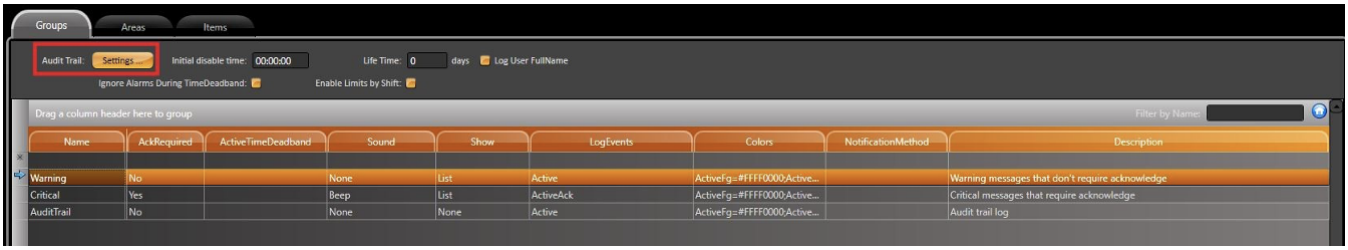
Audit Trail

An audit trail (also called audit log) is a security-relevant chronological record, set of records, and/or destination and source of records that provide documentary evidence of the sequence of activities that have affected at any time a specific operation, procedure, or event.

Settings

To use the Audit Trail function, you must enable it. Go to **Edit** > **Alarms** > **Groups**, and click on the Settings button.



A popup display will open with many checkboxes. Besides the Enable option, you can choose which actions will be stored in the Audit Trail database. The options are as follows:

User Logon/Logoff : Stores informational data on user login/logout.

Active Time	UserName	Message
2/4/2020 2:13:07 PM	User	Logged user: User; 127.0.0.1\RichClient
2/4/2020 2:13:01 PM	Guest	Logged user: Guest; 127.0.0.1\RichClient
2/4/2020 2:12:57 PM	Administrator	Logged user: Administrator; 127.0.0.1\RichClient
2/4/2020 2:12:38 PM	Guest	Logged user: Guest; 127.0.0.1\RichClient

Open/Close Displays: Stores informational data when displays are open or closed.

Active Time	Group	Message
2/4/2020 2:19:27 PM	AuditTrail	Display was closed: About; RichClient
2/4/2020 2:19:25 PM	AuditTrail	Display was closed: SelectPage; RichClient
2/4/2020 2:19:25 PM	AuditTrail	Display was opened: About; RichClient
2/4/2020 2:19:25 PM	AuditTrail	Display was closed: MainPage; RichClient
2/4/2020 2:19:23 PM	AuditTrail	Display was opened: SelectPage; RichClient
2/4/2020 2:16:49 PM	AuditTrail	Display was opened: Header; RichClient
2/4/2020 2:16:49 PM	AuditTrail	Display was opened: MainPage; RichClient

Remote Connections: Stores information on remote client connections (Smart/Rich Clients).

Active Time	Message
2/4/2020 3:26:26 PM	Connection accepted: 192.168.0.16\RichClient; Guest
2/4/2020 3:25:48 PM	Connection closed: 192.168.0.16\SmartClient
2/4/2020 3:25:36 PM	Connection accepted: 192.168.0.16\SmartClient; Guest

Custom Messages: Stores added custom messages.

```
public void MouseLeftButtonDown1(object sender, System.Windows.Input.InputEventArgs e)
{
    @Alarm.AuditTrail.AddCustomMessage("Custom Message added in " + @Server.Now + " by User: " + @Client.UserName);
}
```

Active Time	Message
2/4/2020 2:24:38 PM	Custom Message added in 2/4/2020 2:24:38 PM -03:00 by User: Administrator
2/4/2020 2:23:48 PM	Custom Message added in 2/4/2020 2:23:48 PM -03:00 by User: Guest

Tag Changes: Stores informational data of every tag change.

Active Time	TagName	Message
2/4/2020 3:13:19 PM	Tag.Pressure	Tag changed
2/4/2020 3:13:17 PM	Tag.OnOff	Tag changed
2/4/2020 3:13:14 PM	Tag.OpenClose	Tag changed
2/4/2020 3:13:12 PM	Tag.SelectedIndex	Tag changed
2/4/2020 3:13:10 PM	Tag.Temperature	Tag changed
2/4/2020 3:13:08 PM	Tag.Temperature	Tag changed

Datasets (Insert/Updates or All Commands): Stores information on datasets.

Active Time	Message
2/4/2020 3:06:23 PM	Dataset executed Execute command: Dataset.Query.AlarmSelect; Select * from Alarms; Success; RichClient
2/4/2020 3:06:13 PM	Dataset executed Execute command: Dataset.Query.HistorianSelect; Select * from Table1; Success; RichClient

Operator Actions: Stores information on operator actions.

Active Time	Message
2/4/2020 2:59:52 PM	Report executed Save command: Report.TemperatureReport; RichClient; C:\TemperatureReport.pdf
2/4/2020 2:59:30 PM	Report executed Save command: Report.FurnaceInfoReport; RichClient; C:\FurnaceInfoReport.pdf

Save Reports: Stores information when the save command is executed.

Active Time	Message
2/4/2020 2:59:52 PM	Report executed Save command: Report.TemperatureReport; RichClient; C:\TemperatureReport.pdf
2/4/2020 2:59:30 PM	Report executed Save command: Report.FurnaceInfoReport; RichClient; C:\FurnaceInfoReport.pdf

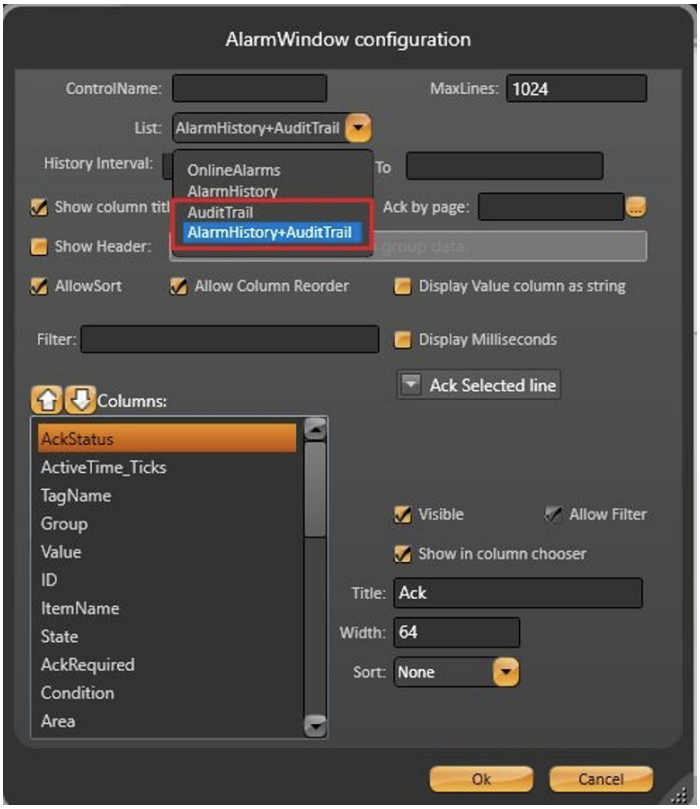
System Warnings: Stores information related to the system.

Active Time	Group	Message
2/4/2020 2:39:19 PM	AuditTrail	DataSet was started
2/4/2020 2:39:15 PM	AuditTrail	Server was started
2/4/2020 2:39:15 PM	AuditTrail	Historian was started
2/4/2020 2:39:15 PM	AuditTrail	Alarm was started

It is possible to enable any of these options during runtime by using the Alarm namespace properties. The syntax is:

```
@Alarm.AuditTrail.<Audit Trail Option>
```

To visualize the stored Audit Trail data, you can add an AlarmWindow element to your display and select the AuditTrail option in the ComboBox list.



Custom Messages

One of the most important features of the Audit trail is the ability to have customizable messages added to a historian database. Custom messages are added in runtime using the method below:

```
@Alarm.AuditTrail.AddCustomMessage(string message, string areaName, string objectName, string value, string
itemName, string auxValue, string comment)
```

where:

- **message**: The custom message to be added to the Audit
- **areaName**: The area related to this custom message
- **objectName**: The object related to this custom message
- **value**: The object value related to this custom message
- **itemName**: The item name
- **auxValue**: The auxiliary value
- **comments**: The comments

The messages can either be text or a concatenation between text and real time info from the project. For messages that are only text, you will need only the message parameter, e.g.:

```
@Alarm.AuditTrail.AddCustomMessage("The day is sunny")
```

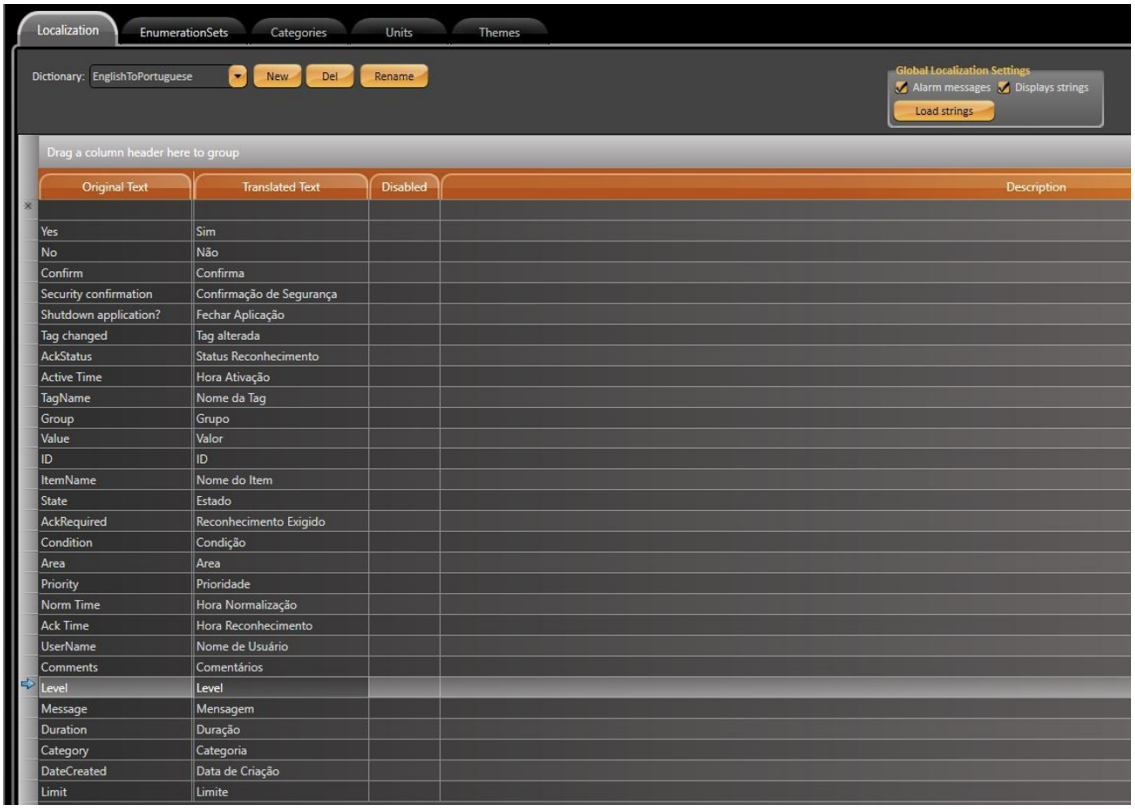
An example on the usage of text and project info is:

```
@Alarm.AuditTrail.AddCustomMessage("User:  "  +  @Client.UserName  +  "  logged");
```

Add Translation to Custom Messages

To translate text into different languages, you first need to create a set of words in a custom dictionary. Go to **Run** > **Dictionaries** > **Localization**. On the top of the display, you will find some buttons:

- **New**: Create a new dictionary
- **Del**: Delete an existing dictionary
- **Rename**: Renames an existing dictionary
- **Load strings**: Load project strings that have the Global Localization setting



To apply this feature to the custom messages in the Audit Trail, you must follow a certain syntax.

- If the message is text only, the default syntax is:

```
@Alarm.AuditTrail.AddCustomMessage("tag changed value, AckRequired");
```

- If the message is text and project info, you must add the curly brackets char "***{ }***" before and after the project info. The message string should look like this:

```
string message = "User: {" + @Client.UserName + "} logged"
```

> **Note**
>
> The alarm database will contain chars " ***{*** " and " ***}*** " in the Message column. The dictionary must also contain the brackets characters.

You must add another string element to the itemName input parameter, as seen below:

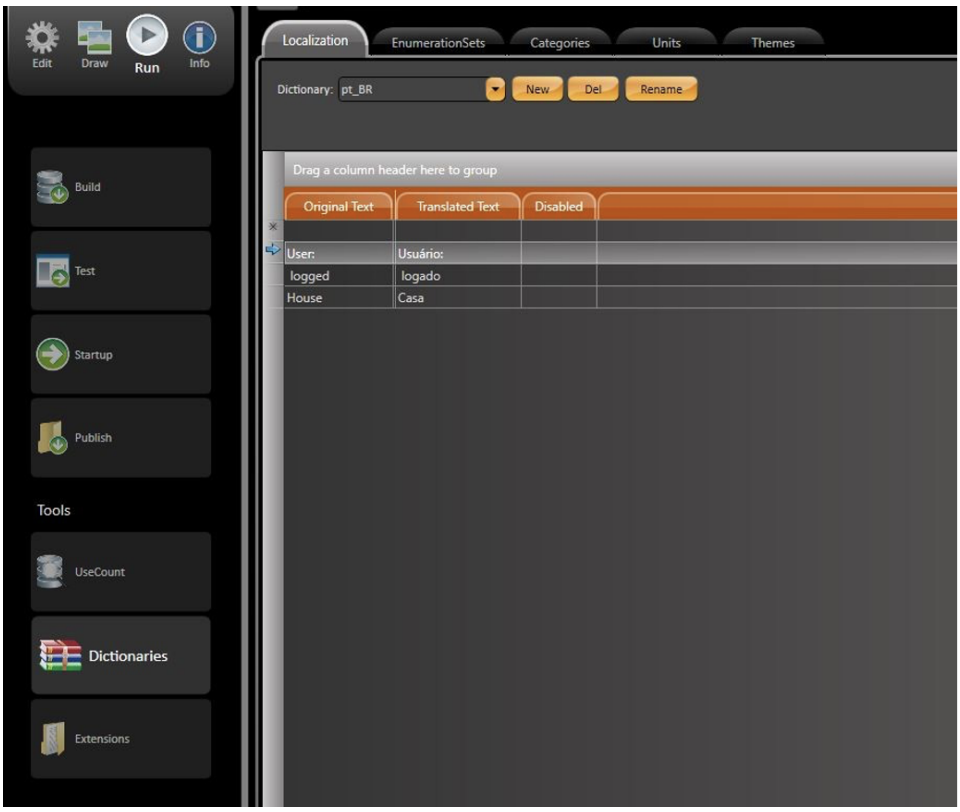
```
string itemName = "{object}"
```

A final AddCustomMessage with localization capabilities should look like this:

```
@Alarm.AuditTrail.AddCustomMessage("User:  {"  +  @Client.UserName  +  "}  logged", null, null, null,
"{object}", null, null);
```

Translating Tags and Tables

For reports with different translation options, the first requirement is the creation of Dictionaries (in **Run** > **Dictionaries** > **Localization**).



To switch between languages, use the property:

```
@Client.Localization = "" // for default dictionary
//or
@Client.Localization = "<Dictionary_Name>"
```

To have a translated Alarm AuditTrail with Custom Messages and Comments in Reports, the addition of a callback function in **Script** > **Classes** > **ClientMa in** is required. This function is called every time the DataGrid object is modified.

The Callback function syntax is as follows:

```
public void OnReportCustomTableCell(string reportName, string columnName, System.Data.
DataRow row, System.Windows.Documents.TableCell tableCell)
{
// Insert Code Here
}
```

The code added to the callback function is presented below:

```
public void OnReportCustomTableCell(string reportName, string columnName, System.Data.DataRow row, System.
Windows.Documents.TableCell tableCell)
{
if (row["ItemName"].ToString() == "{object}")
{
string[]  Message_Split_Parts  =  row[columnName].ToString().Split('{',  '}'); string Translated_Message = "";

for (int i = 0; i <= Message_Split_Parts.Length - 1; i++) {

// Translate the custom message part
Translated_Message += @Client.Locale(Message_Split_Parts[i]);

Run  cellText  =  (tableCell.Blocks.FirstBlock  as  Paragraph).Inlines.FirstInline  as  Run;

// Replace the original message with the translated one. cellText.Text = Translated_Message;
}

}
}
```

The Datagrid language will depend on the dictionary that was enabled when the report was saved.