



## TECHNICAL LIBRARY

AS A SERVICE TO THE  
HYDROCARBON MEASUREMENT  
INDUSTRY, CRT-SERVICES  
CURATES THIS COLLECTION OF  
DIGITAL RESOURCES.



For Microsoft Excel

## **Advanced Topics Reference**

Programming topics,  
Wizards, Tools,  
Controls,  
Alarm Management,  
Trending,  
Relational Databases,  
Client/Server,  
Redundancy,  
Multiple Languages,  
Terminal Services,  
Customization & Troubleshooting

Product	eXlerate 2010 advanced topics reference
Reference number	03-0110-2
Revision	A.1
Date	February 2012
Authors	H.A.J. Kok, H.F.J. Rutjes

**Disclaimer**

*Spirit IT has taken care in the preparation of this book, but makes no expressed or implied warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the issue of the information or programs contained herein.*

**Special note**

*The information contained in this document is the property of Spirit IT B.V., and may not be reproduced (wholly or in part) used or disclosed without the prior consent of Spirit IT B.V. and then on condition only that this notice is included in any reproduction or disclosure. The copyright and the foregoing restriction on copying, use and disclosure extend to all media in which this information may be embodied including magnetic storage.*

Printed in the Netherlands.

Copyright© 2001-2012 Spirit IT B.V., Eindhoven, the Netherlands. All rights reserved.

- ® eXlerate is a registered trademark of Spirit IT B.V.
- ® Microsoft Windows is a registered trademark of Microsoft Corporation.
- ® Microsoft Excel is a registered trademark of Microsoft Corporation.

Visit Spirit IT on the Web: <http://www.spiritIT.com>

# Table of contents

<b>Document Control-----</b>	<b>1-11</b>
Revision Coding -----	1-11
Revision History -----	1-12
<i>Revision A.0</i> -----	1-12
<i>Revision A.1</i> -----	1-12
<b>Chapter 1 - Programming topics-----</b>	<b>1-13</b>
Structuring your applications -----	1-13
<i>Introduction</i> -----	1-13
<i>Worksheet organization</i> -----	1-14
Calculation sheets -----	1-16
<i>Introduction</i> -----	1-16
<i>Calculation sheet layout</i> -----	1-16
<i>Example calculation</i> -----	1-20
<i>Calculation Wizard</i> -----	1-21
<i>Other tips for structured applications</i> -----	1-22
<b>Chapter 2 - Summer/winter-time -----</b>	<b>2-25</b>
Introduction -----	2-25
Adjusting time -----	2-25
<b>Chapter 3 - Wizards and Tools-----</b>	<b>3-29</b>
Introduction -----	3-29
Wizards -----	3-33
<i>Tag &amp; Object wizard (Ctrl+W)</i> -----	3-33
<i>Calculation wizard</i> -----	3-38
<i>Color Wizard</i> -----	3-40
<i>Button Wizard</i> -----	3-43
<i>Language Wizard</i> -----	3-45
Tools -----	3-47
<i>Shape Property Tool</i> -----	3-47
<i>Name Definition Tool</i> -----	3-49
<i>Color Palette Tool (Ctrl+L)</i> -----	3-49
<i>Alarm Tree Tool (Ctrl+M)</i> -----	3-51
<i>Generate Report</i> -----	3-52
<i>Generate HTML</i> -----	3-52
<i>Browse OPC Servers</i> -----	3-53
<i>Communications Options</i> -----	3-53
<i>Show Control Center</i> -----	3-53
<i>Unprotected cells marker (Ctrl+U)</i> -----	3-53
<i>Remove External links</i> -----	3-55
<i>Reset Historical values</i> -----	3-55
<i>Recalculate Application (Ctrl+R)</i> -----	3-55
<i>Import Sheets</i> -----	3-56
<i>Advanced Replace</i> -----	3-57
<b>Chapter 4 - Controls -----</b>	<b>4-59</b>

Introduction-----	4-59
Control Overview -----	4-59
Inserting Controls -----	4-59
Modifying Controls-----	4-60
Control Properties -----	4-61
<i>Backgrounds</i> -----	4-61
<i>Borders &amp; Lines</i> -----	4-65
<i>Margins</i> -----	4-65
<i>Fonts</i> -----	4-66
<i>Formats</i> -----	4-67
<i>Themes</i> -----	4-68
<i>Templates</i> -----	4-69
<i>List Columns</i> -----	4-70

## **Chapter 5 - Alarm management ----- 5-71**

Introduction-----	5-71
Alarm page types-----	5-71
Alarm components -----	5-72
Tag database -----	5-73
Alarm Group Table -----	5-73
Alarm Summary Control -----	5-74
Worksheet: Alarm History -----	5-84
<i>Alarm History - Theory of operation (Advanced topic)</i> -----	5-85
Worksheet: Event History-----	5-87
<i>Event History - Theory of operation (Advanced topic)</i> -----	5-88
<i>Event history buttons</i> -----	5-88
VBA and worksheet functions-----	5-90
Alarm dead-band -----	5-91
Other alarm options-----	5-92

## **Chapter 6 - Trending ----- 6-93**

Introduction-----	6-93
Enabling Trending -----	6-93
Trending Parameters -----	6-93
Visualizing Trend Data -----	6-94
exTrendChart Control -----	6-95
<i>Chart Area</i> -----	6-95
<i>Plot Area</i> -----	6-95
<i>Navigation</i> -----	6-96
<i>Data Cursor</i> -----	6-98
<i>Time Axes</i> -----	6-99
<i>Value Axes</i> -----	6-100
<i>Pen Defaults</i> -----	6-104
<i>Markers</i> -----	6-104
<i>Limits</i> -----	6-105
exTrendPenSelector Control -----	6-106
<i>Linking to a Chart control</i> -----	6-107
<i>Tags</i> -----	6-107
<i>Pen Sets</i> -----	6-108
<i>Pens</i> -----	6-109
exTrendNavigator Control-----	6-111
<i>Selection Area</i> -----	6-111
<i>Automatic zooming</i> -----	6-113

Accessing trend-data directly -----	6-114
Data storage -----	6-115
<i>Reading Trend Files</i> -----	6-115

## **Chapter 7 - Relational Databases----- 7-117**

Introduction -----	7-117
The embedded database -----	7-118
<i>Table layout</i> -----	7-118
<i>Database Identifiers</i> -----	7-119
<i>User definable tables</i> -----	7-119
<i>Driver specific info values</i> -----	7-120
<i>Redundancy &amp; Synchronization</i> -----	7-120
<i>Advanced settings</i> -----	7-121
<i>Corruption &amp; data loss</i> -----	7-122
<i>Troubleshooting</i> -----	7-122
External databases -----	7-123
<i>Configuration</i> -----	7-123
<i>MySQL Database Driver</i> -----	7-123
<i>SQLServer Database Driver</i> -----	7-126
SQL worksheet functions -----	7-128
<i>Introduction</i> -----	7-128
<i>Queries</i> -----	7-128
<i>Views</i> -----	7-129
<i>SQL Table</i> -----	7-130
<i>Scrollable views</i> -----	7-132
SQL VBA functions -----	7-136
<i>The 'SQLCmd' object</i> -----	7-136
<i>Executing SQL statements</i> -----	7-136
<i>Reading SQL results</i> -----	7-137
<i>Writing SQL results to worksheets</i> -----	7-138
<i>Executing a-synchronous SQL statements</i> -----	7-139

## **Chapter 8 - Client & Server----- 8-143**

Introduction -----	8-143
Network Configuration Assistant (xNet) -----	8-144
Server configuration -----	8-145
Client configuration -----	8-148
Advanced settings -----	8-152
Application development -----	8-153
<i>Names</i> -----	8-153
<i>Network overview</i> -----	8-155
<i>Conditional development</i> -----	8-156
<i>Event logging</i> -----	8-157
<i>Writing to IO devices</i> -----	8-157
<i>Synchronized parameters</i> -----	8-158
<i>Shared values</i> -----	8-158

## **Chapter 9 - Redundancy----- 9-161**

Introduction -----	9-161
Device redundancy -----	9-163
<i>Introduction</i> -----	9-163
<i>Configuration</i> -----	9-164
Device communication channel redundancy -----	9-166

<i>Introduction</i> .....	9-166
<i>Multiple active communication channels</i> .....	9-166
<i>Single active communication channel</i> .....	9-167
Server redundancy .....	9-171
<i>Introduction</i> .....	9-171
<i>Duty selection</i> .....	9-172
Network redundancy .....	9-177
<i>Introduction</i> .....	9-177
<i>Network considerations</i> .....	9-178
<b>Chapter 10 - Multiple languages</b> .....	<b>10-179</b>
Introduction .....	10-179
Setup Microsoft Windows to use multiple languages .....	10-180
Application support .....	10-184
<i>Adding multi-lingual support</i> .....	10-184
<i>Language worksheet layout</i> .....	10-185
<i>Adding languages</i> .....	10-186
<i>Adding user defined texts</i> .....	10-189
<i>Multi-lingual Tag Database</i> .....	10-190
<i>Multi-lingual Buttons</i> .....	10-191
<i>Multi-lingual worksheets</i> .....	10-192
<i>Multi lingual VBA code and forms</i> .....	10-193
<i>Language selection</i> .....	10-193
<b>Chapter 11 - Terminal Services</b> .....	<b>11-195</b>
Introduction .....	11-195
Requirements .....	11-195
<i>Operating System</i> .....	11-195
<i>Microsoft Office</i> .....	11-196
<i>eXLerate License</i> .....	11-196
Configuration .....	11-196
<i>Operating System</i> .....	11-196
<i>eXLerate</i> .....	11-197
Using Terminal Services .....	11-197
<b>Chapter 12 - Trouble shooting</b> .....	<b>12-199</b>
Introduction .....	12-199
Real-time data communications problems .....	12-200
Worksheet functions are excessively called .....	12-201
The application does not start up properly .....	12-202
<b>Chapter 13 - User notes</b> .....	<b>13-209</b>

## List of figures and tables

Figure 1-1: Worksheet naming convention .....	1-15
Figure 1-2: Calculation worksheet layout .....	1-17
Figure 1-3: Sub-groups in a calculation sheet .....	1-19
Figure 1-4: Using calculation tags in a worksheet .....	1-20
Figure 1-5: Example calculation .....	1-20

Figure 1-6: Calculation Wizard-----	1-22
Figure 2-1: Time Table -----	2-26
Figure 2-2: Disabled adjustment for daylight saving in Windows -----	2-27
Figure 3-1: Available Wizards in eXLerate -----	3-29
Figure 3-2: Available Tools in eXLerate -----	3-30
Figure 3-3: Invocation of the Tag & Object Wizard-----	3-34
Figure 3-4: Step 1/4 of the Tag & Object wizard -----	3-34
Figure 3-5: Step 2/4 of the Tag & Object wizard -----	3-35
Figure 3-6: Step 3/4 of the Tag & Object wizard -----	3-36
Figure 3-7: Step 4/4 of the Tag & Object wizard -----	3-37
Figure 3-8: Tag & Object wizard progress -----	3-38
Figure 3-9: Invocation of the Calculation Wizard -----	3-38
Figure 3-10: Step 1/1 of the Calculation Wizard -----	3-39
Figure 3-11: Color Table layout-----	3-40
Figure 3-12: Invocation of the Color Wizard -----	3-41
Figure 3-13: Step 1/1 of the Color Wizard -----	3-42
Figure 3-14: Invocation of the Button Wizard -----	3-43
Figure 3-15: Step 1/1 of the Button Wizard -----	3-43
Figure 3-16: Button Wizard completed-----	3-44
Figure 3-17: Invocation of the Language Wizard -----	3-45
Figure 3-18: Step 1/1 of the Language Wizard -----	3-45
Figure 3-19: Language Wizard completed -----	3-46
Figure 3-20: Selection of the Shape Tool -----	3-47
Figure 3-21: Shape properties Tool -----	3-48
Figure 3-22: Names Tool-----	3-49
Figure 3-23: Names Tool with opened name list-box -----	3-49
Figure 3-24: Selection of the Color Palette Tool -----	3-50
Figure 3-25: Color Palette Tool -----	3-50
Figure 3-26: Starting the Alarm directory tree tool -----	3-51
Figure 3-27: Alarm Directory Tool-----	3-52
Figure 3-28: Generate Report -----	3-52
Figure 3-29: Cell formatting dialog showing the protection status -----	3-54
Figure 3-30: Marked unprotected cells -----	3-54
Figure 3-31: Removing external links -----	3-55
Figure 3-28: Import Sheets-----	3-56
Figure 3-28: Advanced Replace-----	3-57
Figure 4-1: Inserting Controls on Worksheets -----	4-59
Figure 4-2: Design mode-----	4-60
Figure 4-3: Inserting on VBA forms -----	4-60
Figure 4-4: Properties button -----	4-60
Figure 4-5: Background types -----	4-61
Figure 4-6: Solid background color-----	4-61
Figure 4-7: Gradient background colors-----	4-62
Figure 4-8: Background pattern -----	4-63
Figure 4-9: Background picture-----	4-64
Figure 4-10: Borders & Lines -----	4-65
Figure 4-11: Margins -----	4-65
Figure 4-12: Fonts-----	4-66
Figure 4-13: Formats-----	4-67
Figure 4-14: Themes -----	4-68
Figure 4-15: Templates -----	4-69
Figure 4-16: List Columns-----	4-70
Figure 5-1: Alarm Group Table -----	5-73
Figure 5-2: Alarm Summary Control -----	5-74



Figure 5-3: Alarm Groups Tree -----	5-75
Figure 5-4: Alarm Groups Properties -----	5-75
Figure 5-5: Alarm Group State Properties-----	5-75
Figure 5-6: Alarm Summary List-----	5-76
Figure 5-7: Alarm List Colors -----	5-77
Figure 5-8: Alarm List Font-----	5-77
Figure 5-9: Using Alarm Filters -----	5-78
Figure 5-10: Configuring Alarm Filters -----	5-79
Figure 5-11: Alarm Filter Properties -----	5-79
Figure 5-12: Real-time Mode Properties-----	5-80
Figure 5-13: Alarm Acknowledgement Buttons -----	5-80
Figure 5-14: Alarm Suppression Buttons-----	5-81
Figure 5-15: Editing one or more Alarms -----	5-81
Figure 5-16: Edit Alarms Dialog -----	5-82
Figure 5-17: Alarm Summary Security Settings -----	5-83
Figure 5-18: Alarm history worksheet-----	5-84
Figure 5-19: An alarm history window with the 4 last alarm messages-----	5-85
Figure 5-20: Event history worksheet -----	5-87
Figure 5-21: Trending related VBA code -----	5-89
Figure 5-22: Alarm dead-band example used for a high limit alarm -----	5-91
Figure 6-1: Insert Trend Controls-----	6-94
Figure 6-2: exTrendChart Control -----	6-95
Figure 6-3: Plot Area properties -----	6-95
Figure 6-4: Plot area margins-----	6-96
Figure 6-5: Navigation properties -----	6-97
Figure 6-6: Data Cursor -----	6-98
Figure 6-7: Data Cursor Properties -----	6-98
Figure 6-8: Data Cursor Date/Time-----	6-98
Figure 6-9: Pen Label Properties-----	6-99
Figure 6-10: Pen Label Color-----	6-99
Figure 6-11: Time Axes properties -----	6-100
Figure 6-12: Value Axes properties-----	6-100
Figure 6-13: Scaling properties -----	6-101
Figure 6-14: Value Axis Styles -----	6-102
Figure 6-15: Percentage Axis -----	6-102
Figure 6-16: Engineering Units Axis -----	6-103
Figure 6-17: Using both Percentage and Engineering Units Axis -----	6-103
Figure 6-18: Showing Engineering Units Axis for specific pen -----	6-103
Figure 6-19: Default Pen Style-----	6-104
Figure 6-20: Markers on the Chart -----	6-104
Figure 6-21: Selecting Markers using the PenSelector -----	6-104
Figure 6-22: Alarm Limits on Chart-----	6-105
Figure 6-23: Limits option in PenSelector -----	6-105
Figure 6-24: Add Limits Column to PenSelector -----	6-105
Figure 6-25: exTrendPenSelector Control -----	6-106
Figure 6-26: PenSelector individual panes -----	6-106
Figure 6-27: Selecting a Target Chart-----	6-107
Figure 6-28: Editing names -----	6-107
Figure 6-29: Tag selection -----	6-107
Figure 6-30: Tags Properties-----	6-108
Figure 6-31: Pen sets-----	6-108
Figure 6-32: Pen Sets Properties -----	6-108
Figure 6-33: Pen-set files -----	6-109
Figure 6-34: Pens Toolbar and List -----	6-109

Figure 6-35: Pens Properties-----	6-109
Figure 6-36: exTrendNavigator Control -----	6-111
Figure 6-37: Selection Area Properties -----	6-112
Figure 6-38: Selection Area -----	6-112
Figure 6-39: Auto Zoom Properties-----	6-113
Figure 6-40: Moving the left- and right edges -----	6-113
Figure 7-1: Embedded Database Path-----	7-118
Figure 7-2: External Database Table -----	7-123
Figure 7-3: SQL query & view relation -----	7-129
Figure 7-4: SQL Table -----	7-130
Figure 7-5: Example exSQLViewQuery(..) worksheet function -----	7-131
Figure 7-6: Creating worksheet variables for scrollable view -----	7-132
Figure 7-7: Select scrollable range -----	7-133
Figure 7-8: Scrollable range formula -----	7-133
Figure 7-9: Creating a scrollbar -----	7-133
Figure 7-10: Formatting the scrollbar -----	7-134
Figure 7-11: Attach scrollbar to variable -----	7-134
Figure 7-12: Update scrollbar maximum value -----	7-135
Figure 7-13: Synchronously executed SQL statements -----	7-139
Figure 7-14: A-synchronously executed SQL statements -----	7-139
Figure 8-1: xNet license detection-----	8-144
Figure 8-2: Server configuration-----	8-145
Figure 8-3: Share Report folder on server -----	8-146
Figure 8-4: Server communication statuses -----	8-146
Figure 8-5: Client configuration-----	8-148
Figure 8-6: Client network drive mapping -----	8-150
Figure 8-7: Client shortcut configuration-----	8-151
Figure 8-8: Client communication statuses -----	8-151
Figure 8-9: Advanced settings -----	8-152
Figure 8-10: Network overview example-----	8-155
Figure 8-11: Network adaptor animation -----	8-155
Figure 8-12: Conditional worksheet function -----	8-156
Figure 8-13: Synchronized parameters -----	8-158
Figure 8-14: Shared value worksheet function -----	8-159
Figure 8-15: Shared value on multiple computers -----	8-159
Figure 9-1: Tag Database with redundant IO-----	9-164
Figure 9-2: Query Tables write to multiple columns in Tag DB -----	9-164
Figure 9-3: Primary and secondary protocols-----	9-167
Figure 9-4: Primary and secondary protocol queries -----	9-167
Figure 9-5: Primary protocol device failure -----	9-168
Figure 9-6: Swapping protocol devices-----	9-168
Figure 9-7: Alternate protocol device -----	9-168
Figure 9-8: Protocol device selection -----	9-168
Figure 9-9: Protocol table with device selection -----	9-169
Figure 9-10: Device selection columns-----	9-169
Figure 9-11: Protocol device selection periodic handler -----	9-169
Figure 9-12: Protocol device selection code -----	9-170
Figure 9-13: Duty selection -----	9-172
Figure 9-14: Local Duty status-----	9-172
Figure 9-15: IO Routing-----	9-174
Figure 9-16: Query Table with routing support -----	9-175
Figure 9-17: Query Table routing columns-----	9-175
Figure 9-18: Query Table routing formula -----	9-176
Figure 9-19: Query Table routing formula pseudo code-----	9-176

Figure 9-20: Network redundancy -----	9-177
Figure 9-21: Redundant network cards on a single network -----	9-178
Figure 9-22: Redundant network cards on separate networks -----	9-178
Figure 10-1: Regional and language options -----	10-180
Figure 10-2: Supplemental language support -----	10-181
Figure 10-3: Installing language packs -----	10-182
Figure 10-4: Skip file copying during language pack install -----	10-183
Figure 10-5: Restart computer after installing language pack -----	10-183
Figure 10-6: Multi -lingual worksheet 'xLanguage' -----	10-184
Figure 10-7: Creating a new language -----	10-184
Figure 10-8: Language Wizard ourput -----	10-185
Figure 10-9: Language worksheet without formatting -----	10-185
Figure 10-10: Language worksheet layout -----	10-185
Figure 10-11: Adding languages -----	10-186
Figure 10-12: Opening the 'Language.xls' file -----	10-187
Figure 10-13: Select desired language -----	10-187
Figure 10-14: Select language column -----	10-187
Figure 10-15: Select 'xLanguage' worksheet -----	10-188
Figure 10-16: Locate first empty column in language worksheet -----	10-188
Figure 10-17: Succesfully added language -----	10-188
Figure 10-18: Use 'Apply Worksheet Changes' after adding a language -----	10-188
Figure 10-19: User defined texts in Language worksheet -----	10-189
Figure 10-20: User defined texts are terminated by two or more empty rows -	10-189
Figure 10-21: Sorting user defined texts -----	10-189
Figure 10-22: Insert column into the Tag Database -----	10-190
Figure 10-23: Rename column to proper language -----	10-190
Figure 10-24: Multi-lingual Button Table -----	10-191
Figure 10-25: Choosing a multi-lingual button text -----	10-191
Figure 10-26: Selecting a new language -----	10-191
Figure 10-27: 'exLanguageText' worksheet function -----	10-192
Figure 10-28: Linking a control to a cell or name -----	10-192
Figure 10-29: Language text referenced by a control -----	10-192
Figure 10-30: Multi-lingual VBA code -----	10-193
Figure 10-31: Multi-lingual user form -----	10-193
Figure 10-32: Obtain currently selected language -----	10-194
Figure 10-33: Selecting a new language -----	10-194
Figure 10-34: Language selection buttons/pictures -----	10-194
Figure 10-35: Assing language selection macro to picture -----	10-194
Figure 11-1: Terminal Services Setup -----	11-195
Figure 11-2: Locating the Remote Desktop Client -----	11-198
Figure 11-3: Remote Desktop Connection Window -----	11-198
Figure 12-1: Example of a VB function causing excessive calling -----	12-201
Figure 12-2: Example of a correct VB function avoiding excessive calling -----	12-201

# Document Control

## ***Revision Coding***

All documents are supplied with a revision code. This code has the following format:

<major revision letter>.<minor revision number>. Initially, the document has revision code A.0. When in the next release of the document minor changes were implemented, the minor revision number increases. When major changes have been implemented, the major revision number increments.

Example document:

- A.0 First revision
- A.1 Second revision with minor changes implemented
- A.2 Third revision, with other minor changes
- B.0 Fourth revision, with (a) major change(s).

The revision coding will be modified for each new release of a document.

All software packages and software modules or components will be provided with a version number. This number consists of three parts: A release number, a major revision number and a minor revision number separated by decimal points. A release number identifies the generation number of the software, the major number refers to the main functionality of the program, seen from the user's point of view, while the minor revision number identify a new software version.

Example program:

- 1.01.001 Initial release
- 1.01.002 Minor change
- 1.02.001 Major change
- 2.01.001 Family change

## ***Revision History***

### ***Revision A.0***

Author : H.A.J. Kok, H.F.J. Rutjes  
Date : May 2002 - November 2011

Initial release of the eXLerate 2010 Reference Manual Volume II.

### ***Revision A.1***

Author : H.F.J. Rutjes  
Date : February 2012

Added 'Import Sheets' and 'Advanced Find & Replace' documentation.

# Chapter 1 - Programming topics

## Structuring your applications

### Introduction

User-defined calculations in **eXlerate** may be placed everywhere: on potentially each worksheet cell, an expression containing worksheet functions may be entered. In various worksheets, for example in the tag database, next to the last database columns, such calculations may be entered. On operator display worksheets, user-defined expressions may be entered, for example to calculate the engineering units of a process parameter.

In itself it is acceptable to use this approach, and technically correct. However, there are a number of drawbacks to this 'at-random' approach:

- It is difficult to obtain an overview of a complicated calculation, because functionality is scattered over various worksheets.
- It is difficult to do maintenance on an application, because each application is specifically built in a unique way. "Where did I put the equation for unit conversions again?"
- Redundancy in application engineering is hard to avoid, because on various sheets, the same result is required. This result is likely to be re-implemented each time it is needed.
- Unstructured applications are likely to be created when calculations are scattered over the workbook. The drawbacks of creating and maintaining unstructured programs are too many to discuss in this context.

It is clear from the above that a structured application has many advantages:

- It is easy to obtain an overview of a complicated calculation, because functionality is grouped together over well-defined worksheets.
- Maintenance on an application can be done by various programmers, since all applications are similarly built according to the same principle.
- Redundancy will be avoided: each calculation is present only once, on a dedicated worksheet area.
- Structured applications are the result, improving both the software quality and project duration. The predictability of application engineering improves.

In this section, you'll learn how to setup your application in a structured way. Especially a *calculation worksheet* will be introduced, and its conventions and usage.

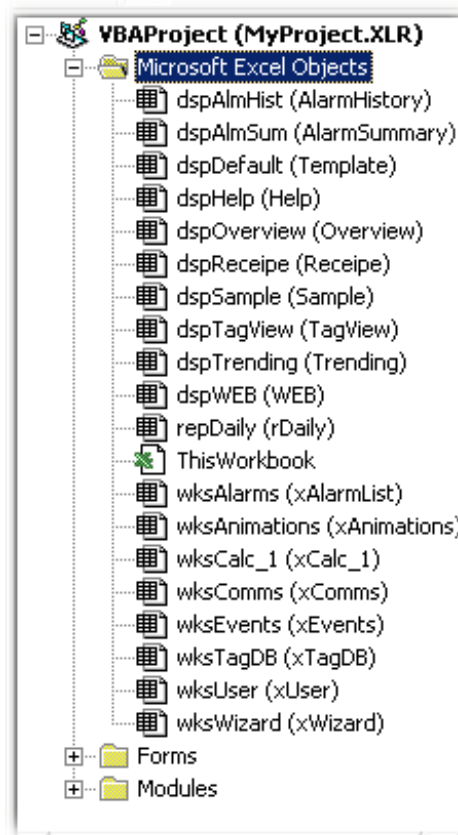
*This reference manual however is not a learning book on structured programming. If you feel uncomfortable with structured programming, please check your bookstore for one of the many titles available on this area.*

*If your application doesn't require any user-defined calculations, you might want to continue to a next chapter, although it would be interesting to take a look at how to setup a structured application.*

## **Worksheet organization**

There seem to be a lot of worksheets in the 'MyTemplate' sample project at first sight. There are many display and report sheets and a number of additional configuration worksheets containing various tables. In **eXLerate**, the following naming convention is used for worksheets:

- Display sheets are worksheets, which are used as operator displays. These worksheets may have any sheet name; internally these sheets have an object name starting with 'dsp'.
- Report sheets are worksheets, which are used as internal report pages. The names of these worksheets start with a lowercase 'r'; internally report sheets have an object name starting with 'rep'.
- Configuration sheets are all other worksheets used by **eXLerate** for its real-time HMI functionality. The name of such a worksheet starts with a lowercase 'x'; internally a configuration sheet has an object name starting with 'wks'. This is a convention used by **eXLerate**, and users are expected to comply with this convention.

**Figure 1-1: Worksheet naming convention**



## Calculation sheets

### Introduction

A calculation sheet is a worksheet containing logically grouped worksheet calculations, which are internally used by the application. The name of a calculation worksheet starts with a lowercase 'x', because it is neither a display nor report sheet. In the 'MyProject' sample, worksheet '**xCalc\_1**' is such a calculation sheet.

In a calculation worksheet, calculation results are orderly grouped. In addition, special support from **eXLerate** is available.

### Calculation sheet layout

Many HMI software packages are built using predefined objects, where a user is able to create one or more instances of an object. For example, a motor operated valve can be implemented using a generic 'valve' object. When the user needs six valves in an application, the valve object is copied six times, and the object properties are defined per valve instance.

In Excel/**eXLerate**, such complete objects do not literally exist as such, although practical implementation is almost identical to the valve example above.

In a spreadsheet, a calculation result can be obtained using various rows for the calculation. For example, a motor operated valve is controlled via nine parameters, which are grouped together on nine consecutive rows in our spreadsheet.

Such a calculation result over multiple rows may be functionally thought of as an 'object'. Rather actually than creating multiple objects, the calculations are copied over multiple columns to create multiple instances of the 'object'.

Calculations for multiple valves may be defined in multiple columns, where each column represents a single valve 'object' instance.

The calculation worksheet is set up to support such multiple sections.

The 'xCalc\_1' calculation sheet looks as follows:

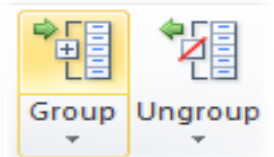
1	2	A	B	C	D	E	F	G	H
1		Calculations 1 (version 1.3)							
2									
3		Group	CalcTag	Description	Type	Value	Store	_1A	Store
11		Colors and naming conventions							
49		Global Settings							
51		Valves							
52				Run valves				R1_V1	
53				Valve closed status	TAG			1	
54				Valve opened status	TAG			1	
55				Valve failure alarm	TAG			0	
56			Calc.ValveIndex1	Status	CALC			2	
58		Valves							
73		Transmitters							
106		Devices							

**Figure 1-2: Calculation worksheet layout**

The worksheet contains a *title line*, a *header line* with various *columns*, and vertically grouped *sections*. A group may be optionally divided in *subgroups*.

The *Title line* is used to identify the calculation worksheet in the application for the application developer. In the 'MyProject' sample project, it contains a major, and a minor version number.

In the figure above, all groups are at outer group-level '1', except group 'Valves', which is opened. 'Valves' contains a sub-group: 'Run valves', and various calculations. The worksheet functions/constants/values under the '\_1A' column are colored blue and green.



The following columns are defined in the header line at row #3:

#### ● Group

This column contains the logical group name, under which the calculations are identified, such as 'Global settings', 'Valves', and 'Transmitters'. The group names are used to row-wise group the items using the 'Data', 'Group' command from the Excel menu bar. When rows are grouped together, small '+', '-', '1' and '2' buttons appear in Excel, with which groups may be opened and closed. To close all groups, click on the '1' button to activate the 1<sup>st</sup> group level. To open all groups to the 2<sup>nd</sup> level, click on the '2' button.

#### ● CalcTag

The name of the 'calculation tag' is defined at this column. A calculation tag is a tag that may be referred to in the application, including reports. eXlerate creates various name objects for this calculation tag.

For example, when the name: `'Calc.MyName'` is entered, **eXLerate** generates a name: `'Calc.MyName'` at the `'_Value'` column\*. More names may be created, depending on more columns starting with an underscore character\*\* (`'_'`).

In the example above, because the section `'_1A'` is defined at calculation tag `'Calc.ValveIndex1'`, the name: `'Calc.ValveIndex1_1A'` is created. Using this method, consistent object names are created for multiple sections, where each additional section has an associated additional column, such as `'_2'`, `'_3'` etc.

A calculation tag name should always start with `'Calc.'` as a convention. If this field is left empty, no name is created.

#### ● **Description**

This optional field contains the user-defined description of the calculation at the `'_Value'` or `'_{number}'` tag.

#### ● **Type**

This optional field contains the user-defined type of the calculation tag, for example: *CONST*, *SET*, *CALC*, or *TAG*, for respectively a constant number, a value set by a VBA-procedure, a calculation, or a value from the tag database.

#### ● **\_Value**

At this column, the *Calculation Wizard* creates the calculation name as defined under *CalcTag*, for calculation tags without additional sections.

#### ● **Store**

At this column, the storage worksheet function `exStorageID (...)` for the `'_Value'` column may be entered. Storage worksheet functions may be used to create retentive constants in a calculation worksheet. Retentive constants are automatically stored on disk, and loaded back from disk at system startup.

#### ● **\_{number}**

Additional columns containing a section identification, for example `'_1'`, or `'_1A'` may be added to the row. An additional section should start with an underscore (`'_'`).

### **Color-coding**

It is strongly recommended to use distinct colors for various items, in order to be able to distinguish the source of an item on a calculation sheet.

---

\* A field should be non-empty to create a name in a calculation sheet

\*\* Such additional fields may be used for additional sections, such as a machine number or metering line

In the 'MyTemplate' project, the following color-coding is used:

Description	Shortcut	Type	Color
Constants	CONST	Constants	Black
Value set in Program code or worksheet	SET	VBA/Edit	Orange
Value calculated in sheet formula	CALC	Formulae	Blue
Input from tag database	TAG	Tag base	Green
To be implemented, not completed yet	???	Error/To Do	Red

**Table 1-1: Color-coding in a calculation sheet**

Using color-coding, best results are obtained to obtain a quick insight in the application.

- The color-coding table above is available in the sample project as the first group: 'Colors and naming conventions'. It may be opened/closed at any time during application development using the '+' and '-' group-buttons at the left hand side of the page.

### Using sub-groups

A sub-group is a member of a certain group. For example, in the group: 'Global Settings', a sub-group is defined: 'Version Control', as in the figure below:

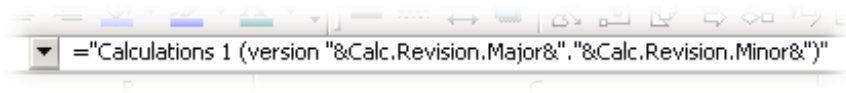
12					
13	Global Settings				
14		Version control			
15	Calc.Revision.Hi	Major revision number	CONST	1	
16	Calc.Revision.Lo	Minor revision number	CONST	3	
17					
18		Max Counters			

**Figure 1-3: Sub-groups in a calculation sheet**

The sub-group has a title under the *Description* field of a group, and a grey bar across all columns which are included in the calculations for that sub-group. In the example above, the revision of the application is maintained with two constants.

The sub-group has two CalcTags defined: 'Calc.Revision.Hi', and 'Calc.Revision.Lo', which are two constants stored in the cell. Since these two constants are not required to be actually retentive, but are changing only when an application engineer modifies the application itself, it are two constants rather than two retentive values, hence the *Store* columns of the two calculation tags are left empty.

In another worksheet, or in VBA, these constants may be referred to with their logical 'object' names 'Calc.Revision.Hi', and 'Calc.Revision.Lo' rather than the direct cell-references, as in the title bar of this calculation worksheet:



**Figure 1-4: Using calculation tags in a worksheet**

Although looking somewhat complex for a simple item like a version number, its value is better to understand when the application contains hundreds or maybe even thousands of calculations and constants.

### Example calculation

In the example calculation below, there will be an index calculated from three digital inputs from two identical motor-operated valves, each on a different production line. The three digital inputs for each valve are obtained from the tag database.

In the group: 'Valves', a sub-group is defined: 'Run-valves'; in the sample application it is the only sub-group. For each valve, there are three inputs from the tag database: a digital input which becomes '1' when the valve is closed, a digital input which becomes '1' when the valve is opened, and a digital input which becomes '1' when an error occurred at the valve's remote control unit.

A calculation should be done to convert valve positions to an index. The index is used for animation of a valve symbol in the application: when the valve is open, it should be colored green etc., according to the table below:

Description	Status	Value	Color
Valve is opened	OPEN	1	Green
Valve is closed	CLOSE	2	Red
Value is traveling	TRAVELLING	3	Yellow
Valve has an illegal status	ILLEGAL	4	Violet

The example calculation is shown below:

Group	CalcTag	Description	Type	Value	Store	Line
Colors and naming conventions						
Global Settings						
Valves						
		Run valves				R1_V1
		Valve closed status	TAG			1
		Valve opened status	TAG			1
		Valve failure alarm	TAG			0
	Calc.ValveIndex1	Status	CALC			2
Valves						

**Figure 1-5: Example calculation**

The three digital inputs from the tag database are colored green, as per color convention.

The calculation is done using the VB function `GetValveIndex(...)`, which is also included in the sample project.

This function converts the three inputs to an index, ranging from 1..4 for CLOSE, OPEN, TRAVEL or ERROR status of the valve.

R1_V1	R1_V2	R1_V3
1	1	
1	0	
0	1	
<code>=GetValveIndex(G53+0,G54+0,G55+0)</code>		



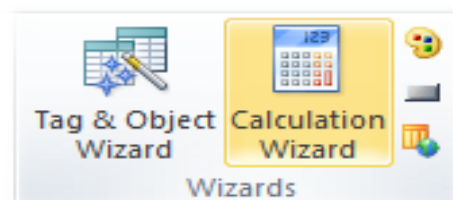
*Note the '+0' addition to the function arguments. All VB functions should be called from a worksheet using the '+0' convention for each argument. The reason why is explained in Chapter 10, 'Worksheet functions are excessively called' on page 12-201, and has to do with internal recalculation issues of Excel.*

There is only one name created for this section: `'_1A'`, which is `'Calc.ValveIndex1_1A'`. Multiple valves for other sections are simply created by copying this column to another column `'_2A'`. There are no names created for the individual valve bits, because no name appears for these tags in the *CalcTag* column.

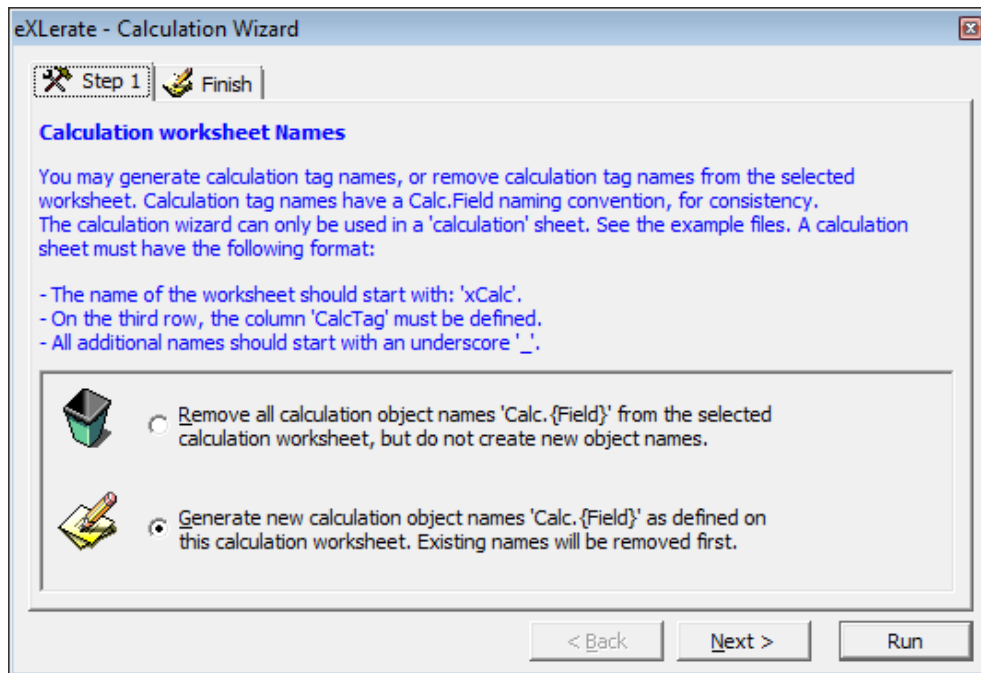
It may be clear that searching for valve status values in the application is easily found: in the calculation sheet, at section 'Valves'. The values may be checked and tested right here in the worksheet, and avoids unexpected application behavior.

## Calculation Wizard

Calculation names in a calculation worksheet are created using the appropriate Wizard from the **eXlerate** ribbon:



The following wizard dialog appears when 'Calculation Wizard' option is activated:



**Figure 1-6: Calculation Wizard**

There are basically two choices: All calculation names may be removed, without generation of new 'Calc.' object names, or alternatively, new object names are generated.

Press <Enter> to activate the default 'Run' button, after which the specified calculation object names are generated.

Verify with a closed Calculation Wizard that the required object names are properly generated.

## ***Other tips for structured applications***

### ***Introduction***

Although calculation worksheets alone are not the only ingredient in creating structured applications, many real-world, professional, high-demanding applications have shown that setting up all your calculations in a project using some sort of calculation worksheet is a good approach for project development.

### ***Color-coding***

Color-coding of the calculation worksheets is another great idea to keep track of the origin of the data, so you are able to understand more quickly how an application is structured.

### **Transparency**

A well-designed application should be both transparent to an application *engineer*, who is responsible of creating a technical correct application, as well as to an application *tester* who is responsible of verifying that an application indeed behaves as defined in the project specification or functional design specification.

Project transparency can be full explored by using the strength of a spreadsheet for calculation purposes.

Most ingredients, such as the configuration tables of **eXLerate**, are designed for transparency and efficient programming: each configuration table can be configured, verified, and carefully tested, both in 'real-world' Runtime mode, or in Verify mode, where expressions and calculations can still be monitored, verified and even modified with data-communications actually running.

### **Simulation & Testing**

Data communications can be simulated to verify all calculations in an application. Step by step, all of the entries in all tables can be verified. When all configuration tables have been thoroughly tested, the application is well tested; it's as simple as that.

For data-communications, a data-scope/debugger is available to check all incoming and outgoing messages.

The configuration of an application can be easily documented as well, simply by printing most or all of the worksheets.

### **Application development**

Functionality still missing in an application, or even in **eXLerate** may be added by a user or application developer. For example, you are able to create your own tools and wizards using VB(A).

All of these ingredients should be more than adequate to create correct, powerful and yet simple applications that are easily tested and maintained afterwards.

And best of all, you are using your most popular spreadsheet program to achieve all this!

In the following chapter, you will be introduced to how Visual Basic for Application is integrated in **eXLerate**, and how to use VBA in your real-time HMI application.



*This page is intentionally left blank.*

# Chapter 2 - Summer/winter-time

## Introduction

*eXlerate* is able to automatically detect a summer- and winter- changeover.

More than that, *eXlerate* is able to automatically update externally connected devices, and change these devices from summertime to wintertime or vice versa.

## Adjusting time

To avoid fiscal integrity problems for many devices with respect to these changeovers, *eXlerate* adjusts the time in two half hour steps, avoiding full hour changes, and thus invalidating hourly based counters.

This segmented changeover method is a generally accepted method for systems requiring summer- and wintertime changeovers, including fiscal metering applications for custody transfer purposes.

Changes to summertime and wintertime are automatically catered for by the internal 'Latch (...)' worksheet functions, as follows:

### ● Summer- to Wintertime changeover

When the time changes from summer- to wintertime, officially, the clock is set backwards for one hour, e.g. at 03:00 at night, the time is adjusted to 02:00. In *eXlerate*, summertime to wintertime changeovers take place in two steps:

- At 02:45, the time is adjusted to 02:15, causing a half hour correction
- Again, at 02:45 the time is adjusted to 02:15, causing the second half hour correction.
- After this sequence, latches of 02:00 – 03:00 contain 'double' hours, since the corresponding hourly interval event took two hours. This is a de-facto standard behavior in time-switching industrial systems, including custody transfer metering systems.

### ● Winter- to summertime changeover

When the time officially changes from wintertime to summertime, the clock is advanced for one hour, e.g. at 02:00 at night at the designated date, the time is advanced for an hour to 03:00. In *eXlerate*, this changeover takes place in the following steps:

- At 02:15, the clock is advanced to 02:45, causing a correction of half an hour. At 03:00, a normal hourly transition takes place. The latched data of 02:00- 03:00 contain only for a half hour of data.

- At 03:15, the clock is advanced again, to 03:45, causing the second correction to take place. At 04:00, a normal hourly transition takes place, again with a half hour of latched data.
- The net result of advancing the clock a full hour is that the fiscal reports contain two reporting hours each containing a half hour of data. Also this behavior is generally accepted by the industry.

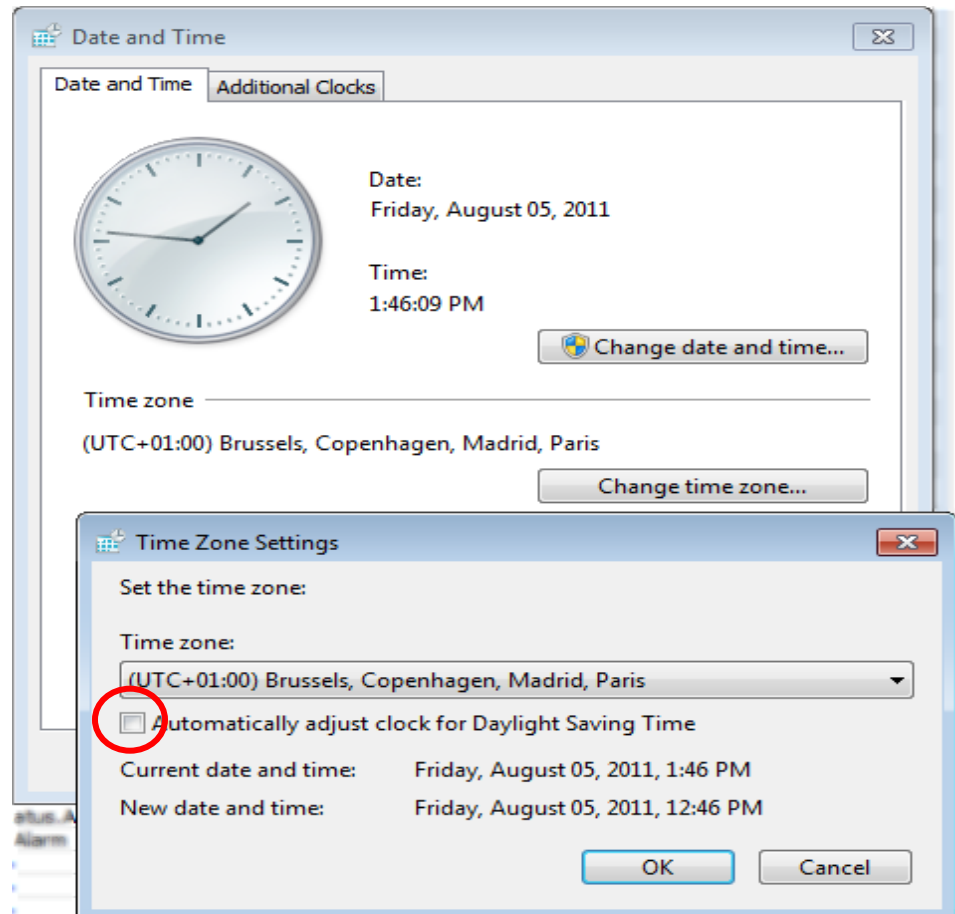
The dates for summer- and winter time changeovers may be defined in the application, in the *Time Table*, and has a layout as follows:

Time table						
rTimeTable						
Smon	Sday	Shr	Wmon	Wday	Wshr	
1	18	5	1	18	10	

**Figure 2-1: Time Table**

The layout is a typical configuration table containing the table name ('rTimeTable'), and columns headers containing the month, day and hour of the moment at which summertime becomes active (the switch-over point), and the month, day, and hour at which the wintertime becomes active.

When changeovers are enabled, the built-in Windows changeovers must be disabled, as follows:



**Figure 2-2: Disabled adjustment for daylight saving in Windows**

To fully implement date/time synchronization with external devices, the following aspects would be needed to cater for as well:

- To effectively enable summer- and winter changeovers, you must call the 'exEnableSummerWinter()' method of the 'Comms' object in VBA once. These settings are stored in the registry and are permanent hereafter.
- To enable time updates to external devices, you must implement an update to the device using the tag database, in which the current time and date should be sent to the device. Sometimes, this is done via regular write registers; in other cases a dedicated number must be written to the device. You could utilize the calculation worksheet for this purpose if a dedicated format is required.

- You could consider implementing a dedicated write-query for this purpose, so the time/date update is sent to the device independently from other write commands to the device.
- You could create a separate routine in VBA, which updates all connected devices simultaneously by triggering the appropriate time registers of these devices, for example using the `'exUpdateForce (...)'` method of the `'Comms'` object.

When summer- and winter changeovers are effectively enabled, a message is sent to the event logger:

`'SummerTime is set for dd/mm/yyyy hh:15:00; WinterTime is set for dd/mm/yyyy hh:45:00. Currently in xxxxxTime'.`

## Chapter 3 - Wizards and Tools

*In the previous chapters, you have already been introduced to most of the available Wizards and Tools of eXLerate.*

*A **Wizard** in this context is a development assistant that generates parts of an application for you, while a **Tool** is a smaller utility to help you with certain aspects of an application.*

*For example, the Color Wizard is called a wizard, because it is able to create/generate a complete color palette table for you, while the Color Palette Tool simply shows to you the relation between an index number and its actual color.*

*Both utilities have in common that both types help you with application development.*

*Under the Tools menu in eXLerate finally, a number of additional miscellaneous functions have been placed that help you in application development.*

*In this chapter, all available wizards, tools and additional miscellaneous functions are discussed. It may therefore be viewed as the reference chapter for development utilities.*

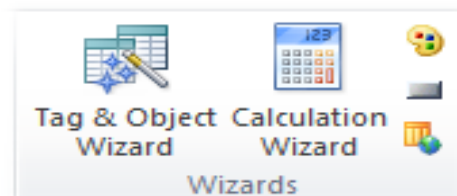
### Introduction

Because of the fact that eXLerate is worksheet oriented – most of its configuration tables is laid down in tables – an application can be constructed from these tables, where the user has the convenience to be able to use Excel as a powerful engineering environment.

The open architecture of Excel allows you to use the powerful environment of Excel, as well as additionally developed tools and wizards. You may use VBA and/or VB for development of your own assistants, tools and wizards. What other SCADA/HMI software development tool allows you to do just that?

In the menu of eXLerate you will be able to locate all wizards and tools.

The following wizards are currently defined:



**Figure 3-1: Available Wizards in eXLerate**

**• Tag & Object Wizard**

This is a development aid that creates functionality on basis of tag data-base definitions, such as worksheet functions, and alarm functionality.

**• Calculation Wizard**

This is a wizard that creates calculation object names for you in a special background worksheet, called a calculation sheet.

**• Color Wizard**

The color wizard is a wizard that is able to efficiently work with color palette tables in eXlerate.

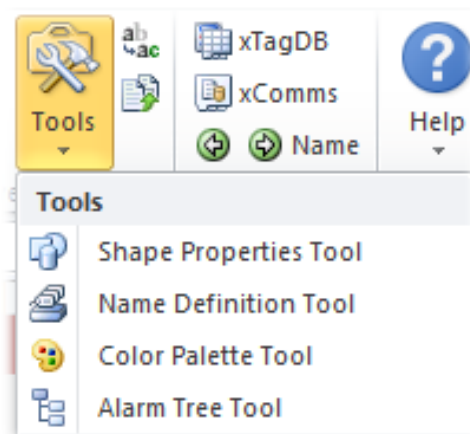
**• Button Wizard**

The Button Wizard creates menu navigation functionality in your application, based on the Button Table.

**• Language Wizard**

The Language Wizard creates a translation table in which applications can define system texts in multiple languages.

The following options are available under the tools section in eXlerate:



**Figure 3-2: Available Tools in eXlerate**

**Tools****• Shape Property Tool**

This tool helps you in finding shapes over various worksheets, and their properties as required for animations.

**• Name Definition Tool (Ctrl+E)**

The standard Excel dialog for names lacks functionality that you rather would like to include during application engineering. This tool offers an alternative for the built-in names dialog of Excel.

● **Color Palette Tool**

This is a tool showing you the relation between a color palette index number, and the actual color for shape animations.

● **Alarm Tree Tool**

This tool visually shows the hierarchical alarm tree structure.

**Generate**

● **Generate Report**

This option allows you to quickly generate a report for engineering purposes.

● **Generate HTML**

This option generates the HTML pages.

**Communications**

● **Browse OPC Servers**

This tool allows you to browse OPC servers and paste Item-ID's into the application.

● **Communications Options**

Displays the communication options.

**Protected Cells**

● **Mark Unprotected Cells (Ctrl+U)**

Marks all unprotected cells in a worksheet.

● **Unmark Unprotected Cells (Ctrl+Shift+U)**

Un-marks all unprotected cells in a worksheet.

**Miscellaneous**

● **Remove External links**

Removes all external links in an application workbook.

● **Reset Historical values**

Reset the historical values of the current application. All retentive intermediate results as maintained in the registry are reset.

● **Recalculate Application (Ctrl+R)**

Re-calculates the whole application.



To the right of the 'Tools' button, two more tools are visible:

- **Import Sheets**  
Imports sheet into the application.
- **Advanced Replace**  
Finds & replaces cells, names, objects, etc...

In the following sections, all wizards and tools will be discussed.

## Wizards

A Wizard in **eXLerate** is a development assistant that generates parts of an application for you. For each wizard, certain options can be defined prior to actually running the wizard. Each option is presented to you in an *option step*. You may use the <Ctrl-TAB> key, or the '<Back>' and '<Next>' buttons to browse through the steps.

When all option steps have been defined, you may actually run the wizard in the final step. In this last step, a list-box is presented in which the progress of the wizard is displayed, and other messages are reported. Errors during this process are also reported to **eXLerate**'s event logger.

The option buttons in each step will be remembered the next time that you run the wizard, so simply by pressing <Enter> the wizard may be re-run using the same options as the last time the wizard was invoked.

### Tag & Object wizard (Ctrl+W)

The tag and object wizard is an important assistant in the automation of application engineering. Various entities in an application can be created using the tag & object wizard:

- **Object names / calculations**

The tag database contains tag definitions. Although cells in the tag database may be directly referred to, it is much better to use logical object names. These names, and the required subsequent calculations may be created manually, or the tag and object wizard is able to generate such names and calculations for you – automatically.

- **Alarms**

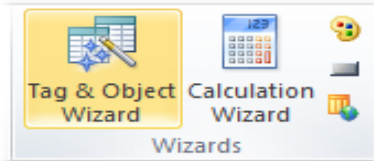
The alarm are automatically created for you when running this wizard, based on alarm properties of a tag at the tag database.

- **Cell error checks**

Rather than checking worksheet cells for errors manually, the Tag & Object wizard scans through all worksheets and reports errors for you.

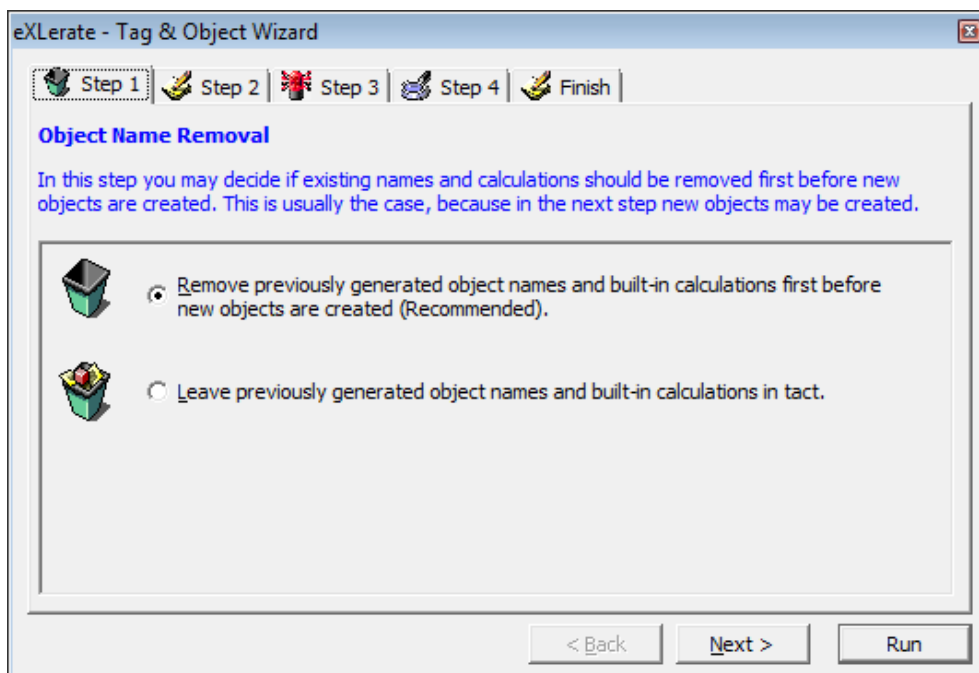
These three components may be all generated in a single action, or alternatively, step by step. The way the components are updated/generated is defined from a typical wizard dialog, where all steps are defined, then finalized, after which the generation process starts.

The tag & object wizard is started from the **eXLerate** ribbon or using Ctrl+W, as follows:



**Figure 3-3: Invocation of the Tag & Object Wizard**

The following wizard dialog appears, when the Tag & Object wizard is actually started:



**Figure 3-4: Step 1/4 of the Tag & Object wizard**

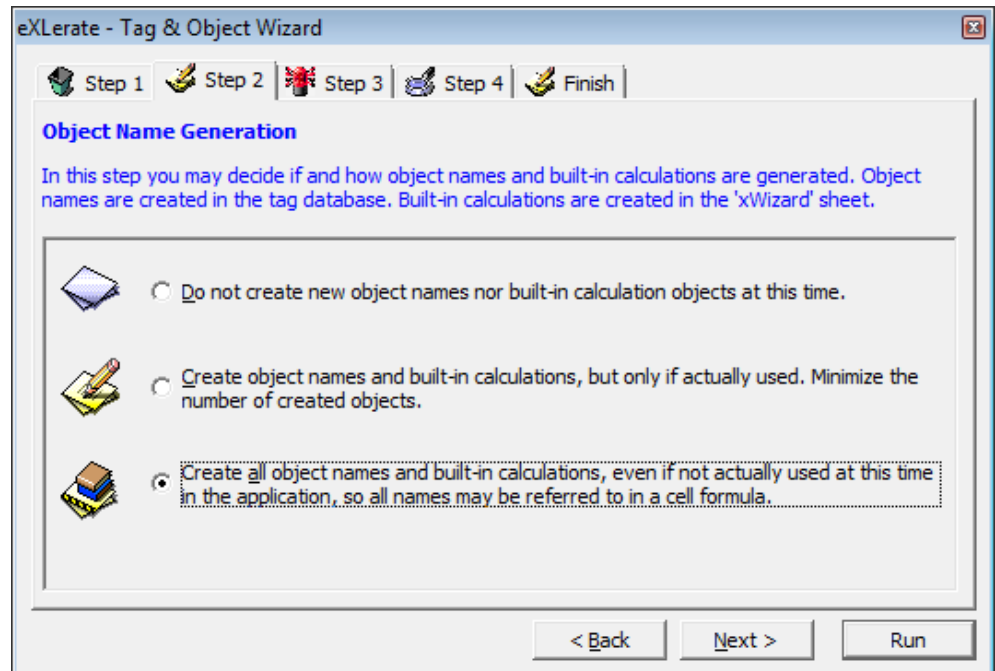
In this step, you define if the results from the previous generation are to be maintained, or to be removed prior to generating new names and built-in calculations.

You should remove previously defined object names and built-in calculations if you have changed the following:



- If you have added, or removed a tag in the tag database;
- If you have modified a field in the tag database that has an associated object name;
- If you have modified periodical calculations (at the *Interval Table* and/or the *tag database*).

In the following step, the generation of new names and objects is defined:



**Figure 3-5: Step 2/4 of the Tag & Object wizard**

You have an option to create all names/calculations, even if the name/object is not used in the application (including report templates), or to only create the in-use object names.

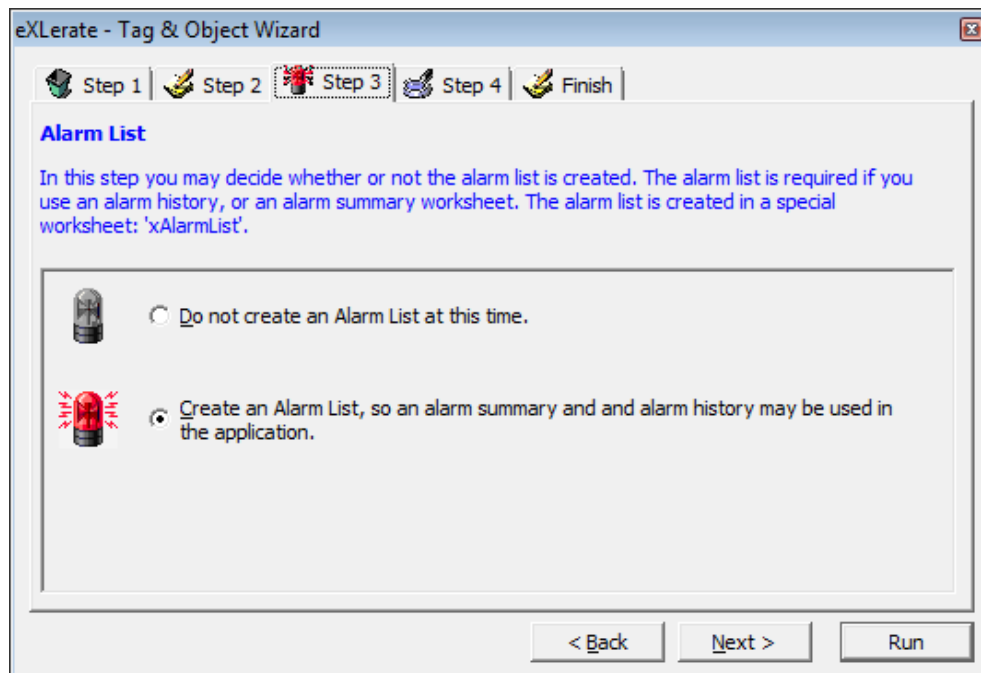
When you are developing an application, you might want to create all object names and built-in calculations, so objects that will be (but aren't yet) referred to in display pages, report templates or calculation sheets, are already defined for you.

When your application is completed as far as using object names, optimize your application by creating only objects that are actually used in an application. This option reduces your application size.

If you have removed existing object names and built-in periodical calculations in the previous step, you want to regenerate these objects in this step.

Only in special situations you might want to remove all object names.

In step 3/4, the alarm list may be generated:



**Figure 3-6: Step 3/4 of the Tag & Object wizard**

The alarm list is a list, which is generated in worksheet: 'xAlarmList'. This alarm list is required when you want to include alarm management in your application.

If you have modified alarm properties of a tag in the tag database, you should include the generation of a new alarm list using this step.

The next step allows you to check for cell faults in your application. It looks as follows:

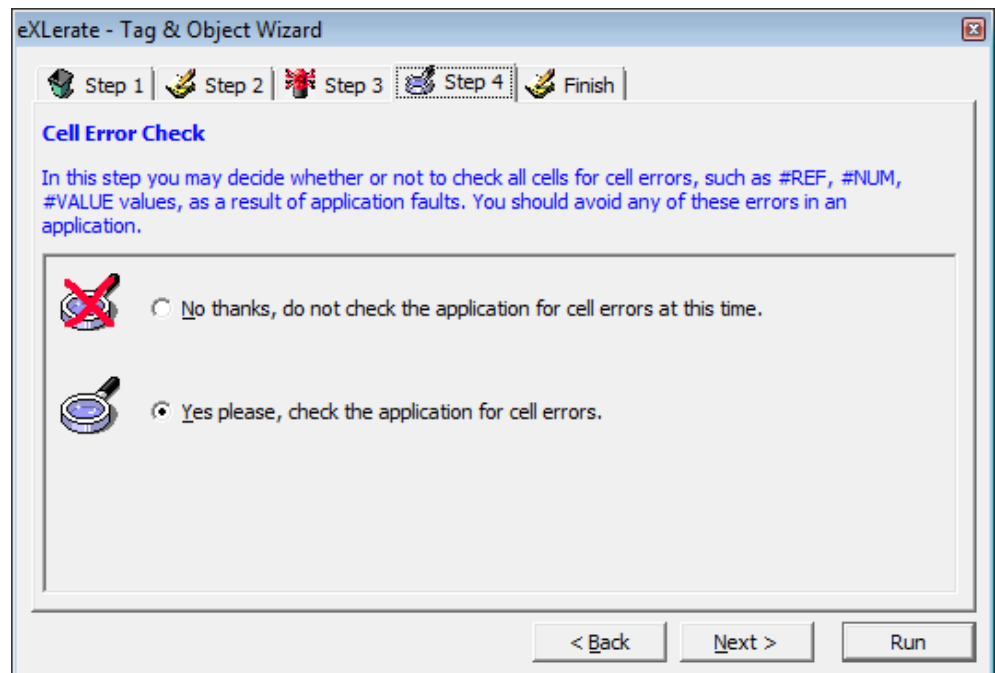


Figure 3-7: Step 4/4 of the Tag & Object wizard

As your application grows, and if any of the previous steps could introduce large changes in your application, checking the application for cell faults is encouraged.

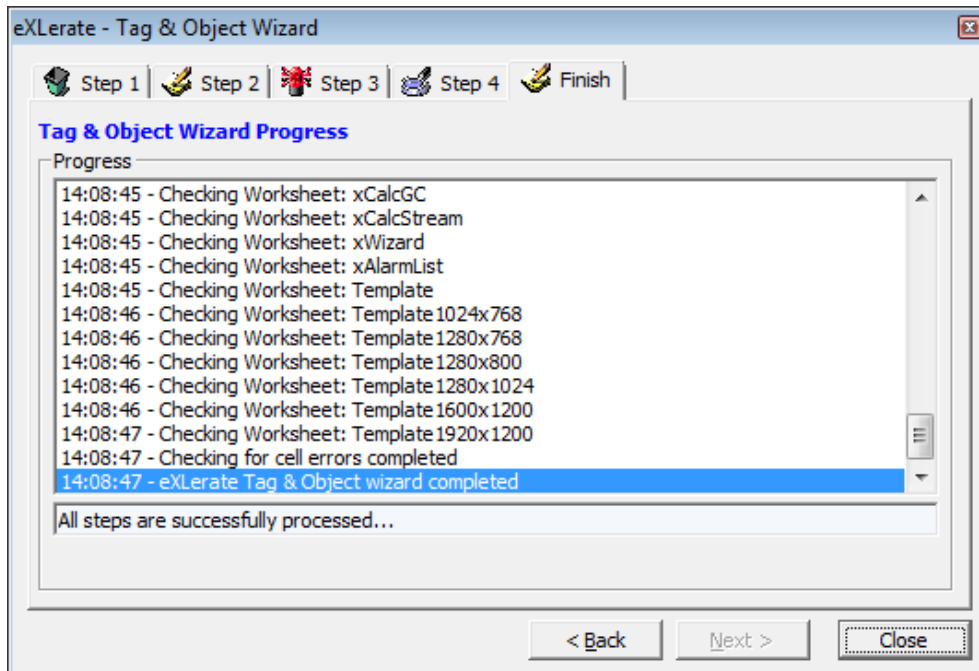


**NOTE:** *An application should not contain any **#NUM**, **#REF**, or **#VALUE** faults, because Excel might become instable when such faults are present in an application.*

In a well-designed and well-programmed application however, such faults can be easily avoided. It is therefore not necessary to check the application every time the Tag & Object wizard is invoked.

When all five steps have been included/excluded as required, 'Run' will actually start the generation process. This may take a short period of time, depending on the size of the application and the speed of your computer.

The progress of the wizard is displayed in a progress-bar. When the wizard has been completed, the generated messages during the generation process may be inspected using the list-box, as follows:



**Figure 3-8: Tag & Object wizard progress**

You can use the scroll-bar at the right of the list-box to see all messages. Using the 'Close'-button, you can inspect your application for the results.

### Calculation wizard

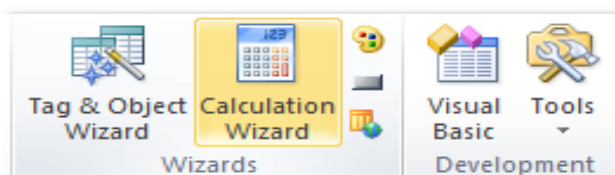


The calculation wizard in **eXlerate** is used to create object names on a calculation worksheet. To learn more about using calculation sheets in an eXlerate application, refer to section 'Calculation sheets' on page 1-16.

The calculation wizard expects the worksheet layout as described in this section.

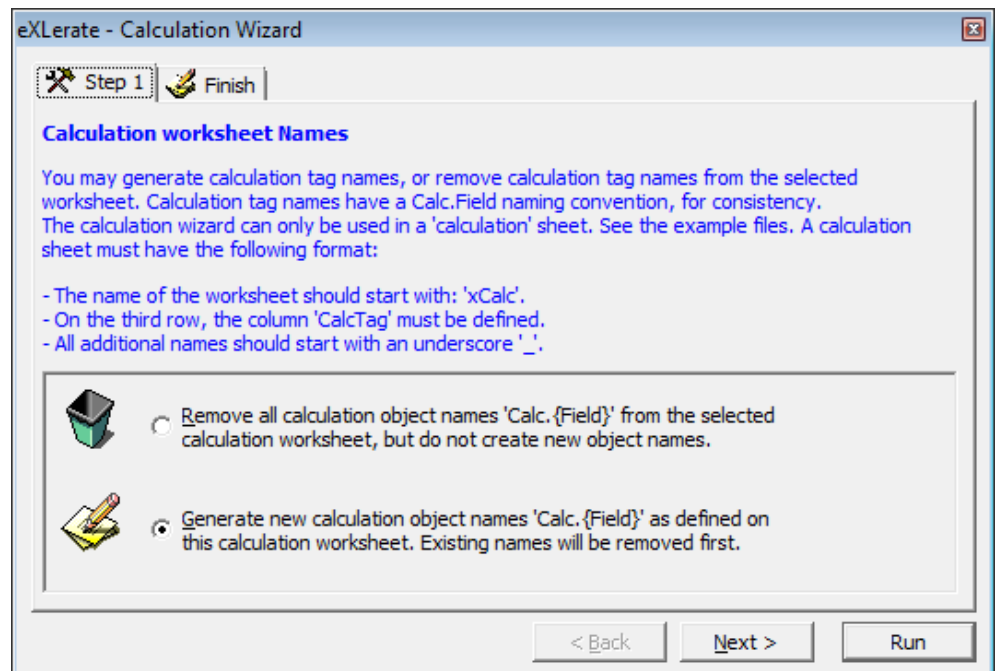


The calculation wizard can only be invoked when the currently selected worksheet is a calculation worksheet. If this is not the case, an error message is displayed. A worksheet is a calculation sheet when the name starts with '**xCalc**', and where on the **third row a header line** is defined containing fields: 'Group', 'CalcTag', 'Description' etc.



**Figure 3-9: Invocation of the Calculation Wizard**

When started from the **eXlerate** ribbon, the following wizard dialog is presented:



**Figure 3-10: Step 1/1 of the Calculation Wizard**

There are two options available:

- Calculation object names may be removed from the current calculation worksheet
- New calculation object names may be created, after existing object names have been removed.

When the active sheet in Excel is not a calculation worksheet containing the layout as required, an error message is displayed when the calculation is invoked with the **Run**-button:

```
09:43:03 - eXlerate Calculation Wizard started
09:43:03 - Error : This worksheet is not a 'calculation sheet'
09:43:03 - eXlerate Calculation Wizard completed
```

When the active sheet is a calculation worksheet, the sheet is processed normally. A calculation worksheet is a worksheet with a layout as given in: 'Figure 1-2: Calculation worksheet layout' on page 1-17.

The progress bar indicates the progress in removal/creation of the object names. After a few seconds, the wizard should be completed without any errors, after which the dialog may be closed to inspect your application for the created results.



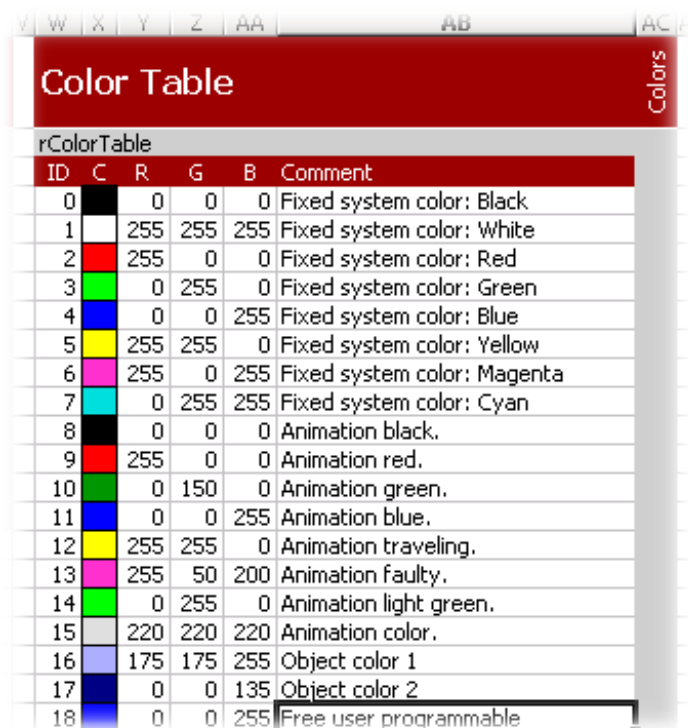
## Color Wizard

The color wizard in **eXlerate** is used to update the color palette in Excel with the colors defined in **eXlerate**.

The current color settings in Excel may be created from a special worksheet table, which is called the *Color Table*. On the other hand, the Color Table may be created from the current color settings in Excel.

## The Color Table

The *Color Table* in **eXlerate** is located in the 'xTables' worksheet, and has the following layout:



ID	C	R	G	B	Comment
0		0	0	0	Fixed system color: Black
1		255	255	255	Fixed system color: White
2		255	0	0	Fixed system color: Red
3		0	255	0	Fixed system color: Green
4		0	0	255	Fixed system color: Blue
5		255	255	0	Fixed system color: Yellow
6		255	0	255	Fixed system color: Magenta
7		0	255	255	Fixed system color: Cyan
8		0	0	0	Animation black.
9		255	0	0	Animation red.
10		0	150	0	Animation green.
11		0	0	255	Animation blue.
12		255	255	0	Animation traveling.
13		255	50	200	Animation faulty.
14		0	255	0	Animation light green.
15		220	220	220	Animation color.
16		175	175	255	Object color 1
17		0	0	135	Object color 2
18		0	0	255	Free user programmable

**Figure 3-11: Color Table layout**

As with most configuration tables in **eXlerate**, the table starts with a table identifier ('rColorTable') which defines which Excel range is associated with the table, a number of field headings (dark red row in the example above), and the data in the table itself below the header line.

The color table has 64 rows, and 6 columns, each with a specific function.

The following columns are defined:

● **ID**

This column contains the ID of the color entry, from 0 thru 63. This value is the color palette index number to be used for shape animations.

● **C**

This column contains a color-formatted cell containing the actual color as defined by the R, G, and B columns.

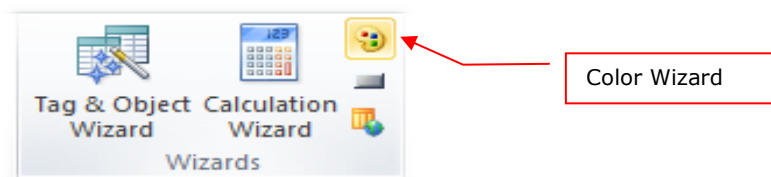
● **R, G, and B**

These columns contain the numerical value of the RGB-color of the entry. In Windows, colors may be expressed in an RGB value, where each component red, green, and blue are represented with a numerical value of 0..255. Pure black is represented as {0,0,0}, while pure white is represented as {255,255,255}.

● **Comment**

Each color in the color table in **eXlerate** is used for a distinct property in the application. For example, pipe symbols with flowing fluids have a color index of 12, and pipe symbols without flowing fluids have a color index of 11. When in an application all pipe colors need to be modified, only one single entry is to be updated, and not the pipe shape objects itself. In this column, the usage of the colors are defined.

The color wizard may be started as follows in **eXlerate**:



**Figure 3-12: Invocation of the Color Wizard**

When this menu option is selected, the following dialog appears:

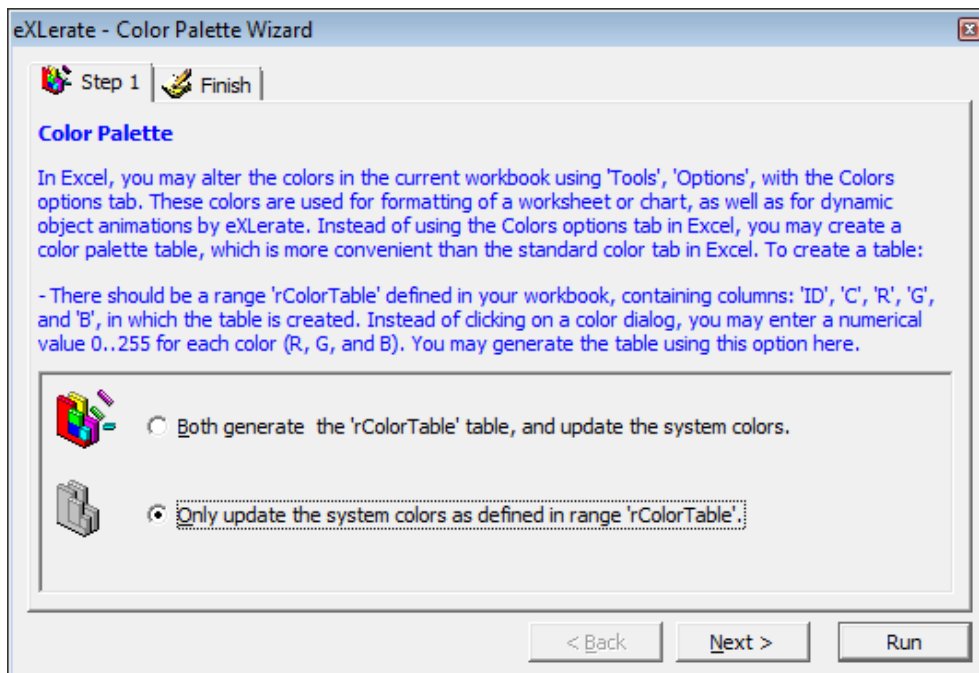


Figure 3-13: Step 1/1 of the Color Wizard

There are two options available:

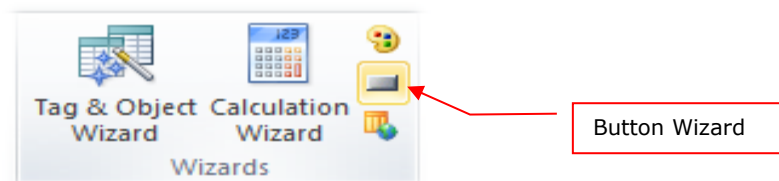
- **Generate a new color palette table, and update system colors**  
This option is able to generate a complete new color palette table, in case your application did not contain a color palette table.
- **Update the system colors only**  
System colors may be updated as well. When the RGB-settings of an entry in the color palette table are empty, the wizard pastes the currently defined color in the table at that entry (which may be thought of as *reading* colors). When the color is defined, and the RGB-settings contain valid numerical values, the currently defined colors are updated from the table (which may be considered as *writing* colors).

The color wizard may be started using the '**Run**'-button:

## Button Wizard

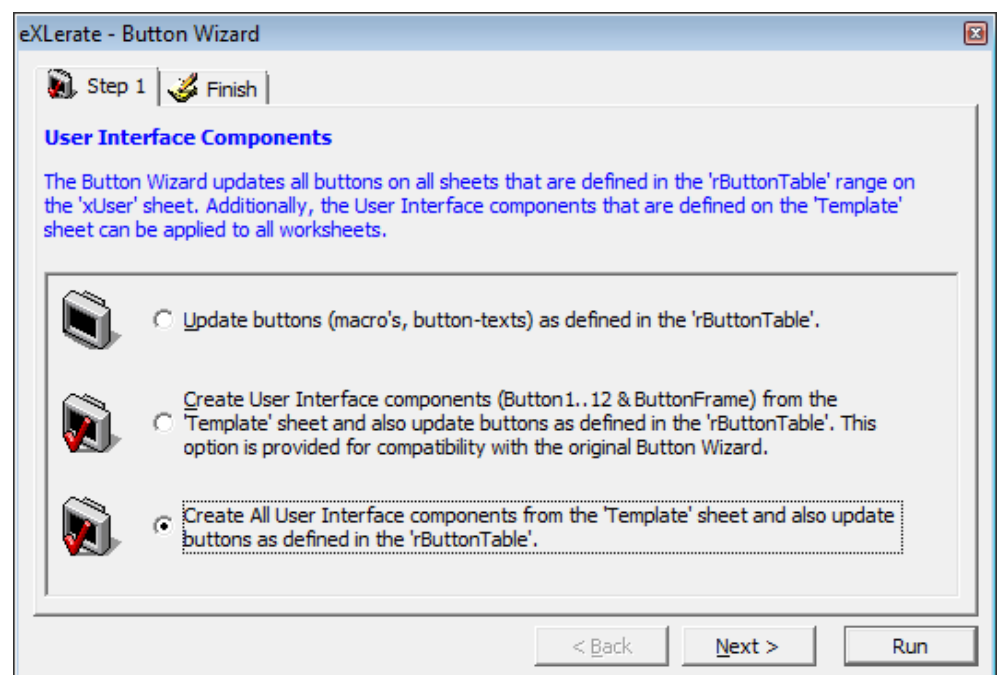
The Button Wizard in eXlerate generates the required functionality for the menu-navigation buttons at the bottom of each display page, as per *Button Table*. The Button Table is located in worksheet 'xTables', and is described in Volume I, 'Menu Navigation', 'The Button Table'.

The Button Wizard should be invoked when the Button Table has been modified, for example when a new display page has been added, or when the menu navigation keys for a display page have been modified. The Button Wizard is started from the eXlerate menu as follows:



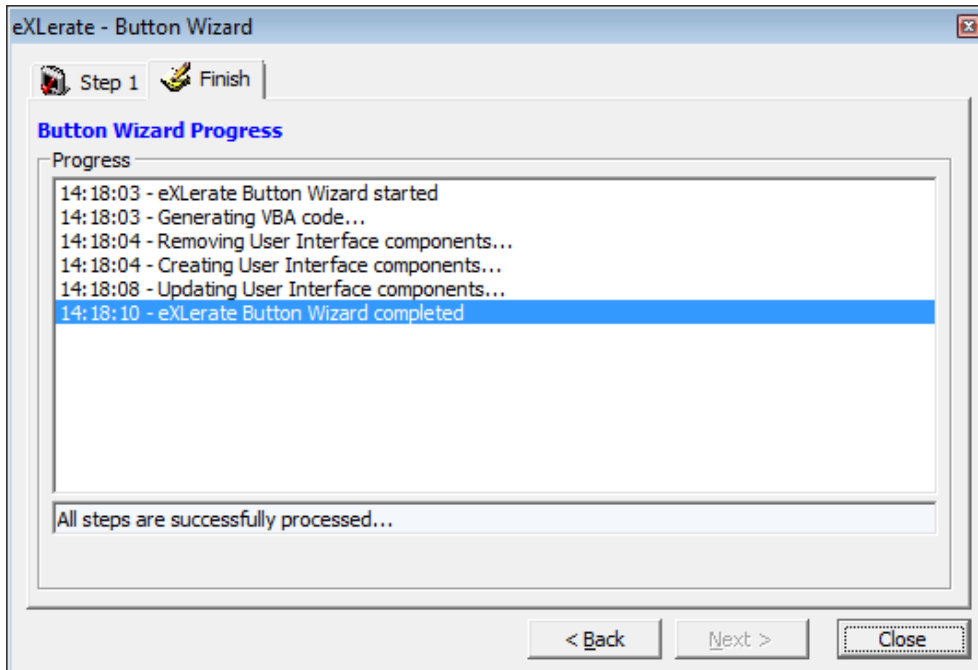
**Figure 3-14: Invocation of the Button Wizard**

When started, the Button Wizard looks as follows:



**Figure 3-15: Step 1/1 of the Button Wizard**

When the Button Wizard is run with '**Run**', the following dialog appears:



**Figure 3-16: Button Wizard completed**

The following actions take place:

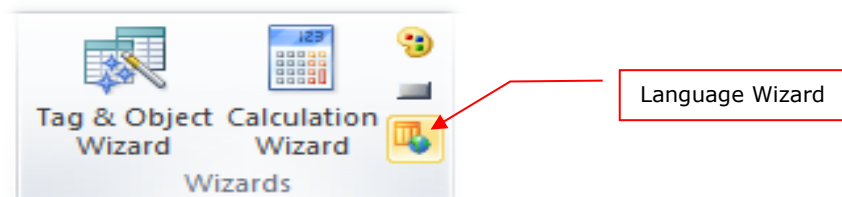
- When at the referred display page as per Button Table, the button objects are defined, its contents are updated according to the Button Table.
- When in the referred display page no button bar is present (yet), the button bar and corresponding frame are copied from the 'Template' worksheet, after which its contents is updated according to the Button Table.
- All required VBA subroutines for the menu navigation is generated as well, in module 'modButtons'. Since this module is erased each time the Button Wizard is invoked, it should not be modified manually. If a user wants to manually modify certain subroutines, these should be moved to another VBA module. In the 'MyProject' project template, 'modAlarms' is used for this purpose.

## Language Wizard

The Language Wizard in **eXlerate** generates the required functionality for the translation of product related texts.

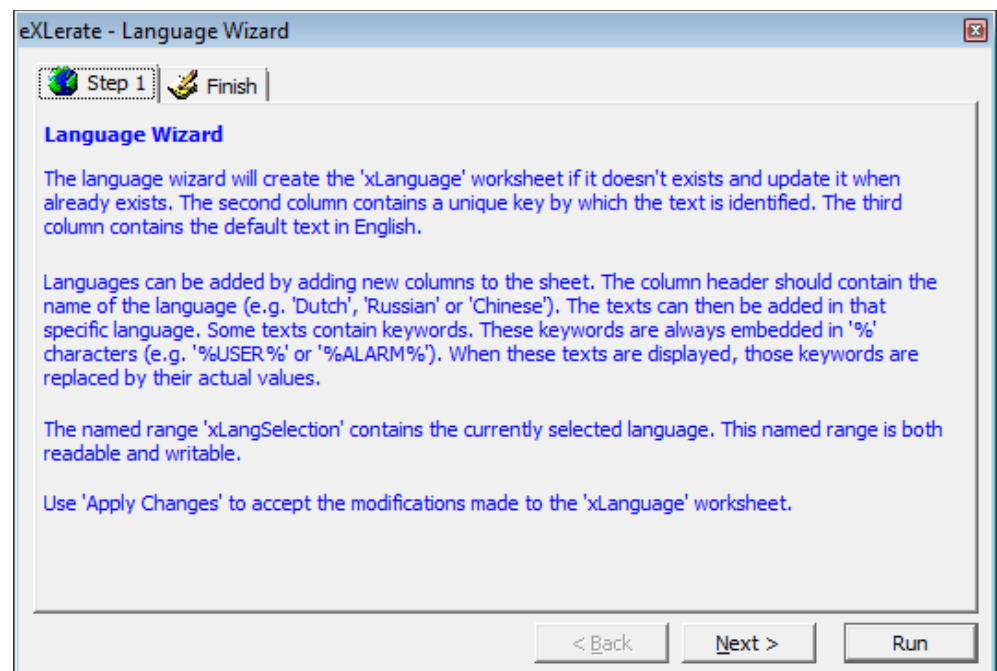
The Language Wizard should be invoked when initially implementing multi-lingual texts or after an upgrade of the product software. When initially running the Language Wizard it will generate a new 'xLanguage' worksheet. Since the product is always being improved, it is possible that new texts become available for translation in later revisions. In that case the Language Wizard should be invoked in order to add the new translatable texts to the 'xLanguage' worksheet. Existing translated texts will always remain unaffected by the Language Wizard.

The Language Wizard is started from the **eXlerate** menu as follows:



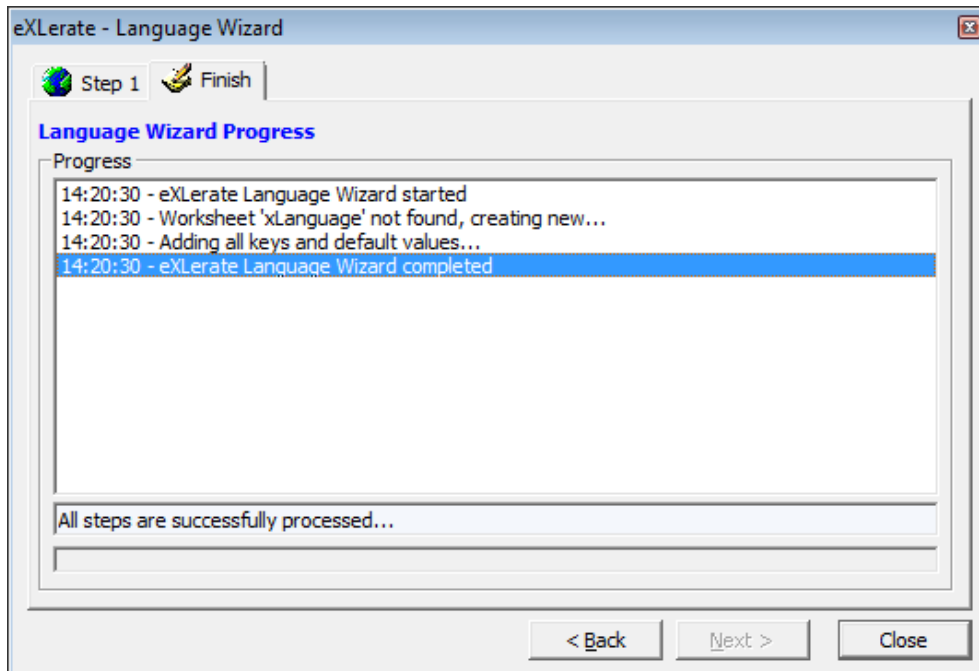
**Figure 3-17: Invocation of the Language Wizard**

When started, the Language Wizard looks as follows:



**Figure 3-18: Step 1/1 of the Language Wizard**

When the Language Wizard is run with '**Run**', the following dialog appears:



**Figure 3-19: Language Wizard completed**

The following actions take place:

- Create new 'xLanguage' worksheet when it doesn't already exist.
- Update 'xLanguage' worksheet when it already exists. Any new translatable texts that have been added to the new revision will also be added to the 'xLanguage' worksheet. Existing texts will always remain unaffected.

## Tools

In an application, there are many tasks with which **eXlerate** assists you. Most Tools have an associated pop-up dialog, while other tasks are directly used from the Tool menu in **eXlerate**.

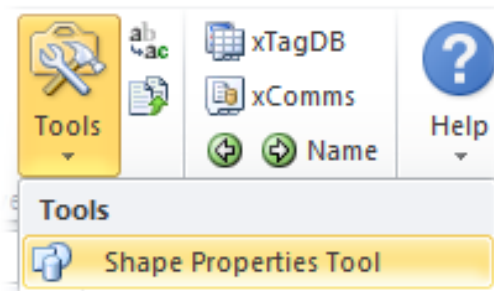
### Shape Property Tool

This tool helps you in finding shapes over various worksheets, and their properties as required for animations. The shape property tool is created to help you in determining the current properties of a shape for animation purposes.

For example, if you want to animate a bar graph, you will need to determine its current size and location on a display page to add animated data to the shape at the Animation Table.

Using the shape property tool, you are able to copy all parameters required for animation from the shape to the table.

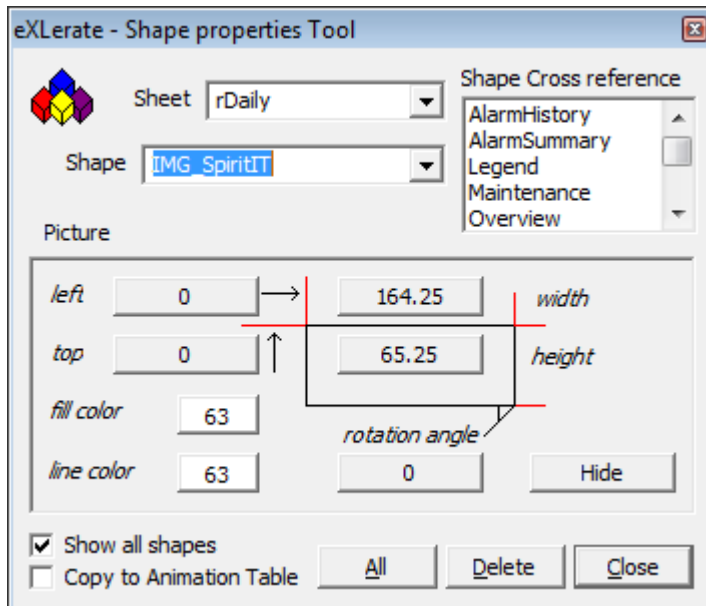
The shape tool is selected from the Tools option in the **eXlerate** ribbon:



**Figure 3-20: Selection of the Shape Tool**

The Shape tool looks as follows:





**Figure 3-21: Shape properties Tool**

On the dialog, a list-box is available with which the animation properties of some or all of the present shapes can be viewed. Each of these properties is presented on a button. The left-position, top-position, shape width, shape height, rotation angle, and the current shape colors are available for copying. Using the '**H**ide'-button, visible shapes can be made invisible, and hidden shapes can be made visible.

When the user clicks the appropriate button, the associated property may be copied to the animation table, at the corresponding column. All properties may be chosen to copy to the table when the '**A**ll'-button is clicked.



The Shape Tool searches for the appropriate location of the shape in the Animation Table. At the right list-box, a cross-reference shows what other display pages contain the selected shape. At the top, the sheet selection list-box presents all worksheet containing shape properties.

With the '**D**ele~~t~~e'-button, the selected shape may be removed from the selected worksheet.



The 'Show All Shapes' check box is used to show only shapes containing non-space names. A shape with one or more spaces in the name is considered as a supporting shape, for example a line (e.g. 'AutoShape 45', or 'Line 67'), whereas shapes for animation purposes are assumed to have a name without space characters (e.g. 'MOV\_123' etc.).

The 'Copy to Animation Table' checkbox is used to either copy the selected (clicked) property directly in the Animation Table, or to copy the value to the clipboard only, where a user may use <Ctrl-V> to retrieve the value from the clipboard. This may be used for indirect usage of the parameter, for example in a formula.

## Name Definition Tool

Prior to Excel 2007, the dialog for managing names in Excel was not so user friendly. Because of this, **eXlerate** supports its own Name Definition Tool. In **eXlerate** you can choose whether to use the standard Excel Names Manager, or the **eXlerate** Name Definition Tool.

The Name Definition tool can be selected from the “Tools” option.

When selected, the following dialog appears:

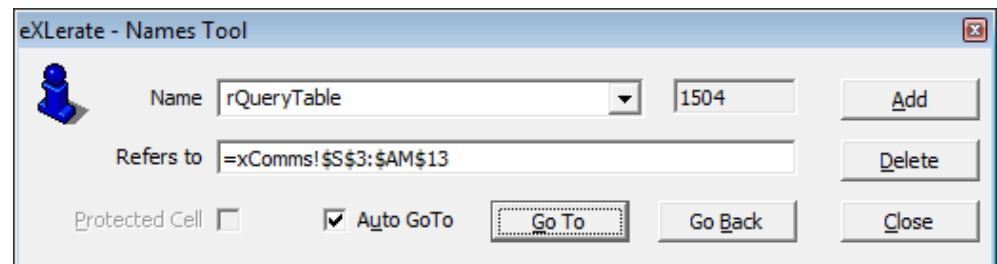


Figure 3-22: Names Tool

The names list-box may be opened, which looks as follows:

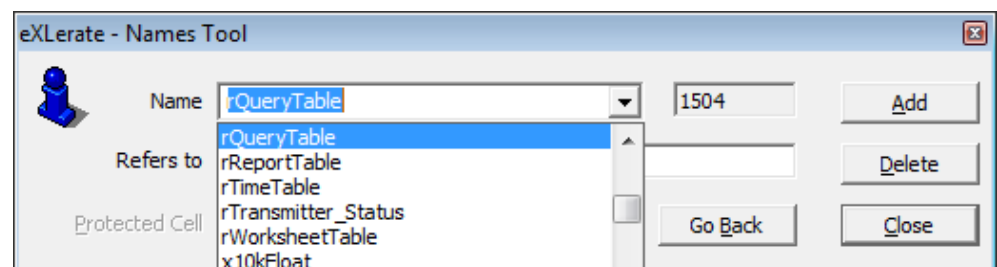
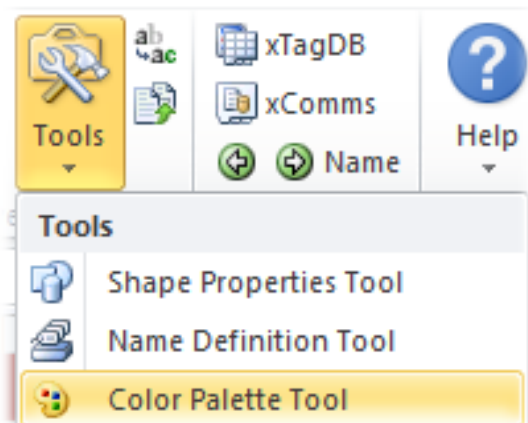


Figure 3-23: Names Tool with opened name list-box

Using the ‘**Go To**’- and ‘**Go Back**’-buttons of this Names Tool you are able to jump to the location of the reference, and to return to the last position. If you still prefer the Excel names dialog, please feel free to do so. This tool is only optional!

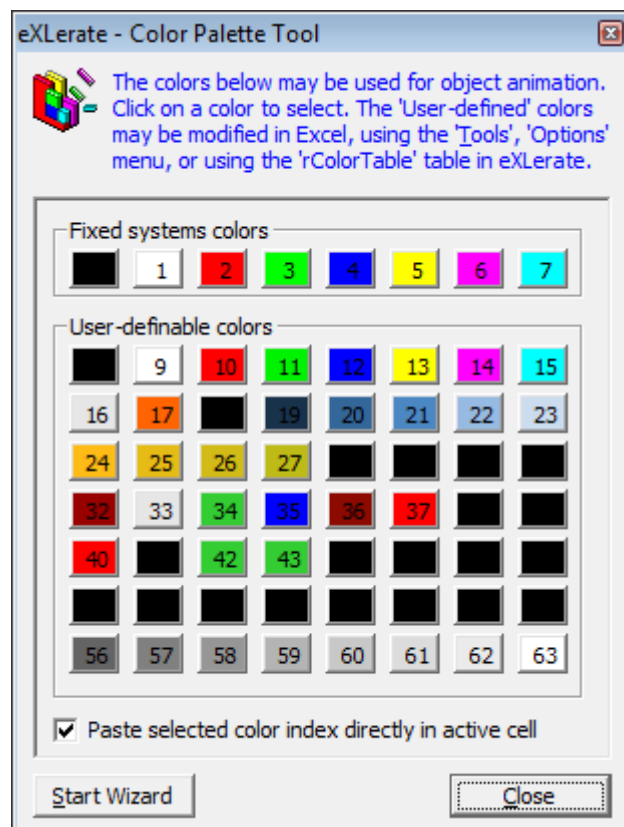
## Color Palette Tool (Ctrl+L)

The Color Palette Tool is a tool showing you the relation between a color palette index number, and the actual color for shape animations. It is selected using the <Ctrl-L> shortcut key directly, or selected from the **eXlerate** Tools menu, as follows:



**Figure 3-24: Selection of the Color Palette Tool**

The following dialog appears:



**Figure 3-25: Color Palette Tool**

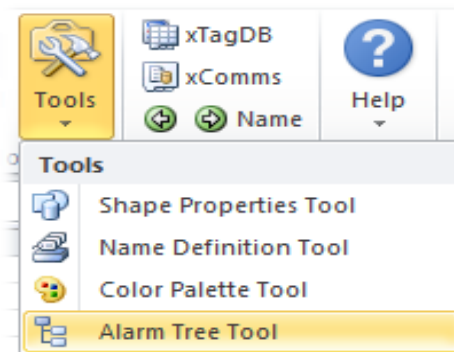
When a color button is clicked, its number is either copied to the clipboard, or copied to the active worksheet cell. The actual action depends on the value of the checkbox on the dialog.

From this dialog, the Color Wizard may be also started.

### **Alarm Tree Tool (Ctrl+M)**

This tool allows the user to browse through the alarm directory tree in the application. The alarm directory tree is defined in the tag database, where a tag is associated with an alarm group, and the alarm parent-child definition table, as defined in range `'rAlarmGroupsTable'` on the `'xTables'` worksheet.

To start the tool, press <Ctrl-M>, or use the eXlerate Tools sub-menu:



**Figure 3-26: Starting the Alarm directory tree tool**

When selected, a dialog as below appears, with which the user can browse, and verify that the currently defined alarm group directory tree has the required parent-/child relation as set up in the Alarm group table.

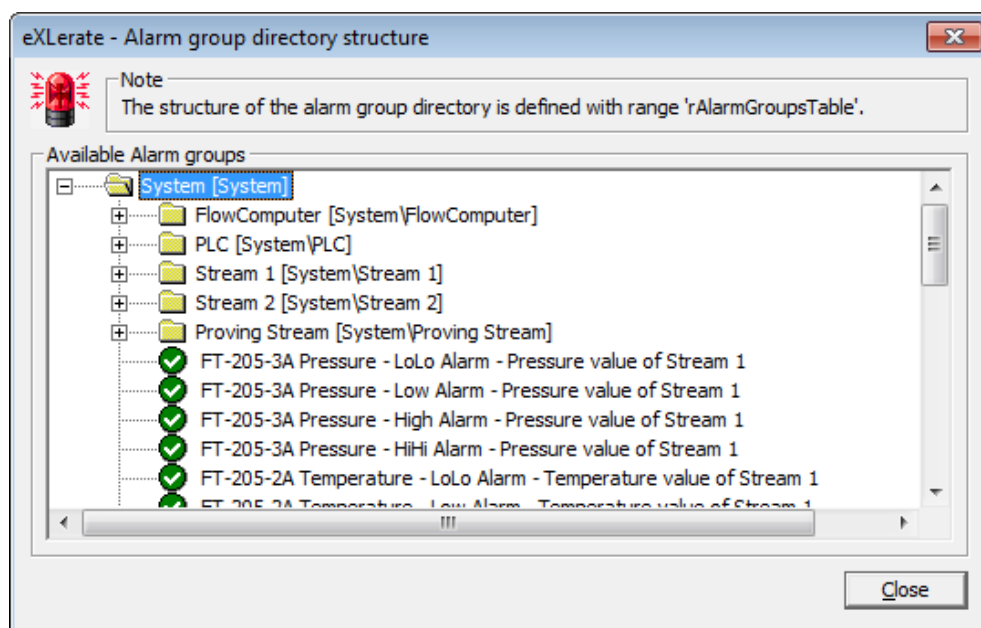


Figure 3-27: Alarm Directory Tool

## Generate Report

The "Generate Report" option displays the list of configured reports which may then be printed or previewed.

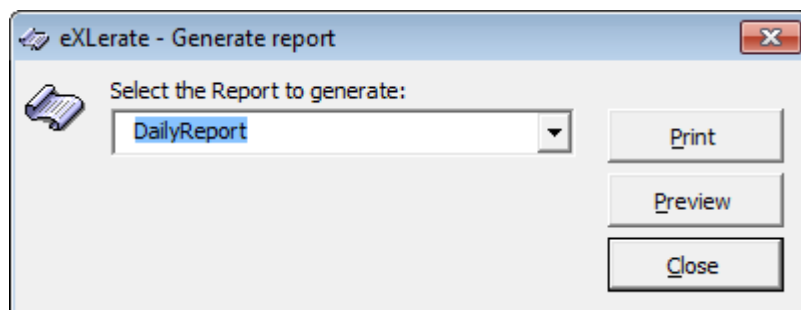


Figure 3-28: Generate Report

Reporting is discussed in detail in the **eXlerate Reference Manual Volume 1**.

## Generate HTML

The "Generate HTML" option generates the HTML pages for web-server support. You can find the output files in "C:\XLRX\HTML".

HTML Generation is discussed in detail in the **eXlerate Reference Manual Volume 1**.

### **Browse OPC Servers**

The “Browse OPC Servers” option allows you to browse OPC Servers. This dialog allows you to select OPC Item ID’s and paste them into the application.

OPC and communications are discussed in greater detail in the **eXlerate Reference Manual Volume 1**.

### **Communications Options**

This option opens up the Communications Options dialog. You can use this dialog to enable/disable diagnostic logging of the communication drivers.

The communications options are discussed in greater detail in the **eXlerate Reference Manual Volume 1**.

### **Show Control Center**

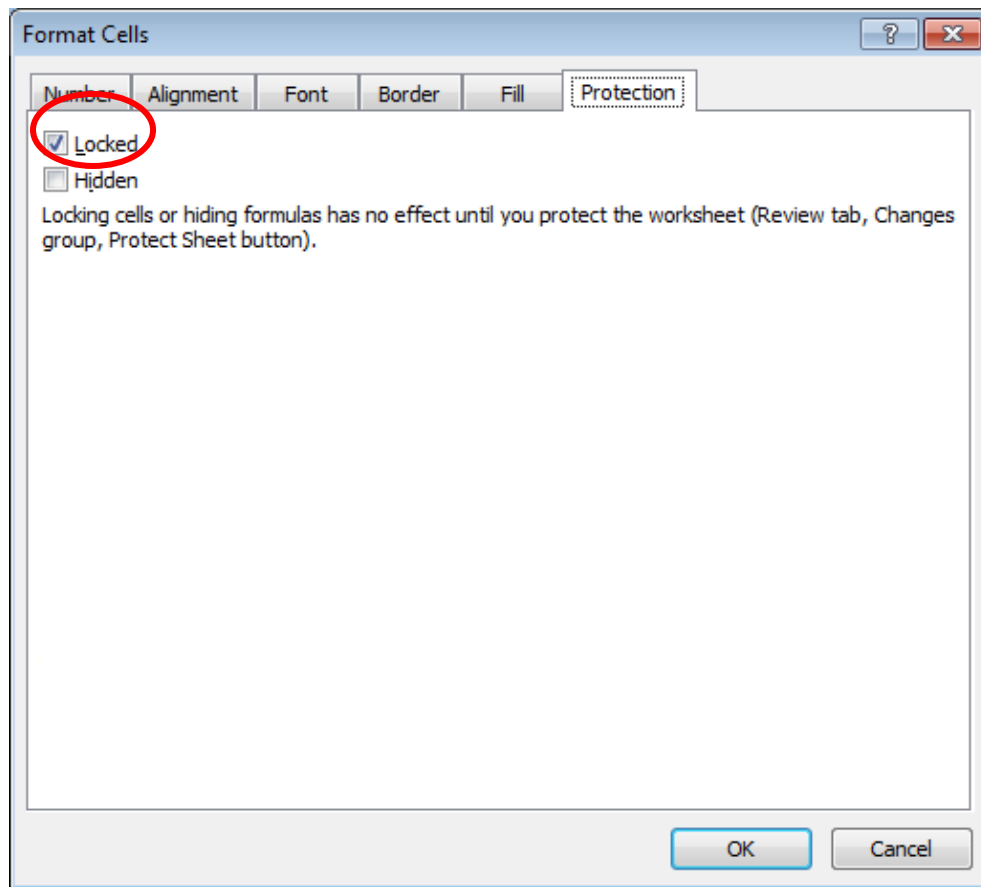
This option Shows the Control Center. The shortcuts, users and global settings may be edited from within the Control Center which is discussed in **eXlerate Reference Manual Volume 1**.

### **Unprotected cells marker (Ctrl+U)**

Worksheet cells play a vital role in an application. A user may be enabled to edit the value of a worksheet cell, depending on the settings of the *Worksheet Table*. Unfortunately it is not very straightforward to see which cells are protected in Excel, and which cells are unprotected. You may select a single cell, and with the tool as displayed at the left, from the **eXlerate** ribbon you can view the protection status of this cell.



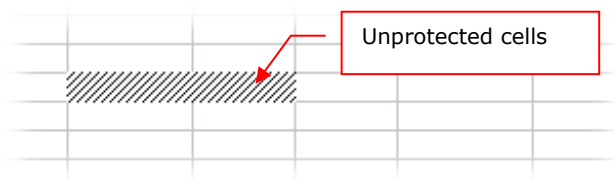
Alternatively, the cell-formatting dialog in Excel displays this status as well:



**Figure 3-29: Cell formatting dialog showing the protection status**

Using the '**Mark Unprotected cells**' (Ctrl+U) option from the **eXlerate** Tools sub-menu, the protection status of *all* worksheet cells is visualized rather than the individual status.

When activated, each unprotected cell is marked as follows:

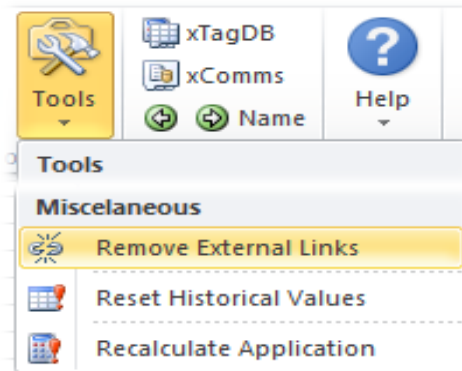


**Figure 3-30: Marked unprotected cells**

The marking can be un-done using the '**Unmark Unprotected cells**' option from the **eXlerate** tools sub-menu, or using the <Shift-Ctrl-U> shortcut key.

### ***Remove External links***

When cells or entire worksheets are copied between workbooks, Excel might link the copied cells or worksheets to the original workbook. These links can be removed using this option:



**Figure 3-31: Removing external links**

Linked applications should be avoided when possible. External links may be also inspected or manually removed using the Excel 'Edit Links' option from the 'Data' ribbon.

### ***Reset Historical values***

This option from the eXLerate Tools sub-menu resets the historical values of the current application. All retentive intermediate results that are maintained and updated in the registry are reset.

Remember, these intermediate values relate to period dependent moving averages, weighted averages, and latch registers.

This tool does not affect trending and log files, since these files exist on disk, and have another mechanism of monitoring their size.

### ***Recalculate Application (Ctrl+R)***

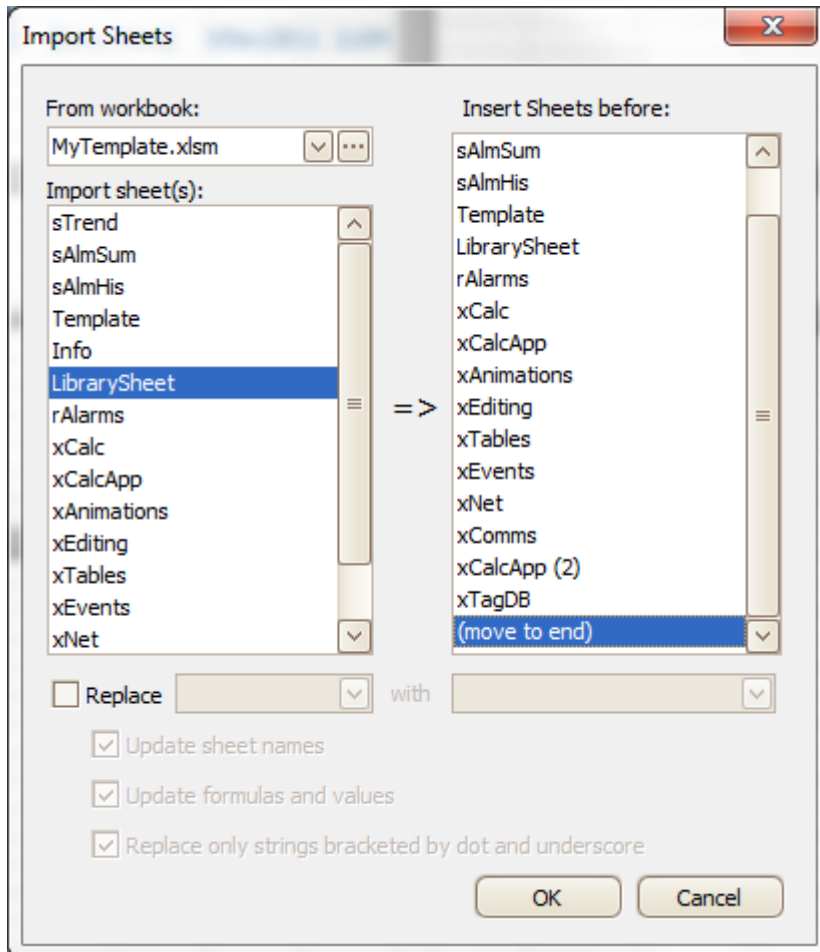
When you modify formulas you may want to re-calculate the application to apply the changes. Many worksheet functions in eXLerate take a trigger-argument of some kind. This trigger argument ensures that a function is called periodically, but not constantly. Some triggers such as 'xNow.Time' are executed every second, while others like 'xAutoRecalc' are updated only at startup or when starting/stopping communications.

Recalculate Application ensures that all functions which are using the 'xAutoRecalc' trigger are re-calculated.



## Import Sheets

This option displays the 'Import Sheets' dialog which allows you to import sheets into your application.



**Figure 3-32: Import Sheets**

At the 'From workbook' option, select the workbook from which you wish to import the sheets. Use the '...' button to load a workbook if it isn't already loaded. Select the sheet you wish to import. Hold the 'Ctrl' or 'Shift' key if you want to select multiple sheets.

## Advanced Replace

Advanced Replace offers the ability to not only find & replace cell values & formulas, but also names, styles, objects (e.g. shapes) and macro names:

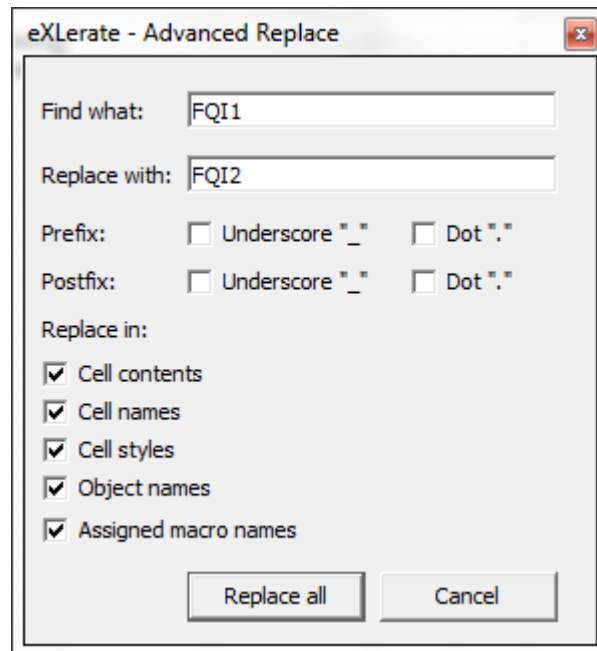


Figure 3-33: Advanced Replace



# Chapter 4 - Controls

## Introduction



**eXlerate** supports a set of controls for alarm management and trending purposes. These controls share a lot of functionality, which is described in this chapter.

## Control Overview

Viewing trend-data in **eXlerate** can be done by inserting Controls on worksheets or VBA forms. These controls can then be linked to each other in a modular fashion. Four types of controls are available which are described in more detail in the following chapters:

Control	Description
exAlarmSummary	Control which displays the alarm summary.
exTrendChart	Chart control which shows the trend-data and allows users to navigate in time. Supports a data cursor for accurately viewing the trend value.
exTrendPenSelector	Control for viewing and modifying the visible pens of a trend chart control.
exTrendNavigator	Optional control which can be used to quickly navigate through large amounts of trend data.
exListView	Generic list-view control.

Table 4-1: Control Overview

## Inserting Controls

To insert a control on a sheet, click one the following buttons on the left toolbar:

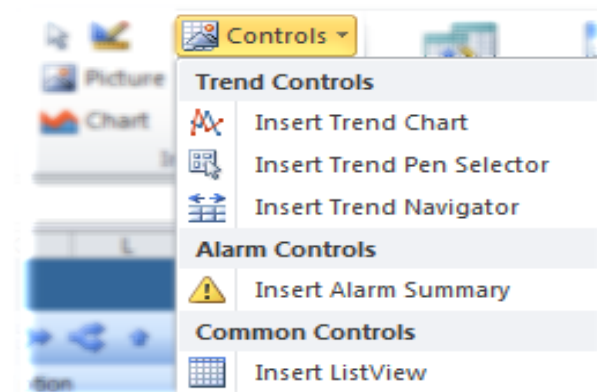


Figure 4-1: Inserting Controls on Worksheets

To move or resize a Control, click the design-mode icon at the top of "Insert" section. When finished click the button again to exit design-mode.

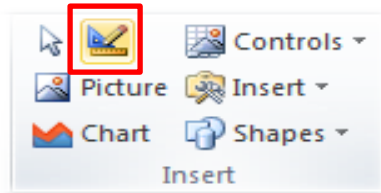


Figure 4-2: Design mode

To insert a trend control on a VBA form, enable the Toolbox and select one of the controls:

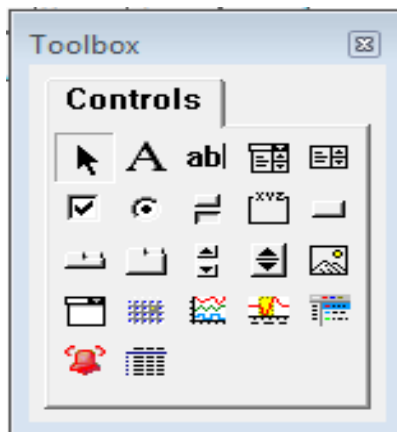


Figure 4-3: Inserting on VBA forms

## Modifying Controls

Controls can be customized to a very high extend. The properties of a control can be changed from the Properties Dialog. This dialog is accessible by clicking the icon in the right-top of the control. The properties button is only accessible when the design-mode is turned off.



Figure 4-4: Properties button

In most Controls it is also possible to double-click the control itself to open the Properties Dialog. For instance, double-clicking the left-axis on an exTrendChart control will open the Properties Dialog and select the "Left axis" tab.

## Control Properties

### Backgrounds

Backgrounds are used when filling a region with a certain background. A background can consist of a single color, gradient colors, a pattern or a picture. The type can be changed by selecting one of the Type options:

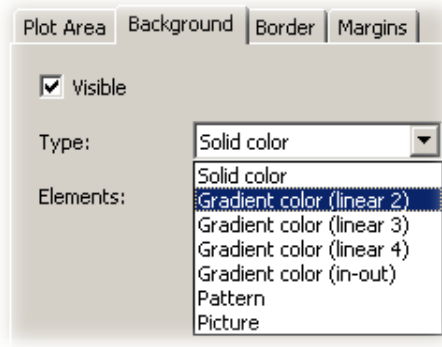


Figure 4-5: Background types

### Solid Color

When "Solid color" is selected the following options become available:

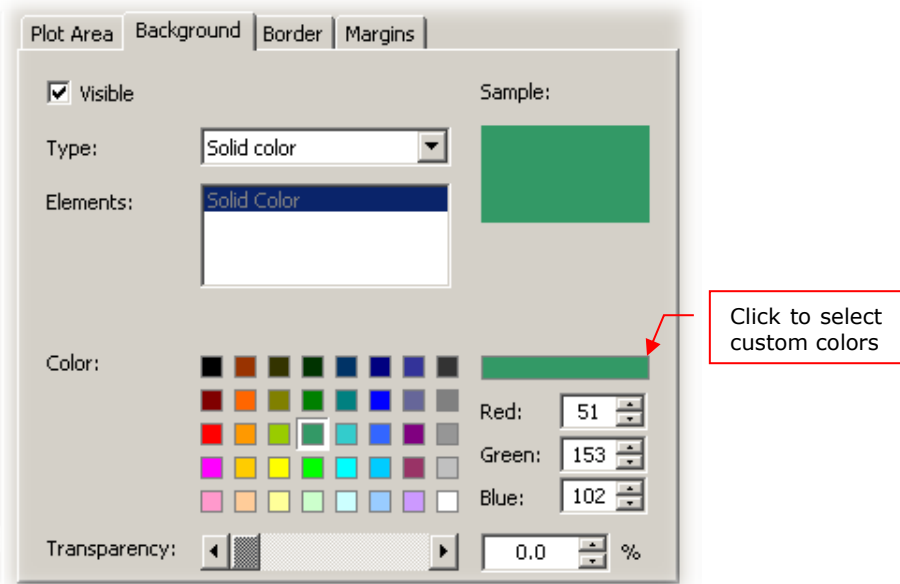
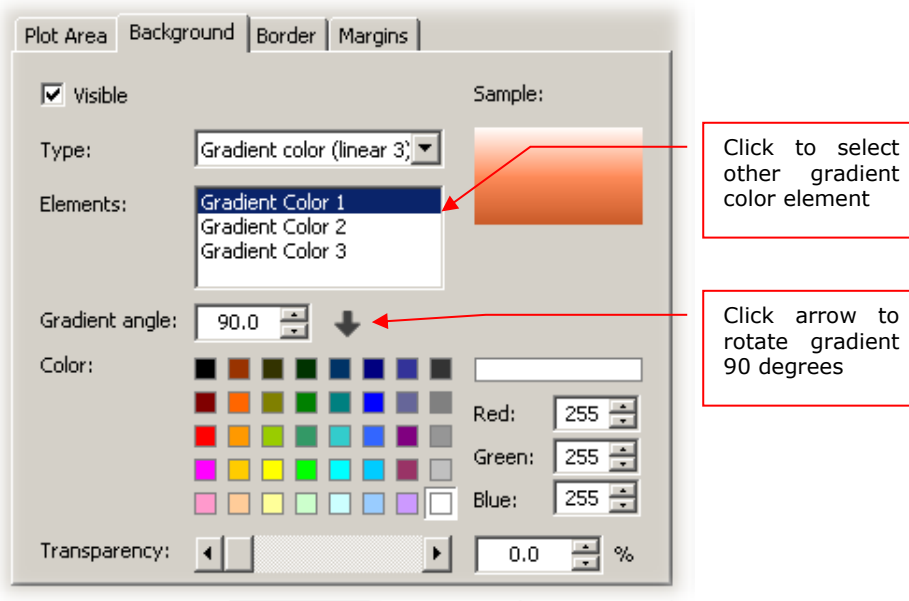


Figure 4-6: Solid background color

The standard color table can be used to quickly select a color. The "Red", "Green" and "Blue" options can be used to change the individual RGB values of the color. Click on the color-bar to open the default Windows Color Picker. When the transparency is changed, the sample shows the results on a white background.

### Gradient colors

Gradient colors may be used to create advanced graphical effects. A gradient always consists of 2 or more colors. Each of these colors can be selected in the "Elements" list:



**Figure 4-7: Gradient background colors**

The "Gradient angle" may be used to change the direction of the color flow. By default the angle is 0 which causes a color to flow from left to right. To change the color flow from top to bottom, change the angle to 90.

## Pattern

Patterns consist of both a foreground and background color. The pattern appearance may be changed by selecting the "Pattern style" option.

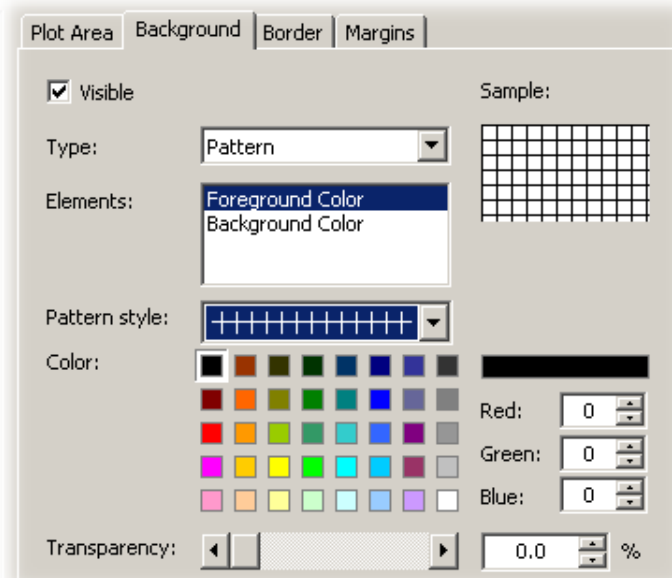
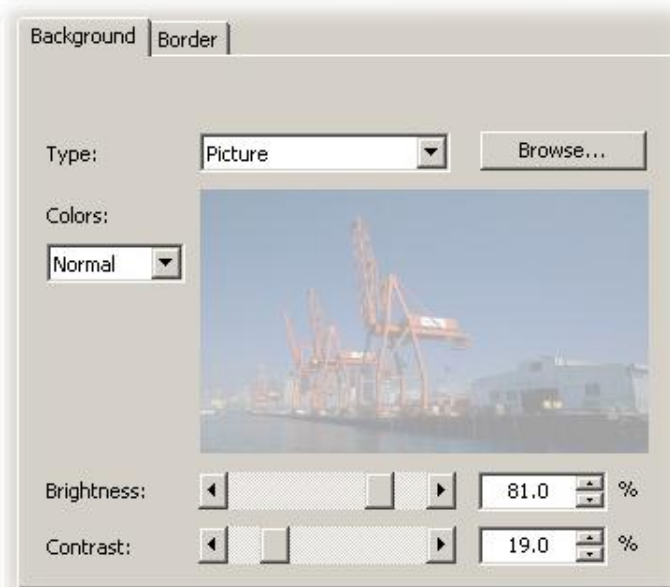


Figure 4-8: Background pattern



## Picture

When "Picture" is selected the following options become accessible:



**Figure 4-9: Background picture**

Click the 'Browse...' button to select a picture from disk. The following picture formats are supported: BMP/GIF/JPG and ICO. Pictures are internally always stored as JPG which reduces the file-size dramatically.

The "Colors" option may be used to draw the picture in Grayscale rather than full-color.

The "Brightness" and "Contrast" options may be used to lighten or darken the picture. If for instance a picture is used as the background of the chart- or plot-area the trend-lines may become unreadable. In this case the picture can be made lighter by increasing the brightness and decreasing the contrast. Generally speaking the brightness should be increased the same amount as the contrast is decreased. The example above shows a picture with equally increased/decreased brightness and contrast settings.

## Borders & Lines

Most properties that support a background also support a border. A border consists of a weight, a style, a color and transparency:

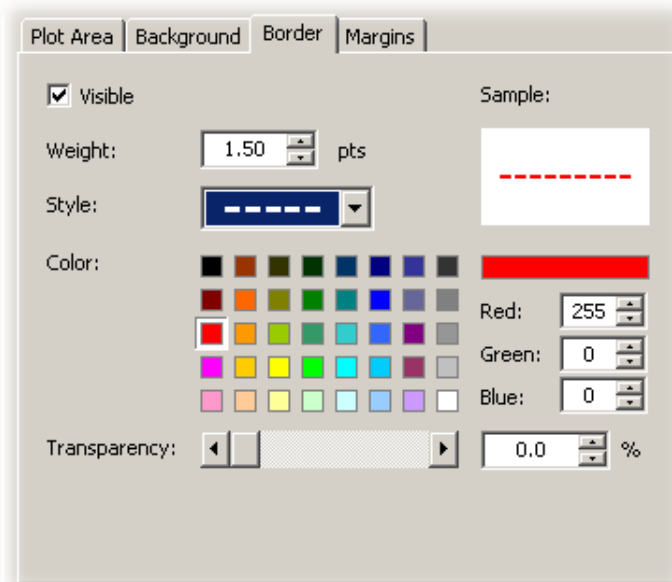


Figure 4-10: Borders & Lines

## Margins

Margins are used for both the Trend Chart Plot Area and labels:

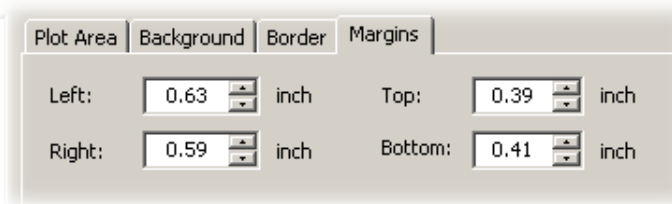


Figure 4-11: Margins

## Fonts

The following font properties are supported:

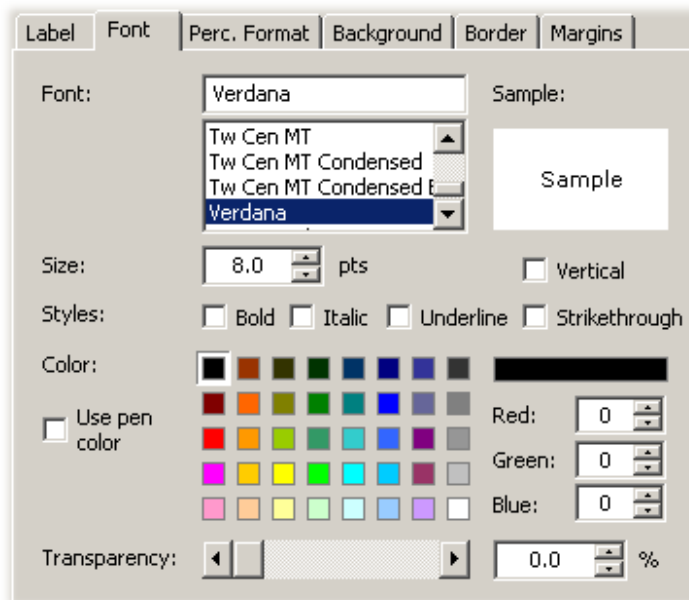


Figure 4-12: Fonts

## Formats

Formats can be used to format numbers and/or date-time values. The example below shows the supported formats of a date-time value. The keyword '{CR}' can be used to create a carriage-return or line-break as shown below:

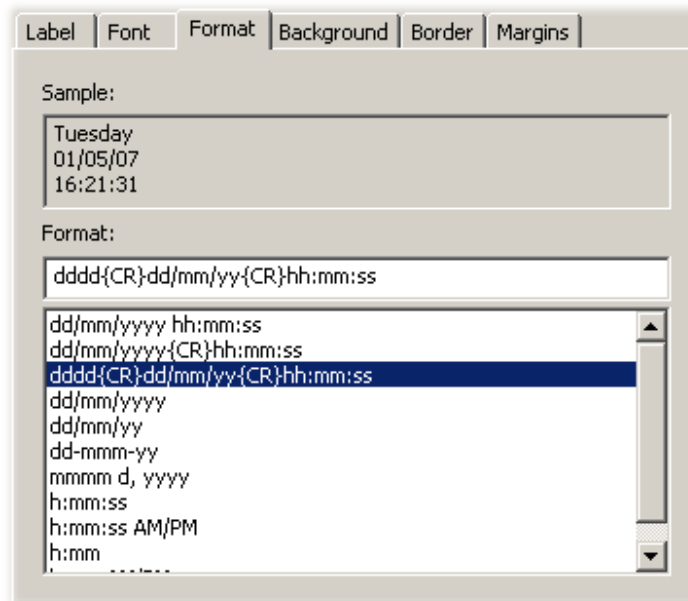
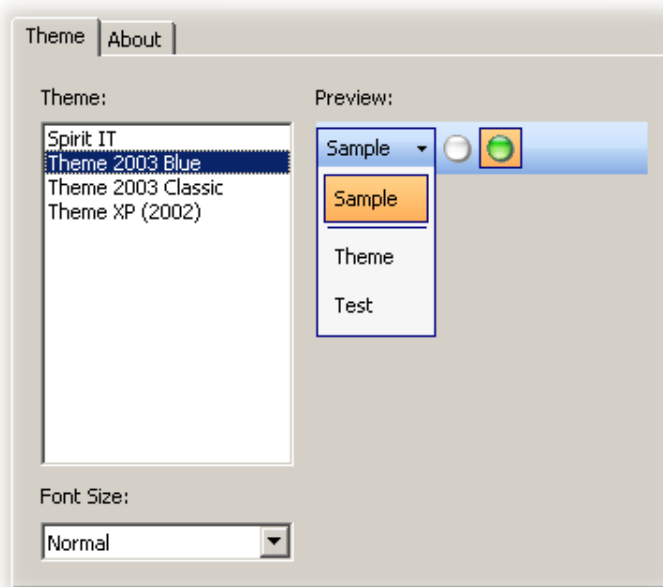


Figure 4-13: Formats

## Themes

eXlerate supports several themes as shown below:



**Figure 4-14: Themes**

The theme is used for drawing the toolbar and other controls such as list controls. Three font-sizes are supported: Normal (8 pt), Large (10 pt) and Extra Large (12 pt).

## Templates

When a control has been configured, its settings may be saved to a template file for re-use purposes.

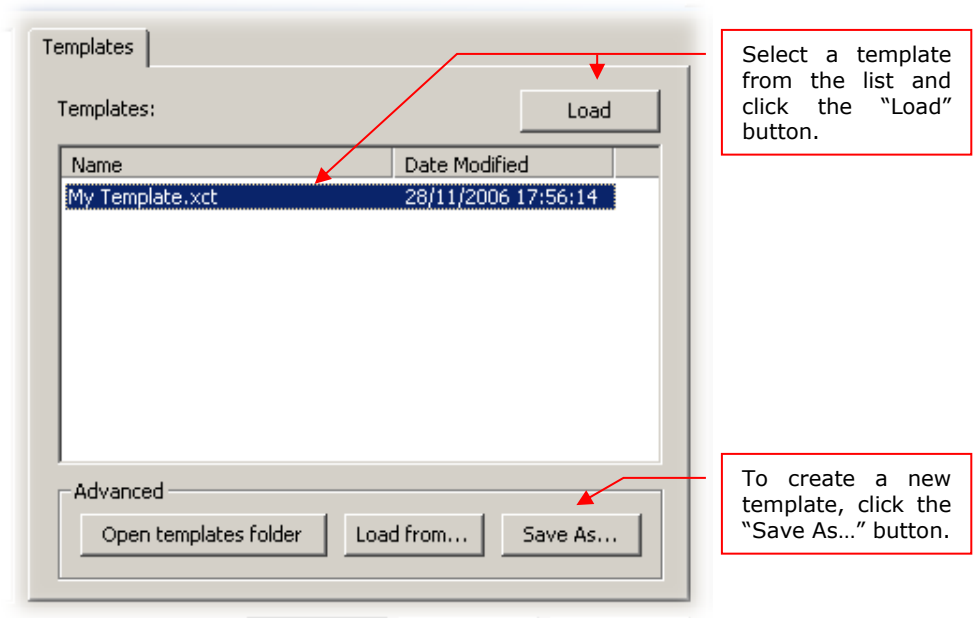


Figure 4-15: Templates

## List Columns

The list shows all the columns of the list control. The properties are shown as columns of the list. Which columns are visible and the width of the columns can be configured in the properties dialog:

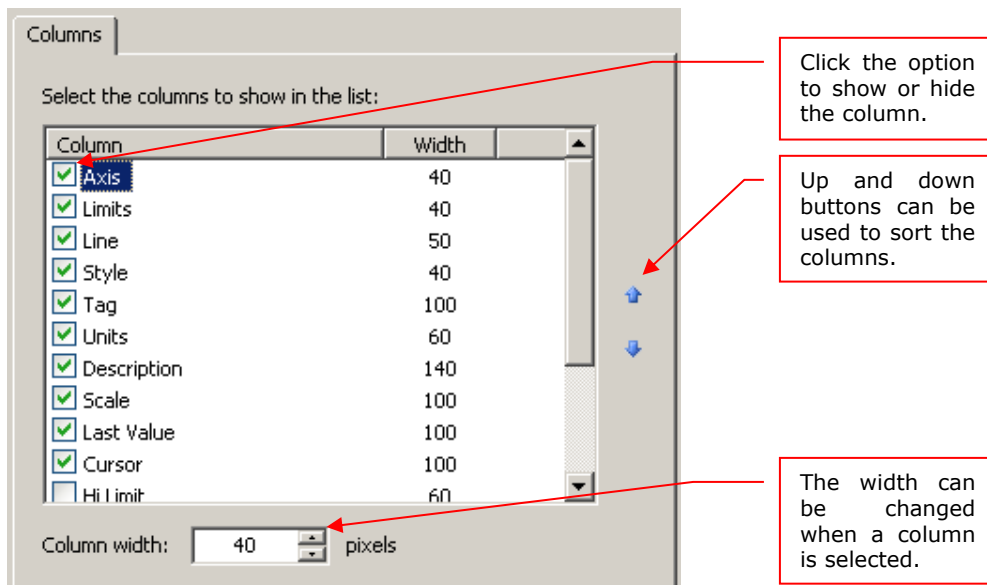


Figure 4-16: List Columns

# Chapter 5 - Alarm management

## Introduction



Alarm management in **eXlerate** has already been introduced in the tutorial chapter, in the Application Reference Manual, and in the description of the tag database, in the section on alarming related fields.

You have perhaps noticed at these introductory pages that alarm functionality may be automatically added to your project when you have alarm fields defined in the tag database.

## Alarm page types

There is a distinct difference between the *Alarm History*, and the *Alarm Summary*, which are the two alarm entities maintained by **eXlerate**. The alarm history is a log of alarm events that have occurred over a period over time, while the alarm summary contains an actual list of tags with defined alarm properties.

Both types of lists are available in **eXlerate** as worksheets. In addition, alarm event messages are sent to the system event logger of the **eXlerate** Control Center where such messages may be printed to the alarm printer, stored on disk, and viewed in the event log.

Alarms may be hierarchically grouped, like a directory structure. There is functionality to acknowledge an alarm or an alarm group, or all alarms.

There may be various alarms simultaneously defined for a single tag. Alarm limits may be entered as constants at the tag database; **eXlerate** will automatically make these alarms limits retentive.

Best of all is that this alarm functionality is automatically available in your project.

The way the alarm summary and alarm history pages look like may be adapted to your own requirements.

You may add other alarm functionality via the VBA and worksheet interface of Excel, with one of the available procedures and alarming options. For example during startup you are able to define a delay after which alarming becomes active, or use different alarm settings at startup.



## Alarm components

Alarm management in eXLerate is facilitated by various items:

- **The tag database**

Since alarm properties of a tag are defined at the tag database, it is the tag database that plays a vital role in alarm configuration. When alarms have been defined at the tag database, the *Tag & Object wizard* should be activated to generate the actual alarm summary list.

- **Alarm Group Table**

Hierarchical alarm groups are defined in a simple table, the *Alarm Group Table*. This table is located in your project workbook, in worksheet 'xTables'.

- **Alarm Summary Control**

This is a control which displays a list of alarms and optionally the alarm tree. Alarms can be acknowledged, suppressed and edited from within the control.

- **Worksheet: AlarmHistory**

In this worksheet, a history of the latest 40 alarm messages is presented. The alarm history as sent to the system event logger is also available to the user, but this history contains other event messages as well, such as process startup- and shutdown messages.

- **VBA and worksheet functions**

There are various functions that can be called from VBA, with which alarms can be groups-wise acknowledged. Other functions can be called to define certain alarm options.

- **Alarm dead-band**

Additional information about the dead-band which can be configured in the tag database.

- **Other alarm options**

Other options such as auto-acknowledgement and alarm blocking.

## Tag database

The tag database plays a vital role in alarm definitions for a tag, because an alarm for a tag is configured at the tag database.



The fields at the tag database which are used for alarming are already described in section '*Alarming related fields*' of the Application Reference Manual.

## Alarm Group Table

Alarms may be logically grouped together in an *Alarm Group*. User acknowledgement of alarms may be done group-wise.

The internal hierarchy of existing alarm groups is defined in the *Alarm Group Table*, where a 'Parent-Child' relation is defined between alarm groups.

The *Alarm Group Table* is a standard eXlerate table located in the 'xTables' worksheet, and looks as follows:

Alarm group Table	
rAlarmGroups	
Parent	Child
System	Line1
System	Line2
System	GC
System	PLC
System	Calc

**Figure 5-1: Alarm Group Table**

The *Alarm Group Table* begins with the range name: 'rAlarmGroups', followed by a two-column table. At the left hand side of the table, the *Parent* is defined. At the right hand side of the table, the *Child* is defined.

When this table is not present, no hierarchy between alarm groups is defined.

## Alarm Summary Control

### Introduction

The Alarm Summary Control can be used to visualize and modify alarms. It consists of three parts, an alarm group tree, a list showing the actual alarms and a toolbar:

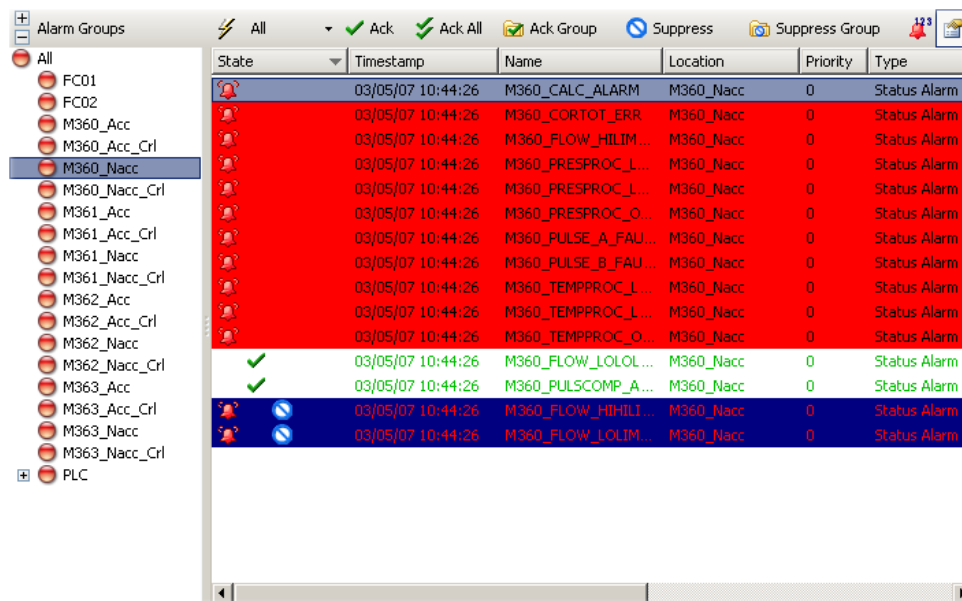



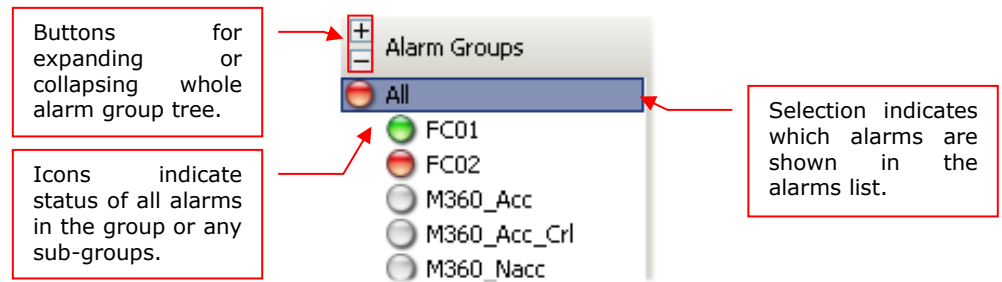
Figure 5-2: Alarm Summary Control

The control is fully customizable as will be described in the following sections.

The control can be inserted on a worksheet or VBA form by clicking the  icon.

### Alarm Groups

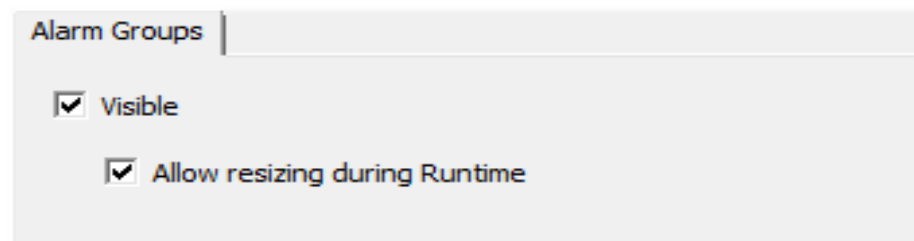
The left part of the control contains all configured alarm groups. The tree can be used for quickly navigating through the alarm groups. Additionally, the icons of the alarm groups indicate whether an alarm is active in the alarm group.



**Figure 5-3: Alarm Groups Tree**

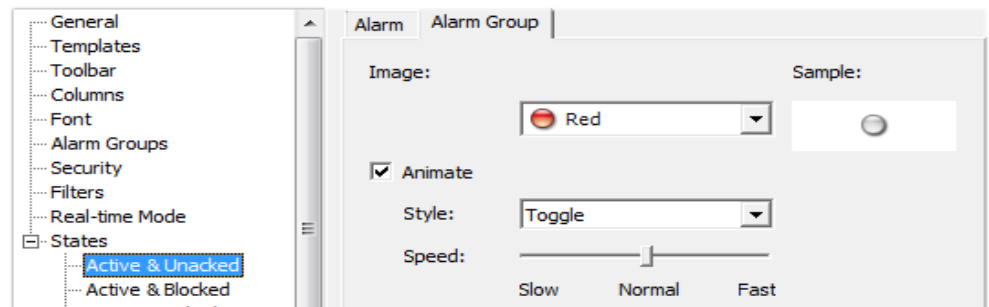
When an alarm group is selected, all alarms that are part of that alarm group or any sub-groups are shown in the alarms list.

The following global properties are supported for alarm groups:



**Figure 5-4: Alarm Groups Properties**

The alarm group indicators can be configured on a per state basis:



**Figure 5-5: Alarm Group State Properties**

When at least one alarm exists in the alarm group, which is in a particular state, the image of that state is used. The priority of the states is shown above. The topmost state ("Active & Unacked") has the highest priority.

## Alarm List

The alarm list shows all alarms of the currently selected alarm-group and filter:

State	Timestamp	Name	Location	Priority	Type	Description	Limit	Deadband
	03/05/07 12:12:29	FC02_ARCHIVE_ALM	FC02	0	Status Alarm	Archive memory alarm	1	
	03/05/07 12:12:25	FC02_PRINTBUF_FULL	FC02	0	Status Alarm	Printer buffer is full	1	
	03/05/07 12:12:31	FC01_CALDATA_ERR	FC01	0	Status Alarm	Calibration data checksum error	1	
	03/05/07 12:12:31	FC02_CALDATA_ERR	FC02	0	Status Alarm	Calibration data checksum error	1	
	03/05/07 12:12:27	FC02_EPROM_ERR	FC02	0	Status Alarm	Flow computer EPROM error	1	
	03/05/07 12:12:31	M360_CORTOT_ERR	M360_Nacc	0	Status Alarm	Correctable totalizer error occurred	1	
	03/05/07 12:12:31	M360_CRL_AOUT_F...	M360_Nacc_Crl	0	Status Alarm	Coriols mA output fixed	1	
	03/05/07 12:12:31	M360_CRL_CAL_INP...	M360_Nacc_Crl	0	Status Alarm	Coriols self-calibration in progress	1	
	03/05/07 12:12:31	M360_CRL_EPROM...	M360_Acc_Crl	0	Status Alarm	Coriols EPROM failure	1	
	03/05/07 12:12:31	M360_CRL_FOUT_O...	M360_Acc_Crl	0	Status Alarm	Coriols frequency out overranged	1	

Figure 5-6: Alarm Summary List

The columns of the list can be clicked in order to sort the alarm list. When the sort order is set to a specific column, a small arrow appears in that column. When the currently sorted column is clicked, the sort order is reversed.

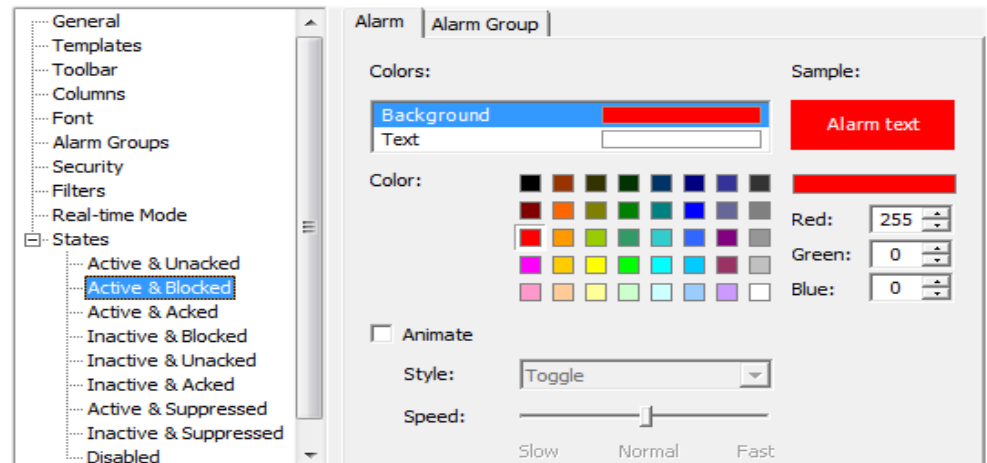
The following columns are supported by the alarms list:

Column	Description
ID	Unique ID that has been assigned to each alarm.
State	Set of icons which indicate the status of the alarm: <ul style="list-style-type: none"> <li> Alarm is active.</li> <li> Alarm has been acknowledged.</li> <li> Alarm is blocked (too many alarm state changes have occurred).</li> <li> Alarm is being suppressed.</li> <li> Alarm has been permanently disabled.</li> </ul>
Timestamp	Date and time at which the alarm was last changed.
Name	Name of the tag (or alias when specified).
Location	Name of the alarm group
Priority	Alarm priority (lower value = higher priority).
Type	One of the following alarm types: <ul style="list-style-type: none"> <li>"Status Alarm"</li> <li>"Low Alarm"</li> <li>"LoLo Alarm"</li> <li>"High Alarm"</li> <li>"HiHi Alarm"</li> </ul>
Description	Alarm description when specified, otherwise tag description.
Limit	Current limit value (no limit is shown for status alarms).
Deadband	Deadband value or nothing when no deadband configured in the tag database.
LastValue	Last value of tag.
Units	Engineering units.
Delay	Alarm delay or nothing when no delay configured in the tag database.
BlockCount	Blockcount of the alarm. When max is reached, the alarm is blocked and will stop changing states until acknowledged.

Column	Description
Format	Format of the tag or nothing when no format configured in the tag database.

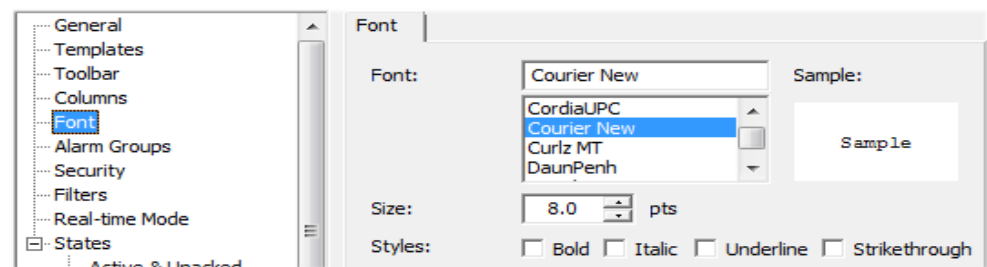
**Table 5-1: Alarm Summary Columns**

The colors used in the alarm list can be configured on a per-state basis:

**Figure 5-7: Alarm List Colors**

When alarms are animated, the background and text colors are switched to create a blinking or toggle-effect.

The font used by the alarm list can be changed to any desired font:

**Figure 5-8: Alarm List Font**

## Alarm States

Alarm colors and the alarm group indicators can be configured on a per-state basis. A state is a combination of one or more properties (e.g. "In Alarm", "Aked", etc.). If an alarm has all the properties which are defined in the state, then that alarm is effectively in that "state". For instance, if an alarm is in alarm, not acknowledged and not blocked, suppressed or disabled, then it is considered

to be "Active & Unacked". On the other hand, if an alarm is disabled then it is considered to be in the state "Disabled" no matter its other properties.

The order of the states is also used to determine which colors to use when drawing an alarm and also to determine which alarm-group indicator to use.

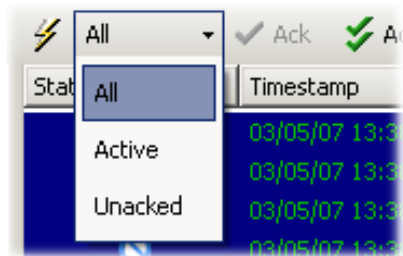
The following states are supported:

State	Rules
Active & Unacked	Alarm must be active, not acknowledged, not suppressed, not blocked and not disabled.
Active & Blocked	Alarm must be active, blocked, not suppressed and not disabled.
Active & Acked	Alarm must be active, acknowledged, not blocked, not suppressed and not disabled.
Inactive & Blocked	Alarm must be inactive, blocked, not suppressed and not disabled.
Inactive & Unacked	Alarm must be inactive, not acknowledged, not blocked, not suppressed and not disabled.
Inactive & Acked	Alarm must be inactive, acknowledged, not blocked, not suppressed and not disabled.
Active & Suppressed	Alarm must be active, suppressed and not disabled.
Inactive & Suppressed	Alarm must be inactive, suppressed and not disabled.
Disabled	Alarm must be disabled.

**Table 5-2: Alarm States**

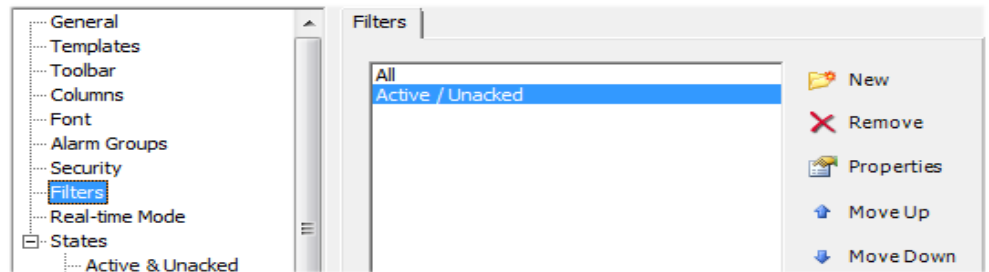
## Filters

Filters can be used to limit the alarms visible in the alarms list. The filters can be selected using the pull-down button in the toolbar:



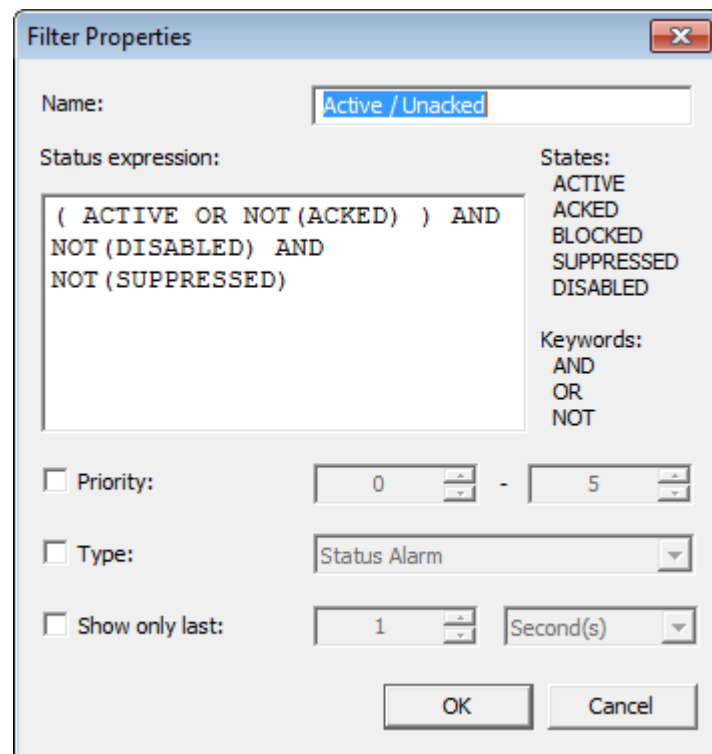
**Figure 5-9: Using Alarm Filters**

Initially 3 filters are available. These filters can be extended and/or modified using the filters property dialog:



**Figure 5-10: Configuring Alarm Filters**

To modify an alarm filter, select the filter and click the “Properties” button. The following dialog shows all the supported properties of a filter:




**Figure 5-11: Alarm Filter Properties**



## Real-time mode

The alarm summary control can be used in two modes: real-time & diagnostical. The real-time mode can be configured to always show the alarms according to a specific alarm-group, filter and sort-order. When the operator changes one of the previous settings, real-time mode is automatically disabled. Real-time mode can be configured to automatically activate after x seconds of user-inactivity. This option is useful when making sure that the alarm summary is always showing the active alarms even if the operator has selected another alarm group, filter or sort order and left the system like that.

Whether real-time mode is enabled or not can be recognized by state of the  button in the toolbar.

The following properties are supported for real-time mode:

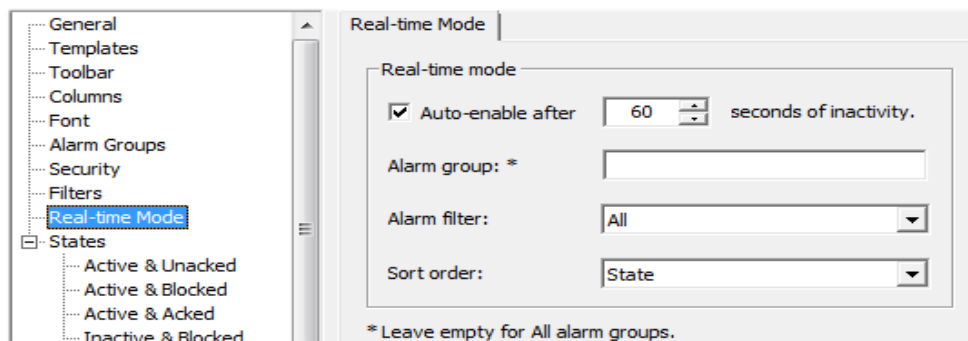


Figure 5-12: Real-time Mode Properties

## Acknowledgement

Alarms can be acknowledged using 3 buttons in the toolbar:

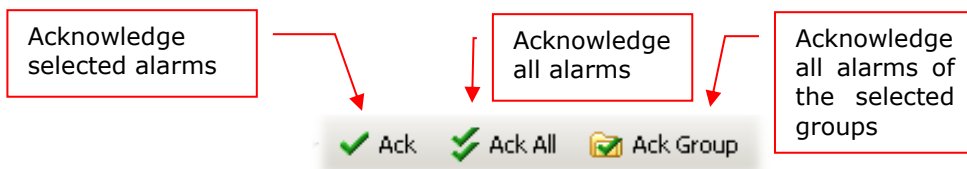

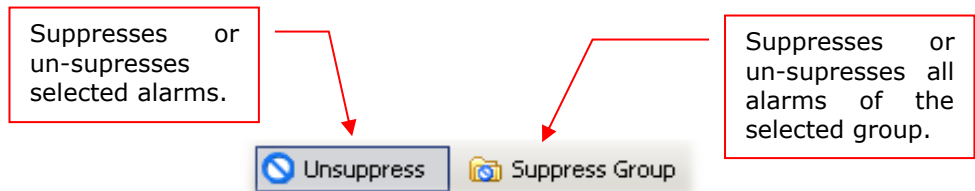


Figure 5-13: Alarm Acknowledgement Buttons

The buttons are enabled only when acknowledgement is possible for the alarms. If the buttons are disabled, the alarms have already been acknowledged or no alarms have been selected. When an alarm has been acknowledged, the  icon is shown in the state column.

## Suppression

Alarms can be suppressed or un-suppressed using the 2 buttons in the toolbar:

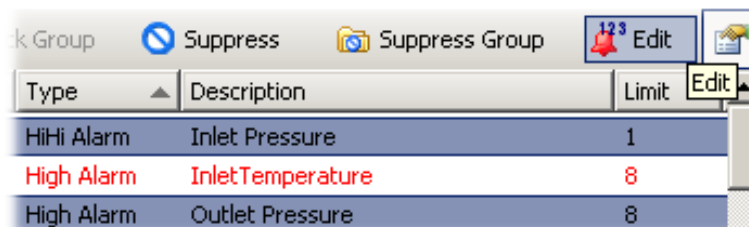


**Figure 5-14: Alarm Suppression Buttons**

The buttons are enabled only when suppression/un-suppression is possible for the selected alarms. If the buttons are disabled, no alarms have been selected. When the button is enabled and highlighted (see "Unsuppress" button in the picture above), all the selected alarms have already been suppressed. Clicking an already highlighted button, will cause all selected alarms to be un-suppressed.

## Editing Alarms

Alarms can be edited either by double-clicking them or using the button in the toolbar. Using the button in the toolbar has the advantage that multiple alarms can be selected for editing:



**Figure 5-15: Editing one or more Alarms**

When editing one or more alarms, the following dialog is shown:

The dialog box is titled "Edit Alarm(s)..." and contains the following sections:

- Limits:** Four input fields with spinners for "HiHi", "High", "Low" (containing 10.1), and "LoLo" (containing 17.01).
- Deadband:** One input field with a spinner.
- Delay [sec]:** One input field with a spinner.
- Buttons:** "OK" and "Cancel" buttons.
- Table:** A table showing the current settings for the selected alarms.

Name	Type	Limit	Deadband	Delay
FT-205-3B Pressure	Low Alarm	10.1		
FT-205-2B Temperature	LoLo Alarm	17.01		

**Figure 5-16: Edit Alarms Dialog**

In the top-part of the dialog, the user can enter new limits, a deadband and a delay. Only those features are enabled which have actually been configured in the Tag Database. For instance, if the "Delay" option is disabled it has not been configured in the Tag Database.

The bottom-part of the dialog shows all alarms that have been selected for editing and also shows the current settings of each alarm.

## Security

Typically, not all users are allowed to suppress or edit alarms. The following security settings can be configured:

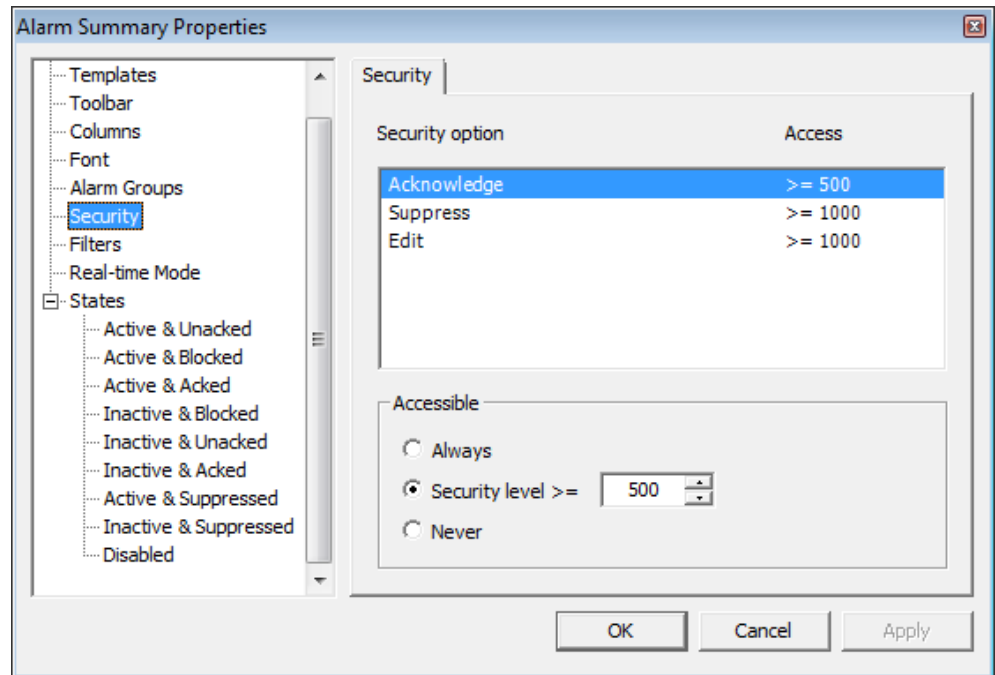
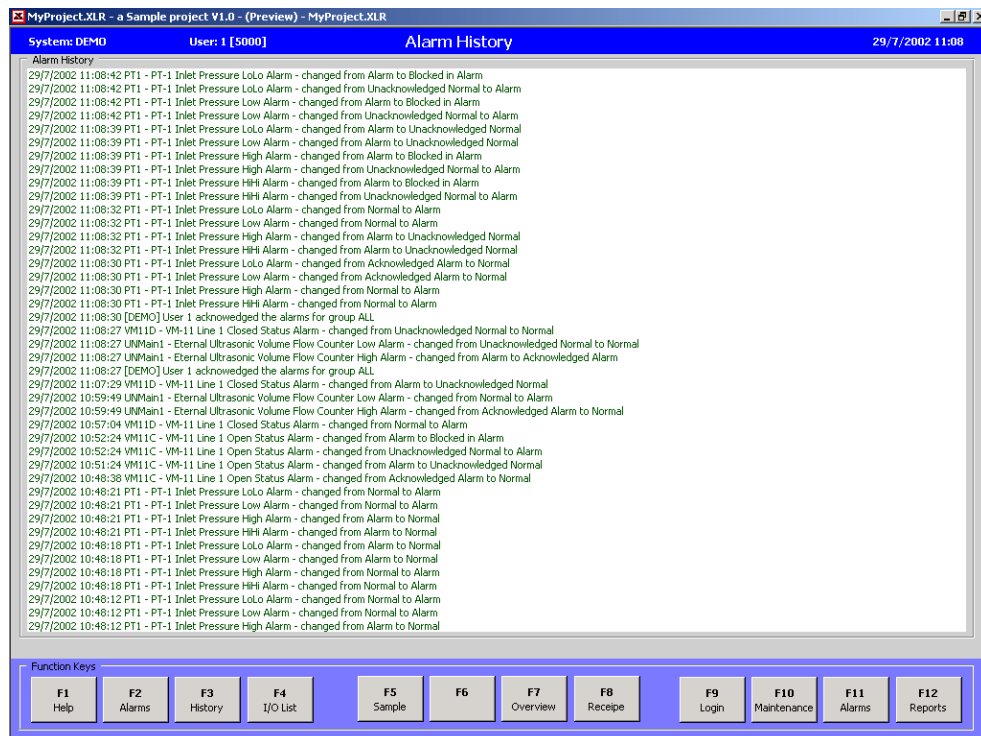


Figure 5-17: Alarm Summary Security Settings

## Worksheet: Alarm History

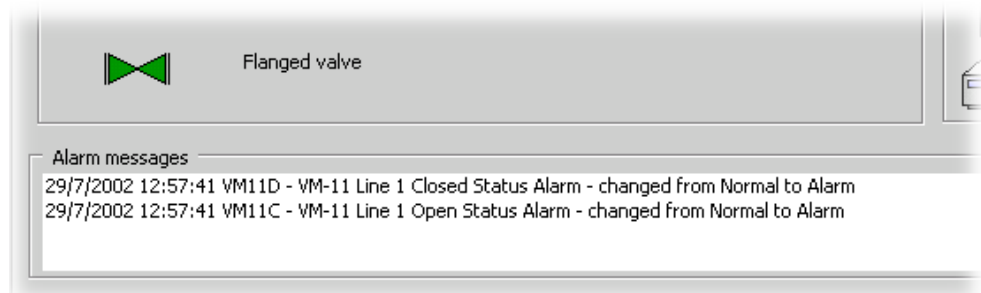
The alarm history is a worksheet containing an historical list with events related to alarm messages, or alternatively, a list with all events of the system. Both types of event history windows are explained in this section.

A typical alarm history looks as follows:



**Figure 5-18: Alarm history worksheet**

The alarm history in the sample project is a worksheet containing alarm messages in a non-scrollable window. It contains the last 42 messages from the alarm manager. The alarm history may contain any amount of messages, depending on the available window size. Another example is given below:



**Figure 5-19: An alarm history window with the 4 last alarm messages**

A scrollable history window may be created by the application developer, as provided with the Alarm Summary in the 'MyTemplate' project.

### **Alarm History - Theory of operation (Advanced topic)**

The alarm history can be created from the eXlerate worksheet function:

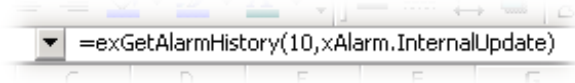
```
exGetAlarmHistory( n, xAlarm.InternalUpdate )
```

which returns an array with  $n$  alarm messages. The second argument to this function ensures updates of the alarm history list. To insert an array formula with 10 rows in a workbook, do the following:

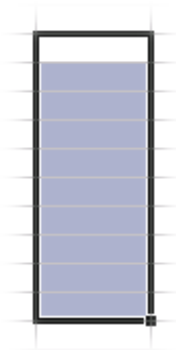
#### **Type the function in a single cell**

In this case, type:

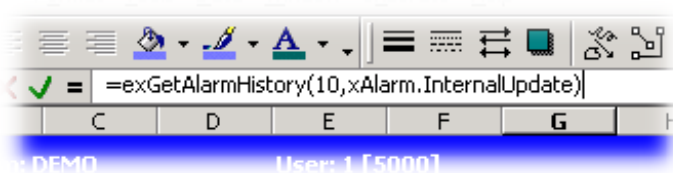
`=exGetAlarmHistory(10, xAlarm.InternalUpdate )`, and press <Enter>. The function is available in a single cell only.



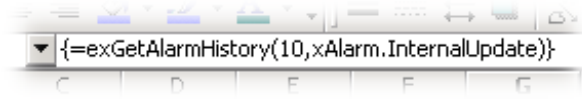
#### **Use the <Shift-Down> key to select 10 rows, while on the worksheet function with the cursor**



#### **Press <F2> to edit the formula**



● Press <Shift-Ctrl-Enter> to enter the array formula



You now have created an array function containing 10 rows.

## Worksheet: Event History

An event history is a worksheet containing an historical list with all system events. The user is able to browse through the date, because all system events in **eXlerate** are disk-based. Typically, all events are stored in directory 'C:\XLRX\Logger', where events are stored each day in a separate file.

A typical event history looks as follows:

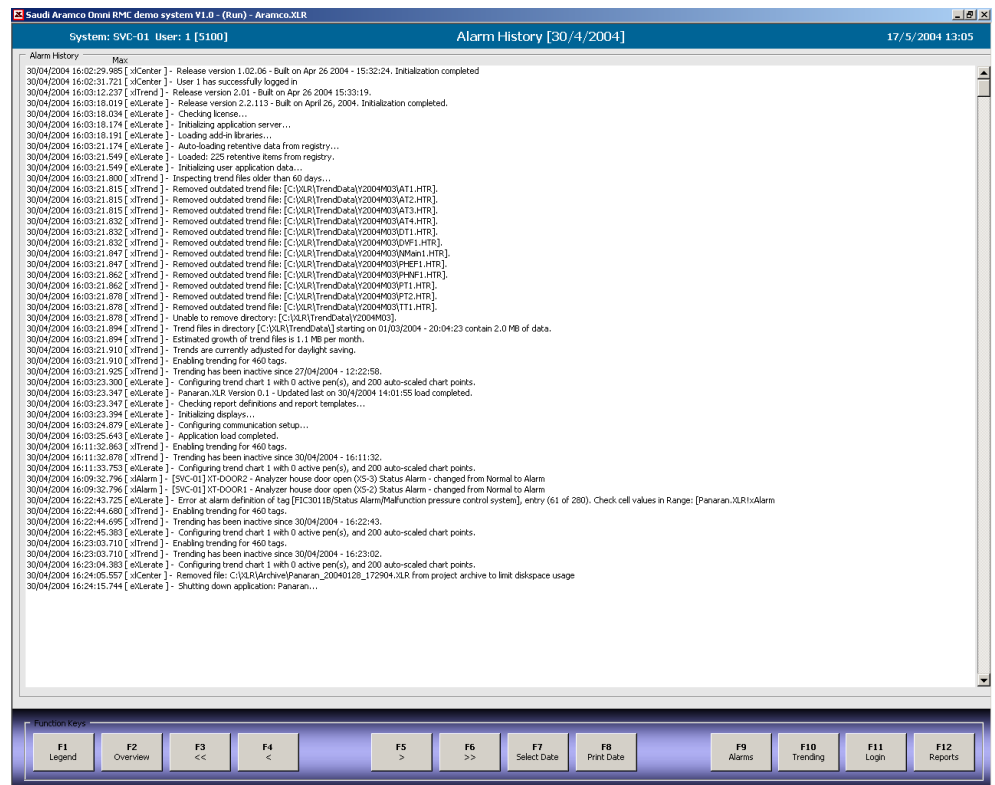


Figure 5-20: Event history worksheet

With the buttons at the bottom of the page, the user can navigate through each day of operation, and select the events. Alternatively, these events can be printed on the event printer.



## Event History - Theory of operation (Advanced topic)

The event history can be created from the **eXlerate** worksheet function:

`exGetEventHistory( n, iStart, strPath, Date, Filter, Retrigger )`, which returns an array with *n* event messages of the specified day.



See the worksheet function reference in the 'Advanced Topics Reference' manual for further details on utilizing this worksheet function.

## Event history buttons

The special buttons at the bottom of the event history worksheet are defined in the *Button Table*, and contains date browsing functions.

The trending section of the *Button Table* contains the following definitions:

Worksheet	Button	Button text	Key	Procedure
AlarmHistory	Button3	F3~<<	{F3}	dspAlmHist.FastBack
AlarmHistory	Button4	F4~<	{F4}	dspAlmHist.Back
AlarmHistory	Button5	F5~>	{F5}	dspAlmHist.Forward
AlarmHistory	Button6	F6~>>	{F6}	dspAlmHist.FastForward
AlarmHistory	Button7	F7~Select Date	{F7}	dspAlmHist.PickDate
AlarmHistory	Button8	F8~Print Date	{F8}	dspAlmHist.PrintHistEventFile

**Table 5-3: Event history button definitions**

## Internal VBA code

The procedure definitions from the *Button Table* above contain references to the VBA-code at the specific event history worksheet, which is internally called: 'dspAlmHist'..

The VBA code at this Excel worksheet contains the following macros/procedures:

```
Sub PickDate()
    frmEventData.Show
End Sub

Sub FastBack()
    Range("HistEventDate").Value = Range("HistEventDate").Value - 7
    Range("HistEventCurr").Value = 0
End Sub

Sub Back()
    Range("HistEventDate").Value = Range("HistEventDate").Value - 1
    Range("HistEventCurr").Value = 0
End Sub
```

```
Sub Forward()  
    Range("HistEventDate").Value = Range("HistEventDate").Value + 1  
    Range("HistEventCurr").Value = 0  
End Sub  
  
Sub FastForward()  
    Range("HistEventDate").Value = Range("HistEventDate").Value + 7  
    Range("HistEventCurr").Value = 0  
End Sub  
  
Sub Scrollbar_Change()  
    On Error Resume Next  
    Shapes("Scrollbar").ControlFormat.Max =  
Range("HistEventMax").Value  
End Sub  
  
Sub PrintHistEventFile()  
Dim sCommand As String  
Dim sFileName As String  
Dim fs  
On Error Resume Next  
    Set fs = CreateObject("Scripting.FileSystemObject")  
    sFileName = Range("HistEventPath").Value & "\" & _  
        Format(Year(Range("HistEventDate").Value), "0000") &  
-  
        Format(Month(Range("HistEventDate").Value), "00") &  
-  
        Format(Day(Range("HistEventDate").Value), "00") &  
".log"  
    If fs.FileExists(sFileName) Then  
        sCommand = "Notepad /p " & sFileName  
        Call Shell(sCommand, vbHide)  
    Else  
        MsgBox "No events available for this date", vbOKOnly,  
"Printing Historical Events"  
    End If  
End Sub
```

**Figure 5-21: Trending related VBA code**

In the code above, which is copied from the 'MyProject' sample, various buttons are linked to the above VBA macros.

## VBA and worksheet functions

The following functions are available for alarm management. These functions are described in more detail in the Function Reference Manual.

● **exAlarmCount (...)**

This is a function returning the number of alarms in a specific mode, such as all active alarms, or all unacknowledged alarms, for a specific alarm group.

● **exSetAlarmDeadband (...)**

This is a procedure which allows the user to change an alarm dead-band value. This advanced function is used in cases where alarm dead-band values are set from within VBA and not directly via the tag database.

● **exSetAlarmLimit (...) & exSetAlarmLimit2 (...)**

This is a procedure which allows the user to change an alarm limit without the standard alarm editor. This advanced function is used in cases where alarm limits are set from within VBA rather than from the standard alarms worksheet.

● **exSetAlarmDelay (...)**

This is a procedure which allows the user to change the alarm delay without the standard alarm editor. This advanced function is used in cases where alarm delays are set from within VBA rather than from the standard alarms worksheet.

● **exAlarmSetModeByGroup (...)**

This procedure which allows a user to programmatically disable or suppress a group of alarms. It may be used for example when a production line is switched off, to disable or suppress all related alarms.

● **exAlarmSetModeByID (...)**

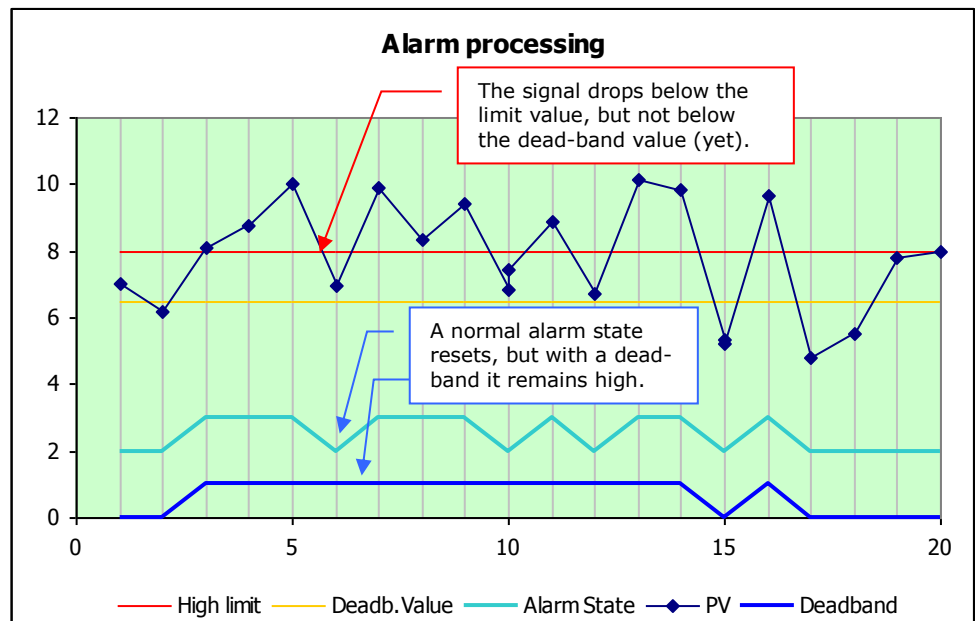
This procedure which allows a user to programmatically disable or suppress an specific alarm.

● **exAlarmShowStatus (...)**

This procedure creates a report containing all active alarms, all disabled alarms, or all suppressed alarms, and sends the report to the system event logger (and hence to the printer, if enabled).

## Alarm dead-band

An alarm dead-band is used to suppress the generation of excessive alarms due to the fact that the process value varies just at the alarm limit value, as in the following example:



**Figure 5-22: Alarm dead-band example used for a high limit alarm**

The process value in the example above is the dark blue dotted line at the top varying between '5' and '10'; the alarm limit is the red line set at '8', and a dead-band value, the yellow line, of '1.5' is defined (and hence drawn at 8-1.5, or 6.5).

The thick light blue line above the bottom line represents the alarm state where no dead-band (or a dead-band value of '0') is applicable. As soon as the process value exceeds the alarm limit, then the alarm state is raised, and as soon as the process value drops below the limit value, the alarm state is reset. In the example above, this introduces 10 alarm state changes.

The thick dark blue line at the bottom of the graph represents the alarm state when a dead-band value of '1.5' is applicable. As can be seen, the smaller changes in the process value are ignored and thus the amount of alarm state changes are only 4.

This is because with a dead-band value, the alarm state is reset when the process value drops below (high limit - dead-band).

For a low alarm, or a low-low alarm this process works similarly, but reversed, i.e. the alarm state is reset when the process value exceeds (low limit + dead-band).

A dead-band is entered at the tag database in a column: "Deadband", and is entered in engineering units. When the column "Deadband" does not exist, the dead-band mechanism is effectively disabled.

To (re-)enable the dead-band mechanism, insert the column in the tag database, and fill-in a non-zero value at the appropriate tag. The dead-band parameter value entered for a tag is used for all combinations, if any, of a high-high alarm, a high alarm, low alarm, and low-low alarm.

## Other alarm options



There are several additional options that can be configured using the VBA callable procedure `exAlarmSetOptions(...)`. See the function reference manual for details regarding alarm options. Options include:

### ● **AllowBlocking**

Alarms may be configured with blocking enabled. When blocking is enabled, no more alarm messages are generated when repeatedly alarms change status from normal to alarm and vice versa without being acknowledged. The amount of times that an alarm changes state before it obtains a 'blocked' status is a user-defined parameter.

### ● **AutoAcknowledge**

Alarms may be configured for automatic acknowledgement. This causes an alarm to automatically being acknowledged when occurring, e.g. the user does not need to manually acknowledge an alarm when it occurs.

Note that when *AutoAcknowledge* is enabled, blocking has no effect.

# Chapter 6 - Trending

## Introduction

In **eXlerate**, the transition between *real-time* trending and *historical* trending is smooth: a real-time trend in **eXlerate** is a special case of an historical trend: it is simply an historical trend with the current time on the right-hand side of the time-axis.



Internally in **eXlerate**, the data recording mechanism is totally separate from the display of trends. Data recording/collection always continues, even when there are no trend displays defined at all in an application. Data recording / collection is defined in the tag database.

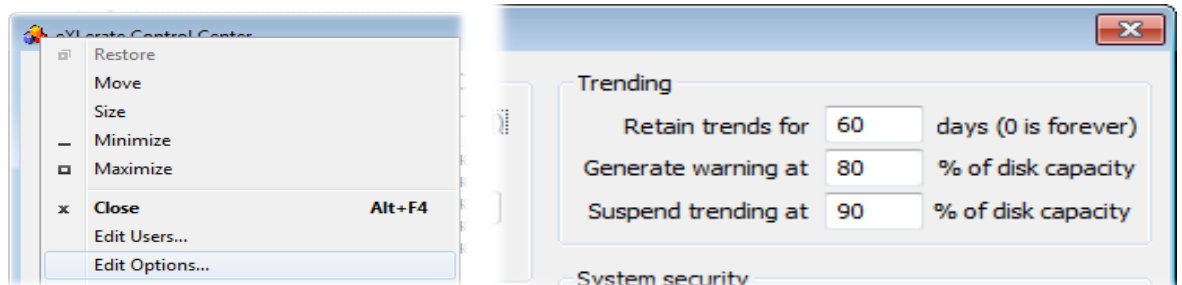
## Enabling Trending

Trending can be enabled on a per tag basis. This is discussed in *Tag database* of the Application Reference Manual. After the 'TrendNorm' property has been specified the Tag & Object Wizard should be run to enable trending for the configured tags.

Additionally, for each application shortcut in the Control Center, the 'Trending Path' must be specified.

## Trending Parameters

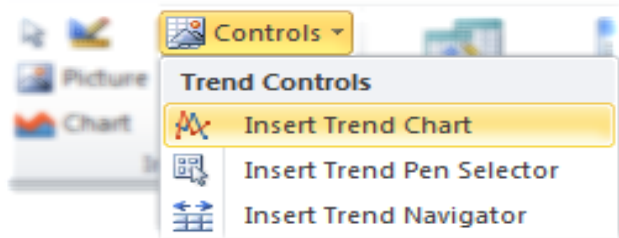
There are several parameters for trending, which are maintained at system level at the Control Center:



The most important is the 'Retain trends for' parameter. This setting controls the lifetime of the historical trend data. To prevent your hard-disk from filling up with trend-data after months or years, it is wise to set some kind of limit (e.g. 60 days = 2 months), depending on the size of the disk.

## Visualizing Trend Data

Trend-data can be visualized using a set of 3 trend-controls. These controls may be inserted into your application using the 'Controls' option in the 'Insert' section of the eXLerate ribbon:



**Figure 6-1: Insert Trend Controls**

Three Trend controls can be distinguished:

Control	Description
exTrendChart	Visualizes the trend-data in a chart. Contains controls for zooming in and out of the trend-data and moving forward and backward in time.
exTrendPenSelector	Allows the user to select and modify the tags to visualize for trending. This control works in conjunction with the 'exTrendChart' control, in that it selects the tags to visualize in the trend-chart.
exTrendNavigator	This control visualizes a larger portion of the trend-data in order to quickly see anomalies in the data. This control also works in conjunction with the 'exTrendChart' control, in that it allows you to quickly select the time-period for the trend-chart.

**Table 6-1: Trend Controls**

To start using the Trend Controls, insert a "Trend Chart" onto a worksheet or VBA Form and give it a proper name. After this, insert a "Trend Pen Selector" and attach its 'Target' to the previously inserted trend-chart.

These controls are discussed in detail furtheron in this chapter.

## exTrendChart Control

This is the main control which visualizes the trend-data in the form of a chart. The control is similar in appearance to a Chart as can be found in Microsoft Excel.

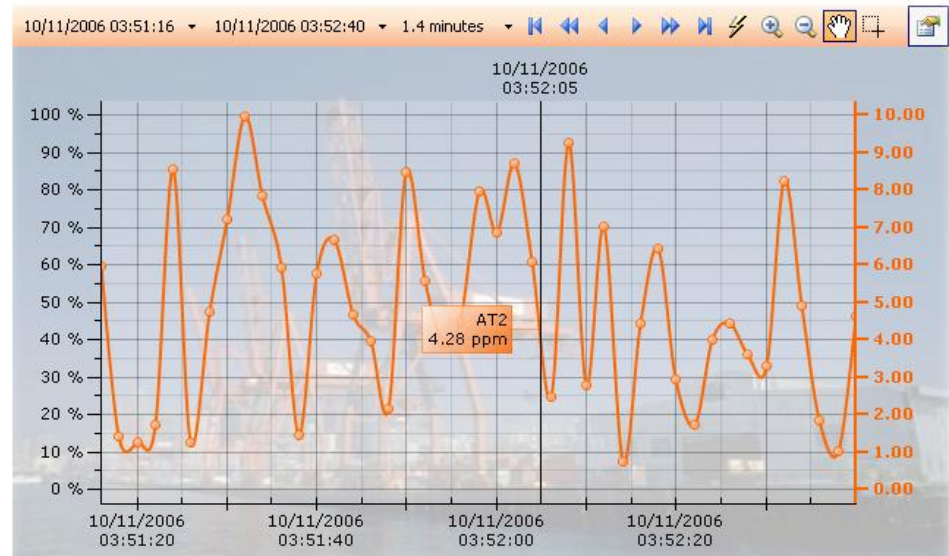


Figure 6-2: exTrendChart Control

### Chart Area

The Chart Area encapsulates the whole control excluding the toolbar. In the example above the Chart Area has a background picture and no border.

### Plot Area

The Plot Area is the region in which the drawing of the trend-lines is performed. The region that is not occupied by the plot-area is used by the axes. The following properties are available for the plot-area:

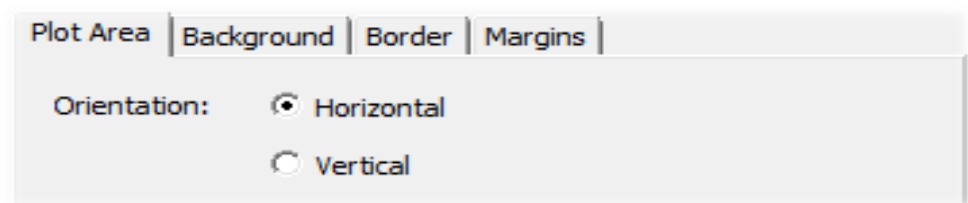


Figure 6-3: Plot Area properties



By default the plot-area is oriented horizontally which creates horizontal time-axes and vertical value-axes. When oriented vertically, the time-axes become vertical and the value-axes become horizontal.

The margins define the available space for the axes. These can be changed in the "Margins" tab or by clicking on an axis and dragging the edge:

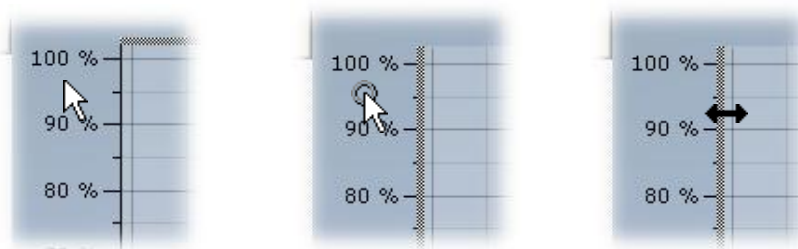
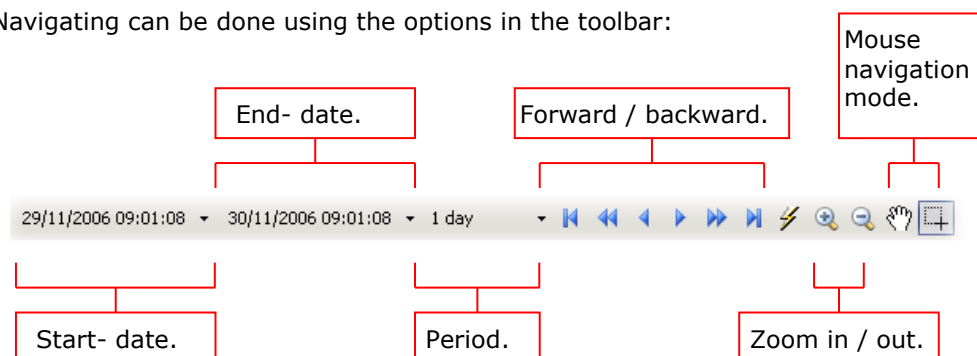


Figure 6-4: Plot area margins

## Navigation

Navigating can be done using the options in the toolbar:



The toolbar is fully configurable and buttons may be removed when desired.

Button	Icon	Description
Start-date		Shows the start-date/time of the chart. When clicked a popup is shown which allows for selecting a specific date/time.
End-date		Shows the end-date/time of the chart. When clicked a popup is shown which allows for selecting a specific date/time.
Period		Shows the difference between the start- and the end of the chart. When clicked a list is shown with predefined periods.
Move to Begin	⏮	Moves the start of the chart to the first time a value was stored for any of the pens.
Fast Backward	⏮	Moves the chart backward fast. By default the fast step-size is set to 100% which causes the chart to move backward a whole page.
Backward	⏮	Moves the chart backward. By default the step-size is set to 25% which causes the chart to move backward by a quarter of a page.
Forward	⏭	Moves the chart forward. By default the step-size is set to 25% which causes the chart to move forward by a quarter of a page.

Button	Icon	Description
Fast Forward		Moves the chart forward fast. By default the fast step-size is set to 100% which causes the chart to move forward a whole page.
Move to End		Moves the end of the chart to the current date/time.
Realtime		Enables or disables realtime-mode. When enabled the end-date remains fixed on the current date/time. Also the data-cursor remains fixed on the current date/time. When any of the other buttons is pressed, realtime-mode is automatically disabled allowing the user to perform historical viewing.
Zoom In		Zooms in on the current cursor position.
Zoom Out		Zooms out. When this button is pressed the value axis-zoom is always reset to 0%-100%.
Moving Mode		Enables mouse-moving mode. When enabled, a hand-cursor appears when moving the mouse over the chart plot-area. When the plot-area is clicked a closed hand appears after which the plot-area may be dragged in any direction.
Zooming Mode		Enabled mouse-zooming mode. When enabled, a cross appears when moving the mouse over the chart plot-area. When the plot-area is clicked a rectangle may be dragged until the mouse button is released. The chart will then zoom onto the selected zoom area.

Table 6-2: Trend Chart Toolbar

The following properties are available for navigation:

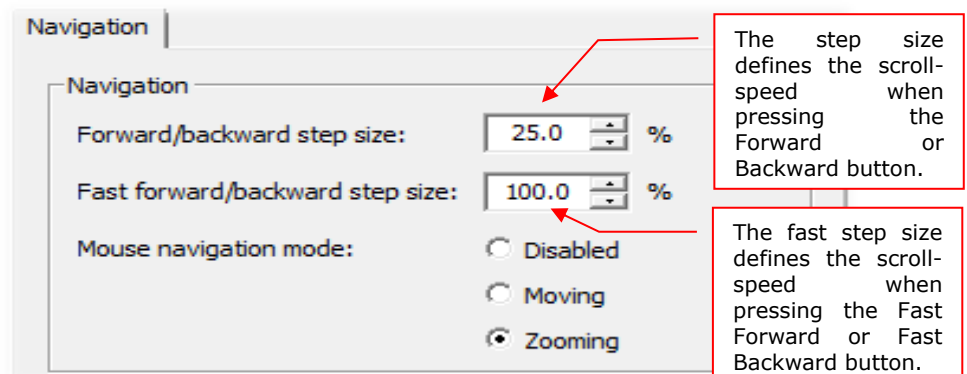


Figure 6-5: Navigation properties

The Properties dialog also contains an option "Zoom Selection". This setting defines the visual appearance when using the mouse zooming-mode to select a zoom-area.

## Data Cursor

The Data Cursor is a tool for accurately viewing pen-values on a specific date/time. The Data Cursor can be put on a specific date/time by moving the mouse over it. When a sizing cursor appears the mouse button can be pressed and the cursor may be moved:

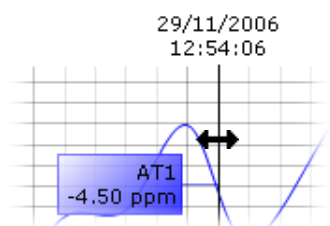


Figure 6-6: Data Cursor

The following properties are available for the Data Cursor. The Background and Border tabs define the visual appearance of the cursor:

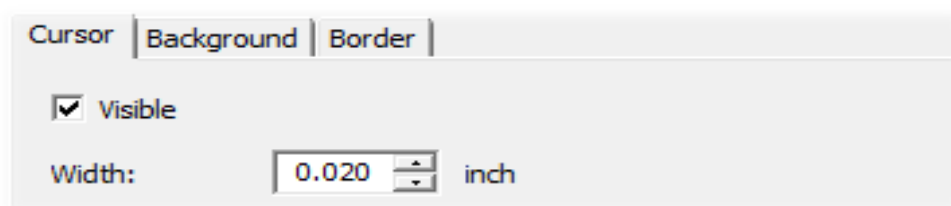


Figure 6-7: Data Cursor Properties

Optionally, the date/time may be displayed on the top- or bottom-axis:

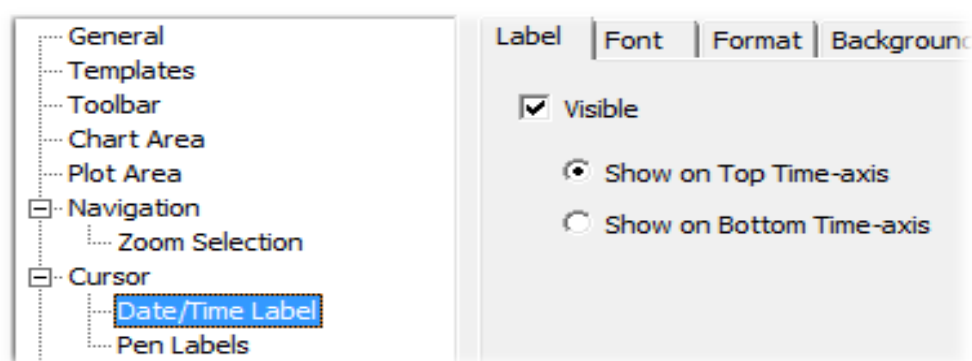


Figure 6-8: Data Cursor Date/Time

## Pen Labels

By default the value of each pen is shown as a label on the chart itself. This label may also be disabled after which the value will only be visible in the PenSelector Control. When enabled, the label can be configured to display additional properties of the tag such as Units & Description:

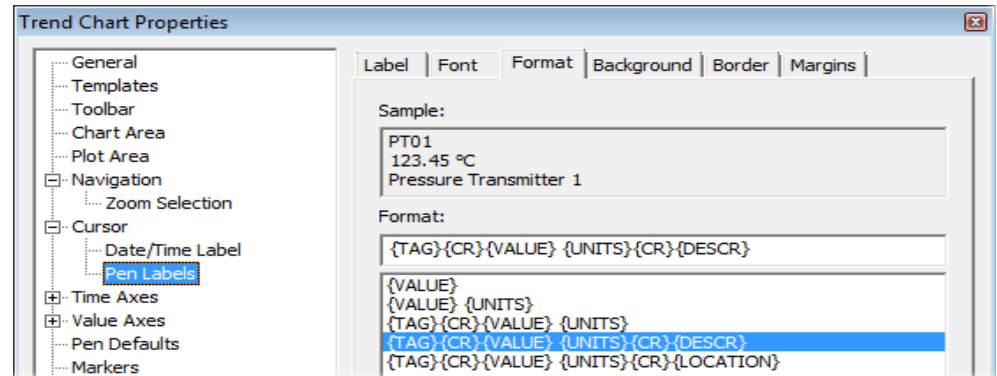


Figure 6-9: Pen Label Properties

By default, the background of the label is displayed in the same color as the pen, this can also be changed so that a fixed color is used instead:

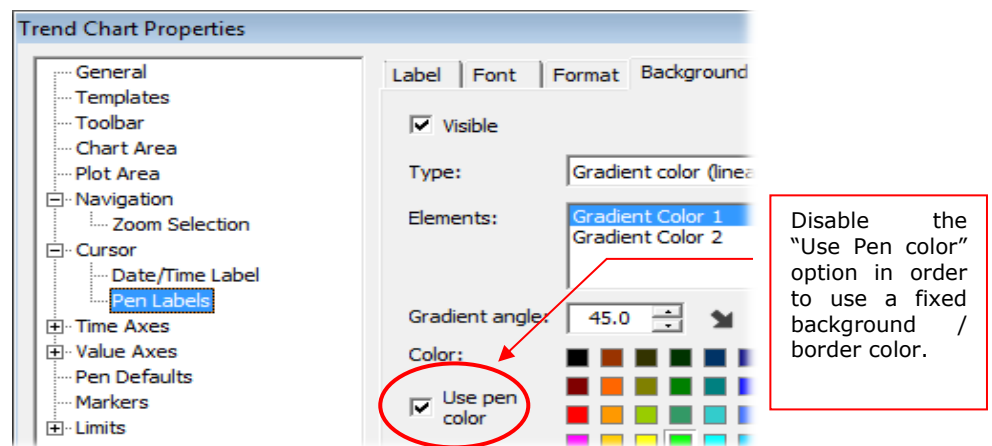


Figure 6-10: Pen Label Color

## Time Axes

Two time axes may be configured. Both the Top & Bottom Axis share certain properties that are accessible through the "Time Axes" properties tab:

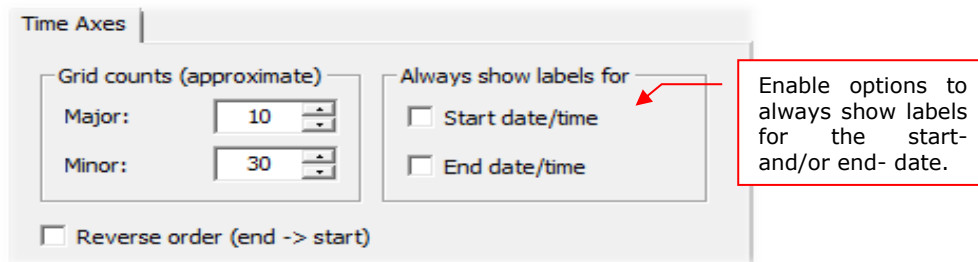


Figure 6-11: Time Axes properties

The grid-counts determine how many tickmarks and labels are shown on the axis. The specified number of grid counts is used as a maximum, **eXlerate** will draw as many tickmarks and labels as properly fit on the chart.

By default the time flows from the left to the right. By enabling the 'Reverse order' option this can be changed so that it flows from the right to the left.

## Value Axes

Two value axes may be configured. Both the Left & Right Axis share certain properties which are accessible through the "Value Axes" properties tab:

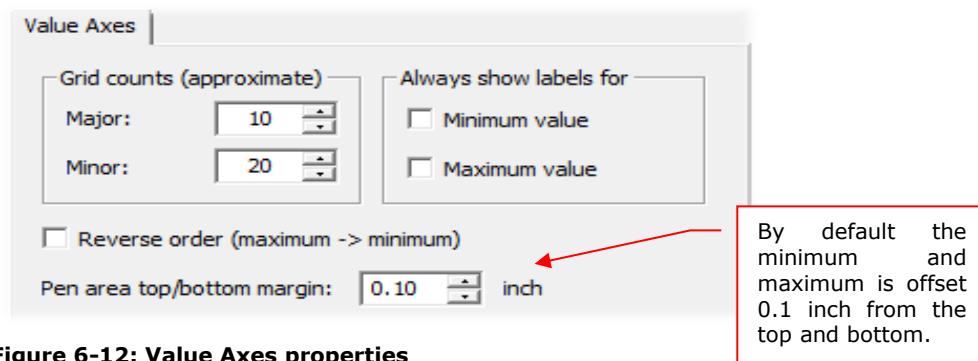


Figure 6-12: Value Axes properties

The grid-counts determine how many tickmarks and labels are shown on the axis. The specified number of grid counts is used as a maximum, **eXlerate** will draw as many tickmarks and labels as properly fit on the chart.

By default the minimum is displayed on the bottom and the maximum on the top. By enabling the 'Reverse order' option this can be changed so that the minimum is displayed on the top and the maximum at the bottom.

## Scaling

eXlerate contains an auto-scaling option which determines the upper- and lower scaling boundaries based on the visible trend-values, and optionally the limits. It is also possible to use a fixed value or a so-called dynamic value. When for instance, the dynamic value "Tag minimum" is used, the "Min" property as configured in the xTagDB is used.

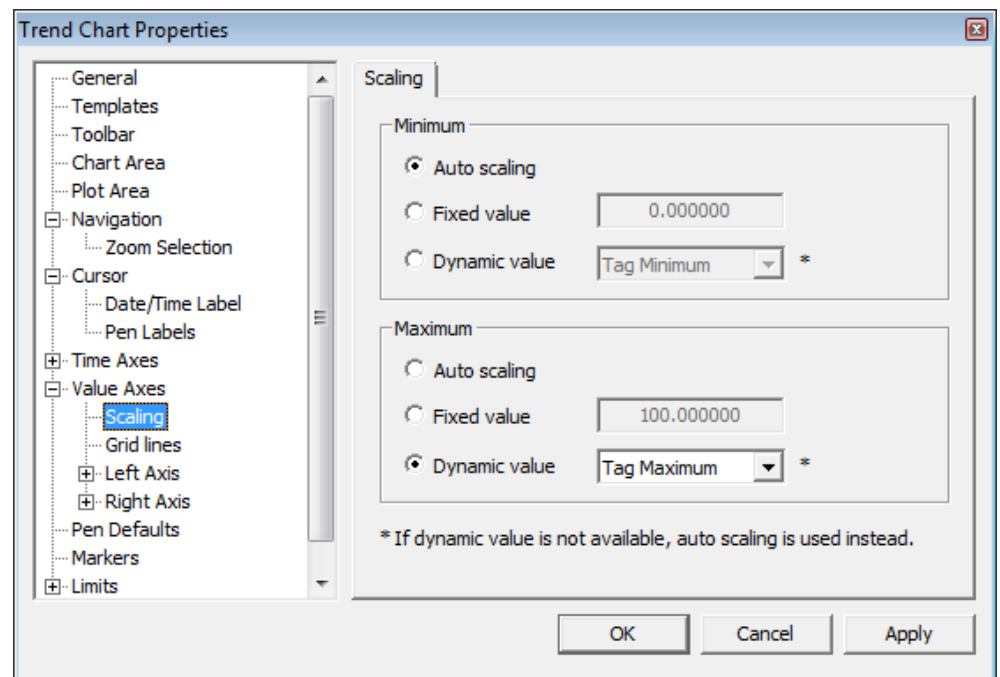


Figure 6-13: Scaling properties

## Custom scaling

Trend-pens may also be scaled manually from VBA. In this case the 'MaximumScaleType' property of the TrendPen should be set to 'exTrendValueAxisScaleTypeFixed', and the 'MaximumValue' should be set to the desired scaling-value. The Minimum can be scaled in the same manner:

```
...
exTrendChart.Pens.Item(1).MaximumScaleType = _
    exTrendValueAxisScaleTypeFixed
exTrendChart.Pens.Item(1).MaximumFixedValue = 100
...
```

### Value Axis Styles

Each of the value axis (left & right) may be configured as either Percentage or Engineering Units:

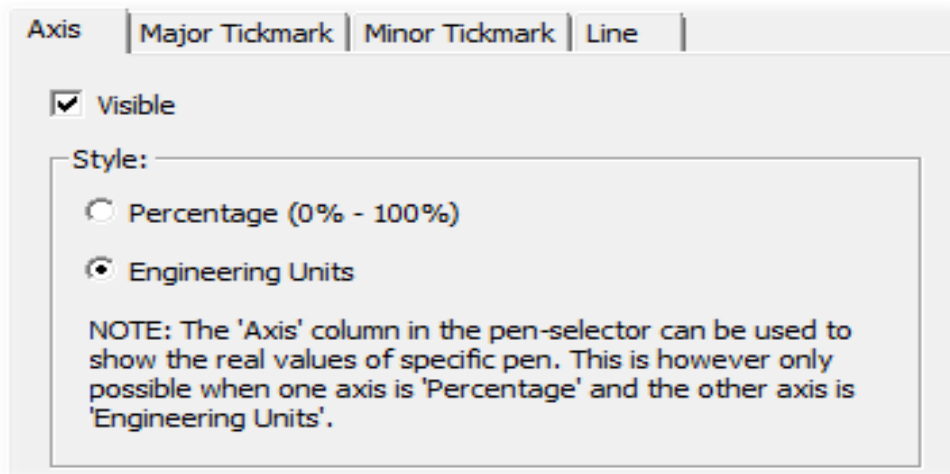


Figure 6-14: Value Axis Styles

When percentage is selected the axis displayed as 0%-100% and each pen is individually scaled. This means that the minimum of each tag is scaled to 0% and the maximum is scaled to 100%:

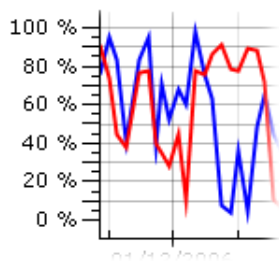


Figure 6-15: Percentage Axis

When Engineering Units is selected the actual trend values are displayed and the all pens are scaled globally. The maximum and minimum of all pens is determined and the highest maximum is used as the top edge and the lowest minimum is used for the bottom edge. In the sample below the same trend values are used as in the Percentage example above.

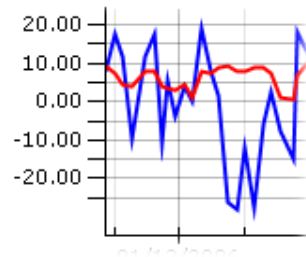


Figure 6-16: Engineering Units Axis

Besides showing either Percentage or Engineering Units, it is also possible to display both. When one axis is Percentage and the other is Engineering Units, each pen is scaled individually and the minimum and maximum of each pen is scaled to 0% - 100%. The Engineering Units Axis can display the values for only **one** pen:

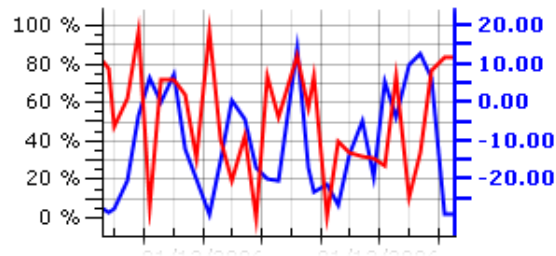


Figure 6-17: Using both Percentage and Engineering Units Axis

In the example above, the right axis (Engineering Units) has been configured to use the pen-color rather than a fixed color. Which pen is displayed on the Engineering Units axis can be selected in the PenSelector control:



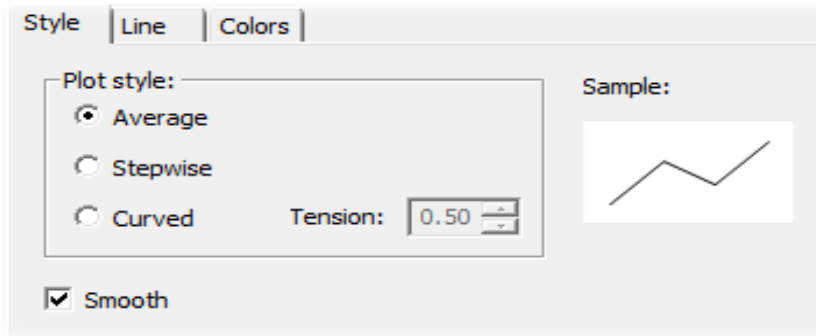
Figure 6-18: Showing Engineering Units Axis for specific pen

The 'Axis' options in the PenSelector are only enabled when the Chart is configured with two different axis.



## Pen Defaults

The defaults for adding new pens to the TrendChart can be configured in the properties dialog:

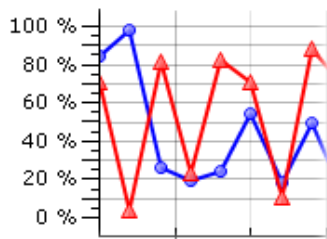


**Figure 6-19: Default Pen Style**

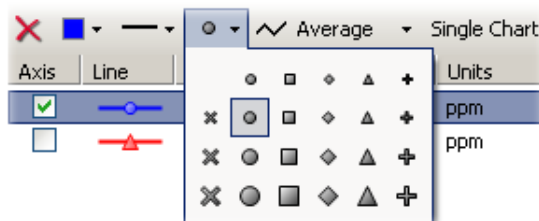
After adding a new pen, the plot-style, line-style and color can be changed by the operator.

## Markers

Optionally markers can be drawn at the actual data points in the chart. Initially when a pen is added no marker is shown. The PenSelector control can be used to select a marker for the pen(s). The background and border of the markers can be configured in the properties dialog.



**Figure 6-20: Markers on the Chart**



**Figure 6-21: Selecting Markers using the PenSelector**

## Limits

When a tag has any alarm limits configured, it is possible to display these limits on the chart:

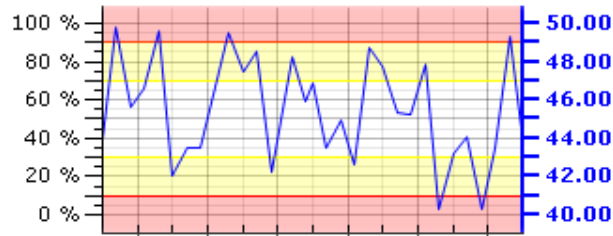


Figure 6-22: Alarm Limits on Chart

The PenSelector control supports a 'Limits' option which shows the limits when selected. When the tag is not configured with any limits, the option is automatically disabled.

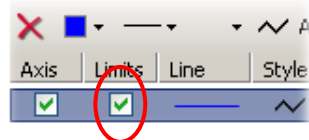


Figure 6-23: Limits option in PenSelector

By default the 'Limits' column is hidden. To show this option open the properties dialog of the PenSelector and enable the appropriate column:

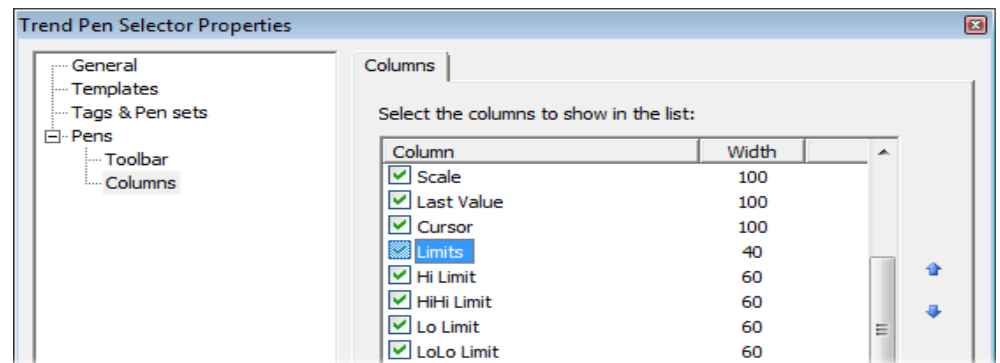
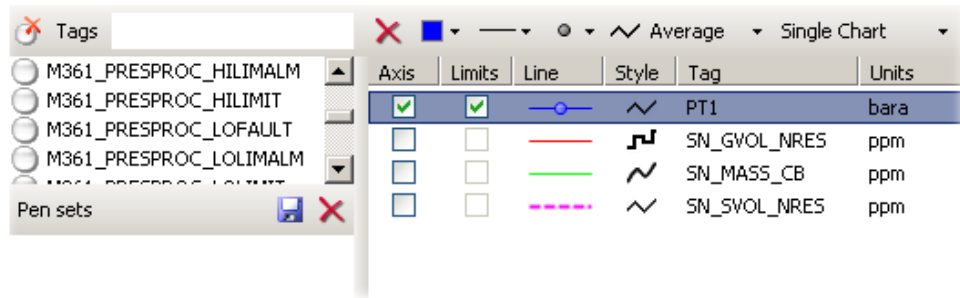


Figure 6-24: Add Limits Column to PenSelector

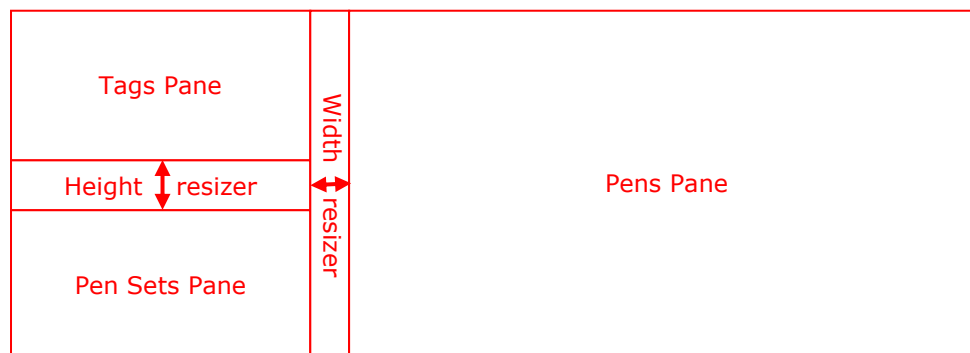
## exTrendPenSelector Control

The PenSelector control combines both the chart legend and the pen selecting mechanism in a single control.



**Figure 6-25: exTrendPenSelector Control**

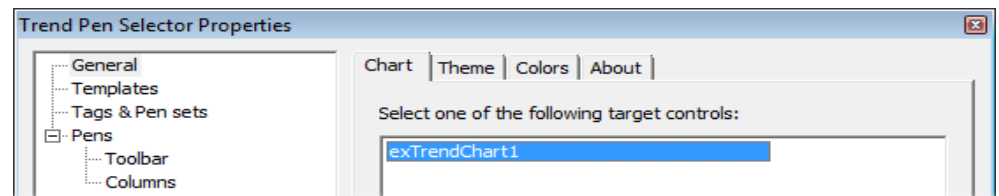
The PenSelector control consists of 3 parts, which can be enabled independently. These 3 parts have been combined in a single control in order to support resizing of the individual panes:



**Figure 6-26: PenSelector individual panes**

## Linking to a Chart control

The first thing that has to be done after inserting a PenSelector control is to attach it to a TrendChart control. After opening the properties dialog a list is shown with all the existing TrendChart controls. One of these controls should be selected as the target control:



**Figure 6-27: Selecting a Target Chart**

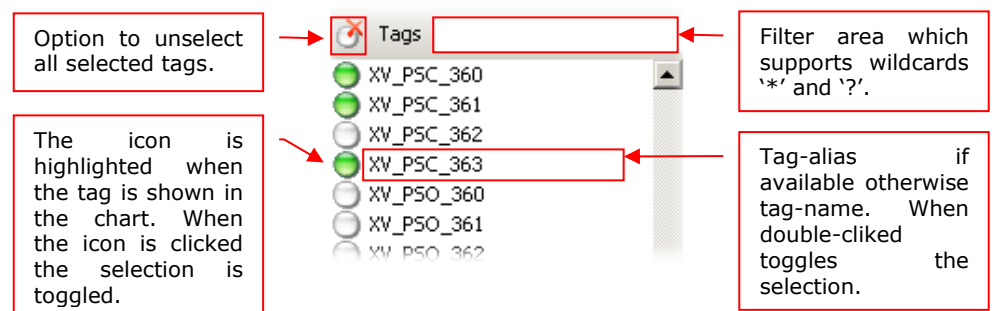
When multiple TrendChart Controls have the same name, they should be renamed to unique names. To rename a Trend Control, enable design-mode and use the Name-box to specify a different name:



**Figure 6-28: Editing names**

## Tags

The tags-pane shows a list of all tags configured for trending. The alias of the tag is shown in the list if available, otherwise the tag-name is displayed:



**Figure 6-29: Tag selection**

The following properties are available for the tags pane:



Figure 6-30: Tags Properties

When the Tags-option is unchecked, the whole tags-pane becomes hidden.

## Pen Sets

The Pen Sets pane shows a list of previously saved pen-sets. A pen-set is a collection of pens and their settings.

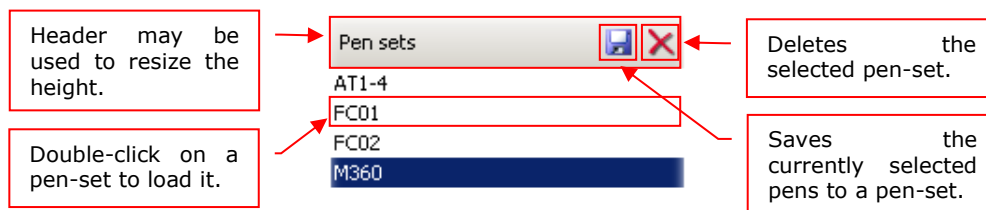


Figure 6-31: Pen sets

The following properties are available for the pen-sets pane:



Figure 6-32: Pen Sets Properties

In order to prevent any user from modifying and deleting pen-sets, the minimal security level for modifying pen-sets can be configured.

Pen sets can be configured in Run-Time mode, so operators can defined their own pen sets. Pen sets can also be pre-configured and then transferred as a file to the target computer.

The following picture shows the PenSets folder when viewed from the Windows Explorer:

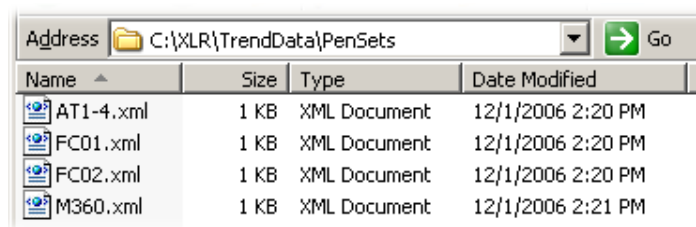


Figure 6-33: Pen-set files

## Pens

The pens-pane shows all currently selected pens and additional properties. A toolbar is provided for modifying the pen properties.

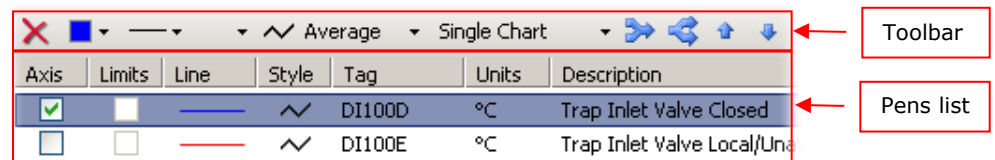


Figure 6-34: Pens Toolbar and List

The pens-pane can be shown or hidden using the properties dialog:

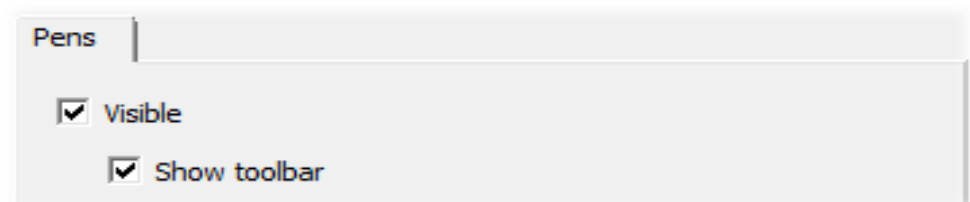











Figure 6-35: Pens Properties

## Pens Toolbar

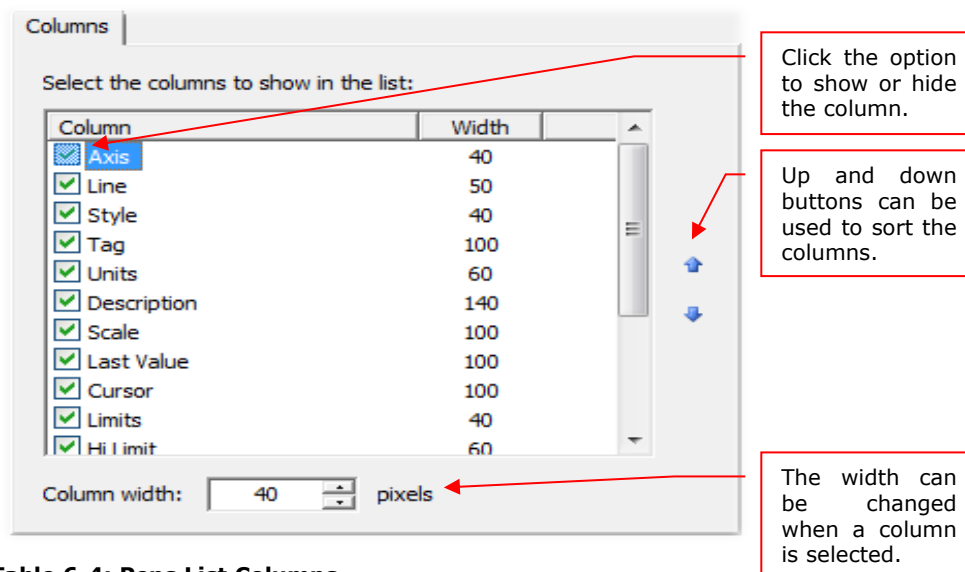
The following options are supported by the pens toolbar:

Button	Icon	Description
Remove		Removes the currently selected pens from the chart.
Color		Selects a different color for the selected pens.
Line		Selects a different line style and weight for the selected pens.
Marker		Selects a marker for the selected pens.
Style		Select the plot-style for the selected pens.
Chart Style		Selects the automatic pen-grouping mode for the chart.
Group		Groups the selected pens together in the same plot-area.
Ungroup		Ungroups the selected pens and creates the separate plot-areas for each pen.
Move Up		Moves the selected pens one position up in the list.
Move Down		Moves the selected pens one position down in the list.

**Table 6-3: Pens Toolbar**

## Pens List

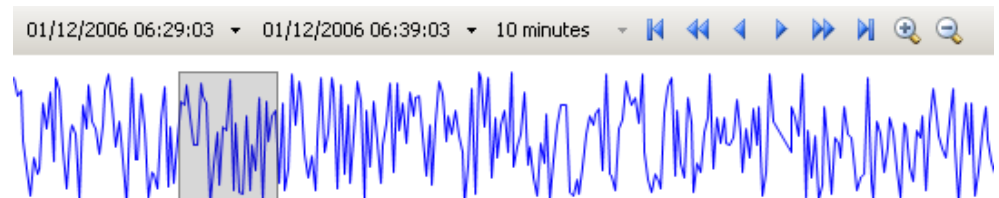
The pens list shows all the pens of the chart and additional properties. The properties are shown as columns of the list. Which columns are visible and the width of the columns can be configured in the properties dialog:



**Table 6-4: Pens List Columns**

## exTrendNavigator Control

The TrendNavigator is an optional control for data inspection and navigation. It shows the same pens as the TrendChart control but for a greater period. The position of the Chart is displayed inside the Navigator and can be moved and resized.



**Figure 6-36: exTrendNavigator Control**

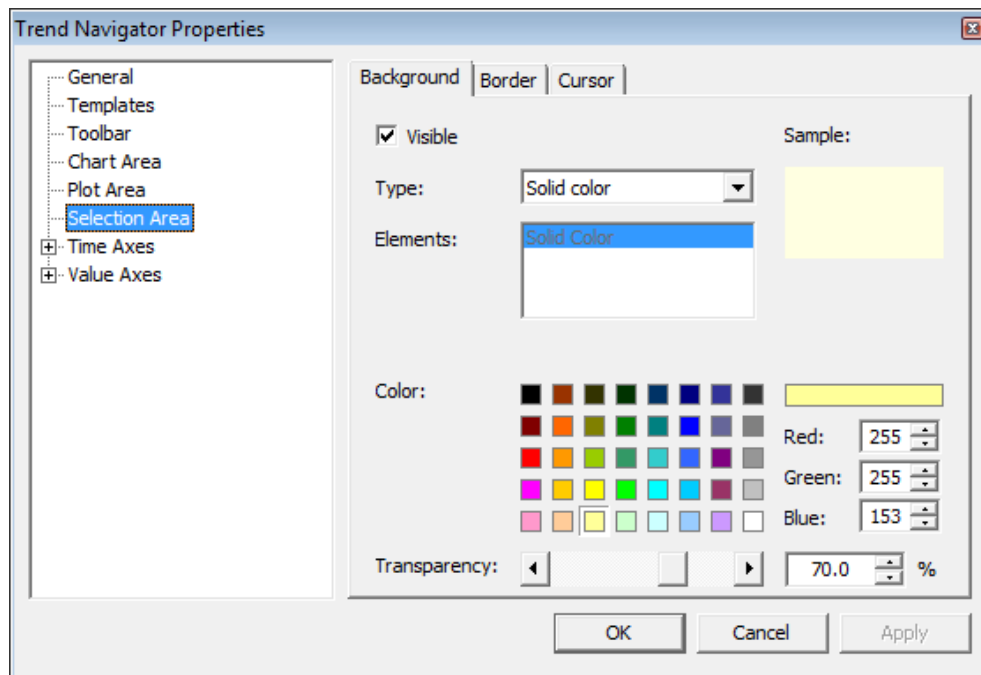
To a large extent, The TrendNavigator has the same graphical capabilities as the TrendChart control. Because of this reason, these will not be discussed explicitly. This chapter is focused on the unique features of the Navigator only.

Before the TrendNavigator can be used it has to be linked to a TrendChart control. See [Linking to a Chart control](#) on page 6-107 on how to do this.

### Selection Area

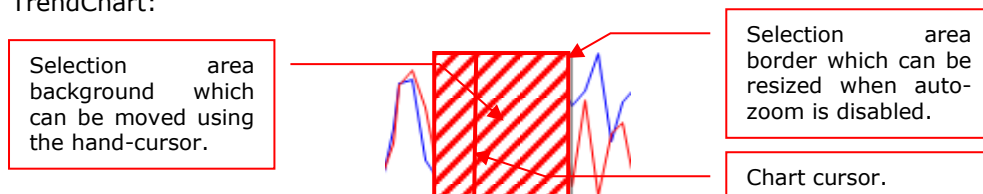
The TrendNavigator shows a transparent rectangle which represents the linked TrendChart control. The background and border of this rectangle can be changed in the properties dialog:





**Figure 6-37: Selection Area Properties**

Optionally it is possible to view the cursor of the TrendChart. This cursor is not interactive; it merely shows the current position of the cursor within the TrendChart:



**Figure 6-38: Selection Area**

## Automatic zooming

The control supports two modes of operation, automatic zooming enabled and automatic zooming disabled. The option can be changed in the properties dialog:

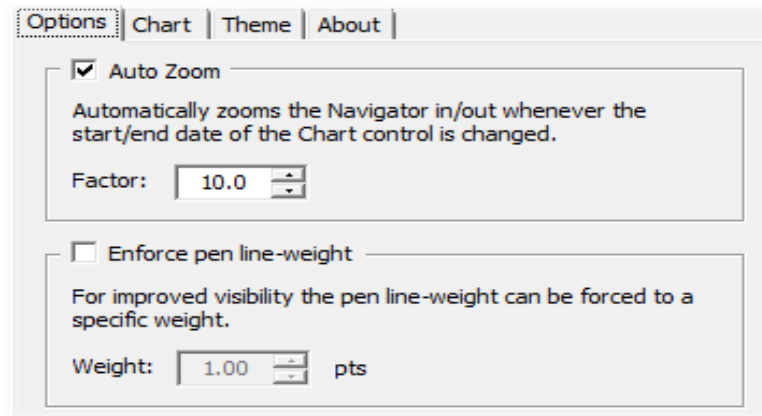


Figure 6-39: Auto Zoom Properties

When automatic-zooming is enabled, the period of the TrendNavigator is linked to that of the TrendChart. Assume that the TrendChart has a period of 1 day and that the Auto Zoom option is set to a factor of 10. This will cause the TrendNavigator to show 10 days of data. When either of the controls is zoomed, the period of the TrendNavigator will always be a factor X of the TrendChart.

When automatic zooming is disabled, the period of the TrendNavigator can be changed without affecting the linked TrendChart. This mode also makes it possible to move the left- and right edges of the chart selection rect:

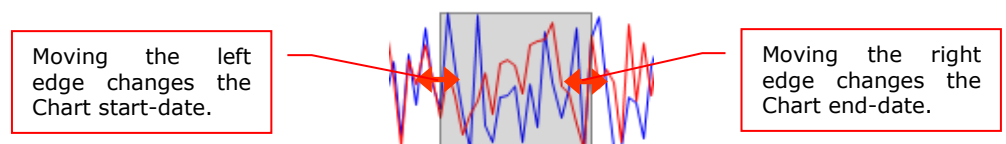


Figure 6-40: Moving the left- and right edges

## Accessing trend-data directly

Besides using the TrendChart control to visualize the trend-data it is also possible to programmatically access the trend-data. The following VBA functions are available for accessing raw trend-data:

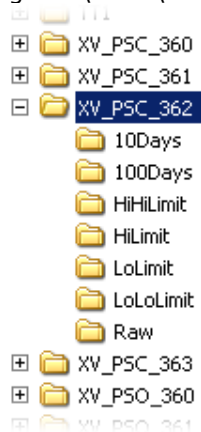
Function	Description
exTrendReadTag(...)	Reads the trend-data for a specific tag over a specific period. The result is returned as an 2 dimensional array which can be directly written to a worksheet.
exTrendReadFile(...)	Reads all the trend-data from a specific trend-file. The result is returned as an 2 dimensional array which can be directly written to a worksheet.
exTrendWriteTagToCSV(...)	Writes the trend-data for a specific tag and period to a CSV-file.

**Table 6-5: Trend Data Functions**

These functions are documented in the "Function Reference Manual".

## Data storage

Trend data is stored on disk in folders and binary files. Each tag that is configured for trending a separate folder is created in the "TrendData" folder (e.g. "C:\XLRX\TrendData"):



Each tag-folder contains 3 or more sub-folders. The sub-folders "Raw", "10Days" and "100Days" are used for storing the actual trend-values. Other sub-folders are optional and are used for storing additional data such as limits. These folders are automatically created by **eXLerate** when necessary. Within these folders, files are created with the extension .xtd (**eXLerate** Trend Data File):

Name	Size	Type	Date Modified
Y2006_M11_D8.UTC0.xtd	102 KB	eXLerate Trend Data File	11/9/2006 12:59 AM
Y2006_M11_D9.UTC0.xtd	186 KB	eXLerate Trend Data File	11/10/2006 12:59 AM
Y2006_M11_D10.UTC0.xtd	132 KB	eXLerate Trend Data File	11/10/2006 1:20 PM

The name of the file identifies the start-date of the file based on UTC (=GMT 0) and has the following layout: **Y**<year>\_**M**<month>\_**D**<day>\_**UTC**<summer-correction>.**xtd**. The year, month and day fields speak for themselves. The summer-correction field is used when **eXLerate** is configured for daylight-saving corrections.

### Reading Trend Files

Trend-data files (.xtd) may be read (from **eXLerate**) using the 'exTrendReadFile(..)' function in VBA. See the "Function Reference Manual" for a description of this function.

*This page is intentionally left blank.*

# Chapter 7 - Relational Databases

*In this chapter, you will learn how to use the embedded relational database in your application, and how to access external relational databases from your application.*

*MySQL has been selected as the embedded database engine for use in an eXLerate environment, because of its programming power, excellent performance, great support as well as its beneficial economic aspects.*

*A powerful and yet simple spreadsheet interface is available in eXLerate to define all your real-time SQL statements directly from a worksheet. Together with the extensive support for VBA, these tools give you the power to build any kind of database application.*

*Another great feature of the embedded database is redundancy support. Databases are automatically synchronized between multiple servers and it requires no engineering effort whatsoever.*

*Programming in the SQL language isn't a nightmare anymore, it's fun!*

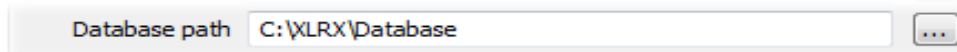
## Introduction

A database is a structured collection of data. It may be anything from a simple shopping list to a picture gallery or the vast amounts of information in a corporate network.

A *relational database* stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility, and fits nicely in the natural spreadsheet concept of rows (records) and columns (fields).

## The embedded database

eXlerate uses MySQL 4.1 as its embedded database engine. Configuration is very easy; just specify the database path in the application shortcut of the Control Center and you're ready to go:



**Figure 7-1: Embedded Database Path**

The embedded database will be created automatically if it doesn't already exist.

### Table layout

Initially the database consists of only 2 tables. The 'events' and the 'command\_log' tables. The 'command\_log' table is used internally by eXlerate to keep multiple databases synchronized, which is discussed later on in this document.

The 'events' table is used to store the alarms & events history. It has the following layout:

Column	Data Type	Description
ID	BIGINT	Unique ID of the event.
DATETIME	DATETIME	Date and time the event was stored.
CLASS	VARCHAR(32)	Event class (e.g. 'Alarms', 'Security', 'Parameter')
TYPE	VARCHAR(32)	Event type (e.g. 'Ack', 'Logoff', 'Change')
LOCATION	VARCHAR(64)	Location (e.g. 'FC01_PRESRVIN_LOFAULT', 'MSC-A')
USER	VARCHAR(64)	Name of the user or 'System'
MESSAGE	VARCHAR(255)	Event message
EXTRA1	VARCHAR(128)	Additional event field 1
EXTRA2	VARCHAR(128)	Additional event field 2

**Table 7-1: Embedded database Event table**

The primary-key is defined for column 'ID', which means that the table will never have multiple records with the same 'ID'. Furthermore, an index is defined on column 'DATETIME'. This index ensures the fast operation of queries when sorting records by date-time.

## Database Identifiers

Every database in **eXlerate** is identified by a unique database ID. The embedded MySQL database can be accessed through the unique database ID '0'. The '0' database is however a special case. It is designed to ease the development of client/server systems, thus it has a different meaning in different scenarios.

The following scenarios can be distinguished:

- Stand-alone. When the '0' database is used on a stand-alone server, it identifies the local embedded database.
- Duty server. Same as the stand-alone server, '0' identifies the local embedded database.
- Standby server. The '0' database identifies the embedded database on the duty server.
- Client. Same as the standby-server, '0' identifies the embedded database on the duty server.

There is 1 more database ID available. It should normally not be used, but is provided in the interest of a complete reference list:

Database ID	Description
0	See scenarios described above.
Local_embedded	Always accesses the database on the local computer.

**Table 7-2: Embedded database ID's**

All other database identifiers automatically refer to external databases configured using the worksheet function 'exConfigureDatabase(...)' or the worksheet 'xExtDB'.

## User definable tables

**eXlerate** can be extended with user definable tables. MySQL stores tables in the form of files. This makes it very easy to copy tables from one database to another. A single table consists of the following 3 files:

File-name	Description
<table>.FRM	MySQL Table & Column definitions File
<table>.MYD	MySQL MyISAM Data File
<table>.MYI	MySQL MyISAM Index File

**Table 7-3: MySQL table files**



These tables can be created using MySQL 4.1, which can be downloaded freely at <http://www.mysql.com>. MySQL stores these files in its default data folder (e.g. "C:\Program Files\MySQL Server 4.1\Data\<database>\") unless specified otherwise.

When copying table files from one database to another, make sure that both databases are stopped. In case of the embedded eXlerate database, make sure the application is completely shutdown. When copying to or from an external MySQL 4.1 database, make sure the MySQL-service is completely stopped.

During startup of eXlerate, the Event Logger will show an event for every user-defined table that has been found.

On redundant systems, user-definable tables are automatically synchronized between the servers. Upon startup each server should contain the same user-defined tables with the same layout. After synchronization, the data in the tables on all servers will be identical.

### Driver specific info values

The embedded database contains a set of driver specific value which can be obtained using the worksheet function 'exSQLLastDriverSpecificInfo(...)' and the VB function 'SQLCmd.GetDriverSpecificInfo(...)'. The following info values are supported by the embedded database:

Info value	Description
mysql_error	MySQL specific error-code.
mysql_time	Time in seconds it took MySQL to execute the SQL statement.
thread	ID of the thread that executed the SQL statement.
affected_rows	Number of rows that have been affected. For instance, when executing an INSERT or UPDATE statement, this info-value can be used to determine how many rows have been affected (e.g. changed by the UPDATE-statement).

**Table 7-4: Embedded database driver specific info values**

### Redundancy & Synchronization

When multiple servers are used, the database is automatically synchronized between these servers. Synchronization is very important in order to achieve a solid redundancy implementation.

Two types of synchronization can be distinguished:

- Initial synchronization. Upon startup a server determines whether it is the duty server or not. If not, it will copy the complete database from the duty server over its own database and show the 'Synchronization' window. After this, both databases will be identical.
- Run-time synchronization. After the database has been completely synchronized, it is constantly kept up-to-date. This is done using a very fast and secure mechanism.

Because the standby server(s) always have an up-to-date database, failure of the duty server will not cause any data-loss. At this point the standby server would become duty and start using its own database. Any connected clients will automatically switch to the database on the new duty server. Clients do not need to and cannot configure an embedded database. Instead, they use the database on the duty server.

## Advanced settings

By default, the embedded database is configured for use in both large and small systems. In most scenarios the default settings should provide optimal performance. These defaults can however be changed in order to fine-tune the application.

Changing these settings can be done using the 'exSetDatabaseProperty(...)' worksheet function. The following properties are supported by the embedded database:

Property	Default	Description
concurrent_connections	5	Maximum number of connections that should be used when accessing the embedded database. Increasing this value may improve the ability to process more SQL statements in parallel.

**Table 7-5: Embedded database driver properties**

### Example:

```
=exSetDatabaseProperty( "local_embedded",  
    "concurrent_connections", 10, xEvent1.Trigger )
```

## ***Corruption & data loss***

In certain situations it is possible that database corruption occurs. The most common reason is a sudden power failure. In that case the MySQL database will not be able to flush all the data to disk and close the tables. Another common reason, is manually copying table-files while the database is still running.

When **eXlerate** starts it will automatically try to repair any corrupted database tables. This may cause some loss of data, depending on when the last flush was executed. These events are logged to the Event Logger upon startup. If for some reason, the database corruption could not be repaired, please contact your **eXlerate** representative.

When using a single server, data loss can never be completely prevented in case of a system failure. To increase reliability, the use of multiple servers is advised. Using 2 servers is commonly considered a good redundant solution and will ensure proper system operation in case of single point of failure.

## ***Troubleshooting***

The embedded MySQL database writes any critical errors and warnings to a log file located in the database root folder. The log-file has the following file-name: **<computer name>.err** (e.g. "NB01.err"). The file can be opened in any ASCII-viewer such as notepad. Make sure the **eXlerate** application is completely shutdown before opening the file, otherwise an error message is shown that the file cannot be accessed.

## External databases

### Configuration



This chapter does not cover the setup and configuration of 3<sup>rd</sup> party database platforms. Instead, it describes the tools supported by **eXlerate** in order to interface with these 3<sup>rd</sup> party databases.

External databases can be configured using two worksheet functions: 'exConfigureDatabase(...)' and 'exSetDatabaseProperty(...)'.

The easiest way of configuring an external database, is to use the 'xExtDB' worksheet. This worksheet provides a template for configuring external databases and can be found in the 'MyDatabase' sample application.

	A	B	C	D	E	F	G	
1	External Database Table (version 0.1)							
2	External Database Properties							
3								
4	ID	Type	Host	Database	User	Password	Port	Co
5	1	mysql	localhost	eXlerate	root	vanish	3306	
6	2							
7	3							

**Figure 7-2: External Database Table**

All databases in **eXlerate** can be accessed through Database ID's. When configuring an external database, a unique ID has to be chosen for the database. These ID's must range from 1..n.

The database type specifies the driver which will be used to communicate with the external database. In the example above, the 'mysql' driver is used to communicate with a MySQL 4.1 database.

All other properties such as 'Host', 'Database', 'User', etc. are driver specific and are explained in the specific driver sections further on in this document.

### MySQL Database Driver

#### Introduction

MySQL is one of the leading database engines in existence. With the introduction of MySQL 5.0 it also supports Triggers & Stored Procedures. **eXlerate** itself uses MySQL internally as its embedded database engine.

MySQL is freely available and can be downloaded at the following URL: <http://www.mysql.com>.

### Supported versions & platforms

The MySQL driver can be used to access external MySQL databases. It supports the following versions of MySQL (other versions and platforms may also work but have not been tested):

Version	Platform
MySQL 4.1	Windows (x86)
MySQL 5.0	Windows (x86)
MySQL 5.1	Windows (x86)
MySQL 5.5	Windows (x86)

**Table 7-6: Supported MySQL versions**

### Configuration properties

In order to configure an external MySQL database, the following set of properties is supported. These can be configured by either using the 'xExtDB' worksheet or by using the 'exSetDatabaseProperty(...)' worksheet function directly.

Property	Default	Description
host	-	Address of the computer where the MySQL server is located. This can be either the computer-name or an IP-address.
User	-	Name of the user used to login into MySQL. (Please note that MySQL supports a comprehensive security system, which requires all computers that will be accessing the MySQL server to be configured in MySQL.)
password	-	Password that is used to login into MySQL.
Port	-	Port that is used to communicate with MySQL. By default when MySQL is installed it uses port 3306.
Database	-	Name of the database to access at the MySQL server.
Concurrent_connections	5	Maximum number of connections that should be used when accessing the MySQL database. Increasing this value may improve the ability to process more SQL statements in parallel.

**Table 7-7: MySQL driver properties**

**Driver specific info values**

The MySQL database contains a set of driver specific value which can be obtained using the worksheet function 'exSQLLastDriverSpecificInfo(...)' and the VB function 'SQLCmd.GetDriverSpecificInfo(...)'. The following info values are supported:

Info value	Description
mysql_error	MySQL specific error-code.
Mysql_time	Time in seconds it took MySQL to execute the SQL statement.
Thread	ID of the thread that executed the SQL statement.
Affected_rows	Number of rows that have been affected. For instance, when executing an INSERT or UPDATE statement, this info-value can be used to determine how many rows have been affected (e.g. changed by the UPDATE-statement).

**Table 7-8: MySQL database driver specific info values**

## SQLServer Database Driver

### Introduction

SQLServer is one of the leading database engines in existence. eXlerate uses OLEDB to access the SQLServer.

### Supported versions & platforms

The SQLServer driver can be used to access external SQLServer databases. It supports the following versions of SQLServer (other versions and platforms may also work but have not been tested):

Version	Platform
SQLServer 2000	Windows (x86)
SQLServer 2005	Windows (x86)
SQLServer 2008	Windows (x86)

**Table 7-9: Supported SQLServer versions**

### Configuration properties

In order to configure an external SQLServer database, the following set of properties is supported. These can be configured by either using the 'xExtDB' worksheet or by using the 'exSetDatabaseProperty(...)' worksheet function directly.

Property	Default	Description
host	-	Address of the computer where the SQLServer server is located. This can be either the computer-name or an IP-address.
user	-	Name of the user used to login into SQLServer. (Please note that SQLServer supports a comprehensive security system, which requires all computers that will be accessing the SQLServer server to be configured in SQLServer.)
password	-	Password that is used to login into SQLServer.
database	-	Name of the database to access at the SQLServer.
concurrent_connections	5	Maximum number of connections that should be used when accessing the SQLServer database. Increasing this value may improve the ability to process more SQL statements in parallel.

**Table 7-10: SQLServer driver properties**

### Driver specific info values

The SQLServer database contains a set of driver specific value which can be obtained using the worksheet function 'exSQLLastDriverSpecificInfo(...)' and the VB function 'SQLCmd.GetDriverSpecificInfo(...)'. The following info values are supported:

Info value	Description
Oledb_error	OleDb specific error-code.
oledb_time	Time in seconds it took the OleDb driver to execute the SQL statement.
thread	ID of the thread that executed the SQL statement.
affected_rows	Number of rows that have been affected. For instance, when executing an INSERT or UPDATE statement, this info-value can be used to determine how many rows have been affected (e.g. changed by the UPDATE-statement).

**Table 7-11: SQLServer database driver specific info values**



## SQL worksheet functions

### Introduction

The SQL interface in **eXlerate** can be fully configured from an Excel worksheet, i.e. no VBA Sub routines are required to connect to a database or to retrieve data from it. However, in some cases it makes more sense to access the database using VBA, this is described in detail in the next section.

This section will cover the usage of the SQL worksheet function in order to read and write from and to a database.

A worksheet is highly suitable for representation of a relational database, where data is organized in records and fields. Using the worksheet functions, these records and fields directly translate into rows and columns, respectively.

### Queries

After a database has been configured, it can be accessed using so-called 'Queries'. A 'Query' in **eXlerate** translates directly to a SQL statement executed on a database.

The following worksheet functions are available for SQL Queries:

Function	Description
exSQLCreateQuery(...)	Create a query with a specific ID and options.
exSQLExecQuery(...)	Execute the query, and return any result set of the query to the worksheet in rows and columns. This is the 'workhorse' function actually executing SQL statements.
exSQLExecRangeQuery(...)	Execute a query, which is based on a range of values rather than a single value. Fields for a single record have a user-definable name in which an array index is used, e.g. VAL00, VAL01, VAL02 to write a complete array in the database using a single update function.
exSQLExecRecordQuery(...)	Execute a query, which is based on a complete database record rather than on a single value. Each field in the record is named in an argument.
exSQLField(...)	Returns a single field of the result set of the corresponding query.
exSQLRowCount(...)	Returns the number of rows returned by the corresponding query.

Function	Description
exSQLColumnCount(...)	Returns the number of columns returned by the corresponding query.
exSQLLastError(...)	Retrieve the last error of the corresponding query, either as a number or a textual description.
exSQLLastDurationTime(...)	Return the time it took to execute to corresponding query.
exSQLLastDriverSpecificInfo(...)	Returns a driver specific value of the corresponding last executed query. This function can be used to obtain for instance, the MySQL specific error-code if accessing a MySQL database server.
exSQLDiagnosticalValue (...)	Returns a diagnostical value of the corresponding query. For instance, this function can be used to determine how many times a query has been executed. Such information can be critical when optimizing an application so that no unnecessary queries are executed.

Table 7-12: Implemented SQL query functions

## Views

A 'View' in **eXlerate** is a view-port on a previously executed SQL Query.

Consider the following metaphor. A book contains more than 1000 pages. When reading it, it is only possible to view 2 pages at once. All the information of the book is at your fingertips, but you can only access a small portion at any given time.

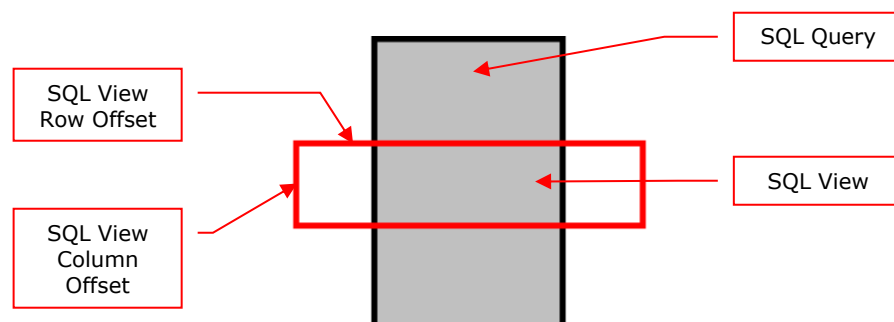


Figure 7-3: SQL query &amp; view relation

SQL queries sometimes produce large result sets. Such queries can take a lot of time, so minimizing the number of times they are executed can be very important. Such a large result set will usually not fit entirely on the screen. In that case a SQL View can be used to show only a section of the data. In order to scroll through the data, a scrollbar can be used which modifies the offsets of the SQL View. Scrollbars are explained in detail further on in this chapter.

The following worksheet functions are available for SQL Views:

Function	Description
exSQLViewQuery(...)	Views a selection of a previously executed SQL query. The selection can be specified by using row- and column-offsets.

**Table 7-13: Implemented SQL view functions**

## SQL Table

The easiest way of configuring SQL queries, is to use the 'SQL Table'. This table is located on the 'xSQL' worksheet. This worksheet is located in the 'MyDatabase' sample application:

A	B	C
1		SQL Table (Version 1.0)
2		SQL Properties
3		
4	ID	Database SQL Statement
5		Alarm History
6	1	0 select datetime from events order by datetime desc limit 1
7	2	0 select datetime,type,location,user,message from events where datetime <= '2005-11-04 00:00:00' and datetir
8		Test

**Figure 7-4: SQL Table**

The 'SQL Table' consists of the following configurable 'SQL Properties':

Field	Description
ID	Unique ID of a query. When an entry in the SQL Table is configured, a query with that particular ID is created. The 'exSQLViewQuery(...)' worksheet function can be used to access the result of a SQL statement.
Database	Database ID (0 = Embedded, 1..n = External). See 'exSQLExecQuery(...)' worksheet function for more details.
SQL Statement	SQL Statement to execute (cannot exceed 255 characters). See 'exSQLExecQuery(...)' worksheet function for more details.

Field	Description
Update Interval	Interval in seconds the query will be executed. Set this value to '0' if the query should only be updated when the SQL statement text itself changes.

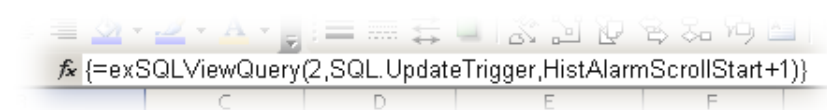
**Table 7-14: SQL Table property fields**

When an SQL statement is executed, its status information is shown in the 'SQL Results' part of the 'SQL Table':

Field	Description
Error Code	Returns the error code of the last executed SQL statement. See 'exSQLLastError(...)' worksheet function.
Error Description	Returns the error description of the last executed SQL statement. See 'exSQLLastError(...)' worksheet function.
Row Count	Returns the number of rows returned by the last executed SQL statement. See 'exSQLRowCount(...)' worksheet function.
Column Count	Returns the number of columns returned by the last executed SQL statement. See 'exSQLColumnsCount(...)' worksheet function.
Time	Returns the time in seconds the last executed SQL statement took to execute. See 'exSQLLastDurationTime(...)' worksheet function.
Execute Count	Returns the number of times the SQL statement has actually been executed. See 'exSQLDiagnosticalValue(...)' worksheet function.

**Table 7-15: SQL Table result fields**

After a SQL statement has been configured using the 'SQL Table' it can be accessed using the 'exSQLViewQuery(...)' worksheet function. This function returns the results of a query in the form of an array.

**Figure 7-5: Example exSQLViewQuery(..) worksheet function**

The first argument of the function identifies the query ID as configured in the 'SQL Table'. The other arguments of the function are explained in detail in the next section.

## Scrollable views

Views can be used in combination with scrollbars to create scrollable views. The row- or column count of a query can be used to set the boundaries of a scrollbar. The scrollbar can then be used to determine an offset for the view on the query.

The worksheet 'ScrollViews' in the 'MyDatabase' project contains working examples of scroll views.

Before configuring a scrollable view, the following condition must be met:

- The SQL query has been configured, either by using the 'SQL Table' or directly by using the 'exSQLCreateQuery(...)' and 'exSQLExecQuery(...)' functions.

### Step 1 : Create worksheet variables

A small set of worksheet variables needs to be created, most of which are self-explanatory:

Variable	Value	Description
Scrollbar current position	0	Current position of the scrollbar (updated by Scrollbar)
Scrollbar window size	10	Number of visible rows in the view
Query ID	3	ID of the SQL query
Query row count	0	Number of rows returned by the query

=exSQLRowCount(  
<QueryID>,<Update Triqquer>)

Figure 7-6: Creating worksheet variables for scrollable view

**Step 2 : Create view array using exSQLViewQuery(...)**

Select the range where the scollable text should appear:

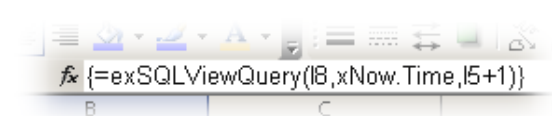
ID	Category	Name	Vendor	Vendor ID

**Figure 7-7: Select scrollable range**

And enter the following formula for the range:

```
=exSQLViewQuery( <Query ID>, <Update Trigger>,
  <Scrollbar current position> + 1 )
```

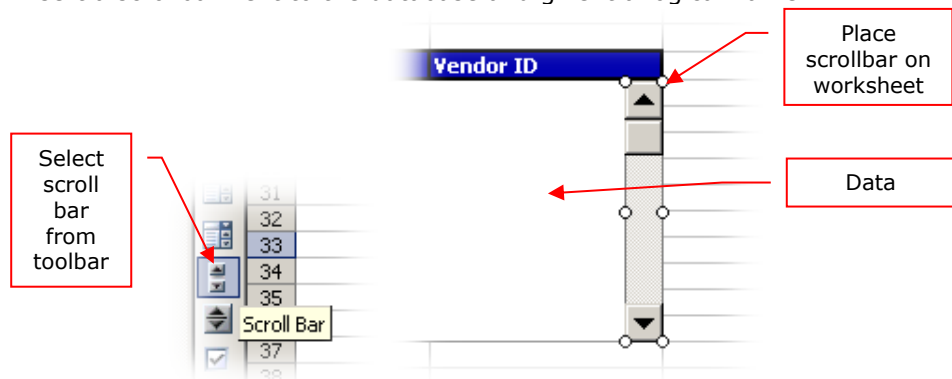
Because, the formula spans multiple cells, 'Ctrl+Shift+Enter' must be used to accept the formula.

**Example:**

**Figure 7-8: Scrollable range formula**

**Step 3 : Create a scrollbar next to the data**

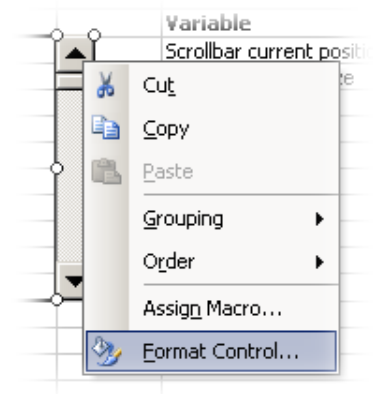
Insert a scrollbar next to the database and give it a logical name.



**Figure 7-9: Creating a scrollbar**

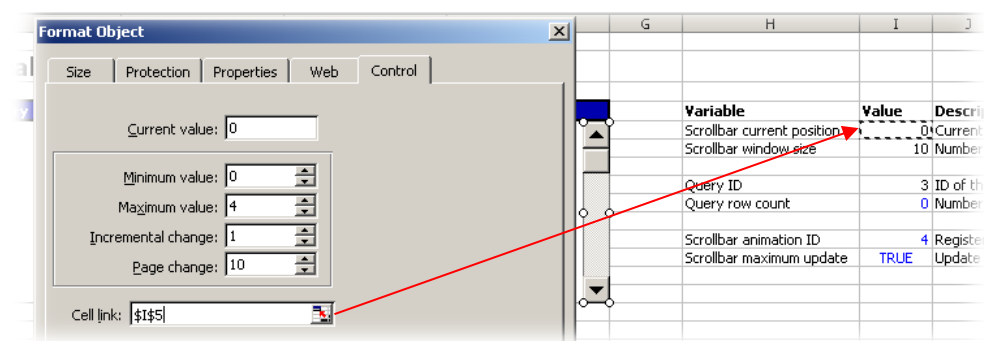
**Step 4 : Attach scrollbar to current position variable**

Right-click the scrollbar and select 'Format Control...'.



**Figure 7-10: Formatting the scrollbar**

On the 'Control' tab attach the 'Cell link' to the 'Scrollbar current position' value.



**Figure 7-11: Attach scrollbar to variable**

**Step 5 : Update the scrollbar boundaries**

Queries can return result sets with changing row counts. This means that the maximum value of a scrollbar needs to be updated whenever a query returns a different row-count. Unfortunately, updating the boundaries of a scrollbar (min/max) is not so straight-forward as updating the current position (see above). The easiest way is to create an animation object and use the 'exShapeMinMax(...)' worksheet function to update the maximum value of the scrollbar:

View ID	SQL	SQL view
Query row count	5	Number of rows returned
Animation ID	1	Registers the scrollbar as an animation object
Set scrollbar maximum	TRUE	Update scrollbar maximum when row-count changes

=exShapeID(  
 <Scrollbar name>,  
 xAutoRecalc)

=exShapeMinMax( <Shape ID>,  
 <Minimum>,<Maximum> )

**Figure 7-12: Update scrollbar maximum value**

The minimum-value should remain '0'. The maximum value is calculated as follows:  $\text{<maximum value>} = \text{<query row count>} - \text{<scrollbar window size>}$ .

**Step 6 : Increase scrollbar response (Optional)**

This step is optional and is not required in order to create a fully functional scroll view.

When the slider on a scrollbar is changed, it may take a little while before the view is actually updated. This is because the view is only updated when a re-calculation is executed, which is normally done once every second. To improve the view-response, the scrollbar has to call the following VBA macro whenever its selection changes.

The following VBA macro should be assigned to the scrollbar:

```
...
Sub ManualRecalc
    exRecalc
End Sub
...
```



## SQL VBA functions

### The 'SQLCmd' object

The 'SQLCmd' object is the main interface for VBA to access the database. Unlike other objects such as 'Trending' and 'Comms', it has to be created explicitly before it can be used. The following example shows several ways on how to create a 'SQLCmd' object:

```
...  
` Declare variable  
Dim oSQL As SQLCmd  
...  
` Create instance of SQLCmd object  
Set oSQL = New SQLCmd  
...
```

It is also possible to combine the declaration and the creation into a single statement:

```
...  
` Declare variable and create it  
Dim oSQL As New SQLCmd  
...
```

When an SQL statement is executed either by using the function 'Execute(...)' or 'ExecuteAsync(...)', the results are stored inside the 'SQLCmd' object. These can be accessed using the other properties and functions of the object. For instance, in order to detect whether the SQL statement was executed successfully, the properties 'ErrorCode' and 'ErrorDescription' can be used.

### Executing SQL statements

This section describes the use of 'synchronously' executed SQL statements. Asynchronously executed SQL statements are explained further on in this document.

The function 'Execute(...)' executes a SQL statement on a database. The first argument specifies the database ID (0=embedded database, 1..n=external database). The second identifies the SQL statement to execute and the third identifies the timeout-value in milliseconds.

The following example executes a SQL statement on the embedded database and waits for a maximum time of 10 seconds for it to complete:

```
...  
` Declare variables  
Dim oSQL As New SQLCmd  
Dim lErr As Long  
...  
` Execute SQL statement  
lErr = oSQL.Execute("0","SELECT COUNT(ID) FROM EVENTS",10000)  
...
```

The return value ('lErr') will contain '0' if the statement was successfully executed. If an error has occurred, the properties 'ErrorCode' and 'ErrorDescription' can be used to determine the reason of the error.

### ***Reading SQL results***

After a SQL statement has been successfully executed, the results can be accessed using the 'Field' property. The arguments for this property identify the row- and column indexes within the result-set. Both are 0-based, so the first element must be accessed using index '0'. The following example shows the use of the 'Field', 'RowCount' and 'ColumnCount' properties:

```
...  
` Declare variables  
Dim lRow As Long  
Dim lColumn As Long  
Dim dSum As Double  
...  
` Iterate through all the fields and count the values  
dSum = 0.0  
For lRow = 0 To oSQL.RowCount - 1  
  For lColumn = 0 To oSQL.ColumnCount - 1  
    dSum = dSum + oSQL.Field( lRow, lColumn )  
  Next  
Next  
...
```

## Writing SQL results to worksheets

When a 'SQLCmd' object contains any data, these results can be written to a worksheet. The easiest way is to copy the 'SQLCmd.Field' property directly to a worksheet cell:

```
...  
oSheet.Range( "A1" ).Value = oSQLCmd.Field( 3, 4 )  
...
```

This works just fine for a couple of cells but can have a major performance impact when writing large amounts of cells. In that case the 'GetArray(...)' function should be used instead. The 'GetArray(...)' function returns a 2-dimensional array containing all the fields in the 'SQLCmd' object. This array can be assigned to a worksheet range in a single function call, which makes it extremely fast:

```
...  
Dim vArray as Variant  
...  
' Copy all the fields into the array  
vArray = oSQLCmd.GetArray()  
...  
' Copy the array to the range  
oSheet.Range( "A1:C1000" ).Value = vArray  
...
```

## Executing a-synchronous SQL statements

### Introduction

Executing SQL statements a-synchronously is a very powerful tool in order to boost performance and increase responsiveness of your application. Especially when executing multiple SQL statements, using the a-synchronous functions can have a profound impact on the performance. The following example shows 5 SQL statements, executed synchronously:

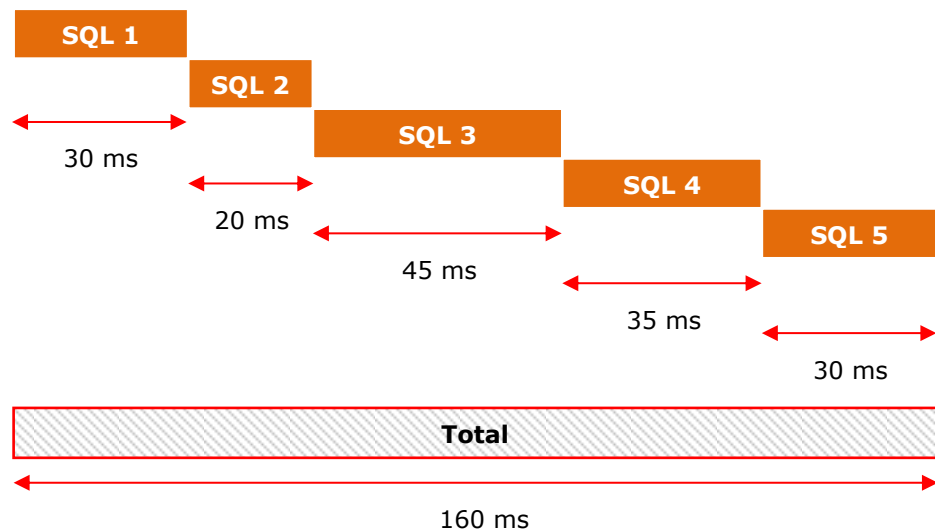


Figure 7-13: Synchronously executed SQL statements

When the same SQL statements are executed a-synchronously the following behavior arises:

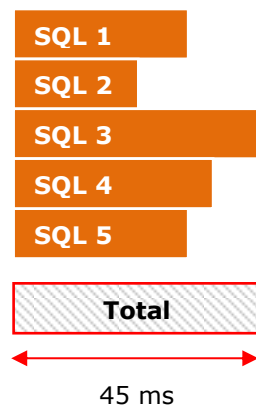


Figure 7-14: A-synchronously executed SQL statements

When executed a-synchronously the total time of execution will depend on the slowest SQL statement in the batch.

The example above shows an 'ideal' situation. In such a situation all SQL statements are executed in parallel without any performance cost. In reality, the hardware (CPU, hard-drive, network) will have more to do at the same time, which has some performance impact. Today's generation of computers is however increasingly capable of parallel execution (Hyper Threading, Multi-core CPU's) and thus using a-synchronous SQL statements will ensure that you get the best performance out of your hardware.

Another important criterion for using a-synchronous SQL statements is responsiveness. Imagine that a single SQL statement is executed which takes 30 seconds to complete. When executed synchronously, it will block the execution of the application for that period. Using the a-synchronous functions will not make it go faster, but when used correctly, will ensure that the application doesn't become blocked. It is also possible to cancel a-synchronous SQL statements when necessary.

### Executing multiple a-synchronous SQL statements

Executing multiple a-synchronous SQL statements can be done using both the 'SQLCmd' and 'SQLCmdBatch' objects.

The 'MyDatabase' sample application contains a working example on a-synchronous SQL statements (Form 'frmASyncSample').

The following a-synchronous functions are available for the 'SQLCmd' object:

Name	Description
ExecuteAsync	Executes an SQL statement on a database and returns immediately.
WaitForAsync	Can be called after 'ExecuteAsync'. Waits x milliseconds for the a-synchronous operation to complete. If the a-synchronous operation is completed before x milliseconds, the function returns immediately. This function can be called multiple times.
CancelAsync	Cancels the last SQL statement executed by 'ExecuteAsync'.
IsAsyncInProgress	Returns 'True' when an a-synchronous operation is still in progress.

**Table 7-16: SQLCmd a-synchronous functions**

The following example demonstrates the use of the 'SQLCmdBatch' object in combination with the a-synchronous functions of the 'SQLCmd' object:

```
...
` Declare variables
Dim oSQL1 As New SQLCmd
Dim oSQL2 As New SQLCmd
Dim oSQL3 As New SQLCmd
Dim oSQLBatch As New SQLCmdBatch
Dim lErr As Long
...
` Add SQL objects to the batch
oSQLBatch.Add oSQL1
oSQLBatch.Add oSQL2
oSQLBatch.Add oSQL3
...
` Execute SQL statements
oSQL1.ExecuteAsync "0", "SELECT ID FROM EVENTS"
oSQL2.ExecuteAsync "0", "SELECT MESSAGE FROM EVENTS"
oSQL3.ExecuteAsync "0", "SELECT LOCATION FROM EVENTS"
...
` Wait for all statements to complete. Let Excel
` process any events every 100 milliseconds.
lErr = oSQLBatch.WaitForAsync(100)
While lErr <> 0
    DoEvents
    lErr = oSQLBatch.WaitForAsync(100)
Wend
...
` Check the error codes of the SQL command objects
If oSQL1.ErrorCode <> 0 Then ...
If oSQL2.ErrorCode <> 0 Then ...
If oSQL3.ErrorCode <> 0 Then ...
...
```

The example shown above is one of many techniques which can be used for a-synchronous SQL statements. It is also possible to use Timers or delays to construct wait-loops.

*This page is intentionally left blank.*

# Chapter 8 - Client & Server

*In this chapter, you will learn how to add client/server support to your application.*

*Using multiple servers is discussed in 0 Redundancy.*

## Introduction

Three types of computers can be identified: Duty (server), Standby (server) and Client in an **eXlerate** environment. When using **eXlerate** to develop client/server systems it is not necessary to create separate applications for these different types of computers. Instead, a single application will serve all these different functions.

From the application engineer point of view, a client application performs a lot of functions the server also performs. For instance, it visualizes the data and allows user interactions. But it doesn't communicate directly with an IO device or generate reports periodically. A client can therefore be considered a "subset" of a server. All the functions it doesn't perform are actually performed by the duty server and the client just reads the results from the duty server. From the end-user point of view, the duty- and standby server as well as each client computer provide full operator functionality.

Most of this "conditional" behavior is done by **eXlerate** internally, such as generating reports periodically. Application specific functionality should however be implemented conditionally. For instance, when periodic reports are generated manually, the VBA code in question should only be run on the duty server.



## Network Configuration Assistant (xNet)

All network related functionality in an application can be configured through a set of worksheet functions. In order to simplify client/server implementation a separate worksheet has been developed by our team which has all these worksheet functions pre-configured. This worksheet is called 'xNet' and is considered the primary means of configuring client/server functionality.

The 'xNet' worksheet is part of the 'MyTemplate' template application. When creating a new application, this sheet is automatically copied into the new application. If your application does not contain a 'xNet' sheet, you may manually copy it from the 'MyTemplate' application to your own application.

The 'xNet' worksheet automatically detects the installed license:

A	B	C	D	E	F	G	H	I
Network Configuration Assistant (version 1.0)						License found: Development		

**Figure 8-1: xNet license detection**

In order to use client/server functionality one of the following licenses should be installed:

- Server license: The computer can be used as a server but also as a client when needed.
- Client license: The computer can only be used as a client.
- Development license: The computer can be used either as a client or a server. The development license is however restricted and does not allow running communications for more than 1 hour at a time.



In case of a single computer no Client/Server functionality is required and the stand-alone license shall be used instead.

## Server configuration

To configure a server you should execute the following steps. These steps apply to both single server systems and multi-server systems with duty/standby selection.

### Step 1 : Configure 'xNet' worksheet

Specify the Computer Name of the server and optionally specify the IP-addresses. The 'xNet' worksheet supports up to 2 servers:

Server Configuration						
ID	Computer Name	IP #1 (optional)	IP #2 (optional)	IP #3 (optional)	IP #4 (optional)	Startup wait-time (seconds)
1	SVC-A					20
2	SVC-B					40

**Figure 8-2: Server configuration**

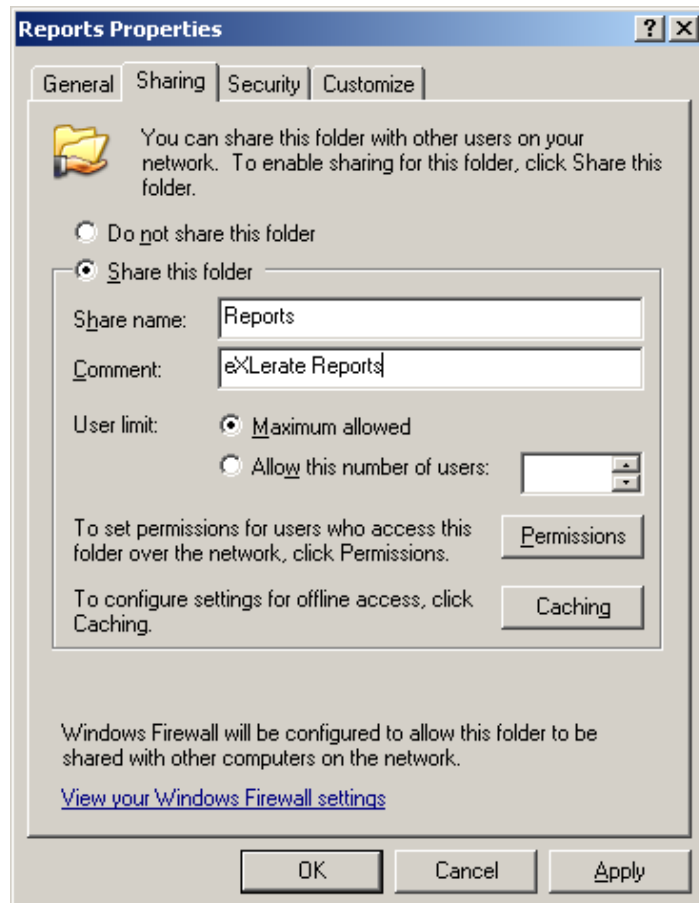
The columns can be configured as follows:

Column	Description
Computer Name	Should contain the name of the computer which is server. Note that, this is not the system-name as configured in the Control Center, but the Computer-name as configured in Windows.
IP #1	Optional IP address by which the computer can be accessed. It is highly recommended to specify an IP address explicitly. Accessing computers on a network merely by using their computer-name is not guaranteed to work. This is especially true for systems that don't use name-servers such as WINS or DNS.
IP #2..4	Optional IP addresses of redundant network cards. A server will always try to communicate over the primary connection unless it fails, then it will try connection 2, 3 and eventually 4.
Startup wait-time	Time in seconds the computer should wait at startup before switching to a particular duty mode. If multiple servers are specified, the duty wait times should be at least 10 seconds apart. This is discussed in more detail further on in this document.

**Table 8-1: Server configurable fields**

**Step 2 : Share Reports- data for clients**

This step is required when clients will connect to the server. In this case a file-share should be created for the Report folder:



**Figure 8-3: Share Report folder on server**

These are all the steps required for configuring a server. Whether the server IP address configuration is correct can be validated using the 'xNet' worksheet. Assume our server is called "SVC-A" and we are viewing the **eXlerate** application on that server. For every IP address that is configured, the status should be 'Ok', when the realtime updating has been started.

Status	Status IP #1	Status IP #2	Status IP #3	Status IP #4	
Ok	Ok	Ok			F

**Figure 8-4: Server communication statuses**

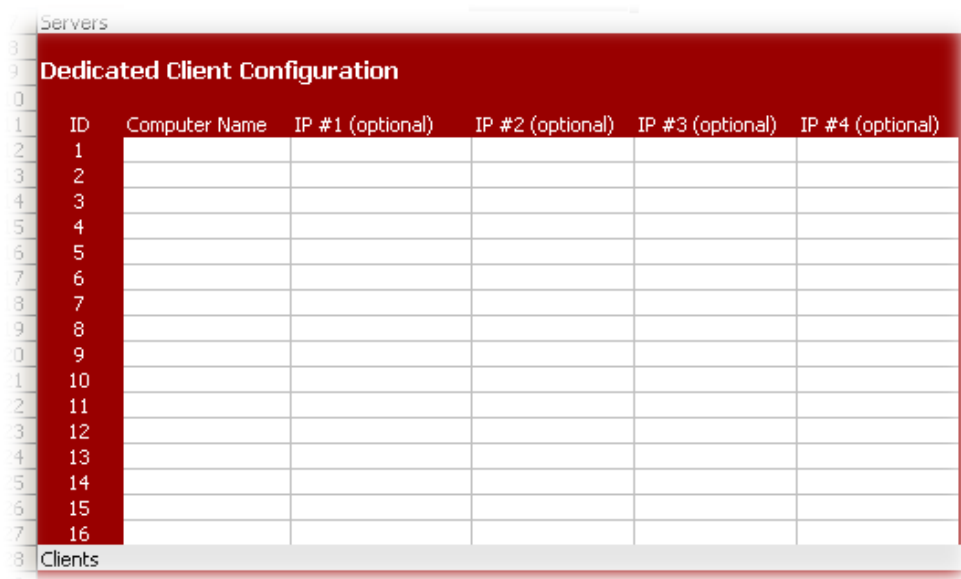
If an invalid IP address was specified, "Not connected" will be shown, which means that the server cannot start listening for incoming connections on that particular IP address because the IP address is not assigned to any network adapter.

## Client configuration

To configure a client you should execute the following steps:

### Step 1 : Configure 'xNet' worksheet

Specify the Computer Name of the client and optionally specify the IP-addresses. The 'xNet' worksheet supports up to 16 dedicated clients by default. If necessary, the number of clients can be increased by adding rows to the bottom of the table and copying the formulas to the new row(s).



ID	Computer Name	IP #1 (optional)	IP #2 (optional)	IP #3 (optional)	IP #4 (optional)
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					
16					

**Figure 8-5: Client configuration**

The term 'dedicated client' was carefully chosen to distinguish between dedicated and non-dedicated clients. A dedicated client is a client which has to be explicitly configured in the application. A non-dedicated client does not have to be configured, but can still connect to the servers and participate. A dedicated client can however be monitored by others and a non-dedicated client cannot. For instance, if an application contains an overview of all connected computers (e.g. 2 servers and 2 clients), all clients on the overview have to be dedicated so that the application can determine their status. A non-dedicated client can for instance, be a service- or engineering laptop which is only used when necessary. By putting the same application on the laptop, it can be used to access the system, effectively becoming a non-dedicated client.

These clients are each identified by a unique ID which are resp. '1..16'. Please note that client- and server ID's are not exchangeable. This means that client ID's cannot be used for functions which require server ID's such as 'exQueryServer(...)'.

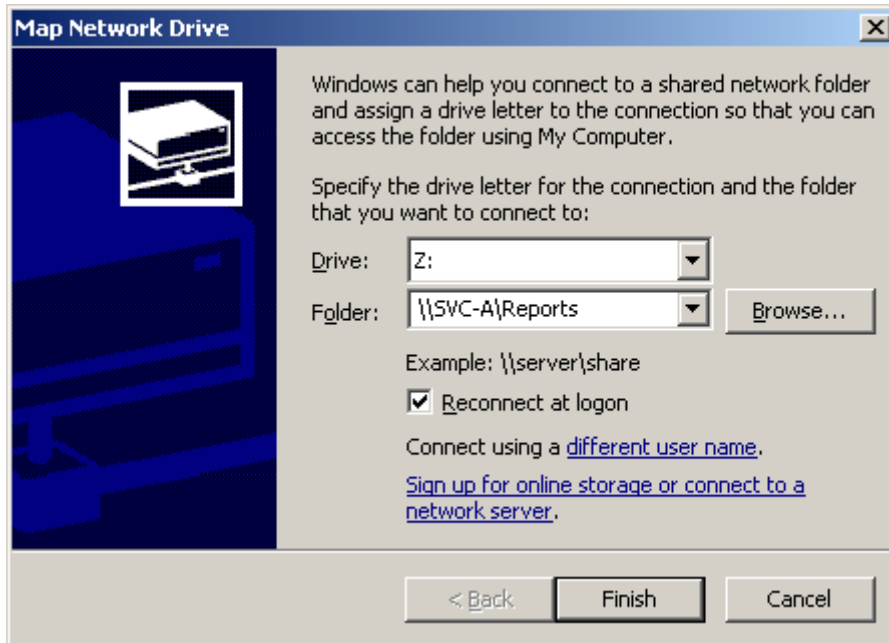
The following columns are configurable:

Column	Description
ComputerName	Should contain the name of the computer which is client. Note that, this is not the system-name as configured in the Control Center, but the Computer-name as configured in Windows.
IP #1	Optional IP address by which the computer can be accessed. It is highly recommended to specify an IP address explicitly. Accessing computers on a network merely by using their computer-name is not guaranteed to work. This is especially true for systems that don't use name-servers such as WINS or DNS.
IP #2..4	Optional IP addresses of redundant network cards. A client will always try to communicate over the primary connection unless it fails, then it will try connection 2, 3 and eventually 4.

**Table 8-2: Client configurable fields**

**Step 2 : Create network drive mappings to server(s)**

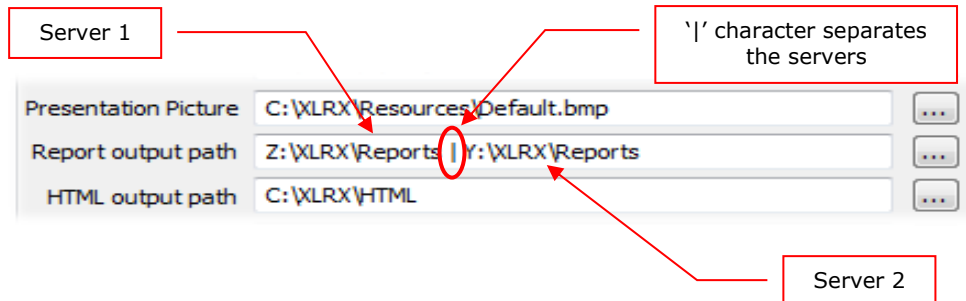
Clients cannot have an embedded database. Instead they rely on the embedded database on the duty server. The same is true for trend- and report files. Clients do not generate these files; instead they read them from the duty server. This is done by the use of a drive-mapping to the shared files on the server(s). For each server to which the client should connect, a network drive-mapping needs to be created. This has to be done for both the Reports- and Trending data:



**Figure 8-6: Client network drive mapping**

### Step 3 : Configure Control Center shortcut

The final step is to configure these network locations in the Control Center shortcut:



**Figure 8-7: Client shortcut configuration**

If multiple servers are used, each server should have its own set of network drive mappings. These drive-mappings can be separated using a '|' character as shown above. Any spaces that are used before or after the '|' character are ignored.

These are all the steps required for configuring a dedicated client. Whether the client IP address configuration is correct can be validated using the 'xNet' worksheet. Assume our client is called "CLT-1" and we are viewing the eXlerate application on that client. For every IP address that is configured, the status should be Ok.

Status	Status IP #1	Status IP #2	Status IP #3	Status IP #4
Ok	Ok	Ok		

**Figure 8-8: Client communication statuses**

If an invalid IP address was specified, "Not connected" will be shown, which means that the client cannot start listening for connections on that particular IP address because the IP address is not assigned to any network adapter.



## Advanced settings

The 'xNet' worksheet contains a separate section for advanced settings.

Setting	Value	Default	Description
Communication Port	9666	9666	Port that is used for TCP/IP co
Duty switchover period	10	10	Time in seconds the system is

**Figure 8-9: Advanced settings**

The following advanced settings are supported:

Setting	Default	Description
Communication Port	9666	TCP/IP port that is used for communicating with other clients and servers.
Duty switchover period	10	Time in seconds the system is forced in a duty selection when switching over. Also, see section 'Duty selection' in 0 Redundancy.
Duty local override	-	When a value is entered, the computer is forced in a fixed duty selection.
Time synchronization (hour)	-	Hour of the day at which periodic time synchronization between clients and server should occur. If this cell is left empty, time synchronization is performed every hour or not at all, depending on the 'Minute' setting discussed below.
Time synchronization (minute)	-	Minute of the hour at which periodic time synchronization between clients and server should occur. If this cell is left empty, no time synchronization is performed at all.

**Table 8-3: Advanced settings**

## Application development

### Names

The 'xNet' worksheet provides an elaborate set of names which can be used for application development. All the names provided by the 'xNet' worksheet are read-only.

The following global names are available:

Name	Data-type	Description
Net.Started	Boolean	TCP/IP port that is used for communicating with other clients and servers.
Net.UpdateTrigger	Date/time	Recalculation trigger which causes the 'xNet' worksheet to be recalculated.
Net.Duty.ID	Number	ID of the currently selected duty server, or '0' if no duty is selected.
Net.Duty.Name	Text	Computer name of the currently selected duty server, or 'None' if no duty is selected.
Net.Duty.SwitchOverride	Number	ID of the server which should become the new duty server. This value is written by the 'exDutySwitch' VBA function.
Net.Local.ServerID	Number	ID of the local server or '0' if the local computer is not a server.
Net.Local.ClientID	Number	ID of the local client or '0' if the local computer is not a client.
Net.Local.IsClient	Boolean	TRUE when the local computer is a client.
Net.Local.IsDutyServer	Boolean	TRUE when the local computer is the duty server.
Net.Local.WaitTimeExpired	Boolean	TRUE when the local computer is still waiting before choosing a duty server (always TRUE on a client).

**Table 8-4: xNet global names**

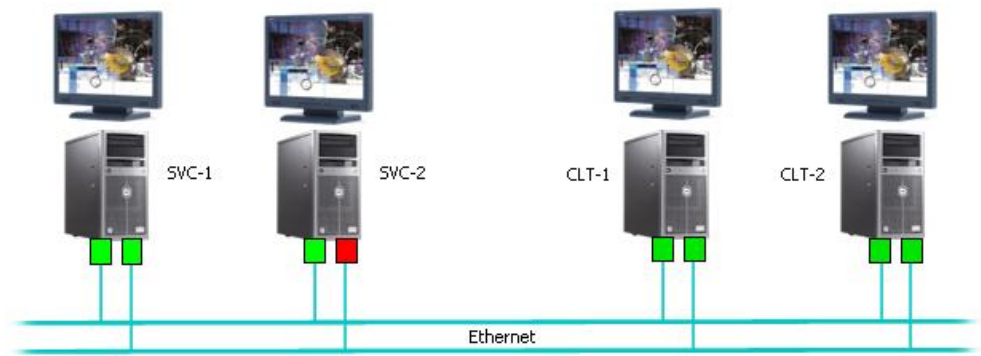
For each server and client the following set of names is available for application development:

Name	Data-type	Description
Net.Server{1..2}.ComputerName	Text	Computer name of the server.
Net.Server{1..2}.Status	Number	Overall communication status of the server. The following status values are supported: 0: Ok 1: No heartbeat received 2: Not connected 3: Not started 4: Not configured
Net.Server{1..2}.Status{1..4}	Number	Communication status of a specific redundant IP address (network adapter). See status description above.
Net.Server{1..2}.Duty	Number	ID of the duty server which is selected by that particular server, or '0' if no duty is selected by the server.
Net.Client{1..16}.ComputerName	Text	Computer name of the client.
Net.Client{1..16}.Status	Number	Overall communication status of the client. The following status values are supported: 0: Ok 1: No heartbeat received 2: Not connected 3: Not started 4: Not configured
Net.Client{1..16}.Status{1..4}	Number	Communication status of a specific redundant IP address (network adapter). See status description above.
Net.Client{1..16}.Duty	Number	ID of the duty server which is selected by that particular client, or '0' if no duty is selected by the client.

**Table 8-5: xNet Client / Server names**

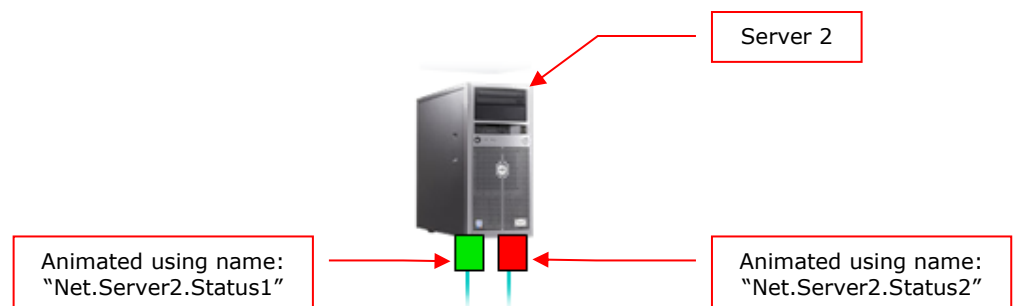
## Network overview

The following example shows a network overview of 2 servers and 2 clients. Both the servers and the clients have 2 redundant network adapters, which are represented by the green and red rectangles. Each rectangle represents the communication status of a network adapter.



**Figure 8-10: Network overview example**

The names from the 'xNet' worksheet can be used to animate the network adapters on the overview:



**Figure 8-11: Network adaptor animation**

## Conditional development

When developing a client/server application it is important to keep in mind that only the duty server should perform specific tasks, such as generating automatic reports controlling valves, etc... Since the application runs on both the servers and the clients, certain functionality should be prevented from running on anything but the duty server.

It is important to know that **all** VBA code is executed on both the servers and clients.

A set of tools is required for conditional development. These tools are provided in the form of worksheet/Visual Basic functions and the names defined on the 'xNet' worksheet.

The following functions are supported for both worksheets and VBA:

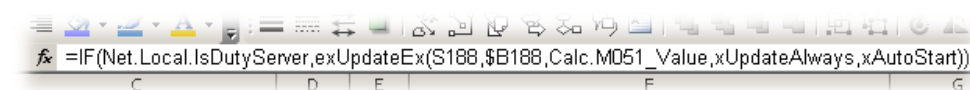
Function	Description
exIsClient	Checks whether the local system is either a client or server.
exIsDutyServer	Checks whether the local computer is the duty server or not.

**Table 8-6: Conditional development tools**



For worksheets it is recommended to not use the functions mentioned above but to instead use the 'Net.Local.IsClient' and the 'Net.Local.IsDutyServer' names as defined on the 'xNet' worksheet. The main reason for this is to improve the Excel calculation performance by minimizing trigger based functions.

Worksheet function example:



**Figure 8-12: Conditional worksheet function**

Visual basic example:

```
' Check required position of the prover 4-way valves
' but do this only the duty server
If exIsDutyServer Then
    CheckPrvValves
End If
```

## Event logging

In client/server systems all events are logged to the central database. This database is stored on the duty server and the standby server has a synchronized copy of this database. Events are also locally logged in the Control Center. These local logs are however not synchronized and differ on the individual computers. The database should be considered the primary location for storing events, not the local event log.

Many functions/mechanisms in **eXlerate** log events to the database. This is all done transparently and does not require any additional engineering effort. There are however some functions which have been tailored for client/server support. The following functions behave in a particular manner when logging events:

Function	Description
exLogChange (Worksheet)	When this function is configured to log to the database, it will only do this when executed on the <b>duty server</b> .
exStoreValue (Worksheet)	When this function is configured to log to the database, it will only do this when executed on the <b>duty server</b> .
exLogEvent (Visual Basic)	When this function is used to log to the database, it will <b>always</b> do this.

**Table 8-7: Conditional event logging**

## Writing to IO devices

Writing to IO devices is possible from both clients and servers. Using the functions 'exUpdateEx(...)', 'exUpdateStrEx(...)' and 'exUpdateVarEx(...)' it is possible to write a value to an IO device. When these functions are used on a worksheet they would be executed on all servers and clients. This would cause each server and client to write the data to the specific IO device.

To restrict the writing to the duty server the following statement can be used for worksheet writes:

### Stand-alone usage:

```
=exUpdateEx( iQuery, iItem, dValue, UpdateMode )
```

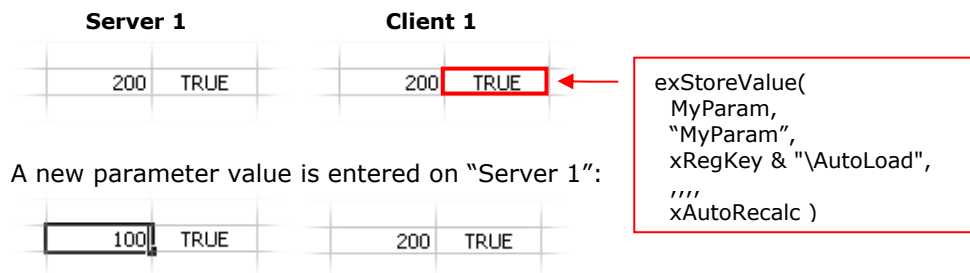
### Client/server usage; only write on duty server:

```
=IF(Net.Local.IsDutyServer, exUpdateEx( iQuery, iItem, dValue, UpdateMode ), FALSE)
```

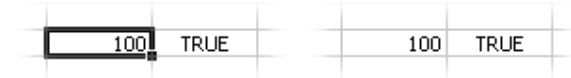
## Synchronized parameters

The 'exStoreValue(...)' worksheet function serves a dual role in client/server systems. Not only is it responsible for retentive storage of parameters to the local registry, it also synchronizes parameters between clients and servers.

Parameters are initialized on startup with the value last stored in the registry of the duty server:



The parameter value on all other servers/clients is automatically updated.



**Figure 8-13: Synchronized parameters**

Parameters can be modified on both clients and servers, after which the changed parameter is distributed to all other clients and servers.



When using the 'exStoreValue(...)' worksheet function in client/server systems it is essential that the 'Trigger' argument contains the value 'xAutoRecalc'. If this is not the case, it is possible that the parameter is not correctly initialized and thus no synchronization will happen.

## Shared values

Shared values are synchronized values which are shared by all clients and servers in a system. They can be read/written conditionally from worksheets and Visual Basic. When compared to synchronized parameters, they have the following different characteristics:

- Synchronized parameters are always written; Shared values are only written when the write-condition is 'True', otherwise they are read.
- Synchronized parameters are retentive; Shared values are not retentive.

Much like a synchronized parameters, a shared value is also identified by a unique name (e.g. "MySharedValue"). No functions are required to register the unique shared value name. Instead, the name is implicitly registered by the shared value functions.

The following example shows the use of shared values on worksheets. Assume that cell 'L30' contains a calculated value, which is different on all computers. The 'exShareValue(...)' worksheet function can be used to write that specific value on a specific system after which it can be read on all others.



**Figure 8-14: Shared value worksheet function**

The second-argument, which is the 'write-condition', compares the current computer with a specific computer-name. This will only evaluate to 'True' on computer 'SVC-1'. The return-value of the function is the shared value. On 'SVC-1', this will be always the same as value as cell 'L30', on all other computers this will be the 'L30' cell as written by 'SVC-1'.

The following example illustrates this mechanism using 'Net.Local.IsDutyServer' as the write-condition:

	SVC-1 (Duty)		SVC-2 (Standby)	
Calculated value	100	Calculated value	105	
exShareValue()	100	exShareValue()	100	exShareValue("MySharedValue", Net.Local.IsDutyServer, L30, xNow.Time)
	CLT-1		CLT-2	
Calculated value	103	Calculated value	98	
exShareValue()	100	exShareValue()	100	

**Figure 8-15: Shared value on multiple computers**

Shared values can also be used from VBA. The following Visual Basic functions are supported for reading and writing shared values: 'exGetSharedValue(...)' and 'exUpdateSharedValue(...)'. The following example illustrates the use of these functions:

```
' Write shared value on duty server and read on others
If Net.exIsDutyServer Then
    Net.exUpdateSharedValue "MySharedValue1", strValue
Else
    Net.exGetSharedValue "MySharedValue1", strValue
End If
```



*This page is intentionally left blank.*

# Chapter 9 - Redundancy

*In this chapter, you will learn how to add redundancy support to your application.*

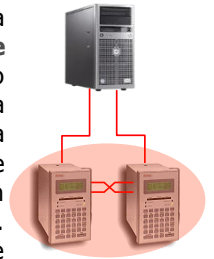
## Introduction

**eXlerate** has an extensive support for redundancy. This allows you to build fail safe systems. Since "A chain is only as strong as its weakest link", all supported levels of redundancy can be extended with multiple backups. For instance, if 2 network cards don't provide enough redundancy, additional network cards can be added in a breeze.

In a typical system, four hardware levels are subject to redundancy:

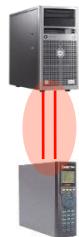
### ● Device redundancy

In case a device fails (e.g. flow computer, PLC); a backup device may take its role. Although **eXlerate** cannot play an active role in this scenario, it will have to interface with these devices. These devices will have a communication channel to the **eXlerate** server so all data is available. **eXlerate** can be configured to choose the data of a preferred device or it can calculate its own values based on the data received by multiple devices. Data of such devices may be stored in a single tag in the database using multiple columns.



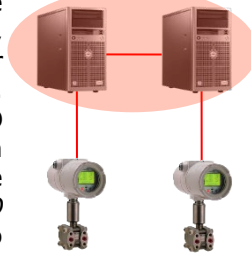
### ● Device communication channel redundancy

Some devices support redundant communication channels. This makes the device less vulnerable to broken cables, electrical problems, etc... **eXlerate** can be configured to take full advantage of this device feature. The application engineer is in full control over the flow of data. Whether the redundant communication channel is only used in the event of a failure, or used for continuous communication, it's all possible. If the communication protocol supports it, the data-flow can also be divided over multiple communication channels to increase overall bandwidth (e.g. load-balancing).



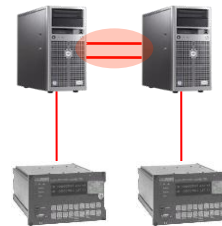
### ● Server redundancy

To protect against a server failure, one or more backup servers may be configured. All servers in an eXlerate environment are identical. A duty/standby concept is used to distinguish between the servers. Only one server is the "duty" server, the others are considered "standby". If the "duty" server fails, a standby server will take on the roll of "duty" server. The standby server(s) continuously synchronize all the data from the duty-server such as reports, averages, database, etc... These synchronized servers allow for bumpless switchovers without any hiccups. Additionally, standby servers can also be used as IO servers. If the "duty" server has a communication problem with a device, it can be configured to use the IO channel of a standby server, which is called *IO routing*. This is all possible without having to switch to another "duty" server.



### ● Network redundancy

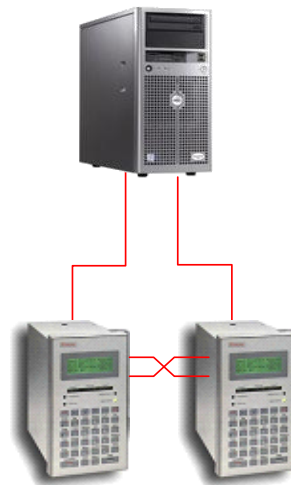
Clients and servers may be equipped with multiple network cards in order to communicate over a single or multiple networks. The configuration involves merely specifying the IP address of the network card. The rest is handled transparently by eXlerate.



## Device redundancy

### Introduction

In case a device fails (e.g. flow computer, PLC), a backup device may take its role. Although **eXlerate** cannot play an active role in this scenario, it will have to interface with these devices. These devices will have a communication channel to the **eXlerate** server so all data is available. **eXlerate** can be configured to choose the data of a preferred device or it can calculate its own values based on the data received by multiple devices.



## Configuration

Please refer to Reference Manual I, Chapter 6 – Data Communications on how to configure communications.

When configuring multiple redundant devices, each device has its own set of protocols and queries. In the tag database, there is however only a single row per redundant IO point:

Description	value	value2	value3
	0.711111128	0.039444443	0.711111128
(CF)	1.041300058	1.04429996	1.041300058
d)	1	1	1
	934.5909424	259.125824	934.5909424
(CF)	1.031999946	1.043400049	1.031999946

**Figure 9-1: Tag Database with redundant IO**

The 'Value' column represents the in-use data point. 'Value2' and 'Value3' represent the actual data received by two redundant devices. Additional devices can simply be added by adding new columns to the Tag Database.

The 'Query Table' also needs to be modified so that the tags are written to the proper column in the Tag Database:

38	1	1	100:85	10	10	50	760	8	xTagDB	64	OK	2	8501
39	1	1	100:90	10	10	50	776	8	xTagDB	64	OK	2	8601
40	1	1	100:95	10	10	50	790	8	xTagDB	64	OK	2	8701
41	1	1	500	30	10	600			OMNI\Batch\FB_%6A_%1%*[^:];%		OK	5	9101
42	1	1	500:100	30	10	600			OMNI\Prove\FP_%6A_%1%*[^:];%		OK	5	9201
43	2	1	30:0	10	10	50	170	9	xTagDB	64	OK	2	0001
44	2	1	0	10	10	50	181	9	xTagDB	64	OK	1	1501
45	2	1	50:0	10	10	50	188	9	xTagDB	64	OK	2	3101
46	2	1	50:5	10	10	50	190	9	xTagDB	64	OK	2	3201

First redundant device  
writes to column 8  
(Value2) in xTagDB

Second redundant device  
writes to column 9  
(Value3) in xTagDB

**Figure 9-2: Query Tables write to multiple columns in Tag DB**

The in use IO point (Value) can be determined using a formula. The application engineer is left free in his/her choice of this formula. For instance, a device may report whether it is the master- or slave device. In that case the following formula may be used:

The image shows an Excel formula bar with the formula `=IF(Calc.FC01.Master=1,I153,H153)`. The formula bar is highlighted, and the background shows a grid with columns C, D, and E visible.

"Calc.FC01.Master" contains the 'master' status of the device. If the first device is master; the first column is used, otherwise the second column is used.

If desired, it is also possible to calculate the average value of both devices:

The image shows an Excel formula bar with the formula `=(I153+H153)/2`. The formula bar is highlighted, and the background shows a grid with column C visible.

This would however not work if one device failed. In that case the average should only be calculated if both devices are valid. In all other cases, the valid device should be used:

The image shows an Excel formula bar with the formula `=IF(Calc.FC01B.Comms.Fail,H153,IF(Calc.FC01A.Comms.Fail,I153,(H153+I153)/2))`. The formula bar is highlighted, and the background shows a grid with columns C, D, E, and F visible.

## ***Device communication channel redundancy***

### ***Introduction***

Some devices support redundant communication channels. This makes the device less vulnerable to broken cables, electrical problems, etc... **eXlerate** can be configured to take full advantage of this device feature. The application engineer is in full control over the flow of data.

Two typical scenarios can be distinguished which are discussed in the following sections.



### ***Multiple active communication channels***

In this setup, all communication channels of a device may be configured as separate protocols. This will cause all data to be transported over all redundant communication channels. **eXlerate** can be configured to select from one of the incoming sets of data. This setup puts however more strain on the server and IO device because it effectively doubles the number of IO communication points.

This setup is very straightforward because each communication channel is configured as a separate device.

Please refer to "Device redundancy" on how to configure this type of redundancy.

## Single active communication channel

This setup uses only one active communication channel per device. The redundant communication channels are used only when the active communication fails. This setup puts much less strain on the system and device while providing the same level of redundancy as "Multiple active communication channels".

For each communication channels (e.g. COM port) a separate protocol needs to be configured:

Protocol Table													
rProtocolTable													
ID	Protocol	Type	Options	MoreOptions	Device	Options	MoreOptions	ModemInit	DialCommand	Options	Tag	Description	Status
1	ModbusMaster	RTU			COM11:19200,n,8,1	0	100,10					Flow computer 01 - Primary Path	0 Closed
2	ModbusMaster	RTU			COM12:19200,n,8,1	0	100,10					Flow computer 01 - Secondary Path	0 Closed
3													

Figure 9-3: Primary and secondary protocols

The primary protocol should have all the communication queries configured. The secondary protocol should only contain a 'validation' query which is necessary for checking the availability of the line:

Query Table																							
rQueryTable																							
ID	Protocol	Device	Interval	Timeout	Retries	SleepTime	Row	Col	Worksheet	Options	MoreOptions	Status	Type	Address	Length	Size	SW	MW	MR	type or query	Current Read	(and write),	
1	1	1	30:0	10	10	50	4	8	xTagDB	64		OK	2	0001	72	16				3	CUR		
2	1	1	0	10	10	50	135	8	xTagDB	64		OK	1	1501	1216	1	5			3	WO		
3	1	1	50:0	10	10	50	152	8	xTagDB	64		OK	2	3101	20	16	6	16		3	SET		
4	1	1	50:5	10	10	50	156	8	xTagDB	64		OK	2	3201	20	16	6	16		3	SET		
5	2	1	25	10	3	50	190	7	xTagDB	2		OK	1	1839	1	1				1	SET		

Figure 9-4: Primary and secondary protocol queries

The final and most complicated step is to configure the device switching mechanism. This mechanism should behave in the following manner:

Whenever the primary protocol loses its connection, it should check the status of the secondary protocol. If the status is OK, the primary protocol should switch to the device in use by the secondary protocol (e.g. COM12). Alternatively, the



secondary protocol should switch to the device previously in use by the primary protocol, so it can monitor the status of that device.

Basically, the protocols exchange devices in such a fashion that the primary protocol always uses the first valid device. In the following example, assume that the primary protocol is using device COM11 and the secondary protocol is using COM12. COM11 is part of Port Server 1, and COM12 is part of Port Server 2. If Port Server 1 fails, COM11 will also fail:

ID	Protocol	Type	Opt	Mon	Dev	Opt	Mon
1	ModbusMaster	RTU			COM11:19200,n,8,1	1	No TX, No RX
2	ModbusMaster	RTU			COM12:19200,n,8,1	4	TX, RX

Primary protocol stops receiving/sending data.  
Secondary protocol still receives/sends data.

**Figure 9-5: Primary protocol device failure**

The primary and secondary protocols should now swap their devices:

ID	Protocol	Type	Opt	Mon	Dev	Opt	Mon
1	ModbusMaster	RTU			COM12:19200,n,8,1	1	No TX, No RX
2	ModbusMaster	RTU			COM11:19200,n,8,1	4	TX, RX

**Figure 9-6: Swapping protocol devices**

Since the protocol table is static, swapping the devices cannot be done by writing a new value into the device column. Instead, two worksheet functions are available for this purpose. The 'exSetAlternateDevice(...)' worksheet function can be used to add 'alternate' devices to a protocol:

Protocol ID	Alternate device index
1	2

Alternate device

=exSetAlternateDevice(1,2,"COM12:19200,n,8,1",xNow.Time)

**Figure 9-7: Alternate protocol device**

Secondly, the 'exSelectDevice(...)' worksheet function can be used to switch a protocol to either its primary device, which is configured 'Device' column of the protocol table, or to an alternate device:

Protocol ID	Cell containing device index to select:
1	0 = no device (disabled)
	1 = primary device
	2..n = alternate device

=exSelectDevice(1,A14,xAutoRecalc)

**Figure 9-8: Protocol device selection**

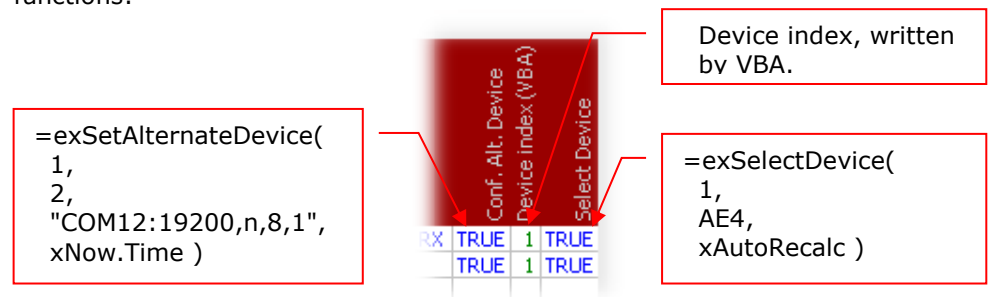
**Chapter 9 - Redundancy - Device communication channel redundancy**

The end result would be the following protocol table with 3 additional columns on the end:

Protocol Table													
rProtocolTable													
ID	Protocol	Type	Options	Device	Options	MoreOptions	ModemInit	DialCommand	Options	Tag	Description	Status	Conf. Alt. Device Device index (VBA)
1	ModbusMaster	RTU		COM11:19200,n,8,1	0	100,10					Flow computer 01 - Primary Path	1 No TX, No RX	TRUE 1 TRUE
2	ModbusMaster	RTU		COM12:19200,n,8,1	0	100,10					Flow computer 01 - Secondary Path	4 TX, RX	TRUE 1 TRUE
3													

**Figure 9-9: Protocol table with device selection**

When looked at in detail, the new columns contain the following worksheet functions:



**Figure 9-10: Device selection columns**

The device index needs to be written from VBA. It is impossible to do this using worksheet functions because it will always result in a circular reference. The resulting VBA code needs to be periodically executed. A good place to do this is the 'OnEvent(...)' event handler defined in the 'modEvents' module:

```
Public Sub OnEvent(iEventID As Integer, strPeriodName As String,
iPeriod As Long)

    Select Case iEventID

        ' 5 seconds timer
        Case 3:

            'Check redundant communication lines
            CheckComPorts

    End Select

Exit Sub
```

**Figure 9-11: Protocol device selection periodic handler**

The determination of the selected device index is best explained using the following Visual Basic code:

```
Sub CheckComPorts

    ' Local data
    Dim iCurrentPrimaryDevIndex As Integer
    Dim iNewPrimaryDevIndex As Integer

    ' Read currently selected device index from worksheet
    iCurrentPrimaryDevIndex = Range("xComms!AI4").Value
    iNewPrimaryDevIndex = iCurrentPrimaryDevIndex

    ' Check primary protocol status
    If Range("xProtocol.1.Status").Value = 0 Then

        ' Status ok, no need for switching

    Else

        ' Status not ok, determine whether to switch or not
        ' Only switch if secondary status is Ok
        If Range("xProtocol.2.Status").Value = 0 Then

            ' Swap devices
            If iCurrentPrimaryDevIndex = 1 Then
                iNewPrimaryDevIndex = 2
            Else
                iNewPrimaryDevIndex = 1
            End If
        End If
    End If

    ' Write new device index to worksheet
    If iCurrentPrimaryDevIndex <> iNewPrimaryDevIndex Then

        ' Switch primary protocol to other device
        Range("xComms!AI4").Value = iNewPrimaryDevIndex

        ' Switch secondary protocol to other device
        Range("xComms!AI5").Value = iNewPrimaryDevIndex
    End If

End Sub
```

**Figure 9-12: Protocol device selection code**

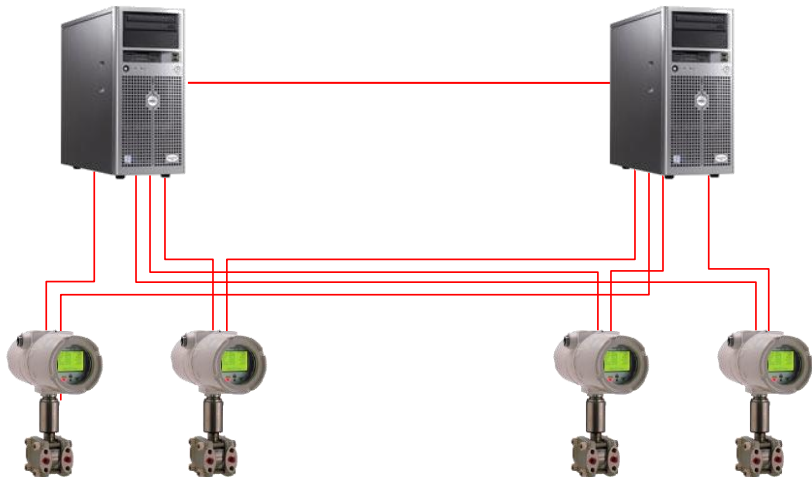
## Server redundancy

### Introduction

**eXlerate** supports fully synchronized servers. This means that standby servers continuously update themselves with the data stored/calculated on the duty server. This also means that standby servers do not: generate reports, calculate totals/averages, etc... Instead, they copy the data from the duty server when it becomes available. This allows **eXlerate** to perform “bump less” transitions when switching from one duty server to another.

In a nutshell, an **eXlerate** standby server is always identical to the duty-server. When the duty server fails, the standby server can immediately take over and ensure data consistency since it was identical to the duty server. When the failed server comes back online, it will synchronize itself with the duty server and take its role as a standby server. The big advantage of fully synchronized servers is data integrity. If for instance, two servers would calculate a time-weighted average, it would be nearly impossible for them to come up with exactly the same result. Both values would be correct, but they would still be marginally different. If in such a case a duty switch would occur; all calculated data would be overwritten by the results of the new duty server, which is undesirable.

No additional configuration is required for multiple servers as opposed to a single server. The application can however be developed in such a way that it performs specific tasks only on the duty server (or a standby server if desirable).



## Duty selection

The duty selection consists of a set of priorities on the 'xNet' worksheet:

8	Clients
9	
0	<b>Duty Selection</b> Running...
1	
2	<b>Priority</b> <b>Server ID</b> <b>Status/Comp</b> <b>Description</b>
3	1: Local override: No override When set, force
4	2: Switch-over override: No override When set, force
5	3: Network detection: Nothing detected Identifies the d
6	4: Local selection: 1 PC08 Contains the ID
7	
8	Selected duty server: 1 PC08 Currently select
9	
0	Duty

**Figure 9-13: Duty selection**

The selection is done in a "first served" fashion. If a server detects that another server is already duty, it will respect the other server and will not become duty itself. If no server is duty, the server who's "Startup wait-time" expires first, will automatically become duty. It is therefore essential that the "Startup wait-time" of the servers is at least 10 seconds apart. Assume there is a power-drop and all servers are shutdown. When the power comes back online all the servers are turned on at the same time. To prevent multiple servers from becoming duty (even though it is just for a short time) at the same time, the "Startup wait-time" has to be at least 10 seconds apart.

## Local duty status

Each server (and client) has its own local duty status which is communicated to all other servers and clients in the system. Therefore, each server and client knows the local duty status of all the others. The local duty status of a particular server can be viewed in the server section of the 'xNet' worksheet:

us IP #3	Status IP #4	Duty Status
		PC08

**Figure 9-14: Local Duty status**

This local duty status is used to determine whether a different server is already the duty server. In that case the other servers respect the duty status and won't become duty themselves.

### ***Manual duty switching***

It is also possible to manually switch to another duty server using the 'exDutySwitch(...)' VBA function which is part of the 'wksNet' module. This function can be called from either a client or a server. It forces all servers into a temporary duty status. By default, the time is 10 seconds but this can be changed in the 'Advanced' section of the 'xNet' worksheet. This period is required to ensure that all servers receive the new duty selection properly. During this time, the server cannot automatically switch to another duty. After the time has expired, the servers remain in the new duty selection.

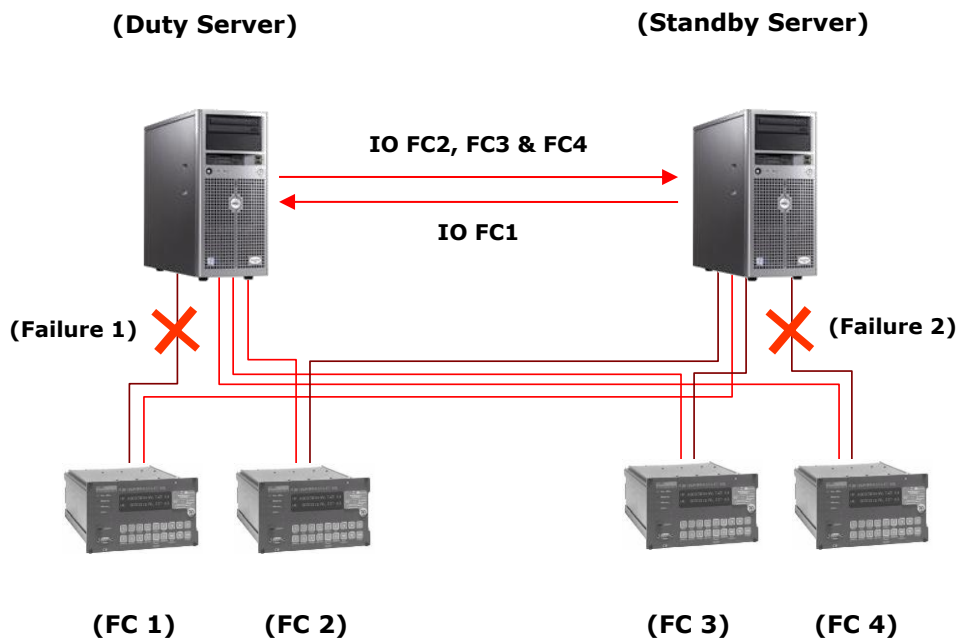
### ***Custom duty selection***

Since the duty selection is part of a worksheet, it can be customized to the fullest extent. The 'xNet' worksheet is provided as a tool to make life easier, but it is not intended to force anyone to use it. Other duty selection strategies may be considered. For instance, an external device may be used which determines who will be duty.

## IO routing

In client/server systems, **eXlerate** uses the IO on the duty server by default. This means that all non-duty computers (clients & standby servers) read and write IO through the duty server. If however, the duty loses communication with a device, a duty switch would be required to restore communication. If this newly selected server also has a communication problem, the system would be crippled. If both servers have the same problem (e.g. No communication with a particular modbus device), then there is no way to workaround this problem. However, if the servers have different problems (Server 1 can't communicate with **device 3**; Server 2 can't communicate with **device 5**) the system availability is not necessarily in danger. **eXlerate** supports a mechanism which can re-route IO communication from the duty server to any other server.

The following figure shows a scenario with 2 points of failures which are not mutually exclusive:



**Figure 9-15: IO Routing**

By default all IO is handled by the duty server, but since the duty lost its connection with FC1 (**Failure 1**), that IO is being re-routed through the standby server. **Failure 2** has no effect on the system availability because it exists only on the standby server.

IO routing is possible at the query level as opposed to protocol level. In some scenarios, multiple devices are attached to a single protocol. For instance, RS-485 multi-drop systems support such architecture. Every communication query is therefore re-routable.

Two worksheet functions are available for query routing. The function 'exQueryStatus(...)' can be used to monitor the status of a query on a particular server. The function 'exQueryServer(...)' function can be used to re-route the query to a particular server. When the function 'exQueryServer(...)' is omitted, eXlerate assumes that the query should be routed through the duty server. This is the default behavior and if no query routing is necessary, no additional configuration work is required.

The following figure shows a Query Table with 4 additional columns which implements the routing mechanism. An example of this is provided in the 'MyNet' sample application:

ID	Protocol	Device	Interval	Timeout	Retries	SleepTime	Row	Col	Worksheet	Options	MoreOptions	Status	Type	Address	Length	Size	SW	MW	MR	Status Server 1	Status Server 2	Determined Server	Selected Server	Update
1	1	1	20	20	3	600	3	7	xTagDB	2		Failed	1	0100	55	1				1	-1	1	1	
2	1	1	20	20	3	600	39	7	xTagDB	2		Failed	2	0200	18	16				1	-1	1	1	FALSE
3	2	1	20	30	10	50	48	8	xTagDB	2		Failed	1	1000	9	1				1	-1	1	1	
4	2	1	20	30	1	50	56	8	xTagDB	2		Failed	1	1100	100	1				1	-1	1	1	
5	2	1	20	30	1	50	98	8	xTagDB	2		Failed	1	1200	100	1				1	-1	1	1	
6	2	1	20	30	1	50	140	8	xTagDB	2		Failed	1	1300	100	1				1	-1	1	1	

Figure 9-16: Query Table with routing support

The additional columns contain the following formulas:

Formula	Status Server 1	Status Server 2	Determined Server	Selected Server
=exQueryStatus(<Query ID>, <Server ID>, <Trigger> )	1	-1	1	1
=exQueryServer(<Query ID>, <Server ID>, <Trigger> )	1	-1	1	1
=<routing formula> (see below)	1	-1	1	1
	1	-1	1	1
	1	-1	1	1
	1	-1	1	1
	1	-1	1	1

Figure 9-17: Query Table routing columns



The **routing formula** implements the actual routing behavior. The following example shows a formula which uses the duty server as the primary route and the standby server as the backup route. If no duty server is selected, it will always use the local IO:

```
=IF(Net.Duty.ID=0,0,IF(AND(Net.Duty.ID=1,A04=0),1,IF(AND(Net.Duty.ID=2,AP4=0),2,Net.Duty.ID)))
```

**Figure 9-18: Query Table routing formula**

In pseudo code the formula would look like this:

```
` If no duty server is selected
If( Net.Duty.ID = 0 ) Then

    ` Then always use the local IO
    result = 0

Else

    ` Is server 1 duty? and it is communicating OK?
    If( Net.Duty.ID = 1 And Query.Status.Server1 = 0 )

        ` Yes it is, lets route the query through that server
        result = 1

    Else

        ` Is server 2 duty? and is it communicating OK?
        If( Net.Duty.ID = 2 And Query.Status.Server2 = 0 )

            ` Yes it is, lets route the query through that server
            result = 2

        Else

            ` No valid query found, use local IO instead
            result = 0

        End If

    End If

End If
```

**Figure 9-19: Query Table routing formula pseudo code**

## Network redundancy

### Introduction

Clients and servers may be equipped with multiple network cards in order to communicate over multiple networks. The configuration involves merely specifying the IP address of the network card. The rest is handled transparently by eXlerate.

Please refer to the previous chapter on how to configure a client or server with multiple IP addresses.

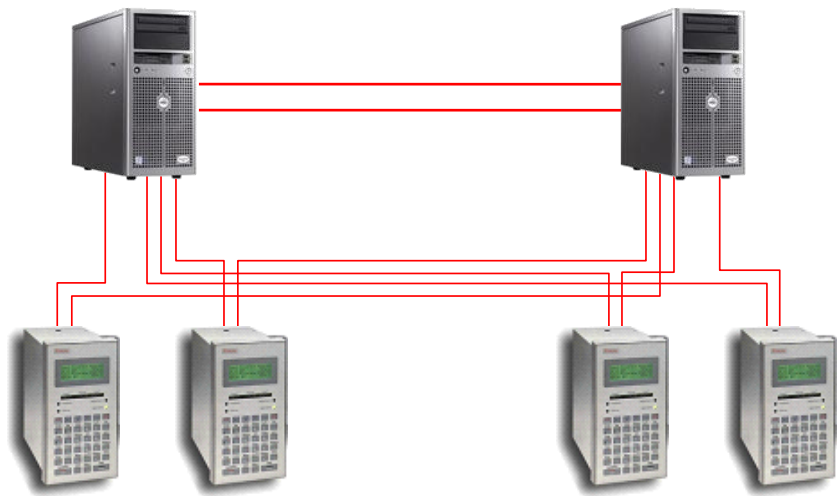


Figure 9-20: Network redundancy

## Network considerations

When using multiple network cards there are some considerations to be made. For instance, should both network cards be in the same IP range? Typically two network architectures can be distinguished:

### Redundant network cards on a single network



**Figure 9-21: Redundant network cards on a single network**

This setup provides redundancy in the form of multiple network cards. If a network card fails, the other network card can take over. However if the switch or router fails, all communication fails as well.

In this setup, all network cards can be either in the same network range or in 2 separate ranges.



If the communication between the servers is completely lost, both servers will become duty because they both think the other server has failed. This behavior is inevitable and can only be prevented by using separate networks.

### Redundant network cards on separate networks



**Figure 9-22: Redundant network cards on separate networks**

This setup provides redundancy in the form of multiple network cards and routers/switches. If a network card of router/switch fails, the backup may be used to resume communication.

In this setup, the redundant network cards need to be on separate IP ranges. For instance, NCA-1 should use IP address 10.0.0.1 and NCB-1 should use 10.0.0.2. NCA-2 and NCB-2 should be in a non-conflicting IP range such as (192.168.0.1 & 192.168.0.2).

# Chapter 10 - Multiple languages

*In this chapter, you will learn how to add multi-lingual support to your application.*

## Introduction

Microsoft Excel already supports multiple languages natively. **eXlerate** however takes this support for multiple languages to the next level.

This chapter will describe how to implement multi-lingual support into applications and how to setup Microsoft Windows for use with these multiple languages.



## Setup Microsoft Windows to use multiple languages

The first step in using multiple languages is to setup Microsoft Windows correctly. Dependant on the installed version of Windows, additional languages need to be installed.

It is not necessary to have a localized version of Windows installed (e.g. German, Russian, etc...). Instead, any version (e.g. English) of Windows will do. The localized versions of Windows additionally translate texts such as: Start-Menu, Control Panel, Print dialogs, etc... into a specific language.

**eXlerate** does require that additional language packs are installed. These language packs can be installed from the *Regional and Language Option* option in the Control Panel.

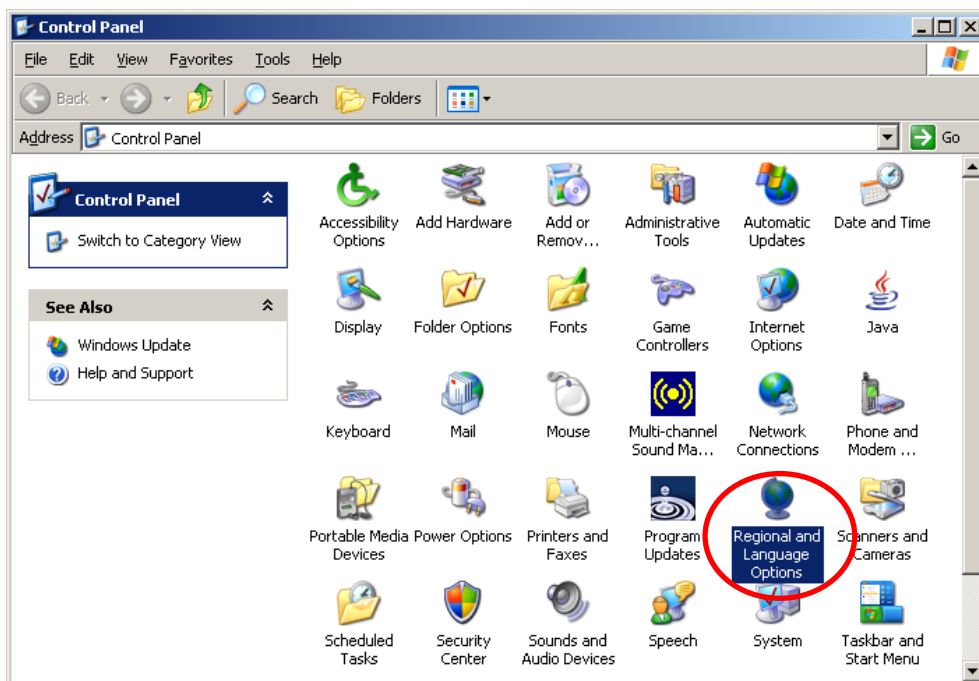


Figure 10-1: Regional and language options

When using East-Asian languages make sure that the Supplemental language support is installed before installing the language packs:

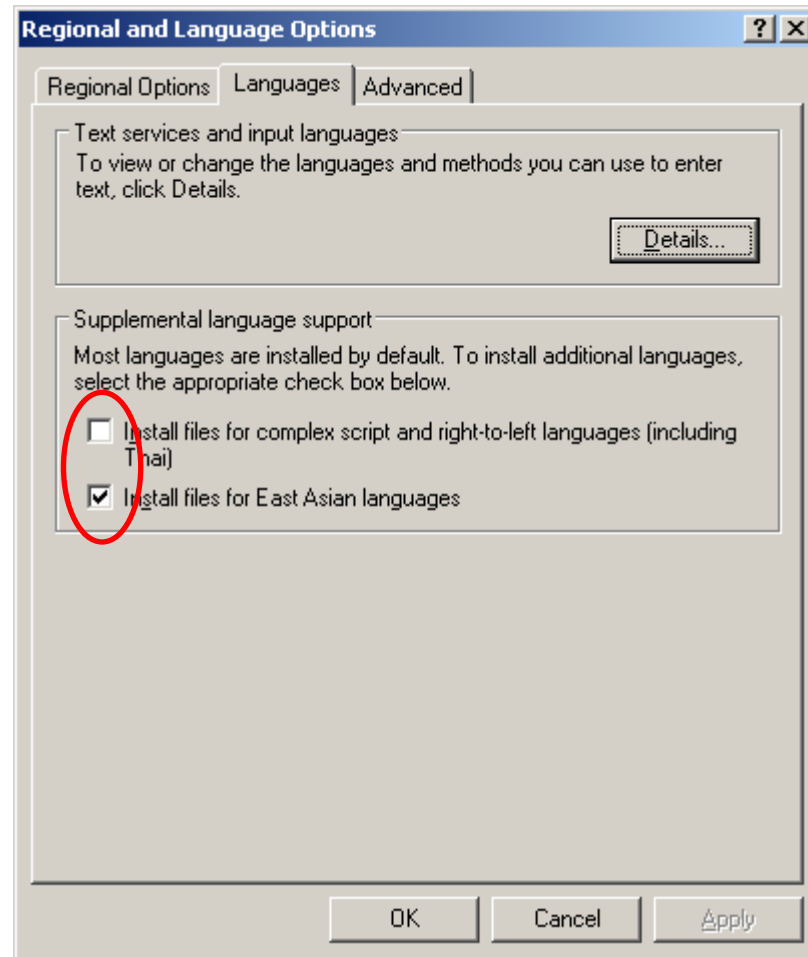


Figure 10-2: Supplemental language support

The language packs can be installed from the 'Advanced' tab on the 'Regional and Language Options' dialog:

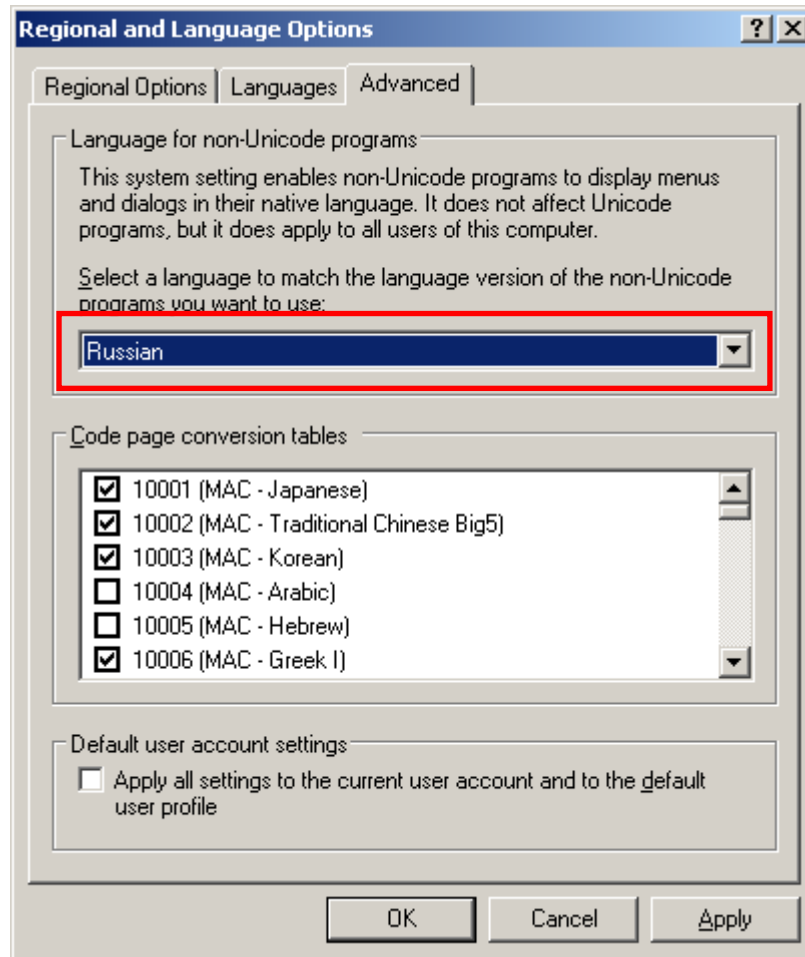
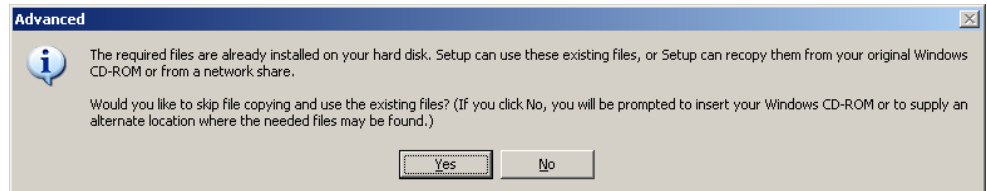


Figure 10-3: Installing language packs

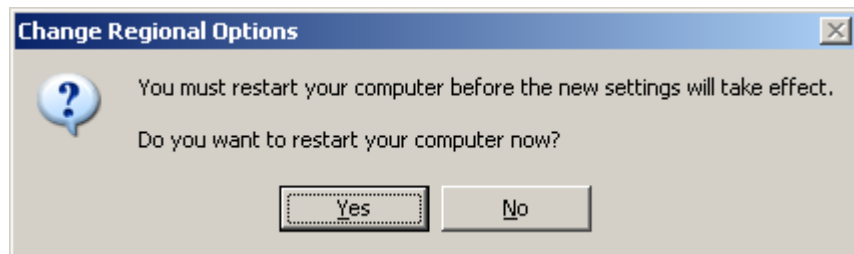
Choose the language that you want using the 'Language for non-Unicode programs' option. In case of multiple foreign languages (e.g. Chinese and Russian), the language with the code-page containing the characters for both languages should be selected. For instance, the Russian code-page does not include any Chinese characters, but the Chinese code-page **does** include Russian characters. The code-pages supported by Windows can be found at the Microsoft website (<http://www.microsoft.com/globaldev/reference/WinCP.mspx>). These code-pages are divided into character sets (e.g. Latin, Cyrillic, Arabic, etc...). Russian for instance, uses the Cyrillic character set. So when selecting 'Russian' or any other Cyrillic language, only the characters in that code-page will be supported. The English characters are supported in all code-pages, so whichever language is selected, English will always work.

After having selected a different language, Windows may or may not prompt with the following message. Select 'Yes' if the message is shown:



**Figure 10-4: Skip file copying during language pack install**

After a new language has been installed the computer needs to be restarted:



**Figure 10-5: Restart computer after installing language pack**



## Application support

eXlerate supports multiple languages through a worksheet called 'xLanguage'. This worksheet is a special case in that it does not contain any worksheet functions. eXlerate recognizes the worksheet by its name, so it is therefore essential that the worksheet is called 'xLanguage'. The layout of this worksheet is also fixed, but can nevertheless be extended with additional languages and custom texts.

	A	B	C	
	Key	Default		Dutch
1				
2	System			
3	Alarms(Dialogs)Caption	Alarm Manager		Alarm Beheerder
4	Alarms(Dialogs)Suppress/Description	Suppress or Enable a single alarm, or all alarms in the indicated group		Onderdruk of geef een enkel alar
5	Alarms(Dialogs)Suppress/EditModeOnly	Alarms can only be suppressed while in 'edit mode'		Alarmen kunnen alleen onderdru
6	Alarms(Dialogs)Suppress/NoValidRow	Currently not a valid row selected		Er is geen geldige rij geselecteerd
7	Alarms(Dialogs)Suppress/SuppressAlarm	Suppress current alarm: %ALARM%		Onderdruk huidige alarm: %ALAR
8	Alarms(Dialogs)Suppress/SuppressButton	Suppress		Onderdruk
9	Alarms(Dialogs)Suppress/SuppressGroup	Suppress current group: %GROUP%		Onderdruk huidige groep: %GRO
10	Alarms(Dialogs)Suppress/UnsuppressAlarm	Enable current alarm: %ALARM%		Geef huidige alarm vrij: %ALAR
11	Alarms(Dialogs)Suppress/UnsuppressButton	Enable		Vrijgeven
12	Alarms(Dialogs)Suppress/UnsuppressGroup	Enable current group: %GROUP%		Geef huidige groep vrij: %GROU
13	Alarms(Messages)AckGroup	User %USER% acknowledged the alarms for group %GROUP%		Gebruiker %USER% heeft alle al
14	Alarms(Messages)DeadbandChanged	User %USER% has changed alarm deadband %ALARM% from %OLD% to %NEW%		Gebruiker %USER% heeft de do
15	Alarms(Messages)DisableAlarm	User %USER% has disabled alarm %ALARM%		Gebruiker %USER% heeft alarm
16	Alarms(Messages)DisableGroup	User %USER% has disabled alarm group %GROUP%		Gebruiker %USER% heeft alarm
17	Alarms(Messages)EnableAlarm	User %USER% has enabled alarm %ALARM%		Gebruiker %USER% heeft alarm
18	Alarms(Messages)EnableGroup	User %USER% has enabled alarm group %GROUP%		Gebruiker %USER% heeft alarm

Figure 10-6: Multi -lingual worksheet 'xLanguage'

## Adding multi-lingual support

There are two ways to add multi-language support to an application. The easiest way is to copy the 'xLanguage' worksheet from a sample application such as 'MyTemplate'. It is also possible to let the Language Wizard generate a fresh 'xLanguage' worksheet:

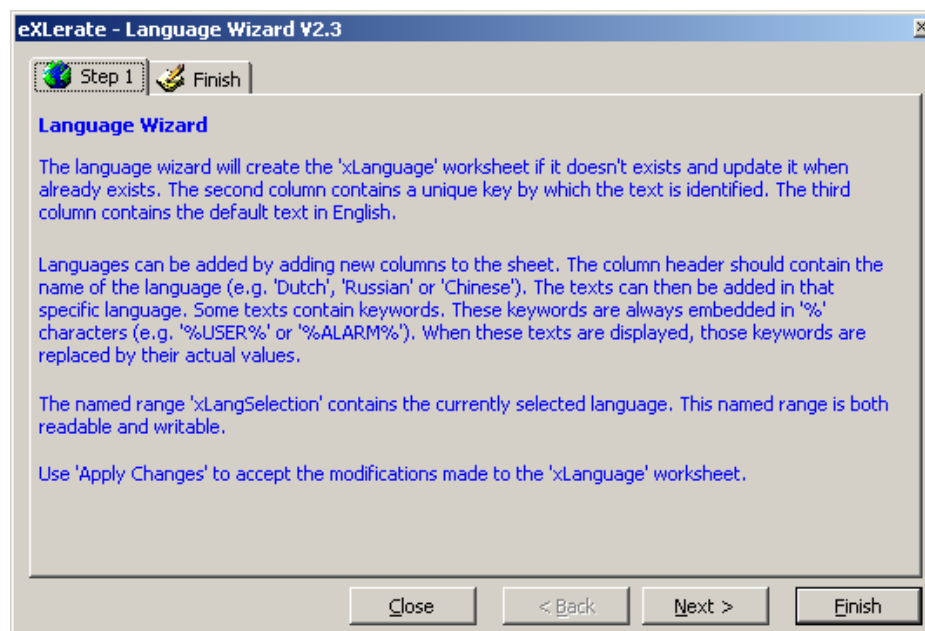


Figure 10-7: Creating a new language

When 'Run' is pressed, the 'xLanguage' worksheet is automatically created:

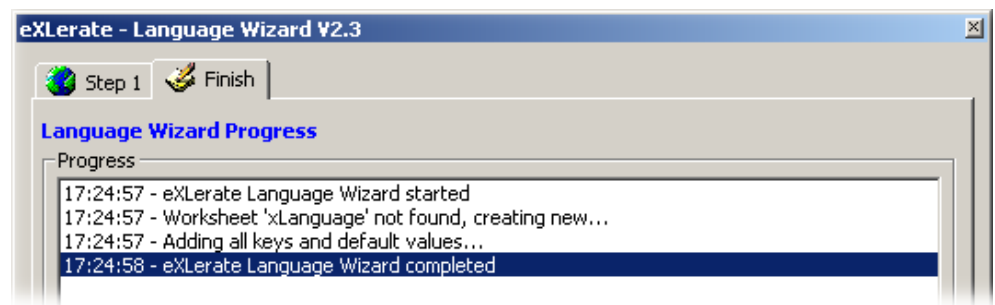


Figure 10-8: Language Wizard output

The newly created worksheet will however have no default formatting, so this will have to be applied manually:

	A	B	C	D	E	F	G	H
1		Key	Default					
2	System							
3		Alarms\Dialogs\Caption	Alarm Manager					
4		Alarms\Dialogs\Suppress\Description	Suppress or Enable a single alarm, or all alarms in the indicated group					
5		Alarms\Dialogs\Suppress\EditModeOnly	Alarms can only be suppressed while in 'edit mode'					
6		Alarms\Dialogs\Suppress\NoValidRow	Currently not a valid row selected					
7		Alarms\Dialogs\Suppress\SuppressAlarm	Suppress current alarm: %ALARM%					
8		Alarms\Dialogs\Suppress\SuppressButton	Suppress					
9		Alarms\Dialogs\Suppress\SuppressGroup	Suppress current group: %GROUP%					

Figure 10-9: Language worksheet without formatting

## Language worksheet layout

The layout of the 'xLanguage' worksheet is as follows:






	A	B	C	D
1	Key	Default	 	Dutch 
2	System			
3	Alarms\Dialogs\Caption	Alarm Manager		Alarm Beheerder
4	Alarms\Dialogs\Suppress\Description	Suppress or Enable a single alarm, or all alarms in the indicated group		Onderdruk of geef een o
5	Alarms\Dialogs\Suppress\EditModeOnly	Alarms can only be suppressed while in 'edit mode'		Alarmen kunnen alleen o
6	Alarms\Dialogs\Suppress\NoValidRow	Currently not a valid row selected		Er is geen geldige rij ges
7	Alarms\Dialogs\Suppress\SuppressAlarm	Suppress current alarm: %ALARM%		Onderdruk huidige alarm
8	Alarms\Dialogs\Suppress\SuppressButton	Suppress		Onderdruk
9	Alarms\Dialogs\Suppress\SuppressGroup	Suppress current group: %GROUP%		Onderdruk huidige group
10	Alarms\Dialogs\Suppress\UnsuppressAlarm	Enable current alarm: %ALARM%		Geef huidige alarm vrij: %
11	Alarms\Dialogs\Suppress\UnsuppressButton	Enable		Vrijgeven
12	Alarms\Dialogs\Suppress\UnsuppressGroup	Enable current group: %GROUP%		Geef huidige group vrij

Figure 10-10: Language worksheet layout

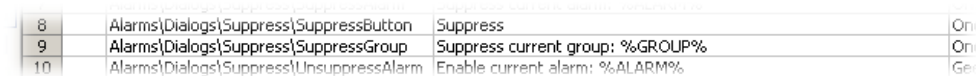
The first three columns are fixed. Each column that follows and has a non-empty value in row 1 is considered an additional language. The country flags (  ) are merely informational and have no functional purpose. The first two rows are also fixed.

Generally speaking, each language consists of its own column in the language worksheet. A row on the other hand can be either a class row:



2	System	Alarm Dialogs\Confirm	Alarm Manager
---	--------	-----------------------	---------------

Or a language row:



8	Alarms\Dialogs\Suppress\SuppressButton	Suppress	On
9	Alarms\Dialogs\Suppress\SuppressGroup	Suppress current group: %GROUP%	On
10	Alarms\Dialogs\Suppress\UnsuppressAlarm	Enable current alarm: %ALARM%	Ge

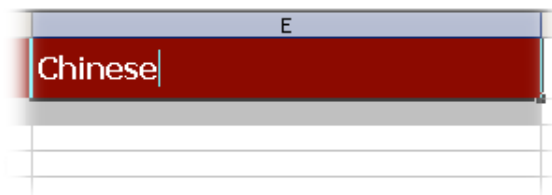
A class row starts with a text in the first column identifying the class to which the succeeding language rows belong. The language worksheet always starts with the 'System' class and succeeding language rows.

Language keys usually consist of several words separated by '\' characters (e.g. "Alarms\Dialogs\Suppress\NoValidRow"). This syntax is not obligatory but it is recommended because it improves readability and extensibility. All the keys in the 'System' class are fixed and cannot be changed.

Some language texts support dynamic keywords which are replaced by specific values when the text is displayed. For instance, the language key "Alarms\Dialogs\Suppress\SuppressGroup" supports the keyword '%GROUP%', which when displayed is replaced by the actual name of the alarm group.

## Adding languages

Languages can be easily added to the application by creating a new column in the language worksheet and giving it the name of the language:



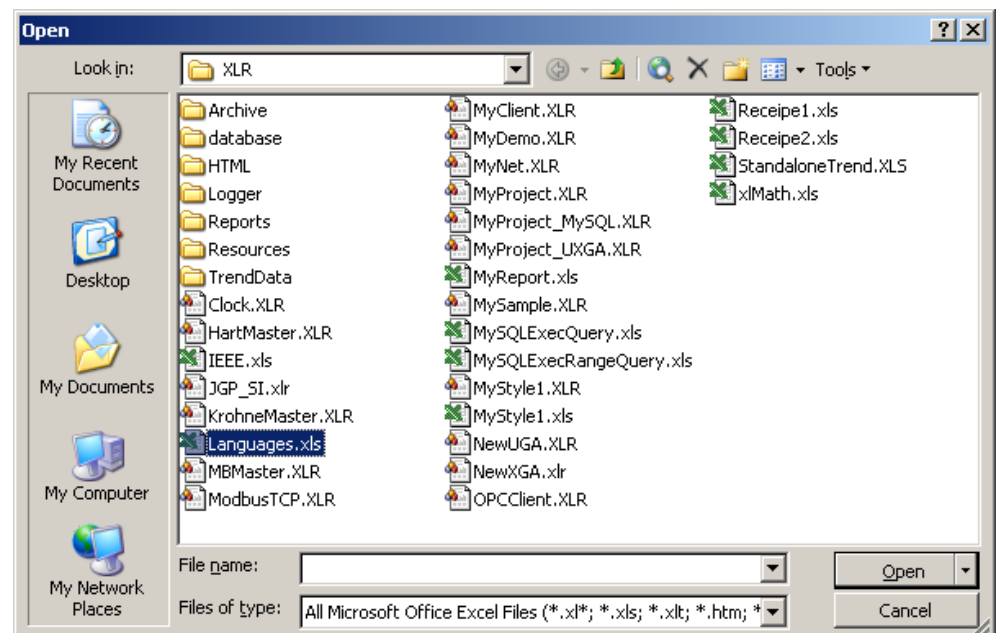
	E
Chinese	

**Figure 10-11: Adding languages**

After that, the actual language texts need to be specified. In case a text is not filled in, the text from the "Default" column will be used.

**eXlerate** is outfitted with a set of default languages which can be easily copied into the language worksheet. If your desired language is not included, please contact your **eXlerate** supplier. These languages are placed in the 'Languages.xls' file which is stored in the application directory (e.g. C:\XLR\Languages.xls'). The following steps show how to copy a language into your application. These steps assume that your **eXlerate** application is open and in design mode:

Open the 'Languages.xls' file:



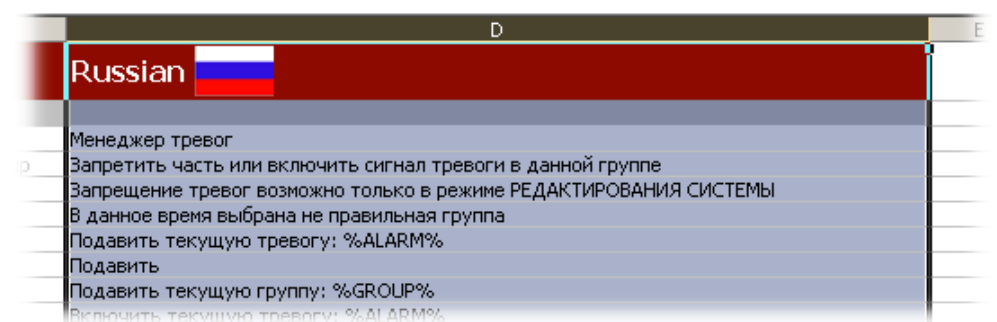
**Figure 10-12: Opening the 'Language.xls' file**

Select the worksheet containing the desired language:



**Figure 10-13: Select desired language**

Select the column containing the language and copy it to the clipboard:



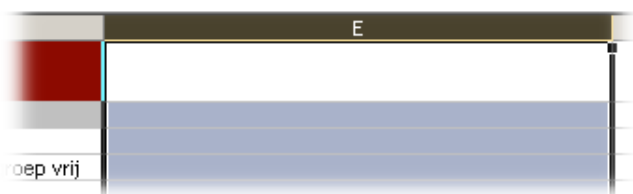
**Figure 10-14: Select language column**

Go back to the original application and select the 'xLanguage' worksheet:



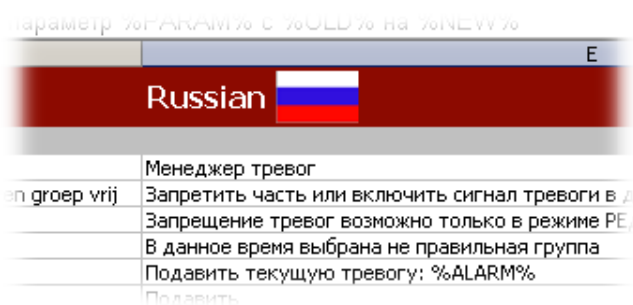
**Figure 10-15: Select 'xLanguage' worksheet**

Select the first empty column:



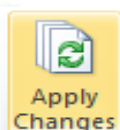
**Figure 10-16: Locate first empty column in language worksheet**

Use 'Paste' to add the language to the worksheet. The end-result is a language worksheet with the desired language:



**Figure 10-17: Successfully added language**

After any modifications to the language worksheet, use "Apply Worksheet Changes" before testing the new language:



**Figure 10-18: Use 'Apply Worksheet Changes' after adding a language**

## Adding user defined texts

By default, the language worksheet contains only 'System' texts. These texts are used for dialogs, log texts, notifications, etc... which are generated by **eXlerate** rather than the application. It is also possible to extend the language worksheet with user defined texts, which can be used in worksheets and VBA. User defined texts must always start with a new class:

101	Trending\Formulas\Default	Default [Average]
102	Trending\Formulas\Interpolate	Interpolate
103	Trending\Formulas\Last	Last
104	<b>Legend</b>	
105	Legend\Body	Body
106	Legend\ClosedPosition	Closed position
107	Legend\Coriolis	Coriolis Flow meter
108	Legend\Densitometer	Densitometer
109	Legend\Differential	Differential
110	Legend\FaultyStatus	Faulty status
111	Legend\Flow	Flow

Figure 10-19: User defined texts in Language worksheet

**eXlerate** will interpret all user defined texts until two or more empty rows are encountered:

137	Dialogs\Reports>Select	Select
138	Dialogs\Reports\PrintMonthlyReports	Print Monthly Reports
139	Dialogs\Reports\PrintDailyReports	Print Daily Reports
140		
141		
142		
143		

Two empty rows causes eXlerate to stop processing any further user defined texts.

Figure 10-20: User defined texts are terminated by two or more empty rows

Tip: After adding user defined texts, the 'Sort' option can be used to quickly sort keys within a group:

103	Trending\Formulas\Last	Last
104	<b>Legend</b>	
105	Legend\Body	Body
106	Legend\ClosedPosition	Closed position
107	Legend\Coriolis	Coriolis Flow meter
108	Legend\Densitometer	Densitometer
109	Legend\Differential	Differential
110	Legend\FaultyStatus	Faulty status
111	Legend\Flow	Flow
112	<b>Common</b>	
113	Legend	Legend

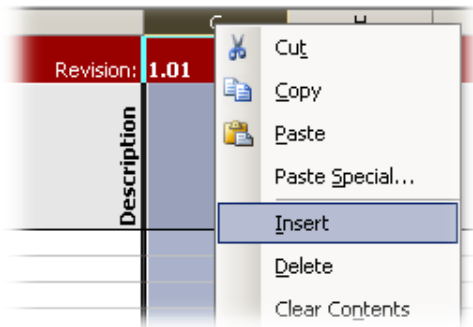
Sort...

Figure 10-21: Sorting user defined texts

After any modifications to the language worksheet, "Apply Worksheet Changes" should be used before testing the modifications.

## Multi-lingual Tag Database

The Tag Database supports multi-language texts for its tag descriptions and alarm descriptions. These additional language texts can be added to the Tag Database directly and don't have to be configured through the 'xLanguage' worksheet. To add multi-lingual descriptions to the Tag Database, insert a column into the Tag Database after the "Description" column:



**Figure 10-22: Insert column into the Tag Database**

Rename the column into "Description\_<Language>". The "<Language>" section should be replaced by the name of the language (e.g. "Dutch", "Russian", "Chinese"):



**Figure 10-23: Rename column to proper language**

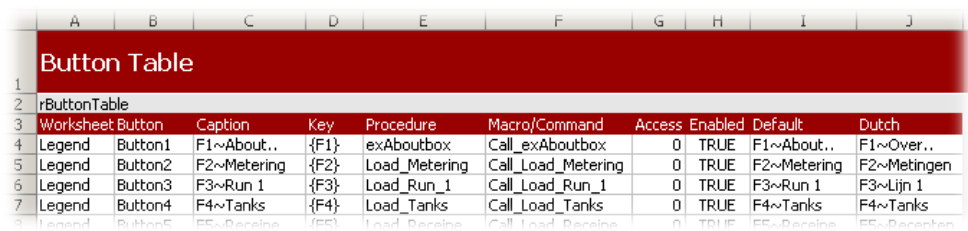
If the currently selected language is "Russian", the tag descriptions from the column "Description\_Russian" will be used for event logging and displaying. The column "Description" is used if the selected language is not explicitly configured or "Default" is selected.

The same mechanism applies to alarm descriptions. If alarm descriptions are explicitly configured using the "AlarmDesc" column, the additional languages can

be added by adding new columns and appending the "<Language>" postfix (e.g. "AlarmDesc\_Russian", "AlarmDesc\_Dutch").

## Multi-lingual Buttons

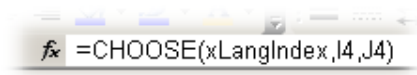
The Button Table does not use the 'xLanguage' worksheet for its button texts. Instead, it uses the CHOOSE(...) function to select from multiple columns containing language texts.



	A	B	C	D	E	F	G	H	I	J
1	<b>Button Table</b>									
2	rButtonTable									
3	Worksheet	Button	Caption	Key	Procedure	Macro/Command	Access	Enabled	Default	Dutch
4	Legend	Button1	F1~About..	{F1}	exAboutbox	Call_exAboutbox	0	TRUE	F1~About..	F1~Over..
5	Legend	Button2	F2~Metering	{F2}	Load_Metering	Call_Load_Metering	0	TRUE	F2~Metering	F2~Metingen
6	Legend	Button3	F3~Run 1	{F3}	Load_Run_1	Call_Load_Run_1	0	TRUE	F3~Run 1	F3~Lijn 1
7	Legend	Button4	F4~Tanks	{F4}	Load_Tanks	Call_Load_Tanks	0	TRUE	F4~Tanks	F4~Tanks
8	Legend	Button5	F5~Receives	{F5}	Load_Receives	Call_Load_Receives	0	TRUE	F5~Receives	F5~Receives

**Figure 10-24: Multi-lingual Button Table**

In the example above, columns I and J contain the texts for the English (Default) and Dutch language. Column C uses the CHOOSE(...) function to select from one of those languages:



**Figure 10-25: Choosing a multi-lingual button text**

The 'xLangIndex' -name contains the 1-based index of the currently selected language. This index is automatically updated whenever a new language is selected.

For this mechanism to work properly, the order of the language-columns in the Button Table should be identical to the order of language columns in the 'xLanguage' worksheet. If this is not the case, the indexes will be different and the wrong language text will be selected.

After a new language is selected, the actual texts on the buttons should be updated. The 'exSetButtonText(...)' Visual Basic function can be used to update the texts on all the buttons from the Button Table:

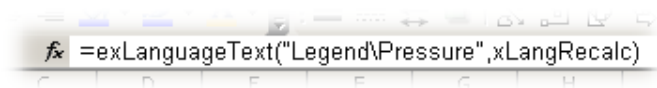
```
Sub SelectLanguage_Dutch()
    Range("xLangSelection").Value = "Dutch"
    exSetButtonText
End Sub
```

**Figure 10-26: Selecting a new language**



## Multi-lingual worksheets

Worksheets can be extended with multi-lingual support via the 'exLanguageText(...)' worksheet function. This worksheet function returns the text associated with a Language Key in the currently selected language. If the text is not available in the currently selected language, the default text is returned. The 'Legend' worksheet of the 'MyProject' application contains working examples of this feature.



**Figure 10-27: 'exLanguageText' worksheet function**

The following example shows the result of the 'exLanguageText(...)' worksheet function:



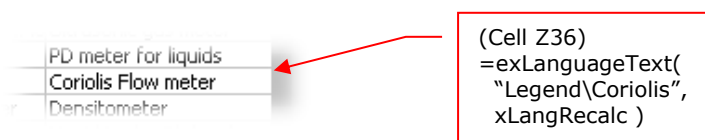
The function uses the trigger 'xLangRecalc' which causes the function to be re-calculated every time a different language is selected.

In case of group- and text boxes an additional step is required. The text of a control can be linked to a name or a cell by pressing 'F2' while the control is selected:



**Figure 10-28: Linking a control to a cell or name**

Since it is not possible to enter the 'exLanguageText(...)' function directly into the formula, an additional reference is required. An easy way is to use a cell on a non-visible part of the worksheet and then refer to it from the control. This cell should contain the 'exLanguageText(...)' function:



**Figure 10-29: Language text referenced by a control**

## Multi lingual VBA code and forms

Forms and VBA code can be extended with multi-lingual support using the 'exLanguageText(...)' Visual Basic function. This worksheet function returns the text associated with a Language Key in the currently selected language. If the text is not available in the currently selected language, the default text is returned. This function is identical in functionality to the worksheet function 'exLanguageText(...)' except that it does not have the trigger-argument.

The following example illustrates the use of the 'exLanguageText(...)' to show a multi-lingual message box.

```
` Ask user to open the valve or not?
strPrompt = exLanguageText( "Messages\OpenValve" )
iRes = exMsgBox( strPrompt, vbYesNo, "eXLerate" )
```

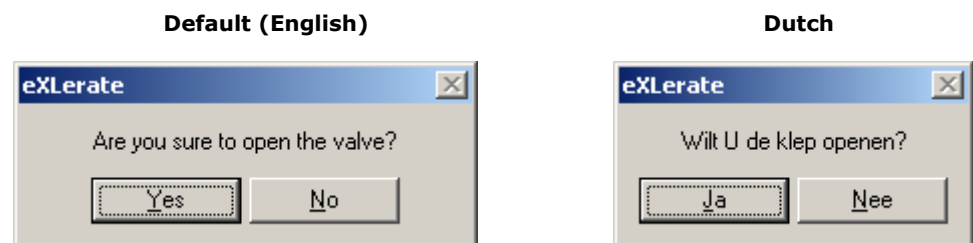


Figure 10-30: Multi-lingual VBA code

When a form is activated, the texts on the form are dynamically filled in with the currently selected language. The following example illustrates the use of 'exLanguageText(...)' to fill in the text upon form activation. The 'frmReports' user-form in the 'MyProject' application contains the following working example:

```
Private Sub UserForm_Activate()

    'Set multi-lingual texts
    Caption = exLanguageText("Dialogs\Reports\Caption")
    frmFrame.Caption = exLanguageText("Dialogs\Reports\Select")
    btnMonthly.Caption = _
        exLanguageText("Dialogs\Reports\PrintMonthlyReports")
    btnDaily.Caption = _
        exLanguageText("Dialogs\Reports\PrintDailyReports")

End Sub
```

Figure 10-31: Multi-lingual user form

## Language selection

The currently selected language can be read/written using the 'xLangSelection' name. This name is part of the 'xWizard' worksheet. If the name is not available, run the 'Tag & Object Wizard' to create the name. To display the currently selected language on a worksheet, the following formula can be used:



**Figure 10-32: Obtain currently selected language**

To select a language, the name of the language needs to be written to the 'xLangSelection' name:

```
Sub SelectLanguage_Russian()
    Range("xLangSelection").Value = "Russian"
    exSetButtonText
End Sub
```

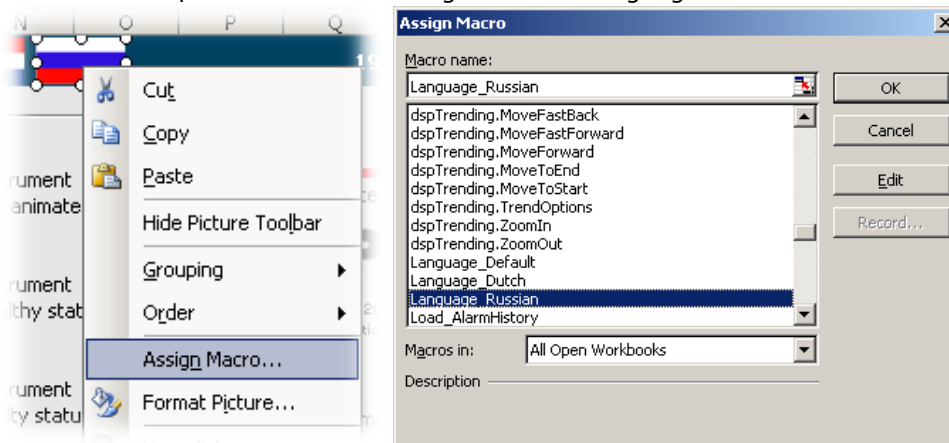
**Figure 10-33: Selecting a new language**

In order to make the selection possible from a worksheet, a button or picture can be used:



**Figure 10-34: Language selection buttons/pictures**

This button or picture should be assigned to the language selection macro:



**Figure 10-35: Assigning language selection macro to picture**

# Chapter 11 - Terminal Services

## Introduction

Terminal Services, also known as “Remote Desktop Services”, offers the ability to view and manage an **eXlerate** system remotely using one or more remote desktop sessions (RDP). With Remote Desktop, the **eXlerate** program runs on the Server, but is visible on a Client computer. This gives the benefit that you do not need to install any software on your Client computer in order to view or manage the **eXlerate** system.

The following overview gives an example of a terminal services setup:

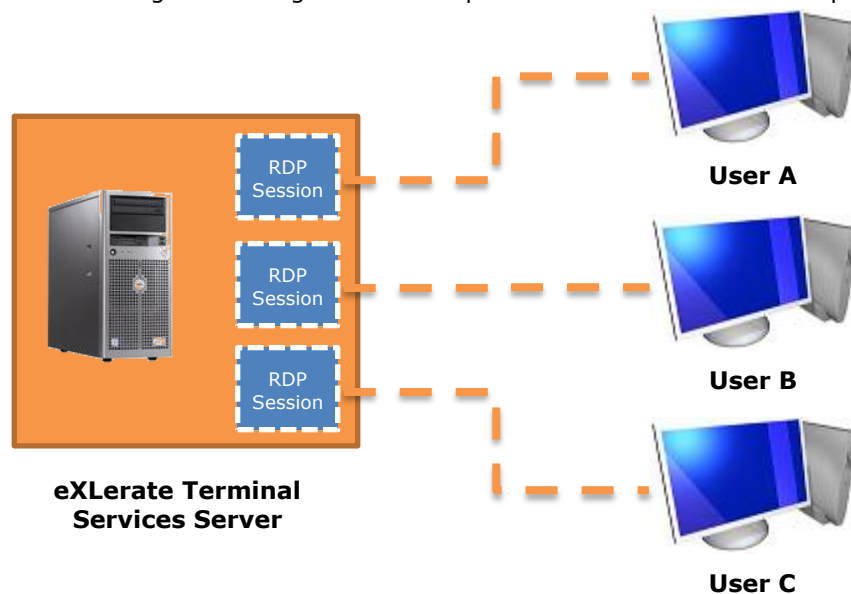


Figure 11-1: Terminal Services Setup

## Requirements

In order to use the **eXlerate** Terminal Services, the following software and licenses are required.

### Operating System

**eXlerate** Terminal Services Mode can be used with any supported version of Microsoft Windows. The Desktop operating systems, Windows XP, Windows Vista and Windows 7 however only support 1 simultaneous remote desktop connection and are therefore not suited for running multiple remote desktop instances. In order to do this you need **Windows Server 2003** or **Windows Server 2008(R2)** with the **Terminal Services Role** enabled and the appropriate licenses (**TS CAL's**).

## Microsoft Office

All supported Microsoft Office versions can be used in combination with Terminal Services. The licensing is however different for these versions.

Office	Editions & Licenses
Office 2003	You need to have a <b>Retail or Enterprise</b> edition of Microsoft Office 2003.
Office 2007	In order to use Office 2007 in combination with Terminal Services you need an <b>Enterprise or Volume License</b> key. ( <a href="http://support.microsoft.com/kb/828378">http://support.microsoft.com/kb/828378</a> )
Office 2010	In order to use Office 2010 in combination with Terminal Services you need an <b>Enterprise or Volume License</b> key. ( <a href="http://support.microsoft.com/kb/828378">http://support.microsoft.com/kb/828378</a> )

**Table 11-1: Required Microsoft Office licenses for Terminal Services**

## eXlerate License

In order to use eXlerate in combination with Terminal Services, a license option is required. Please make sure you have purchased the 'eXlerate Terminal Services' license-option prior to configuring Terminal Services with eXlerate.

## Configuration

Using eXlerate in combination with Terminal Services requires configuration at both the Operating System level and the eXlerate level.

### Operating System

In order to setup Microsoft Windows to use Terminal Services, make sure the following requirements are met. It is outside the scope of this document to describe these steps in detail, please consult the Internet if you need information about a particular topic.

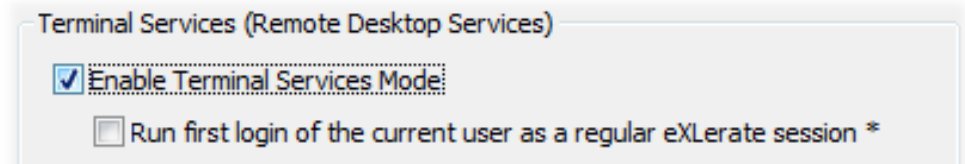
#	Step
1	Install 'Remote Desktop Services' role on Windows Server.
2	Purchase and install 'RDS CALs' for the users or computers which are to connect to the server.
3	Set the remote desktop services setting 'Restrict each user to a single session' to 'No'.
4	Create a Windows User called 'eXlerate_Remote' and give it 'Administrator' rights.

**Table 11-2: Operating System Configuration**

## eXlerate

Now that Windows is configured correctly, setup the user 'eXlerate\_Remote' so that it automatically starts **eXlerate** and launches the appropriate application (see *eXlerate Reference Manual Volume I*).

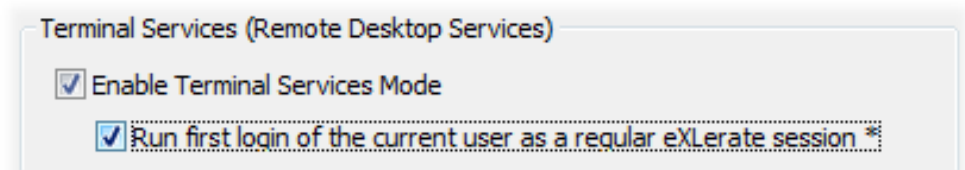
In the **eXlerate** Control Center, select the 'Enable Terminal Services Mode' option from the 'Options' dialog:



When this option is enabled, every **eXlerate** application that is started on the system runs in 'Terminal Services Client Mode'. Using this mode, a single application shortcut may be started multiple times (one in each terminal services session) on the same computer.

### Terminal Services on a Duty/Standby/Standalone server

You may run a separate server for Terminal Services Clients, but you may also combine a Duty/Standby server, or a Standalone system with Terminal Services support. In this case you can use the 'Run first login of the current user as a regular **eXlerate** session' option to ensure that your first launched session runs as a regular **eXlerate** session (i.e. **eXlerate** Server Mode or **eXlerate** Standalone Mode).

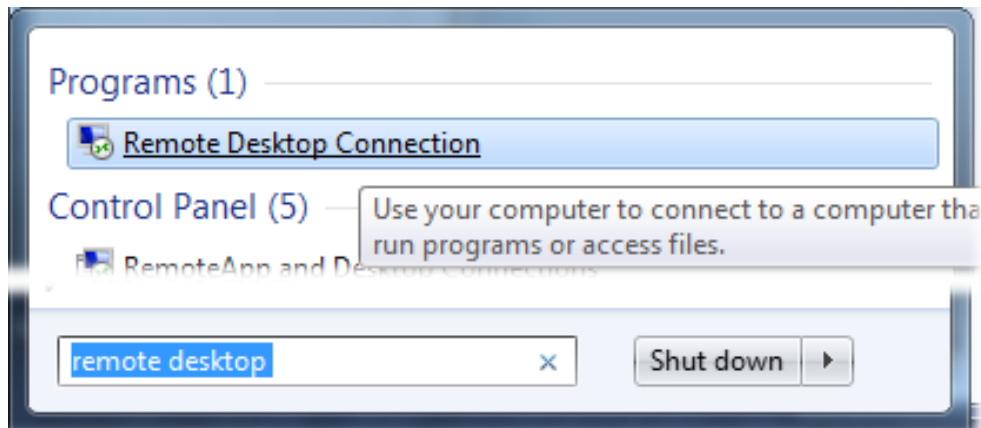


### Client/Server support

In order to use Terminal Services, your application needs to contain client/server support. In short this means that it requires a 'xNet' sheet and the server table should contain the names and IP-addresses of the server(s) you which to connect to. Please read *Chapter 8 - Client & Server* on how to add client/server support to your application.

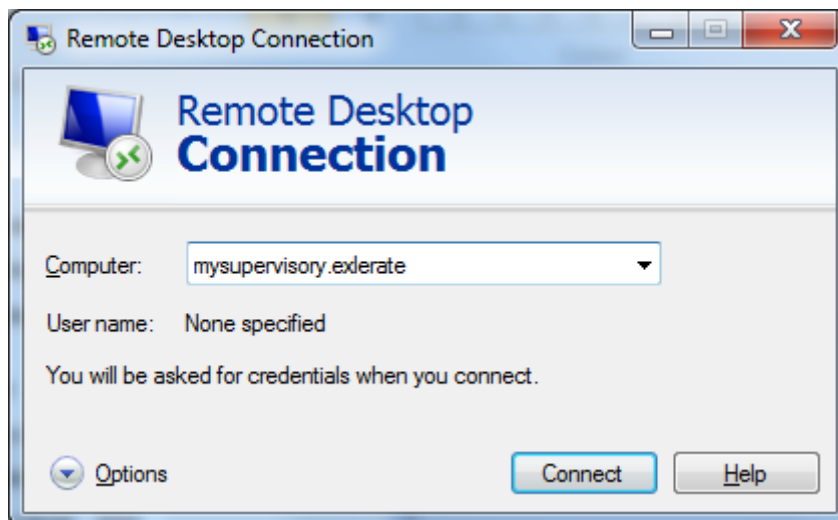
## Using Terminal Services

After Terminal Services is correctly configured you can start using it. To remotely access the **eXlerate** Terminal Services system, start the 'Remote Desktop Client'. You can find the remote desktop client by typing "remote desktop" in the search-bar of the Windows 7 start-menu:



**Figure 11-2: Locating the Remote Desktop Client**

After pressing 'Enter' the 'Remote Desktop Connection' window appears:



**Figure 11-3: Remote Desktop Connection Window**

After you click 'Connect' it will connect to the **eXlerate** Terminal Services system. By default the size of the desktop will be adjusted to the screen resolution of the client computer. The **eXlerate** application may however have been developed for a specific resolution. To open a remote desktop connection with a specific screen resolution you can use the following command-line:

```
mstsc /v:<IP-address/host> /w:<width> /h:<height>
```

**Example:**

```
mstsc /v:10.0.0.105 /w:1680 /h:1050
```

# Chapter 12 - Trouble shooting

## *Introduction*

This chapter is meant for trouble shooting of not-so-trivial problems that an application developer may run into.

Trivial problems in this context are issues like: 'why doesn't the animation of my shape work as I expected?' or 'Why is the outcome of my calculations not correct?' You will not find answers on such questions in this section, or any section for that matter.

The types of problems you may run during application development into are perhaps far more diverse and complex than can ever be solved for you in this "cookbook" style chapter. Fortunately, you have an excellent development tool with Excel to dig into such issues. If you need assistance in application development/engineering you might consider obtaining external help.

Defensive programming is a good remedy against getting problems in the first place. This manual is not a learning book for defensive programming, but the instructions in this reference manual should help you out to solve the problems discussed below.

There are problems known while working with Excel; try the internet and you will find such issues, although Microsoft has solved all issues in the specified Excel versions that are needed for eXlerate to operate correctly, reliably, and stable.

If you have some doubt in working with Excel in an industrial environment, you should be reassured because of the fact that today a myriad of applications throughout the world are running in harsh industrial environments for 24 hours per day, 365 days per year with only a minimum system administration or maintenance requirements. Excel and other Office components are by its millions of user a very well tested application suite!

If you might need further help, please contact your local distributor or Spirit IT at ([techsupport@spiritit.com](mailto:techsupport@spiritit.com)).



## ***Real-time data communications problems***

One of the most challenging issues during application development are problems encountered with real-time data communications.

Unfortunately it is not possible to predict what the solution would be in your case. Please notice the following remarks about data-communication issues:

- You should preferably have Spirit IT confirm your communication requirements **prior** to purchasing **eXLerate** to make sure that your requirements can be properly dealt with.
- There is a low-level debugging tool built-in in **eXLerate**, the *data-scope*, with which you are able to see data-messages back and forth. Utilize the data-scope, and carefully examine messages and associated replies from external devices.
- There are various communication samples available that are ready-to-run, and may be of help for your situation. Please open the workbook samples, and take a look if these samples indeed solve your problems.
- In many cases, the available parameters in the *Protocol Table* and *Query Table* are not correctly setup for the job. Carefully examine the requirements for your external devices, and check the corresponding functionality in the example drivers of **eXLerate** and the documentation.
- If you still doubt the fact that the available communication drivers in **eXLerate** are equipped for the job that you have in mind, you should contact Spirit IT, or your local distributor directly to discuss your special requirements.
- Spirit IT offers help to novice users in training courses, and in various maintenance and support programs. If required, on-site technical support is further possible. Check your price list for the available options.

## Worksheet functions are excessively called

In Excel, worksheets are recalculated as efficient as possible to optimize system performance.

Due to the resulting behavior of these advanced recalculation optimizations and associated strategy in Excel, parameters to VBA functions should be entered as *expressions* rather than single parameters.

When parameters are directly passed without being expressions, the worksheet functions may and will be called multiple times in a single recalculation cycle, with null arguments rather than the actually supplied arguments.

In its simplest form, an expression constructed with argument *iArgument* is '*iArgument*+0'.

A simple expression for a string argument would be '*strArgument* & ""' or equivalent.

Such expressions are sufficient to avoid multiple 'idle' recalculations.

These expressions for arguments are applicable especially in user-defined VBA functions, such as functions `GetxxxIndex(...)` in module *modRange*.

Example:

```
=MyFunction($A$1,$B$2,$C$3, 2)
```

**Figure 12-1: Example of a VB function causing excessive calling**

Instead, the function should be called as follows:

```
=MyFunction($A$1+0,$B$2+0,$C$3+0, 2)
```

**Figure 12-2: Example of a correct VB function avoiding excessive calling**

## The application does not start up properly

When installing a new version of **eXLerate**, where your existing applications should be running with, occasionally, existing applications do not start up properly. This problem may occur due to a library version issue in Microsoft Excel. All you have to do is to change the existing reference to the **eXLerate** ActiveX DLL, in a quick procedure as described below.

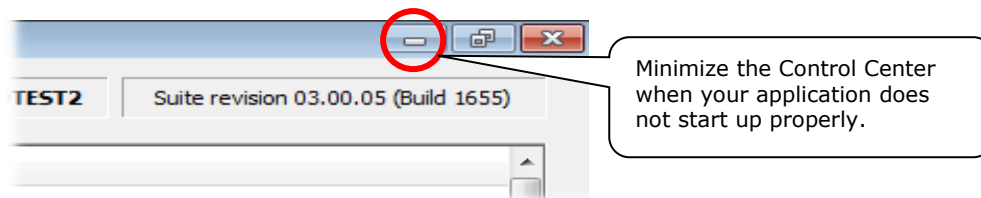
The complete procedure is done in less than 10 seconds.

In most cases, Windows automatically recognizes the new library, in which case your existing applications will run without any problems.

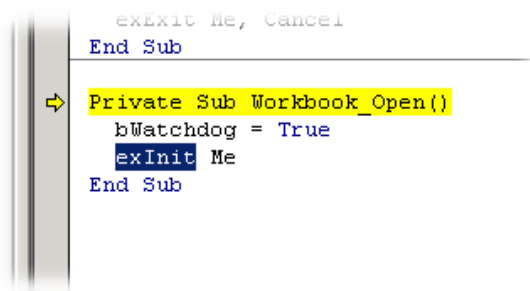
If this is not the case however, you will have started your application in the Control Center, and after the startup command (Runtime or Design), nothing happens. If this is the case, then follow the steps below:

### Step 1

Minimize the main window of the control center.

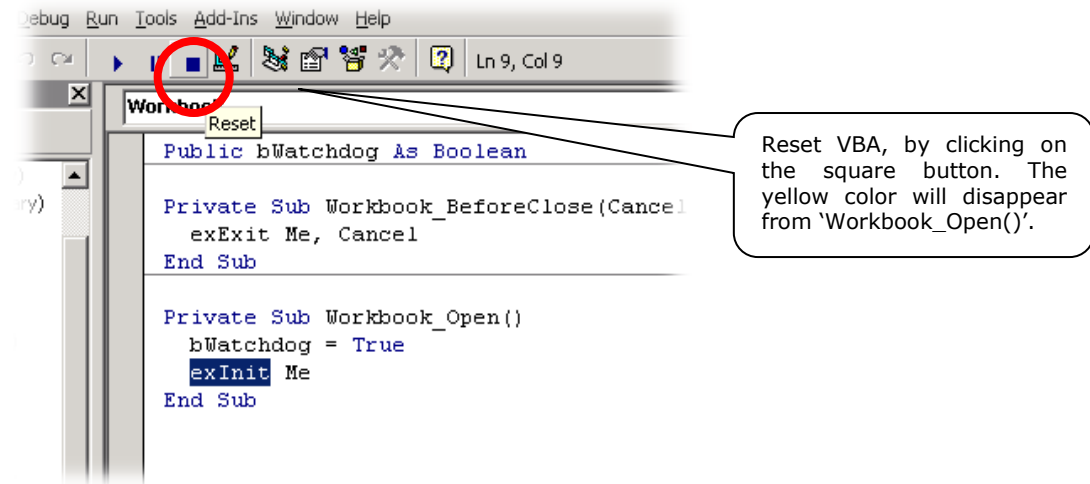


Visual Basic for Applications is stopped at the first command it does not recognize. You can quickly switch to the VBA environment using the F11 key. Most likely this is in the Workbook\_Open() built-in macro. VBA highlights both the procedure and the offending statement.

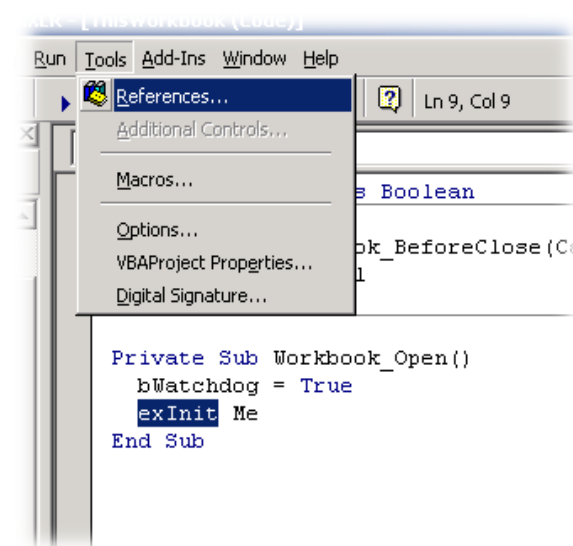


**Step 2**

Stop Visual Basic for Applications by clicking on the VBA toolbar 'Reset' button.

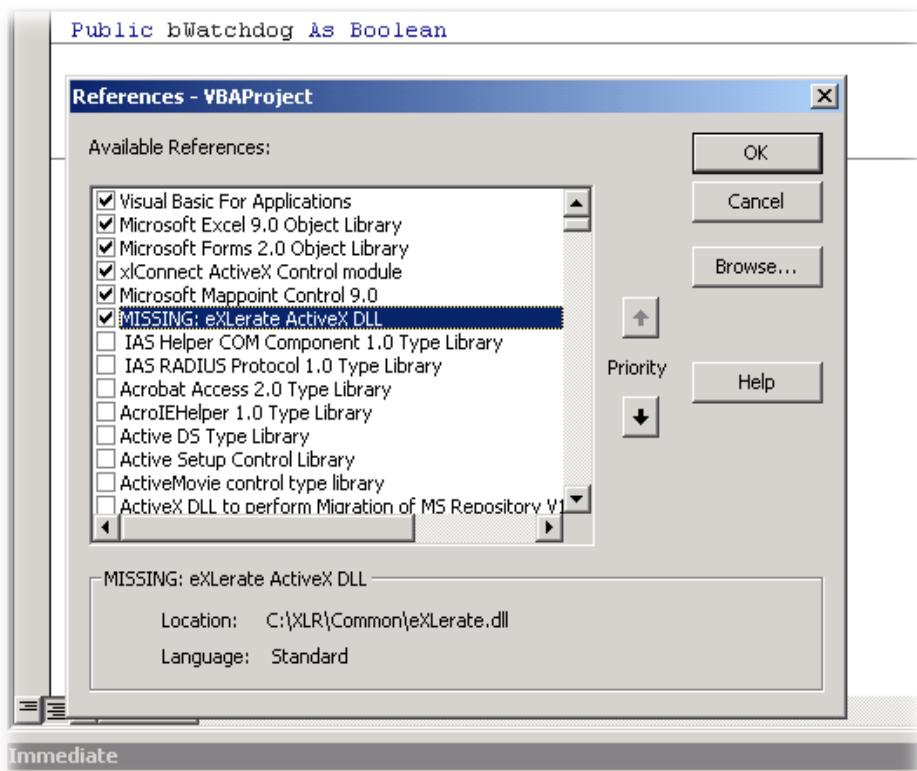
**Step 3**

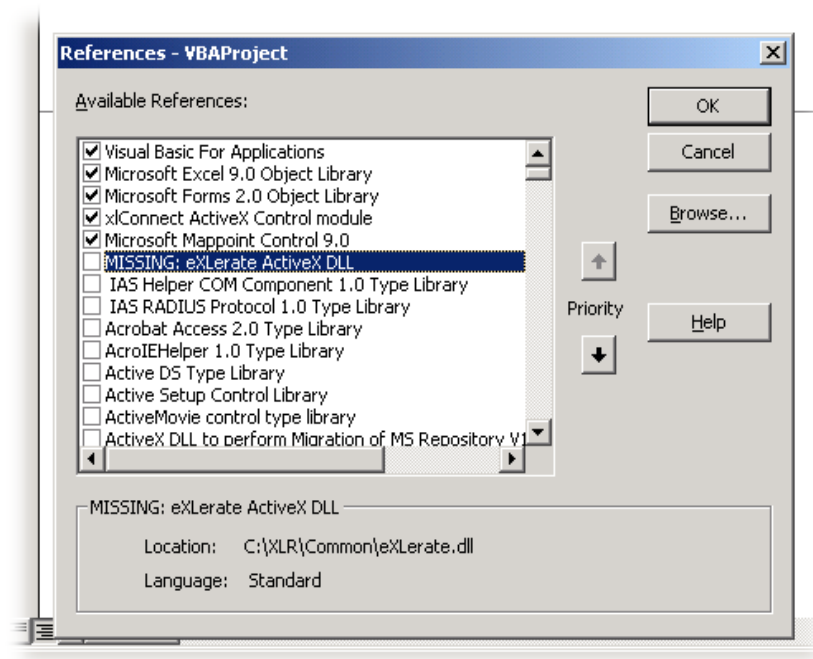
Open the referenced modules in VBA, by opening the 'References' dialog from the 'Tools' menu in Visual Basic, as below:



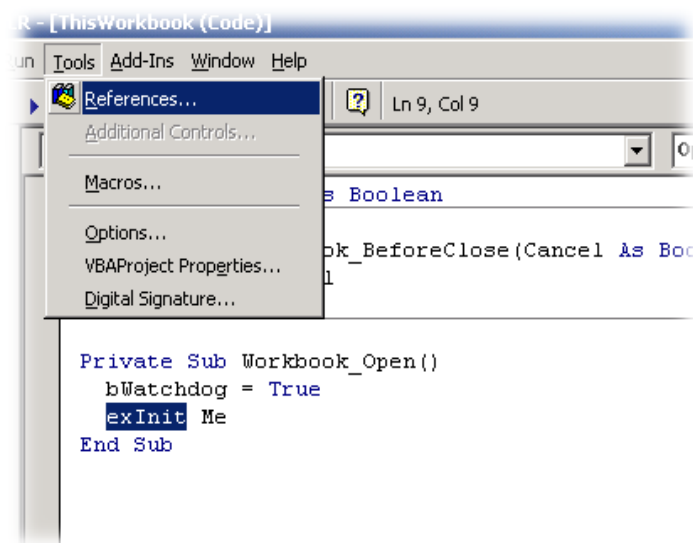
**Step 4**

Remove the reference to the apparently unknown module: 'eXLerate.DLL', by turning off the checkbox at the line: MISSING: eXLerate ActiveX DLL, as shown below:



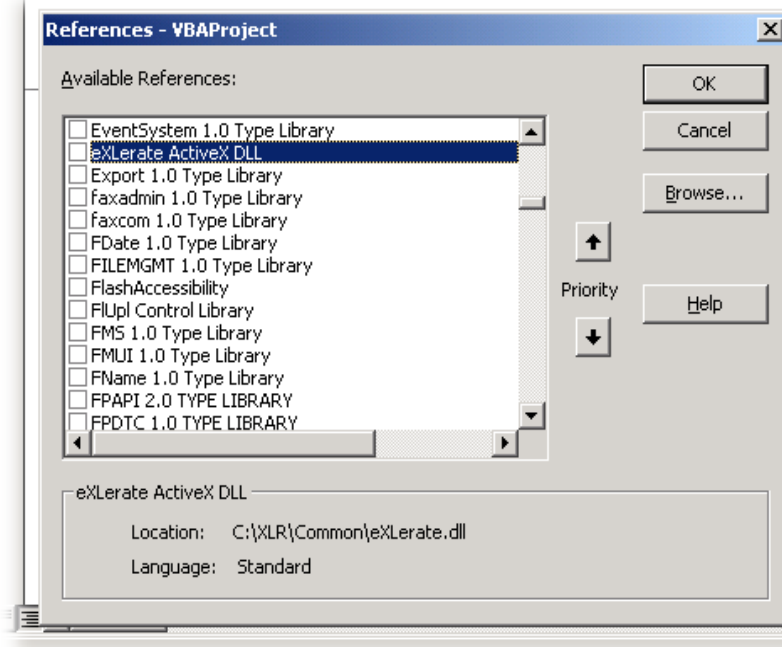
**Step 5**

Close the dialog again. Then, re-open the References dialog again:

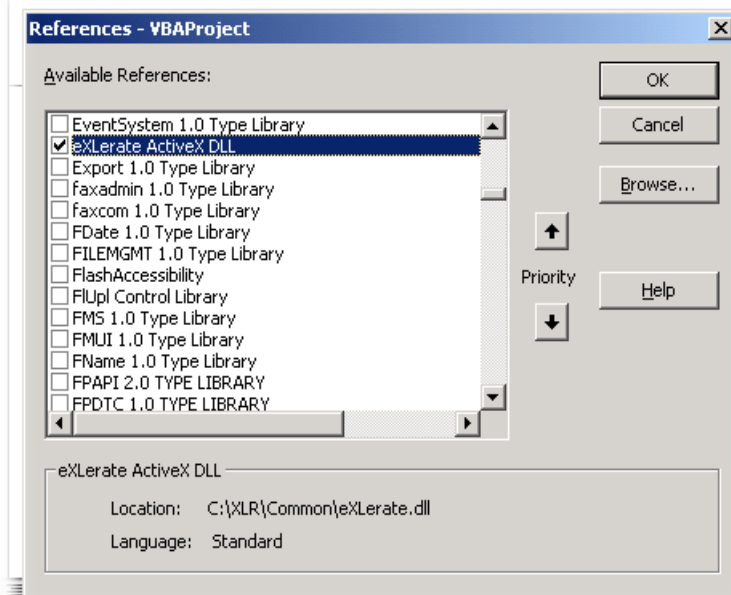


**Step 6**

Scroll down in the reference list and find the eXLerate ActiveX DLL:

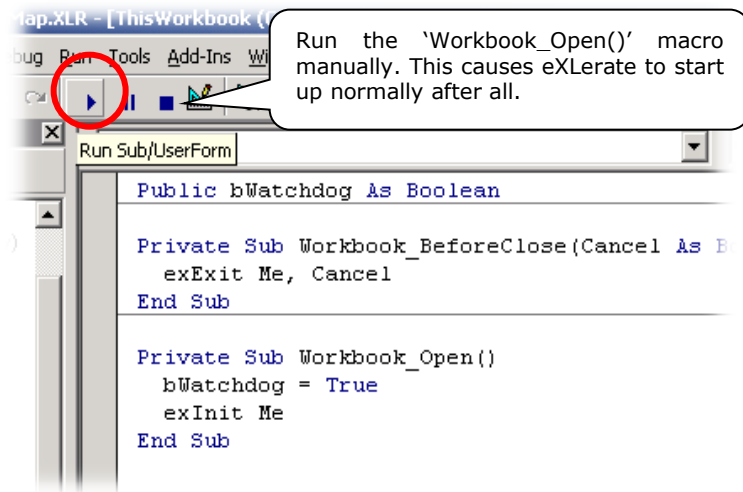
**Step 7**

Enable the eXLerate ActiveX DLL by enabling the checkbox:



**Step 8**

Close the dialog, and now execute manually the 'Workbook\_Open()' macro in VBA. Your application now will startup properly after all.

**Step 9**

Save your application when in design mode. Your problem should have been solved. If the problem still persists, please contact your technical representative.



Contact address:

**Spirit IT B.V.**

Prof. Dr. Dorgelolaan 20, 5613 AM Eindhoven, The Netherlands

fax: +31 40 23 69 605

mailto: [eXLerate@spiritIT.com](mailto:eXLerate@spiritIT.com)

## **Chapter 13 - User notes**





