | **ServerEventResource** | |

## GET /ServerEvents/{id}
Retrieves a single server event in a session.

| Info | | Description |
|------|------|------|
| Authorization | Optional | Unauthorized access may only return a subset of data. |
| Parameters | *sid* | Required session id. |
| Path | {id} | The server event id to retrieve. |
| Output Data | **ServerEventResource** | |

## PUT /ServerEvents/{id}
Updates a single server event in a session in its entirety. If a property is not specified, it will revert to its original and default value.

| Info | | Description |
|------|------|------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The server event id to update. |
| Input Data | **ServerEventResource** | |

## PATCH /ServerEvents/{id}
Updates specified properties for a single server event in a session. If a property is not specified, it will not be changed.

| Info | | Description |
|------|------|------|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The server event id to update. |
| Input Data | **ServerEventResource** | |

## DELETE /ServerEvents/{id}
Delete a single server event in a session.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The server event id to delete. |

## Folders

### POST /Folders/{id}
Creates a new folder.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The folder id to create. |
| Output Data | **FolderResource** | |

### POST /Folders/{id}
Renames an existing folder.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The folder id to rename. |
| Input Data | **FolderRenameResource** | |
| Output Data | **FolderResource** | |

### GET /Folders/{id}
Checks if a folder exists or not.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Path | {id} | The folder id to check. |
| Output Data | **FolderResource** | |

### DELETE /Folders/{id}
Deletes a folder.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |

| | | |
|---|---|---|
| Path | {id} | The folder id to delete. |
| Input Data | **FolderResource** | |

## Reports

### GET /Reports/List
Retrieve the list of folders and reports available to the current role.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| Output Data | **ReportFoldersResource** | |

### POST /Reports/Execute/{type}
Execute the selected report.

| Info | | Description |
|---|---|---|
| Authorization | Required | |
| Parameters | *sid* | Required session id. |
| | *ReportPath* | Required report path. |
| | *Sorts* | Optional **SortsResource** array. |
| | *Filters* | Optional **FiltersResource** array. |
| Path | {type} | The execution file type, one of: |
| | | • **html** |
| | | • **csv** |

## Data Definitions

A:     Authorized Only
W:     Writable Property (Y: Yes, N: No, C: Create Only)
R:     Required Property (Y: Yes, N: No, C: Create Only)

### Sessions

### SessionResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | N | - | - | The SID for this session |
| AppUrl | String | Y | N | - | - | The URL to access the Exago Application as session. |
| Page | String | Y | Y | N | ExagoHome | The page to access within the Exago Application. |

| | | | | | | |
|---|---|---|---|---|---|---|
| Action | String | Y | Y | N | Default | The action to take when initially accessing the application, one of: |

- **Default** – The default action
- **Home** – Open the home page
- **NewReport** – Open the New Report Wizard for a Standard Report
- **NewCrossTabReport** – Open the New Report Wizard for a CrossTab Report
- **NewDashboardReport** – Open the New Report Wizard for a Dashboard Report

| | | | | | | |
|---|---|---|---|---|---|---|
| ShowTabs | Boolean | Y | Y | N | true | Whether tabs should be shown. |
| ShowErrorDetail | Boolean | Y | Y | N | false | Whether error details should be shown. |

**DataSources**

**DataSourceListItemResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | N | - | - | The ID for this data source |
| Name | String | Y | N | - | - | The name for this data source |

**DataSourceResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | N | - | - | The ID of the data source |
| Name | String | Y | Y | N | | The name of the data source |
| DbType | String | Y | Y | N | Mssql | The type of the data source, one of: |

- **MsSql**
- **MySql**
- **Postgres**
- **Oracle**
- **DB2**
- **Informix**
- **Assembly**
- **WebService**

| | | | | | | |
|---|---|---|---|---|---|---|
| Connection | String | Y | Y | N | | The data source connection string |
| Schema | String | Y | Y | N | | The default data schema |

**Joins**

**JoinListItemResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | N | - | - | The ID for this join |

**JoinResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | N | - | - | The ID of the join |
| EntityFrom | String | Y | C | C | | The id of the from entity |
| EntityTo | String | Y | C | C | | The id of the to entity |
| JoinType | String | Y | Y | N | Inner | The type of join, one of:<br>• **Inner**<br>• **LeftOuter**<br>• **RightOuter**<br>• **FullOuter** |
| RelationshipType | String | Y | Y | N | OneToOne | The type of relationship, one of:<br>• **OneToOne**<br>• **OneToMany** |
| Weight | Integer | Y | Y | N | 0 | The weight of the join |
| JoinColumns | JoinColumnsResource[] | Y | Y | Y | | The columns to join |

## JoinColumnsResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| ColumnFrom | String | Y | Y | Y | | The id of the from column |
| ColumnTo | String | Y | Y | Y | | The id of the to column |

## Roles

## RoleListItemResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | N | - | - | The ID for this role. |

## RoleResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | C | C | - | The ID for this role. |
| IsActive | Boolean | Y | Y | N | false | Whether this role is active, NOTE. Only one role may be active at a time. |

## RoleSettingsResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| ReportPath | String | Y | Y | N | - | The report path |
| ReadFilterValues | Boolean | Y | Y | N | - | When true, filter values are read from the database |
| DbTimeout | Number | Y | Y | N | - | Maximum number of seconds a single query is allowed to run |
| ScheduleManagerViewLevel | String | Y | Y | N | - | One of:<br>• **User**<br>• **Company**<br>• **All** |

| | | | | | | |
|---|---|---|---|---|---|---|
| LanguageFile | String | Y | Y | N | - | The language file string |
| ServerTimeZoneOffset | Number | Y | Y | N | - | Used to convert server time to client time |
| DateFormat | String | N | Y | N | - | The date format string for this role |
| TimeFormat | String | N | Y | N | - | The time format string for this role |
| DateTimeFormat | String | N | Y | N | - | The datetime format string for this role |
| SeparatorSymbol | String | N | Y | N | - | The character to use as a number separator symbol |
| CurrencySymbol | String | N | Y | N | - | The character to use to represent currencty |
| Decimal Symbol | String | N | Y | N | - | The character to use as a decimal symbol |
| ShowGrid | Boolean | N | Y | N | - | When true, the default Report Viewer option to show grid will be true. |
| ShowCrossTabReports | Boolean | N | Y | N | - | When true, the crosstab designer is accessible |
| ShowExpressReports | Boolean | N | Y | N | - | When true, the express report designer is accessible |
| ShowExpressReportsGrouping | Boolean | N | Y | N | - | When true, express report grouping is accessible |
| ShowExpressReportsFormulas | Boolean | N | Y | N | - | When true, express report formulas are accessible |
| ShowExpressReportsStyling | Boolean | N | Y | N | - | When true, express report styling is accessible |
| ShowExpressReportsThemes | Boolean | N | Y | N | - | When true, express report themes are accessible |
| ShowStandardReports | Boolean | N | Y | N | - | When true, the standard report designer is accessible |
| ShowScheduleReports | Boolean | N | Y | N | - | When true, scheduling reports is accessible |
| ShowScheduleReportsManager | Boolean | N | Y | N | - | When true, the schedule report manager is accessible |
| ShowScheduleReportsEmail | Boolean | N | Y | N | - | When true, emailing reports is accessible |

**RoleEntityResourceCollection**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| IncludeAll | Boolean | Y | Y | N | false | When true, all entities except specified in the Entities property are included. When false, only the entities specified in the Entities property are included. |
| Entities | RoleEntityResource[] | Y | Y | N | | The list of entities that are included or excluded (depends on IncludeAll) |

**RoleEntityResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|

| | | A | W | R | | |
|---|---|---|---|---|---|---|
| Id | String | Y | Y | N | | The id of the entity this role is controlling. |

## RoleFolderResourceCollection

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| IncludeAll | Boolean | Y | Y | N | false | When true, all folders except specified in the Folders property are included. When false, only the folders specified in the Folders property are included. |
| ReadOnly | Boolean | Y | Y | N | false | When true, all folders except specified as readonly in the Folders property are read only. When false, only the folders specified in the Folders property as readonly are read only. |
| AllowManagement | Boolean | Y | Y | N | false | When true, folder management is allowed |
| Folders | RoleFolderResource[] | Y | Y | N | | The list of folders that are included or excluded (depends on IncludeAll) |

## RoleFolderResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | Y | N | | The id of the folder |
| ReadOnly | Boolean | Y | Y | N | false | When true, this folder is read only |
| Propogate | Boolean | Y | Y | N | false | When true, the readonly property is propogated to the children of this folder. |

## RoleDataObjectRowResourceCollection

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| DataObjectRows | RoleDataObjectRowResource[] | Y | Y | N | | The data object rows included. |

## RoleDataObjectRowResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | Y | Y | N | | The id of the data object |
| Filter | String | Y | Y | N | | The filter string |

## Settings

## SettingsResource

| Name | | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| ReportPath | String | Y | Y | N | | The report path |
| ReadFilterValues | Boolean | Y | Y | N | | When true, filter values are read from the database |
| DbTimeout | Number | Y | Y | N | | Maximum number of seconds a single query is allowed to run |
| ScheduleManagerViewLevel | String | Y | Y | N | All | One of:<br>• **User**<br>• **Company**<br>• **All** |
| LanguageFile | String | Y | Y | N | | The language file string |
| ServerTimeZoneOffset | Number | Y | Y | N | | Used to convert server time to client time |
| DateFormat | String | N | Y | N | | The date format string |
| TimeFormat | String | N | Y | N | | The time format string |
| DateTimeFormat | String | N | Y | N | | The datetime format string |
| SeparatorSymbol | String | N | Y | N | , | The character to use as a number separator symbol |
| CurrencySymbol | String | N | Y | N | $ | The character to use to represent currencty |
| DecimalSymbol | String | N | Y | N | . | The character to use as a decimal symbol |
| ShowGrid | Boolean | N | Y | N | | When true, the default Report Viewer option to show grid will be true. |
| ShowCrossTabReports | Boolean | N | Y | N | | When true, the crosstab designer is accessible |
| ShowExpressReports | Boolean | N | Y | N | | When true, the express report designer is accessible |
| ShowExpressReportsGrouping | Boolean | N | Y | N | | When true, express report grouping is accessible |
| ShowExpressReportsFormulas | Boolean | N | Y | N | | When true, express report formulas are accessible |
| ShowExpressReportsStyling | Boolean | N | Y | N | | When true, express report styling is accessible |
| ShowExpressReportsThemes | Boolean | N | Y | N | | When true, express report themes are accessible |
| ShowStandardReports | Boolean | N | Y | N | | When true, the standard report designer is accessible |
| ShowScheduleReports | Boolean | N | Y | N | | When true, scheduling reports is accessible |
| ShowScheduleReportsManager | Boolean | N | Y | N | | When true, the schedule report manager is accessible |
| ShowScheduleReportsEmail | Boolean | N | Y | N | | When true, emailing reports is accessible |

**Parameters**

**ParameterListItemResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | N | - | - | The ID for this parameter |

## ParameterResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| IsHidden | Boolean | Y | Y | N | true | Whether this parameter is hidden to the user |
| Id | String | N | C | C | - | The ID for this parameter |
| DataType | String | N | Y | N | String | The data type of the parameter, one of:<br>• **String**<br>• **Date**<br>• **Integer**<br>• **Decimal** |
| Value | String | N | Y | N | "" | The value of the parameter |
| PromptText | String | N | Y | N | "" | The text to prompt the user for the value. |

## Entities

## EntityListItemResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | N | - | - | The ID for this entity |
| Name | String | N | N | - | - | The name of this entity |

## EntityResource

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | C | C | - | The ID for this entity |
| Name | String | N | Y | Y | - | The name of this entity |
| DataSourceId | Integer | Y | Y | Y | - | The DataSource Resource ID this entity gets its data from |
| DataType | String | Y | Y | N | Table | The type of data this entity is, one of:<br>• **Assembly**<br>• **File**<br>• **Function**<br>• **Procedure**<br>• **SqlStmt**<br>• **Table**<br>• **View**<br>• **WebSvc** |
| DataName | String | Y | Y | Y | - | The name of the data object for this entity |
| CategoryName | String | Y | Y | N | - | The category group of the entity. |
| Schema | String | Y | Y | N | - | The database schema for the entity. |

| | | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| KeyColumns | String[] | Y | Y | N | - | List of key columns for the entity |
| TenantColumns | TenantResource[] | Y | Y | N | - | List of tenant columns for the entity. |

**TenantResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Column | String | Y | Y | N | - | Tenant column name |
| Parameter | String | Y | Y | Y | - | Parameter associated with this tenant column. |

**EntityFieldListItemResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | N | - | - | The ID for this entity field |
| Name | String | N | N | - | - | The name of this entity field |

**EntityFieldResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | N | - | - | The ID for this entity field |
| Name | String | N | Y | Y | - | The name of this entity field |
| Type | String | N | Y | N | <actual data type> | The type of data this entity field is, one of:<br>• **String**<br>• **Date**<br>• **DateTime**<br>• **Time**<br>• **Int**<br>• **Decimal**<br>• **Float**<br>• **Bit**<br>• **Guid**<br>• **Image** |
| IsFilterable | Boolean | N | Y | N | true | When true, this field is filterable |
| IsVisible | Boolean | Y | Y | N | true | When true, this field is visible to users |

**Functions**

**FunctionListItemResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | N | - | - | The ID for this function |

**FunctionResource**

| Name | Type | A | W | R | Default | Description |
|---|---|---|---|---|---|---|
| Id | String | N | C | C | - | The ID for this function |
| Description | String | N | Y | N | | The description for the function |

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| | | | | | | (only writeable if custom function) |
| MinArgs | Integer | N | Y | N | 1 | The minimum number of arguments for this function (only writeable if custom function) |
| MaxArgs | Integer | N | Y | N | 1 | The maximum number of arguments for this function (only writeable if custom function) |
| IsCustom | Boolean | Y | N | - | - | Whether this function is a custom function or a built-in function |
| Language | String | Y | Y | N | CSharp | The language the custom function is written in, one of:<br>• **CSharp**<br>• **JavaScript**<br>• **VisualBasic** |
| Namespaces | Array of String | Y | Y | N | | The namespaces to include for this function |
| References | Array of String | Y | Y | N | | The references to include for this function |
| ProgramCode | String | Y | Y | N | | The program code for this function |

## Server Events

### ServerEventListItemResource

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| Id | String | N | N | - | - | The ID for this server event |
| Name | String | N | N | - | - | The name for this server event |

### ServerEventResource

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| Id | String | N | C | C | - | The ID for this server event |
| Name | String | N | Y | N | | The name for this server event |
| Code | String | Y | Y | N | | The program code for this function |
| Language | String | Y | Y | N | CSharp | The language the custom function is written in, one of:<br>• **CSharp**<br>• **JavaScript**<br>• **VisualBasic** |
| Namespaces | Array of String | Y | Y | N | | The namespaces to include for this function |
| References | Array of String | Y | Y | N | | The references to include for this function |

## Folders

### FolderResource

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| Id | String | Y | Y | Y | - | The ID for this folder |
| Name | String | Y | Y | Y | | The name for this folder |
| Exists | String | Y | Y | Y | | The status for this folder |

### FolderRenameResource

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| BasePath | String | Y | Y | Y | - | The path for this folder |
| OldName | String | Y | Y | Y | | The old name for this folder |
| NewName | String | Y | Y | Y | | The new name for this folder |

### Reports

### ReportFoldersResource

| Name | Type | A | W | R | Description |
|------|------|---|---|---|-------------|
| name | String | Y | N | Y | The name of the folder or report |
| leaf_flag | Boolean | Y | N | Y | True: Report; False: Folder |
| readonly_flag | Boolean | Y | N | Y | Read-only status for current role |
| type | String | Y | N | N | Type of report, one of:<br>• **standard**<br>• **express**<br>• **chained**<br>• **dashboard**<br>• **crosstab**<br>Null if it is a folder. |
| user | String | Y | N | N | The current role. |
| entity | ReportFoldersResource[] | Y | N | N | List of reports and sub-folders. |

### SortsResource

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| EntityName | String | Y | Y | Y | | Data object to sort on |
| ColumnName | String | Y | Y | Y | | Column to sort on |
| AscendingFlag | Boolean | Y | Y | N | True | True: Ascending; False: Descending |

### FiltersResource

| Name | Type | A | W | R | Default | Description |
|------|------|---|---|---|---------|-------------|
| EntityName | String | Y | Y | Y | | Data object to filter on |
| ColumnName | String | Y | Y | Y | | Column to filter on |
| DataType | Data Type string | Y | Y | N | string | The data type, one of:<br>• **string**<br>• **date** |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  |  |  |  |  |  | • **integer**<br>• **bit**<br>• **numeric**<br>• **float**<br>• **decimal**<br>• **guid**<br>• **datetime** |
| OperatorType | Operator Type enum | Y | Y | N | 0 | An enum representing operator type, one of:<br>• 0: EqualTo<br>• 1: LessThan<br>• 2: LessThanOrEqualTo<br>• 3: GreaterThan<br>• 4: GreaterThanOrEqualTo<br>• 5: NotEqualTo<br>• 6: StartsWith<br>• 7: NotStartsWith<br>• 8: EndsWith<br>• 9: NotEndsWith<br>• 10: Contains<br>• 11: NotContains<br>• 12: Between<br>• 13: NotBetween<br>• 14: OneOf<br>• 15: NotOneOf |
| Values | String[] | Y | Y | Y |  | Array of values to filter with |

## Return Codes

For any request the below return codes are defined. Each resource may define more return codes as well as better define the return codes provided here.

| Status Code | Description |
|---|---|
| 200 | The request was completed successfully. The document in the entire-body, if any, is a representation of some resource. |
| 201 | The request was completed successfully. A new resource has been created at the URL specified in the Location header of the response. |
| 204 | The request was completed successfully. There is no content in the body. |
| 400 | The request was bad on the client side. The document in the entire-body, if any, is error data describing the problem. |
| 401 | The request wasn't authorized to access the resource. The document in the entire body, if any, is error data describing the problem. |
| 404 | The requested resource was not found. The document in the entire body, if any, is error data describing the problem. |
| 409 | The request caused a conflict amongst two resources. The document in the entire body, if any, is error data describing the problem. |
| 500 | There was a problem on the server side. The document in the entire-body, if any, is error data describing the problem. |

## Error Data

All codes 4xx and 5xx will have a log entry in the WebReports log. When possible, the document body returned with the status takes the following representation:

| Name | Type | Description |
|---|---|---|
| Code | String | A string token indicating the reason for the error, for programmatic consumption. |
| Message | String | A more verbose text describing the error, for developer consumption. |
| UserMessageId | String | The language id to display to the user as the error, for user consumption. |
| StackTrace | String | The server-side stack trace if one exists |

### Error Codes

| Code | Description |
|---|---|
| Unknown | Unknown error |
| UnAuthorized | UnAuthorized access was attempted |
| ResourceNotFound | Resrouce was not found at the specified URL |
| ResourceConflict | Resource conflicts with an already existing resource |
| MissingResource | Request must contain a resource |
| MissingResourceId | Request must contain a resource with a user supplied ID to identify location of resource |
| InvalidResourceId | Request must contain a resource with a valid ID to identify location of resource |
| InvalidState | Request contains a resource in an invalid state |

# Troubleshooting

The following chapter details techniques for troubleshooting issues that may arise when using Exago.

## See Full Error Details

When an error occurs in Exago, a generic error message is displayed.

An error has occurred that is preventing completion of this request. Please contact your adminstrator.

HTTP Status code: 200.

✓ OK
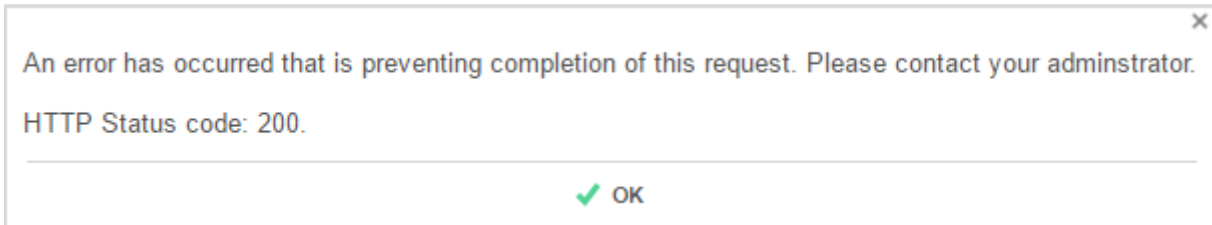
This generic message is meant to prevent end users from seeing the full stack trace of the error.

There are two ways to see detailed error messages.

1. If you are accessing Exago directly in a browser:

    a. Append '**?showerrordetail=true**' to the url. **Ex.**
       …/Exagohome.aspx?showerrordetail=true

    b. Refresh the page and recreate the error.

2. If you are accessing Exago through the Api:

    a. Using the **.Net Api** call the method GetUrlParamString and set showErrorDetail to True.
       -OR-
       Using the **Web Service Api** call the method GetUrlParamString2 and set showErrorDetail to True.

    b. Enter Exago through the Api and recreate the error.

    NOTE. The status code on the generic error message corresponds to standard html error codes. For example if the status code is 408 it means there was a request timeout. For status code 200 the html completed successfully and the error lies elsewhere.

If you would like more details after seeing the full error message please see the section **Read the Log File**.

## Read the Log File

Exago keeps a text log of when certain tasks are performed. For example each time a page or report is loaded, each time an error occurs or when various phases of execution happen.

To access the Log file:

1. Set "Write Log File" to True in **Other Settings** of the Administration Console.

**Exago Inc.**

2. Recreate the error you are investigating.

3. Navigate to the folder specified in the Temp Path of **Main Settings**. If this is blank go to <webapp_dir>/Temp.

4. Open the file **WebReportsLog.txt**. Scroll to the bottom of log for the most recent activity.

   NOTE. Occasionally IIS may lock this file and prevent the log from being written. To correct this reset, IIS, delete the file WebReportsLog.txt and repeat steps 2-4.

NOTE. If 'Enable Remote Report Execution' is set to True in the **Scheduler Settings** the report execution will be recorded in the **Scheduler Log**.
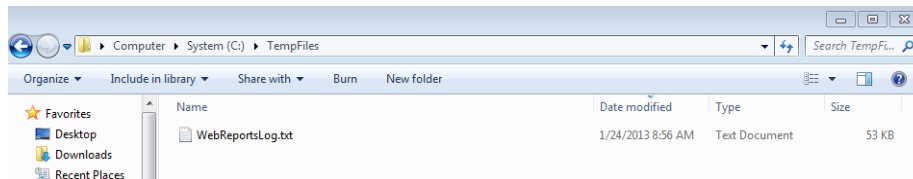


## Scheduler Log

Similar to the main application the Exago Scheduler Service maintains a log file. Considering the Scheduler can reside on a different machine than the main application the log file is written where the Scheduler is installed.

To access the Scheduler Log file:

- Set <logging> to True in the file <scheduler_dir>\Config\ ExagoScheduler.xml

- Rerun the scheduled report you are investigating.

- Open the file <scheduler_dir>\ExagoScheduler.log. Scroll to the bottom of the log for the most recent activity.

## Web Service Log

Similar to the main application Exago Web Service maintains a log file. Considering the Web Service can reside on a different machine than the main application the log file is written where the Web Service is installed.

To access the Api Log file:

- Set <writelog> to True in the file <websvc_dir>\Config\ WebReportsApi.xml

- Rerun the project that makes the Api calls you are investigating.

- Open the file <websvc_dir>\Config\ WebReportsApiLog.txt. Scroll to the bottom of the log for the most recent activity.

# Check the Version, Connections & Permissions

This section will detail a few useful things to check when troubleshooting an issue within Exago.

## Verifying Folder Permissions

A common issue when installing or updating Exago is to not set read/write permissions on various folders the application uses. For these folders the user accessing them via the IIS app pool must have read write permissions. For the default app pool this is the IIS user ('iis_iusrs' for Windows 7, Vista & 2008, 'aspnet' for Windows XP or Server 2003). The folders that require read/write permissions are listed below.

- **<webapp_dir>/Config** – this folder contains the configuration settings loaded and modified by the Admin Console.

- **<webapp_dir>/Temp** – this folder is used to store some temporary files.

- The folder specified in the **Report Path** of the **Main Settings**.

- The folder specified in the **Temp Path** of the **Main Settings**.
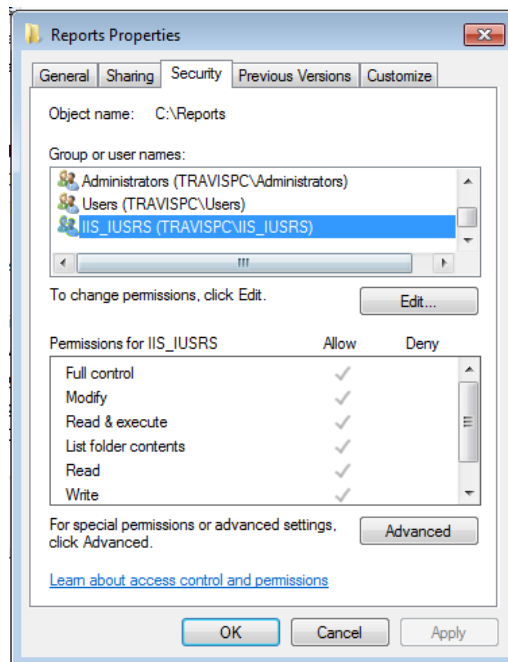


## Verifying Administration Settings

Some settings in the **Administration Console** are used to connect to other programs such as databases, Web Services, .Net Assemblies or the External Interface module. Each of these items will have a green check button (✔) to verify they are connecting correctly.

The following settings can be checked:

- The **Data Source**(s) being utilized by the report you are investigating.

- The 'Schedule Remoting Host' and 'Remote Execution Remoting Host' in **Scheduler Settings**.

- The 'External Interface' in **Other Settings**.
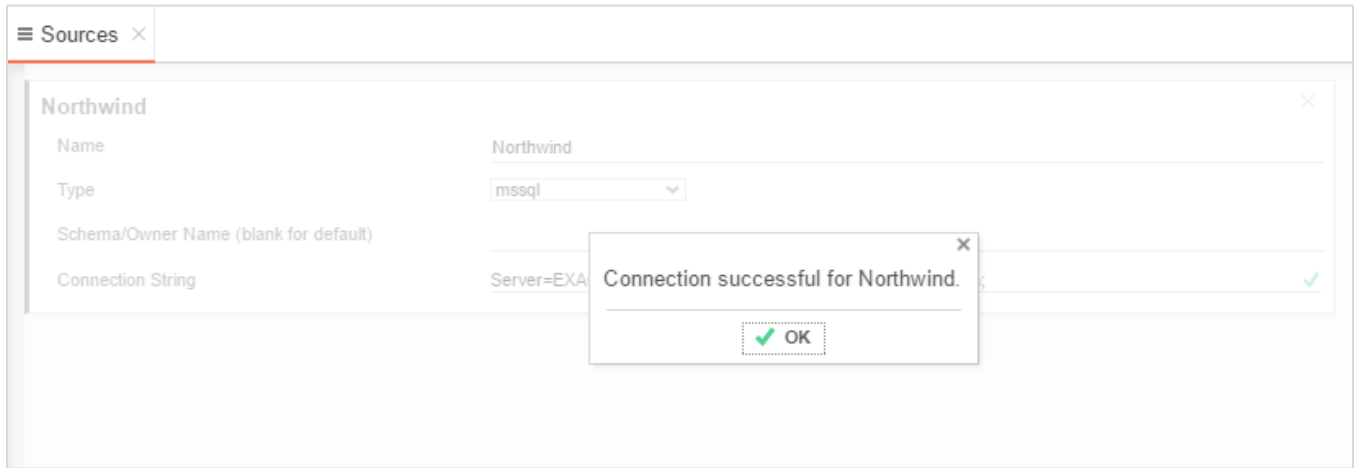
## Verifying Versions

When updating Exago occasionally issues arise because the Scheduler Service or Web Service Api have not also been updated. For the Scheduler and Web Service Api to function properly they must be running the same version and build as the Exago application.

> NOTE. When using the .Net Api you will need to copy the dlls from the updated Exago application to the bin directory of your host application.
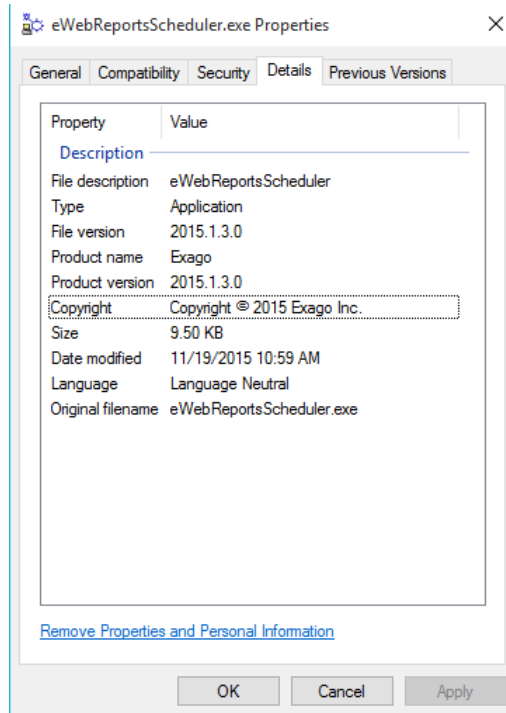
To check the version of **Exago**:

- Navigate to the home page (default exaghome.aspx).

- Press Ctrl + Shift + V
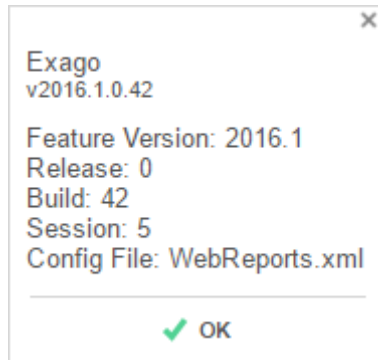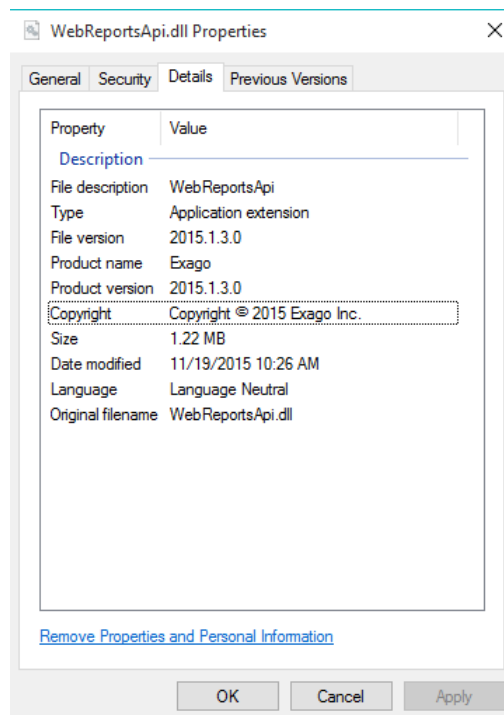


To check the version of **Scheduling Service**:

- Locate the file <scheduler_dir>/eWebReportsScheduler.exe.

- Right click on this file and select 'Properties'

- Navigate to the 'Details' tab.

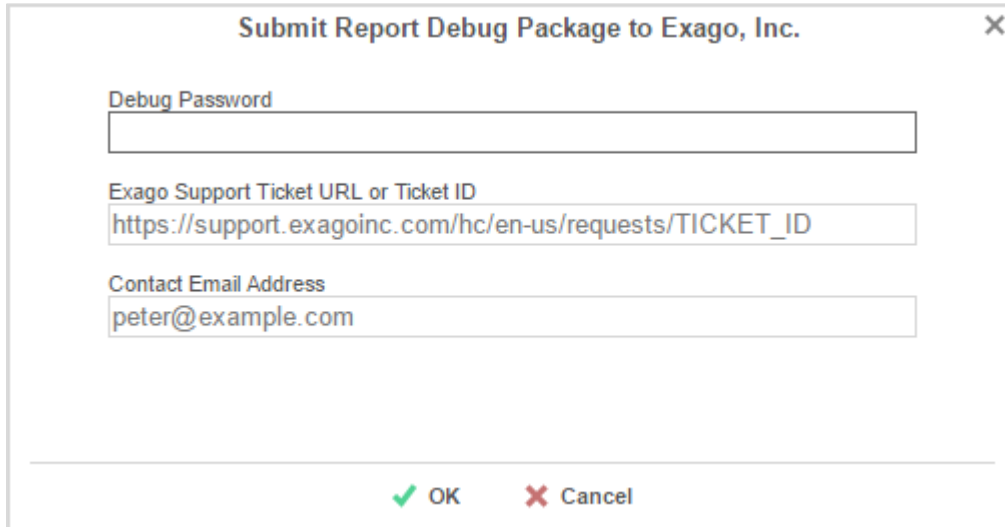To check the version of **Web Service Api**:

- Locate the file <websvc_dir>/bin/ WebReportsApi.dll.

- Right click on this file and select 'Properties'

- Navigate to the 'Details' tab.

# Submitting a Debug Package

If the "Debug Password" is set in **Other Settings** of the Administration Console, a client will have the ability to submit Debug Packages automatically to Exago, Inc. via the internet. The client will need to select the report that they are having a problem with and press **Ctrl**+**Shift**+**X**. This keystroke will bring up the Debug Package submission window.

> NOTE. The Debug Package consists of the same files that are created via "Enable Debugging," which is also located in **Other Settings** of the Administration Console. These files are encrypted, and then sent to a Web Service that resides at the Exago Support site.



To send a Debug Package to Exago Support:

1. From the Main Menu, select the problematic report.

2. Press Ctrl+Shift+X.

3. In the Submit Debug Package window, enter the debug extraction password (this is set in **Other Settings**), preexisting Ticket URL or ID, and contact email address.

4. Click the 'Ok' button.

5. Fill in any information for Parameters or Filters if they are set for the report.

6. A success/failure message will display when the process finishes.

If submitting a debug package fails see **Manually Creating a Debug Package**.

## Manually Creating a Debug Package

If submitting a debug package fails then you can set 'Enable Debugging' to True in the **Other Settings** of Administration Console to manually create the files needed for debugging. These files can be zipped and emailed to **support@exagoinc.com**.

NOTE. Before creating a Debug Package verify that 'Enable Remote Report Execution' in **Scheduler Settings** is set to False.

To manually create a Debug Package:

1. Create the folder Debug where Exago is installed. Make sure this folder has the same read/write permissions as the Report and Temp Folders.

2. Set 'Enable Debugging' in **Other Settings** to True.

3. Execute the problematic report. A copy of the report, the configuration settings and a data set will be created in '.\Debug'.

4.  Zip these three files together and email them to **support@exagoinc.com**.

Exago, Inc.
Two Enterprise Drive
Shelton, CT 06484 USA
203.225.0876
**http://www.exagoinc.com**