Introduction

Welcome to the exciting world of SpiritIT eXLerate!

Using SpiritIT eXLerate, you are able to create your complete real-time HMI applications completely from the well-known and most popular Microsoft® Excel environment.

This manual is the reference manual with which a developer is able to create a full-featured, full-blown real-time HMI application.

There are two reference manuals:
- The ‘Application reference manual’, with the installation and setup guide, a tutorial, the control center reference, and application development guide.
- This ‘Advanced topics reference’ manual with various additional information about the built-in Wizards & Tools, the worksheet- and VB function reference, database management system extensions, and various other topics.

For more information

All publications of SpiritIT eXLerate are available for free download from:

<table>
<thead>
<tr>
<th>Publication</th>
<th>Search for:</th>
</tr>
</thead>
<tbody>
<tr>
<td>SpiritIT eXLerate application</td>
<td>IM/eXL-EN</td>
</tr>
<tr>
<td>reference manual</td>
<td></td>
</tr>
<tr>
<td>SpiritIT eXLerate advanced topics</td>
<td>CM/eXL-EN</td>
</tr>
<tr>
<td>reference manual</td>
<td></td>
</tr>
<tr>
<td>function reference manual</td>
<td></td>
</tr>
<tr>
<td>SpiritIT release notes</td>
<td>RN/eXL-EN</td>
</tr>
</tbody>
</table>
Contents

1 Introduction to SpiritIT eXLerate ........................................3
   Advantages .................................................................3
   Purpose of this manual ...............................................3
   How this manual set should be used ................................4
   Abbreviations ..........................................................5
   Terms and definitions ................................................7
   Document conventions ...............................................8

2 Programming topics ....................................................9
   Structuring your applications .....................................9
   Calculation sheets ......................................................10

3 Summer/winter-time ..................................................14
   Introduction ....................................................................14
   Adjusting time ............................................................14

4 Wizards and tools ......................................................16
   Introduction ....................................................................16
   Wizards ..........................................................................17
   Tools ..............................................................................20

5 Controls ........................................................................25
   Introduction ....................................................................25
   Control overview .........................................................25
   Inserting controls .........................................................25
   Modifying controls .......................................................25
   Control Properties .......................................................25

6 Alarm management ....................................................29
   Alarm page types ........................................................29
   Alarm components ......................................................29
   Tag database ...................................................................29
   Alarm group table .......................................................29
   Alarm summary control ...............................................30
   Worksheet: Alarm History ............................................33
   Worksheet: Event History ..............................................34
   VBA and worksheet functions ......................................35
   Alarm dead-band ........................................................36
   Other alarm options .....................................................36

7 Trending .........................................................................37
   Introduction ....................................................................37
   Enabling trending ........................................................37
   Trending parameters ....................................................37
   Visualizing trend data ..................................................37
   exTrendChart control ..................................................37
   exTrendPenSelector control ..........................................42
   exTrendNavigator control ..............................................44
   Accessing trend-data directly .......................................44
   Data storage .................................................................45

8 Relational Databases ...................................................46
   Introduction ....................................................................46
   The system database ...................................................46
   System tables ...............................................................46
   Foreign databases ........................................................48

9 Servers, clients & printers .............................................56
   Introduction ....................................................................56
   Network configuration sheet (xNet) ..............................56
   Server/Client configuration ...........................................57
   Printers configuration ...................................................58
   Advanced settings .......................................................58
   Application development ..............................................58

10 Redundancy ..............................................................63
    Introduction ....................................................................63
    Device redundancy ......................................................63
    Device communication channel redundancy ................64
    Server redundancy ........................................................66
    Network redundancy ...................................................68

11 Multiple languages ..................................................70
    Introduction ....................................................................70
    Application support ....................................................71

12 Terminal services .....................................................76
    Introduction ....................................................................76
    Requirements ................................................................76
    Configuration .............................................................76
    Using terminal services ..............................................77

13 Troubleshooting ........................................................78
    Introduction ....................................................................78
    Real-time data communications problems ....................78
    Worksheet functions are excessively called ..................78
    The application does not start up properly .......................79

14 Revisions ......................................................................81

SQL worksheet functions ..............................................50
SQL VBA functions .......................................................52
1 Introduction to Spirit\textsuperscript{IT} eXLerate

Advantages

With Spirit\textsuperscript{IT} eX\textsuperscript{L}erate, you are able to develop your process visualization applications in a convenient and familiar spreadsheet environment, and yet, for the operator generate a robust and more than complete application.

In an application created with Spirit\textsuperscript{IT} e\textsuperscript{X}Lerate, which is stored in just one (!) single standard Excel workbook, you can:

- Obtain real-time values from external devices, such as a process controller, flow computer, or other device, directly into a spreadsheet cell utilizing the available communication drivers or from an OPC server.
- Visualize the obtained data in a display page using the powerful possibilities of Excel.
- Animate shape objects, such as a bar graph, with the real-time data. Shapes may be created with the office shape library, or may be imported from external tools, such as Microsoft\textregistered Visio, or even AutoCAD\textregistered.
- Create many types of alarms for a tag, which can be printed, logged on disk or shown on a window on the display.
- Trend these values, for an historical overview of a value, in a real-time or historical trend graph.
- Perform extensive calculations on the obtained data, using directly an '=' formula in Excel.
- Generate professional-looking reports of the obtained, derived and all other data, graphics and tables as available in Spirit\textsuperscript{IT} e\textsuperscript{X}Lerate.
- Create additional Visual Basic for Applications code to be even more flexible and powerful, with which you can create your own dialogs, functions and subroutines.
- Add database functionality in your application utilizing one of the available database standards, such as, MySQL (which is also as embedded database), SQL Server or any other OLEDB type driver.
- Secure your applications, so an operator is restricted in his Windows environment.
- Publish your display pages in HTML/XML/VML format onto the Internet using the built-in support of Excel.
- Skip going through a steep learning curve as is required for all other process visualization software packages. And there is no programming required to do so.
- Trouble-shoot in an open environment, without black boxes.
- Do much, much more, utilizing the fully open architecture of the Microsoft Office environment. All of these possibilities as mentioned above can be realized from within your familiar Microsoft Excel environment.

Purpose of this manual

This Spirit\textsuperscript{IT} e\textsuperscript{X}Lerate 2016 reference manual is written for a variety of readers:

The application developer, who is interested in all details required to develop a complete real-time application with Spirit\textsuperscript{IT} e\textsuperscript{X}Lerate.

He or she is assumed to be acquainted in general with visualization software.

It would be an advantage if the application developer is familiar with the basics of some other real-time HMI or SCADA software package.

A more generally interested reader, who wonders if the capabilities and features of Spirit\textsuperscript{IT} e\textsuperscript{X}Lerate will satisfy his/her project requirements.

Both types of readers are assumed to be familiar with the environment of Microsoft Office members, especially Microsoft Excel.

Where the more generally interested reader is expected to be commonly acquainted with Excel, the application developer is assumed to have a thorough understanding of at least the following aspects of Excel:

- Worksheet/workbook organization
- Named ranges, tables
- '=' Worksheet formula syntax
- Cell formatting
- Macro recording and playback

Although not absolutely required, a programmer is also assumed to have a good understanding of the programming environment of Visual Basic for Applications.

When one of the above areas is looking unfamiliar to the application programmer, looking for one of the - more than many – excellent study books on Microsoft Excel would be a great idea.
How this manual set should be used

This reference manual is setup both as a manual, in which a newcomer is quickly introduced with the principles and techniques of real-time application development, as well as a reference manual, in which all details of application engineering can be found. The manual is setup in two volumes, as follows:

Application reference manual
In the first chapters, an introduction is given to the manual. Various terms and definitions as used throughout this book are given.

In chapter 2: ‘Getting Started’ the user is acquainted with hardware and software requirements for SpiritIT eXLerate, the available options, and explains how to install the software on the computer. The content of the end-user license agreement is added for convenience. In this chapter, the usage of the License number and corresponding authorization key are explained.

In chapter 3: ‘Tutorial’, the user is introduced to the concept of SpiritIT eXLerate and the basic ingredients of a complete application, and is added to this reference to quickly start application development in SpiritIT eXLerate.

In the next chapters, fully detailed information is given on all software components.

In chapter 4: ‘Control Center reference’, a full explanation is given on using the SpiritIT eXLerate Control center program.

In chapters 5-10: ‘Application development’, an overview of the components in a complete application is given, so the user knows how start with the development of an application. Other issues of an application are discussed as well, which completes the process of engineering an application in this chapter.

In chapter 11: ‘Reporting’ is described, so a user is able to add reports to an application, and able to publish reports or display pages as HTML files on a web-server.

Advanced topics reference manual
In this volume, many advanced topics, such as structuring your application, and how to utilize calculation worksheets in an application are described in the first chapters.

The next chapters of this volume give detailed information on how to use the various wizards and tools in case not already discussed in the Reference Manual.

In the Function API Reference chapters, all worksheet functions and VBA interfaces are described in detail, with all parameters, comments and return values. This chapter is the core reference part of the SpiritIT eXLerate API in Microsoft Excel. In ‘Trouble shooting’, help is offered when things might not work out the way you have expected.

In the next chapter, you will learn how to add a relational database based on MySQL to your application, and how to use a relational database in an application.

MySQL has been selected as the database engine typically used in a SpiritIT eXLerate environment, because of its programming power, excellent performance as well as its beneficial economic aspects.
### Abbreviations

Throughout this document the following abbreviations are used:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td>Analog Input</td>
</tr>
<tr>
<td>AO</td>
<td>Analog Output</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASCII</td>
<td>American Standard Code for Information Interchange</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CPU</td>
<td>Central Processing Unit</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital to Analog Converter</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed Computing Environment</td>
</tr>
<tr>
<td>DCOM</td>
<td>Distributed Component Object Model</td>
</tr>
<tr>
<td>DCS</td>
<td>Distributed Control System</td>
</tr>
<tr>
<td>DDE</td>
<td>Dynamic Data Exchange</td>
</tr>
<tr>
<td>DI</td>
<td>Digital Input</td>
</tr>
<tr>
<td>DLL</td>
<td>Dynamic Link Library</td>
</tr>
<tr>
<td>DO</td>
<td>Digital Output</td>
</tr>
<tr>
<td>EGU</td>
<td>Engineering Units</td>
</tr>
<tr>
<td>EIA</td>
<td>Electrical Industries Association</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HART</td>
<td>Highway Addressable Remote Transducer</td>
</tr>
<tr>
<td>HMI</td>
<td>Human Machine Interface</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>IEEE</td>
<td>Institute for Electrical and Electronics Engineers</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standards Organization</td>
</tr>
<tr>
<td>MES</td>
<td>Management Execution System</td>
</tr>
<tr>
<td>MMI</td>
<td>Man Machine Interface</td>
</tr>
<tr>
<td>MIC</td>
<td>Machine Identification Code</td>
</tr>
<tr>
<td>ODBC</td>
<td>Open Database Connectivity</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Linking and Embedding</td>
</tr>
<tr>
<td>OSI</td>
<td>Open System Interconnection</td>
</tr>
<tr>
<td>OPC</td>
<td>OLE for Process Control</td>
</tr>
<tr>
<td>P&amp;I</td>
<td>Piping and Instrumentation Diagram</td>
</tr>
<tr>
<td>PC</td>
<td>Personal Computer</td>
</tr>
<tr>
<td>PCB</td>
<td>Printed Circuit Board</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Controller</td>
</tr>
</tbody>
</table>

An interface that allows an application to interact with an application or operating system, in our case, Spirit<sup>®</sup> eXlerate. Most of the Spirit<sup>®</sup> eXlerate API is implemented through Excel worksheet functions.

**API**

- **Application Programming Interface**
  - A set of standard numerical values for printable, control, and special characters used by PCs and most other computers. Other commonly used codes for character sets are ANSI, Unicode, and EBCDIC (Extended Binary-Coded Decimal Interchange Code, used by IBM for mainframe computers).

**COM**

- **Component Object Model**
  - Standard for distributed objects, an object encapsulation technology that specifies interfaces between component objects within a single application or between applications. It separates the interface from the implementation and provides APIs for dynamically locating objects and for loading and invoking them (see ActiveX and DCOM).

**DCOM**

- **Distributed Component Object Model**
  - Microsoft’s protocol that enables software components to communicate directly over a network in a reliable, secure, and efficient manner. DCOM is based on the DCE-RPC specification and works with both Java applets and ActiveX components through its use of the COM. See also ActiveX.

**DCE**

- **Distributed Computing Environment**
  - Definition from the Open Software Foundation, DCE provides key distributed technologies such as RPC, distributed naming service, time synchronization service, distributed file system and network security.

**DDE**

- **Dynamic Data Exchange**
  - A relatively old mechanism for exchanging simple data among processes in MS-Windows.

**DI**

- **Digital Input**

**DLL**

- **Dynamic Link Library**
  - A file containing a collection of Windows functions designed to perform a specific class of operations. Most DLLs carry the .DLL extension, but some Windows DLLs, such as Gdi32.exe, use the .EXE extension. Functions within DLLs are called (invoked) by applications as necessary to perform the desired operation.

**DO**

- **Digital Output**

**GUI**

- **Graphical User Interface**

**HART**

- **Highway Addressable Remote Transducer**
  - A protocol defined by the HART Communication Foundation to exchange information between process control devices such as transmitters and computers using a two-wire 4-20mA signal on which a digital signal is superimposed using Frequency Shift Keying at 1200 bps.

**HMI**

- **Human Machine Interface**
  - Also referred to as a GUI or MMI. This is a process that displays graphics and allows people to interact with the control system in graphic form. It may contain trends, alarm summaries, pictures, and animations.

**I/O**

- **Input/Output**

**IEEE**

- **Institute for Electrical and Electronics Engineers**

**ISO**

- **International Standards Organization**

**MES**

- **Management Execution System**
  - A level of monitoring of a process control system that is above the PLC and HMI level, where data analysis and integration with other aspects of a company such as accounting and purchasing play a significant role.

**MMI**

- **Man Machine Interface**

**MIC**

- **Machine Identification Code**
  - License code of Spirit<sup>®</sup> eXlerate which uniquely identifies your computer.

**ODBC**

- **Open Database Connectivity**
  - A standardized application programmer’s interface (API) for databases. It supports Visual Basic, Visual C++, and SQL for Access, Paradox, Text, Excel and many more database standards.

**OEM**

- **Original Equipment Manufacturer**

**OLE**

- **Object Linking and Embedding**
  - A protocol specification by which an object, such as a photograph, a spreadsheet, video, sound, etc., can be inserted into and used by an application. Renamed by Microsoft into ‘ActiveX’.

**OSI**

- **Open System Interconnection**
  - An ISO standard for worldwide communications that defines a networking framework for implementing protocols in seven layers. Control is passed from one layer to the next, starting at the application layer in one station, proceeding to the bottom layer, over the channel to the next station and back up the hierarchy.

**OPC**

- **OLE for Process Control**
  - A particular COM interface specification. Applications which implement the OPC interface are able to inter-operate without the developer needing to control both the server and client development. In essence, by following the OPC interface, clients and servers from different manufacturers can communicate and interact successfully. The OPC interface is designed to offer the types of interactions that are typical of process I/O hardware such as PLC, DCS and direct I/O boards. Spirit<sup>®</sup> eXlerate 2016 is OPC DA 2.05 compliant and ABB is a member of the OPC Foundation.

**P&I**

- **Piping and Instrumentation Diagram**

**PC**

- **Personal Computer**

**PCB**

- **Printed Circuit Board**

**PLC**

- **Programmable Logic Controller**
  - A specialized device used to provide high-speed, low-level control of a process. It is programmed using Ladder Logic, or some form of structured language, so that engineers can program it. PLC hardware may have good redundancy and fail-over capabilities.
| **RPC** | Remote Procedure Call  
| A form of application-to-application communication that hides the intricacies of the network by using an ordinary procedure call mechanism. It is a tightly coupled synchronous process. |
| **RS232** | EIA standard for point to point serial communications in computer equipment |
| **RS422** | EIA standard for two-wire differential unidirectional multi-drop serial |
| **RS485** | EIA standard for two-wire differential bidirectional multi-drop serial communications in computer equipment |
| **RTU** | Remote Terminal Unit |
| **SCADA** | Supervisory Control and Data Acquisition |
| **SQL** | Standard Query Language |
| **SVC** | Supervisory Computer |
| **TCP/IP** | Transmission Control Protocol/Internet Protocol.  
| The control mechanism used by programs that want to speak over the Internet. It was established in 1968 to help remote tasks communicate over the original ARPANET. |
| **TTL** | Transistor-Transistor Logic |
| **UART** | Universal Asynchronous Receiver & Transmitter |
| **URL** | Uniform Resource Locator.  
| The global address for documents and resources on the World Wide Web. |
| **VBA** | Visual Basic for Applications.  
| The official name is “Visual Basic, Applications Edition.” VBA is Microsoft’s common application programming (macro) language for Excel, PowerPoint, Visio, Access, Project, Word, and the Visual Basic programming environment. |
| **VBE** | Visual Basic for Excel.  
| Although VBA is a general name for the Visual Basic in Office, more specifically, in Excel, the term VBE is used as well. |
| **VML** | Vector Markup Language.  
| An XML based graphics rendering language that describes how an object should be drawn on web pages resulting in more flexibility for the developer and faster, smaller graphical images. |
| **XLL** | Excel Link Library.  
| Special formatted DLL, which is recognized by Excel as extension library. In an XLL, typically worksheet calculations are defined. For example, the xlMath library from ABB containing petrochemical equations has an XLL format. |
| **XML** | Extensible Markup Language. A specification for Web documents that allows developers to create custom tags that enable the definition, transmission, validation and interpretation of data contained therein. |
Terms and definitions

Throughout this manual the following additional terms and definitions are used:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous</td>
<td>A type of message passing where the sending task does not wait for a reply before continuing processing. If the receiving task cannot take the message immediately, the message often waits on a queue until it can be received.</td>
</tr>
<tr>
<td>ActiveX</td>
<td>A family of Microsoft object technologies, formerly called OLE, based on the Common Object Model (COM), serving nowadays as the foundation of many internet products. See also COM/DCOM/OLE.</td>
</tr>
<tr>
<td>C/C++</td>
<td>C is a low-level compiled programming language popular for real-time applications because of its precision and rapid execution times. C++ is an object-oriented superset of C.</td>
</tr>
<tr>
<td>Client/server</td>
<td>A network architecture in which each computer or process on the network is either a client or a server. Clients rely on servers for resources, such as files, devices, and even processing power. Another type of network architecture is known as a peer-to-peer architecture. Both client/server and peer-to-peer architectures are widely used, and each has unique advantages and disadvantages. Client/server architectures are sometimes called two-tier architectures.</td>
</tr>
<tr>
<td>Device driver</td>
<td>A program that sends and receives data to and from the outside world. Typically a device driver will communicate with a hardware interface card that receives field device messages and maps their content into a region of memory on the card. The device driver then reads this memory and delivers the contents to the spreadsheet.</td>
</tr>
<tr>
<td>Engineering units</td>
<td>Engineering units as used throughout this manual refers in general to the units of a tag, for example ‘bar’, or °C, and not to a type of unit, as with ‘metric’ units, or ‘imperial’ units.</td>
</tr>
<tr>
<td>Ethernet</td>
<td>A LAN protocol developed by Xerox in cooperation with DEC and Intel in 1976. Standard Ethernet supports data transfer rates of 10 Mbps. The Ethernet specification served as the basis for the IEEE 802.3 standard, which specifies physical and lower software layers. A newer version, called 100-Base-T or Fast Ethernet supports data transfer rates of 100 Mbps, while the newest version, Gigabit Ethernet supports rates of 1 gigabit (1000 megabits) per second.</td>
</tr>
<tr>
<td>Event</td>
<td>Anything that happens that is significant to a program, such as a mouse click, a change in a data point value, or a command from a user.</td>
</tr>
<tr>
<td>Exception</td>
<td>Any condition, such as a hardware interrupt or software error-handler, that changes a program's flow of control.</td>
</tr>
<tr>
<td>Fieldbus</td>
<td>A set of communication protocols that various hardware manufacturers use to make their field devices talk to other field devices. Fieldbus protocols are often supported by manufacturers of sensor hardware. There are debates as to which of the different fieldbus protocols is the best. Popular types of fieldbus protocol include Modbus, Hart, Profinbus, DeviceNet, InterBus, and CANopen.</td>
</tr>
<tr>
<td>Kernel</td>
<td>The core of Spirit® eXlerate that handles basic functions, such as hardware and/or software interfaces, or resource allocation.</td>
</tr>
<tr>
<td>Peer-to-peer</td>
<td>A type of network in which each workstation has equivalent capabilities and responsibilities. This differs from client/server architectures, in which some computers are dedicated to serving the others. Peer-to-peer networks are generally simpler, but they usually do not offer the same performance under heavy loads. Peer-to-peer is sometimes shortened to the term P2P.</td>
</tr>
<tr>
<td>Polling</td>
<td>A method of updating data in a system, where one task sends a message to a second task on a regular basis, to check if a data point has changed. If so, the change in data is sent to the first task. This method is most effective when there are few data points in the system. Otherwise, exception handling is generally faster.</td>
</tr>
<tr>
<td>Process visualization software</td>
<td>A system for monitoring and controlling for production processes, and managing related data. Typically such a system is connected to external devices, which are in turn connected to sensors and production machinery. The term 'process visualization software' in this document is generally used for software with which SCADA software, HMI software, or supervisory computer software applications can be built. In this document, although strictly not correct, the terms 'SCADA', 'HMI', 'supervisory', and 'process visualization' are alternately used, and refer to the computer software applications that can be realized with Spirit® eXlerate.</td>
</tr>
<tr>
<td>Protocol</td>
<td>An agreed-up format for transmitting data between two devices. In this context, a protocol mostly references to the Data Link Layer in the OSI 7-Layer Communication Model.</td>
</tr>
<tr>
<td>Query</td>
<td>In SCADA/HMI terms a message from a computer to a client in a master/client configuration utilizing the message protocol with the purpose to request for information. Usually, more than 1 data-point is transmitted in a single query.</td>
</tr>
<tr>
<td>Real-time</td>
<td>The characteristic of determinism applied to computer hardware and/or software. A real-time process must perform a task in a determined length of time. The phrase &quot;real-time&quot; does not directly relate to how fast the program responds, even though many people believe that real-time means real-fast.</td>
</tr>
<tr>
<td>Real-time database</td>
<td>A flat database designed for quick, deterministic response. Not to be confused with a relational database, a real-time database is like a hub—a lively transfer point where data can be updated and sent virtually instantaneously, from and to many processes at the same time. In Spirit® eXlerate, the contents of the real-time database can be stored in the system registry to avoid startup and shutdown problems in an application.</td>
</tr>
<tr>
<td>Registry</td>
<td>A database that contains information required for the operation of Windows, plus applications installed under Windows.</td>
</tr>
<tr>
<td>Resource</td>
<td>Any component of a computing machine that can be utilized by software. Examples include: RAM, disk space, CPU time, real-world time, serial devices, network devices, and other hardware, as well as O/S objects such as semaphores, timers, file descriptors, files, etc.</td>
</tr>
<tr>
<td>Synchronous</td>
<td>A type of message passing where the sending task waits for a reply before continuing processing.</td>
</tr>
<tr>
<td>System registry</td>
<td>The registry in Windows is an internal register, in which many properties and values from the operating system, such as class definitions, file associations, file types etc. are stored. The system registry is also used as a replacement of the older ‘.ini’ files, in which a specific program can store its internal settings and properties. Spirit® eXlerate also uses the registry for storage of both its component properties, as well as for storage of real-time settings and user-defined parameters.</td>
</tr>
<tr>
<td>Tag</td>
<td>A ‘tag’ as used within this document refers to a data point existing in the tag database, with a number of properties, such as its assigned I/O address, current value, engineering units, description, alias name, and many others.</td>
</tr>
<tr>
<td>Visual Basic</td>
<td>A graphical programming language and development environment created by Microsoft in 1990, and currently the de-facto standard for scripting in applications like Microsoft Office. All macros in Office are created in Visual Basic.</td>
</tr>
<tr>
<td>Web Server</td>
<td>A computer that has server software installed on it and is used to deliver web pages to an intranet/Internet.</td>
</tr>
</tbody>
</table>
Document conventions

Specific keys, i.e. function keys, editing keys, cursor direction keys etc., are presented with the text on top of the key enclosed between ‘<’ and ‘>’ characters. For example <F1> refers to function key 1 while <Esc> refers to the key with the text ‘Esc’ imprinted. Sometimes the user is assumed to press two keys simultaneously. If this is the case those keys are specified separated by a ‘-’ character. So when <Ctrl-F1> or <Ctrl-a> appears in the text the user should press and release the keys <Ctrl> and <F1> or <Ctrl> and ‘a’-key simultaneously.

When the book symbol as displayed at the left appears in the text in this manual, a reference is made to another section of the manual. At the referred section, more detailed, or other relevant information is given.

A symbol as displayed at the left indicates that the user may read further on the subject in one of the sample workbooks as installed on your machine.

When in this manual a symbol as displayed at the left appears in the text, certain specific operating instructions are given to the user. In such as case, the user is assumed to perform some action, such as the selection of a certain object, worksheet, or typing on the keyboard.

When an important remark is made in the manual requiring special attention, the symbol as displayed to the left appears in the text.
2 Programming topics

Structuring your applications

Introduction

User-defined calculations in SpiritIT 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 SpiritIT 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 SpiritIT 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 SpiritIT eXLerate, and users are expected to comply with this convention.

Figure 2.1 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 SpiritIT 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 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 Figure 2.2, 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 1st group level. To open all groups to the 2nd 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. SpiritIT eXlerate creates various name objects for this calculation tag. For example, when the name: ‘Calc.MyName’ is entered, SpiritIT 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:

* 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

---

Figure 2.2 Calculation worksheet layout

---
‘Calc.ValIndex1_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.

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Shortcut</th>
<th>Type</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constants</td>
<td>CONST</td>
<td>Constants</td>
<td></td>
</tr>
<tr>
<td>Value set in Program code or worksheet</td>
<td>SET</td>
<td>VBA/Ed</td>
<td>Orange</td>
</tr>
<tr>
<td>Value calculated in sheet formula</td>
<td>CALC</td>
<td>Formulae</td>
<td></td>
</tr>
<tr>
<td>Input from tag database</td>
<td>TAG</td>
<td>Tag base</td>
<td>Green</td>
</tr>
<tr>
<td>To be implemented, not completed yet</td>
<td>??</td>
<td>Error/To Do</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 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:

![Figure 2.3 Sub-groups in a calculation sheet](image)

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:

```
fx = "Calculations 1 (version "&Calc.Revision.Major&", "&Calc.Revision.Minor")"
```

Figure 2.4 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 on the next page:
**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:

![Example Calculation Table](image)

The calculation wizard reports the progress and found errors into the output window:

![Output Window](image)

By default, the calculation wizard updates the current calculation sheet. It is possible to update all calculation sheets via the drop-down action button.

**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 Spirit IT 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.

**Calculation wizard**

Calculation names in a calculation worksheet are created using the appropriate wizard from the Spirit IT eXLerate ribbon:
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 SpiritIT 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 SpiritIT eXLerate, and how to use VBA in your real-time HMI application.
3 Summer/winter-time

Introduction

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

More than that, SpiritIT 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](image)

Figure 3.1 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:

![Disabled adjustment for daylight saving in Windows](image)

Figure 3.2 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, use 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’.
4 Wizards and tools

In the previous chapters, you have already been introduced to most of the available wizards and tools of Spirit\textsuperscript{IT} 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 Spirit\textsuperscript{IT} 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:

- **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:

- **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 tools visually shows the hierarchial 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 & 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.

The Tag & Object wizard is started from the eXLerate ribbon or using Ctrl + W, as follows:

![Figure 4.3 Invocation of the Tag & Object wizard](image)

The Tag & Object wizard reports the progress and found errors into the output window:

![Figure 4.4 Output of the Tag & Object wizard](image)

The output window presents the overview of all errors found by the Tag & Object wizard:
**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 9.

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 sheet. 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.

**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:

<table>
<thead>
<tr>
<th>ID</th>
<th>C</th>
<th>R</th>
<th>G</th>
<th>B</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>White</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>Red</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>255</td>
<td>Green</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>255</td>
<td>0</td>
<td>0</td>
<td>Blue</td>
</tr>
</tbody>
</table>

As with most configuration tables in eXlerate, the table starts with a table identifier (‘ColorTable’) 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 is defined.

The color wizard may be started as follows in eXlerate:
When this menu option is selected, the following dialog appears:

![Color wizard dialog](image)

**Figure 4.10 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:

![Button wizard invocation](image)

**Figure 4.11 Invocation of the Button wizard**

When started, the Button wizard looks as follows:

![Button wizard dialog](image)

**Figure 4.12 Step 1/1 of the Button wizard**

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

![Button wizard completed](image)

**Figure 4.13 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:

![Language Wizard](image)

Figure 4.14: Invocation of the Language wizard

When started, the Language wizard looks as follows:

![Language Wizard](image)

Figure 4.15 Step 1/1 of the Language wizard

When the Language wizard is run with ‘Run’, the following dialog appears:

![Language Wizard](image)

Figure 4.16 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:

![Shape tool](image)

Figure 4.17 Selection of the Shape tool

The Shape tool looks as follows:

![Shape properties tool](image)

Figure 4-18: 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 'Hide'-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 'All'-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 'Delete'-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:

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

![Names tool with opened name list-box](image)

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:

![Selection of the Color palette tool](image)

The following dialog appears:

![Color palette tool](image)
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:

![Alarm tree tool](image)

**Generate report**
The “Generate report” option displays the list of configured reports which may then be printed or previewed.

![Generate report](image)

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:
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 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 display the ‘Import sheets’ dialog which allows you to import sheets into your application.

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:

![Advanced Replace Window](image)

Figure 4.30 Advanced replace
5 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:

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

Table 5.1 Control overview

Inserting controls

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

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.

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.

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.
Solid color
When “Solid color” is selected the following options become available:

- 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:

- 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.

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

Picture
When “Picture” is selected the following options become accessible:

- When “Picture” is selected the following options become accessible.
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:

**Fonts**

The following font properties are supported:

- **Font**
  - Tahoma
  - Stencil
  - Syfaen
  - Symbol

- **Size**: 8.0 pts
  - Vertical
- **Styles**: Bold, Italic, Underline, Strikethrough
- **Color**: Red: 0, Green: 0, Blue: 0
- **Use pen color**: Off
- **Transparency**: 0.0%

**Margins**

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

**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:

- **Sample**: 13/12/2015 14:23:45
- **Format**: dd/mm/yyyy{CR}hh:mm:ss
Themes

eXlerate supports several themes as shown below:

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.

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:

To create a new template, click the “Save As…” button.
6  Alarm management

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: AlarmSummary
    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:

<table>
<thead>
<tr>
<th>Alarm Group Table</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parent</strong></td>
</tr>
<tr>
<td>System</td>
</tr>
<tr>
<td>System</td>
</tr>
<tr>
<td>System</td>
</tr>
<tr>
<td>System</td>
</tr>
<tr>
<td>System</td>
</tr>
</tbody>
</table>

Figure 6.1 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:

![Alarm summary control](image)

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.

![Alarm groups tree](image)

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.

Alarm list

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

![Alarm summary list](image)

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:

The following global properties are supported for alarm groups:

![Alarm groups properties](image)

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

![Alarm group state properties](image)

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.
<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique ID that has been assigned to each alarm.</td>
</tr>
<tr>
<td>State</td>
<td>Set of icons which indicate the status of the alarm:</td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Alarm is active." /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Alarm has been acknowledged." /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Alarm is blocked (too many alarm state changes have occurred)." /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Alarm is being suppressed." /></td>
</tr>
<tr>
<td></td>
<td><img src="image" alt="Alarm has been permanently disabled." /></td>
</tr>
<tr>
<td>Timestamp</td>
<td>Date and time at which the alarm was last changed.</td>
</tr>
<tr>
<td>Name</td>
<td>Name of the tag (or alias when specified).</td>
</tr>
<tr>
<td>Location</td>
<td>Name of the alarm group</td>
</tr>
<tr>
<td>Priority</td>
<td>Alarm priority (lower value = higher priority).</td>
</tr>
<tr>
<td>Type</td>
<td>One of the following alarm types:</td>
</tr>
<tr>
<td></td>
<td>“Status Alarm”</td>
</tr>
<tr>
<td></td>
<td>“Low Alarm”</td>
</tr>
<tr>
<td></td>
<td>“LoLo Alarm”</td>
</tr>
<tr>
<td></td>
<td>“High Alarm”</td>
</tr>
<tr>
<td></td>
<td>“HiHi Alarm”</td>
</tr>
<tr>
<td>Description</td>
<td>Alarm description when specified, otherwise tag description.</td>
</tr>
<tr>
<td>Limit</td>
<td>Current limit value (no limit is shown for status alarms).</td>
</tr>
<tr>
<td>Deadband</td>
<td>Deadband value or nothing when no deadband configured in the tag database.</td>
</tr>
<tr>
<td>LastValue</td>
<td>Last value of tag.</td>
</tr>
<tr>
<td>Units</td>
<td>Engineering units.</td>
</tr>
<tr>
<td>Delay</td>
<td>Alarm delay or nothing when no delay configured in the tag database.</td>
</tr>
<tr>
<td>BlockCount</td>
<td>Blockcount of the alarm. When max is reached, the alarm is blocked and will stop changing states until acknowledged.</td>
</tr>
<tr>
<td>Format</td>
<td>Format of the tag or nothing when no format configured in the tag database.</td>
</tr>
</tbody>
</table>

Table 6-1: Alarm summary columns

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

![Alarm list colors](image)

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:

![Alarm list font](image)

**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”, “Acked”, 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:

<table>
<thead>
<tr>
<th>State</th>
<th>Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active &amp; Unacked</td>
<td>Alarm must be active, not acknowledged, not suppressed, and not disabled.</td>
</tr>
<tr>
<td>Active &amp; Blocked</td>
<td>Alarm must be active, blocked, not suppressed, and not disabled.</td>
</tr>
<tr>
<td>Active &amp; Acked</td>
<td>Alarm must be active, acknowledged, not blocked, and not disabled.</td>
</tr>
<tr>
<td>Inactive &amp; Blocked</td>
<td>Alarm must be inactive, blocked, not suppressed, and not disabled.</td>
</tr>
<tr>
<td>Inactive &amp; Unacked</td>
<td>Alarm must be inactive, not acknowledged, blocked, not suppressed, and not disabled.</td>
</tr>
<tr>
<td>Inactive &amp; Acked</td>
<td>Alarm must be inactive, acknowledged, blocked, not suppressed, and not disabled.</td>
</tr>
<tr>
<td>Active &amp; Suppressed</td>
<td>Alarm must be active, suppressed, and not disabled.</td>
</tr>
<tr>
<td>Inactive &amp; Suppressed</td>
<td>Alarm must be inactive, suppressed, and not disabled.</td>
</tr>
</tbody>
</table>
State | Rules
--- | ---
Disabled | Alarm must be disabled.

Table 6.2 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:

![Filter Selection](image)

Figure 6-9: Using alarm filters

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

![Filter Properties](image)

Figure 6-10: 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:

![Filter Property Properties](image)

Figure 6-11: Alarm filter properties

Real-time mode
The alarm summary control can be used in two modes: real-time & diagnostic. 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:

![Real-time mode properties](image)

Figure 6.12 Real-time mode properties

Acknowledgement
Alarms can be acknowledged using 3 buttons in the toolbar:

- **Acknowledge selected alarms**
- **Acknowledge all alarms**
- **Acknowledge all alarms of the selected groups**

![Acknowledgement buttons](image)

Figure 6.13 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:

- **Suppresses or un-suppresses selected alarms.**
- **Suppresses or un-suppresses all alarms of the selected group.**

![Suppression buttons](image)

Figure 6.14 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:

![Edit button]

Figure 6.15 Editing one or more alarms

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

![Edit alarms dialog]

Figure 6.16 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:

![Alarm summary security settings]

Figure 6.17 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:

![Alarm history worksheet]

Figure 6.18 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:

![An alarm history window with the 4 last alarm messages]

Figure 6.19 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**
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 a historical list with all system events.

A typical event history looks as follows:

![Figure 6.20 Event history worksheet](image)

With the aid of additional buttons, the user can filter the events based on filtered time windows. Additionally, a report of these (filtered) events can be generated.

**Event history - Theory of operation**

All system events are also stored on disk, one file for each day. Typically, these files are stored in ‘C:\XLRX\Logger’.

The event history can be constructed by extracting information from these files via an 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:

<table>
<thead>
<tr>
<th>Worksheet</th>
<th>Button</th>
<th>Button text</th>
<th>Key</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlarmHistory</td>
<td>Button3</td>
<td>F3&lt;&lt;</td>
<td>(F3)</td>
<td>dspAlmHist.FastBack</td>
</tr>
<tr>
<td>AlarmHistory</td>
<td>Button4</td>
<td>F4&lt;</td>
<td>(F4)</td>
<td>dspAlmHist.Back</td>
</tr>
<tr>
<td>AlarmHistory</td>
<td>Button5</td>
<td>F5&gt;</td>
<td>(F5)</td>
<td>dspAlmHist.Forward</td>
</tr>
<tr>
<td>AlarmHistory</td>
<td>Button6</td>
<td>F6&gt;&gt;</td>
<td>(F6)</td>
<td>dspAlmHist.FastForward</td>
</tr>
<tr>
<td>AlarmHistory</td>
<td>Button7</td>
<td>F7~Select Date</td>
<td>(F7)</td>
<td>dspAlmHist.PickDate</td>
</tr>
<tr>
<td>AlarmHistory</td>
<td>Button8</td>
<td>F8~Print Date</td>
<td>(F8)</td>
<td>dspAlmHist.PrintHistEvent</td>
</tr>
</tbody>
</table>

Table 6-3: 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, see Figure 6.21.

In this code left, which is copied from the ‘MyProject’ sample, various buttons are linked to the above VBA macros.

```vba
Sub PickDate()
    frmEventDate.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

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 a 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:

![Alarm processing diagram](image)

The signal drops below the limit value, but not below the dead-band value (yet).

A normal alarm state resets, but with a dead-band it remains high.

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 equals or 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. 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.
7 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:

![Insert trend controls](image)

Figure 7.1 Insert trend controls

Three Trend controls can be distinguished:

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exTrendChart</td>
<td>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.</td>
</tr>
<tr>
<td>exTrendPenSelector</td>
<td>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.</td>
</tr>
<tr>
<td>exTrendNavigator</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 7.1 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 further on 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.

![exTrendChart control](image)

Figure 7.2 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:

![Plot Area Properties](image)

Figure 7.3 Plot area properties

**Navigation**
Navigating can be done using the options in the toolbar:

<table>
<thead>
<tr>
<th>Button</th>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start-date</td>
<td></td>
<td>Shows the start-date/time of the chart. When clicked a popup is shown which allows for selecting a specific date/time.</td>
</tr>
<tr>
<td>End-date</td>
<td></td>
<td>Shows the end-date/time of the chart. When clicked a popup is shown which allows for selecting a specific date/time.</td>
</tr>
<tr>
<td>Period</td>
<td></td>
<td>Shows the difference between the start- and the end of the chart. When clicked a list is shown with predefined periods.</td>
</tr>
<tr>
<td>Move to Begin</td>
<td></td>
<td>Moves the start of the chart to the first time a value was stored for any of the pens.</td>
</tr>
<tr>
<td>Fast Backward</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Backward</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Forward</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Fast Forward</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Move to End</td>
<td></td>
<td>Moves the end of the chart to the current date/time.</td>
</tr>
<tr>
<td>Realtime</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Zoom In</td>
<td></td>
<td>Zooms in on the current cursor position.</td>
</tr>
<tr>
<td>Zoom Out</td>
<td></td>
<td>Zooms out. When this button is pressed the value axis-zoom is always reset to 0%-100%.</td>
</tr>
<tr>
<td>Moving Mode</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Zooming Mode</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 7.2 Trend Chart Toolbar

The following properties are available for navigation:

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:

![Plot Area Margins](image)

Figure 7.4 Plot area margins
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:

![Data Cursor](image)

**Figure 7.6 Data Cursor**

The following properties are available for the Data cursor. The background and border tabs define the visual appearance of the cursor:

![Data cursor properties](image)

**Figure 7.7 Data cursor properties**

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

![Data cursor date/time](image)

**Figure 7.8 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 Pen selector control. When enabled, the label can be configured to display additional properties of the tag such as Units & Description:

![Pen label properties](image)

**Figure 7.9 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:

![Pen label color](image)

**Figure 7.10 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 7.11 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 7.12 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 7.13 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:

\[
\text{exTrendChart.Pens.Item(1).MaximumScaleType} = \text{exTrendValueAxisScaleTypeFixed} \\
\text{exTrendChart.Pens.Item(1).MaximumFixedValue} = 100
\]

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

Figure 7.14 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%.
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 7.16 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 7.17 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 7.18 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 7.19 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 7.20: Markers on the chart**

**Limits**

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

**Figure 7.22 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.
By default the ‘Limits’ column is hidden. To show this option open the properties dialog of the PenSelector and enable the appropriate column:

**Figure 7.23 Limits option in PenSelector**

**Figure 7.24 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 7.25 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 7.26 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 7.27 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 7.28 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 7.29 Tag selection**

The following properties are available for the tags pane:

**Figure 7.30 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.

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

- **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**: Selects 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.

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:

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

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

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:
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 7.36 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 42 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 7.37 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 7.38 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 7.39 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 7.40 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exTrendReadTag(...)</td>
<td>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.</td>
</tr>
<tr>
<td>exTrendReadFile(...)</td>
<td>Reads all the trend-data from a specific trend-file. The result is returned as an 2 dimensional array</td>
</tr>
</tbody>
</table>
Function | Description
--- | ---
 | 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 7-5: 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”):

- XY_PSC_360
- XY_PSC_361
- **XY_PSC_362**
  - 10Days
  - 100Days
  - HHiLimit
  - HHLimit
  - LLoLimit
  - LLoLimit
  - Raw
- XY_PSC_363
- XY_PSC_360

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):

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.
8 Relational Databases

In this chapter, you will learn how to use relational databases in eXLerate applications.

eXLerate provides a powerful and yet simple spreadsheet interface to define all your real-time SQL statements directly from a worksheet. The flexible VBA interface gives more control on the database, including parallel execution of SQL queries.

A great feature of eXLerate is redundancy support for its system database, which is automatically synchronized between multiple servers without any programming effort.

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.

In typical eXLerate applications, databases are used for storing historical data of the connected measurement and control devices.

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 system database

eXLerate uses the system database (previously known as the embedded database) for event logging and to synchronize data between redundant servers. The system database is identified with ID '0'.

Since version 4.2, eXLerate supports two types for the system database: the embedded system database and the external system database.

The embedded system database

By default, the embedded system database does not require any additional database server. Configuration is easy; just specify the database path in the application shortcut of the Control Center and you’re ready to go:

![Embedded database path](image)

Figure 8.1 Embedded database path

In case more than one eXLerate application exists, the database path for each application can be configured to be the same path or a different path.

At startup, the embedded database will be created automatically if it doesn’t already exist. If the embedded database exists at the startup time, eXLerate will check if it is not corrupted and contains the required system tables.

The embedded system database is not accessible externally.

The external system database

Since version 4.2, eXLerate supports a system database residing in an external database server instead of the embedded system database. The benefit of the external system database is that it can be accessed by any SQL Client e.g. for troubleshooting or for data transfer to other systems.

The external database server used for the system database must be MySQL compatible(*) and must run on the same computer where eXLerate runs. Furthermore, the database engine must be ‘MyISAM’ for all tables.

(*) ABB Spirit IT advises to use and tests this functionality on the MariaDB database server. Other MySQL compatible database servers, including MySQL, are not tested but may work as well.

The external system database is also configured in application shortcut of the Control Center.

![Configuration of the external database](image)

Figure 8.2 Configuration of the external database

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server address</td>
<td>Address of the computer where the database server is located. This must refer to the local PC, so either 'localhost' or '127.0.0.1'.</td>
</tr>
<tr>
<td>Server port</td>
<td>Port that is used to communicate with database server. Typically port 3306 is used.</td>
</tr>
<tr>
<td>User name</td>
<td>Name of the user used to login to the database server.</td>
</tr>
<tr>
<td>User password</td>
<td>Password that is used to login to database server. This password is stored without encryption.</td>
</tr>
<tr>
<td>Database</td>
<td>Name of the database to access. In case more than one eXLerate application uses the database server, the database for each application can be configured to be the same or a different database.</td>
</tr>
</tbody>
</table>

Note: definition of the external system database overrules the definition of the embedded system database.

System tables

The system database contains the following tables that are required for eXLerate:
- EVENTS – used for storing alarm transitions, system events and application-specific events.
- COMMAND_LOG – used for data synchronization between redundant servers

**System table EVENTS**
The ‘Events’ table is used to store the alarms & events history. It must contain at least the following fields:

<table>
<thead>
<tr>
<th>Column</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>BIGINT</td>
<td>Unique ID of the event</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>Date and time the event was stored.</td>
</tr>
<tr>
<td>CLASS</td>
<td>VARCHAR(32)</td>
<td>Event class (e.g. ‘Alarms’, ‘Security’, ‘Parameter’)</td>
</tr>
<tr>
<td>TYPE</td>
<td>VARCHAR(32)</td>
<td>Event type (e.g. ‘Ack’, ‘Logoff’, ‘Change’)</td>
</tr>
<tr>
<td>LOCATION</td>
<td>VARCHAR(64)</td>
<td>Location (e.g. ‘FC01_PRESPRVIN_LOFAULT’, ‘MSC-A’)</td>
</tr>
<tr>
<td>USER</td>
<td>VARCHAR(64)</td>
<td>Name of the user or ‘System’</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>VARCHAR(255)</td>
<td>Event message</td>
</tr>
<tr>
<td>EXTRA1</td>
<td>VARCHAR(128)</td>
<td>Additional event field 1</td>
</tr>
<tr>
<td>EXTRA2</td>
<td>VARCHAR(128)</td>
<td>Additional event field 2</td>
</tr>
</tbody>
</table>

Table 8.1 Required fields for the 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 application is identified by its unique database ID. The ID ‘0’ always refers to the system database:

- Stand-alone system Database ID ‘0’ identifies the local system database.
- Server/Client system Database ‘0’ identifies the system database on the duty server.

If the application needs to access the local database regardless of whether the current server is duty or standby, then it can use the database identifier “local_embedded”. It is not recommended to access the “local_embedded” database ID.

Besides of the system database the application can connect to other databases referred further as foreign databases. Foreign databases are not synchronized. To connect to foreign databases you use the worksheet function ‘exConfigureDatabase(…)’, usually defined on a worksheet named ‘xExtDB’, ‘xDatabases’ or ‘xSQLTables’. Foreign databases must have an ID ‘1’ or higher.

**User definable tables**
eXLerate can be extended with user definable tables. For the system database the ‘MyISAM’ storage engine must be used. ‘MyISAM tables are stored in the form of files that can be copied from one database to another. A single table consists of the following 3 files:

<table>
<thead>
<tr>
<th>File-name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;table&gt;.FRM</td>
<td>MyISAM Definition File</td>
</tr>
<tr>
<td>&lt;table&gt;.MYD</td>
<td>MyISAM Data File</td>
</tr>
<tr>
<td>&lt;table&gt;.MYI</td>
<td>MyISAM Index File</td>
</tr>
</tbody>
</table>

Table 8.3: MySQL table files

When copying table files from one database server to another, make sure that both database servers are stopped. In case of the eXLerate system database, make sure the application is completely shut down. When copying to or from an external database server, make sure the database server program or 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:

<table>
<thead>
<tr>
<th>Info value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql_error</td>
<td>MySQL specific error-code.</td>
</tr>
<tr>
<td>mysql_time</td>
<td>Time in seconds it took MySQL to execute the SQL statement.</td>
</tr>
<tr>
<td>thread</td>
<td>ID of the thread that executed the SQL statement.</td>
</tr>
<tr>
<td>affected_rows</td>
<td>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).</td>
</tr>
</tbody>
</table>

Table 8.4: Embedded database driver specific info values

**Redundancy & synchronization**
When multiple servers are used, the system 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 adatabase. 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:

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrent_connections</td>
<td>5</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 8-5: 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 date 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 shut down before opening the file, otherwise an error message is shown that the file cannot be accessed.

Foreign databases
Besides of the (embedded or external) system database, the eXLERATE application may connect to one or more additional databases by configuring a number of worksheet functions.

Configuration
This chapter does not cover the setup and configuration of 3rd party database platforms. Instead, it describes the tools supported by eXLERATE in order to interface with these 3rd party databases.

⚠️ Foreign databases can be configured using two worksheet functions: ‘exConfigureDatabase(…’) and ‘exSetDatabaseProperty(…)’.

The sample application MyDatabase.xlrx contains an example on how to configure a foreign database. The ‘xExtDB’ worksheet provides a template for configuring foreign databases.

<table>
<thead>
<tr>
<th>ID</th>
<th>Type</th>
<th>Host</th>
<th>Database</th>
<th>User</th>
<th>Password</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>mysql</td>
<td>localhost</td>
<td>eXLERATE</td>
<td>root</td>
<td>vendor</td>
<td>3306</td>
</tr>
</tbody>
</table>

Figure 8.3 Table of foreign databases

All databases in eXLERATE can be accessed through Database ID’s. When configuring a foreign 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 foreign database. In the example above, the ‘mysql’ driver is used to communicate with a database named ‘eXLERATE’.

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
MySQL is one of the leading databases and there are several different database servers that support the MySQL database, like MariaDB.
Configuration properties
In order to configure a foreign 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>-</td>
<td>Address of the computer where the MySQL server is located. This can be either the computer-name or an IP-address.</td>
</tr>
<tr>
<td>User</td>
<td>-</td>
<td>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.)</td>
</tr>
<tr>
<td>password</td>
<td>-</td>
<td>Password that is used to login into MySQL.</td>
</tr>
<tr>
<td>Port</td>
<td>-</td>
<td>Port that is used to communicate with MySQL. By default when MySQL is installed it uses port 3306.</td>
</tr>
<tr>
<td>Database</td>
<td>-</td>
<td>Name of the database to access at the MySQL server.</td>
</tr>
<tr>
<td>Concurrent_connections</td>
<td>5</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 8.6 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:

<table>
<thead>
<tr>
<th>Info value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mysql_error</td>
<td>MySQL specific error-code.</td>
</tr>
<tr>
<td>Mysql_time</td>
<td>Time in seconds it took MySQL to execute the SQL statement.</td>
</tr>
<tr>
<td>Thread</td>
<td>ID of the thread that executed the SQL statement.</td>
</tr>
<tr>
<td>Affected_rows</td>
<td>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).</td>
</tr>
</tbody>
</table>

Table 8.7 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.

Configuration properties
In order to configure a foreign 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.

<table>
<thead>
<tr>
<th>Property</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>-</td>
<td>Address of the computer where the SQLServer server is located. This can be either the computer-name or an IP-address.</td>
</tr>
<tr>
<td>user</td>
<td>-</td>
<td>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.)</td>
</tr>
<tr>
<td>password</td>
<td>-</td>
<td>Password that is used to login into SQLServer.</td>
</tr>
<tr>
<td>database</td>
<td>-</td>
<td>Name of the database to access at the SQLServer.</td>
</tr>
<tr>
<td>Concurrent_connections</td>
<td>5</td>
<td>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.</td>
</tr>
</tbody>
</table>

Table 8.8 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:

<table>
<thead>
<tr>
<th>Info value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oledb_error</td>
<td>OLEDB specific error-code.</td>
</tr>
<tr>
<td>oledb_time</td>
<td>Time in seconds it took the OLEDB driver to execute the SQL statement.</td>
</tr>
<tr>
<td>thread</td>
<td>ID of the thread that executed the SQL statement.</td>
</tr>
<tr>
<td>affected_rows</td>
<td>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).</td>
</tr>
</tbody>
</table>

Table 8.9 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:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exSQLCreateQuery(...)</td>
<td>Create a query with a specific ID and options.</td>
</tr>
<tr>
<td>exSQLExecQuery(...)</td>
<td>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.</td>
</tr>
<tr>
<td>exSQLExecRangeQuery(...)</td>
<td>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.</td>
</tr>
<tr>
<td>exSQLExecRecordQuery(...)</td>
<td>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.</td>
</tr>
<tr>
<td>exSQLField(...)</td>
<td>Returns a single field of the result set of the corresponding query.</td>
</tr>
<tr>
<td>exSQLRowCount(...)</td>
<td>Returns the number of rows returned by the corresponding query.</td>
</tr>
<tr>
<td>exSQLColumnCount(...)</td>
<td>Returns the number of columns returned by the corresponding query.</td>
</tr>
<tr>
<td>exSQLLastCount(...)</td>
<td>Retrieve the last number of columns returned by the corresponding query.</td>
</tr>
<tr>
<td>exSQLLastDriverSpecificInfo(...)</td>
<td>Returns a driver specific value of the corresponding query. This function can be used to obtain for instance, the MySQL specific error-code if accessing a MySQL database server.</td>
</tr>
<tr>
<td>exSQLDiagnosticalValue(...)</td>
<td>Returns a diagnostic 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.</td>
</tr>
</tbody>
</table>

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.

The following worksheet functions are available for SQL Views:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exSQLViewQuery(...)</td>
<td>Views a selection of a previously executed SQL query. The selection can be specified by using row- and column-offsets.</td>
</tr>
</tbody>
</table>

Table 8.11 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:

The ‘SQL table’ consists of the following configurable ‘SQL properties’:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Unique ID of a query. When an entry in the SQL Table is configured, a query with that particular ID is created. The ‘exSQLExecQuery(...)’ worksheet function can be used to access the result of a SQL statement.</td>
</tr>
<tr>
<td>Database</td>
<td>Database ID (0 = System, 1..n = Foreign). See ‘exSQLExecQuery(...)’ worksheet function for more details.</td>
</tr>
</tbody>
</table>

Table 8.10 Implemented SQL query functions

Figure 8.4: SQL query & 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 furtheron in this chapter.

The following worksheet functions are available for SQL Views:

<table>
<thead>
<tr>
<th>Table 8.11 Implemented SQL view functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>exSQLViewQuery(...)</td>
</tr>
</tbody>
</table>

Figure 8.5 SQL table
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Statement</td>
<td>SQL Statement to execute (cannot exceed 255 characters). See 'exSQLExecQuery(…)' worksheet function for more details.</td>
</tr>
<tr>
<td>Update Interval</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Table 8.12 SQL table property fields**

When an SQL statement is executed, its status information is shown in the ‘SQL Results’ part of the ‘SQL table’:

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

**Table 8.13 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.

```
=exSQLViewQuery( Query ID, Update Trigger, Current position + 1 )
```

**Step 1 : Create worksheet variables**

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

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

Select the range where the scrollable text should appear:

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

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

**Example:**

```
=exSQLViewQuery(18, xNow.Time, 15+1)
```

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

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

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

Right-click the scrollbar and select ‘Format Control…’.

![Figure 8.11 Formatting the scrollbar](image)

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

![Figure 8.12 Attach scrollbar to variable](image)

**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 straightforward 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:

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

![Figure 8.13 Update scrollbar maximum value](image)

The minimum-value should remain ‘0’. The maximum value is calculated as follows: `<maximum value> = <query row count> - <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:

```vba
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:

```vba
' 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:

```vba
' 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. A-synchronously 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=system database, 1..n=foreign 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 system database and waits for a maximum time of 10 seconds for it to complete:

```vba
' 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

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 8.14 Synchronously executed SQL statements

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

Figure 8.15 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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExecuteAsync</td>
<td>Executes an SQL statement on a database and returns immediately.</td>
</tr>
<tr>
<td>WaitForAsync</td>
<td>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.</td>
</tr>
<tr>
<td>CancelAsync</td>
<td>Cancels the last SQL statement executed by ‘ExecuteAsync’.</td>
</tr>
<tr>
<td>IsAsyncInProgress</td>
<td>Returns ‘True’ when an a-synchronous operation is still in progress.</td>
</tr>
</tbody>
</table>

Table 8.14: 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:

```vba
' 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.
9 Servers, clients & printers

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

Using multiple servers is discussed in 10 Redundancy.

Introduction

You may have multiple computers with the same functionality to protect against a single failure or to allow operation control on different locations. An eXLerate project configuration allows this functionality by the Server & Client configuration. You can have the following types of computers in an eXLerate environment: Standalone, Servers and Clients. A standalone computer communicates directly with IO devices, generates reports and maintains an internal database for alarms & events but does not connect to or shares data with other computers. Servers can communicate directly with IO devices, generate reports and maintain an internal database for alarms & events and can connect to other servers and client to share and synchronize data. A Client visualizes the data and allows user interactions, but it doesn’t communicate directly with an IO device or generate reports periodically. For all the functions a client doesn’t perform, it reads the results from the Servers. From the end-user point of view, all servers as well as all clients provide full operator functionality.

For the servers, a duty/standby concept is used. Only one server is the “duty” server, the others are considered “standby”. If the “duty” server fails, a standby server will take over the role of “duty” server without any hiccups. The standby server(s) continuously synchronize all the data from the duty server such as reports, averages, and database.

Client computers can be either dedicated client or non-dedicated client. A dedicated client is explicitly configured in the application and can be monitored by others. A non-dedicated client is configured in the application but still can connect to the servers and participate, however it cannot be monitored by the other. A non-dedicated client can for instance, be a service- or engineering laptop which is only used when necessary.

When using eXLerate to develop server/client systems it is not necessary to create separate applications for these different types. Instead, a single application will serve all these different functions. 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 sheet (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 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 can be found in the ‘MyNet’ sample application. If you want to implement client/server functionality and your application does not contain the (latest) ‘xNet’ sheet, you insert it by opening the ‘MyNet.xlrx’ as a second workbook in the eXLerate application you want to add/update the functionality (so both files are open in the same excel instance). The ‘xNet’ sheet and related functionality will be automatically inserted into your application. If an old ‘xNet’ worksheet is already available in the application, it will be updated.

The ‘xNet’ worksheet automatically detects the installed license:

<table>
<thead>
<tr>
<th>Network Configuration Sheet</th>
<th>V2.4</th>
<th>License: Server</th>
</tr>
</thead>
</table>

Figure 9.1 xNet license detection

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

- Server license
- Client license

In case of a single computer with no Client/Server functionality required, the stand-alone license shall be used.
Server/Client configuration

To configure servers and clients, 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 computers and optionally specify the IP-addresses in the **Project Configuration** section. Default the sheet supports up to 4 Servers and 8 Clients - up to 4 network addresses each.

<table>
<thead>
<tr>
<th>Name</th>
<th>IP1</th>
<th>IP2</th>
<th>IP3</th>
<th>IP4</th>
<th>Allow/Duty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server 1</td>
<td>192.168.1.1</td>
<td>192.168.2.1</td>
<td></td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>Server 2</td>
<td>192.168.2.1</td>
<td>192.168.2.2</td>
<td></td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>Server 3</td>
<td>192.168.3.1</td>
<td>192.168.3.2</td>
<td></td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>Server 4</td>
<td>192.168.4.1</td>
<td>192.168.4.2</td>
<td></td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>Client 1</td>
<td>192.168.1.11</td>
<td>192.168.2.11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 2</td>
<td>192.168.1.12</td>
<td>192.168.2.12</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client 8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9.2 Server configuration**

The columns can be configured as follows:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Computer name as configured in Windows (Control Panel\System and Security\System). Note: this is not the system name as configured in the eXLerate Control Center.</td>
</tr>
<tr>
<td>IP 1</td>
<td>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.</td>
</tr>
<tr>
<td>IP 2..4</td>
<td>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.</td>
</tr>
<tr>
<td>Allow Duty</td>
<td>Specifies if a server can become the Duty Server. Only Duty Servers are allowed to communicate directly with IO devices. If this is set to FALSE, the Server cannot become duty, but it will have the synchronized databases and reports locally, in contrast with clients that retrieve the data from the Duty server and not retain it locally.</td>
</tr>
</tbody>
</table>

**Table 9.1 Server/Client configurable fields**

The ‘xNet’ worksheet has a second section **Test configuration** to set-up computers and IP addresses for testing purposes. When enabling ‘Alternative configure for testing’, these computer names and IP addresses will be used instead of the project set-up. It allows you to test the application without having to change the project configuration.

Whether the server IP address configuration is correct can be validated using the ‘xNet’ worksheet. The **Network configuration table** shows the status for every IP address that is configured.

**Figure 9.3 Server communication statuses**

<table>
<thead>
<tr>
<th>Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>No computer/IP address configured</td>
</tr>
<tr>
<td>3</td>
<td>Not started eXLerate communication has not been started</td>
</tr>
<tr>
<td>2</td>
<td>Not connected IP address cannot be found on the network</td>
</tr>
<tr>
<td>1</td>
<td>No heartbeat IP address is valid, but the application can’t connect to the remote application</td>
</tr>
<tr>
<td>0</td>
<td>Running The application can connect to and communicate with eXLerate application on the IP address specified</td>
</tr>
</tbody>
</table>

**Table 9.2 Network status**

**Step 2: Share reports for clients**

Clients do not have the reports locally. Clients do not generate these files; instead they read them from the duty server. The same is true for database and trend files. For the reports, the location of the files need to be shared by the use of a drive-mapping to the server(s).

First create a file-share should for the eXLerate Report folders on the servers:

![File-share for eXLerate Report folders](image)

Then on the Client, create a network drive-mapping for each server to which the client can connect to:
Printers configuration

The ‘xNet’ worksheet also allows to configure up to 4 printers. Specify the name of the printer and optionally specify the IP- addresses in the Project configuration section. You have to select if the printer is used as a Report or as an Event printer. When for a type multiple printers are available, the printer which has the best status will be selected as the “in use” printer (priority from Printer 1 down to Printer 4). These “in use” printers can be referred by the names Net.ReportPrinter and Net.EventPrinter in your application.

```
<table>
<thead>
<tr>
<th>Name</th>
<th>IP #1..4</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Printer 1</td>
<td></td>
<td></td>
<td>Printer name as configured in Windows.</td>
</tr>
<tr>
<td>Printer 2</td>
<td></td>
<td></td>
<td>Optional IP address by which the printer can be accessed.</td>
</tr>
<tr>
<td>Printer 3</td>
<td></td>
<td></td>
<td>Specifies the printer to be used as Report printer or as Events printer</td>
</tr>
<tr>
<td>Printer 4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Table 9.3 Server configurable fields

If multiple servers are used, each server should have its own set of network drive mappings. These drive mappings should be separated using a ‘|’ character as shown above. Any spaces that are used before or after the ‘|’ character are ignored. The order is always as Server 1 | Server 2 and so on.

Advanced settings

The ‘xNet’ worksheet contains a separate section for advanced settings. The following advanced settings are supported:

```
<table>
<thead>
<tr>
<th>Setting</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port</td>
<td>9666</td>
<td>TCP/IP port that is used for communicating with other clients and servers.</td>
</tr>
<tr>
<td>Update Interval</td>
<td>3</td>
<td>Calculation cycle time in seconds for Network functions on this sheet.</td>
</tr>
<tr>
<td>Switchover period</td>
<td>10</td>
<td>Time in seconds the system is forced in a duty-selection when switching over (minimal 3 times the update interval).</td>
</tr>
<tr>
<td>Local Override</td>
<td>0</td>
<td>When a value is set, the computer is forced in a fixed duty selection, e.g. when set to ‘2’ the 2nd server will be forced to be the duty server.</td>
</tr>
<tr>
<td>Time synchronization</td>
<td>3</td>
<td>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 below.</td>
</tr>
<tr>
<td>Time synchronization</td>
<td>30</td>
<td>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.</td>
</tr>
<tr>
<td>Cell Naming</td>
<td>Single cell names</td>
<td>Cell naming used for reference in application: “Range names” (ref with INDEX function) OR “Single cell names” (more names, for backwards compatibility) – see section Application development below for detailed explanation.</td>
</tr>
<tr>
<td>Max. Print Jobs</td>
<td>5</td>
<td>Maximum number of print jobs allowed for the printers.</td>
</tr>
</tbody>
</table>
```

Table 9.5 Advanced settings

Application development

Names

The ‘xNet’ worksheet provides an elaborate set of names which can be used in your application. All the names provided by the ‘xNet’ worksheet are read-only. The names available depends on the Cell Naming setting.
When set to Single cells names the following names are available:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net.Server[x].Name</td>
<td>Computer name of the Server (x), with (x) being the server number 1, 2, ...</td>
</tr>
<tr>
<td>Net.Server[x].IP1</td>
<td>IP addresses for Server (x).</td>
</tr>
<tr>
<td>Net.Server[x].IP2</td>
<td></td>
</tr>
<tr>
<td>Net.Server[x].Status</td>
<td>Overall communication status of the server (x), as a number between 0 and 4</td>
</tr>
<tr>
<td>Net.Server[x].Status1</td>
<td>Communication status of a specific IP address (network adapter).</td>
</tr>
<tr>
<td>Net.Server[x].Status2</td>
<td></td>
</tr>
<tr>
<td>Net.Server[x].Duty</td>
<td>ID of the duty server which is selected by that particular server; '0' if</td>
</tr>
<tr>
<td>Net.Server[x].Name</td>
<td>if no duty is selected by the server.</td>
</tr>
<tr>
<td>Net.Client[x].IP1</td>
<td>IP addresses for Client (x).</td>
</tr>
<tr>
<td>Net.Client[x].IP2</td>
<td></td>
</tr>
<tr>
<td>Net.Client[x].Name</td>
<td>Computer name of the Client (x), with (x) being the Client number 1, 2, ...</td>
</tr>
<tr>
<td>Net.Client[x].Status</td>
<td>Overall communication status of the Client (x), as a number between 0 and 4</td>
</tr>
<tr>
<td>Net.Client[x].Status1</td>
<td>Communication status of a specific IP address (network adapter).</td>
</tr>
<tr>
<td>Net.Client[x].Status2</td>
<td></td>
</tr>
<tr>
<td>Net.Client[x].Duty</td>
<td>ID of the duty server which is selected by that particular Client; '0' if</td>
</tr>
<tr>
<td>Net.Printer[x].Name</td>
<td>if no duty is selected by the Client.</td>
</tr>
<tr>
<td>Net.Printer[x].IP1</td>
<td>IP addresses for Printer (x).</td>
</tr>
<tr>
<td>Net.Printer[x].IP2</td>
<td></td>
</tr>
<tr>
<td>Net.Printer[x].Status</td>
<td>Overall communication status of the Printer (x), as a number between 0 and 4</td>
</tr>
<tr>
<td>Net.Printer[x].Status1</td>
<td>Communication status of a specific IP address (network adapter).</td>
</tr>
<tr>
<td>Net.Printer[x].Status2</td>
<td></td>
</tr>
</tbody>
</table>

Table 9.6 Single Cell Names

When using the single cell names, you can directly refer to the individual cell values. For example, if you want to refer to the overall status of server 2, use a formula like:

=INDEX(Net.Servers.Status{2,1})

Independent if the Cell naming setting, the following global names are available:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Net.Configured</td>
<td>Check if the license is eXLerate Server Edition or eXLerate Client edition</td>
</tr>
<tr>
<td>Net.Started</td>
<td>Check if network is configured and if the communication is started</td>
</tr>
<tr>
<td>Net.StartTime</td>
<td>Time when the communications were started</td>
</tr>
<tr>
<td>Net.UpdateTime</td>
<td>Time of last calculation cycle for network functions</td>
</tr>
<tr>
<td>Net.WaitTimeExpired</td>
<td>Checks if the system is still waiting for Duty Server determination</td>
</tr>
<tr>
<td>Net.Local.ServerID</td>
<td>ID of the local server or '0' if the local computer is not a server.</td>
</tr>
<tr>
<td>Net.Local.ClientID</td>
<td>ID of the local client or '0' if the local computer is not a client.</td>
</tr>
<tr>
<td>Net.Local.IsClient</td>
<td>TRUE when the local computer is a client.</td>
</tr>
<tr>
<td>Net.Local.IsDutyServer</td>
<td>TRUE when the local computer is the duty server.</td>
</tr>
<tr>
<td>Net.Duty.ID</td>
<td>ID of the currently selected duty server, or '0' if no duty is selected.</td>
</tr>
<tr>
<td>Net.Duty.Name</td>
<td>Computer name of the currently selected duty server, or ‘None' if no duty is selected.</td>
</tr>
<tr>
<td>Net.Duty.SwitchOver.Timer</td>
<td>Timer counting when Duty switch-over occurs to prevent 2nd switch-over occurring within the Switchover period</td>
</tr>
</tbody>
</table>

Table 9.8 xNet global names

Pre-defined animations

The ‘xNet’ sheet contains pre-defined animations for the status of the Servers, clients and printers. The animations are based on grouped shapes: 1 grouped shape for each Server/Client/Printer.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server[x]</td>
<td>Main name of the (grouped) object for server (x), with (x) being the server number 1, 2, ... Set visible when name is set for corresponding Server, else set to invisible</td>
</tr>
<tr>
<td>Server[x].Link1</td>
<td>Sub-shape of the main object to show a specific link. Set visible when an IP address is set for this network, else set to invisible.</td>
</tr>
<tr>
<td>Server[x].Link2</td>
<td></td>
</tr>
<tr>
<td>Server[x].Flt1</td>
<td>Sub-shape of the main object to show the status of a specific link. Set visible when communication over an IP address fails for this network, else set to invisible.</td>
</tr>
<tr>
<td>Server[x].Flt2</td>
<td></td>
</tr>
</tbody>
</table>
The following example shows a network overview of 2 servers and 2 clients. Both the servers and the clients have 2 redundant network adapters.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client{x}</td>
<td>Main name of the (grouped) object for Client {x}, with {x} being the Client number 1, 2, ... Set visible when name is set for corresponding Client, else set to invisible</td>
</tr>
<tr>
<td>Client{x}.Link1</td>
<td>Sub-shape of the main object to show a specific link. Set visible when an IP address is set for this network, else set to invisible.</td>
</tr>
<tr>
<td>Client{x}.Link2</td>
<td>Sub-shape of the main object to show a specific link. Set visible when an IP address is set for this network, else set to invisible.</td>
</tr>
<tr>
<td>Client{x}.Flt1</td>
<td>Sub-shape of the main object to show the status of a specific link. Set visible when communication over an IP address fails for this network, else set to invisible.</td>
</tr>
<tr>
<td>Client{x}.Flt2</td>
<td>Sub-shape of the main object to show the status of a specific link. Set visible when communication over an IP address fails for this network, else set to invisible.</td>
</tr>
<tr>
<td>Printer{x}</td>
<td>Main name of the (grouped) object for Printer {x}, with {x} being the Printer number 1, 2, ... Set visible when name is set for corresponding Printer, else set to invisible.</td>
</tr>
<tr>
<td>Printer{x}.Flt</td>
<td>Sub-shape of the main object to show the status of a specific link. Set visible when communication over an IP address fails for this network, else set to invisible.</td>
</tr>
<tr>
<td>Printer{x}.Status</td>
<td>Sub-shape of the main object to show the status of a specific link. Status indication set to RED when printer reports an error / YELLOW when printer reports a warning, else set to invisible.</td>
</tr>
<tr>
<td>Printer{x}.Busy</td>
<td>Sub-shape of the main object to show the status of a specific link. Status indication set to visible when printer is busy, else set to invisible.</td>
</tr>
</tbody>
</table>

Table 9.9 Single cell names

Event logging
In client/server systems all events are logged to the database. This database is stored servers and all the servers have a synchronized copy of this database. Events are also locally logged in the eXLerate 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:
Table 9.11 Conditional event logging

### Writing to IO devices
Writing to IO devices is possible from both servers and clients. Using the functions ‘exUpdateEx(...)', ‘exUpdateStrEx(...)' and ‘exUpdateVarEx(...)' it is possible to write a value to an IO device. These functions are available both on worksheets as in VBA. When these functions are used on a worksheet they will be executed on all servers and clients, causing 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:
  ```vba
  =exUpdateEx(iQuery, Item, dValue, UpdateMode)
  ```

- Client/server; only write on duty server:
  ```vba
  =IF(Net.Local.IsDutyServer, exUpdateEx(iQuery, Item, dValue, UpdateMode), FALSE)
  ```

In VBA the function `exUpdateForce` is available and commonly to write values to an external device. This function is not available on worksheets.

### 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:

<table>
<thead>
<tr>
<th>Server 1</th>
<th>Client 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
</tr>
</tbody>
</table>

A new parameter value is entered on “Server 1”:

- `exStoreValue(MyParam, "MyParam", xRegKey & \"AutoLoad\", .... xAutoRecalc)`

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

| 100 | TRUE | 100 | TRUE |

Figure 9.12 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 parameter, 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.

```
=exShareValue("MySharedValue","xComputerNames="MDC-1",L30,xNow.Time)
```

Figure 9-13: 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:

```
=exStoreValue("MySharedValue","xComputerNames="MDC-1",L30,xNow.Time)
```
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:

```vba
' 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
```

Figure 9.14 Shared value on multiple computers
10 Redundancy

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

Introduction

Spirit® 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. Spirit® 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 role 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. Spirit® 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:
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:

```
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:
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:

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:

The end result would be the following protocol table with 3 additional columns on the end:

When looked at in detail, the new columns contain the following worksheet functions:

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:
Case 3:

' Check redundant communication lines
CheckComPorts

End Select
Exit Sub

Figure 10.11 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

' Write new device index to worksheet
If iCurrentPrimaryDevIndex <> iNewPrimaryDevIndex Then

' Switch primary protocol to other device
Range("xComms!AI4").Value = iNewPrimaryDevIndex

End If

End Sub

Figure 10.12: Protocol device selection code

Server redundancy

Introduction
SpiritIT 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:
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:

![Figure 10.14 Local Duty Status](image)

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 work around 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:
The additional columns contain the following formulas:

\[ = \text{exQueryStatus}( \langle \text{Query ID}, \langle \text{Server ID}, <\text{Trigger} \rangle \rangle) \]

\[ = \text{exQueryServer}( \langle \text{Query ID}, \langle \text{Server ID}, <\text{Trigger} \rangle \rangle) \]

\[ = = \langle \text{routing formula} \rangle \]

Figure 10.18 Query table routing formula

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:

\[ \text{If no duty server is selected} \]
\[ \text{If( Net.Duty.ID = 0 ) Then} \]
\[ \text{Then always use the local IO} \]
\[ \text{result = 0} \]
\[ \text{Else} \]
\[ \text{Is server 1 duty? and it is communicating OK?} \]
\[ \text{If( Net.Duty.ID = 1 And Query.Status.Server1 = 0 )} \]
\[ \text{Yes it is, lets route the query through that server} \]
\[ \text{result = 1} \]
\[ \text{Else} \]
\[ \text{Is server 2 duty? and is it communicating OK?} \]
\[ \text{If( Net.Duty.ID = 2 And Query.Status.Server2 = 0 )} \]
\[ \text{Yes it is, lets route the query through that server} \]
\[ \text{result = 2} \]
\[ \text{Else} \]
\[ \text{No valid query found, use local IO instead} \]
\[ \text{result = 0} \]
\[ \text{End If} \]
\[ \text{End If} \]

Figure 10.19 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.

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
This setup provides redundancy in the form 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**

![Redundant network cards on separate networks](image)

Figure 10.22 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).
11 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.

When using East-Asian languages make sure that the Supplemental language support is installed before installing the language packs:

![Supplemental language support](image1.png)

The language packs can be installed from the ‘Advanced’ tab on the ‘Regional and Language Options’ dialog:

![Installing language packs](image2.png)

Figure 11.1 Regional and language options

Figure 11-2: Supplemental language support

Figure 11.3 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 11.4 Skip file copying during language pack install

After a new language has been installed the computer needs to be restarted:

Figure 11.5 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.

Figure 11.6 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:

When ‘Run’ is pressed, the ‘xLanguage’ worksheet is automatically created:

Figure 11.7 Creating a new language

The newly created worksheet will however have no default formatting, so this will have to be applied manually:

Figure 11.8 Language wizard output

Figure 11.9 Language worksheet without formatting

**Language worksheet layout**

The layout of the ‘xLanguage’ worksheet is as follows:

Figure 11.10 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:

<table>
<thead>
<tr>
<th>2</th>
<th>System</th>
</tr>
</thead>
</table>

Or a language row:

<table>
<thead>
<tr>
<th>8</th>
<th>Alarms\Dialogs\Suppress\SuppressButton</th>
<th>Suppress</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>Alarms\Dialogs\Suppress\SuppressGroup</td>
<td>Suppress current group: '%GROUP%'</td>
</tr>
</tbody>
</table>

A class rows 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:

<table>
<thead>
<tr>
<th>Chinese</th>
</tr>
</thead>
</table>

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:\XLRLanguages.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:

![Open the 'Languages.xls' file](image1)

Select the worksheet containing the desired language:

![Select desired language](image2)

Select the column containing the language and copy it to the clipboard:

![Select language column](image3)

Go back to the original application and select the 'xLanguage' worksheet:

![Select 'xLanguage' worksheet](image4)

Select the first empty column:

![Locate first empty column in language worksheet](image5)

Use 'Paste' to add the language to the worksheet. The end-result is a language worksheet with the desired language:

![Succesfully added language](image6)
After any modifications to the language worksheet, use “Apply Worksheet Changes” before testing the new language:

Figure 11.18 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:

Figure 11.19 User defined texts in Language worksheet

eXLerate will interpret all user defined texts until two or more empty rows are encountered:

Figure 11.20: 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:

Figure 11.21 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 11.22 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 11.23 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.

Figure 11.24 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:
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:

```vba
Sub SelectLanguage_Dutch()
    Range("xLangSelection").Value = "Dutch"
    exSetButtonText
End Sub
```

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.

```vba
=exLanguageText("Legend\\Pressure", xLangRecalc)
```

The following example shows the result of the ‘exLanguageText(…)’ worksheet function:

Default (English) Dutch

![Control in Dutch]

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:

```vba
Private Sub UserForm_Activate()
    'Set multi-lingual texts
    Caption = exLanguageText("Dialogs\\Reports\\Caption")
End Sub
```

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 11.28 Linking a control to a cell or name

Default (English) Dutch

![Control in Dutch]
```

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")
```

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")
End Sub
```
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:

```
= xLangSelection
```

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
```

In order to make the selection possible from a worksheet, a button or picture can be used:

![Picture object]

This button or picture should be assigned to the language selection macro:
12 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:

![Terminal Services Setup Diagram](image)

Figure 12.1 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 7, Windows 8.1 and Windows 10, 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 either Windows Server 2008 R2 or Windows Server 2012 or Windows Server 2012 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.

<table>
<thead>
<tr>
<th>Office</th>
<th>Editions &amp; Licenses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Office 2019</td>
<td>In order to use Office 2019 or Office 2016 or Office 2013 or Office 2010 in combination with Terminal Services you need a Volume License. (<a href="http://www.microsoft.com/licensing">http://www.microsoft.com/licensing</a>)</td>
</tr>
<tr>
<td>Office 2016</td>
<td></td>
</tr>
<tr>
<td>Office 2013</td>
<td></td>
</tr>
<tr>
<td>Office 2010</td>
<td></td>
</tr>
</tbody>
</table>

Table 12.1 Required Microsoft Office licenses for Terminal Services

**SpiritIT 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.

<table>
<thead>
<tr>
<th>#</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Install ‘Remote Desktop Services’ role on Windows Server.</td>
</tr>
<tr>
<td>2</td>
<td>Purchase and install ‘RDS CALs’ for the users or computers which are to connect to the server.</td>
</tr>
<tr>
<td>3</td>
<td>Set the remote desktop services setting ‘Restrict each user to a single session’ to ‘No’.</td>
</tr>
<tr>
<td>4</td>
<td>Create a Windows User called ‘eXLerate_Remote’ and give it ‘Administrator’ rights.</td>
</tr>
</tbody>
</table>

Table 12.2: Operating System Configuration

**SpiritIT 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:

![Enable Terminal Services Mode Dialog](image)

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 wish to connect to. Please read 9 Servers, clients & 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:

![Remote Desktop Connection](image1)

Figure 12.2 Locating the remote desktop client

After pressing ‘Enter’ the ‘Remote desktop connection’ window appears:

![Remote Desktop Connection Window](image2)

Figure 12.3 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
```
13 Troubleshooting

Introduction

This chapter is meant for troubleshooting 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 ass 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 ABB at (nl-spiritit-support@abb.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 ABB 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 ABB, or your local distributor directly to discuss your special requirements.
- ABB 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 13.1 Example of a VB function causing excessive calling
Instead, the function should be called as follows:
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.

Minimize the Control Center when your application does not start up properly.

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.

Run the 'Workbook_Open()' macro manually. This causes eXLerate to start up normally after all.
14 Revisions

Revision A
Date February 2012
- Initial release of eXLerate 2010 Reference Manual Volume II.
- Added ‘Import Sheets’ and ‘Advanced Replace’ documentation.

Revision B
Date December 2016
- Update to eXLerate 2016.
- Update to ABB lay-out.
- New document code: COI/eXL2016-EN.

Revision C
Date September 2018
- New document code: CM/eXL-EN.
- Reintroduce revisions chapter.
- Provisional support for Microsoft Office 2019 is added.
- Windows 8 removed from software requirements.
- Clarified exact behavior of limit alarms at set points.

Revision D
Date April 2019
- Support for Microsoft Office 2019 added.
- Updated chapter Relational Databases