CRT

ACCURATE MEASUREMENT

Technical Library

As a Service to the HydroCarbon Measurement Industry, CRT-SERVICES Curates this collection of Digital resources.

www.crt-services.com
www.crtsupply.com

11133 Interstate 45 S Suite O
Conroe, Texas 77302
(713) 242-1190

eXLerate 2010

For Microsoft Excel

Application Reference Manual

Introduction, Tutorial
Application Development
Report Generation

Product	eXLerate 2010 application reference manual
Reference number	03-0110-1
Revision	A.0
Date	November 2011
Authors	H.A.J. Kok, H.F.J. Rutjes, J.A.M. de Greef

Disclaimer

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

Special note

Printed in the Netherlands.

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

Table of contents

List of figures

List of tables

List of equations

Document Control

Revision Coding

All documents are supplied with a revision code. This code has the following format:
<major revision letter>.<minor revision number>. Initially, the document has revision code A.0. When in the next release of the document minor changes were implemented, the minor revision number increases. When major changes have been implemented, the major revision number increments.

Example document:

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

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

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

Example program:

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

Revision History

Revision A.0

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

Initial release of the eXLerate 2010 Reference Manual Volume I.

Chapter 1 - Introduction to eXLerate

Manual set

Welcome to the exciting world of eXLerate 2010!

Using 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:

- *This 'Application Reference Manual', with the installation and setup guide, a tutorial, the control center reference, and application development guide.*
- *The '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.*

In the first chapter of this manual, the eXLerate software is introduced to a novice user.

Advantages

With eXLerate, 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 eXLerate, 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® Visio, or even AutoCAD®.

- 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 **eXLerate**.
- 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 **eXLerate** 2010 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 **eXLerate**.

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 **eXLerate** 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 chapter, an introduction is given to this 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 **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 **eXLerate** and the basic ingredients of a complete application, and is added to this reference to quickly start application development in **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 **eXLerate** Control center program.

In chapters 5-13: '*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: '*Report generation*' 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 **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 an 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:

AI	**A**nalog **I**nput
AO	**A**nalog **O**utput
API	**A**pplication **P**rogramming **I**nterface An interface that allows an application to interact with an application or operating system, in our case, eXLerate. Most of the eXLerate API is implemented through Excel worksheet functions.
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange. 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 (used by Windows 3.1+), Unicode (used by Windows 95 and Windows NT), and EBCDIC (Extended Binary-Coded Decimal Interchange Code, used by IBM for mainframe computers).
COM	**C**omponent **O**bject **M**odel 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).
CPU	**C**entral **P**rocessing **U**nit
DAC	**D**igital to **A**nalog **C**onverter

DCE	**D**istributed **C**omputing **E**nvironment 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.
DCOM	**D**istributed **C**omponent **O**bject **M**odel 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.
DCS	**D**istributed **C**ontrol **S**ystem
DDE	**D**ynamic **D**ata **E**xchange A relatively old mechanism for exchanging simple data among processes in MS-Windows.
DI	**D**igital **I**nput
DLL	**D**ynamic **L**ink **L**ibrary. 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	**D**igital **O**utput
EGU	**E**n**g**ineering **U**nits
EIA	**E**lectrical **I**ndustries **A**ssociation
GUI	**G**raphical **U**ser **I**nterface
HART	**H**ighway **A**ddressable **R**emote **T**ransducer. 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	**H**uman **M**achine **I**nterface. Also referred to as a GUI or MMI. This is a process that displays graphics and allows people to interface with the control system in graphic form. It may contain trends, alarm summaries, pictures, and animations.
I/O	**I**nput/**O**utput
IEEE	**I**nstitute for **E**lectrical and **E**lectronics **E**ngineers
ISO	**I**nternational **S**tandards **O**rganization
MES	**M**anagement **E**xecution **S**ystem. 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	**M**an **M**achine **I**nterface
MIC	**M**achine **I**dentification **C**ode. License code of **eXLerate** which uniquely identifies you computer.
ODBC	**O**pen **D**ata **B**ase **C**onnectivity. 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	**O**riginal **E**quipment **M**anufacturer
OLE	**O**bject **L**inking and **E**mbedding. 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	**O**pen **S**ystem **I**nterconnection. 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	**O**LE for **P**rocess **C**ontrol. 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. eXLerate 2010 is OPC DA 2.05 compliant, and Spirit IT B.V. is a member of the OPC Foundation.
P&ID	**P**iping and **I**nstrumentation **D**iagram
PC	**P**ersonal **C**omputer
PCB	**P**rinted **C**ircuit **B**oard
PLC	**P**rogrammable **L**ogic **C**ontroller. 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	**R**emote **P**rocedure **C**all 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	**R**emote **T**erminal **U**nit
SCADA	**S**upervisory **C**ontrol **a**nd **D**ata **A**cquisition
SQL	**S**tandard **Q**uery **L**anguage
SVC	**S**uper**v**isory **C**omputer

TCP/IP	**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol. 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	**T**ransistor-**T**ransistor **L**ogic
UART	**U**niversal **A**synchronous **R**eceiver & **T**ransmitter
URL	**U**niform **R**esource **L**ocator. The global address for documents and resources on the World Wide Web.
VBA	**V**isual **B**asic for **A**pplications. 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	**V**isual **B**asic for **E**xcel. Although VBA is a general name for the Visual Basic in Office, more specifically, in Excel, the term VBE is used as well.
VML	**V**ector **M**arkup **L**anguage. 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	E**x**cel **L**ink **L**ibrary. 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 Spirit IT containing petrochemical equations has an XLL format.
XML	**E**xtensible **M**arkup **L**anguage. 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:

Asynchronous	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.
ActiveX	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.
C/C++	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.
Client/server	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
Device driver	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.
Engineering units	Engineering units as used throughout this manual refers in general to the units of a tag, for example 'bar', or 'ºC', and not to a type of unit, as with 'metric' units, or 'imperial' units.
Ethernet	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.

Event	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.
Exception	Any condition, such as a hardware interrupt or software error-handler, that changes a program's flow of control.
Fieldbus	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, Profibus, Devicenet, InterBus, and CANopen.
Kernel	The core of **eXLerate** that handles basic functions, such as hardware and/or software interfaces, or resource allocation.
Peer-to-peer	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.
Polling	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.
Process visualization software	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 **eXLerate**.
Protocol	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.

Query	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.
Real-time	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 "real-time" does not directly relate to how fast the program responds, even though many people believe that real-time means real-fast.
Real-time database	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 eXLerate, the contents of the real-time database can be stored in the system registry to avoid startup and shutdown problems in an application.
Registry	A database that contains information required for the operation of Windows NT and Windows 95, plus applications installed under Windows NT and Windows 95. The Windows Registry takes the place of Windows 3.1+'s REG.DAT, WIN.INI, and SYSTEM.INI files, plus PROFILE.INI files installed by Windows 3.1 applications. The Registry also includes user information, such as user IDs, encrypted passwords, and permissions. Windows NT and Windows 95 include RegEdit.exe for editing the Registry. The Windows NT and Windows 95 Registries differ in structure, and thus are incompatible.
Resource	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.
Synchronous	A type of message passing where the sending task waits for a reply before continuing processing.

System registry	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. 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.
Tag	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.
Visual Basic	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.
Web Server	A computer that has server software installed on it and is used to deliver web pages to an intranet/Internet.

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.

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.

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

Chapter 2 - Getting Started

Introduction

In this chapter, the reader is guided through the installation process of the setup program of the CD-ROM of **eXLerate**. *It further explains about various versions of the* **eXLerate** *software package.*

In this chapter, the license usage is also explained. Please read this section carefully, as it is important to obtain you permanent license as soon as possible.

Hardware requirements

eXLerate runs best on a Personal Computer with at least **1 GHz** with a minimum of 1 GB or more of RAM installed, depending on the application. The hard-disk should have at least 200MB of free disk space for trending file storage space, log-files and reports. The **eXLerate** software itself requires about 50MB-80MB of disk-space.

A fact is that the more memory is available, the better the performance is of Excel and hence the performance of **eXLerate**. For larger or more complex applications, **2GB** or more is recommended.

Software requirements

eXLerate requires Microsoft Windows XP, Windows Vista, Windows 7, Windows Server 2003 and Windows Server 2008. Both **32- and 64-bit** editions are supported of these Operating Systems are supported. We strongly recommended either **Windows Server 2003** or **2008** when using **eXLerate** in a production environment. These server operating systems often contain better drivers and better support for diagnosing problems.

eXLerate requires either **Microsoft Excel 2007 or 2010** to be installed. **Microsoft Excel 2003** (SP3) is also supported, however with certain limitations (it cannot be used with the new .XLRX file-format). Only the **32 bit** editions of Microsoft Office are supported.

Always make sure that the **latest Service Pack and Updates** are installed for Windows and Office to ensure optimal stability and protection against viruses.

Microsoft Office Editions

There are various editions of Microsoft Office available, e.g. Home Edition, Professional Edition, etc... **eXLerate** works with any of these editions because they all contain Microsoft Excel. As a recommendation we recommend either **Microsoft Office Standard** or **Professional edition**. This will also give you certain support rights from Microsoft in case you have trouble installing Office.

We further recommend the **English** version of Microsoft Office, although other languages will also work. The documentation may refer to certain Excel features in the English version of Excel which may be called differently in other languages.

Note that **eXLerate** is a **Microsoft Windows** based program, it will not work on other platforms also running Microsoft Office such as Mac OS-X. Furthermore, **eXLerate** requires a **32 bit edition of Microsoft Office**, it will not run on a 64 bit edition of Excel.

Installation & setup of Microsoft Excel

Microsoft Excel is an important component of **eXLerate**. Before installing **eXLerate**, make sure that Microsoft Excel is installed.

To install Microsoft Excel, insert the CD or DVD in your drive and run the setup. The following screen-shots depict Microsoft Excel 2010, the setup for Microsoft Excel 2007 however is very similar.

Enter your Product Key

After launching the setup, enter your product key and press 'Continue'.

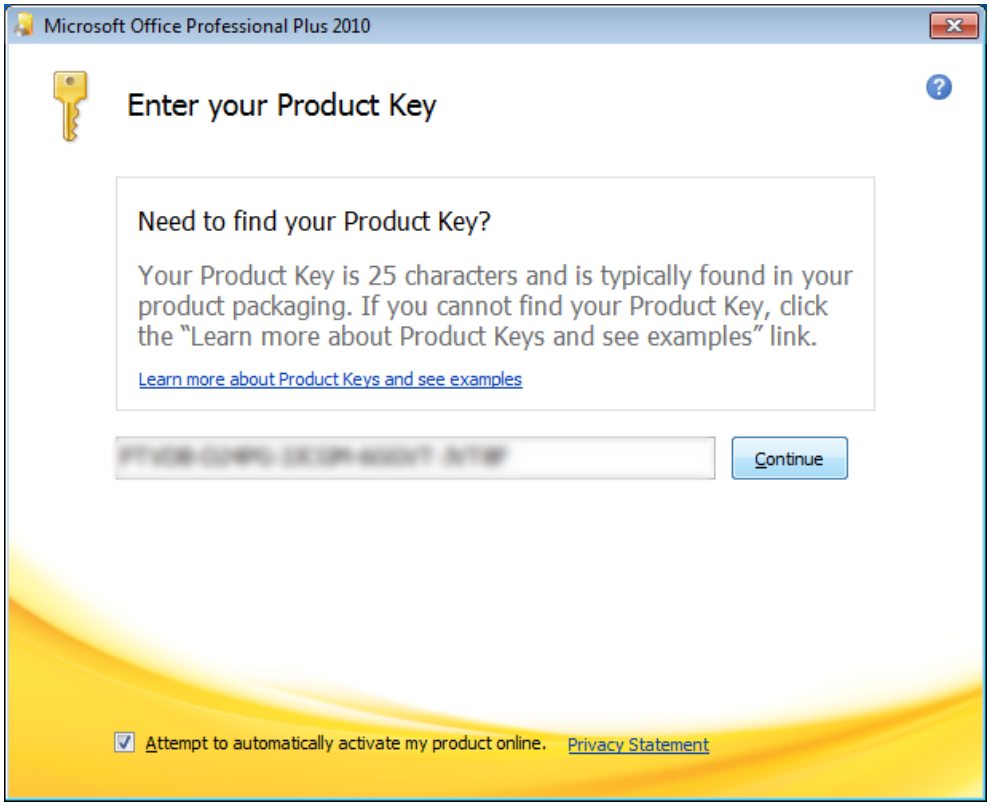


Figure 2-1: Office setup – Enter Product Key

Installation Type

After 'Accepting the terms of agreement', you may choose to either install Microsoft Office right away (*Install Now*), or *Customize* the installation. Both options will install all features required for running eXLerate. The *Customize* option will allow you to install only those features that are absolutely necessary for eXLerate in order to save disk space.

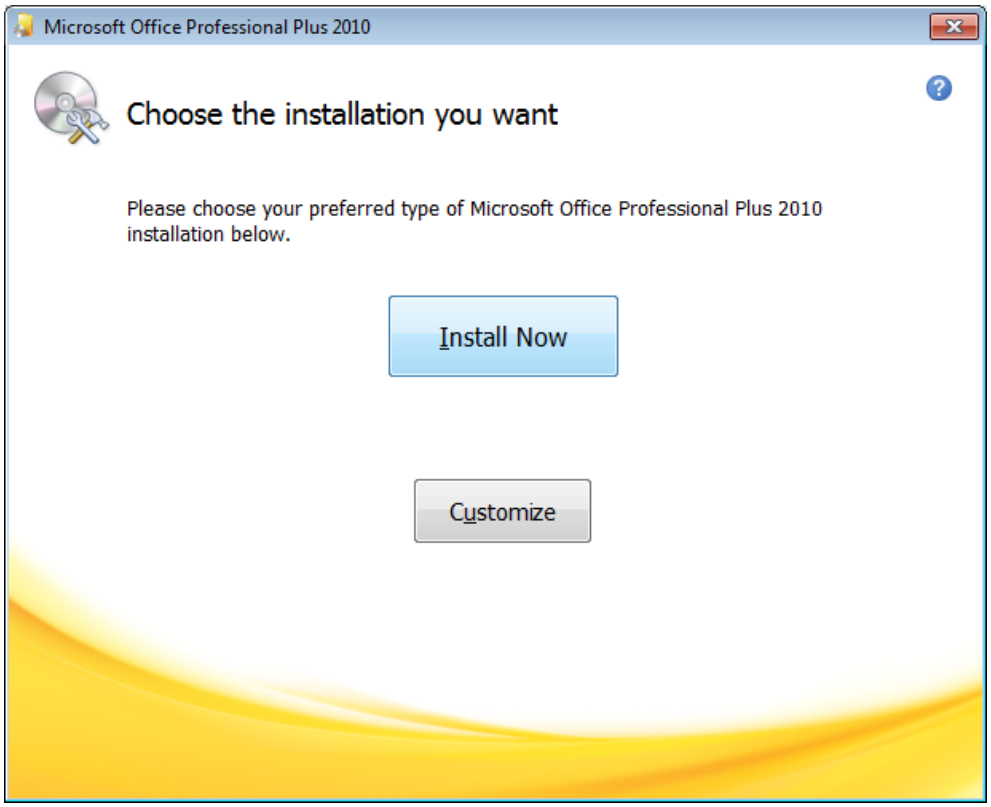


Figure 2-2: Office setup – Installation Type

Customizing the Installation

When choosing Customize make sure that at least "Microsoft Excel" and "Visual Basic for Applications" (VBA) is selected. Click *Install Now* to start the installation.

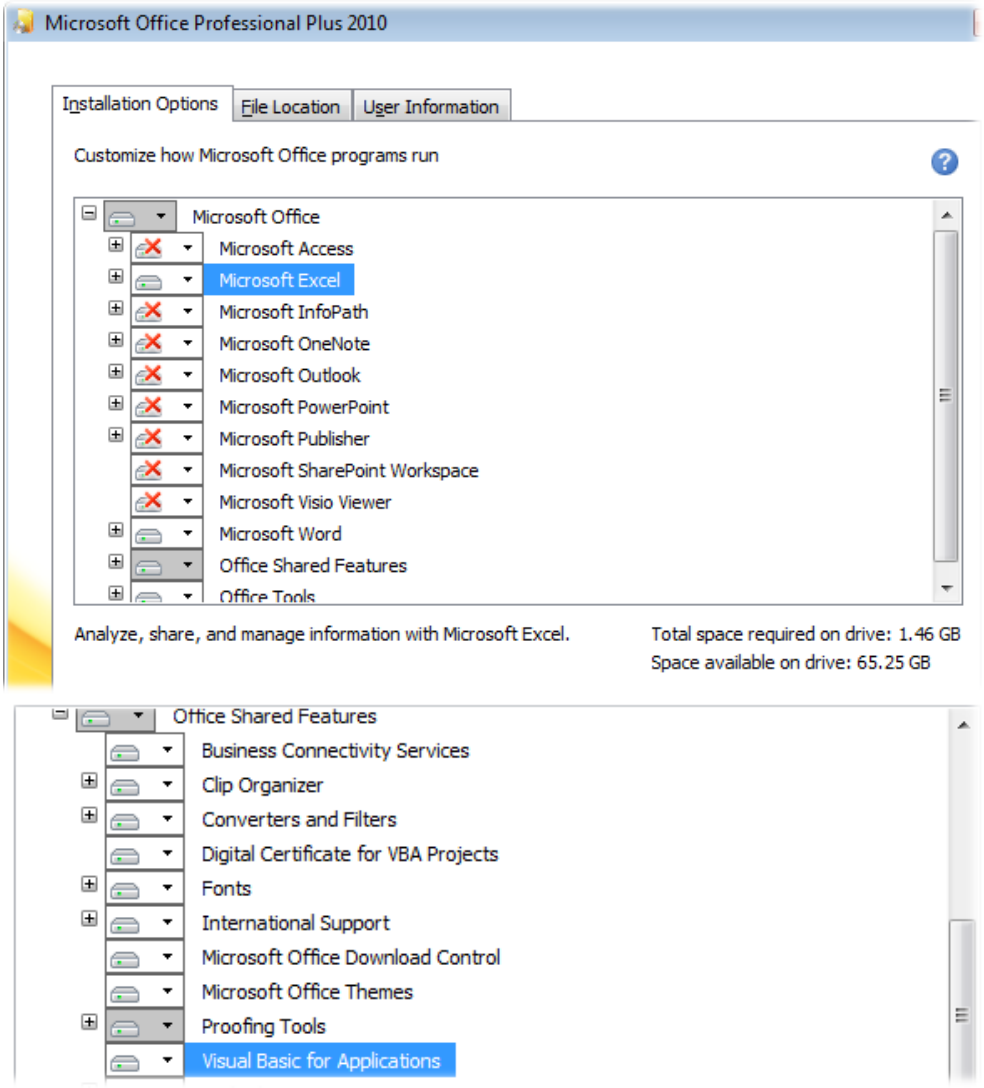


Figure 2-3: Office setup – Customize

Completing the Installation

After Microsoft Office is installed, it is ready to be used. Click *Close* to exit the installation program. At this point it is strongly advised to **install any Service Packs and Updates** that are available for Microsoft Office.

Figure 2-4: Office setup – Completing the installation

eXLerate editions and options

In traditional process visualization software packages, various options are usually based on limitations on both its functionality as well as on the number of tags that can be created within the package. These limitations are mostly there for commercial aspects of the software: the bigger the number of tags, or the more functionality, the higher the price of the software will be.

eXLerate comes in various flavors as well, each with its specific qualifications with respect to tag database size and functionality. However, this size or functionality is not as far restricted as with other packages, since in Microsoft Excel, theoretically each worksheet cell may be thought of as a 'tag', since each cell may contain a number or equation, can be presented to the operator, and may be formatted.

Restrictions only apply on number of values obtained from external devices, called: real-time data, and further on built-in support with respect to automatically generated calculations for reporting purposes. **eXLerate** does not limit the number or size of the worksheets in your project.

Effectively, due to the open character of Excel, your application may therefore contain as many tags as you want. The following versions of **eXLerate** are available:

Editions	I/O Tags
eXLerate 2010 Lite Edition license	**75**
eXLerate 2010 Basic Edition license	**150**
eXLerate 2010 Standard Edition license	**300**
eXLerate 2010 Extended Edition license	**750**
eXLerate 2010 Pro Edition license	**1500**
eXLerate 2010 Pro Edition license	**3000**
eXLerate 2010 Pro Edition license	**6000**
eXLerate 2010 Pro Edition license	**32K**
eXLerate 2010 Full I/O Edition license	**Unlimited ***

Table 2-1: Available editions of eXLerate with respect to I/O tags

Besides these I/O tag size related options, various other options are available, such as math libraries for specific technologies and industries. Please take a look at http://www.spiritIT.com for an update on the available products.

Installing the software on your computer

Assumed pre-installation

Make sure you have installed Microsoft Excel 2003, Microsoft Excel 2007, or Microsoft Excel 2010 prior to installing **eXLerate**. See the previous section '*Software requirements*' for more details.

In order to install the software onto your hard disk, insert the disk in your CD-ROM drive, and run the **eXLerate** setup program. The following display page appears:

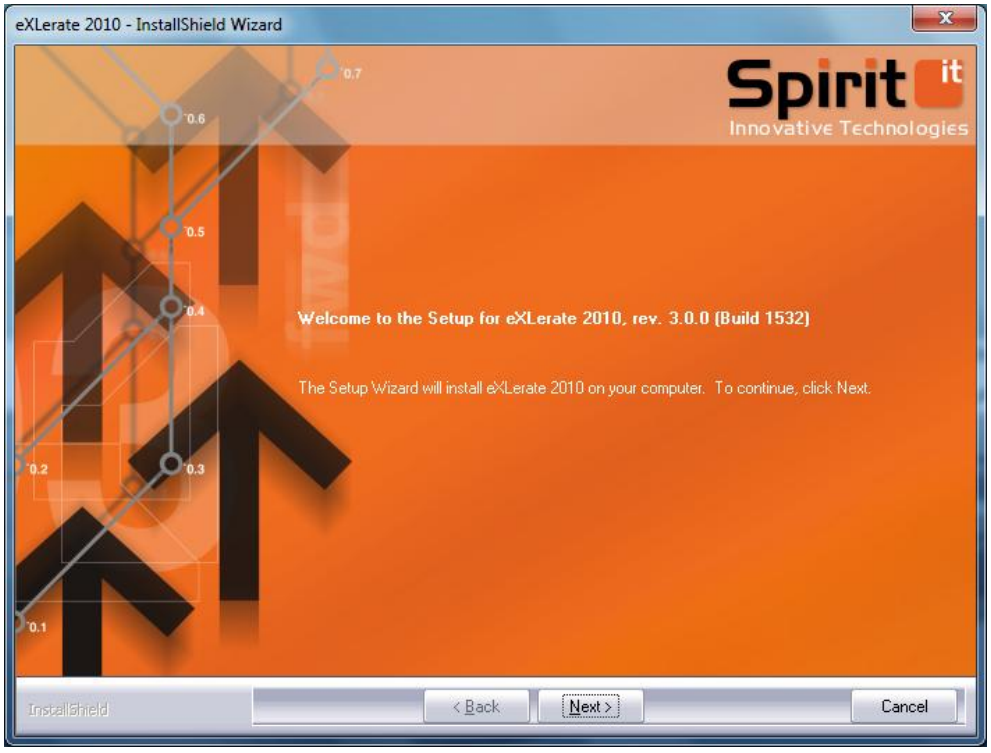


Figure 2-5: eXLerate setup program

This page tells the user which version will be installed, or in case or upgrading which version will be upgraded. The "**Next**"-button may be pressed, after which the End User License Agreement dialog is displayed.

End user license agreement

The end-user license agreement has the following contents:

SOFTWARE PRODUCT LICENSE
After installation an evaluation version of this product will run for the period of 15 days. Please contact Spirit IT for a valid license authorization key. A valid license authorization will not expire at all.

This end user license agreement grants you the following rights:

Application Software
You may install and use one copy of the SOFTWARE PRODUCT, or any prior version for the same operating system, on a single computer. The primary user of the computer on which the SOFTWARE PRODUCT is installed may make a second copy for his or her exclusive use on a second computer such as a portable computer.

Storage/Network Use
You may also store or install a copy of the SOFTWARE PRODUCT on a storage device, such as a network server, used only to install or run the SOFTWARE PRODUCT on your other computers over an internal network; however, you must acquire and dedicate a license for each separate computer on which the SOFTWARE PRODUCT is installed or run from the storage device. A license for the SOFTWARE PRODUCT may not be shared or used concurrently on different computers.

OTHER LIMITATIONS
Separation of components. The SOFTWARE PRODUCT is licensed as a single product. Its component parts may not be separated for use on more than one computer.

Rental. You may not rent, lease, or lend the SOFTWARE PRODUCT.

Software Transfer
You may permanently transfer all of your rights under this EULA, provided you retain no copies, you transfer all of the SOFTWARE PRODUCT (including all component parts, the media and printed materials, any upgrades, this EULA, and, if applicable, the Certificate of Authenticity), and the recipient agrees to the terms of this EULA. If the SOFTWARE PRODUCT is an upgrade, any transfer must include all prior versions of the SOFTWARE PRODUCT.

Termination
Without prejudice to any other rights, Spirit IT may terminate this EULA if you fail to comply with the terms and conditions of this EULA. In such event, you must destroy all copies of the SOFTWARE PRODUCT and all of its component parts.

COPYRIGHT
All title and copyrights in and to the SOFTWARE PRODUCT (including but not limited to any images, photographs, animations, video, audio, music, text, and "applets" incorporated into the SOFTWARE PRODUCT), the accompanying printed materials, and any copies of the SOFTWARE PRODUCT are owned by Spirit IT. The SOFTWARE PRODUCT is protected by copyright laws and international treaty provisions. Therefore, you must treat the SOFTWARE PRODUCT like any other copyrighted material except that you may install the SOFTWARE PRODUCT on a single computer provided you keep the original solely for backup or archival purposes. You may not copy the printed materials accompanying the SOFTWARE PRODUCT.

LIMITED WARRANTY
Spirit IT warrants that (a) the SOFTWARE PRODUCT will perform substantially in accordance with the accompanying written materials for a period of ninety (90) days from the date of receipt, and (b) any Support Services provided by Spirit shall be substantially as described in applicable written materials provided to you by Spirit, and Spirits support engineers will make commercially reasonable efforts to solve any problem issues. Some states and jurisdictions do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you. To the extent allowed by applicable law, implied warranties on the SOFTWARE PRODUCT, if any, are limited to ninety (90) days.

NO OTHER WARRANTIES
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, SPIRIT IT DISCLAIMS ALL OTHER WARRANTIES AND CONDITIONS, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT, WITH REGARD TO THE SOFTWARE PRODUCT, AND THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES. THIS LIMITED WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS. YOU MAY HAVE OTHERS, WHICH VARY FROM STATE/JURISDICTION TO STATE/JURISDICTION.

LIMITATION OF LIABILITY
TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT SHALL SPIRIT IT OR ITS SUPPLIERS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, OR ANY OTHER PECUNIARY LOSS) ARISING OUT OF THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE PROVISION OF OR FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF SPIRIT IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, SPIRIT IT' ENTIRE LIABILITY UNDER ANY PROVISION OF THIS EULA SHALL BE LIMITED TO THE GREATER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR U.S.$5.00; PROVIDED, HOWEVER, IF YOU HAVE ENTERED INTO AN EXLERATE SUPPORT SERVICES AGREEMENT, SPIRIT IT' ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT. BECAUSE SOME STATES AND JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF LIABILITY, THE ABOVE LIMITATION MAY NOT APPLY TO YOU.”

When the user has accepted the end-user license agreement, the customer information must be entered. Next, dialogs are displayed in which the user is opted for various choices:

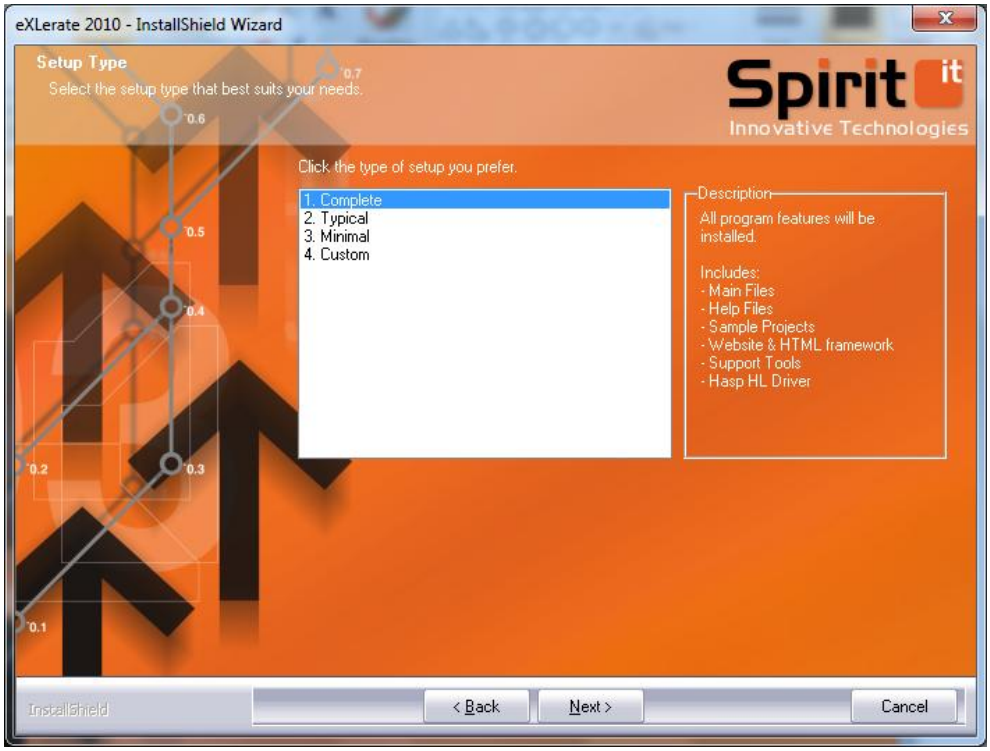


Figure 2-6:Choosing the Setup type

- Complete installs all files.
- Typical installs all files except all the samples.
- Minimal installs only the files that are absolutely necessary for running the product (no samples, no help, no HTML framework and no tools).
- Custom install which allows to user to install various components optionally and into customizable file locations. For example, a user may want to only install the main program files, without any sample applications, in which case the appropriate check mark box may be turned off, as in the example below:

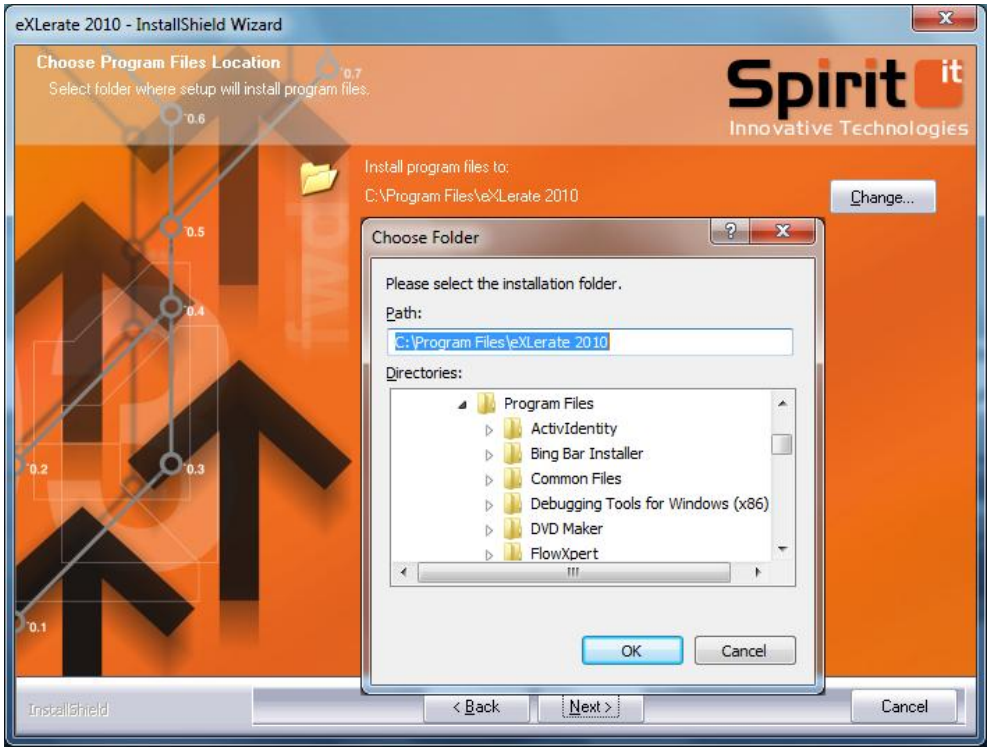


Figure 2-7: Choosing a program files installation location

- The location of the Program Files in which the setup program will install the software. This may be at an existing program folder, or setup will create a special folder for you.

Figure 2-8: Choosing a project files installation location

- The location of the sample projects, resource files and HTML framework. By default this location is the folder C:\XLRX, but this can be changed into any other folder.

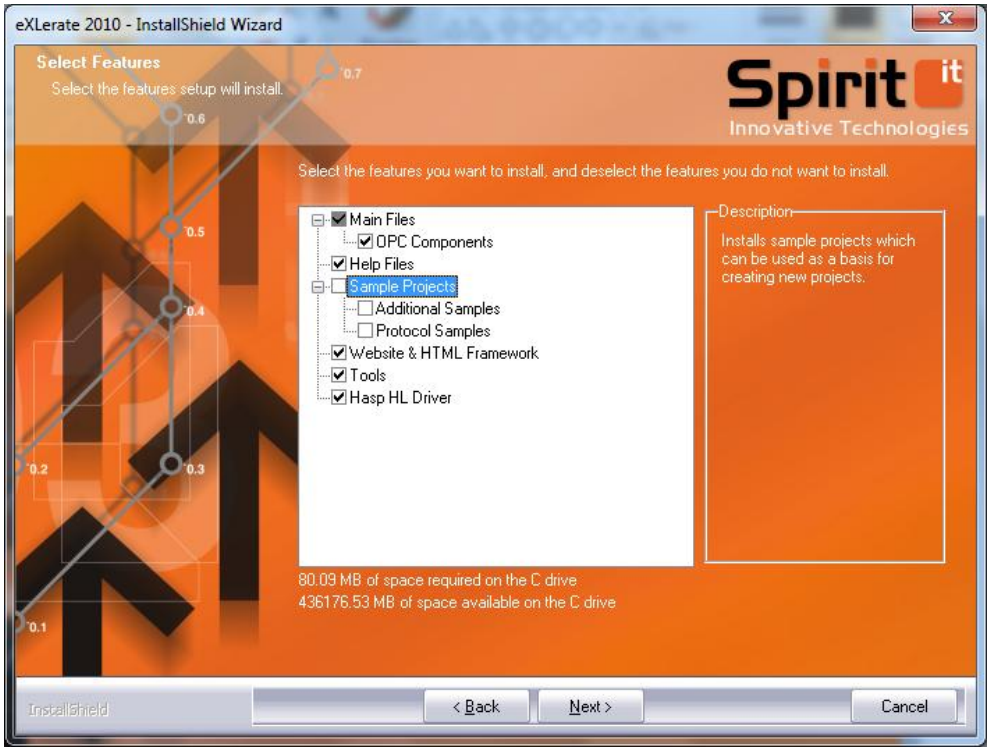


Figure 2-9: Individual feature installation

- The features that are selected for install. When selecting a feature, a detailed description is displayed in the description box. For example, in the figure above, the Protocol Samples feature is selected and the description shows which files are part of that feature.

Before the software actually starts copying the software, the user is prompted to review the installation settings. Clicking Install will start copying the files and the following progress bar is presented:

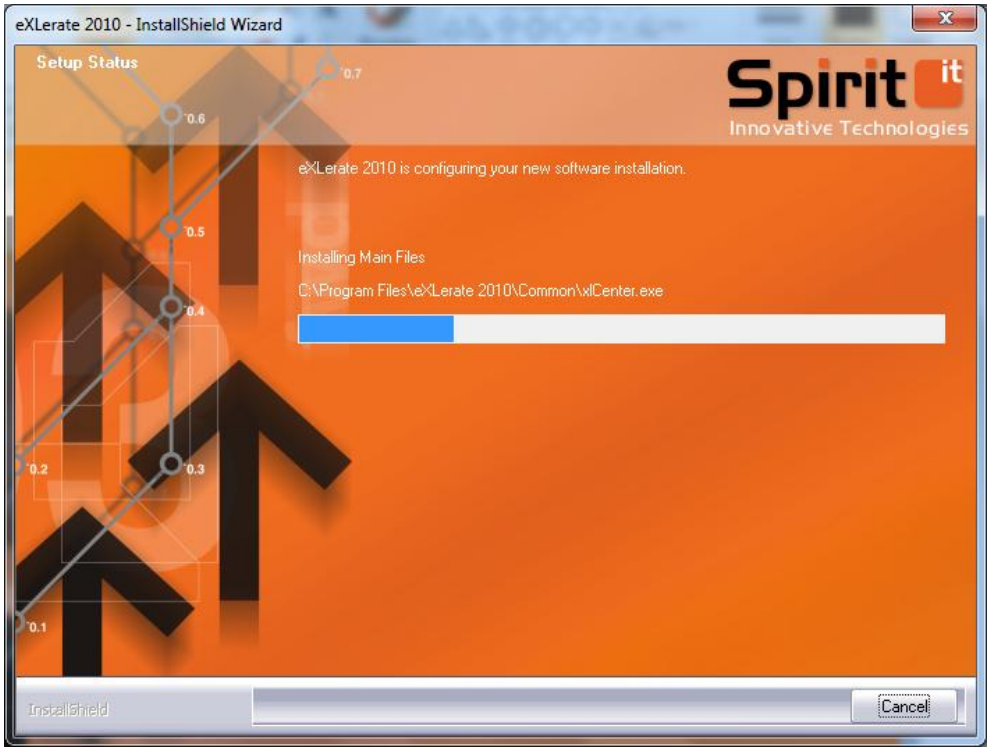


Figure 2-10: Copying files at installation

When the copying process has been completed, the user is presented with a dialog containing license information.

License number and authorization key

eXLerate uses a software based license system, preventing the use of awkward hardware dongles at the back of the computer. The software license is based on the data that you have entered, as well as on the machine that the software is installed on.

If the machine on which the software is installed is replaced with another machine, you may request for a new authorization code, which will be generated and sent to you by Spirit.

A license authorization code is sent to you via e-mail or fax, and may be entered in the system using the "License Manager". The "License Manager" is located in the "eXLerate 2010" menu of the Windows Start menu.

The following information must be entered at the license dialog during the setup process:

End-User	Your full name, or the name of your department using the eXLerate software, for example: "John Smith", or "Engineering".
Company	The name of your organization to which the license will be granted, for example "MyCompany, Inc."
System	The name of your system (by default your computer-name is used).

The following dialog is presented in the client license dialog:

Figure 2-11: Entering license information

When you have installed the software for the first time, a temporary license is automatically installed on your machine, with which you may run most portions of the software for a period of 15 days.

In this period, you should send your name, organization, license number and your MIC (Machine Identification Code) to Spirit at the back of this manual, after which you will receive your **permanent** authorization code.

If you have previously installed the software, the parameters of the previous installation are displayed, which may be corrected if required.

If you already have an installed permanent license, this license remains active. During product upgrades, no additional licenses have to be installed at all.

When the setup process has been completed, the following dialog may or may not appear on your display. This depends on the files that were already installed on your machine.

When the following dialog appears, the computer must be restarted prior to using the software:

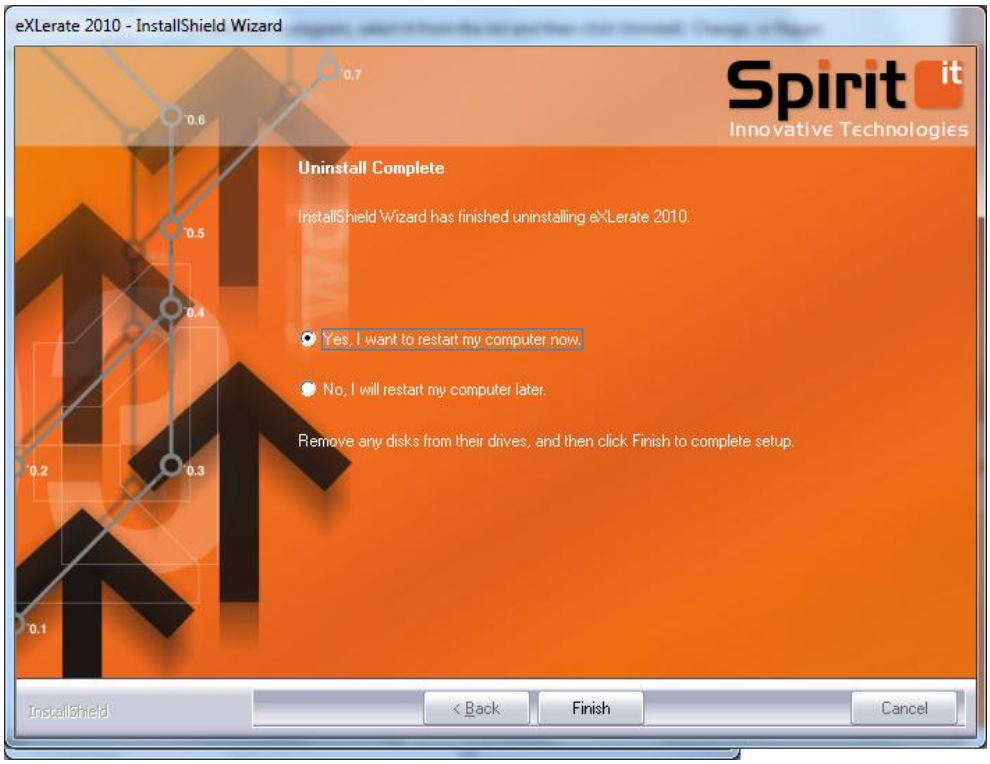


Figure 2-12: Setup has been completed

Please restart the computer if this message is displayed, after which the installation process is completed.

Project Files

By default the project files are installed in the 'C:\XLRX' location. This directory contains the eXLerate project files (.XLR;.XLRX), samples, and additional support files. After the first installation some sub-directories are automatically created, others are created when running eXLerate. The following sub-directories are created:

Directory	Description
<C:\XLRX>	Location for the .XLRX project files
<C:\XLRX>\Archive	Location for automatically generated project backup files
<C:\XLRX>\Cache	Location for temporary cache files
<C:\XLRX>\Database	Location for the database (e.g. event-log)
<C:\XLRX>\HTML	Location for the website files
<C:\XLRX>\Logger	Location for logs
<C:\XLRX>\Reports	Location for generated reports
<C:\XLRX>\Resources	Location for additional graphical resources (e.g. .jpg, .bmp)
<C:\XLRX>\Templates	Location for project component templates (e.g. TrendChart defaults).
<C:\XLRX>\TrendData	Location for the trending files

Table 2-2: Additionally created directories

After the installation new projects can be added to eXLerate through the Control Center application. The default file locations of each project (e.g. reports, trend-files) point to the directories above. These may be changed to any location (e.g. "D:\Reports",

Using the License Manager

In order to use the product, either a software license must be installed or a hardware-key (dongle) must be attached to the system.

The License Manager can be used to view the currently installed license(s) or install software based licenses. The program can be found in the Start menu of Windows, named "License Manager".

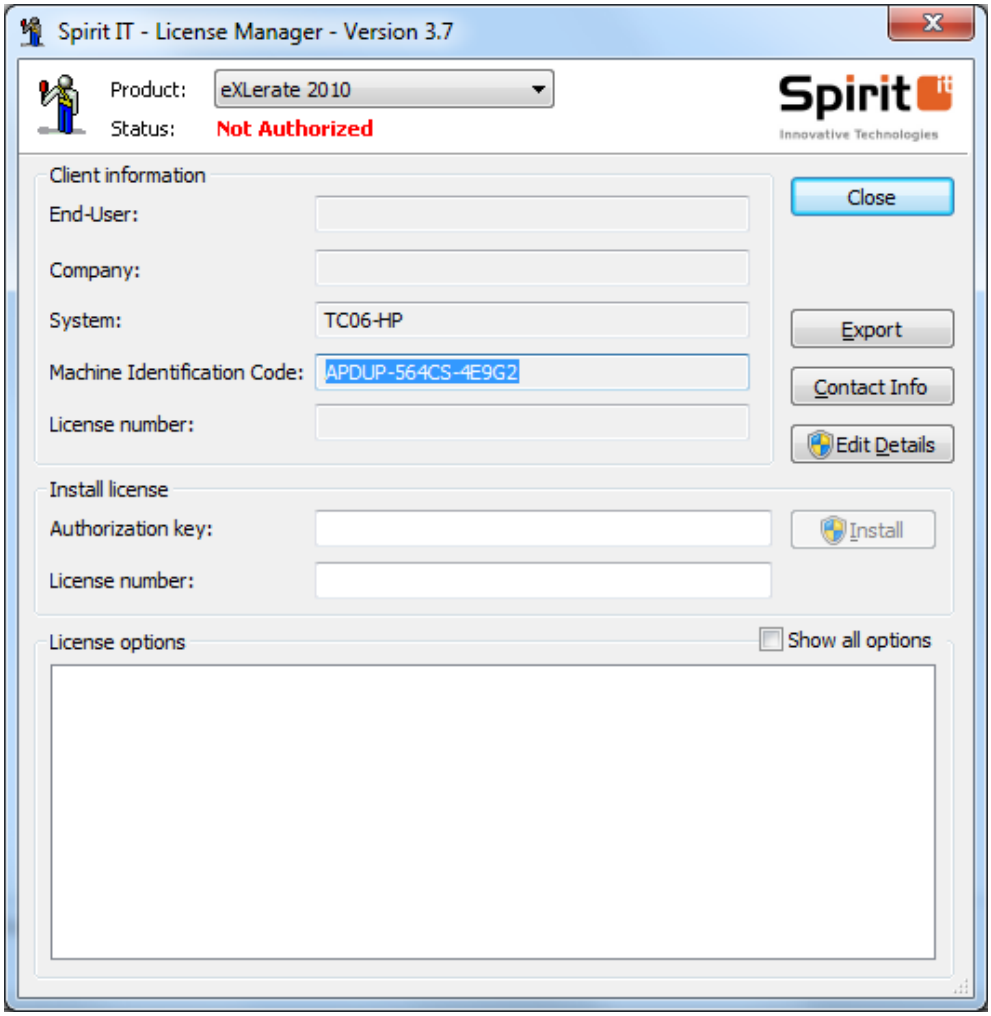


Figure 2-13: License Manager

The "Status" field shows the current status of the selected product. In the example above, an evaluation license is installed which is about to expire in 1 day. The user is urged to obtain a permanent license from SpiritIT in order to continue to use the software.

Requesting a software based license

A software based license can be requested directly from Spirit IT, e.g. by sending an email to license@spiritIT.com.

The following information should be included in the email so that the request can be handled quickly and efficiently.

- End-User
- Company
- System
- Machine Identification Code (MIC)
- License Number
- License options (tag-count, Flow-Xpert support, etc…

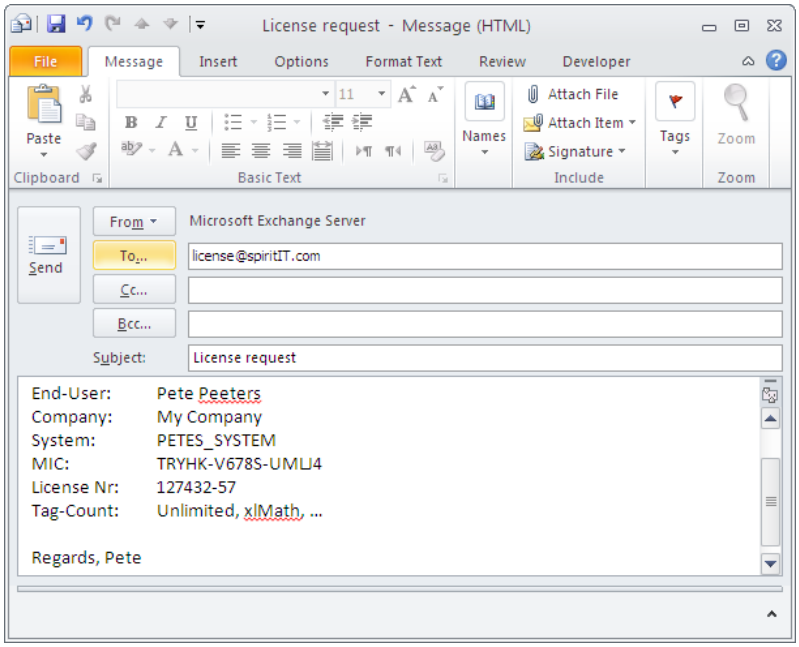


Figure 2-14: Requesting a software based license

If you don't know which options you need or have any other questions, don't hesitate to contact our sales or support team.

Installing a software based license

After purchasing a license you should have received an "Authorisation Key" and "License Number".

Both the Authorization-key and License Number can be copied (using Ctrl+C/V) to the relevant fields of the License Manager. After this, the button "Install" should become enabled.

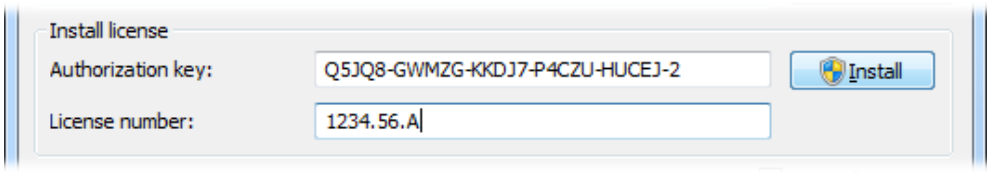


Figure 2-15: Installing a license

After pressing "Install" a message will appear that the license was installed successfully.

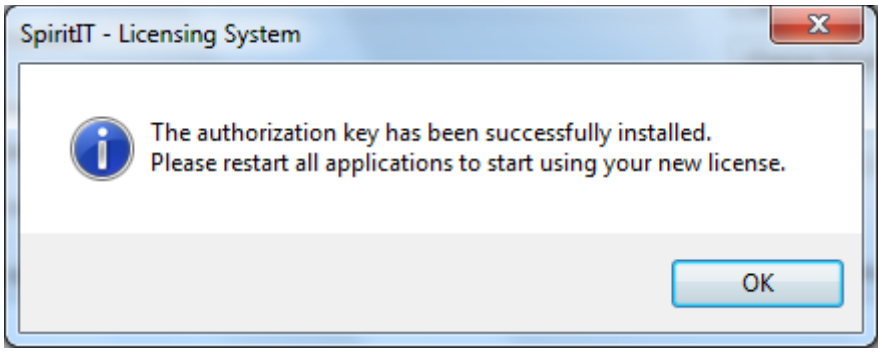


Figure 2-16: License successfully installed

If installing the license doesn't work, please verify that:

- No Hardware-Key is attached (there is no green icon at the top)
- Both the Authorization-Key and License Number have been correctly entered.
- The date/time of your system is correct. Authorization-Keys can only be installed up to 15 days after they have been issued.
- You are installing the license onto the correct system.

If you are still unable to install the license, please contact Spirit IT.

Hardware Keys (Dongles)

Alternative to software based licenses; hardware-keys can be used as well. A hardware-key is a small device which connects to the USB port of your computer.

Hardware-keys are particularly useful for commissioning- and service personal that can take the hardware-keys with them and be always sure that they have the correct license rights for authoring a system.

When a hardware-key is attached, the License Manager displays a green hardware-key icon at the top of the program.

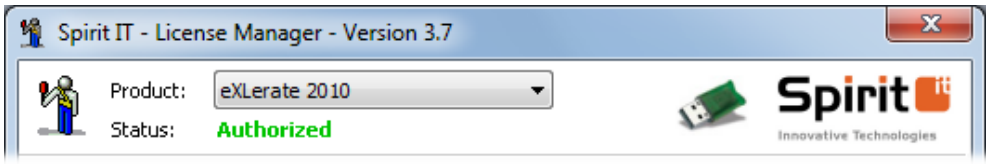


Figure 2-17: Hardware-key attached

If no green hardware-key icon is displayed, then make sure your hardware-key is properly attached and that the Hasp HL Driver is installed. Please read the next section on how to manually install the hardware-key driver.

A hardware-key can contain licenses for multiple products. Whenever the hardware-key is attached, the key overrules the software based licenses.

Hardware-keys can be purchased directly from Spirit IT, please contact license@spiritIT.com on how to obtain a hardware-key and licenses.

Installing the Hardware Key Driver (HASP HL)

Whenever a hardware-key is attached to an USB port for the first time, Windows will try to install a driver for it. Only when a correct driver is installed will the hardware-key function properly.

In some cases Windows will not be able to install a driver automatically (e.g. not connected to the internet) and the driver has be installed manually. Spirit IT distributes this driver on your product CD under the name "Hasp HL Driver".

eXLerate also install the driver setup-package onto your system and it can be accessed from the Windows Start Menu; "eXLerate 2010\Hasp HL Driver".

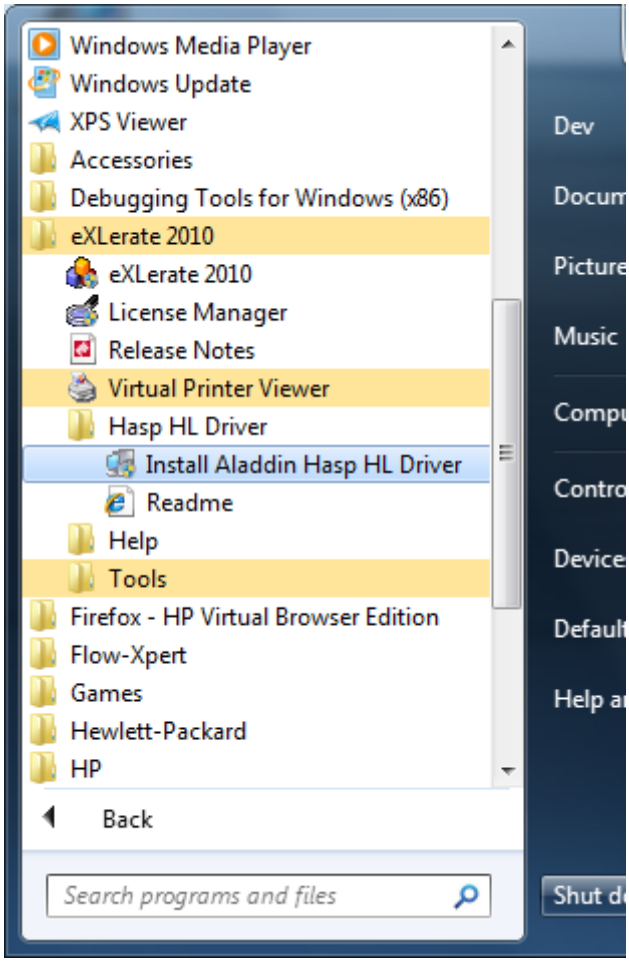


Figure 2-18: Hasp HL Driver

After the Setup package has been launched, the following window is displayed.

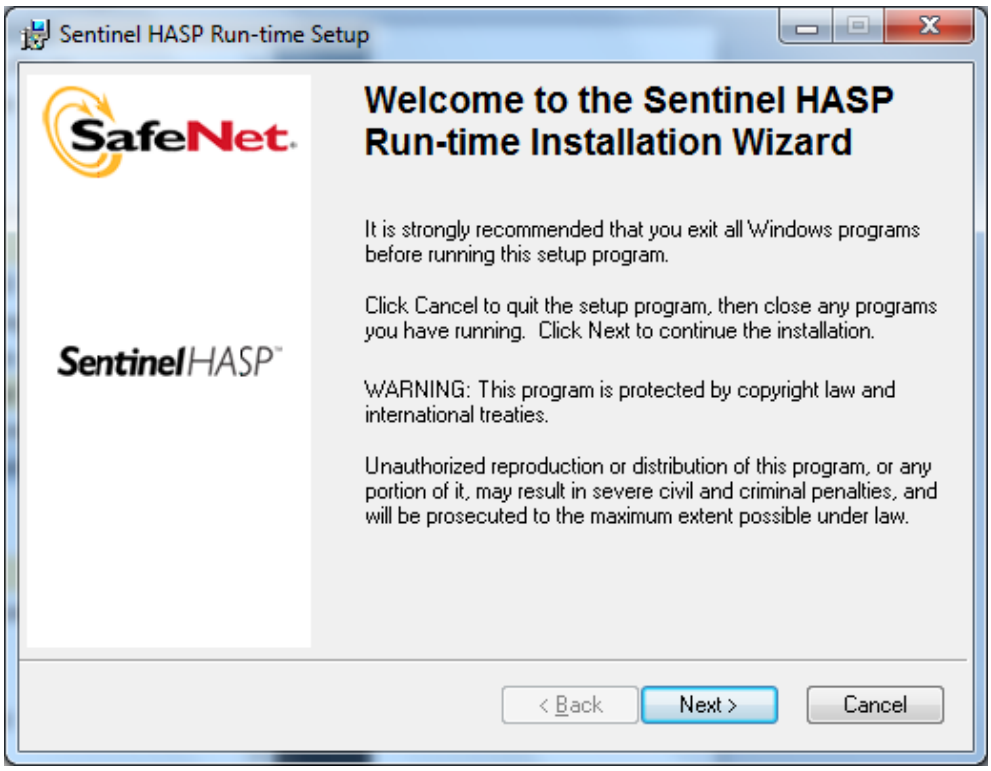


Figure 2-19: Installing Hasp HL Driver

Follow the instructions of the program to install the driver.

After successful installation of the driver, the License Manager should display the green hardware-key icon at the top of the program.

If the hardware-key is still not functioning, try downloading the latest driver from http://www3.safenet-inc.com/support/hasp/enduser.aspx.

This page is intentionally left blank.

Chapter 3 - Tutorial

In this chapter the reader is introduced to general concepts of **eXLerate**. *At first, a short introduction will be given on the advantages of using* **eXLerate**.

Step by step, the user will learn how to start a ready-to-run project, which is installed on the computer by the setup program.

This chapter is added for a novice user, who wants to learn about using **eXLerate**.

If you are already familiar with the concepts of **eXLerate**, *you might want to only briefly look at this chapter, and move forward to one of the next chapters.*

Introduction

eXLerate is a full-featured real-time HMI software package, based on Microsoft Excel. Many functions, features and wizards have been added to:

- Make sure that you have a full-featured real-time HMI software package that is able to offer you more than competitive HMI functionality.
- Allow you to quickly create your projects using structured application engineering techniques. Save tremendously on human resources, because of the powerful and well-known environment.
- Automatically generate built-in calculations, generate reports and XML / VML based HTML pages.

In this chapter an overview of the product will be given, to familiarize with the possibilities of **eXLerate**.

In the following chapters, more detailed information is presented. Using this manual, you are able to develop real-time HMI applications on your own, and all from within Excel.

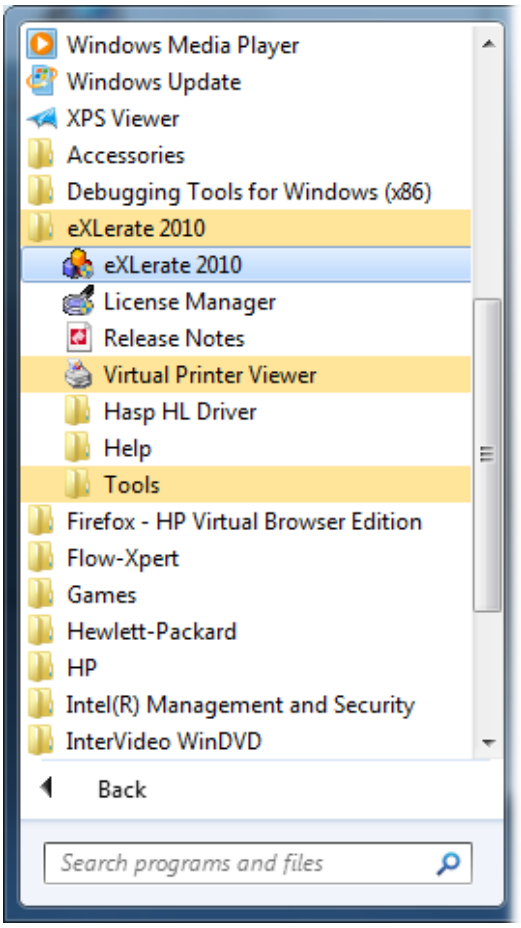
The eXLerate 2010 Control Center

The **eXLerate** 2010 environment is controlled from a special program, which is called the "Control Center". This is a Windows executable program in which all programs that are made available for the operator are defined, including **eXLerate** applications. Any other type of application may be installed in the Control Center as well, for example a word processor, or print utility program.

The Control Center is used for the following tasks:

- Control, start, monitor, or terminate the installed applications, including **eXLerate** projects.
- Print logged event messages to the printer.
- Maintain a list with users and passwords, and allow a user to login.
- Act as a program shell, in case the regular Windows Program Manager is not accessible for the operator in a production environment.
- Maintain common **eXLerate** properties, such as auto-logoff time, alarm idle time, and various other properties.

From the Windows Program Manager, start the Control Center by activating the "**eXLerate** 2010" icon as in the figure on the right of this page.



The following display is presented after a short welcome splash window:

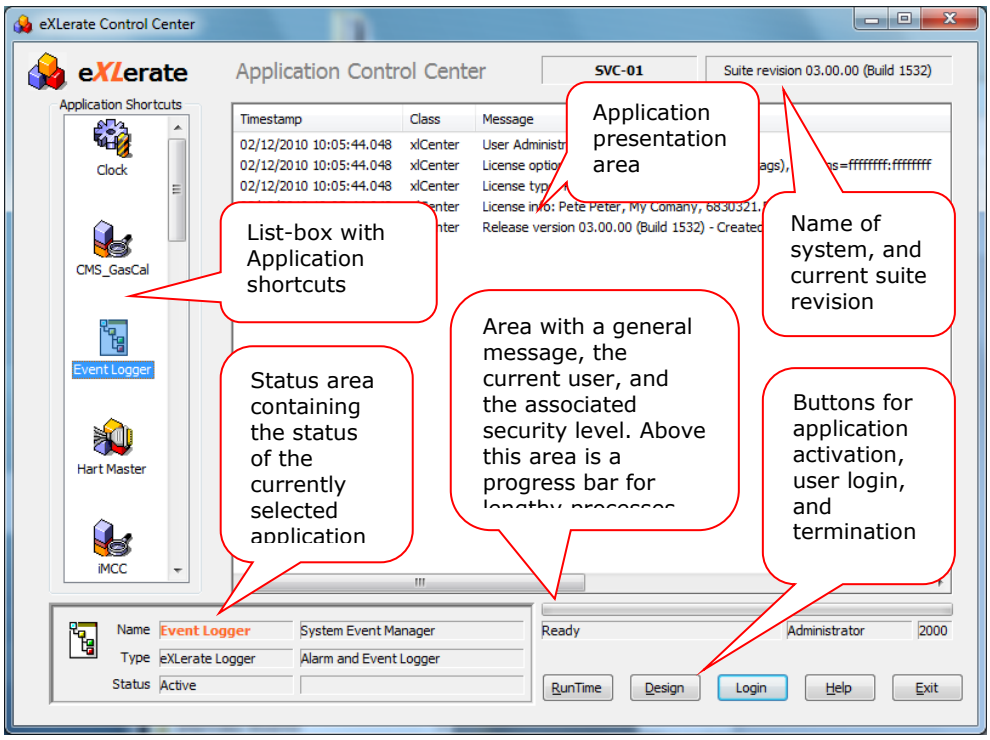


Figure 3-1: the eXLerate Control Center dialog with different areas

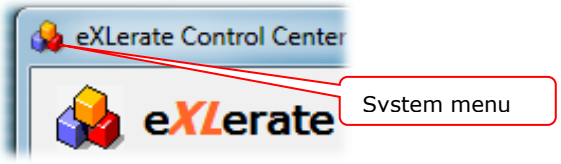


Figure 3-2: System menu at the left-top part

The display page is divided into several areas:

- At the left-hand side of the display, a scrollable list box is available containing all currently defined application shortcuts.
- At the left-hand top, there is a system menu containing a menu for user maintenance, and the control center property dialog.
- In the middle of the display, a large window is available holding the presentation area of the currently selected application. In case the event monitor is selected, this area contains a scrollable list-box holding the last logged event messages. In case of another application, a presentation bitmap may be showed.

- At the right top corner, the current name of the computer is presented ("DEMO" in this case), and the current revision of the **eXLerate** suite.
- At the left-hand bottom of the display, there is a status area containing the current monitored status of the selected application.
- At the right-hand bottom of the display, there is a button area with buttons: '**RunTime**', '**Design**', '**Login**', '**Help**', and '**Exit**'.
- Just above the button area, there are three status sections; one section containing the last system message, the currently logged-in user, and the security level of the current user. Above the status sections is a progress bar to show the progress on lengthy operations.

In order to start any of the application shortcuts in the application shortcut bar at the left of the display page, you must be first logged in. At application installation with the **eXLerate** setup.exe program, various factory-default user accounts are created:

User	Password	Level	Description
guest	guest	10	Low level guest account. Typically used to only browse through display pages, not to change vital process data.
operator	operator	500	Higher level operator level. Typically used to do all of the above, and additionally print out generated reports etc.
engineer	engineer	1000	Higher level engineer account, typically used do all of the above plus alter process data, such as alarm limits and other process parameters.
administrator	admin	2000	Highest level, with which a user is allowed to add users to the system.

Table 3-1: Setup' generated user accounts

You are advised to alter these passwords for your own projects.

Although the usage of security levels is freely programmable, the above suggested security levels would be more than sufficient for most types of applications.

These levels are also implemented in one of the installable off-the-shelf project templates.

In order to login to **eXLerate** and set your security level, click on the '**Login**'-button on the Control Center button bar at the bottom of the page, after which the following Login dialog is presented:

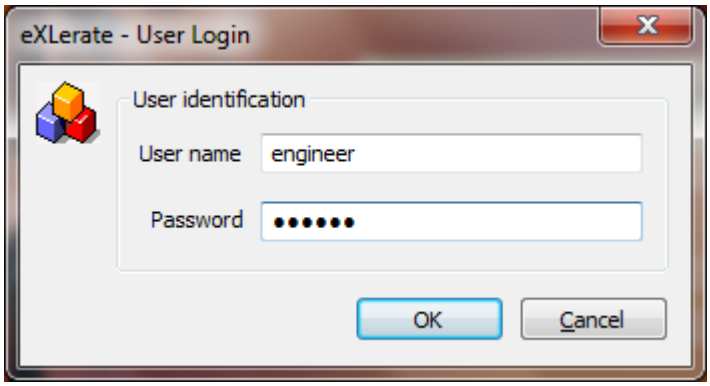


Figure 3-3: Login dialog in the Control Center

Enter the User name "engineer", and password "engineer", and accept the input by pressing the '**OK**'-button, or the <Enter>-key.

When the event monitor is selected as active application shortcut, a message like:

"`dd/mm/yyyy hh:mm:ss [xlCenter] – User 'engineer' has successfully logged in`"

is logged in the system event logger.

Now various buttons are enabled, for example the '**Design**'- and '**Runtime**'-buttons, which are used to launch an application in design-mode, or runtime-mode.

Operating modes of eXLerate

eXLerate basically is aware of four operating modes:

- **Design mode**
  (Application engineering, no real-time updates active)
- **Runtime mode**
  (Control System Operation, with real-time updates active)

- **Preview mode**
  (Preview of Control System Operation without real-time updates active)

- **Verify mode**
  (Application engineering, real-time updates active)

When started from the Control Center, the user may select to start an application in Runtime mode, or in Design mode. These are the two basic modes.

In Runtime mode, an application is normally running for daily operation of a control system with data communications running, while is Design mode, an application engineer may modify an application.

Preview mode is like runtime mode, except that there is no data communication running. When in Preview mode, the user may press <Esc> to return to design mode. Preview may be used to test/browse the user interface navigation buttons for the application, preview a completed display page for layout etc.

Verify mode is selected from Design mode, when data communications is started. Verify mode allows for checking of animations and equations **with** data communications/updates running. This is a unique **eXLerate** mode.

Application launch

In order to start the an application, select it with your mouse in the application shortcut bar.

You may start an application shortcut in **RunTime**-mode, or in **Design**-mode.

When an application is started in RunTime-mode, the application is actually running, with all communication devices activated. Only display pages are presented to the operator, and display pages cannot be changed. When in RunTime-mode, the operator is allowed to navigate through the application using the function-key button bar at the bottom of the display.

When an application is started in Design-mode, it is available for application engineering.

Press the '**Design**'-button, after which the application is loaded, initialized, and started in Design-mode.

During the application startup, various startup messages are logged on the main logger window of the Control Center. An example of such a message is:
"`dd/mm/yyyy hh:mm:ss [eXLerate] - Initializing application…`", and
"`dd/mm/yyyy hh:mm:ss [eXLerate] - Initializing displays…`"

These messages may be used to closely monitor the entire application startup process, and is typically used during the application development process.

After a project has been started in **eXLerate** with the Control Center, the Control Center itself is minimized, and disappears as an icon into the system tray, where the user may restore the Control Center to full screen view by clicking once on the **eXLerate** icon.

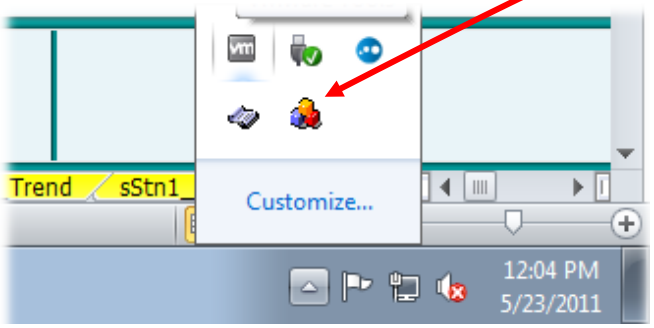


Figure 3-4: The system tray with the eXLerate icon

Once the application is started, the following screen appears:

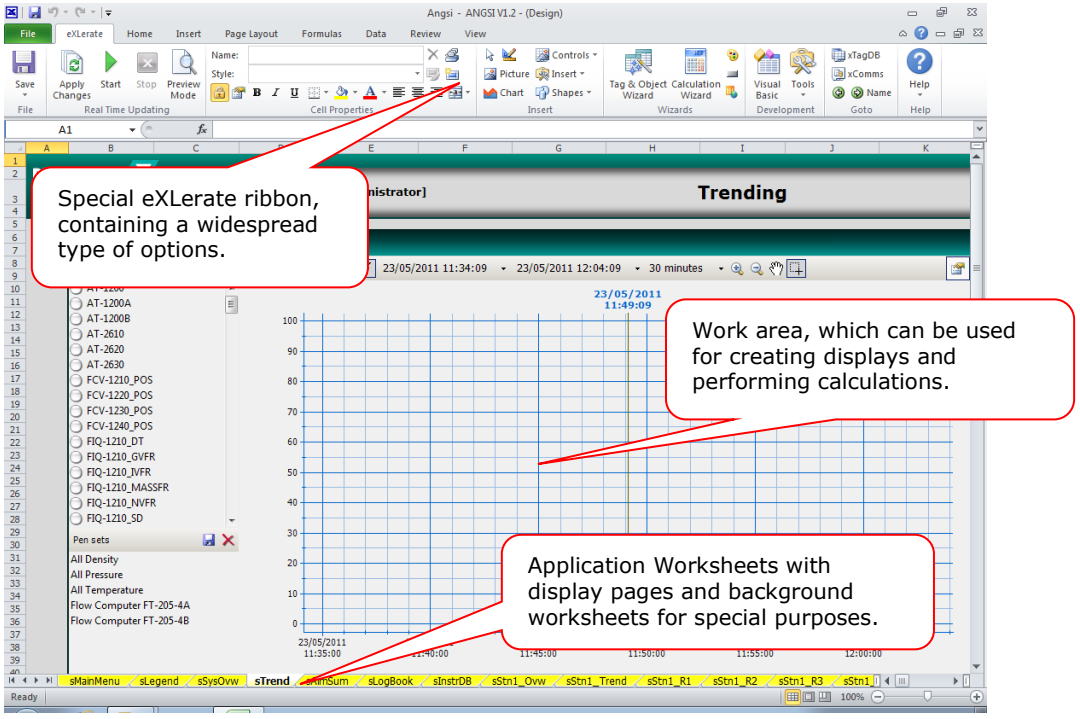


Figure 3-5: eXLerate main window

The eXLerate Ribbon

At the ribbon in Excel, a special 'eXLerate' item is available. The ribbon is only visible in Design-Mode, and is used for application development.

The **eXLerate** ribbon looks as follows:

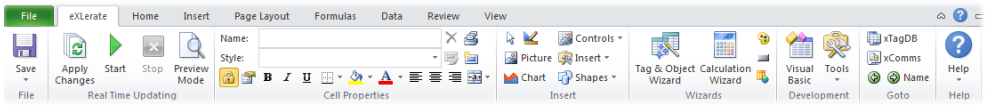


Figure 3-6: The eXLerate ribbon in Design-mode

In the **eXLerate** ribbon there are various sections, which will be described shortly in this section.

File section

The file-section contains the '**Save**' button, prominently visible for quick access. Opening and creating new applications can be done through the Control Center interface.

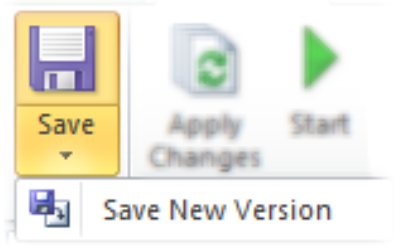
Save (Ctrl+S)

The Save button saves your changes to the application project file.

Whenever the application is saved, a backup is created of the old application file in the archive directory (e.g. "C:\XLRX\Archive).

Figure 3-7: Save-option

Save New Application



Below the Save button, a small arrow is visible. When pressed, the 'Save New Version' option appears:

Figure 3-8: Save New Version -option

When clicked the '**Save New Version**' dialog appears:

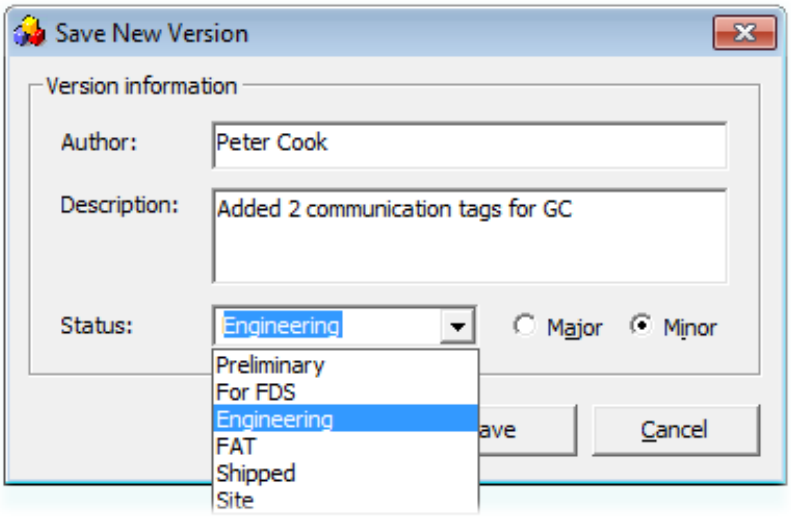


Figure 3-9: Save New Version dialog

The '**Save New Version**'-option performs the same actions as the Save button, and also creates or updates the 'xVersion' sheet.

The 'xVersion' sheet contains information about which version of eXLerate was used to save the application, the status of the application, the name of the application engineer, and the time of the last version update. If no 'xVersion' sheet is present, it will be created with default settings and layout.

If an 'xVersion' sheet is present, the **eXLerate** version will be checked when the application is started. If the version stored in the application does not match the running version of **eXLerate**, a warning is shown and the user is urged to save the application with updated version information. This is important so that problems with applications can be traced to specific versions of **eXLerate** and the engineer who made changes to the application. When the '**Save New Version**'-button is used, a form shows up which allows the engineer to change some values. The preset status values can be changed on the 'xVersion' sheet.

Real Time Updating section

The Real Time Updating section contains the most important options for applying configuration changes and starting/stopping communications.

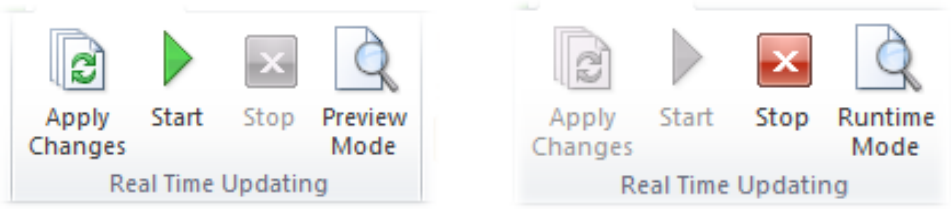


Figure 3-10: Real Time Updating section

Apply Worksheet Changes (Ctrl+K)

This option needs to be selected when the user has changed a table in a background worksheet that has an effect on **eXLerate**'s internal configuration, for example the 'xTagDB' tag database worksheet, the 'xComm' worksheet for real-time data communications, or the 'xAnimations' worksheet for shape animation settings. Further in the manual the exact conditions are given under which the Apply Worksheet Changes command should be activated.

Start (Ctrl+T)

Starts real-time data communications or simulations. This also starts any intervals configured in the 'rIntervals' table and animations. When started, the workbook is automatically calculated every second.

Stop (Ctrl+O)

Stops real-time data communications. Also see 'Start'. This option is enabled when communications are running.

Preview Mode/ Runtime Mode (Ctrl+N)

This option may be used by the user to switch from design-mode to preview-mode, or to Runtime mode. Preview mode is used to preview display pages, when no real-time updates are currently running. When updates are actually active, this option is called: '**Runtime Mode**', which is the normal operation mode. When real-time updates are not started however, switching to runtime mode is used to preview the created display pages. This is a helpful tool during display development. Changing from Runtime/Preview mode back to Design mode may be done using the **eXLerate** Control Center, by clicking on 'Design'-mode, or by pressing the <Esc>-key while displaying pages.

Cell Properties section

The Cell Properties section contains a variety of often used Excel commands as well as the ability to edit names and styles.

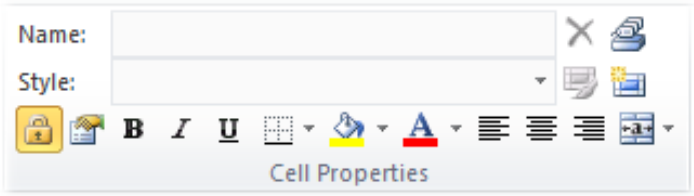


Figure 3-11: Cell Properties section

Some of the options in this section require additional attention.

Lock Cell


The concept of locking and unlocking cells in **eXLerate** is an important one. In general the following rule applies: Whenever a cell is locked, it cannot be modified when Real Time Updating is started. This is important when designing a User Interface Display and you want users to only edit certain cells. Only those **cells that are not locked will be editable**. This also applies to the 'xTagDB' sheet, so make sure that changeable cells are not locked.

Names

The name section allows the creation and removal of names. Names are a vital part of **eXLerate**. By using names (sometimes also referred to as 'named ranges'), an application can be simplified. Instead of referring to a cell-address (e.g. "B78") it is possible to refer to "MeterPressure", which is far easier to understand. Names can be created for a single cell or a multiple cells (range).

The current name of the cell is automatically displayed in this section. If no name exists, then the edit-box is empty. To create a new name, select the cell and type a new name in the Name edit-box, following by the ENTER key.

By clicking on the  icon, the Names Manager is displayed which shows an overview of all available names.

Styles

You may want to define a certain number format for all pressure related values. Using styles you can create such a definition and rapidly apply it to your application. The styles section shows the style that is selected for the current cell. To choose a different cell for the selection, choose a different style from the combo-box or create a new style.

Insert Section

The Insert section contains the most common options for inserting and selecting objects.

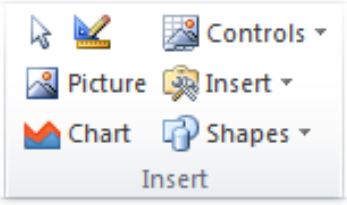


Figure 3-12: Insert section

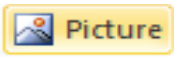
Select Objects

This option toggles the select objects state. This useful Excel option is located on the main **eXLerate** ribbon for your convenience. When selected, this option makes it easier to select multiple objects such as shapes or controls using a dragging motion. When not selected (default), Excel will select multiple cells when holding the left mouse button and using a dragging motion.

Design Mode

This Excel option should not be confused with the **eXLerate** Design-option. This option toggles the Excel Design Mode, which makes it possible to select Active-X controls such as the **eXLerate** Trend Chart control. Only when this option is selected is it possible to move, size, rename or delete Active-X controls.

Picture

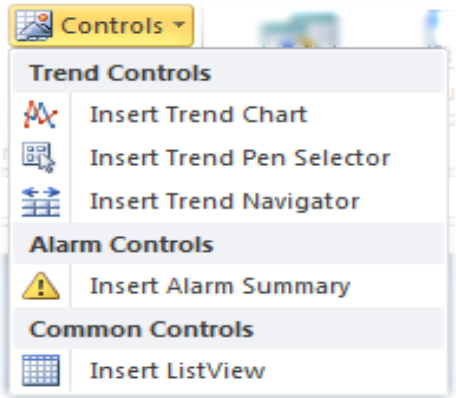


The Picture button inserts a new picture into the application. These pictures are then embedded in the application-file.

Chart

The Chart button inserts a new standard Excel Chart into the application.

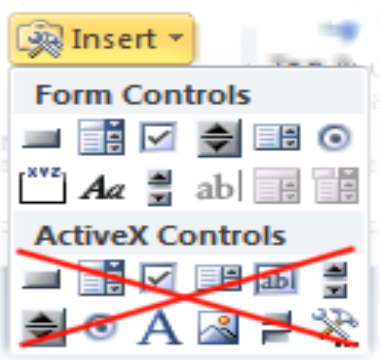
Controls



The Controls option allows you to insert **eXLerate** specific controls such as trend-charts, alarm-summaries and generic list-views.

After inserting a control, the Excel Design Mode is selected in which you can move and size the control. Click the Design Mode button to enter or exit this mode.

Insert

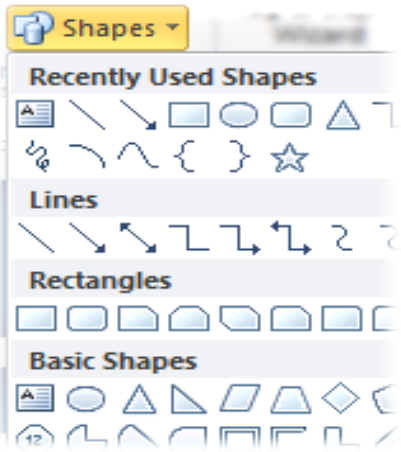


The Insert option allows you to insert standard Excel controls such as buttons and check-boxes.

We advise to **only use the Form Controls** and **not** the ActiveX controls.

Apart from the standard Excel Form controls, shapes may also be used as buttons.

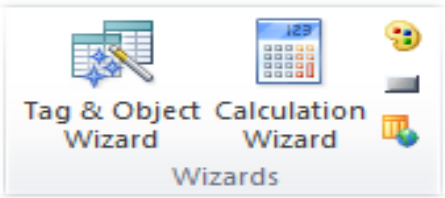
Shapes



The Shapes option allows you to insert a large variety of standard shapes.

Apart from all standard shapes, shapes can be easily grouped together to form new shapes. The 'Library' sheet also contains a set of predefined shapes and pictures.

Wizards Section



A wizard in **eXLerate** is like a Tool, except that the result of a wizard has a bigger impact on your application, because a Wizard adds items to your existing workbook, for example an alarm list, menu navigation buttons, or even a complete color table.

Figure 3-13: Wizards section

Tag & Object Wizard (Ctrl+W)

The Tag & Object wizard is a powerful tool for application generation. It is used to automatically create various pieces of your application, such as tag object names, periodical calculations, alarm summary pages, or navigation bars at your display pages.

Calculation Wizard

The calculation wizard creates *calculation tag names* in a special worksheet, called a *calculation sheet*. A calculation sheet is a specially structured and formatted worksheet containing calculations of your application.

Color Wizard

Animations use colors from a predefined palette. These colors are located on a *Color Table* on the 'xTables' worksheet. The Color Wizard copies these colors to the palette of Excel after which they become active in Excel..

Button Wizard

The button wizard creates the function key button bar at the bottom of each display page with all associated VBA macros from the corresponding definitions in the *Button Table* in your application. The function key bar is used for menu navigation through your display pages. The Button Table is located in the 'xTables' worksheet. Using the Button Wizard, creation of menu navigation functionality in your application is highly automated.

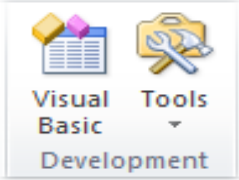
Language Wizard

The language wizard creates the 'xLanguage' worksheet for your application. This language worksheet can be used to extend the application with multiple languages. It can also be used for modifying system texts such as dialog texts and alarm texts.

The difference between the Tag & Object wizard and the calculation wizard is that the Tag & Object wizard generates various pieces of your application, at various worksheets, while the calculation wizard only generates *calculation tags* at a calculation sheet.

These wizards are discussed more in detail in the '*Advanced Topics Reference*', so you can have both books opened at the pages of interest.

Development Section



The Development section gives access to the Visual Basic (VBA) programming environment and a range of tools. Visual Basic is also accessible through the keyboard short (ALT+F11).

Figure 3-14: Development section

Tools

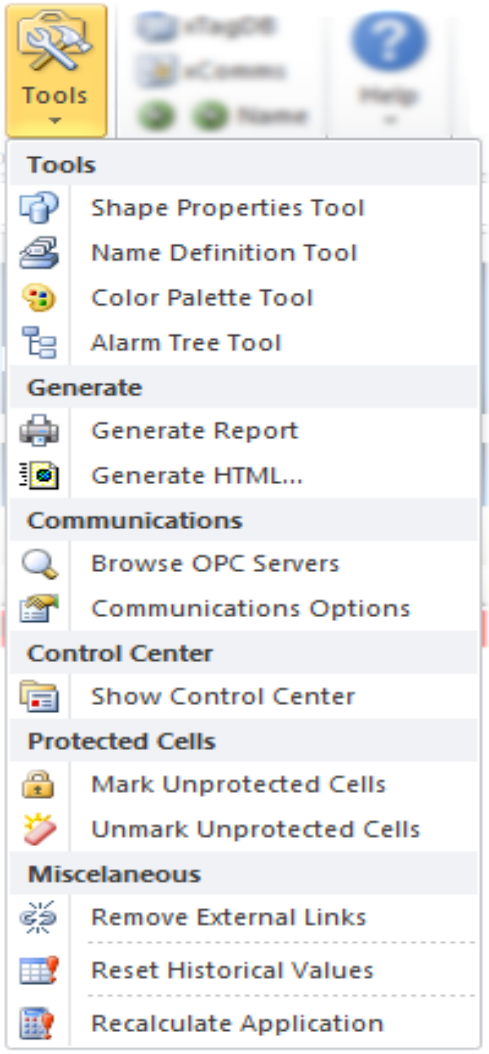


Figure 3-15: Tools

Shape Property Tool

The Shape Property Tool is a tool showing the properties for shapes, especially for shapes that are to be animated with live data. Using this tool, existing shape properties may be copied into the *Animation Table*. In addition, a cross reference shows the usage of a shape over various worksheets.

Name Definition Tool
The Name Definition Tool is a pop-up dialog showing all currently defined (object) names in the application. It has similar functionality as the Excel built-in Names tool, except that the presented list-box of the Name Definition Tool is more elaborate than the built-in names tool.

Color Palette Tool
The Color Palette Tools gives an overview of the currently defined color palette, and their index numbers, which are required for animations.

Alarm Tree Tool
This tool shows the currently configured alarm directory tree. In **eXLerate**, an alarm for a tag in the tag database may be grouped in a directory-oriented tree structure. With this tool, the currently configured alarm tree is displayed.

Generate Report
The 'Generate Report' option may be used to generate a predefined report. Reports are generated using a report template. A report template is a standard Excel worksheet, in a separate report workbook, or internally in the current application, in which reports for your application are defined. Reports may be also automatically generated, for example on predefined intervals, or on a special event.

Generate HTML
This option is available to force generation of the defined HTML pages in the application. Normally, HTML pages are generated periodically and automatically by **eXLerate** at a defined interval. Using this option however, all defined HTML objects are generated at user request.

Browse OPC Servers
This option displays the OPC Server Browser.

Communications Options
This option shows the diagnostic and logging options for the communication protocols. The protocols themselves are configured on the 'xComms' worksheet,

Show Control Center
This option shows the Control Center application.

Mark Unprotected Cells


Because worksheet cells may be protected during runtime mode, and it is not evident to see which cells are protected and which aren't, this tool clearly marks all unprotected cells in the current worksheet with a light pattern.

Unmark Unprotected Cells
This menu options un-marks the cells that were marked with the previous command.

Remove External Links

Sometimes a workbook contains external links to another workbook, for example because of the fact that cells-values or worksheet functions were copied from another workbook. Using this option, such external links are removed, and all external link references are reverted to the current application workbook. To check if there are external links in the first place, see the 'Edit', 'Links' menu in the standard Excel menu. When the 'Links' menu option is grey, no external links exist in the current application. This option should be used with caution.

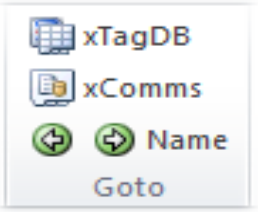
Reset Historical Values

The automatically calculated values, for example hourly averages may be reset to 0 using this option, for example to test the application, and remove all existing counters and historical values.

Recalculate Application

When this option is chosen, all open workbooks are recalculated. The system flag: '`xAutoRecalc`', which is used to force recalculate expressions in Excel on a system restart is updated as well. In addition, shape animations are updated.

Goto Section



The Goto section contains several navigation options for quickly navigating the application. The 'xTagDB' and 'xComms' options represent worksheets. When pressed, these worksheets are selected,

Figure 3-16: Goto section

Goto Name (Ctrl+Q)

This option searches for the definition of the referred name in the current cell. For example, when the current cell contains: '=xTR1TA.Value', this option jumps to the cell bearing the name 'xTR1TA.Value'.

Goto Last Position (Ctrl+Shift+Q)

This command returns to the last cell that the editor was located at before the 'Goto Name' option was activated.

Help Section

The final section in the **eXLerate** ribbon is the Help section. It provides access to the documentation, license manager and the about-screen.

Worksheet functions

In **eXLerate**, a large number of worksheet functions are available to implement the required functionality for your application.

Functions may be inserted using the 'Insert Function' button on the formula-bar.

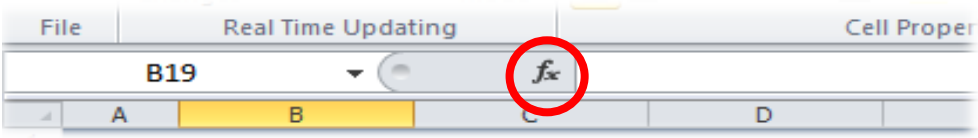


Figure 3-17: Insert worksheet function option

You may insert any type of worksheet function, **eXLerate** and non-**eXLerate** specific. You may also create your own worksheet functions that may be inserted in an **eXLerate** application, or in any other type of Excel workbook.

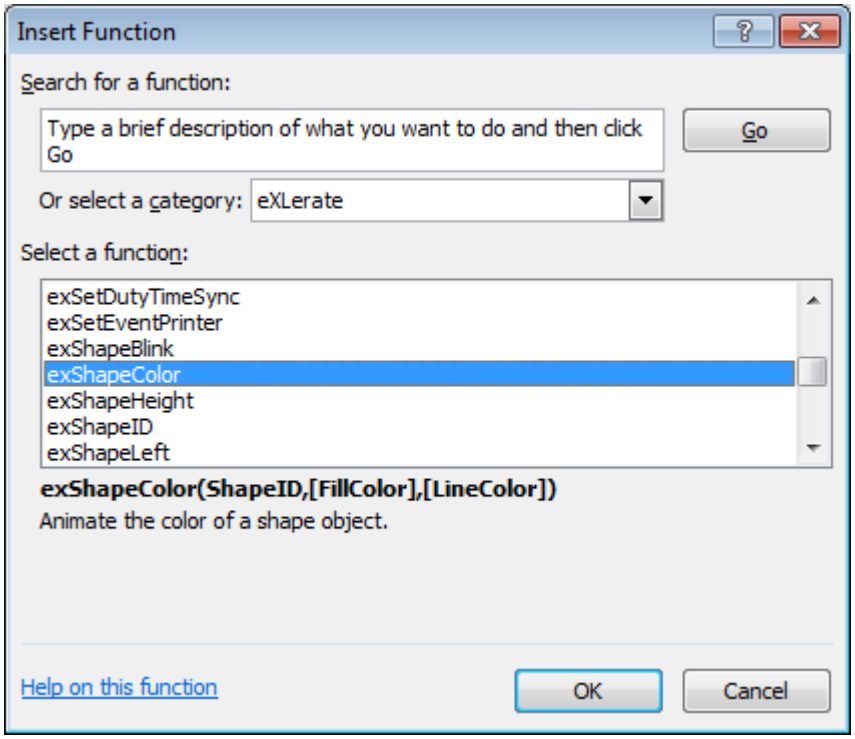


Figure 3-18: Insert Function dialog

For each worksheet function specific help is available by clicking the 'Help on this function' shortcut.

For more complex worksheet functions, the Tag & Object Wizard is available. You may also examine one of the supplied example projects for additional help on using the worksheet functions of eXLerate.

Browsing through the application

In the following sections, we play around a little bit, and see which components an eXLerate project exist of.

Browsing through various worksheets

The worksheets at the bottom of Excel may be browsed through, just like any other Excel workbook. Please check briefly the following displays of the application:

- **Template**
  This worksheet contains a template for a new display page. It contains a standard header section with user information, display name, and the current date and time. At the bottom of the display there is a set of buttons, with the name '**Button1**', '**Button2**',,…,'**Button12**'. These buttons as well as the surrounding frame may be used as a template for all other display page buttons.

- **LibrarySheet**
  The 'LibrarySheet' contains a predefines set of pictures and shapes. These object may be copied to other worksheets using the standard Copy/Paste commands in Excel. You can extend the 'LibrarySheet' with your own objects.

- **sAlmSum (Alarm Summary)**
  The alarm summary is a worksheet containing a control which displays the alarm-summary.The alarms are updated automatically during runtime mode. The user may define the looks of the alarm summary page. If you don't want an alarm summary in your application, you may remove this worksheet from your workbook.

- **sAlmHis (Alarm History)**
  The alarm history display page mainly exists for convenience during runtime operation of a system. It contains alarm specific messages, which is a subset of the logged event messages as available on disk and the eXLerate Control Center program. If you don't need an alarm history window, you may delete this worksheet from your application.

- **sTrend (Trending)**
  The trending display page is a standard Excel worksheet containing three trend controls which work together. In the tag database you determine which and how tags are to be recorded in the trend database. In this 'sTrend' worksheet the recorded data is displayed. If you do not require

trending in your application, and it is not part of your license, you may remove this worksheet from your application.

- **Various sample sheets**
  There are various sample display pages, reports, HTML pages and sample sheets, which may be studied, as part of your **eXLerate** training, freely be modified or removed as required.

- **A number of background sheets starting with an 'x'**
  There are a number of background worksheets with a special purpose. These sheets are internal **eXLerate** sheets, and are used for application engineering. All **eXLerate** applications contain at least a tag database ("xTagDB"), and a number of configuration tables ("xTables", "xAnimations", "xEvents", "xComms"), which the user may modify.

  There are also two hidden worksheets ("xWizard", and "xAlarmList"), which are generated by the Tag & Object wizard of **eXLerate**. These two worksheets are typically the last two sheets in your application.

- **Your own worksheets**
  You may add as many display pages and worksheets, as you would like. It is recommended that you separate your worksheets into display pages with just visualization and animation, and worksheets for calculations. This eases the process of testing your application. Tips on using calculation worksheets are discussed in the '*Advanced Topics Reference*' manual.

Starting real-time updates

Let's now revert back to the dynamics of **eXLerate**, and go back to the **eXLerate** ribbon.

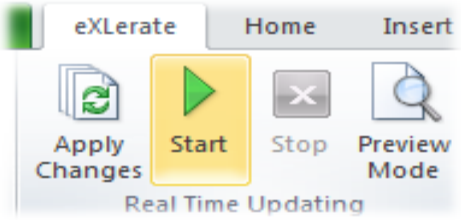
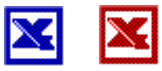


Figure 3-19: Starting real-time data updating

The '**Start**' option enables real-time-updating. Real-time-updating causes Communication protocols to run, animation to blinks, databases to activate, network connections to be established, any many more.



Also watch the icon of eXLerate itself change from its normal blue color (not updating) into red (real-time updating) at the left top corner of your display.

In your normal applications, data would be coming from external devices, such as a process computer, (remote) MySQL database server, DCS/SCADA system, flow computer, recorder, transmitter, electronic meter or any other equipment capable of external communications, including another Personal Computer. Please check available communication drivers and/or other sources of real-time updates at your local distributor or at Spirit IT.

Switching to Runtime/Preview mode

Now switch to runtime mode, by selecting 'Runtime mode' or 'Preview mode', and watch the Excel development environment disappear.

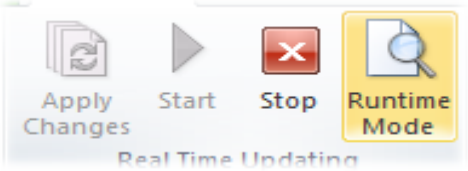


Figure 3-20: Switching to runtime mode

When real-time-updates is started, this option is called 'Runtime Mode', otherwise it is called 'Preview Mode'.

Instead, operator pages appear, but now without any Excel menu, and without the possibility or make modifications in the worksheets. You may now use the function buttons at the bottom of the page. Browse through the application display pages using function keys <**F1**> through <**F12**>.

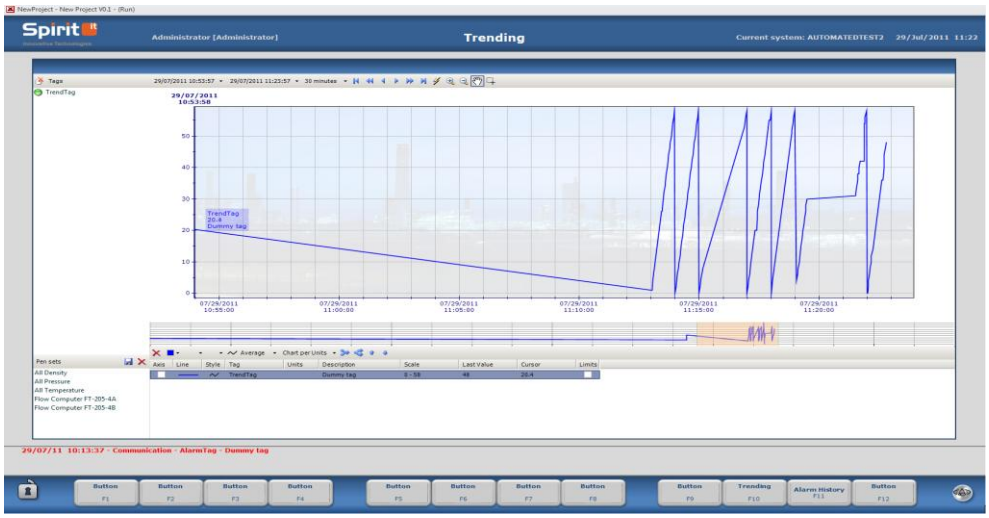


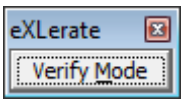
Figure 3-21: Previewing display pages in Runtime mode

In Runtime/Preview mode

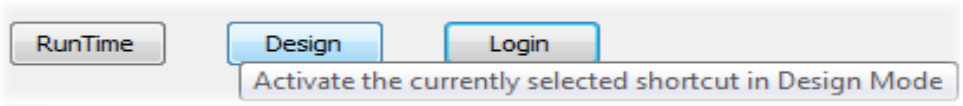
When real-time updates are indeed active, pressing the <Esc>-key has only effect when the current access level allows for 'Design mode'.

When in Runtime/Preview mode, there are two ways to revert back to design mode:

- By pressing the <Esc>-key, this shows a small exit button in the application. Note that pressing the <Esc>-key has only effect when the current access level allows for 'Design Mode'.



- By switching to the Control Center (which is minimized in the Windows Taskbar Tray); and selecting the '**Design**' button.



The tag database worksheet

Please revert back to Design-mode for now, and select the worksheet called: "xTagDB". The tag database worksheet is the 'beating heart' of an **eXLerate** application, since it contains both the definition of all tags in the system, as well as its current value.

If you have trouble in finding this sheet, there is a convenient shortcut for this sheet located in the 'Goto'-section of the **eXLerate** ribbon.

Please note that:

- For each tag, one row in the worksheet is available. There may be as many tag rows as your current **eXLerate** license allows for. In the example project template, the tags are ordered in logical groups, like 'PLC', 'Line 1', 'System' etc. A row in the tag database may be considered as a record in a relational database.

- There are a number of columns. Each column represents a property of the tags, such as its tag name, its current value, engineering units, or alarm limit value. A column may be considered as a field in a record in a relational database.

- Most columns in the sample project are mandatory and required by **eXLerate**, while other columns are user-defined, and are available for user convenience. User-columns may be added and deleted, but the required columns for **eXLerate** should not be removed.

- Some columns have a trivial meaning, such as 'Group', or 'Tag name', while other columns seem more ambiguous, such as column 'P_Min', or 'Query'. In '*Application* ' on page 5-103 onwards, the tag database is fully described.

- Note that when you set the cursor over the current value of a tag, the name box at the right top of the display in Excel shows a logical name, such as 'xAlarmTag.Value'. Instead of referring to cell 'G23' it is much better to use 'xAlarmTag.Value' if referring in an equation to the current value of tag 'PT1'. **eXLerate** has the ability to automatically generate such logic object names. You need the Tag & Object wizard to generate such objects names for you.

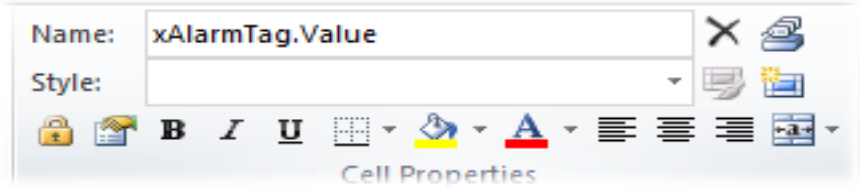


Figure 3-22: Usage of logical names in the tag database

Check various columns, and watch how the data in the 'Value' column automatically changes, when your real-time updates are still active.

Please note that the tag database worksheet is protected during real-time updates. All cells that are setup for live updates should therefore be unprotected. The lock-symbol, displayed in the figure above at the left indicates whether or not a cell is protected.

Normally, the entire 'Value' column is unprotected, unless a formula is entered in a cell rather than a straight value.

The tag database looks as follows:

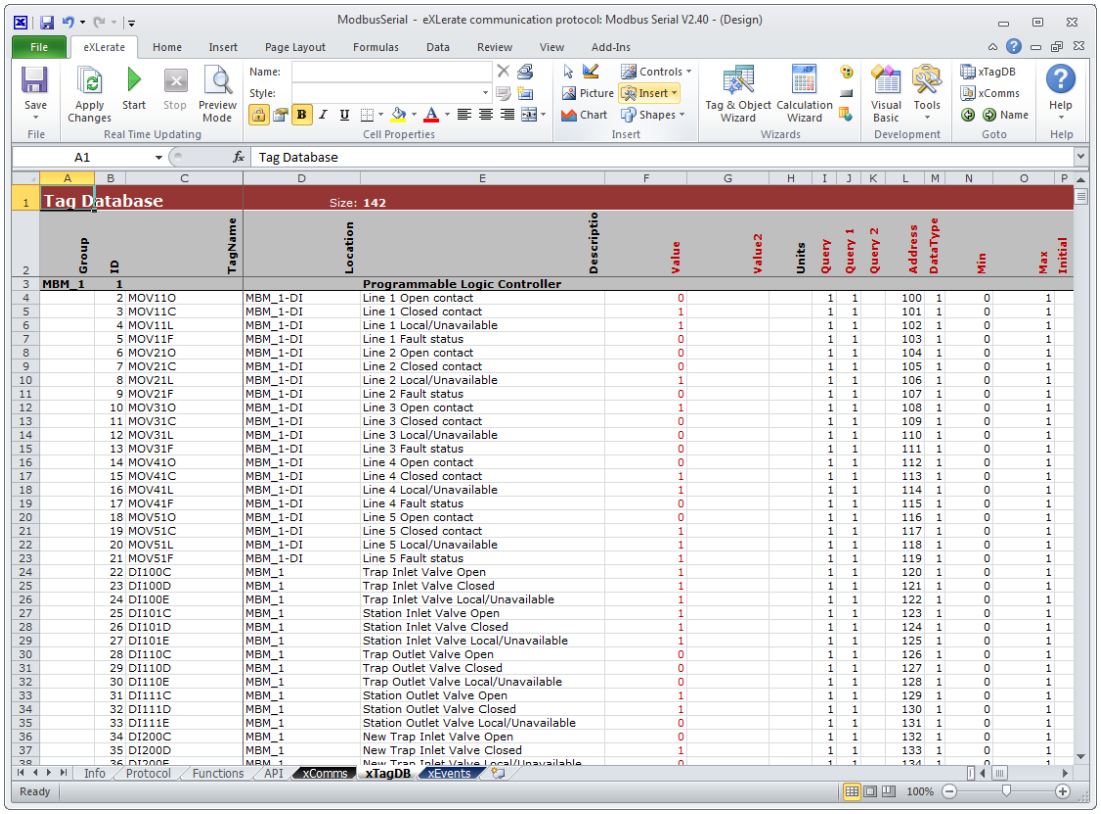


Figure 3-23: worksheet: 'xTagDB' containing the tag database of eXLerate

Internally, e*XL*erate continues calculating during the Excel cell edit-mode.
When you have examined some or most of the tag database, go back to the 'Sample' display page by clicking on the appropriate tab at the bottom of the Excel workbook.

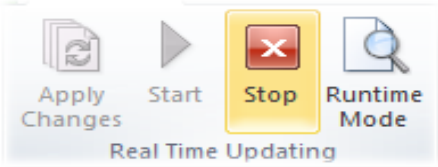


Figure 3-24: Stop real-time data updates

For now, stop real-time updating via the e*XL*erate menu, and take a look at various cells in the worksheet. Displaying a number from the tag database is very easy in Excel: simply refer to the appropriate cell in the tag database, as with the pressure value at the top of the display page, which shows: '=xTR1TA.Value' in the formula bar of Excel. It's that simple!

Shape animations

Now that you have seen value changing directly in a spreadsheet, it is about time to start looking for animated objects. How does that work in **eXLerate**?

eXLerate is able to dynamically set various properties of a *shape*. A shape is a drawing object in the Microsoft Office family products. Shapes can be user-drawn, or imported into an application. Properties that **eXLerate** is able to change, are the color, the size, position, rotation angle, and visibility.

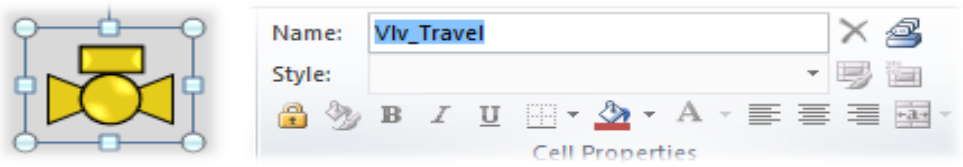


Figure 3-25: A selected shape, and the Excel name-box with the shape name

Click on the left top valve in the 'Sample' display, and check the name box at the left top of the display. Check that the name of the valve is 'valve_11' in the template project.

Now type <Ctrl-1>, or right-click the object and select 'Format Shape' to display the properties of this shape object. The following object property dialog appears:

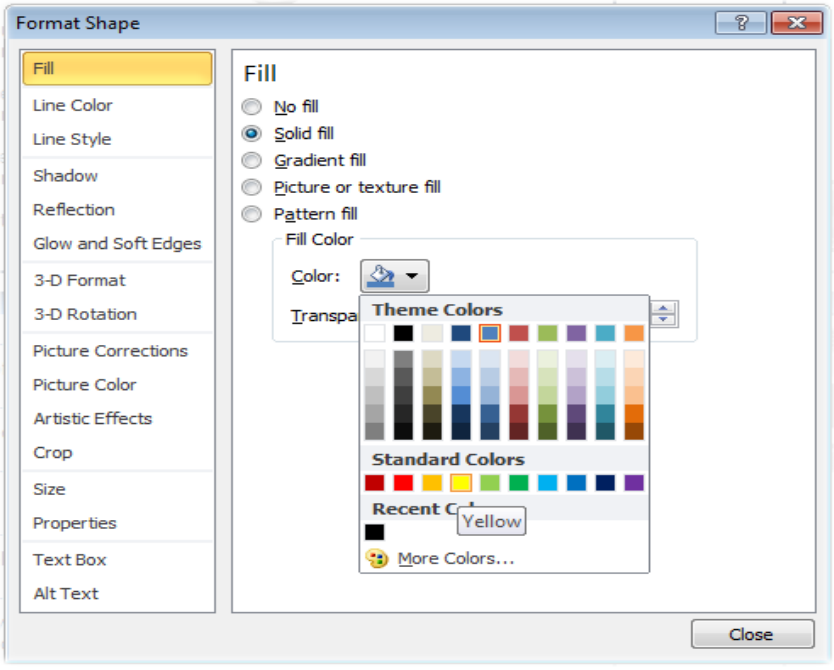


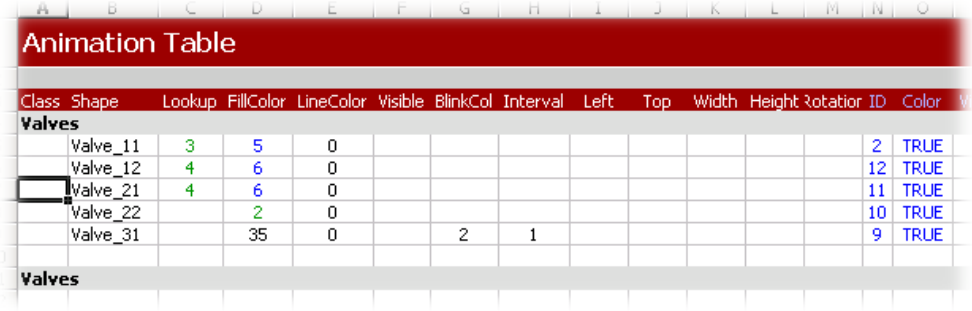
Figure 3-26: Shape Object format dialog

Using this shape object format dialog, colors, lines, size, shape protection settings, and other properties may be defined. For example, change the color into another color, and see the shape changes on the worksheet. **eXLerate** also uses these shape properties internally for its data-driven shape animations, just like you are able to change these properties using the Format Object dialog.

The Animation Table

In **eXLerate**, animations of shape objects in all display pages are stored in just one single table, which is called the **Animation Table**. The Animation Table is stored in worksheet 'xAnimations'.

In order to display the Animation Table, select the 'xAnimations' worksheet, and verify that there is table with object names, and that the object 'valve_11' is one of the entries in the table.



Animation Table

Class	Shape	Lookup	FillColor	LineColor	Visible	BlinkCol	Interval	Left	Top	Width	Height	Rotatior	ID	Color
Valves														
	Valve_11	3	5	0									2	TRUE
	Valve_12	4	6	0									12	TRUE
	Valve_21	4	6	0									11	TRUE
	Valve_22		2	0									10	TRUE
	Valve_31		35	0		2	1						9	TRUE
Valves														

Figure 3-27: Fragment of the Animation Table

The *Animation Table* has various rows, one for each animated object, and a number of columns, one for each animation property.

Columns at the **left**-hand side of the table contain actual shape **properties**, i.e. the settings in the application for the object, while columns at the **right**-hand side of the table contain worksheet **functions** for the appropriate shape.

The following column properties are available for an animated shape:

Shape, *FillColor*, *LineColor*, *Visible*, *BlinkCol*, *Interval*, *Left*, *Top*, *Width*, *Height*, and *Rotation*.

The values at the table at various rows contain the actual dimensions for a shape. For example, the 'Valve_11' object might have a defined FillColor of '2', and a 'LineColor' of '0'.

If you are looking for a relation between a number, and its actual color, press <Ctrl-L>, for the Color palette Tool, as follows:

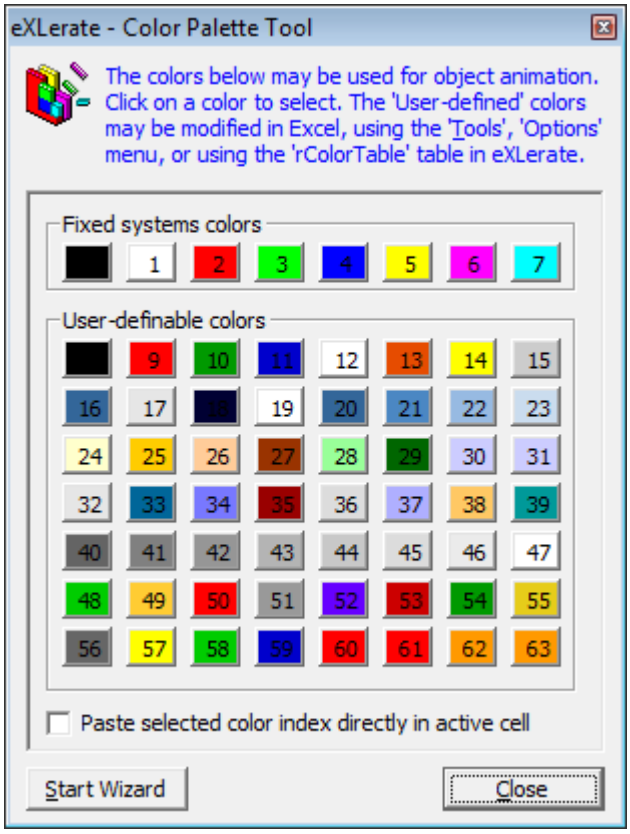


Figure 3-28: Color Palette Tool

You may click on a color to paste the associated color index directly into a cell formula box, or, when the checkbox at the dialog isn't checked, paste it into an expression with the <Ctrl-V> paste key.

For now, close the Animation color palette dialog box, and let's continue with the explanation of the animation table.

The internal **eXLerate** functions, which are used for shape animations are stored at the right-hand side of the animation table, in the following columns:

ID, *Color*, *Visible*, *Blink*, *Pos*, *Size*, and *Rotate*.

In the reference chapters you'll learn all about using these functions. For now, it is important that you understand the concept of **eXLerate** regarding shape animations:

- **eXLerate** stores all animations of your application in a single table, the *Animation Table*;

- Current property values are stored in the Animation Table at the left-hand side, in various columns. For example to change the color of a valve-shape, the fill-color may contain a worksheet function, which depends on the status of a valve.
- These properties are passed to eXLerate using worksheet animation functions. All animation functions are stored at the right-hand side of the animation table.

If you want to add an animation of a shape, do the following:

- Simply add a row in the animation table using copy and paste of a full row, and fill in the name of your shape object in the copied row at the column 'Shape';
- Set the properties that you want to animate under the appropriate columns, for example the object color, size or position;
- Make sure that the worksheet functions at the right-hand side of the table are correct; this is required to tell eXLerate that you want to animate a shape. Remove unused cell functions.

Perhaps the most convenient way of adding a new animation is to simply copy an entire line, using the <Ctrl-C>, <Ctrl-V> keys of Windows, and change the copied name to your own shape object. That's all!

Page Navigation

At the bottom of each display, there is a frame containing 12 buttons. These buttons are used for page navigation. In the sample project, the function key button-bar looks as follows:

Figure 3-29: Section of a button-bar for page navigation

Each button has a corresponding function-key, <F1>..<F12>, which may be used to navigate through the display pages. This allows for systems without a pointing device, where only a keyboard is present.

The text on each button has the function key name displayed in **bold** text, and the associated display page or other function displayed below, such as 'Login', or 'Reports'.

Internally, a button is linked to a VBA macro, which is responsible for selection of another display page or pop-up menu.

Right-click on an individual button, and verify that a VBA-macro is linked to the button.

For example, the <F2>-button has the text: 'Alarms', and internally a link to the macro: '`Call_Load_AlarmSummary`'.

You may verify this by right-clicking the <F2> button while in design-mode, after which the pop-up menu as given on the left is shown. Verify that there is a link to the foresaid macro exists.

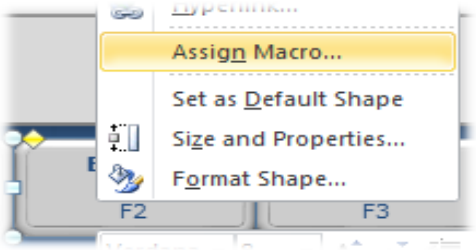


Figure 3-30: Macro assigned to a navigation-button

In **eXLerate**, the creation of such a function key button-bar, and the underlying menu navigation, are automated utilizing the Button wizard of **eXLerate**.

eXLerate uses the button bar in the 'Template' window as a template for all other display pages, so you don't have to worry about exact placement of a button-bar at each display page.

eXLerate copies this button bar to all other display pages when invoking the *Button Wizard*. The contents of the button bar, i.e. both the text on the button-face as well as the functionality 'behind' each button is stored in an Excel table called the *Button Table*. The *Button Table* is located at the 'xTables' worksheet.

The *Button Table* and associated automated functionality allows for easy configuration of your display pages, saving your precious time for more important aspects of application engineering.

Other tables

Now that you have been viewing the 'xTables' worksheet, let's examine the other tables presented at this worksheet; you will need this sheet as you will start develop your own applications.

When the worksheet 'xTables' is in O*utline*[*] view, only the table headers are shown, as follows:

[*] *A worksheet in outline view is a standard feature of Excel: all of the grouped columns are 'closed'*

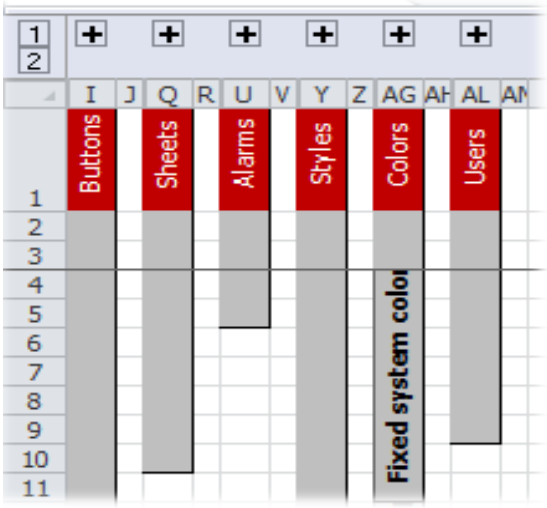


Figure 3-31: The 'xTables' worksheet in Outline view

Various tables at this worksheet have already been introduced to you. To complete your overview, an overview is given below of all relevant tables:

- The *Button Table* is used to define buttons for menu navigation. *This table is required for each application*.
- The *Worksheet Table* is used to define certain runtime worksheet settings, such as worksheet protection, security level for visibility, security level for editing, scroll-range, HTML-page range, and HTML refresh settings. *This table is required for each application*.
- The *Alarm Group* table is used to define alarm groups, which are hierarchically organized. *This table is optional for an application*.
- The *Styles* table is used to define Excel styles. Styles contain formatting settings such as Bold, Font-Size, etc…, and can by quickly applied to a cell. *This table is optional for an application*.
- The *Color Table*, which defines the color palette with 56 colors in your application. *This table is optional for an application*.
- The *User Level Table*, which defines which user levels are defined in your application. *This table is optional for an application*.
- The *Time Table*, which is used to define changeovers for summertime and wintertime in your application. *This table is optional for an application, and may be removed when no summertime and/or wintertime changeovers are used*. Summer- and wintertime changeovers are discussed in the '*Advanced Topics Reference*' manual.

You can open each table by clicking on the '+'-button. When the *Worksheet Table* is opened, the worksheet may look as follows:

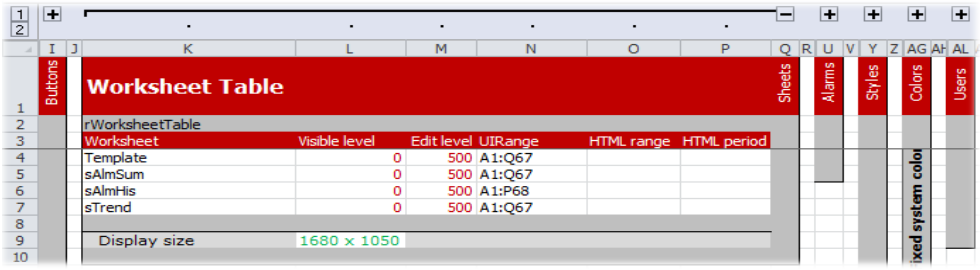


Figure 3-32: xTable worksheet opened with the Worksheet Table 'opened'

If you feel more comfortable with a worksheet without outline views, you may remove the column groups; it is only used to structure your application.

To remove the outline view, Select the relevant columns. In the example above, select columns K-P using the mouse. From the 'Data' ribbon in Excel, select 'UnGroup'. The outline view disappears. When only cells K2..P2 are selected, Excel shows the following dialog to verify if the 'UnGroup' command is related to columns, or to rows. In this case, select 'Columns'.

You may add you own tables to this worksheet. It is recommended to add other tables for your own application, to this 'xTables' worksheet, so all 'background' tables are collected in a single worksheet. You wouldn't have to search a lot to find a table in your application.

Conclusion

In this chapter you were acquainted with the most common features of a complete **eXLerate** application:

- You have started an application in design-mode, in runtime mode, and in preview mode. In the next chapters, you can read a lot more about the control center and application engineering.
- You have started and stopped real-time data communications and viewed changing of live data and objects in Excel. In the next chapters, you will learn at lot more about real-time data communications.
- You have looked at various standard display pages, such as a trending page, an alarm summary, an alarm history and a tag view page. In the next chapters, you will learn to create your own display pages.
- You have looked at various **eXLerate** application components, such as the tag database, and the Animation Table. In the next sections, you will read all about the configuration tables in **eXLerate**.
- You have looked at menu-navigation in an **eXLerate** application, and the *Button Table*.
- You were able to look at various additional tools, such the color wizard. In the next sections, all of the built-in development tools in **eXLerate** will be introduced to you. In the '*Advanced Topics Reference*', the wizards and tools are discussed in detail.

Chapter 4 - Control Center reference

Introduction

The Control Center in eXLerate is the main executable in which all projects in eXLerate are controlled, monitored, added, removed, changed, started, stopped, and maintained.

In Chapter 3 - Tutorial, important functionality of the Control Center was already discussed.

In this chapter all remaining functionality regarding user accounts, application shortcut properties, the system event logger and other issues is discussed.

Control center functions

The Control Center takes care of the following:

- eXLerate project control and maintenance.
- Monitoring, printing, and disk-storage of all event messages that are generated with an eXLerate component or module.
- Maintenance of eXLerate users and their associated security levels.
- External executables, which are allowed for the operator during runtime operation of a system. Any executable file may be added to the Application shortcut list, for example a word processor, the Windows Notepad, or the License Manager license utility.
- System security. The Control Center is able to prevent certain Windows actions, such as using the Windows key or task switching, depending on the current user level. This function allows eXLerate to be used in environments where a user is not allowed to use the computer for anything else but the running eXLerate application.

A shortcut to the Control Center program is created by the setup program in the Start Menu of Windows, so eXLerate may be conveniently started using the Windows start menu.

The Application Control Center may be also used to automatically launch a project in Runtime mode, so your project may be automatically started at computer power-on.

In the following sections, Control Center functionality as far as not already discussed in *Chapter 3 - Tutorial*, on page 3-52 above will be discussed in detail.

The Control Center, a dialog application

When the Control Center has been successfully launched, a dialog similar to *Figure 4-1: the eXLerate Control Center program* is displayed:

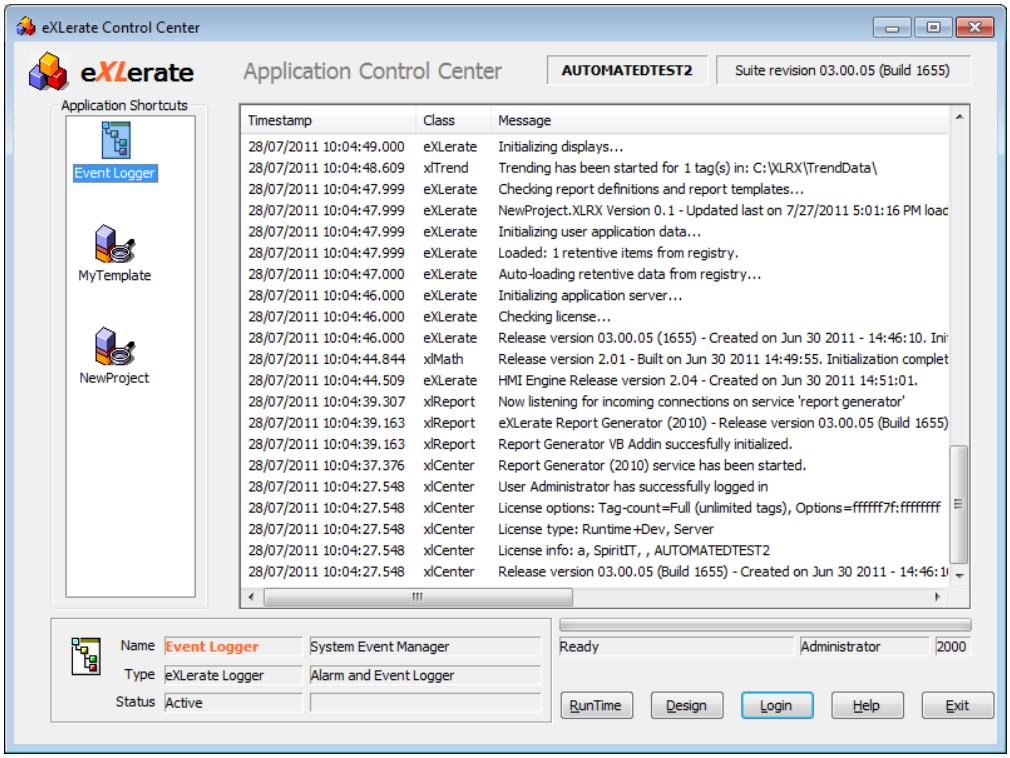


Figure 4-1: the eXLerate Control Center program

Since most buttons are grey at Control Center launch, we should login first, to be able to start using the Control Center.

Functionality in the Control Center is coupled to the security level of a user.

User Accounts

Refer to section *The eXLerate 2010 Control Center* on page 3-52 onwards for an explanation of available login accounts and associated security levels.

Basically, a user has a name, a password, and a security level. The security level is internally used by the Control Center itself, and may be additionally used by an application developer to create application dependent security functions.

For example, one could create a special dialog in VBA, which is only accessible for certain users, e.g. a motor operated valve may only be opened or closed with a button in a dialog when the user is an official operator, and not by a guest just looking at the display pages.

Users may be created only by a user having administrator rights, i.e. a user with security level of at least '2000'.

In order to add/delete/modify the list with current users, go to the System Menu of the Control Center, and select the 'Edit Users…' menu option.

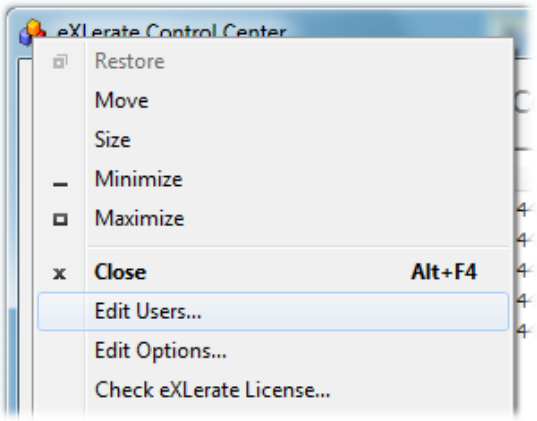


Figure 4-2: Edit Users option

If your security level is not sufficient to modify the current user accounts, the 'Edit Users…' option is disabled, and cannot be selected.

When selected from the system menu, the following dialog is presented:

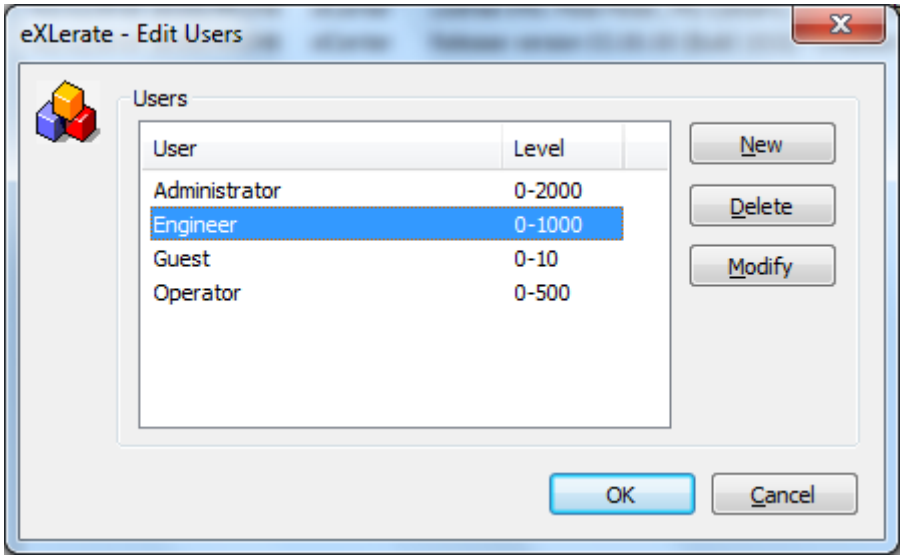


Figure 4-3: Edit Users dialog

Users can be added/remove/modified using the appropriate buttons. When editing a user the following dialog is presented:

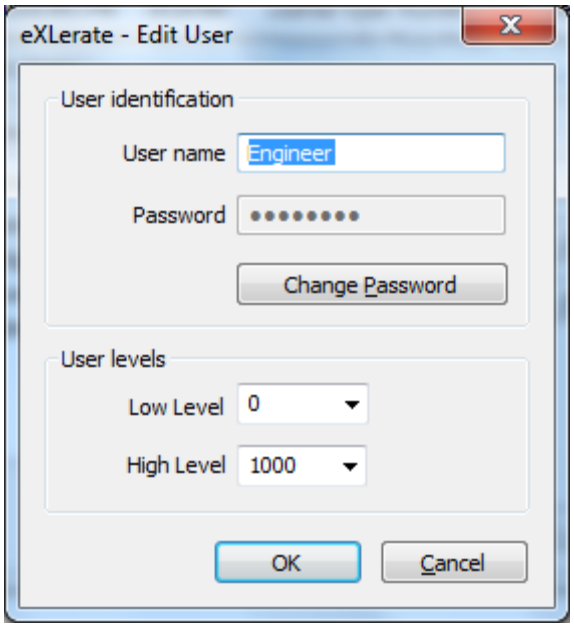


Figure 4-4: Edit User dialog

For each user, a name, a password, and two security levels should be defined. In the following text, you will find out why there are two security levels available.

Security levels are both used by the Control Center for application access, as well as in an application. Application development discusses security in a next chapter.
Application access is coupled to various actions:

1. To start an application in Runtime mode,
2. To start an application in Design mode, and
3. To close a currently running application.

Access is granted to one of the actions above if the level of the currently active user lies between the minimum and the maximum level. This allows for flexible application engineering.

Security strategy

For example, an application developer may start an application in design-mode, to make a modification to a certain display page, but (s)he may not be allowed to actually start an application in Runtime mode. Running an application would be only allowed for an operator.

On his turn, an operator may be allowed to operate a control system, but not to make changes in an existing application.

In addition, an operator may be allowed to start a project at computer startup, but not allowed to terminate a running application.

Flexibility is the keyword in **eXLerate** on security level usage!

If this all sounds quite complex to you, forget all about it. In the "MyTemplate" sample project, user levels are already implemented for you.

Application shortcuts

An application shortcut in the **eXLerate** Control Center is similar to a regular program shortcut in Windows, except that in **eXLerate**, additional properties exist for a shortcut.

To show the current properties of a shortcut, you may right-click an application shortcut, after which the application shortcut menu appears on the dialog:

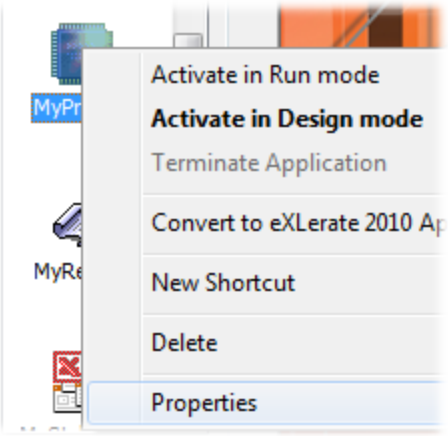


Figure 4-5: The application shortcut menu

Using the application shortcut menu, existing entries may be deleted, new entries may be added, and the selected application may be started in Runtime mode, in Design mode, or may be terminated.

Shortcut property dialog

For now, select the 'Shortcut properties' menu option, which shows the shortcut property dialog, as follows:

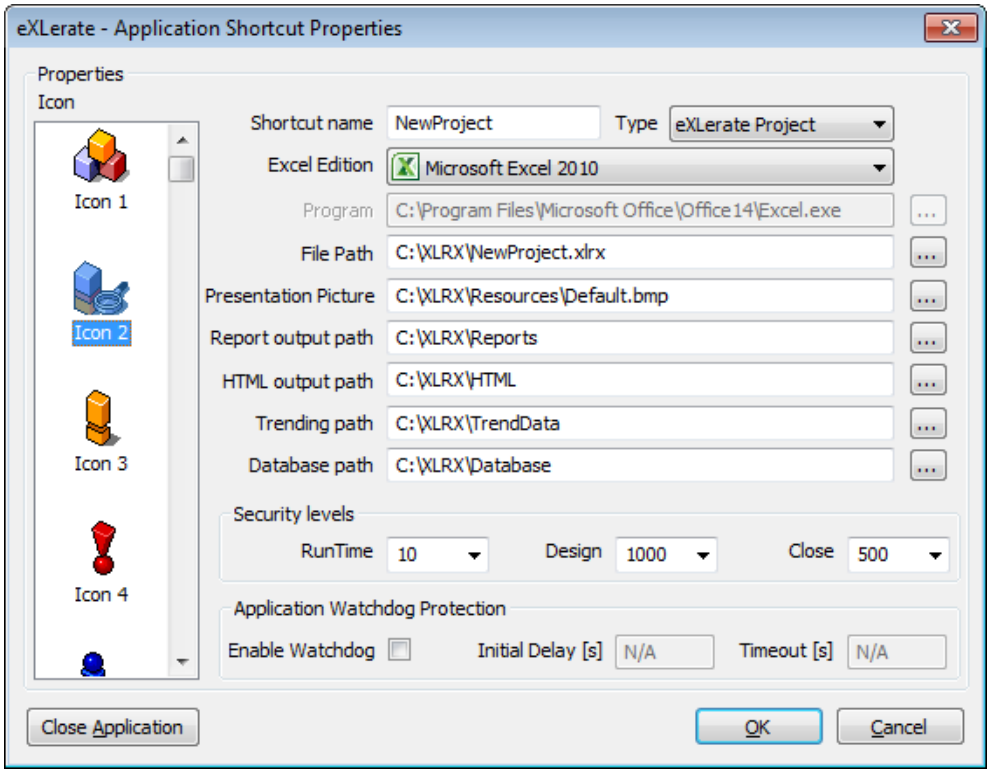


Figure 4-6: Application shortcut property dialog

This dialog may be also displayed while double-clicking the application shortcut. In this dialog, you may define the following properties:

- **Icon**
  For each application shortcut, an *icon* may be selected from the icon list at the left of the dialog. Double-click on an icon to set it as the current icon. The current icon is highlighted.

- **Shortcut Name**
  The *shortcut name* is the name of the application shortcut as appearing in the shortcut list at the left-hand side of the Control Center main dialog. This name will be used as a key-value, under which various application parameters are stored in the system registry.

- **Type**
  The *application type* defines what type of application is defined 'behind' the shortcut. The type may be one of: 'Windows application', 'eXLerate Logger', 'eXLerate Project', 'Program Shell', 'Task Manager', or 'Microsoft Excel'. For our **eXLerate** projects, choose 'eXLerate Project' as the default type. Other application types are available for operator convenience, in order to create a complete but restricted Windows environment.

- **Program**
  The *program* is the executable, which corresponds with the entry type. The name of the Windows executable should be exactly defined, including the absolute path to the executable, so the control center knows where to find the file.

- **File Path**
  The *file path* of an application are usually the document name, but may include other parameter as well. In case of an **eXLerate** application, it should be the name of your project workbook, including its absolute path.

- **Presentation Picture**
  The *presentation picture* is the name of an image file (.BMP or .JPG), which is used in the presentation area of the Control Center. In case of an event logger shortcut, no image is used. Instead, the logger window itself is used as presentation area. The image file should have an aspect ratio of about 3:2 to maintain the aspect ratio of the original bitmap file. When no presentation bitmap is selected, the event logger will be displayed instead when this application shortcut is selected.

- **Report output path**
  The *report output path* defines the location where your **eXLerate** application stores generated reports. In **eXLerate**, you may define report templates. Each time **eXLerate** generates a report from your templates, the generated report is stored in this location, for example on a directory on a remote file server.

- **HTML output path**
  eXLerate allows you to automatically generate HTML pages from your display pages. These HTML pages are stored in this *directory*.

- **Trending path**
  eXLerate allows you to store historical trend files in a dedicated directory. In addition, when old-style trending is used, client-only trending may be configured for client/server systems, where typically only the server computer is writing to trending files, but each client is able to retrieve trending information. The actual path for trending files is defined with this entry.

  On systems that use the new trending controls, the path is ignored on clients.

- **Database path**
  eXLerate allows you to use an embedded database. Such a database is required for client/server systems and optional for stand-alone systems. If the database doesn't already exist it is automatically created.

- **Security levels**
  Each application in the Control Center may be started in Runtime mode, in Design mode, or may be terminated. Using the pull-down box, a *security*

level may be assigned to each action. A security level of '0' means that all users are allowed to perform the associated action.

When the current settings at the dialog are accepted with the '**OK**'-button, all changes are stored in the registry. With '**Cancel**', the previous settings are retained.

- **Watchdog protection**
  eXLerate allows you to monitor an application with a built-in *application watchdog*. This watchdog of xlCenter is a timer routine that periodically checks if the application is still responding. If this is the case, than nothing happens, but when an application does not respond within the predefined timeout to the watchdog mechanism, the control center will terminate the application, and automatically restart the application again. Your application will not respond for example when in VBA, a programming loop does not end (e.g. while…wend without ever stopping.

  The application watchdog may be disabled (which is the default behavior), or enabled using various options. An initial delay may be used to only start the watchdog mechanism after the initial delay is elapsed. The timeout value after which the application should respond may be user defined as well. Typical values are 30s for both the initial delay and the timeout value.

System parameters and options

There are a number of parameters that influence the operation of **eXLerate**. These parameters include the name of the event printer, and various directories.

WARNING: Please only change these parameters when well understood, because changing these parameters without a thorough knowledge of their impact may introduce unpredictable results and may cause fatal application errors.

In order to monitor or alter these options, select the options dialog from the Control Center system menu.

When the system menu is selected, the menu below is displayed.

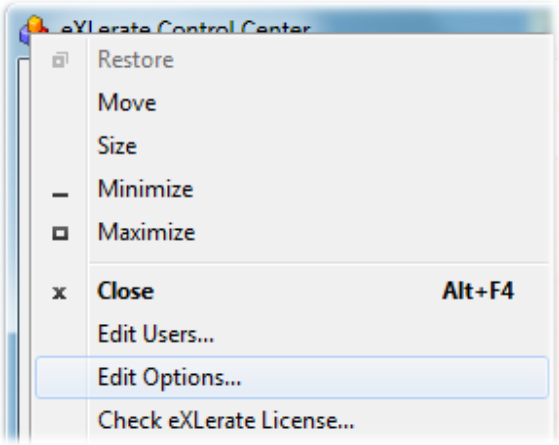


Figure 4-7: Edit Options in system menu

Select '**Edit Options...**', after which all user-definable options are displayed, in the following dialog:

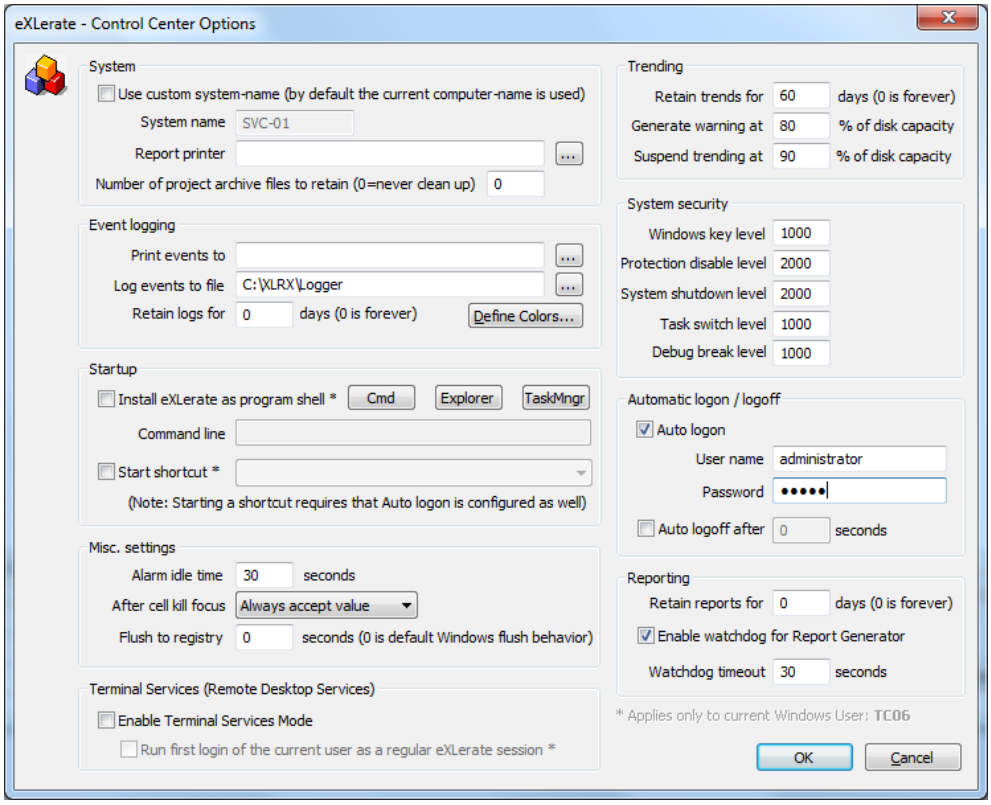


Figure 4-8: Control Center Options

The following properties and options may be defined:

System options

- **System name**
  A custom *system name* may be entered using this dialog. By default, the computer-name is used as system name. The system name is available in the **eXLerate** project, as a defined object name ('xSystemName'). In Excel, you may use the name in a worksheet function, i.e. '=xSystemName'. Please note that only the internal name within the **eXLerate** environment is changed; the system name does not affect the computer/network name.

- **Report printer**
  This field defines the *printer* to which generated reports are automatically printed. The button '**...**' right to the edit control may be used to browse the system for an existing printer. When no printer is configured, the default Windows printer is used. Additionally, an **eXLerate** project may configure a printer for each report that is configured. This will overrule the printer as configured in the Control Center.

- **Number of project archive files**
  This parameter defines the maximum number of backup files that the Control Center maintains per project. When all files are to be maintained, '0' may be entered. For example, when set at 25, the latest 25 archive files are maintained, and older files are removed from the archive directory. Made available for the user to prevent disk capacity problems. Up to 1000 backup files may be defined for each project.

Event logger options

- **Print events to**
  In case events and alarms are to be continuously printed, for example when a HMI application is in normal operation, the *name of the printer* can be defined using this dialog. This may include network printers, or locally connected printers. Standard Windows printers are supported by **eXLerate**. In case a printer goes off-line, or generates an error, e.g. an 'Out of paper' error, **eXLerate** may report this alarm to the operator via a defined worksheet name (use: '=xPrinterErrorStatus' in a worksheet to obtain an error flag). The condition of printers is checked every minute by **eXLerate**. The button '**...**' right to the edit control may be used to browse the system for an existing printer. While in runtime, the events printer can be temporarily changed using the 'exSetEventPrinter(…)' worksheet- or Visual Basic function.

- **Log events to**
  The event logger of **eXLerate** is responsible of storing each generated event in the event logger window, printing events to the assigned printer, and to store each event in a disk file. The *directory* in which the log files are stored, is defined using this edit parameter. The button '**...**' right to the edit control may be used to browse the computer for an existing path.

- **Retain logs for**
  This parameter determines for *how many days* .LOG files (which are in fact ASCII files containing all logged events and alarms) and database records in the events table will remain available on the computer. **eXLerate** removes older files from the hard disk and database, in order to limit it to a defined size. A '0' indicates forever, i.e. no files or database records are removed from the system.

- **Define colors ...**
  This button allows for user definition of colors of event messages. This feature is added to allow easy recognition of certain event messages in a system, because a specific display color can be attached to certain messages. For example, al alarm message can be colored with white text on a red background.

Startup options

- **Install eXLerate as Program Shell**
  eXLerate may be installed as a so-called system '*shell*', when this option is checked. In this case, no standard Windows desktop is available for the

user. This may be installed to prevent a system from being used for anything else but for running an **eXLerate** project, for example in production environments. These limitations are only applicable for appropriate user levels.

- **Program shell command line arguments**
  When **eXLerate** is set as Windows program shell, *additional command line arguments* may be used, such as logging in as a specific user, or by starting a certain application in runtime mode. See section: '*Command line arguments*' on page 4-101 on the syntax of command line arguments.

- **Start shortcut**
  Select a shortcut here which is started (in runtime mode) whenever the **eXLerate** Control Center is started. In order for this to work properly, a user with a sufficient security level needs to be logged in. To do this, see the next section on Automatic Logon.

There are three additional buttons available at the dialog, which causes the following programs to be started:

Cmd Starts the Windows 'CMD' process, which is the command line interpreter for Windows. Used for system administration purposes.

Explorer The Windows explorer is started when this button is pressed. Used for system administration purposes.

TaskMngr The Windows task manager is started when this button is pressed. Used for system administration purposes.

Misc. settings

- **Alarm idle time**
  This parameter specifies for how long the system *suppresses alarms* at system startup. This prevents generation/printing of many less useful alarms, being part of the startup process of a system rather than being actually alarm conditions. After this period, the alarm manager effectively starts monitoring events and alarms. When set at '0', no alarm idle time is defined at all.

- **After cell kill focus**
  This is an advanced option which controls how cells behave after they have been edited in runtime mode. For on-screen keyboards it may be useful to change this setting to a different value to get the desired behavior.

- **Flush to registry**
  eXLerate stores certain retentive data such as parameters, totals, etc… in the Windows registry. This option controls how frequently this data is

physically stored on your hard-drive. By default, the option is set to 0, which means that Windows itself determines when and how often the data is physically written to your hard-drive. For instance, when the computer is very busy, Windows will flush data later on to ensure smooth system operation. By setting this option to a fixed value, the default Windows behavior is circumvented and flushing occurs at fixed intervals.

Trending options

- **Retain trends for**
  This parameter contains the *number of days* that trend data will remain on the hard disk. When set at 0, the trending module will try to store as much data as possible, until the disk is full. This is not recommended as operation of Windows and all programs will halt as well.

- **Generate warning at**
  This parameter is used by **eXLerate** to generate a *warning* when the disk reaches a full percentage of the specified amount, e.g. when set at 80%, an alarm is generated when the hard disk becomes full for 80%.

- **Suspend trending at**
  This parameter is used by **eXLerate** to temporarily *suspend trending*. When the disk full percentage exceeds this parameter, all trend recording is suspended. When the disk full percentage becomes lower again, trending is resumed. This prevents a disk-full situation, which causes an instable Windows system.

System security options

- **Windows key level**
  eXLerate protects tampering of the system in a running application. With this parameter, the *security level* is defined below which the Windows key will be disabled, so no other tasks can be started.

- **Protection disable level**
  This parameter defines at which security level certain *protections*, such as disabling of minimize and maximize buttons of a running **eXLerate** project, will be switched off.

- **System shutdown level**
  This parameter defines at which security level *System Shutdown* will be disabled.

- **Task switch level**
  This parameter defines at which security level *task switching* will be disabled.

- **Debug break level**
  This parameter defines at which security level the ESC-key can be used to switch to “Verify” mode when in “Runtime”.

Automatic logon / logoff options

- **Auto logon**
  When **eXLerate** starts, an automatic logon can be achieved by the use of this parameter. When the checkbox is enabled and a valid user/password combination is provided, **eXLerate** will automatically logon upon startup.

- **Auto logoff**
  When a user is logged in, an automatic logoff may be effectuated via this parameter. When the checkbox is disabled, no *automatic logoff* takes place. **eXLerate** monitors activity of mouse and keyboard. When no activity has been detected for the auto-logoff time, the log-off takes place automatically.

Report generator options

- **Retain reports for**
  This parameter defines for how long generated reports are retained on the system. By default this option is set to 0 which means that all generated reports are retained indefinitely. To prevent the hard-drive from filling up, the parameter can be set to clean-up any reports that are more than x days old.

- **Enable watchdog**
  The report generator is a service program which is launched whenever an **eXLerate** project is started. By default this option is enabled which caused the report generator program to be restarted if anything unexpected happens (faulty/buggy printer drivers may cause the report generator to exit unexpectedly). It is strongly advised to keep this option enabled.

- **Watchdog timeout**
  This is the timeout-value in seconds used by the *Enable watchdog* option mentioned above. If the report generator isn't responding for more than x seconds, the Control Center terminates it and restarts it. Situations have been witnessed where a printer driver causes the report generator to block for a long time. In these cases increasing the time-out value may help to prevent **eXLerate** from restarting the report generator, in case the printer driver is simply *taking a long time*.

Terminal Services options

- **Enable Terminal Services Mode**
  This parameter enabled Terminal Services Client Mode. This option allows you to run multiple client instances of **eXLerate** on a single machine using Terminal Services (=Remote Desktop).

- **Run first logon of the first user as..**
  When enabled, runs the first login for of the current user as a regular **eXLerate** session (e.g. as a Server-session). Use this option when you want to run Terminal Services on the same machine that is a duty/standby or a standalone server.

Application control

When an application is started from the Control Center, it is important to acknowledge what exactly happens.

Application startup

At startup in Design or Runtime mode, the following actions take place:

1. A copy is made of the original specified eXLerate project workbook. The copy is created for security reasons. This copy of the original project workbook is created in the Windows temporary directory, e.g. 'C:\Temp'. The actual directory depends on Windows setup.

2. The executable corresponding with the application shortcut is started as a separate Windows process, with the copy of the project workbook as its argument, in which case the copy of the workbook is opened.

3. A special startup sequence takes place, which creates the typical eXLerate environment rather than a standard Microsoft Excel environment. Note that the Excel icon of an eXLerate project is colored red rather than the standard green icon. During this startup sequence, various event messages are logged in the event logger, which is displayed as an active window by the Control Center.

4. When the application is started up normally, the Control Center itself is minimized to the System tray, where it can be re-activated with a single mouse click.

5. When the application has not started properly, the Control Center does probably not disappear into the system tray. In this case, minimize the Control Center manually, and check the application status carefully, or monitor the messages as logged to the event logger carefully for unusual messages. A possible reason can be the usage of additional controls or components, or an improper eXLerate installation process. Check the Trouble shooting Guide at the end of this manual in that case.

Saving an application

6. In order to save changes made to your application, use the Save button at the top or the Ctrl+S shortcut. Before overwriting the application file, eXLerate makes a backup of the original application in the archive directory (e.g. "C:\XLRX\Archive").

Saving an application

7. When the Design or Runtime -button is pressed while an application is already running, it is activated in the specified mode, and not re-

spawned. This prevents running of multiple project instances of the same project, which would cause unpredictable results.

Terminating an application

8. A running application may be also terminated, either from the application itself using <Alt-F4>, using the standard "File", "Exit" menu option in Excel, or via the Control Center, via the control menu of the application shortcut (Right-click an application shortcut, then select the 'Terminate Application' menu option.

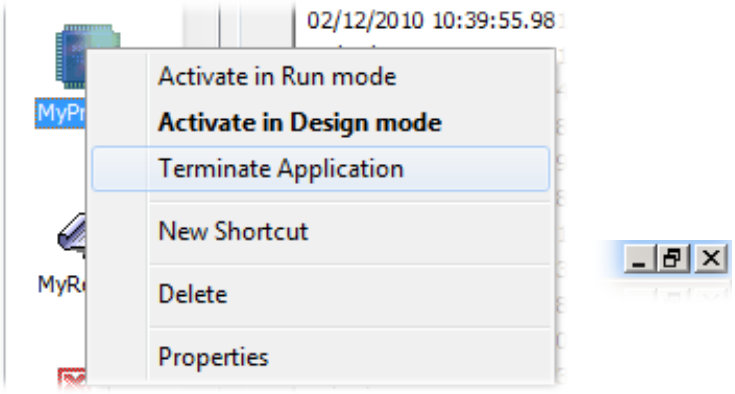


Figure 4-9: Terminate application menu, and Windows close box (right)

When an application is terminated from Excel via <Alt-F4>, the "File", "Exit" menu, the close box, or from the Control Center (See figure above), the application workbook is closed in an orderly manner.

When the application is terminated, **eXLerate** checks if the security level allows for such action. If sufficient, the following warning message is displayed:

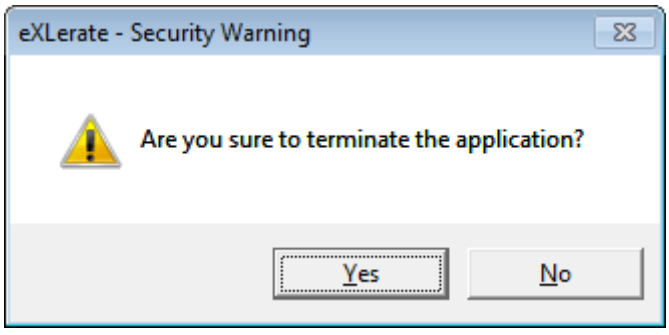


Figure 4-10: Security warning message

If however the application that is to be closed, is not saved yet, the following message pops up instead:

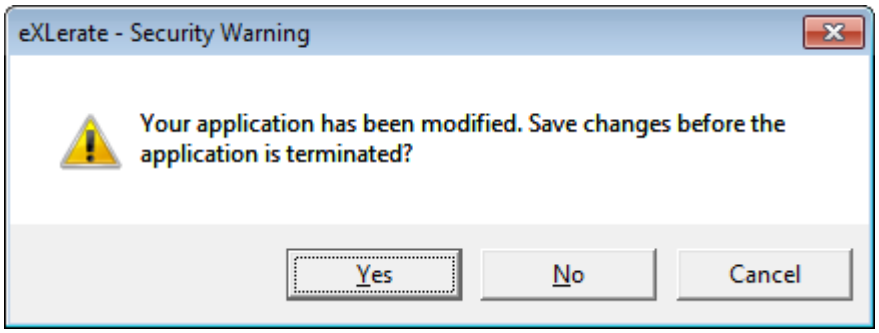


Figure 4-11: Exiting an unsaved application

You may save your project first prior to exiting Excel, or decide after all that the changes need not to be saved.

Finally, the shutdown process may be monitored by the user.

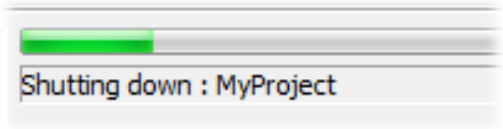


Figure 4-12: Application shutdown from the Control Center

The progress bar of the Control Center indicates the duration of the shutdown process, which may vary based on application size and processor speed.

Typically, the shutdown process should be completed within 10 seconds.

Command line arguments

Although designed as a Windows compliant program with a standard user interface, i.e. a windows dialog, the Control Center may be additionally started from the command line with a number of command line arguments, for automated system startup.

Syntax

The control center application may be started using the following syntax:

```
xlcenter.exe [[-user {Username}] -pswd {Password}][ -exec
{Application}][ -open {Application}] [ -wait {Delay}]
```

with:

Command	Description
user	{Username} is the User name to login with
pswd	{Password} is the password corresponding with {Username}
exec	{Application} is the name of the shortcut or the whole path of the application to start in runtime mode with
open	{Application} is the name of the shortcut or the whole path of the application to start in design mode with
wait	Wait for {Delay} seconds before the exec command actually proceeds

Table 4-1: Command line arguments of the Control Center

This page is intentionally left blank.

Chapter 5 - Application development

Introduction

Now that you have been introduced to an **eXLerate** *application, it's about time to learn more about the details on behalf of application development.*

In this chapter, the details of application components are discussed. These components include, but are not limited to:

- *Project management*
- *Graphical user interface and drawing*
- *The tag database, the beating heart of an application*
- *Real-time data communications*
- *Interval events and associated calculations*
- *Animation Table details*
- *Cell editing*
- *Alarm management*
- *Historical trending*

Besides the components mentioned above, there are a number of tools and wizards in **eXLerate** *that help you with project development. These are the following:*

Wizards

- *Tag & Object wizard*
- *Calculation wizard*
- *Color wizard*
- *Button wizard*
- *Language wizard*

Tools

- *Worksheet organizer Tool*
- *Tag property Tool*
- *Quick-jump Tool*
- *Cell marker Tool*
- *Name editor Tool*

In the 'Advanced Topics Reference' manual, also the wizards and tools are described in detail. The worksheet functions and the VBA API are discussed in the Function Reference. This volume contains also other advanced programming topics.

Development steps

In the next chapters, a complete application will be developed. During this process, various steps are thoroughly discussed, and examples are given of these various steps and components. An application is generated in various steps, which are discussed in the following chapters:

- **Start a new project**
  A new project file is created using an existing project. All steps needed to create a workbook are discussed.

- **Tag database**
  The tag database is discussed and manipulated. Available fields in the tag database are explained to the user, and their usage.

- **Worksheet components**
  Various worksheet components are introduced and discussed to the user, such as animated shapes and real-time values. Various tools are discussed with the user.

- **Data communications**
  Real-time data communications are added to the application, which 'connects' an eXLerate application to the outside world.

- **Interval based calculations**
  Various automatically generated calculations may be added to an application. This is discussed in a dedicated section containing various examples.

- **Shape animations**
  Shapes on display pages are discussed an added to the application. Shapes may be dynamically changed by color, size, position or rotation angle, set blinking, or shown/hid.

- **Menu navigation**
  Menu navigation in eXLerate is managed via a single table, and a template navigation bar, which may be automatically added to an application.

- **Cell editing**
  Cell editing in eXLerate is managed via a single table, which can be manually added to an application.

- **Alarm management**
  Alarm management functionality, including an alarm summary, and an alarm history are added to an application.

- **Trending**
  Real-time and historical trending is added to the application.

- **Report generation**
  Report creation, storage, and generation are explained further in the next

chapter, and are added to the application as well, including pages in HTML format.

Start a new project in eXLerate

When you are about to create your own real-time HMI project, there are a number of steps to consider. This section explains what to do when a new eXLerate project is to be created. These steps include:

- **Creating your own project workbook.**
  You may create your own project workbook. This may be done by creating an entire new workbook in Excel, and by adding required basic components, or by copying an existing project to your own project.

- **Adding a shortcut.**
  Application shortcuts may be automatically, or manually added in eXLerate. Specific security rights should be assigned to a new project as well.

- **Adding/removing components.**
  With an existing project, certain application components, such as a trend, alarm page, or any other component may be added to your application.

Note *During the discussions in this chapter, we assume that you are logged in properly, and that you have sufficient security rights for application engineering*.

Creating your own project workbook

If you want to create your own project, all you have to do is to copy an existing workbook to a new workbook. Remember, a full eXLerate application is nothing more than just a single workbook!

You can also start from scratch using the 'MyTemplate' application. Just double-click the 'MyTemplate.xlrx' file and follow the instructions. When 'MyTemplate' is opened it allows you to create a new project in a specific resolution. Select the appropriate screen resolution and click the 'Create New Application' button.

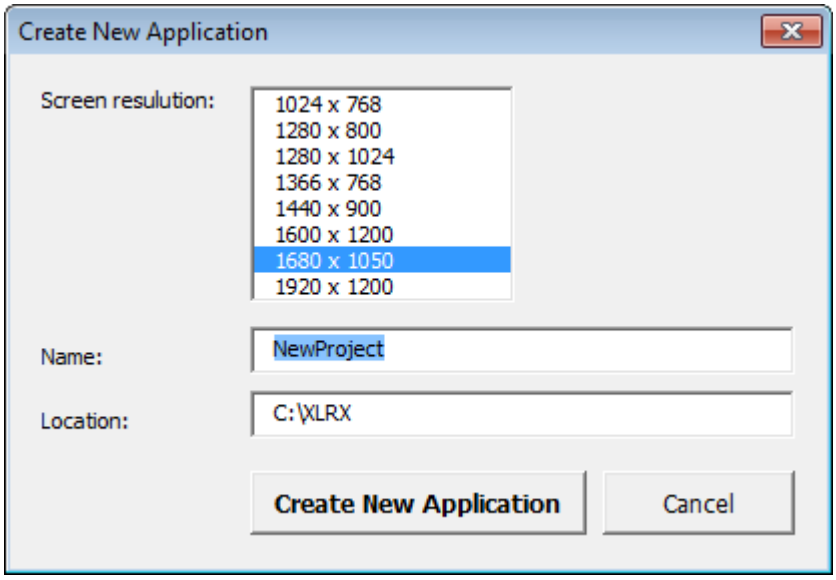
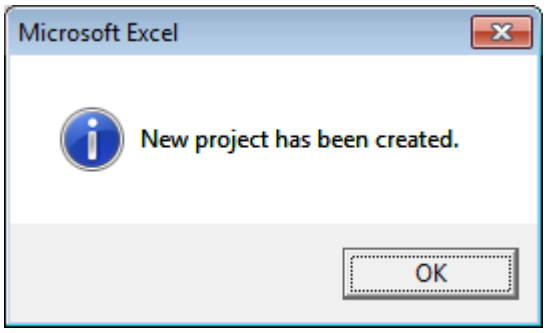


Figure 5-1: Create New Application dialog

The contents of the 'MyTemplate' application is now automatically copied into a new application. If you select 'Cancel' in the dialog above, you may edit the template yourself. This way you can create your own company specific template and use that when creating new applications.

Upon success a new project-file is automatically created and a shortcut is added to the Control Center.



The tag database

In this section details of the tag database are given. You are assumed to have read the tutorial chapters, where an introduction was given on the tag database.

Purpose of the tag database

In **eXLerate**, the tag database manages all **external**, and *most* **internal** variables in a project.

The word 'database' is used as a general classification, i.e. the tag database is not a pure relational database, even though the tag database is arranged as a table, with records (rows) and fields (columns).

An *external* tag is a variable retrieved from an external device, such as a PLC or flow computer, usually via a serial device, such as an RS-232 or RS-485 port. Alternatively, external tags may be sent to the device.

An example of an external tag retrieved from an external device is an analog input from a PLC; an example of a tag, which is sent to a device would be an analog output.

An *internal* tag is a variable internally used in an application, for example a calculated result, which is used in the **eXLerate** application for a report.

There may be many 'internal tags' defined in **eXLerate**, even *outside* the tag database, because in fact each regular worksheet cell in all worksheets in Excel may be considered as internal tags, traditionally spoken*.

A tag in the tag database of **eXLerate** has various properties:

- Important properties of a tag may be referred to with a logical name, such as 'xTR1TA.Value', or 'xTR1TA.Units', in addition to a standard Excel name such as 'xTagDB!F245'. You need the *Tag & Object wizard* to create such object names for your tag database.
- A tag may be selected for real-time and historical trending.
- A tag may be defined as 'retentive', in which case its current value is automatically and permanently stored in the system registry. At system startup, its current value is automatically retrieved from the registry.
- A tag may have additional calculations defined, which are automatically created and maintained by **eXLerate**, such as moving averages, or latched values.
- A tag may retrieve its value from an external device. If this is the case, certain additional parameters, as its PLC address should be defined.

---

* *Referring to other SCADA/HMI software suppliers*

- The current value of a tag may be simulated, so your application may be thoroughly tested.
- A tag may have alarm properties assigned, so **eXLerate** automatically monitors the value for various high/low/state alarms.
- Additional project and engineering information may be added to a tag database. You may add your own fields (columns) to suit your own needs and preferences, as long as **eXLerate** is able to locate the required fields.

NOTES: *Although you may add fields for your own requirements, predefined column names may not be altered. This is because **eXLerate** uses these columns internally.*
The number of tags that you may have in your application depends on your license.

Although you may add variables in other worksheets, it is highly recommended to define tags in the tag database because of application support as mentioned above.
During real-time updates, the tag database worksheet is protected.

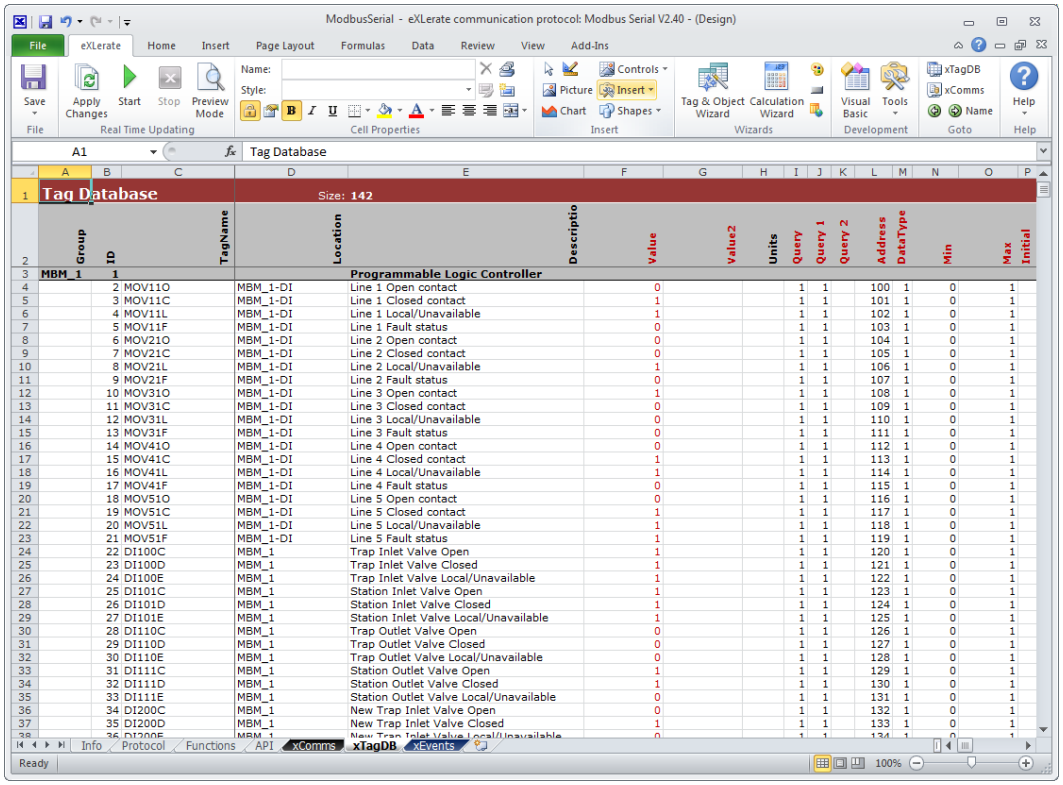


Figure 5-2: The tag database in eXLerate

Fields may be grouped together. Although not essential, in the project samples, the grouped fields have a distinct color.

Various fields of the tag database are used to define certain automatically generated calculations ('P_XXXX' fields, e.g. 'P_Min'), object names ('TagName', 'Alias', 'Value', 'Units'), or automatically generated alarms ('Salarm', 'Lalarm', 'LLAlarm', 'Halarm', 'HHAlarm', 'Priority', 'Delay', 'AlarmDesc', and 'AlarmGroup').

After these fields have been defined or modified in the tag database, the **eXLerate** *Tag & Object wizard* must be used first to automatically generate the required calculations for you. You might want to read more about the Tag & Object wizard in the '*Advanced Topics Reference*', chapter '*Wizards and Tools*'.

The following columns are defined in **eXLerate**:

General fields

- **Group**
  Tags are arranged in groups. You will see at the left-hand side of a tag database a group symbol (vertical line around the group, with a "+", or "-" at the end of a group to open, or close a group. Groups may be created multi-leveled in Excel, but only single level groups are supported in **eXLerate**. When this column is used, you define a group rather than a normal tag. Columns to be filled in are Tagname, and location. All other columns remain empty. An example of a group is: "PLC", or "Line 12".

- **ID**
  This is a sequence number, starting from 1 for the first tag, until the last tag. Group names also have an ID. The Tag & Object wizard automatically generates an ID, when object names are generated.

- **TagName**
  This is the name under which the tag will be known for the application programmer: a tag name defines the internally used object names. The tag name may be looked at as a 'key'-field in a relational database. A tag name follows the same restrictions as standard names in Excel. An example of a TagName is: 'PT1_1', or 'XFC-01'. Since tag names are used for object names, spaces are not allowed in a tag name.

- **Alias**
  The alias name may contain a different name for the tag, to be used in display pages and reports, pen selections for trending and for alarming. Usually, alias names are longer than a tag name, and may contain characters that are not allowed for tag names, e.g. a valid alias name may be: '##$%^'. When an alias name is not defined, the tag name is automatically used.

- **Location**
  The location of a tag may be optionally defined. The location is used in

alarming. There may be a difference between the location, and the group to which a tag belongs.

- **Description**
  The description of a tag is used throughout the application, for example in alarming, and for trend pen selection.

- **Value, Value2,…,Value5**
  The current value(s) of a tag, as last retrieved from (an) external device(s), or as last simulated, is stored in this field. The current value of a tag may be referred to in the application with 'x'{TagName}.Value, e.g. 'xTR1TA.Value', or 'xPT1.Value3'. If a tag has no external data source, a formula may be entered as well. If you have extensive calculations in your application, you may want to add a special calculation sheet rather than to add expressions or worksheet functions right here in the tag database. Value2…Value5 may be used for multiple tags, where a single tag database entry is used for data from various parallel running devices, for example dual redundant flow computers.

- **Units**
  The engineering units as used in the application for this tag, e.g. 'bar', 'm/s', 'gallon', or 'feet'. When entered, **eXLerate** creates an object name for this field (e.g. 'xTR1TA.Units').

- **TrendNorm**
  The trend norm is a field that defines whether the tag should be recorded for trending. When this field is left blank, the tag value is not trended at all. When filled in, it is used as a norm for trending. If the expression: $|CurrentValue - Last\,\mathrm{Re}\,cordedValue| \geq TrendNorm$ is true, then a new sample is recorded. When TrendNorm = 0, all tag values are recorded.

- **Format**
  This optional field can be used to display the tag-value in a specific format. The format is in the same style as Excel formats but it should be preceded by a single-quote character otherwise Excel recognizes it as a value rather than a string:

- **Retentive**
  When the value of this field is set to '1', the current value will be permanently stored in the system registry. At system startup, the value will be automatically reloaded.

- **Type**
  The type of this tag, for example 'AO' (analog output), 'AI' (analog input), 'DI' (digital input), or 'DO' (digital output). You may add your own

abbreviations as well. Only used as engineering aid, not currently used by **eXLerate** internally.

- **User-address**
  The user address is an optional user-definable address, for example an internal PLC address. May be left blank. The actual address as referred to in data communications is stored in column: 'Address'. Only used as engineering aid, not currently used by **eXLerate** internally.

- **HTML**
  This field is a user-defined field, used for the built-in HTML page support for **eXLerate**. This field may contain a formula, which is used for HTML page generation. Support for HTML pages is explained in chapter 13 in this manual. An object name x{Tag}.HTML will be generated for each non-empty field by the Tag & Object wizard.

Interval related fields

In the tag database, you may specify if an automatic calculation should be periodically calculated. For example, an hourly average may be automatically calculated on behalf of reporting. Interval related fields start with a 'P_', for periodical calculations, and are followed by the period as defined in the event table, for example: 'hour'. A field: 'P_hour' is then used to define the parameters for the interval based calculations.

To fully understand all fields you might want to read the section on interval events and period calculations first, in section '*Interval*' on page 7-149 onwards or skip this part if you are not interested in automatically generated period calculations.

- **P_xxxx fields**
  Various fields defining automatically calculated interval values for the tag. For example, 'P_hour' and 'P_day' columns refer to hourly and daily period columns, but all intervals are user defined and may therefore be configured differently. You may enter one of the following tokens in the field: '**L**' (creates a latched (= periodically clocked) value for the tag value of the current period), '**M**' (creates a Moving average for the tag value of the current period), '**W**' (creates a Weighted average for the tag of the current period).

- **WeighFactor**
  Used in combination with weighted averages. The value specified here determines the weight for each value to be averaged into the weighted average. It should be an accumulative value, for example a total flow. **eXLerate** calculates the difference between two interval periods to determine the weigh factor to be used for this tag. An example of a weighted average is a flow weighted average. The field should contain a reference, not a value, e.g. 'xFT1.Value' is a correct reference, but '=xFT1.Value' is not (because it is a value).

Communication related fields

The following fields are used in conjunction with the Protocol Table (used to define the in-use communication protocols) and the Query Table (which defines the queries used for each protocol. A query is a data message containing a read or write request with 1 or more values from an external device.

To learn more about communications with **eXLerate**, you might want to check '*Data communications*' from page 6-127 onwards first.

- **Query**
  This is the query number in which the tag retrieves its data. It should be an index to the Query Table, starting from 1. Various protocols support multiple queries for a single item, where queries are comma-separated.

- **Address**
  Each numerical value might have an address corresponding with a register in an external device, such as a PLC. At this location, the address relative to the message query should be defined. The actual number to be filled in depends on the query definition and communication protocol that corresponds with the tag. In Modbus, a valid address would be: '1500:12' which specifies bit #12 in a register with address 1500. For the HART protocol, a valid address example would be: 'R:4', specifying a number at offset 4 in the response message.

- **DataType**
  Each tag in **eXLerate** has a distinct data type. Most tags are numerical, but in **eXLerate**, also text strings, single bits, or other data types are defined. The following data types are currently supported:

Datatype	Value	Description
xBit	1	Coil (in 16 bit word)
xByte	2	8 bit unsigned integer
xShort	3	16-bit signed integer (C WIN32 short)
xWord	4	16 bit unsigned integer
xUInt24	5	24 bit unsigned integer
xLong	6	32-bit signed integer (C WIN32 long)
xDWord	7	32 bit double word (unsigned integer)
xFloat	16	32 bit single precision IEEE floating point, 'standard' byte order 4321
xRevFloat	17	32 bit single precision IEEE floating point, reversed byte

Datatype	Value	Description
		order 2143
xDouble	18	64 bit double precision IEEE floating point
xShortFloat	19	16 bit integer to scaled floating point
xIntelFloat	20	32 bit single precision floating point, byte order 1234
xWordFloat	21	32 bit integer to scaled floating point
xRevDouble	22	64 bit double precision IEEE floating point, byte order 21436587
xBCD	32	32 bit BCD value, with 8 nibbles of 4 bits each, each nibble coded 0..9
xTimeDate	33	64 bit time date string, in 8 bytes, as follows: <YY><MM><DD><hh><mm><ss><xxyy> YY: Year (0-99), MM: Month (1-12), DD: Day(1-31), hh: Hour (0-23), mm: Minute (0-59), ss: Seconds (0-59), xxyy: User added value, 0..65535
xTimeStamp	34	64 bit time date string, in 8 bytes, as follows: <1><YY><MM><DD><1><hh><mm><ss> YY: Year (0-99), MM: Month (1-12), DD: Day(1-31), hh: Hour (0-23), mm: Minute (0-59), ss: Seconds (0-59), xxyy: User added value, 0..65535
xAdcFloat	37	12 bits Analog input 0-4095 to float, direct value from a ADC
x10kFloat	38	12..16 bits Analog input 0-10000 to float, converted value from a PLC
xBitInQWord	39	Single bit in a 64-bits word, with its upper 32 MSB bits reset
xLowQWord	40	32-bits word in a 64-bits quadruple word, with its lower LSB bits reset
xString6	64	6 byte, 8 character packed ASCII string (HART)
xString12	65	12 byte, 16 character packed ASCII string (HART)
xString24	66	24 byte, 32 character packed ASCII string (HART)
xString10	67	10 character string.
xString80	68	80 character string.

Datatype	Value	Description
xString	69	A null-terminated string.
xString8	70	An 8-byte character string, used in some devices.
xString16	71	A 16 byte character string, used in some devices.
xVariant	80	An OLE originated Variant data type. Used in the OPC drivers.

Table 5-1: Supported data types

The data type is available as a predefined constant in eXLerate, so to specify a 32-bit floating point variable, specify '=xFloat' in this field rather than '16'.

- **Min**
  Minimum value of the tag, when the input is simulated. Also used for minimum scale value in trending, and for internal value scaling in case the xShortFloat, xAdcFloat, x10KFloat, or xWordFloat data type is used.

- **Max**
  Maximum value of the tag, when the input is simulated. Also used for maximum scale value in trending, and for internal value scaling in case the xShortFloat, xAdcFloat, x10KFloat, or xWordFloat data type is used.

- **Initial**
  An initial value of the tag at system startup, i.e. the current value of the tag internally, until actual communications have been started. Optional field.

- **ScaleMin**
  Minimum scaling value for a tag. Optional field which is used for additional scaling of a value, for all data types.

- **ScaleMax**
  Maximum scaling value for a tag. Optional field which is used for additional scaling of a value, for all data types.

Explanation
All values coming into the tag database from external devices may be additionally scaled with user defined constants before the value is stored in the tag database. In addition, values in the tag database may be scaled with additional constants before being sent to external devices.

For *inbound* data, i.e. data coming from the driver into the tag database, the following equation applies:

$$tag.value = ScaleMin + driver.value * (ScaleMax - ScaleMin)$$

Equation 1: 'inbound' data scaling

For *outbound* data, i.e. data to be sent to external device, the following equation applies:

$$driver.value = \frac{(tag.value - ScaleMin)}{ScaleMax - ScaleMin}$$

Equation 2: 'outbound' data scaling

For example, when *ScaleMax* is set to 1000, and *ScaleMin* is set to 0, all values are multiplied with a constant factor of 1000. In your application, you may use the tag value, and automatically, when the tag value is sent back to the external device, it is divided by a 1000.
When no additional scaling is required, these columns may be left empty.

- **Update**
  This last field related to data-communications is used in **eXLerate** to enter a worksheet function with which data is written from the worksheet into a connected external device. In **eXLerate**, the `exUpdateEx(..)`, `exUpdateVarEx(..)` and `exUpdateStrEx(..)` worksheet functions are used for updates from your tag database worksheet to an external device. When no updates are required, because the tag is read-only with respect to the device, this field may be left empty. See one of the sample projects of how to update a device using the `exUpdateXXXEx`(..) worksheet functions.

- **OPCMode**
  This field is used to determine whether the tag values should be available in the OPC Server. Valid values are 'r' and 'w', which are described in detail in chapter Chapter 6 - Data Communications.

Alarming related fields

The following fields are related to the built-in alarm manager of **eXLerate**. When no alarms are to be generated for a tag, the fields below should be left empty.

The alarm manager is explained in Advanced Reference Manual.

The *Tag & Object wizard* should be run after changes made to the fields below to effectively create the alarm entries for the alarm manager. The alarm manager is the run-time part of alarms in **eXLerate**.

The following fields are available in the tag database for alarming:

- **AlarmDesc**
  Alarm description. When left empty, the general '*Description*' field is used as alarm descriptor in the alarm manager.

- **AlarmGroup**
  Alarm *group* to which the tag belongs. Alarms may be grouped into hierarchical, unique groups, much like a directory tree structure. An operator may acknowledge a group rather than an individual alarm.

- **Priority**
  Each alarm has an assigned *priority*, ranging from 1 to a user defined maximum value. Although the priority is assigned, displayed and printed, the priority of an alarm has just an informational character.

- **SAlarm**
  This field contains the *state alarm level*, when filled in. When not filled in, no state alarm will be defined for this tag. A state alarm is generated when the current value of the tag is equal to this 'Salarm' field. A state alarm is usually associated with digital signals, such as a valve.

- **LLAlarm**
  The 'LLAlarm' field is used to define a *low-low alarm level*. When this field is defined, the Tag & Object wizard creates a low-low alarm entry in the alarm list for this field. An alarm event is generated by the alarm manager when the current tag value drops below this level.

- **LAlarm**
  The 'LAlarm' field is used to define a *low alarm level*. When this field is defined, the Tag & Object wizard creates a low alarm entry in the alarm list for this field. The alarm manager generates a low alarm event if the current tag value drops below this level. A 'LAlarm' may be used as a warning limit, while the 'LLAlarm' field may be used as a critical error limit.

- **HAlarm**
  The 'HAlarm' field is used to define a *high alarm level*. When this field is defined, the Tag & Object wizard creates a high alarm entry in the alarm list for this field. The alarm manager generates a high alarm event when

the current tag value exceeds this high level. A 'HAlarm' may be used as a warning limit, while the 'HHAlarm' field may be used as a critical error limit.

- **HHAlarm**
  The 'HHAlarm' field is used to define a *high-high alarm level*. When this field is defined, the Tag & Object wizard creates a high-high alarm entry in the alarm list for this field. The alarm manager generates a high-high alarm event when the current tag value exceeds this high-high level.

- **Deadband**
  The 'Deadband' field is used to set a so-called *dead-band* value for a high-high, high, low, or low-low limit alarm. This parameter, which may be entered in engineering units for each tag, is used to suppress jittering alarms that are caused by the fact that the process value gets close to an alarm limit value. This column is optional, and inserted in the tag database when required. When omitted, a value of '0' effectively disables the dead-band mechanism.

- **Delay**
  The delay field is used to define a delayed alarm. When defined, an alarm should exist for this period in time, in seconds, before the alarm manager actually generates an alarm event. This delay may be used to prevent the generation of rapidly changing alarms.

Worksheets as display pages

Now that you have been familiarized in the previous section with drawing of lines, shapes, bitmaps and all other drawing components in Excel, it is time to start discussing how to create a dynamically changing display page using Excel worksheets and eXLerate.

At first some general considerations are given.

Worksheet components

In what SCADA/HMI software packages often refer to as 'display pages' are in eXLerate simply plain Microsoft Excel worksheets. Although you will quickly get familiarized in using worksheets for display pages, this simple fact seems unbelievable for newcomers.

"*Can I build serious graphical user interfaces using Excel worksheets*?" is a question often raised.

Yes, you can! Beautifully, easily, and with many powerful components!

Although this manual is not meant as a general learning book on Microsoft Excel, you will learn all about building display pages in this section.

A worksheet in Excel may contain:

- **Worksheet cells**
  A worksheet cell may contain text, numbers, or worksheet functions. Standard Excel worksheet functions may be used, including your own written VBA functions, and array functions. Worksheet cells may be formatted utilizing many Excel cell properties. In addition, conditional formatting may be used in a worksheet cell. Conditional formatting allows for animated cells. User add-in library functions may be used as well, for example a math library for specific calculations.

- **Charts**
  A chart may consist of any of the more-than-many available standard charts in Excel. A chart is used by eXLerate for display of real-time and historical trending, but may be additionally used for display of any data, for example a temperature profile in an oven recipe.

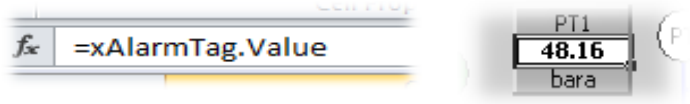
- **Shape objects**
  Besides cell data and charts, worksheets may contain the shape objects as discussed in the previous section. A shape object is a Microsoft Office item, which is used for drawing of lines, basic figures like rectangles, circles, ellipses, arrows, flowchart symbols, call-outs and much more. Shape objects may be animated by eXLerate, as we have seen in the tutorial chapter and the previous section.

- **Miscellaneous objects**
  In Excel, bitmaps (for example your company logo in a report or display),

or any other ActiveX component may be inserted as well. Although those external objects are sometimes impressive, and may indeed be added, an application developer should be aware of the fact that system instability is often caused by inferior components, or components not designed for real-time systems. The best approach would be to first develop & test your application with standard **eXLerate** components (which are stable), and then, in a later stage, add such additional components. "If it works, it works"!

Worksheet cells

A worksheet cell may contain text, numbers, or worksheet functions. In order to show a number from another worksheet, for example the tag database, all you have to do is refer to a cell in that worksheet with the familiar '=' syntax in Excel, e.g. '=xTR1TA.Value'.



Standard Excel worksheet functions may be used, including your own written VBA functions, and array functions. User add-in library functions may be used as well, for example a math library for specific calculations.

To insert a worksheet function, click on the 'Paste Function' button on the **eXLerate** toolbar. You may insert functions. For each function argument, Excel offers help on entering the correct value, as in the dialog below.

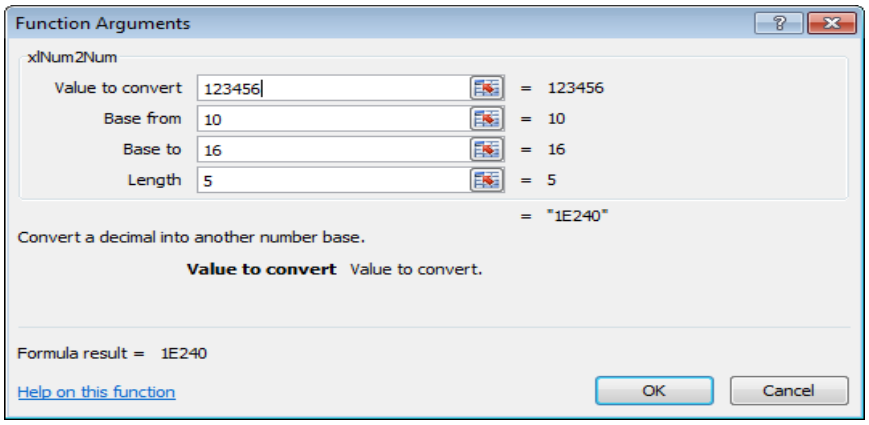


Figure 5-3: Entering a worksheet function into a cell

Note *If you feel uncomfortable while using such numerical expressions or functions in worksheet cells, it is now really time to start looking for a good Excel learning book…*

Named items rather than plain cell references

eXLerate relies on named ranges in Excel rather than on direct cell references, because working in a structured way calls for such approach.

For example, to calculate the average value of two pressures, it is better to use:

```
=(xTR1TA.Value + xPT2.Value)/2
```

then using:

```
=(xTagDB!G34 + xTagDB!G35)/2
```

Names can be easily created using the 'Name' controls in the eXLerate ribbon:

The 'Name' edit-box automatically displays the name defined on the current cell or range. To create or rename a name, just type a new name end press <Enter>.

To check which names are available in your application, you can use the Excel 'Name Manager', as shown below:

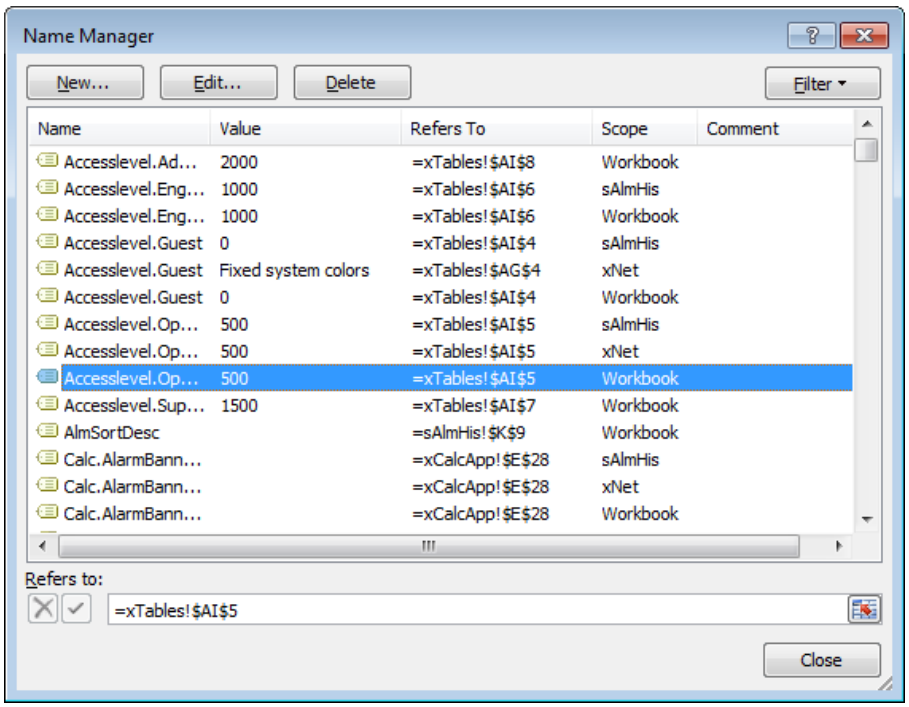


Figure 5-4: Name Manager of Excel

In this chapter, you will learn how to use object names. You may use the tag database as the main data source in your application.

Using the **eXLerate** *Tag & Object wizard*, you may create your own object names from values in the tag database. The Tag & Object wizard is discussed separately in the 'Advanced Topics Reference' of the **eXLerate** manuals.

Using the **eXLerate** *Calculation Wizard*, you may create special calculation worksheets containing *calculation tags*. These object names are available to allow you to create structured applications. Creating a structured application is highly recommended, if you want to spend your time as efficient as possible. An advantage of using separate calculation sheets in your application is that you will be able to separate data communications tags, and derived calculations.

Tag database items in display pages

In many cases, you might want to display data directly from the tag database.

Remember, the tag database is a background worksheet in the application in which all of the external I/O points are defined.

To display items from the tag database into your display pages, you can refer to the name of a tag, e.g. 'xTR1TA.Value' rather than to a cell reference, such as 'xTagDB!$G$35'.

The precise syntax for referring to items from the tag database in a worksheet cell is:

'x'{TagName}.{UsedFieldName}

with:

'x'	Standard prefix in **eXLerate** for automatically generated object names.
{TagName}	Name of the tag, as defined in the column: 'TagName' in the tag database
{UsedFieldName}	A used field of the tag in the tag database. A used field is a non-empty field. The fieldname is the name of the corresponding column. The following field names are supported using the above syntax: 'Value', 'Units', 'Calculated', 'SAlarm', 'Lalarm', 'LLAlarm', 'Halarm', 'HHAlarm', 'AlmDead-band', 'AlmRaised', 'AlmOptions', 'AlmDelay', 'Criteria', and 'Alias'. In section: '*The tag database*' on page 5-107 onwards of this chapter all details on using the tag database is given.

After an expression has been entered, the current value of the referred cell is presented. You might want to press <Ctrl-R>, 'Recalculate Workbook' to recalculate you project workbook and update the inserted number.

Note *Calculation **must** be set at manual in eXLerate, for performance reasons.*

Cell formatting

When a number, worksheet function, array function or regular expression has been entered, it may be formatted using <Ctrl-1>, or by right clicking a cell and choosing 'Format Cells…' from the context-menu. The following dialog appears:

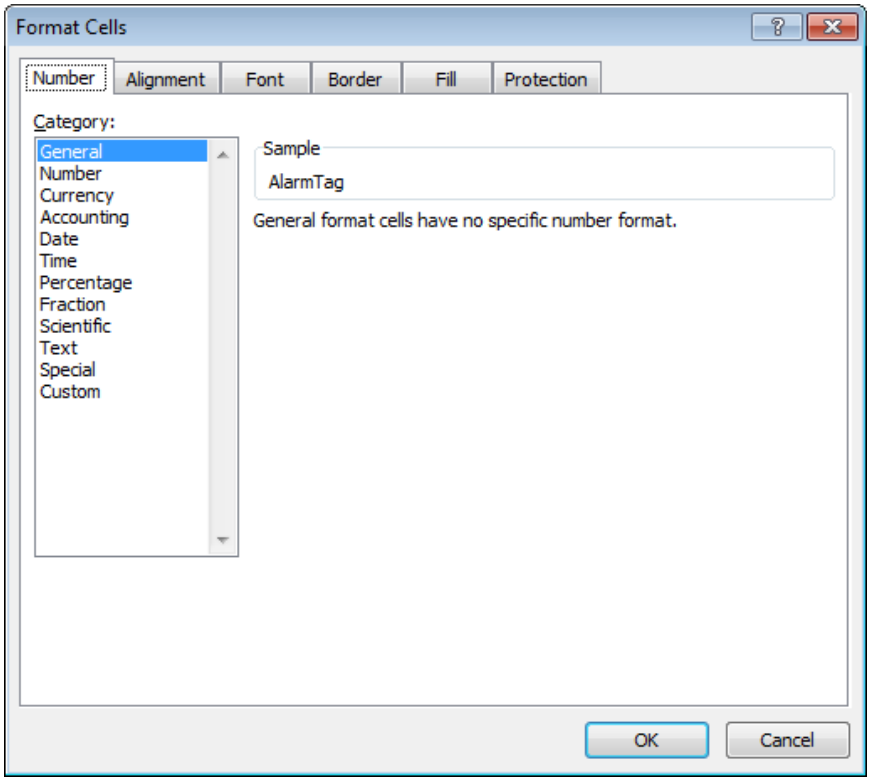


Figure 5-5: Formatting a cell

Instead of formatting each cell independently, you may also use a predefined style in Excel. In eXLerate, support is built-in for using these styles. For now, play around and format your cell for your taste and specific requirements.

Cells can be dynamically formatted using *range animations*. Range animations are similar to shape animations, where a color can be given to a range of cells using worksheet functions exRangeColor(…), exRangeBlink(…) etc. Range animations are virtually unlimited, because the number of formats is not limited to the three formats as with conditional formatting.

Charts in display pages

One of the most impressive features of using Excel in a real-time HMI environment is the usage of charts. Even the real-time and historical trending module in eXLerate uses Excel charts for display of trend data.

In the examples below, some standard graphs are presented, to give you a quick impression of possibilities.

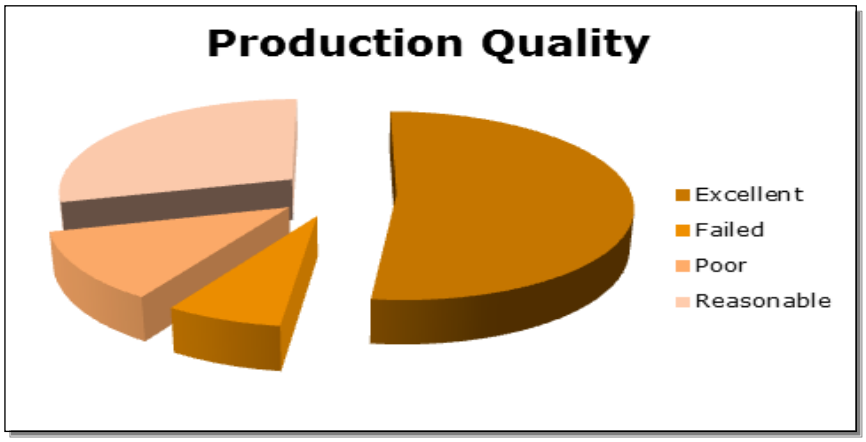


Figure 5-6: Pie-chart type example

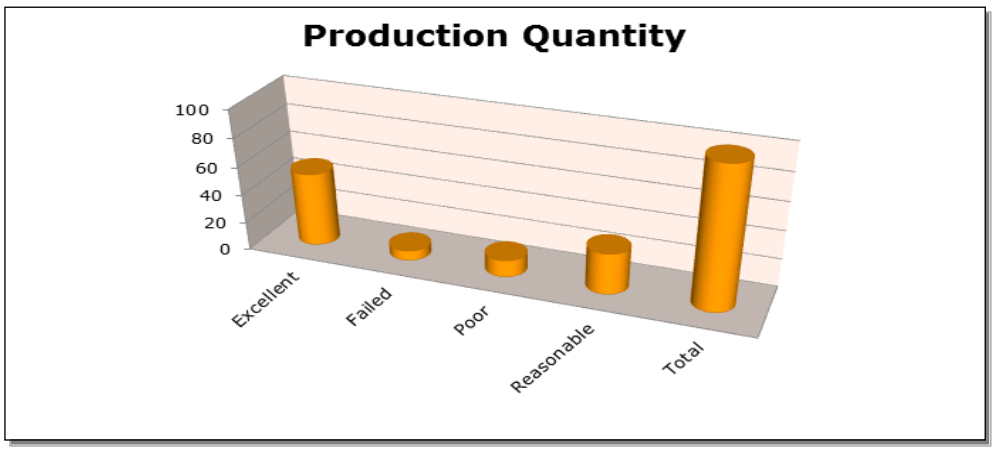


Figure 5-7: Columns in a display page example

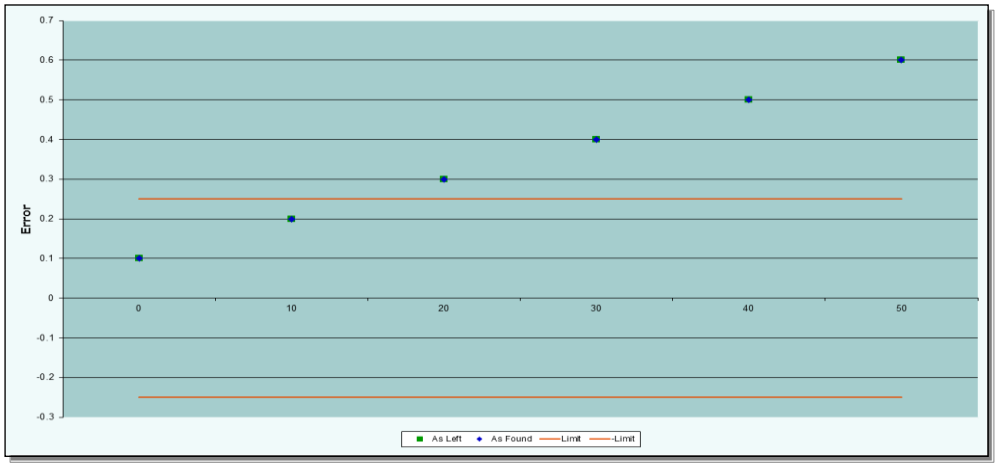


Figure 5-8: Scatter-chart type example

Please note that these charts are dynamic too! When the source data changes (and this will happen if related to an external I/O point, such as a pressure or temperature), the chart will change as well.

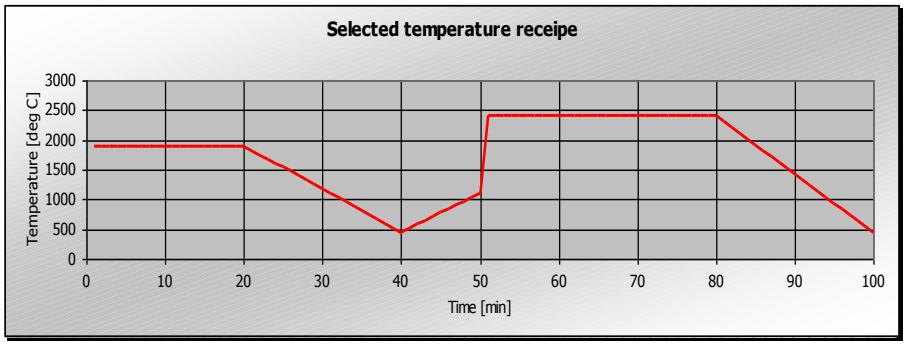


Figure 5-9: Temperature profile example

All built-in charts of Excel may be used in your display pages.

Note *If you do not feel comfortable in using Excel charts, and you want to nevertheless use a chart in your application, you might want to pick up one of the many excellent titles on this subject in your bookstore*.

When using X-Y charts as in some of the examples above, make sure to minimize the number of data-points to present in your graph, because the larger the number of data-points, the more resources in your computer will be used.

Shapes in display pages

In the previous sections you were introduced with Excel shapes. Shapes may be animated by **eXLerate**, as you have seen in the tutorial chapter as well. See the *Animation Table* for details on animating shapes.

Important to realize is that **eXLerate** uses the name of a shape for animations in the Animation Table. These names have a **global** scope in the application.

A global scope means that all occurrences of all shapes in all display pages with this global name are animated identically.

You may use <Ctrl-C>, <Ctrl-V> to copy and paste shapes between display pages, and change the size and perhaps rotation angle of each object individually. You may even create two totally different shapes in two display pages. As long as their name is identical, the shapes may be animated using the same definition.

Example.

You may create a valve symbol on one display, and a much bigger valve symbol in another page.

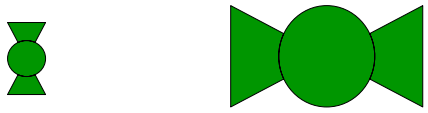


Figure 5-10: Two instances of the same "valve_11" object on two displays

If you give them both the same name, e.g. “valve_11”, there need to be only one entry in the Animation Table to animate both different objects. In the example, it would not be smart to animate the position, size or rotation angle of these two valve objects, since it would modify the objects accordingly; you would want to change the colors only (green on open/ red on close).

This page is intentionally left blank.

Chapter 6 - Data communications

Introduction

The purpose of real-time data communications in eXLerate is to be able to exchange information from one or more external devices, such as a Process Logic Controller, with the tag database of the computer running eXLerate.

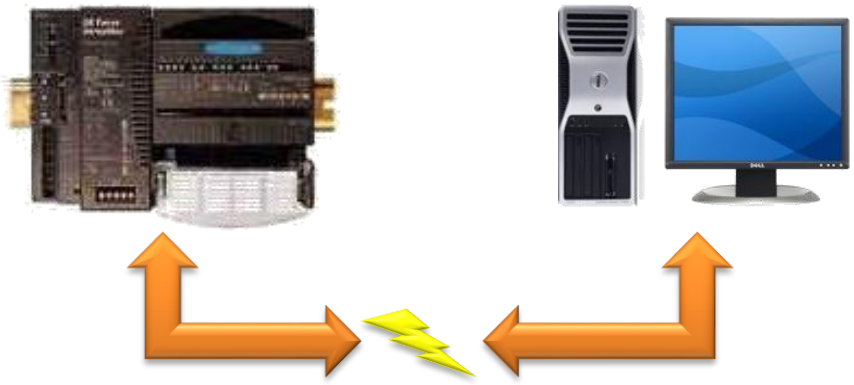


Figure 6-1: Bi-directional data communication between a device and a PC

Process or configuration data **from** the device may be retrieved on an interval basis, under programmatic control, at manual request, or in a combination of these methods.

Data may be sent **to** an external device using an event-driven update technique, or alternatively on an interval, programmatically, manually, or a combination of these methods.

Exchange of information takes place in an agreed format for transmitting data between two devices, a *protocol*, which is usually *query* based. A query in this context is a data message from a computer to a client in a master/client configuration utilizing the protocol to request for information.

Many different protocols and worse, protocol *variations* of many hardware and software manufacturers have been developed over the years, each with certain advantages, some drawbacks, and implemented on numerous devices.

Usually, more than 1 data-point is transmitted in a single query. A single data-point corresponds with a single tag value in eXLerate.

Most protocols are based on serial ports utilizing the RS-232 / RS-485 / RS-422 interface standard, or on the Ethernet standard but can be based on other hardware as well, such as 4-20mA (HART), or in fact any other suitable hardware interface.

Multi-drop or point-to-point communications

There are basically two different approaches in the network architecture to communicate with external serial devices:

- **Multi-drop**
  In a multi-drop system, most or all devices are connected to the supervisory computer using a single communication line. All of the devices have a unique address, and each device is polled by **eXLerate** for data sequentially.

- **Point-to-point**
  In a point-to-point setup, most devices are connected to the supervisory computer using multiple communication lines, where each device has a dedicated communications line. The devices may have identical addresses, and each device is polled by **eXLerate** for data in parallel.

A typical multi-drop RS-485 system may look as follows:

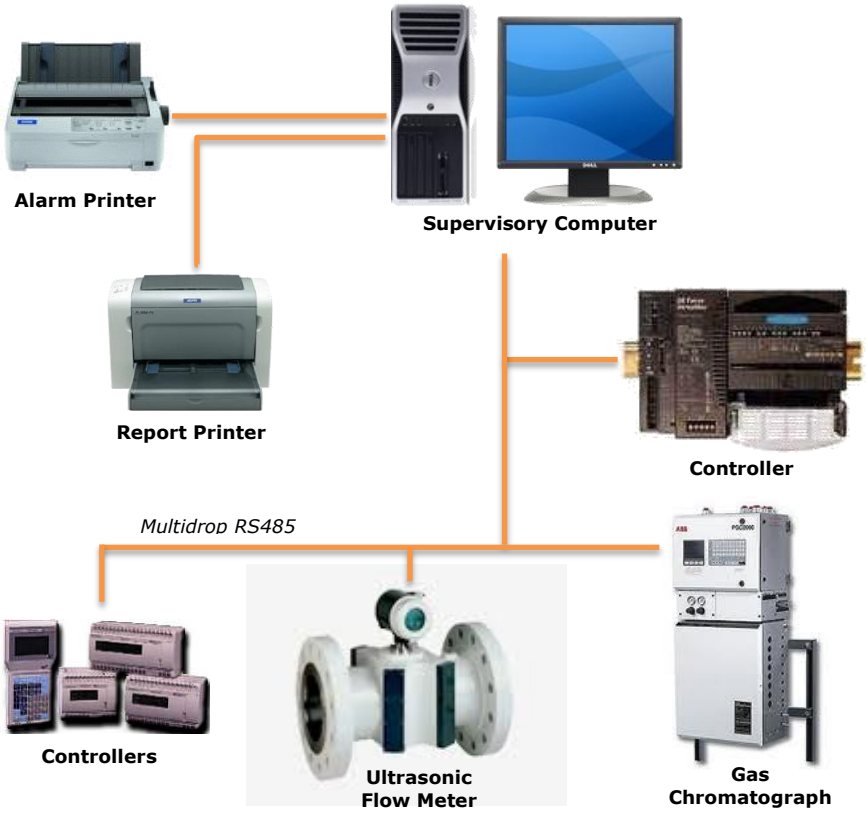


Figure 6-2: Example of a multi-drop system

eXLerate supports multi-dropping for its serial communication drivers. The advantage of multi-dropped communications is the fact that hardware requirements are simple: there is only one single communication path required, over which all of the data communications is handled. The drawback however is slower throughput of data, since only single communication port must sequentially poll all connected devices.
Another drawback is the fact that in a multi-drop system, a failing device is able to seriously delay or even stop communications of all devices.

A typical point-to-point architecture (based on both RS232 and Ethernet) looks as follows:

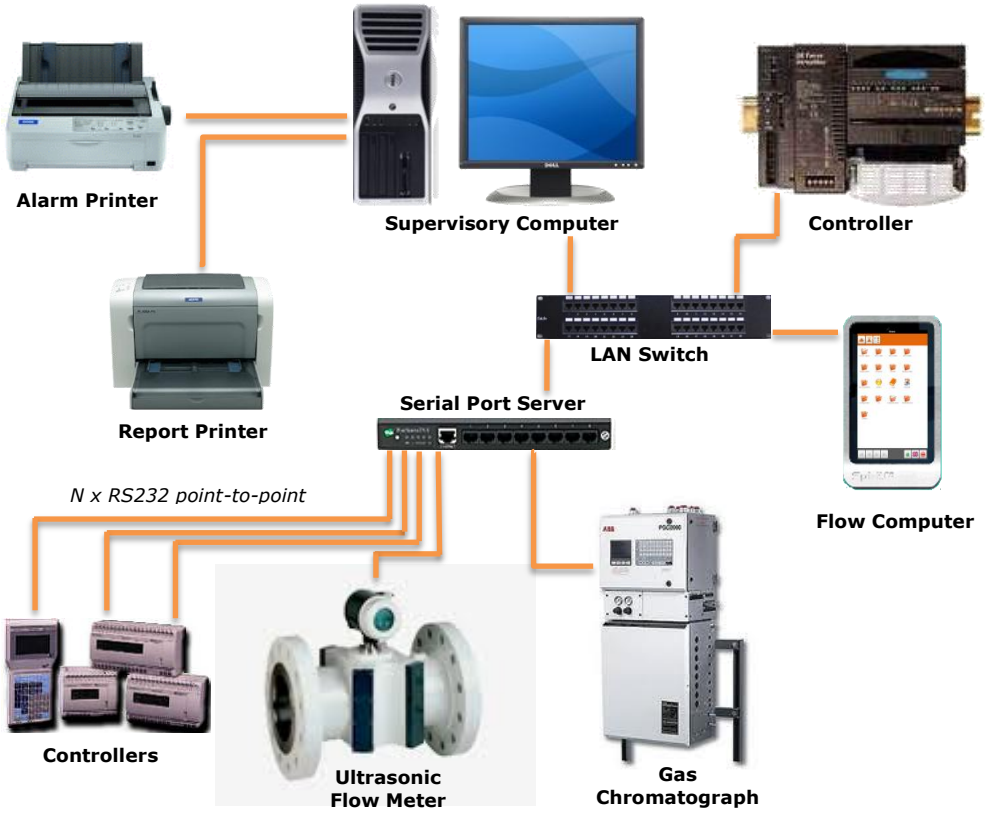


Figure 6-3: Point-to-point communications

Each of the connected devices has a dedicated serial port with which the connection with the supervisory computer is realized. The serial PortServer is equipped with as many serial ports as there are external devices available, and an Ethernet connection with the supervisory computer.

The advantage is much higher data throughput, because all of the devices are polled for data in parallel, at a higher hardware cost.

Also combinations of multi-drop and point-to-point connections are possible, since **eXLerate** fully supports both architectures.

eXLerate also supports Ethernet based architectures, in which case peer-to-peer communications are achieved using an Ethernet LAN (intranet), or even the Internet to connect devices in a network. **eXLerate** supports Modbus/TCP for this purpose.

Simplified data-model

Although there are many differences in data communications caused by the variation of implemented protocols, most communication protocols in **eXLerate** have a multiple query-based structure, i.e. each of the *n* protocols in a project has *m* data queries (a 'query' in this manual is also referred to as a 'poll-block', 'message', or 'frame', asking for data, or sending data) in which *z* data-points, or more specific, tag values are enclosed.

Example: In a system there may be 5 Modbus devices defined, each configured with 4 queries. Each query is a read-request asking for 8 register values.

Communication protocols are running asynchronously in the computer, and each protocol is able to serve *m* data queries, from which *z* tag values retrieve or send new data at each cyclic data-poll. This n x m x z model is presented in the following figure:

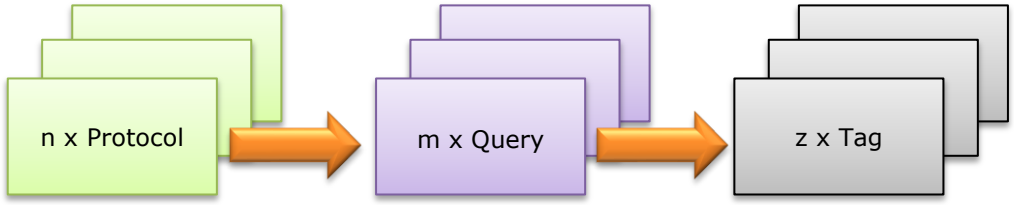


Figure 6-4: Data model of eXLerate communications

Protocols utilize specific hardware, such as serial ports, or Ethernet interface cards to connect a server with its clients. Each protocol in **eXLerate** is running in an independent programs execution thread, and processes its associated queries sequentially.

A query defines which data items are transferred, the direction of the dataflow (from client to server or vice-versa), and how this transfer takes place: periodically, event-based, programmatically, manually, or a combination of these methods. Queries may only ask for data (a read-only query), may only write data (a write-only query), or may have combined read/write commands specified.

It is the task of **eXLerate** to allow for both flexible as well fast data throughput using this model without sacrificing ease and maintainability of its configuration.

In **eXLerate**, the three involved objects (protocol, query, and tag) are implemented with just three simple worksheet tables: the Protocol Table, the Query Table, and the tag database, containing the retrieved real-time data.

Data updates from external devices

A query has several properties that are defined in the *Query Table*. For example, the interval time between two consecutive read-polls, the number of retries, or the read command code are entered in the Query Table, as well as a number of options defining exactly how data is retrieved from external devices.

Data may be read:

- Cyclic, at a user defined interval, for example every second
- Through a subroutine or macro in VBA, where a trigger may be given to start a read-poll.
- Manually, from a user-definable dialog. In the user dialog, the trigger may be given by the press of a button to actually start reading data.

Several advanced techniques are available to read data as flexible as possible. A special design has been used to cater for devices going off-line and back online. Such update techniques are optionally available in **eXLerate**: in a simple application reading of data is configured easily, while more complex applications these more advanced techniques may be used.

When data of a corresponding query is successfully transferred from a device to the computer, or vice versa, an offline/online event is sent to the **eXLerate** application, and the status column of the query in the Query Table is automatically updated.

Data in a query is sent to **eXLerate**, directly in the tag database worksheet, and may be additionally logged in a .LOG file for further analysis. This is a great tool for debugging your applications!

In the '*Advanced Topics Reference*' manual, various VBA functions, Sub routines and API for the above mentioned communication options are discussed in detail.

Data updates to external devices

Data may be written back to a device as well. For example, a master protocol may use a write command to update a value in a PLC, or a slave protocol may reply to an external master with a read response.

The process of writing data to an external device is usually *event-driven*, i.e. when the value changes in the application, it is immediately written to the external device, allowing the application to instantaneously respond to actions.

For example, when the operator issues a valve-close command by clicking on a valve object on the screen, an immediate write request may follow (in case a master device is handling the data transactions).

Alternatively, various properties are defined in a query to allow for periodical updates as well. The *interval* parameter in a query defines this behavior.

Controlling real-time data communications

Data communications may be started both in runtime mode as well as in design-mode, using the **eXLerate** menu. The following options are available from the **eXLerate** design-mode menu:

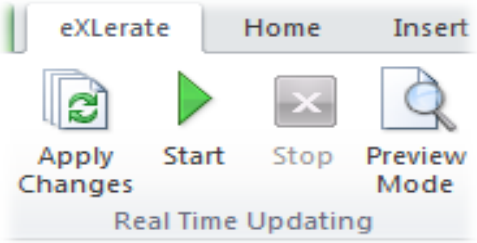


Figure 6-5: Communications related menu options

Configuring real-time data communications

Real-time data communications in an **eXLerate** project is configured from only two worksheets:

- '**xComms**'
  In this worksheet, the ActiveX control responsible for real-time data communications is present. From this worksheet, all communication protocols and message queries in the project are defined in the Protocol Table, and the Query Table respectively.

- '**xTagDB**'
  In this worksheet, the tag database worksheet, all settings for an individual tag are defined, including the properties needed for data-communications.

Since all real-time data communications are just configured from those two sheets, configuration of real-time data communications is orderly defined in an **eXLerate** application.

For example, printing the current communication settings of all protocols and associated messages does not take more than the printing of single worksheet.

When the Protocol Table, the Query Table, or the tag database are modified in eXLerate, the 'Apply Worksheet Changes' option should be activated prior to starting communications.

Protocol samples

For each specific protocol in eXLerate, a working sample workbook is available, in which available options, settings, and other parameters are defined. These settings can be copied into your project workbook. Check the specific workbooks for working examples and additional information of the protocols that you need in your application.

xlConnect, the protocol manager

xlConnect is the eXLerate family member with which you are able to 'connect' to external devices, i.e. the component that manages all external data-communication protocols.

Besides protocol management, cyclic interval processing is also taken care of by xlConnect. Cyclic interval processing is further discussed in section: '*Interval*' on page 7-149 onwards.

xlConnect is implemented as an ActiveX control, that is inserted in the 'xComms' worksheet. It looks as follows:

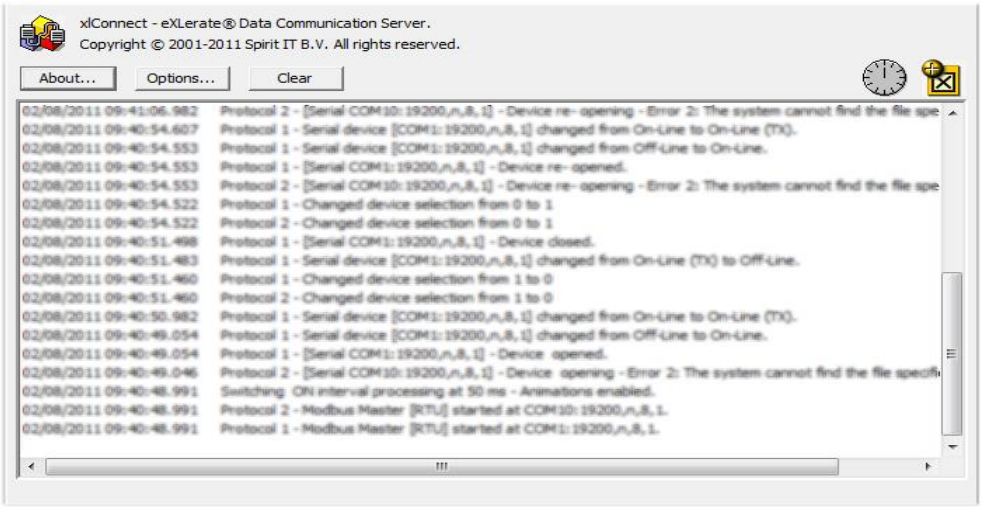


Figure 6-6: Main window of xlConnect with data scope/event logger

There are several items present on the control's main dialog window:

- an '**About...**'-button
  When this button is clicked, a pop-up dialog appears showing the available communication protocols with a short description and version number for each protocol. From this about box, license information may be displayed as well, so you can verify that you have the correct authorization code installed.

- The '**Options...**'-button
  The 'Options'-button at xlConnect's main window is used to set a number of message debugging options for each protocol, and for cyclic interval events.

- The '**Clear**'-button
  The event logger display will be cleared when this button is clicked.

- The **Clock** icon
  The clock icon is grey when no cyclic interval events are currently defined or sent to Excel, and orange when cyclic interval events are active, and sent periodically to Excel. At each new received event, the clock progresses to a next 'hour' on the icon.

Icon	Description
	Cyclic interval events are not currently active
	Cyclic interval events are currently active. At each new event, the hour pointer jumps to the next hour.

Table 6-1: Cyclic interval events animated icon

- The **Status** icon
  This icon is used to present the current status of the xlConnect ActiveX control to the user. Its presence is just informational, and has the following meaning:

Icon	Description
	The communication settings are correctly configured, and real-time data communications are currently updating Excel.
	Interval processing of communications are temporarily stopped, because the user has clicked once on this icon, or was stopped from the **eXLerate** menu. Communications may be resumed by clicking again on the status icon, or via the **eXLerate** menu.

Icon	Description
	Communications is programmatically paused, and may be continued. Usually not visible for the user, unless communications are set up via VBA.
	There was a warning issued when communications were set up. Communications are configured when the user issues the 'Apply Worksheet changes' from the **eXLerate** menu.
	There was a fault issued during communication set up, and therefore communications is unable to start. Correct the problem, which is logged in the system event logger, and try again.
	The configuration has been programmatically set up correctly. Normally not visible for the user, unless communications are set up via VBA.
	The configuration is not defined yet for xlConnect. Its database is still empty.
	There was a fault issued during communication configuration. Correct the problem, of which details are logged in the system event logger, and try again.

Table 6-2: Animated state icons of xlConnect

- The local **Logger/data scope** window
  The logger window of xlConnect is a local logger, which may be used as a data scope logger window, and a fault/warning message logger for messages related to data communications. To enable logging of message of xlConnect, use the Options dialog, and set the output to the local window. Data scope messages may be alternatively sent to the central event logger in the Control Center of **eXLerate**, and to a .LOG file for off-line analysis.

Protocol options

The protocol options dialog of xlConnect is selected with the appropriate button, after which the following dialog appears:

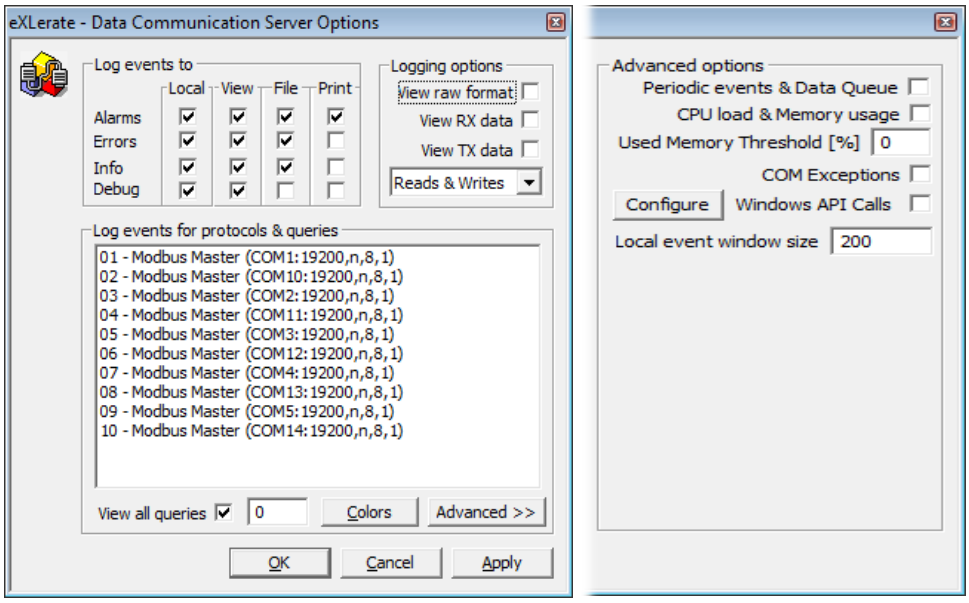


Figure 6-7: Communication server properties

For event logging, various logging devices may be enabled as output for generated events. 'Local' means the local window of xlConnect, 'View' is the logger window view of the Control Center, 'File' is a .LOG logger file in which event messages are stored, and 'Print' is the alarm/event printer in **eXLerate**. There are various message types: 'Alarms', 'Errors', 'Info', and 'Debug'. An *Alarm* is a message issued during run-time, while an *Error* is generated as a result of a faulty configuration. *Info*, and *Debug* are additional message types.

The data scope options are defined in the lower part of the dialog, where the device that may be monitored is entered, and the type of data scope messages to be logged. In the protocol window, a selection can be made to generate additional, protocol dependent messages.

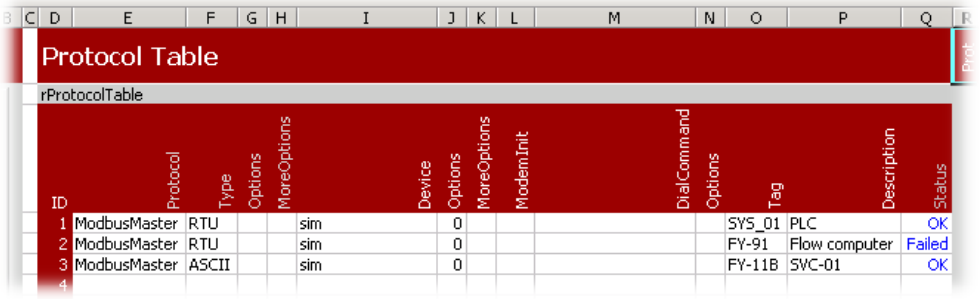
The stress slide bar is used to increase the communication speed with a defined factor, and may be used during application development for stress-testing your project. In the bar graphs, the current processor load, as well as memory usage is displayed, to tune the generated stress to an optimum level.

In the release version of your project, logging and/or stress factors should be avoided as much as possible, since logging has an impact on system resources in terms of processor utilization.

The Protocol Table

The *Protocol Table* is a table located in the 'xComms' worksheet, and contains the list of all communication protocols currently defined in the project. Multiple protocols are available, all running in parallel. For each protocol, a thread is created in Windows to ensure optimum system performance.

The protocol table has the following basic layout:



Protocol Table

rProtocolTable

ID	Protocol	Type	Options	MoreOptions	Device	Options	MoreOptions	ModemInit	DialCommand	Options	Tag	Description	Status
1	ModbusMaster	RTU			sim	0					SYS_01	PLC	OK
2	ModbusMaster	RTU			sim	0					FY-91	Flow computer	Failed
3	ModbusMaster	ASCII			sim	0					FY-11B	SVC-01	OK

Figure 6-8: Protocol Table layout

As with most specific configuration tables in **eXLerate**, the table starts with a table identifier ('rProtocolTable') 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.

The protocol table has various columns, each with a specific function:

- **ID**
  This is an index number of the protocol table, starting from 1, and up to and including the number of used protocols in the application. Note that there may be empty entries in the table.

- **Protocol**
  The key name of the protocol to be specified, for example: 'SBUS', or 'ModbusMaster'. Only protocols in accordance with your license key may be specified.

- **Type**
  Per protocol, a variation may be optionally entered, such as 'RTU', or 'ASCII' in case of Modbus. Per communication protocol, a sample worksheet file is available, in which most common settings for various communication protocols are defined.

- **Protocol Options and MoreOptions**
  At these fields, specific protocol dependent options may be entered. In the example above, no additional options are specified. The following options are supported:

Protocol	Options		MoreOptions
Modbus Master	RTU RTU**nn** ASCII ASCII**nn**	standard RTU message format **nn** may be 16, 32, 48, or 64 for fixed register size addressing mode standard ASCII message format **nn** may be 16, 32, 48, or 64 for fixed register size addressing mode	Delay time, in 0.1 [sec] units between two queries
Modbus Slave	RTU RTU**nn** ASCII ASCII**nn**	standard RTU message format **nn** may be 16, 32, 48, or 64 for fixed register size addressing mode standard ASCII message format **nn** may be 16, 32, 48, or 64 for fixed register size addressing mode	SIM for simulation of data
Modbus Client	RTU	standard RTU message format	
Clock	HOPF WHARTON	Specifies a Hopf DCF receiver Specifies a Wharton receiver	**RdCmd, Intv, Tmout, Tolr** RdCmd Read command byte Invt Interval time, in sec TmOut Timeout, in sec Tolr Tolerance of time, in sec When nothing is specified, default: "?", 300, 10, 2 (Hopf) or "T", 300, 10, 2 (Wharton) is used.
HART Master	None		Delay time, in 0.1 [sec] units between two queries
HART Slave	None		SIM for simulation of data
OPC client	OPC 2.0 OPC 1.0	compliance with the OPC DA 2.05 specification compliance with the OPC DA 1.0 specification	Option bits: **0x0001** Use OPCENUM **0x0002** Use cached values

Table 6-3: Protocol Options/MoreOptions settings

- **Device**
  This field is used to specify the device to be used for the protocol, i.e. a serial port (e.g. "COM56:9600,n,8,1"), or a TCP/IP address, in case of an Ethernet device (e.g. "192.168.0.10"). May be set to 'Sim' to simulate the device, in which case the protocol and pertaining data is simulated. Exact usage depends on the actual protocol; see the application example files.

- **Device Options**
  For the device hardware, two additional option fields are available.

  In case of a *serial* device, the Device Options are used for RTS switching. RTS switching allows for hardware/software control of the RTS line during signal transmission, for various signal converter support, for example when using the HART protocol. For the OPC client protocol, this is the time in seconds an item write action is allowed to take.

Options	[RtsMode[, TxDelay[, OffDelay[, RtsOptions]]]]
RtsMode	0: No RTS switching is supported (RTS remains low) 1: Use built-in Windows' RTS_TOGGLE switching option 2: Use software controlled RTS switching, positive logic 3: Use software controlled RTS switching, negative logic 4: RTS remains high
TxDelay	Transmission delay, in [ms]. Used with RtsMode 2 and 3, for a delay between a switched RTS line, and the moment at which the data is transmitted.
OffDelay	Inactivation delay, in [ms]. Used with RtsMode 2 and 3, for a delay between completion of data transmission, and inactivation of the RTS line.
RtsOptions	Options bits, only used in RtsMode 2 and 3 0x01: Use event based RTS control 0x02: Force flushing of data before RTS is dropped

Table 6-4: RTS options for serial hardware

Use these options with great precaution, since it highly affects system performance. The options only depend on the hardware used. If you have an application using RTS switching, please refer to one of the example worksheets where working settings are given.
In case of an Ethernet based protocol, the port number is specified in this field. For the ModbusClient protocol, always port 502 should be used.

- **Device MoreOptions**
  This is a reserved field, for future usage.

- **ModemInit**
  In case a modem will be connected to the system, a modem initialization string can be defined for the modem, e.g. a 'ATZ' or alike command may be sent to the modem prior to calling a telephone number.

- **DialCommand**
  This field allows for entering a dial command, such as: "ATDT +31402961234". A dial command entered at this field is only available during the configuration pass of communications, and cannot be used to

dynamically change telephone numbers while communications are running. In order to change a number, stop data communications, and then restart.

- **Options**
  Several options may be passed to the modem dialer, to setup various timers and timeout counters, as follows:

Options	[CmdTimeout[, Retries[, FailTimeout[, HangupDelay]]]]
CmdTimeout	Command timeout, in [0.1 sec] units
Retries	Number of retries on a command
FailTimeout	Connection timeout, or failed timeout, in [0.1 sec] units
HangupDelay	No activity timeout & hangup, in [0.1 sec] units

Table 6-5: Modem options

- **Tag**
  Optional field in which the name of the device, associated with the protocol may be defined, e.g. 'PLC-22' to designate a PLC in a system. May be left empty; not internally used by **eXLerate**.

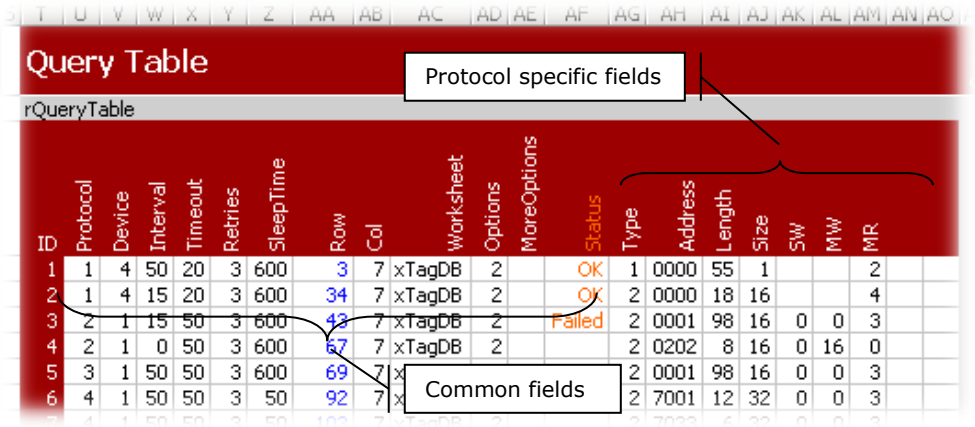
- **Description**
  A description of the device may be entered here as well, e.g. 'Main Controller in the system'; not internally used by **eXLerate**.

- **Status**
  This field is a calculated field, of which the content depends on the status of the associated queries. In case one or more queries go off-line, this status field increases with one, yielding to a combined status of which a non-zero value indicates that one or more associated queries (logical devices) have gone off-line.

The Protocol Table may be group-wise opened or group-wise closed using the Excel plus ('**+**') and minus ('**-**') buttons at the top of the worksheet.

The Query Table

All queries in the current project are defined from a single table, called the *Query Table*, which resides in the 'xComms' worksheet. The table has the following general layout:



ID	Protocol	Device	Interval	Timeout	Retries	SleepTime	Row	Col	Worksheet	Options	MoreOptions	Status	Type	Address	Length	Size	SW	MW	MR
1	1	4	50	20	3	600	3	7	xTagDB	2		OK	1	0000	55	1			2
2	1	4	15	20	3	600	34	7	xTagDB	2		OK	2	0000	18	16			4
3	2	1	15	50	3	600	43	7	xTagDB	2		Failed	2	0001	98	16	0	0	3
4	2	1	0	50	3	600	67	7	xTagDB	2			2	0202	8	16	0	16	0
5	3	1	50	50	3	600	69	7	x				2	0001	98	16	0	0	3
6	4	1	50	50	3	50	92	7	x				2	7001	12	32	0	0	3

Figure 6-9: The Query Table

As with most specific configuration tables in **eXLerate**, the table starts with a table identifier ('rQueryTable') 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.

The Query Table may be group-wise opened or group-wise closed using the Excel plus ('**+**') and minus ('**-**') and/or the '**1**' and '**2**' outline level buttons at the top of the worksheet.
The exact layout of the columns in this table depends on the associated protocol details.

The table has a size of at least 21 columns. Various fields are available for all protocols, while other fields are protocol specific.

The fields are described below:

- **ID [1]**
  This is an index number of the Query table, starting from 1, and up to and including the number of used queries in the application. Note that there may be empty entries in the table.

- **Protocol [2]**
  This field links to the Protocol Table, and designates a query to an existing protocol. In the example above, the first two queries (1 and 2) are linked to the first protocol (1), i.e. the first protocol has two queries defined.

- **Device [3]**
  This field specifies a device ID, which is usually a number. Most (multi

drop) protocols have a device ID, which should be specified for each message. For example, in Modbus devices 1..254 are legal device IDs. For the OPC client protocol, the device refers to a group name rather than a numerical value.

- **Interval [4]**
  The interval time applies to the time between two consecutive read- or write-polls, and is entered in 0.1 [sec] units, e.g. 30 specifies 3.0 second intervals. The relative start time of a query may be also defined, e.g. when the user specifies: '20:15', there will be a 2.0 second interval update for this query, which starts at 1.5 seconds after the 2.0 second interval time has elapsed. This option allows for time-wise distributed queries. To immediately send the first query at communication start, you may specify 20:-20.

- **Timeout [5]**
  The timeout parameter specifies the time, in 0.1 [sec] units, before a retry of the query is sent to the device in the event that no response message is received, or that the device is set to sleep, in case no retries have been defined.

- **Retries [6]**
  The number of retries in case of a timeout is specified with this parameter. When elapsed, the device is set to sleep.

- **SleepTime [7]**
  This parameter specifies the sleep time, in 0.1 [sec] units that the associated device is put in after all retries have been elapsed, and still no response is received at a message query. A sleep time is used to let other devices, which share the same protocol hardware, prevail communications over a failing device. When the sleep time has elapsed, communications are resumed.

The data of a query is sent to **eXLerate**, and a worksheet, usually the tag database receives this data, as a contiguous array of values. The exact destination of the array containing the new data is specified in the following three parameters: row, column, and worksheet name. In the sample projects, the column number is fixed, because all values arrive in the same column in worksheet 'xTagDB', while the row number is automatically calculated with the Excel worksheet function `MATCH(..)`.

- **Row [8]**, and **Col [9]**
  Specifies the row and column number of the first value in the array containing new data of the corresponding query, e.g. 10, 5 specifies 'R10C5', in Excel terms cell '$E$10'.

- **Sheet [10]**
  Specifies the worksheet name in which the array with data is to be written. In the sample projects, this is the 'xTagDB' worksheet.

Options [11]

The options field is used for all protocols, and specifies certain advanced details regarding data updates for associated items. The following options are currently supported:

Option	Description
xBlockWrites	Only use block-writes to a device, e.g. only write all fields in a query at once rather than either value belonging to the query has changed. When not specified, items may be individually updated (default).
xNewDataOnly	Only send new data to Excel; if retrieved data is equal to the data already available in Excel then no update takes place. This option prevents unnecessary calculation updates. When not specified, every read query causes an associated update in Excel (default).
xTransparentRead	Allow for reading of data while write updates are currently pending. When not specified, read polls are postponed during a write update (default).
xForcedWrites	Allow pending write commands to be executed after a device has gone back on-line. When a device has gone off-line, write queries cannot be sent to a device. This option allows for automatic update of the data once the device in online again. When not specified, no automatic write updates takes place, i.e. the user has to ensure these updates (default).
xNoReadOnce	Write-only queries are normally updated initially, if there is a read command defined for the query (default). This option disables even an initial read.
xItemUpdates	Items in this query are updated in Excel individually rather than group-wise, a query at a time. This option is available for the OPC client, but not available for all communication protocols. Check the protocol samples.
xNoSleepAll	Prevents all queries from going to sleep when at least one query in the protocol fails. This option is available for the Modbus communication protocols.
xWriteOnly	Items can only be written and are never updated in Excel. This option is available for the OPC client, but not available for all communication protocols. Check the protocol samples.
xWriteAll	Writes the whole query-block, when at least one item in the query was changed.

Table 6-6: Query options

Query options may be combined together, by adding individual settings. For example, option 'xNewDataOnly+xForcedWrites' means option 'xForcedWrites' (Allow pending write commands), plus option 'xNewDataOnly' (Only new data to Excel). If you do not feel comfortable defining such details as described above, refer to one of the working examples in one of the supplied workbooks.

- **MoreOptions [12]**
  This field is used for additional options. This option is currently only used by the OPCClient protocol where it represents a bitmask of the following options: 1: UseOPCEnum (by default the registry is used), 2: ReadFromCache (by default reads are from the device).

- **Status [13]**
  The status field is updated automatically by eXLerate. If a query goes off-line, a non-zero value is stored at this field. While on-line, a '0' is stored at this location. This query status field is used to update the protocol status field, by summing all query status fields of the appropriate queries, using the SUMIF(..) worksheet function. You may want to check this formula at the 'xComms' worksheet in the Protocol Table at this moment to see how the status of a query is processed.

Fields 14..21+ are used to specify the type of query, and are protocol dependent. Check the working sample workbooks for an example of the protocol you require. Below, fields 14..21 for Modbus (serial and TCP), HART, ASCII and IEC 870 protocol are included as an example.

- **Fields 14..21 (Modbus)**
  For a Modbus query, usually referred to as a poll-block, fields 14..21 are used to describe the poll-block type (14: *Type*), the start address of the query (15: *Address*), the number of registers of the query (16: *Length*), the register bit-size (17: *Size*), the single write command (19: *SW*), the multiple register write command (20: *MW*), and the multiple register read command (21: *MR*). See the Modbus workbook for working examples on various Modbus devices.

- **Fields 14..21 (OPC)**
  For an OPC query, usually referred to as a group, fields 14..21 at present is only used to describe the group type (14: *Deadband*). See the OPC Client workbook for working examples on various OPC clients. The OPC Group name is entered at the *Device* field. Fields *Retries* (6), and *SleepTime* (7) are currently not used for an OPC client.

- **Fields 14..21 (HART)**
  For the HART protocol, queries are referred to as HART commands. Fields 14..21 are used to specify the details of the command: long / short message type (14: *Type*), Master address (15: *Address*), Number of preambles (16: *Preambles*), number of bytes in the HART command (17:

Command bytes), number of bytes in the reply (18: *Reply bytes*), and HART command (19: *Command*). Fields 20 and 21 are not used in Hart.

- **Fields 14..21 (ASCII)**
  For an ASCII protocol, a query specifies the ASCII characters to be sent, and the location of data in the reply message: query type (14: *Type*), reply template string (15: *Template*), Number of registers in reply (16: *Registers*), number of <CR><LF> terminated strings in reply (17: *Size*), and command string (19: *Command*).

- **Fields 14..21 (IEC 870)**
  For a IEC 870 compliant query, usually referred to as a frame or telegram, fields 14..21 are used to describe the message type (14: *Type*), the data set number DN of the user data (15: *Dataset*), the size of the request frame (17: *OutSize*), the size of the reply frame (18: *InSize*), the Send/Reply command (19: SR), the Send/Confirm command (20: *SC*), and the Request/Respond command (21: *RR*). See the IEC 870 workbook for working examples on various IEC 870 compliant devices.

Advanced communication topics

Although many of the communication features within **eXLerate** operate implicitly and automatically, in some applications there may be the need to obtain further control of communications.

For example, when a device goes offline, and comes back online after some time, you might want to do something with pending write requests. A pending write request is a value that should have been written to the device, but because the device was offline never got that far. **eXLerate** is able to automatically write pending requests to the device when the device comes back on-line.

Advanced communication options are utilized using various methods:

- Using the options settings of a ***query***
- Using the ***`exUpdateEx(..)`*** worksheet function in the Update column at the tag database
- Using several ***VBA*** functions

Most tags with an external data source are read-only, i.e. the external device determines its value, and **eXLerate** simply retrieves its current value by periodically polling for data using one of the defined queries.

Tags may be also write-only, i.e. **eXLerate** determines the current value of a tag, for example by using your own worksheet function to determine its current value.

A write-only value may be updated to the external device using an *event driven* update, a *periodical* update, a *latched* update, or a combination of these three basic techniques.

- **Event-driven**
  At each change of the current value, the worksheet functions `exUpdateEx(..)`, `exUpdateStrEx(..)` and `exUpdateVarEx(..)` are used to signal xlConnect that the value must be written immediately. This is the most common technique, enabled by **eXLerate** by default. Check the function reference for details of using the `exUpdate***Ex(..)` worksheet functions.

- **Periodically**
  A cyclic interval update will write the current value repeatedly to the external device, even when it did not change. The interval value of the corresponding query must be set <>0 to enable periodical updates of write-only values.

- **Latched**
  Latched updates are updates that are grouped together. For example, you may want to send updates to an external device in a single update for all tags rather than to send each value separately. The trigger for such a common update may be controlled via VBA. Latched updates are important for redundant systems, where there may be two computers that need to be synchronized. The worksheet function `exUpdateEx(..)` is needed, in combination with the VBA function `exSetUpdateMode(..)` to configure latched updates. Check the function reference for details of using `exSetUpdateMode(..)`.

A combination of these techniques is quite easily enabled: you may want to update a value event-driven, and periodically. This is enabled by simply setting an interval value at the query definition, or by using the `exSetUpdateMode(..)` in VBA user function `OnEvent(..)`. The latter function is a predefined Sub in VBA, which is periodically called by **eXLerate**, in which you may add your own functionality.

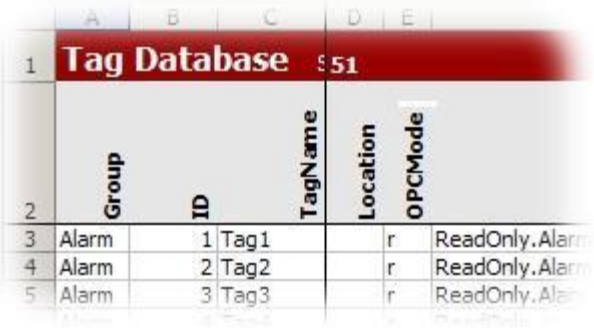
It is also possible to create read/write tags, although great care should be taken in the design of read/write tags to avoid update conflicts. To specify a read/write tag, the corresponding query should be made read/write. A query may be considered as the 'engine' behind the data taking care of read or write requests.

OPC Server

The OPC Server (Data Access 1.0/2.0) is an integrated part of eXLerate and the only communication protocol that does not need to be specified explicitly. The OPC Server makes tag data as found on the xTagDB sheet available to external applications, on the computer running eXLerate and/or over a network connection.

OPC Server Configuration

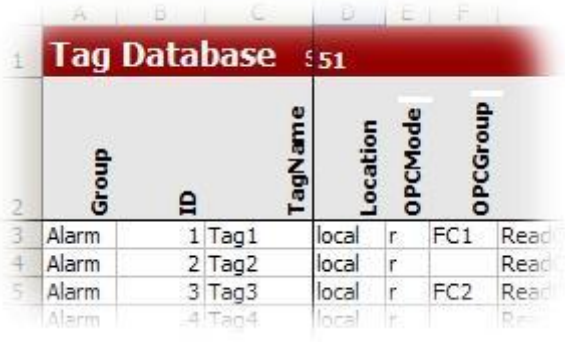
The default configuration for eXLerate will not start the OPC Server. If external access to the tag values is desired, an 'OPCMode' column has to be added to the xTagDB sheet. In the OPCMode column, the access mode of the tags can be



	A	B	C	D	E
1	Tag Database				
2	Group	ID	TagName	Location	OPCMode
3	Alarm	1	Tag1		r
4	Alarm	2	Tag2		r
5	Alarm	3	Tag3		r

Figure 6-10: OPCMode column

defined. Two values are allowed in the OPCMode column: "r" for a read-only tag and "w" for a read-write tag. When no value is specified, the tag will be hidden from the OPC Server.



	A	B	C	D	E	F	
1	Tag Database						
2	Group	ID	TagName	Location	OPCMode	OPCGroup	
3	Alarm	1	Tag1	local	r	FC1	Read
4	Alarm	2	Tag2	local	r		Read
5	Alarm	3	Tag3	local	r	FC2	Read

Figure 6-11: OPCGroup column

Optionally an 'OPCGroup' column can be added. This column will supply an OPC group name for the tag. If the OPCGroup column is missing completely or one of its fields is empty, the location value is used as OPCGroup.

After adding columns to the xTagDB sheet, make sure the xComm sheet is correct (especially the 'Query table' 'col' column). Moreover, adding/removing OPC Server functionality requires the tag & object wizard to be run.

OPC Server monitor

When the OPC Server is configured and the **eXLerate** application is running an extra icon will be shown in the windows notification area. When a user is logged

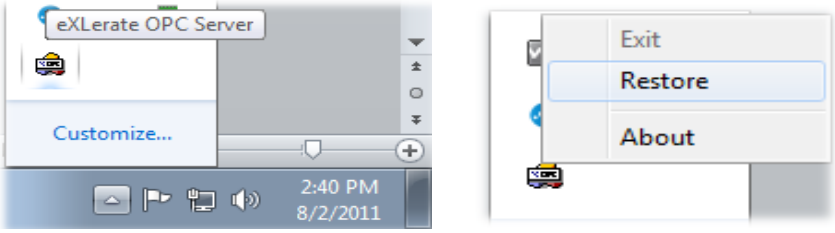


Figure 6-12: Windows notification area with xlOPC, xlCenter and report icons

with sufficient privileges (>=2000), clicking this icon or using the restore item on its context menu will open the **eXLerate** OPC Server monitor window. The **eXLerate** OPC Server monitor allows users to view which tag values are currently available in the OPC Server:

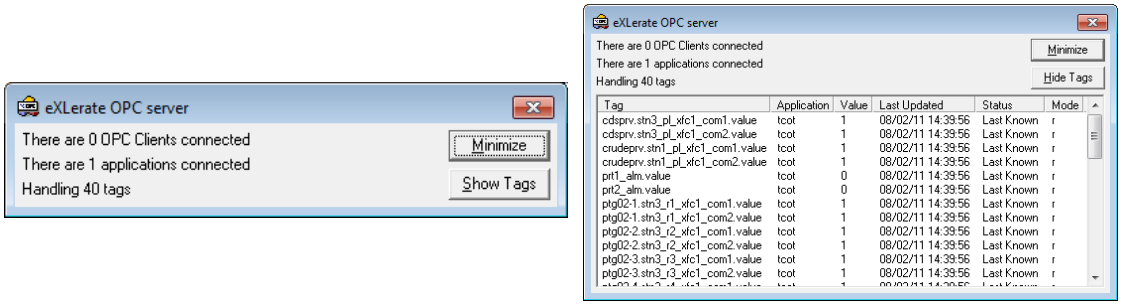


Figure 6-13: OPC monitor with tags hidden and shown

Normal behavior for the **eXLerate** OPC Server monitor is to keep running as long as there are OPC clients connected, an **eXLerate** application is open with exported tags or the window has been opened by a user. If the **eXLerate** OPC Server monitor is minimized and there are no OPC clients attached or applications running, it will automatically shut down after 60 seconds.

Chapter 7 - Intervals and Periods

Introduction

In your application there might be a need to periodically calculate certain values, for example an hourly average value of the current pressure, the daily total flow in a metering system, or a weekly production total.

You might want to automatically print a report containing calculated values based on the associated period, for example in a fiscal metering system.

In addition, you may want to create another recurring event, for example a 10 second event, in which you alternately open and close a valve in a PLC.

If you do not use such advanced options as VBA code that you periodically want to call, or periodical calculations, you might want to skip reading this section; this part is particularly added to eXLerate to automate the process of creating periodical data for reporting purposes in a fiscal environment.

Interval vs. period

The functionality mentioned above is implemented in eXLerate using two basic entities:

- A recurring *interval event* causing a report to be printed, or to trigger your VBA code in which you open/close a valve, and
- Functionality with which *periodical quantities* are calculated, such as moving averages or latched totals.

The recurring event in which certain (VBA) actions takes place in eXLerate is called an *interval event*, or simply *event;* the recurring periods over which the associated calculations take place is called a *calculation period*, or simply *period*.

It is obvious that the two entities interact highly: you need a defined period to calculate a daily average, and an event associated with the period to actually print the report.

Supported interval event types are: *second*, *minute*, *hour*, *day*, *week*, *month*, *quarter*, and *year*.

The user defined interval events and associated calculation periods are defined in a single table called the *Interval Table*.

Required calculations for a specific tag are configured in the tag database, in special fields (columns). The names of these fields start with 'P_', and are followed with the corresponding period name, e.g. 'P_Hour' contains period

definitions for hourly data. These fields may be generated by the *Tag & Object wizard*.

Interval events/calculation periods must be entered in ascending order in the *Interval Table*.

Example
In an application, a period of 8 hours is defined in which a report must be generated in which the average value of a pressure transmitter, 'PT1', is calculated. The report must be printed at every 8-hour cycle.

The 8-hour period, the calculation of the average value of the pressure transmitter during these 8 hours, the generation of a user-defined report as well as printing of the report may be implemented using built-in features of eXLerate.

Supported calculations

In eXLerate 2010 there are currently three types of periodical calculations defined, which are specified in the appropriate period columns of the tag data-base:

- A latched value specified with an '**L**'
- A moving average specified with an '**M**',
- A weighted average specified with a '**W**'

Latched values

A latched value acts as a sample and hold register, which takes the current value of a tag, and stores that value in memory for the duration of the associated period. When the period elapses a new value is sampled for the duration of this period.

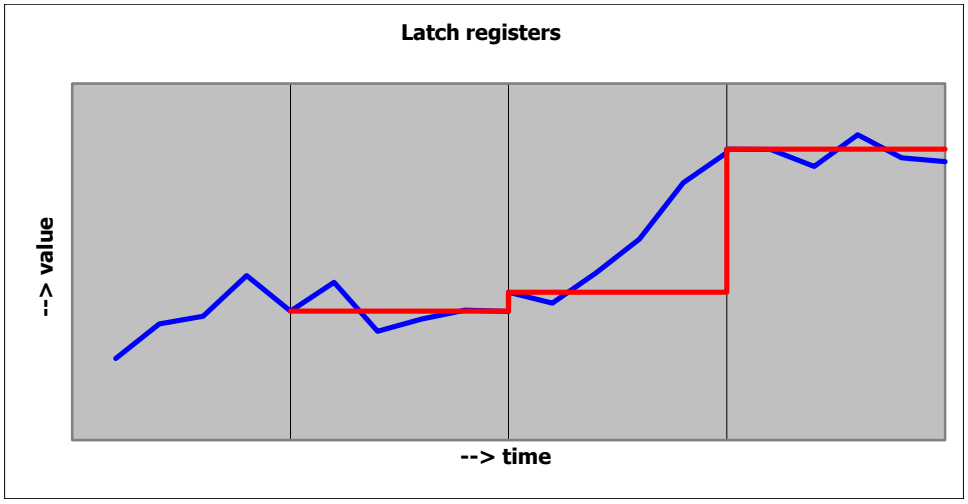


Figure 7-1: Sample and Hold registers, or 'Latches'

In the figure above, the blue line represents a varying signal, for example an hourly flow obtained from a process computer. Every new hour, represented by the black vertical lines, a new value is latched in the latch registers, of which the value is displayed with the red line.

Latches are especially useful to keep values for a certain period, such as 'previous day', or 'previous hour' data. Latches are retentive: at system startup, the last stored values are retrieved from the system registry.
Latch calculations always yield to an *array* of results rather than a single value.

Moving averages

A moving average is a calculated average of a real-time value over a certain period of time.

In a report, such process values may be reported, for example on an hourly basis, e.g. at each hour in the report, the value presented at that hour is a moving hourly average of that value.

Time is used as the basis of a moving average, for example with a window of one hour and a resolution of one second.

In **eXLerate**, all intermediate values for a moving average are persistent via the system registry to be able to recover from a shutdown situation.

The equation for a moving average value in **eXLerate** is given as follows:

$$P_{avg} = \frac{\int_{t0}^{tn} (P_i * t_i)}{\int_{t0}^{tn} (t_i)}$$

With:

P_{avg}	: Moving average value of parameter P in period $t_0..t_n$
P_i	: Current value of parameter P during interval time t_i
t_i	: Interval time between two consecutive samples, in [s]
t_0	: Start time of period in which the average is calculated
t_n	: End time of period in which the average is calculated

Note:
The start time t_0 and end time t_n are progressing in time, so the average value is calculated over a sliding, or moving window.

Equation 3: Moving average equation

Weighted averages

A weighted average is a calculated average of a real-time value, where rather a weigh factor is used than time for calculation of the average value. When the weigh factor increases, the contribution of the process parameter to the weighted averages increases.

A weigh factor may be based on an accumulative value, with which the difference between the previous and the current value is calculated. The calculated difference may be used as weigh-factor. Alternatively, a flow rate signal may be used as weigh factor.

The equation for a flow weighted average value is given as follows:

$$P_{avg} = \frac{\int_{t0}^{tn} (P_i * Q_i)}{\int_{t0}^{tn} (Q_i)}$$

With:

P_{avg}	: Weighted average of parameter P in period $t_0..t_n$
P_i	: Current value of parameter P during interval time t_i
Q_i	: Weigh factor between two consecutive intervals, for example an accumulative flow value
t_0	: Start time of period in which the average is calculated
t_n	: End time of period in which the average is calculated

Equation 4: Weighted average equation

In the example below, an example is given of the calculations involved.

	P	Q	P*Q	P*Q/Qt
7:00	15	2000	30000	0.60
8:00	30	100	3000	0.06
9:00	20	0	0	0
10:00	20	100	2000	0.04
11:00	15	1000	15000	0.30
Arithmetic average	20			

Sum		3200 (Qt)	50000	1.00
Weighted average of parameter P			15.625	

Parameter P, reported at 9:00 is not used for the averaging process, because the weigh factor was zero during that hour. The weighted average is calculated by dividing 50000 by 3200, which yields 15.625. The arithmetic average is 20.

The Interval Table

The Interval Table, which defines all cyclic interval events and associated calculation periods in **eXLerate**, has the following layout:

Interval Table

rIntervals

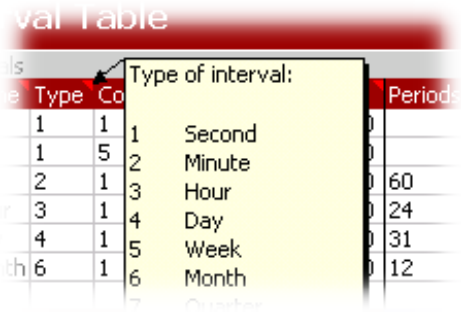
ID	Name	Type	Count	MM	DD	hh	mm	ss	Periods	StartOn	ResetBy	Previous	Current	Description
1		1	1					00						The application calculation event
2		1	5					09						5 second interval
3	Min	2	1					23	60	12		2	3	1 minute interval with in total 60 periods
4	Hour	3	1				02	00	24	0		16	17	1 hour interval with in total 24 periods
5	Day	4	1			08	02	30	31	0		15	16	1 day interval with in total 31 periods
6	Month	6	1		01	08	03	00	12	0		5	6	1 month interval with in total 12 periods

<--- Events | Period processing --->

Figure 7-2: The Interval Table

As with most specific configuration tables in **eXLerate**, the table starts with a table header with caption: Interval Table', the Excel range name ('rIntervals') which defines the table for **eXLerate**, a number of field headers, and the data in the table itself.

The Interval Table may be group-wise opened or group-wise closed using the small Excel plus ('**+**') and minus ('**-**') buttons and/or the level '**1**' and level '**2**' buttons at the top of the worksheet.



A small help/comment for each field in the Interval Table is available.

Select the appropriate column of the table, and a little pop-up help window appears in which explanation follows on usage of the column, as in the figure left.

There are 15 columns in the Interval Table. Columns 1-9 define the interval events in the application; columns 10-14 may define additional associated calculation periods. Column 15 may contain a general description of the interval.

Interval events may exist without an associated calculation period, but a calculation period can never exist without a preceding interval event.

Intervals must be entered at the table in *ascending* order to allow for *cascading* period calculations.

The columns in the table have the following purpose:

- **ID [1]**
  This is a key index number of the table, starting from 1, and up to and including the number of used recurring intervals in the application. Note that there may be empty entries in the table.

- **Name [2]**
  The name of each interval is internally used as a key, and should be unique. The name of an interval is used throughout the application, and determines the column name in the tag database associated with this period. For example, the 'Hour' interval has a associated tag database column name: 'P_Hour', in which for each tag the automatic periodical calculations are defined for that period. There may be empty entries in the table, in which case no associated column name for the tag database is associated or created – preventing generation of automatic period calculations for this interval.

- **Type [3]**
  The type of periodic interval can be one of: second (use predefined constant '=xSec'), minute ('=xMin'), hour ('=xHour'), day ('=xDay'), week ('=xWeek'), month ('=xMonth'), quarter ('=xQuarter'), and year ('=xYear'). This is the basic type, which will be combined with the size parameter to obtain the cyclic events associated with a period.

- **Count [4]**
  The count of a period determines at *how many* of the intervals specified at *type* an event is triggered by **eXLerate**. For example, if you want a period defined of 5 seconds, set *type* to '1', and *count* to '5'. A period of 3 minutes is defined by setting *type* to '2', and *count* to '3'.

- **Fields MM/DD/hh/mm/ss [5-9]**
  These fields determine the actual start moment of the interval event. For example, a minute interval may be created starting at hh:mm:10, i.e. 10 seconds after the whole hour. Another example is a daily interval, starting at 08:00, which could be the start time at which you may want to print out your daily report.

  COMMENT
  The above fields allow for creation of latency in consecutive interval events, i.e. a small delay may be built in before the system progresses from one interval to the next interval, for example for report generation. This is useful in systems where data must be retrieved from external systems before a report is generated, e.g. the system may wait a few

seconds to allow external data to be retrieved prior to the generation of a new hour. This is why hourly events should start a few seconds later than exactly on the hour, and a daily event perhaps a minute later than the exact new day, etc. Of course, no data will be lost when using such latency; only the moment at which a report is generated is affected.

- **Periods [10]**
  This optional field contains the number of interval periods that are used for a full period calculation cycle. When not defined, no period calculations are available for this interval. This makes the interval event-only.

  COMMENT
  For example, you may have 24 hours in a single day for which you may want to calculate hourly totals, i.e. every hour, an hourly event is to be generated, and after 24 hours, the cycle is completed, after which a new day starts. *Type* would be set to '3' (hour) in this case, *Count* would be set to '1', to obtain a calculated average one every hour, and *Periods* would be set to '24', to define that there are 24 calculated periods of 1 hour for a full cycle.

- **StartOn [11]**
  This optional field is used to define at which interval event the calculation period starts. In the example above, you might want to start a daily report to be printed at 08:00, which means that this field would have a value of '8'. The StartOn field relates to the *Periods* field. If omitted, a new calculation period always starts on '0', which is the first period.

- **ResetBy [12]**
  The *ResetBy* field is an optional field, which may contain the name of a higher interval, which will reset the periodical data of this interval, but only **after** the referred higher calculation period has been processed. It is available for advanced period calculations.

  COMMENT
  If not defined, the periodical data of this interval will be automatically reset after the *StartOn* period event has occurred. This postponed reset of periodical data, which is defined with the *ResetBy* field allows for calculation results at the higher period based on calculations of a lower period. For example, a daily total could be based on hourly results, generated by a lower hourly event.

- **Previous [13], and Current [14]**
  These fields are filled by **eXLerate** when a new interval event has been triggered, and may be used by the application for additional functionality. *Previous* and *Current* are updated by **eXLerate** at each new event, and cycle between 0 and *Periods*.

COMMENT
For example, to define the name of a report, the *Previous* field may be used as tab name in a report workbook. The values are always set by **eXLerate**, and therefore read-only for the user.

- **Description [15]**
  This is a user-definable field, not needed for **eXLerate**, but added for user convenience.

Interval processing

The Interval Table is processed by **eXLerate** during runtime on various levels.

Each period of the Interval Table has an associated period field in the tag database. For example, the 'Hour' period has an associated 'P_Hour' column in the tag database, where the user may specify which periodical calculations are required for the tag during the period.

All period calculations are updated at each new interval event. This process is visualized as follows:

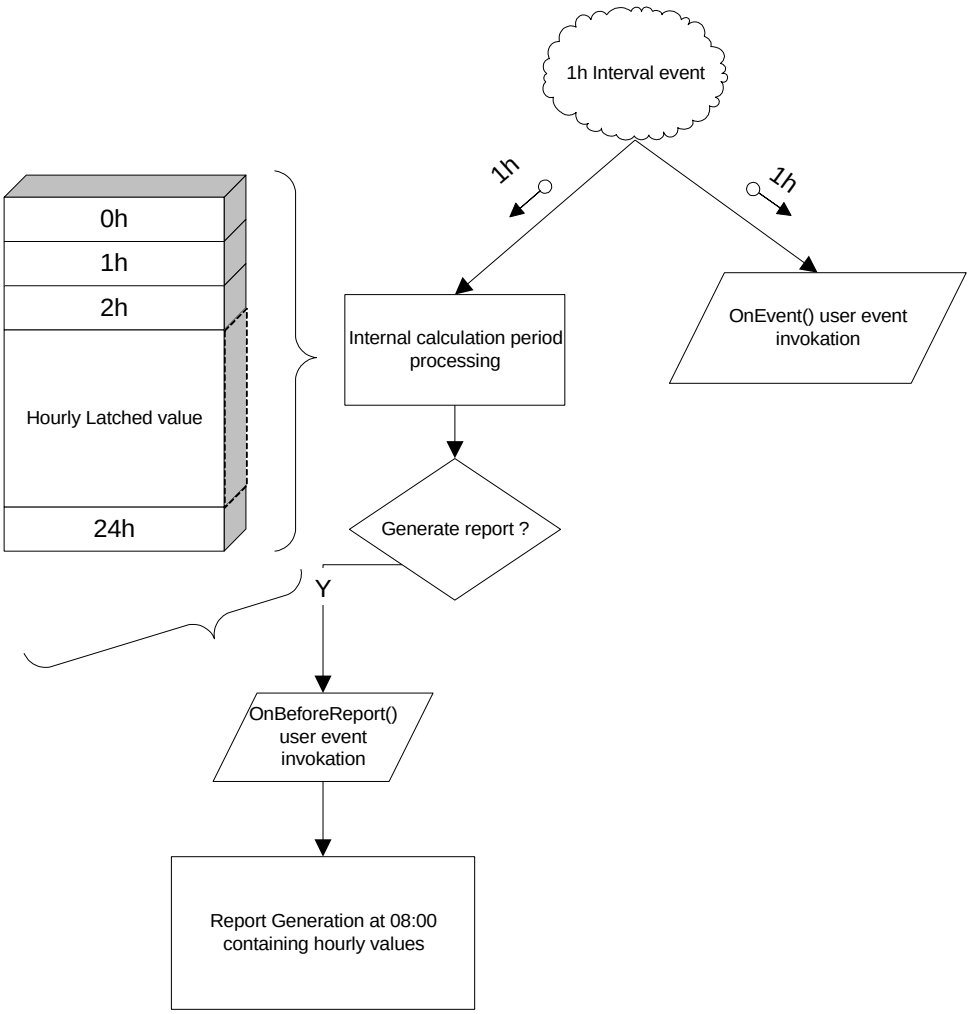


Figure 7-3: hourly interval event & period processing example

The '`OnEvent()`' and '`OnBeforeReport()`' events as mentioned in the flow chart above are user-definable VBA subroutines that are called by **eXLerate** on certain events. These events, and other events are discussed in detail in the Advanced Topics Reference, '*User Application Events*'.

Generated objects

When the required periodical calculations have been specified at the appropriate period columns in the tag database for each tag and for each period, the Tag & Object wizard may be started to actually generate the required objects and corresponding calculations. These objects are generated in the 'xWizard' worksheet.

To learn more about using the Tag & Object wizard, check the 'Advanced Topics Reference' manual.

Latches are configured with the '**L**' token in the appropriate period column in the tag database. When a tag value is latched, the following objects are available in eXLerate, after the Tag & Object wizard has been used to generate these objects:

Object name	Description
x{tag}.{Period}.Current	The current latched value of {tag} at period {Period}
x{tag}.{Period}.Previous	The previously latched value of {tag} at period {Period}
x{tag}.{Period}.rPeriods	A range, of size *Periods*, containing all latched values of {tag} at period {Period}

Figure 7-4: Latch objects

The Tag & Object wizard creates both the objects as well as the worksheet (array-)values containing the required data. For latches, an array of values is returned. The size (Rows x Columns) of the array is *Periods* x 1.

This result array may be pasted in the application using a standard Excel array formula. An array formula in Excel is entered as follows:

- Select the range, for example 24 rows and a single column with the cursor
- Press <F2> to edit the expression
- Enter the following: '`=xTR1TA.Hour.rPeriods`'
- Press <Shift-Ctrl-Enter> to enter the array formula

Now the range holds the latched data

Moving and *weighted averages* are configured with respectively the '**M**' and '**W**' token in the appropriate period column in the tag database.

The following objects are available after invocation of the Tag & Object wizard:

Object name	Description
x{tag}.{Period}.Mavg	The moving average value of {tag} over the {Period} period. The value returned is a single value, not an array of values.
x{tag}.{Period}.Wavg	The weighted average value of {tag} over the {Period} period. The value returned is a single value, not an array of values.

Figure 7-5: Average objects

Cascading calculations

Latches, moving averages and weighted averages will be automatically cascaded, i.e. the result of a preceding period may be used for the next period. The basic idea is to achieve maximum resolution at a minimum resource usage (processor load and memory consumption). Cascading is automatically enabled for calculations of consecutive periods.

In the figure below, the relationship between the *Interval Table* and the associated columns in the tag database are highlighted. At the right bottom cutout, cascaded latches are defined, which create daily and monthly latches for moving averages, based on minutes and hours.

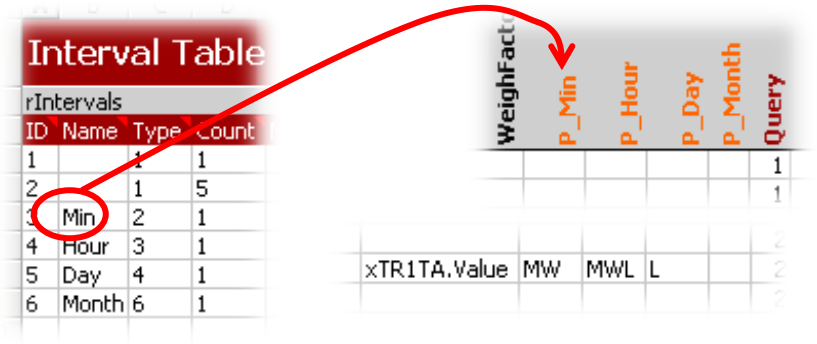


Figure 7-6: Relation between Interval Table, and tag database columns 'P_...'

In the example above for tag 'xTR1TA', the latched periods contain data for all previous period averages. The periods are as expected from the logical name: the 'Min' period contains 60 x 1 minute intervals, and lasts thus 1 hour; the 'Hour' period contains 24 x 1 hour intervals, and lasts thus 1 day, etc.

The 'Min' period has a weighted and a moving average, with the highest resolution based on maximum 3600 samples, e.g. each second a new sample is used for calculation of a minute average.

The 'Hour' period with 24 periods has a *cascaded* moving average: at the end of each hour, the averaged 60 min sample from the preceding 'Min' period is used for the 'Hour' average, thus cascading the moving average from the previous period in the next period. There is also a weighted average defined, and a latch, which creates both a latch for 'xTR1TA', every hour, as well as a latch for the hourly averages, ánd a latch for the preceding period averages.

The 'Day' period contains a latch only. The effect of this latch is that both the current value of 'xTR1TA' is latched, and the preceding 'Min' and 'Hour' averages.

A 'rule-of-thumb' in cascading is that it is best to cascade moving and/or weighted averages of the same period, and/or one preceding period; cascading more seems doubtful, e.g. latching a month based on minute data.

In the tables below, the created object names of the example cascaded period calculations are listed.

For the 'Min' period in the example, the following object names are created:

Object	Description
xTR1TA.Min.Mavg	The moving average value of PT1 over the 'Min' period. The value returned is a single value, not an array of values.
xTR1TA.Min.Wavg	The weighted average value of PT1 over the 'Min' period. The value returned is a single value, not an array of values. The weigh factor is the current value of xTR1TA.

Table 7-1: Objects created for period: 'Min'

This period contains only two values. These values will be used for cascaded period calculations below to obtain the highest resolution average.

For the 'Hour' period in the example, the following object names are created:

Object	Description
xTR1TA.Hour.Mavg	The moving average value of PT1 over the 'Hour' period. The value returned is a single value, not an array of values. The calculation internally uses the result from 'xTR1TA.Min.Mavg' to obtain a highest resolution resulting average value.
xTR1TA.Hour.Wavg	The weighted average value of PT1 over the 'Hour' period. The value returned is a single value, not an array of values. The calculation internally uses 'xTR1TA.Value' to obtain the resulting average value.
xTR1TA.Hour.Current, xTR1TA.Hour.Previous, xTR1TA.Hour.rPeriods	Latch results, with xTR1TA.Value as latched input value. The xTR1TA.Hour.rPeriods is the array with latched results. The data is latched at the end of the 'Hour' period, which is every hour. The total range contains 24 hours, or a full day of data.
xTR1TA.Min.MAvg.Hour.Current, xTR1TA.Min.MAvg.Hour.Previous, xTR1TA.Min.MAvg.Hour.rPeriods	Cascaded latch results, with xTR1TA.Min.Mavg as input value, taken from the previous 'Min' period. This causes a continuation of the moving averages with a higher resolution input from the preceding period. *Recommended for moving averages.*
xTR1TA.Hour.MAvg.Hour.Current, *xTR1TA.Hour.MAvg.Hour.Previous,* *xTR1TA.Hour.MAvg.Hour.rPeriods*	Cascaded latch results, with xTR1TA.Hour.Mavg as input value, taken from this 'Hour' period. The result is the creation of another type of moving average, with a lower resolution input than in the latches above. *Not recommended for moving average*.
xTR1TA.Min.WAvg.Hour.Current, *xTR1TA.Min.WAvg.Hour.Previous,* *xTR1TA.Min.WAvg.Hour.rPeriods*	Cascaded latch results, with xTR1TA.Min.Wavg as input value, taken from the previous 'Min' period. Because of the nature of weighted averages not recommended. See the equations for weighted averages.
xTR1TA.Hour.WAvg.Hour.Current, xTR1TA.Hour.WAvg.Hour.Previous, xTR1TA.Hour.WAvg.Hour.rPeriods	Cascaded latch results, with xTR1TA.Hour.Wavg as input value, taken from this 'Hour' period. The result is the creation of cascaded weighted averages. *Recommended for weighted averages.*

Table 7-2: Objects created for period: 'Hour'

There are now two possibilities for using latched averages: the 'Hour' hourly updated values, or the 'Min' higher resolution data. Usage depends on your application, and the averaging type. Typically, use the averages based on data of the preceding period, in this case the 'xTR1TA.Min.MAvg.Hour.rPeriods' range

for moving averages, and the averages based on the current period for weighted averages.

For the 'Day' period of the example objects of Figure 7-6 on page 7-160 above, the following data objects are available:

Object	Description
xTR1TA.Day.Current, xTR1TA.Day.Previous, xTR1TA.Day.rPeriods	Latch results, with xTR1TA.Value as latched input value. The xTR1TA.Day.rPeriods is the array with latched results. The data is latched at the end of the 'Day' period, which is once every day. The total range contains 31 days, or a full month of data.
xTR1TA.Min.MAvg.Day.Current, xTR1TA.Min.MAvg.Day.Previous, xTR1TA.Min.MAvg.Day.rPeriods	Cascaded latch results, with xTR1TA.Min.Mavg as input value, taken from the first 'Min' period. The result is however unlikely to be used in your application, because it contains gaps of data.
xTR1TA.Hour.MAvg.Day.Current, xTR1TA.Hour.MAvg.Day.Previous, xTR1TA.Hour.MAvg.Day.rPeriods	Cascaded latch results, with xTR1TA.Hour.Mavg as input value, taken from the previous 'Hour' period. This causes a continuation of the moving averages with a higher resolution input from the preceding period. *Recommended for moving averages.*
xTR1TA.Min.WAvg.Day.Current, *xTR1TA.Min.WAvg.Day.Previous,* *xTR1TA.Min.WAvg.Day.rPeriods*	Cascaded latch results, with xTR1TA.Min.Mavg as input value, taken from the first 'Min' period. Not recommended.
xTR1TA.Hour.WAvg.Day.Current, *xTR1TA.Hour.WAvg.Day.Previous,* *xTR1TA.Hour.WAvg.Day.rPeriods*	Cascaded latch results, with xTR1TA.Hour.Wavg as input value, taken from the previous 'Hour' period. Because of the nature of weighted averages not recommended. See the equations for weighted averages.

Table 7-3: Objects created for period: 'Day'

In this example, we have created moving and weighted averages for hourly, daily, and monthly reporting purposes, with the definition of just three tokens in the tag database: 'MW', 'MWL', and 'L' for minute, hour, and day periods respectively.

Weighted averages are not created if the weigh factor isn't entered. The weigh factor as entered in *WeighFactor* should be the name of a tag. The value of this tag is assumed to be time-wise incrementing, i.e. at every new call, its value increases, as is the case with an eternal flow. Alternatively, this weigh factor may be a (flow-)rate.

Calculation triggers

eXLerate generates so-called *calculation triggers* for the above mentioned interval events. These calculation triggers are internally used by worksheet functions of eXLerate, and may be used for your own application development.

For interval based calculations and optional cascaded calculations, the sequence in which the periodical events as defined in the *Interval Table* take place is quite important.

For example, when an hourly report is to be printed, it is important *first* to calculate a new result for that hour, *then* to store the result for printing in the appropriate register, *after* which the old result is to be reset to 0 for a new hour.

Such an event causing a function to be re-calculated by Excel is called a calculation *trigger*.

Various calculation triggers are defined for each period in the interval table.

When an interval event takes place in eXLerate, for example for period *Hour*, the following updates take place:

- The internal variable: '`xPeriod.Hour.PreTrigger`' is triggered (by letting eXLerate change its value). It is called a trigger because Excel will recalculate all worksheet functions referring to this flag.
- The *previous* period field in the *Interval Table* is updated with a new previous value. The variable is called: '`xPeriod.Hour.Previous`' in the application.
- The *current* period field in the *Interval Table* is updated with a new current value. The current period field is called: '`xPeriod.Hour.Current`'.
- The variable: '`xPeriod.Hour.PostTrigger`' is updated.
- A report for that period is actually generated, using the correctly triggered function results.

This process can be visualized as follows:

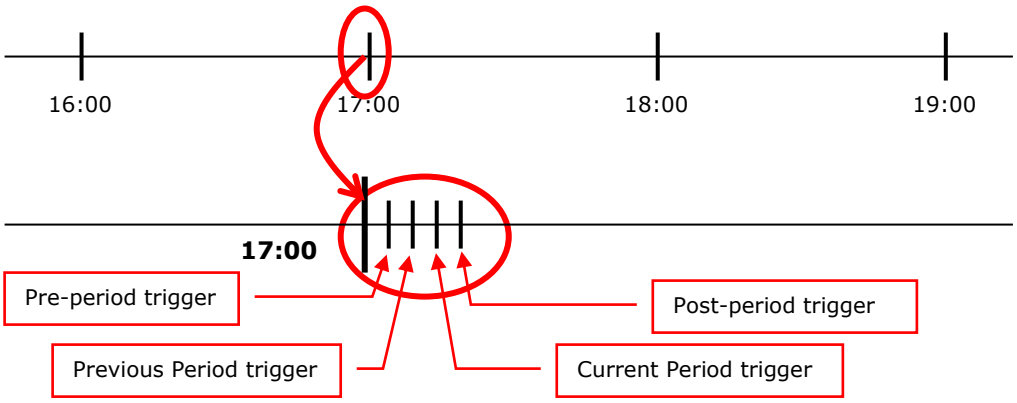


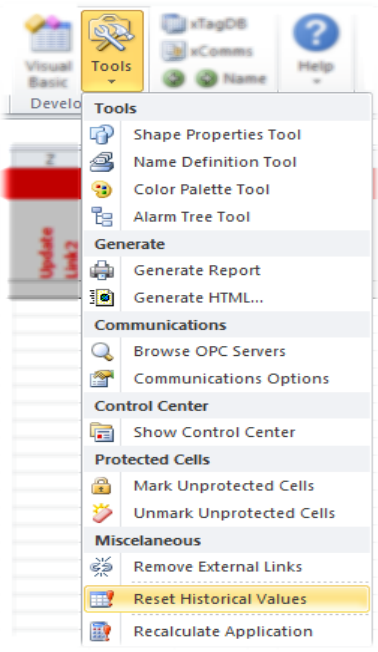
Figure 7-7: Cascaded calculation triggers on an interval event

At 17:00, an event is defined, causing a sequence of events to take place, being the pre-period trigger, previous period update, current period update, and a post-period trigger.

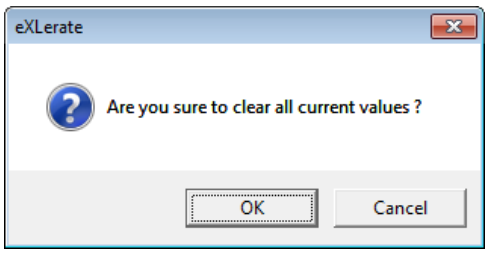
Usage of worksheet functions in combination with the above mentioned calculation triggers is also discussed in the function reference in the 'Advanced Topics Reference' manual.

Resetting historical values

The running historical values in an application may be reset using the 'Reset Historical Values' option from the Tools option in **eXLerate**.



A confirmation dialog is shown, as follows:



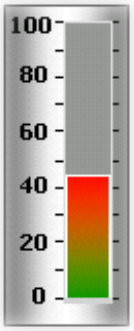
When 'OK' is activated, the historical values are reset to 0. Historical values in this context are intermediate running data for latches and averages. Trend and log files, or report output files are not touched by this command.

This page is intentionally left blank.

Chapter 8 - Object animations

Introduction

The process of animating shape objects is already introduced in 'Chapter 3 - Tutorial', from page 3-75 onwards. If you skipped this chapter, it is perhaps a good idea to read this chapter now and get familiarized with shape animations.



eXLerate uses standard Excel shapes and their accessible** properties internally for its data-driven shape animations.

Regular worksheet cells may be animated as well using a standard Excel feature: 'Conditional formatting'. With conditional formatting, you are able to set the font style, underline, color, and strikethrough font properties, border properties, and pattern properties for cell-colors.

Refer to '*Cell formatting*' on page 5-122 for usage of dynamic cell formatting in Excel; this section discusses how to animate a shape object.

A shape is a Microsoft Office drawing object, that **eXLerate** uses to animate a display page, for example a bar graph as displayed above.

This section basically explains how to animate a shape object by using the *Animation Table*, and the functionality behind the Animation Table.

In addition, worksheet cell *ranges* may be animated as well. Using range animations, named ranges, or individual (unnamed) cells may be set to a specific color, or may have a blinking attribute set.

The Animation Table

In **eXLerate**, animations of shape and range objects in all display pages are stored in a single table, the *Animation Table*.

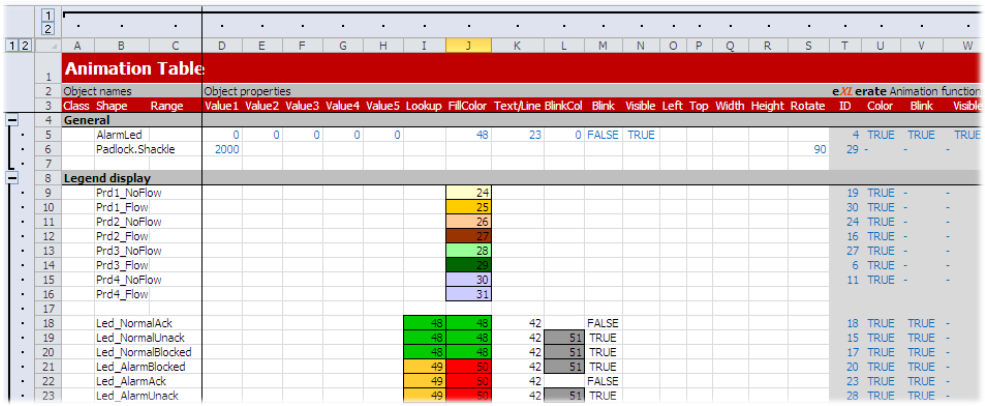
The Animation Table is stored in worksheet 'xAnimations' in the 'MyTemplate' project sample, but may be located elsewhere.

If you need to animate an object in **eXLerate**, an entry in the Animation Table must be created for it.

The Animation Table has various rows, one for each animated object, and a number of columns, one for each animation property.

** *Using Microsoft COM (Component Object Model) techniques*

Columns at the **left**-hand side of the table contain actual shape **properties**, i.e. the shape property values for the shape object, as follows:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	**Animation Table**																						
2	Object names			Object properties																eXLerate Animation function			
3	Class	Shape	Range	Value1	Value2	Value3	Value4	Value5	Lookup	FillColor	Text/Line	BlinkCol	Blink	Visible	Left	Top	Width	Height	Rotate	ID	Color	Blink	Visible
4	General																						
5		AlarmLed		0	0	0	0	0		48	23	0	FALSE	TRUE						4	TRUE	TRUE	TRUE
6		Padlock.Shackle		2000															90	29	-	-	-
7																							
8	Legend display																						
9		Prd1_NoFlow								24										19	TRUE	-	-
10		Prd1_Flow								25										30	TRUE	-	-
11		Prd2_NoFlow								26										24	TRUE	-	-
12		Prd2_Flow								27										16	TRUE	-	-
13		Prd3_NoFlow								28										27	TRUE	-	-
14		Prd3_Flow								29										6	TRUE	-	-
15		Prd4_NoFlow								30										11	TRUE	-	-
16		Prd4_Flow								31													
17																							
18		Led_NormalAck							48	48	42		FALSE							18	TRUE	TRUE	-
19		Led_NormalUnack							48	48	42	51	TRUE							15	TRUE	TRUE	-
20		Led_NormalBlocked							48	48	42	51	TRUE							17	TRUE	TRUE	-
21		Led_AlarmBlocked							49	50	42	51	TRUE							20	TRUE	TRUE	-
22		Led_AlarmAck							49	50	42		FALSE							23	TRUE	TRUE	-
23		Led_AlarmUnack							49	50	42	51	TRUE							28	TRUE	TRUE	-

Figure 8-1: Animation Table shape *properties*

The following *properties* are available:

- **Class**
  A class is like a group, where the application developer is able to create groups containing identical shape objects, such as valves, transmitters, devices etc. In the 'MyTemplate' application the classes are row-wise grouped, so a user is able to select the group level with the '1' and '2' group levels, or use the '**+**' and '**-**' buttons. This is an optional property (may be left empty).

- **Shape**
  The name of a shape object. Shapes have a global scope, i.e. various sheets may contain the same shape object, even though the shape object itself differs between worksheets. For example, one may define a 'valve' shape, which may be a different shape on various display pages: it may be a large valve on display 1, a rotated shape on display 2, and even a totally different object on display 3. As long as the shape with this name is located on a worksheet, it may be animated using this name. This is a mandatory property (must be non-empty, as well as unique in the Animation Table). If shapes are grouped together, the sub-shapes can be accessed using a '.'-character as separator. So, if a grouped shape is called 'MyGroup' and within that group a shape exists with name 'Valve', then the shape name would be: 'MyGroup.Valve'.

- **Range**
  The name of a range object. Ranges have a workbook scope, where a name is attached to a unique range in the workbook.

- **Lookup**
  This optional column may be used to implement user-defined color lookup tables. In the 'MyTemplate' sample, various shape colors are calculated via this lookup column in conjunction with a user-defined table (e.g.

'rValveLookup'). The idea is to convert a real-time value from the tag database into a color lookup value, so a shape object changes color at certain conditions based on "live" data. Various color lookup tables are also located on the 'xAnimation' worksheets.

- **FillColor**
  This field specifies the animation color index of the shape object. This is an optional property (may be left empty).

- **TextColor**
  This field specifies the text-color index and applies to range animations only. This is an optional property (may be left empty).

- **LineColor / BackColor**
  The line / background color index specifies the color to be used for lines in a shape, identical with the fill color property, or sets the cell background color for range animations. This is an optional property (may be left empty).

- **Visible**
  Optional field which defines the visibility of a shape object. If a shape should be made invisible, the current value of this field should be 0; otherwise the shape will remain visible. To actually set the visibility property of a shape, the corresponding **eXLerate** worksheet function should be inserted as well, at the right hand side of the Animation Table. This is an optional property (may be left empty).

- **BlinkCol, Interval**
  Shapes and ranges can be made blinking with a steady interval of about 0.5 seconds if a blink color is applied to *BlinkCol*, and *Interval* contains an expression. If *Interval* yields to non-zero, the shape or range will start blinking. Blinking in this context is a toggle of colors between *FillColor* and *BlinkCol* for shapes, and a toggle of colors between *BackColor* and *BlinkCol* for ranges. These are optional properties (may be left empty).

- **Left**
  Specifies the left position of a shape object, to be used for dynamically moving shape objects on a display. This is an optional property (may be left empty).

- **Top**
  Specifies the top position of the shape object, to be used to dynamically move shape objects on a display. This is an optional property (may be left empty).

- **Width**
  Specifies the width of the shape object, to be used to dynamically size shape objects on a display. This is an optional property (may be left empty).

- **Height**
  Specifies the height of the shape object, to be used to dynamically size shape objects on a display. This is an optional property (may be left empty).

- **Rotation**
  The angle at which the shape object rotated, to be used to dynamically rotate shape objects on a display. The rotation is a number between 0-360. This is an optional property (may be left empty).

The Shape Properties Tool

Shape properties may be viewed using the ***S**hape Properties Tool*, which is a tool pop-up dialog, activated from the **eXLerate**, Tools option. The current properties as described above are presented in the dialog. By pressing one of the buttons, the associated property is copied directly in the Animation Table at the appropriate cell.

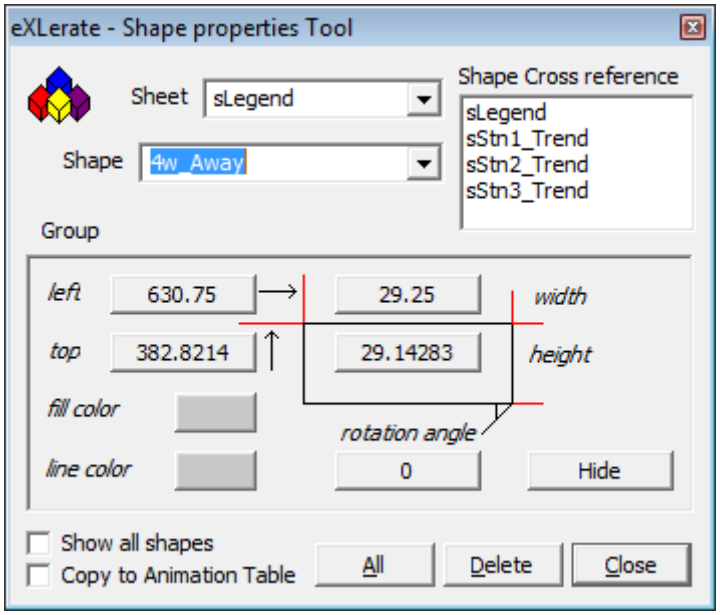


Figure 8-2: Shape properties Tool

The Shape properties Tool is a handy tool to obtain current positioning and size values from a selected shape on a worksheet in the Animation Table, as a starting point for shape animations.

The Animation Table functions

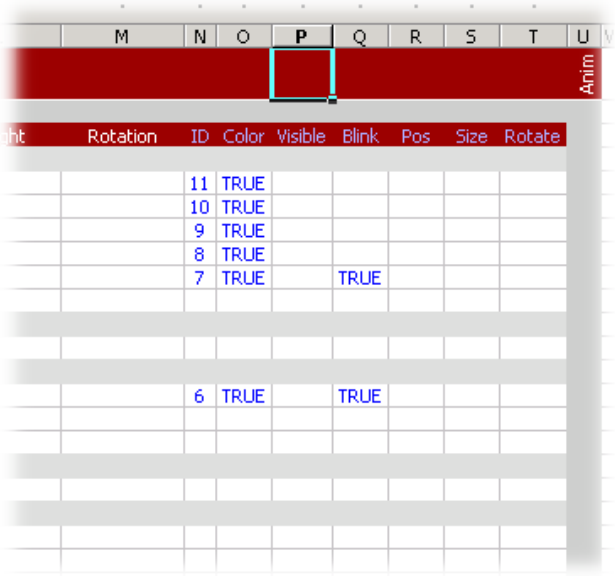


Figure 8-3: Animation Table Shape *functions*

Columns at the **right**-hand side of the *Animation Table* optionally contain worksheet **functions** to actually animate the shape. If a certain animation is not desired, the appropriate column should remain empty.

Note the **color usage** of cells: Constants have a *black* font, items from the tag database are colored *green*, and worksheet functions have a *blue* color. We recommend using these font colors throughout your application, to be able to distinct constants, functions and real-time data easily in your project.

Although it is not a good engineering practice to create display pages that resemble a "Las Vegas Christmas tree", it is possible to combine various worksheet functions for a single shape object: you may change colors, the position or size of a blinking or invisible object.

The following columns are available in the Animation Table for shape animations:

- **ID**
  This column contains a worksheet function with an animation ID. To create a new shape animation, a new animation ID should be created using this function. The ID is automatically generated in the table by **eXLerate** by means of the `exShapeID(…)` worksheet function. There are two arguments to this function: the shape object name, e.g. "valve_11", and a *recalculation trigger*, which causes Excel to recalculate the current worksheet function at startup.

- **Color**
  The color column contains optionally the `exShapeColor(…)` worksheet function. Arguments are the object ID (from the previous column), the FillColor, and the LineColor, from the $D and $E columns at the left hand size of the Animation Table.

- **Visible**
  A shape may be made invisible or visible again using the `exShapeVisible(…)` worksheet function. Arguments to this function are the object ID, and a visible/invisible flag. An object is made visible if this argument yields non-zero.

- **Blink**
  A shape may be set blinking with a steady frequency of about 0.5 Hz using the `exShapeBlink(…)` argument. Arguments to this function are the object ID, the line color, the blink color, and an interval. Applying a non-zero value to the interval column will set the shape object blinking during runtime.

- **Pos**
  There are various positioning worksheet functions to be added to the Animation Table: `exShapeLeft(…)` to set the x-position of the shape, `exShapeTop(…)` to set the y-position of the shape, or `exShapePos(…)` to set both the x and y position. Coordinates on the display start at (0,0) at the left top corner of the display, incrementing to about 10000 to the right at cell-column IV, and to 68000 to the bottom at cell-row 65535. Arguments of these functions include the object ID, and the x-position/x-position or both properties.

- **Size**
  There are various sizing worksheet functions to be added to the Animation Table: `exShapeWidth(…)` to set the width of the shape, `exShapeHeight(…)` to set the height of the shape, or `exShapeSize(…)` to set both the width and height of the shape. Arguments of these functions include the object ID, and the height/width or both properties.

- **Rotate**
  The rotation angle may be set at various shapes (but not all). The rotation angle is defined from 0º to 360º. The `exShapeRotate(…)` function may be used to set the rotation angle of a shape.

The Animation Color Table

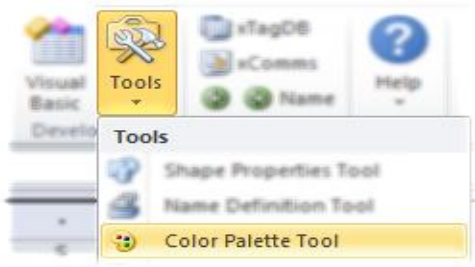
Shapes and ranges may be animated using a predefined palette of colors. These colors are stored in the 'rColorTable' range of the 'xTables' sheet.

AA	AB	AC	AD	AE	AF
Color Table					
rColorTable					
ID	C	R	G	B	Comment
0		0	0	0	Fixed system color: 'xBlack'
1		255	255	255	Fixed system color: 'xWhite'
2		255	0	0	Fixed system color: 'xRed'
3		0	255	0	Fixed system color: 'xGreen'
4		0	0	255	Fixed system color: 'xBlue'
5		255	255	0	Fixed system color: 'xYellow'
6		255	0	255	Fixed system color: 'xMagenta'
7		0	255	255	Fixed system color: 'xCyan'
8		0	0	0	Design color black
9		255	0	0	Design color red
10		0	153	0	Design color green
11		0	0	204	Design color blue

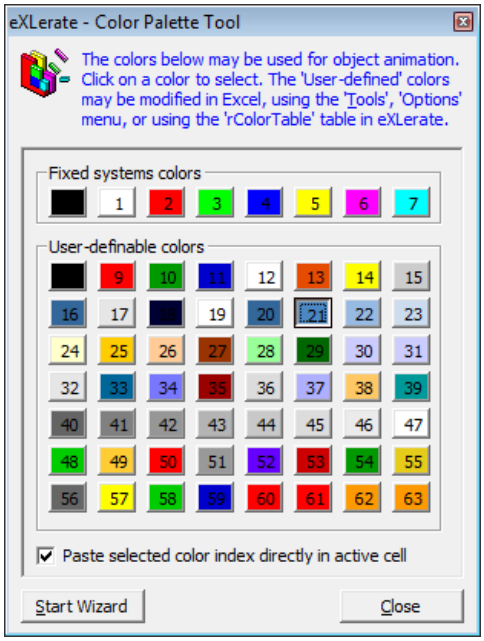
Figure 8-4: Color table

The first 8 colors in the *Color Table* are fixed and cannot be changed. The remaining 56 colors may be changed to any color. These colors can be changed by setting the '**R**ed', '**G**reen' and '**B**lue' to a value between 0 and 255. After changing a value the Color Wizard must be run, which makes the colors active in Excel.

The color-indexes from the *Color Table* can be easily inserted into a cell using the Color Palette Tool (Ctrl+L):



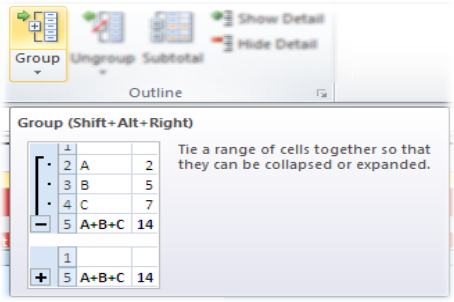
When a color-button is pressed, the index of that color is automatically inserted into the current cell.



Adding animations to your project

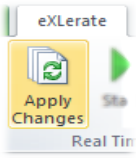
To add animated objects to your project, the only thing you have to do is to draw or import the shape object in your worksheet, and add the entry at the Animation Table, preferably at the appropriate shape class.

For example, if you have a valve object, you might want to add the animation for your object at the 'valves' class. If the targeted class does not exist yet, you may easily add your class to the Animation Table.



Use the Excel 'Group' and 'Ungroup' options from the 'Data' ribbon to create groups for your classes. Creating groups highly structures your application, especially when the amount of animations exceeds a single page on your computer's monitor.

Remember that just the name of a shape object has a project global scope, i.e. you may have various objects "valve_11", one on every worksheet, and each a different shape, but there has to be only one shape entry in the Animation Table to animate all objects.



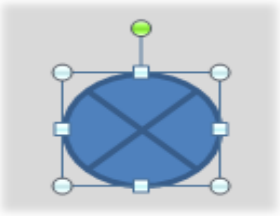
After completion of the appropriate property columns at the left of the table and worksheet functions at the right hand side of the table, you should choose the 'Apply Worksheet Changes' option from the ribbon to let **eXLerate** digest your table changes.

The next time that you start communications in runtime mode, the objects are animated according to the Animation Table.

Example:

In the animations worksheet itself, we will create and animate a new shape using 3 simple steps.

- **Create a new shape**
  Insert a new shape in worksheet 'xAnimations', just for testing. For example,



create a summing junction shape from the Excel shapes library:

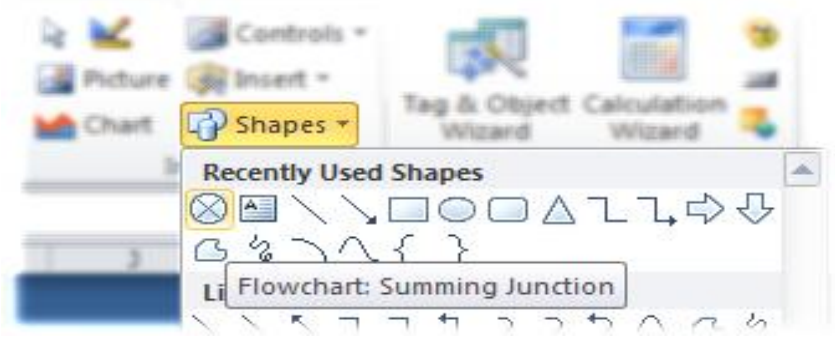


Figure 8-5: Inserting a summing junction shape

Please note that when holding the <Shift>-key while resizing the object with the mouse, both x- and y-dimensions are resized equally, so in our case the circular shape remains circular. The shape gets a default name assigned by Excel, for example: 'FlowChart: Summing Junction 1'.

Figure 8-6: Default and given name of the new shape

Change the name of the shape in: 'MyShape' using the Name box on the right hand side of the display page.

Add the shape to the animation table

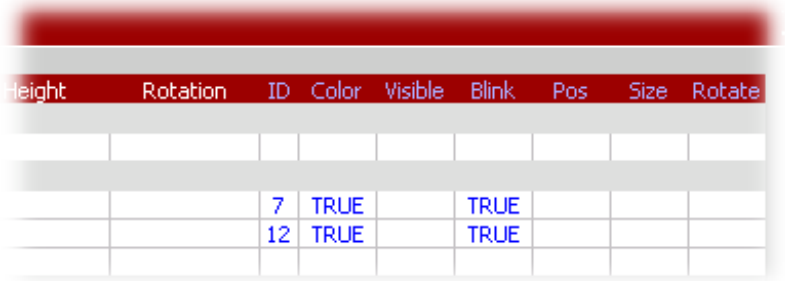
We will create a new entry in the Animation Table, by setting its name in the 'Shape' column, and additional properties, as follows:

Class	Shape	Lookup	FillColor	LineColor	Visible	BlinkCol	Interval	Left	Top
Valves									
Lamps									
	LED_1		35	0		2	1		
	MyShape		52	0		35	1		

Figure 8-7: The shape properties at the Animation Table

The 'FillColor' is set to '52', which is in our case orange, 'LineColor' is 0 (fixed color black), and as blinking color ('BlinkCol') we will choose '35', which is dark red in our case. The color numbers are retrieved using the Color Palette Tool, opened with <Ctrl-L>. When a certain color is clicked at this tool, the value may be inserted directly into the cell. The 'Interval' field is set to '1', to enable blinking.

- **Fill-in the actual animation functions**
  For now, we will only register the shape in **eXLerate**, set the fill color, and let the shape blink at a 2 Hz interval. Simplest is to copy the required columns from other shapes using <Ctrl-C>, <Ctrl-V>. The functions should be looking as follows:



Height	Rotation	ID	Color	Visible	Blink	Pos	Size	Rotate
		7	TRUE		TRUE			
		12	TRUE		TRUE			

Figure 8-8: Shape functions to be filled-in

The ID as returned from the worksheet function '`exShapeID(…)`' in the example above is '12', but could be any number, which depends on the number of already existing entries of the table. Other functions used are: '`exShapeColor(…)`' to set the fill color of the shape, and '`exShapeBlink(…)`', which is responsible for blinking the shape object.

To test the newly created shape, start real-time updating.

Chapter 9 - Menu navigation

Introduction

Menu navigation is an important aspect in application development, because it is an important aspect for the look-and-feel of the application and therefore deserves the necessary attention, and yet it is usually a time-consuming, lesser interesting part for an application engineer to actually implement.

Figure 9-1: Function-key button bar in eXLerate

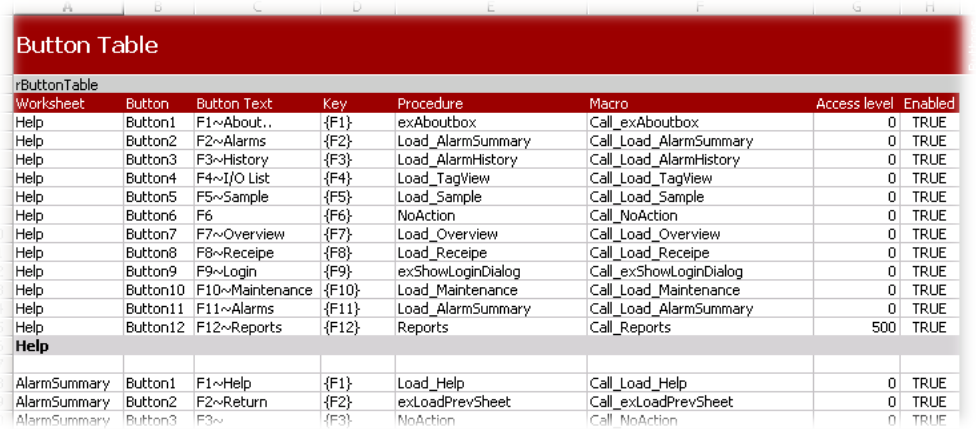
eXLerate automates the generation of display navigation in your project. A function key button-bar may be defined once, in the template display page, with which **eXLerate** is able to generate the page-specific navigation behavior for each display page.

Page-specific functionality is stored in the *Button Table*, which is the first table on the 'xTables' worksheet. In the Button Table the navigation buttons <F1>…<F12> may be defined, but in addition other shape objects with which a user is able to select another display page. In fact, any VB-macro can be activated by clicking on the shape. Such definitions may be defined in the Button Table as well.

This section explains how to use the available functionality in **eXLerate** to automatically add display page navigation to your application, and how to use the *Button Table*.

The Button Table

The Button Table looks as follows:



Button Table

rButtonTable

Worksheet	Button	Button Text	Key	Procedure	Macro	Access level	Enabled
Help	Button1	F1~About..	{F1}	exAboutbox	Call_exAboutbox	0	TRUE
Help	Button2	F2~Alarms	{F2}	Load_AlarmSummary	Call_Load_AlarmSummary	0	TRUE
Help	Button3	F3~History	{F3}	Load_AlarmHistory	Call_Load_AlarmHistory	0	TRUE
Help	Button4	F4~I/O List	{F4}	Load_TagView	Call_Load_TagView	0	TRUE
Help	Button5	F5~Sample	{F5}	Load_Sample	Call_Load_Sample	0	TRUE
Help	Button6	F6	{F6}	NoAction	Call_NoAction	0	TRUE
Help	Button7	F7~Overview	{F7}	Load_Overview	Call_Load_Overview	0	TRUE
Help	Button8	F8~Receipe	{F8}	Load_Receipe	Call_Load_Receipe	0	TRUE
Help	Button9	F9~Login	{F9}	exShowLoginDialog	Call_exShowLoginDialog	0	TRUE
Help	Button10	F10~Maintenance	{F10}	Load_Maintenance	Call_Load_Maintenance	0	TRUE
Help	Button11	F11~Alarms	{F11}	Load_AlarmSummary	Call_Load_AlarmSummary	0	TRUE
Help	Button12	F12~Reports	{F12}	Reports	Call_Reports	500	TRUE
Help							
AlarmSummary	Button1	F1~Help	{F1}	Load_Help	Call_Load_Help	0	TRUE
AlarmSummary	Button2	F2~Return	{F2}	exLoadPrevSheet	Call_exLoadPrevSheet	0	TRUE
AlarmSummary	Button3	F3~	{F3}	NoAction	Call_NoAction	0	TRUE

Figure 9-2: The Button Table layout

As with most configuration tables in eXLerate, the table starts with a table identifier ('rButtonTable') 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 button table has as many rows as there are buttons and/or keys for the application, and various columns, each with a specific function.

The last button definition of each worksheet just contains the worksheet name in a grey colored row. Although not required, it is maintained to create a clear distinction between display pages in the table.

The following columns are present:

- **Worksheet**
  This column contains the worksheet at which the button is to be present, e.g. 'Help', or 'Trending'. Button definitions at the same worksheet are grouped together. When there are key-definitions global for all worksheets, the name of the worksheet remains empty, for example a common print key <Ctrl-P>. This keystroke is used to print the current display page on the printer.

- **Button**
  The name of the button object, e.g. 'Button1', 'MyButton', at which the procedure will be linked to.

- **Button Text**
  The text to be placed on the button. A new line will be inserted for every tilde ('~') character in the text. The first line of the text will be placed **bold**, the rest of the text in plain font.

Key
The keystroke to be associated with the button. Special keys are defined between curly brackets '{' and '}'. The following special keys are available:

Key	Literal key text to be entered
Backspace	{BACKSPACE} or {BS}
Break	{BREAK}
Caps Lock	{CAPSLOCK}
Clear	{CLEAR}
Delete or Del	{DELETE} or {DEL}
Down Arrow	{DOWN}
End	{END}
Home	{HOME}
Ins	{INSERT}
Left Arrow	{LEFT}
Num Lock	{NUMLOCK}
Page Down	{PGDN}
Page Up	{PGUP}
Right Arrow	{RIGHT}
Up Arrow	{UP}
F1 Through F15	{F1} through {F15}

Table 9-1: Key codes to be entered

Key codes may be also used in combination with the <Alt>-, <Shift>- and/or <Ctrl> keys, in which case the character key is preceded with the following key state character codes:

Key state	Character	Character name
Shift	+	Plus
Ctrl	^	Caret
Alt	%	Percent sign

Table 9-2: Key state codes

We strongly recommend only using function keys and keys with either <Shift>, <Ctrl>, and/or <Alt>, and to avoid using regular keys for macro calling.

- **Procedure**
  The name of the macro or procedure to be invoked when the button is pressed can be entered at this column. **eXLerate**'s *Button Wizard* automatically creates a VBA procedure: 'Load_{display}' for each display page, e.g. when the user inserts a display: 'MyDisplay', the *Button Wizard* creates a VBA procedure (also referred to as a '`Sub`') in VBA module: '`modButtons`' with the name: '`Load_MyDisplay`'. Other procedures, such as built-in dialogs of **eXLerate** may be defined as well, such as the user login dialog ('`exShowLoginDialog`'), or a user-defined VBA macro.

- **Macro**
  The name of the actual macro is entered at this 'wrapper' column by the Tag & Object wizard; a user should not edit this column directly. This additional column is required, because in Excel, only macros defined in modules can be used with buttons. For example, the user login dialog is not a macro, but an **eXLerate** procedure ('`exShowLoginDialog`'), and needs therefore an additional 'wrapper' macro to invoke this dialog from a command button.

- **Access Level**
  At this column, the minimum access code that a user should own before the button may be activated, is entered. For example, when a 'Guest' with security level '10' should not be allowed to acknowledge an alarm, the button to acknowledge an alarm with should contain a higher access level than '10'.

- **Enabled**
  This column may be used to programmatically enable/disable the action of the button. When this field yields to TRUE, the button is enabled. When the expression yields to FALSE, clicking this button has no effect.

Button Wizard

From the **eXLerate** ribbon, start the Button Wizard, as follows:

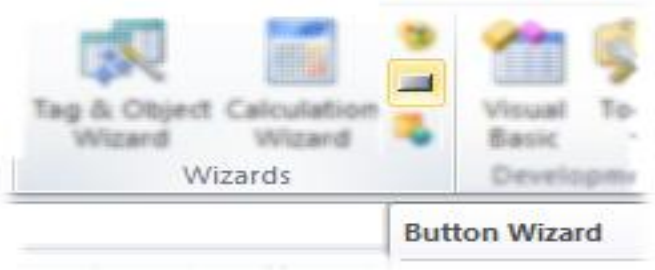


Figure 9-3: Invocation of the Button Wizard

The Button Wizard will be showed, as follows:

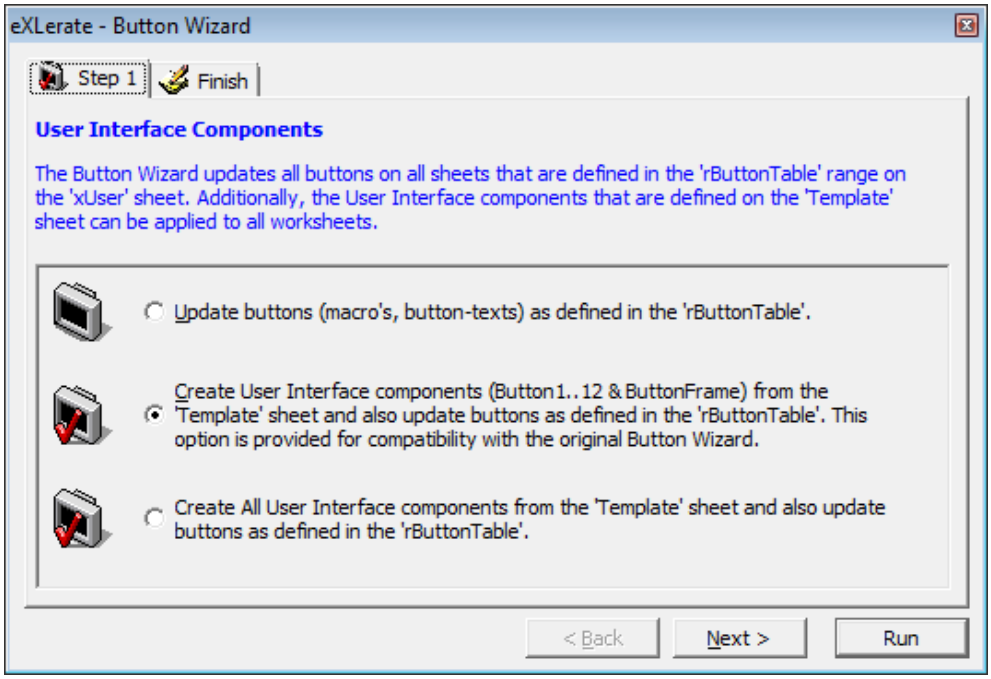


Figure 9-4: Button Wizard, step 1

The Button Wizard contains 3 options:

- **Update buttons**
  When this option is selected, all the existing buttons on all the displays are updated to reflect the contents of the 'rButtonTable' range. No buttons are created/removed when using this option.

- **Create User Interface components (Button1..12)**
  When this option is selected, the 'ButtonFrame' including the buttons 'Button1..12' are copied from the 'Template' sheet to all the display sheets.

- **Create All User Interface components**
  When this option is selected, all user interface components (shapes) from the 'Template' sheet are copied to all the display sheets.

When this step has been properly setup, press 'Finish' to actually create the User interface components, and create the menu navigation functionality, as follows:

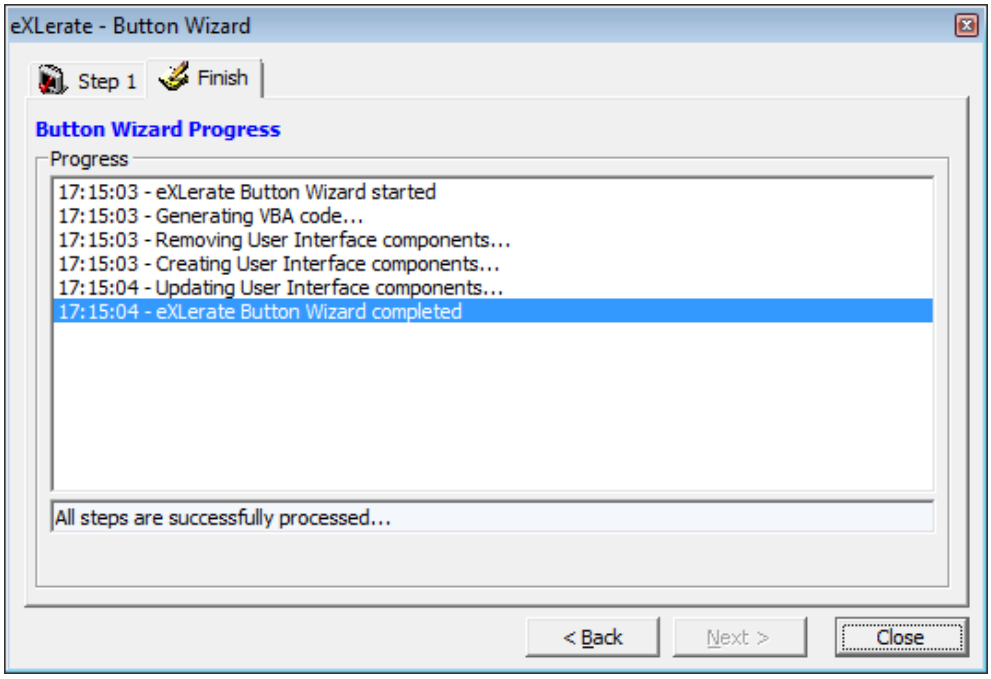


Figure 9-5: Button wizard generates menu navigation items

The Button Wizard has copied the selected user interface elements to all display pages, and filled for each display page the buttons from the Button Table definition.

You are now able to test the menu navigation when switching to RunTime/Preview mode from the **eXLerate** menu. You may press <ESC> to end the preview mode, and revert to Design mode.

When the Button Wizard is invoked another time, the options per step are remembered from the last time the wizard was invoked, so simply press 'Run', or <Enter> to actually recreate the button-bar.

Chapter 10 -Cell editing

Introduction

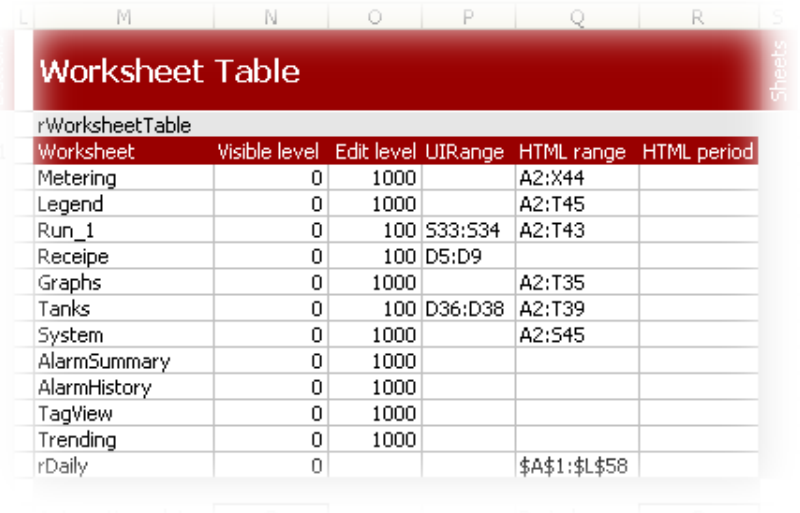
Editing cells is an important aspect in application development which often does not get the necessary attention. Checking whether the user has entered a correct value, in the proper format, within the proper limits is usually a time-consuming, lesser interesting part for an application engineer to actually implement.

***eXLerate** enhances the standard editing behavior of Excel in multiple ways, for example it contains an advanced editing mechanism which does not temporarily freeze display updates while editing is in progress. Also, user feedback is provided using user-friendly popup balloons instead of message boxes.*

Editing Range

Before a cell can be edited in *runtime* or *preview* mode, it must comply to the following rules:

- **The cell must be included in the UI range of the worksheet**
  Before a cell can be edited, it must be included in the UI range of the worksheet. This range can be changed in the Worksheet Table of the xTables worksheet as shown below. Also the Edit level is used to verify whether the current user has sufficient access rights to edit the cells.



Worksheet Table

rWorksheetTable

Worksheet	Visible level	Edit level	UIRange	HTML range	HTML period
Metering	0	1000		A2:X44	
Legend	0	1000		A2:T45	
Run_1	0	100	S33:S34	A2:T43	
Receipe	0	100	D5:D9		
Graphs	0	1000		A2:T35	
Tanks	0	100	D36:D38	A2:T39	
System	0	1000		A2:S45	
AlarmSummary	0	1000			
AlarmHistory	0	1000			
TagView	0	1000			
Trending	0	1000			
rDaily	0			$A$1:$L$58	

Figure 10-1: Worksheet Table

- **The cell is unlocked**
  When *runtime* or *preview* mode is activated, all display worksheets are locked. In order to edit a cell, the cell itself must be unlocked.

The Editing Table

In eXLerate, the properties of editable cells of all display pages are stored in a single table, the *Editing Table*.

The Editing Table is typically stored in worksheet 'xEditing', but may be located elsewhere.

In order to configure additional properties for editable cells (e.g. data type, min/max validations), an entry in the Editing Table must be created for it.

The Editing Table has various rows, one for each editable cell and a number of columns, one for each editing property.

Columns at the **left**-hand side of the table contain actual editing **properties**, i.e. the edit property values for the editable cell, as follows:

	A	B	C	D	E	F	G	H	I
1	Editing Table								
2	Object names		Edit Properties						
3	Class	Cell	Target	Target Type	Group	Edit Type	Edit Type Alert	Min	Max
4	**Fields**								
5		Tanks!D36	Tanks!E36	2		2	Only numerical values are allowed!	100	200
6		Tanks!D37	Tanks!E37	2		3		4	
7		Tanks!D38	Tanks!E38	2		4			
8		Receipe!D5	Receipe!E5	2	Receipe	1			
9		Receipe!D6	Receipe!E6	2	Receipe	1			
10		Receipe!D7	Receipe!E7	2	Receipe	1			
11		Receipe!D8	Receipe!E8	2	Receipe	1			
12		Receipe!D9	Receipe!E9	2	Receipe	1			
13		Run_1!S33		1		5	Please specify the date in the following format: %FORMAT%		
14		Run_1!S34		1		6	Please specify the time in the following format: %FORMAT%		
15									

Figure 10-2: Editing Table *properties*

The following *properties* are available:

- **Class**
  A class is like a group, where the application developer is able to create groups containing identical editable cells, such as alarm limits, gas components etc. This is an optional property (may be left empty).

- **Cell**
  The address of a cell to edit. This can be either a text containing a value such as "Tanks!D36" or a `exCellProperties(…)` formula. The `exCellProperties(…)` worksheet function can be used to obtain properties of a cell such as the address. For instance,

`'=exCellProperties( Tanks!D37, xAddress, xAutoRecalc)'` returns the address of Tanks!D37, which is in fact "Tanks!D37". The big advantage of using the `exCellProperties(…)` function is that Excel treats the first argument as a cell reference and not a text. Therefore, when copying such a cell to the next row, the reference will automatically be updated (e.g. Tanks!D37 → Tanks!D38).

- **Target**
  The target to which the edited value should be written upon accepting the value. This can be useful when the editable cell identifies for instance, an alarm limit or a communication tag. In case of a alarm limit, the target should identify the name of the alarm limit and the **Target Type** should be *xTargetAlarmLimit.* See **Target Type** for a complete list of supported targets. No value needs to be specified when **Target Type** contains *'xTargetNone'*. Multiple targets can be configured by separating them using the ',' character (e.g. "Tanks!E36**,**Tanks!E37")

- **Target Type**
  This property forms a pair with the **Target** property. The following target types are supported:

Target Type	**Target**	**Description**
xTargetNone (1)	Not used	No target is configured.
xTargetCell (2)	Address of the cell in the form *'Sheet!A1'* or a named range.	Upon accepting the new value is written to the cell specified in **Target**.
xTargetName (3)	Named range which refers to a cell	Upon accepting the new value is written to the named range specified in **Target**.
xTargetComm (4)	Name of communication tag	Upon accepting the new value is written to the communication tag specified in **Target**.
xTargetAlarmLimit (5)	Name of the alarm (e.g. "xMOV11.HAlarm")	Upon accepting the new value is written to the alarm limit specified in **Target**.
xTargetAlarmDead band (6)	Name of the alarm (e.g. "xMOV11.LAlarm")	Upon accepting the new value is written to the alarm deadband specified in **Target**.
xTargetAlarmDelay (7)	Name of the alarm (e.g. "xMOV11.SAlarm")	Upon accepting the new value is written to the alarm delay specified in **Target**.

Table 10-1: Target Types

- **Group**
  Identifies the group to which an editable cell belongs. This property affects the target update behaviour. When no group is specified, the target is updated immediately after the user accepts a new cell value. When a group is specified, an explicit call to the function `'exAcceptEditGroup(…)'` must be made to accept all the edited values in a group. This mechanism can be used to accept one or more values by clicking on a button. For example, when editing a gas composition, the individual components should be accepted altogether. See **Accepting Edit Groups** for a more detailed description on this subject.

- **Edit Type**
  The type of data which can be entered by the user. The following edit types are supported:

Edit Type	Description
xWholeNumber (1)	Only whole numbers are accepted. Non whole numbers are automatically rounded to whole numbers.
xDecimal (2)	Only numbers are accepted.
xText (3)	All input values are accepted.
xList (4)	The user must select a value from a pre-defined list. The list can be configured using the `'exEditType(…)'` function. A list consists of a 2 dimensional array with 2 columns and 1 or more rows. The first column contains the value which is copied to the optional target and the second column contains the display item which is shown to the user. See **Edit Lists** for a more detailed description on this subject.
xDate (5)	Only date values are accepted. By default the Windows Regional Settings short date format is used for editing dates. It is however possible to use a custom date-format, by specifying the date format (e.g. yyyy/mm/dd) as the fourth argument to the `'exEditType(…)'` function. See **Date and Time Edit Formats** for a more detailed description on this subject.
xTime (6)	Only time values are accepted. By default the Windows Regional Settings time format is used for editing times. It is however possible to use a custom time-format, by specifying the time format (e.g. hh:mm:ss) as the fourth argument to the `'exEditType(…)'` function. See **Date and Time Edit Formats** for a more detailed description on this subject.

Table 10-2: Edit Types

- **Edit Type Alert**
  The text that is shown when the user enters a value that is not conform

the specified edit type. For instance, when the user enters a value '19b' into a cell which only accepts whole numbers, an alert is shown that the entered value is not valid. When no specific alert is specified, the default alert text *"Value '%INPUT%' does not match the specified datatype"* is displayed. The following special keywords can be used in the alert text:

Keyword	Description
%INPUT%	Is replaced by the currently edited value.
%FORMAT%	Is replaced by the in-use date or time format when the edit-type is *xDate* or *xTime*.

Table 10-3: Edit Type Alert Keywords

- **Min**
  Minimal allowed value. In case the edit-type is *xText*, this value identifies the minimal required length of the text. This is an optional property (may be left empty).

- **Max**
  Maximum allowed value. In case the edit-type is *xText*, this value identifies the maximum allowed length of the text. This is an optional property (may be left empty).

- **Min / Max Alert**
  The text that is shown when the user enters a value that is outside the min/max limit. When no specific alert is specified, the default alert texts *"Value '%INPUT%' should be greater or equal to '%VALIDATION%'"* and *"Value '%INPUT%' should be less or equal to '%VALIDATION%'"* are displayed. The following special keywords can be used in the alert text:

Keyword	Description
%INPUT%	Is replaced by the currently edited value.
%VALIDATION%	Is replaced by the validation limit which has been exceeded.
%FORMAT%	Is replaced by the in-use date or time format when the edit-type is *xDate* or *xTime*.

Table 10-4: Validation Alert Keywords

- **Access Level**
  Minimal security level which is required to edit the cell. This is an optional property (may be left empty).

- **Enabled**
  Enables or disables cell editing completely. Set this value to 'FALSE' to completely disable editing for the specific cell.

Accepting Edit Groups

Accepting Edit Groups is all about copying the content of an editable cell to its target, either by clicking a button or running a macro.

- **Edit the cell(s)**

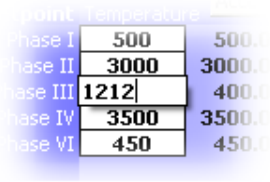


Figure 10-3: Editing a cell

- **Click the button**

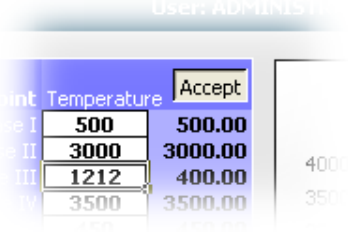


Figure 10-4: Clicking a button

- **See how the new value (1212) gets accepted**

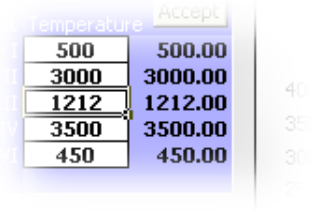
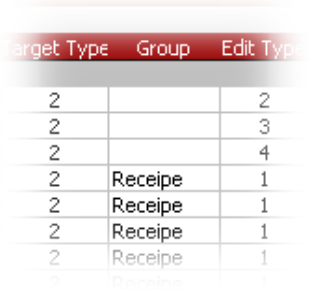


Figure 10-5: See how the new value gets accepted

The following is required to make an editable cell copy its value into another cell, alarm limit, communication tag, etc…, by clicking a button:

- **Specify a group for the cell in the Editing Table**



rget Type	Group	Edit Typ
2		2
2		3
2		4
2	Receipe	1
2	Receipe	1
2	Receipe	1
2	Receipe	1

Figure 10-6: Groups in the Editing table

- **Run the Button Wizard**

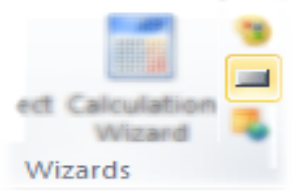


Figure 10-7: Button Wizard

This will automatically generate a VBA function for accepting the edit group (e.g. AcceptEditGroup_Receipe(…))

- **Add an entry to the Button Table**

Button12	F12~Reports	{F12}	Reports	Call_Reports
Button 53	Accept		AcceptEditGroup_Receipe	Call_AcceptEditGroup_Receipe

Figure 10-8: Button Table

Add an entry to the button table and specify **AcceptEditGroup_<edit group>** as procedure name.

See the previous chapter on how to configure the Button Table.

In some cases it is preferable to perform a check on all the values before accepting them. For instance, in case of a gas composition it would be useful to verify that the individual components add up to 100%. In this case a macro can be used to perform this check and accept the values using the `'exAcceptEditGroup(…)'` function.

Example:

```
Sub AcceptGasComponents()

     ' Verify that gas composition is 100%
     …

     ' Accept the new gas composition
     exAcceptEditGroup "Receipe"

End Sub
```

The Editing Table

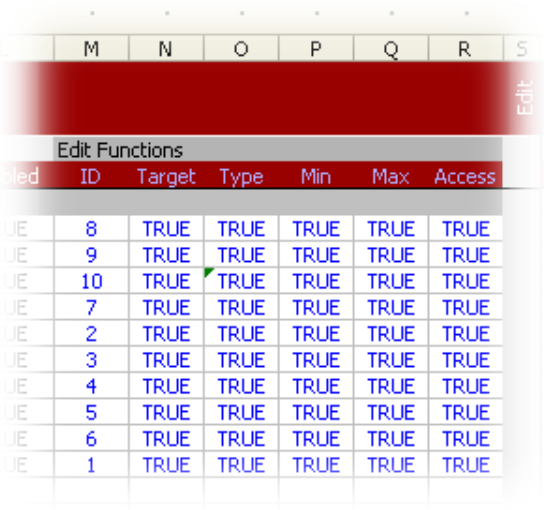


Figure 10-9: Editing Table *functions*

Columns at the **right**-hand side of the *Editing Table* optionally contain worksheet **functions** to actually set the editing properties. If a certain property is not desired, the appropriate column may be deleted.

The following columns are available in the *Editing Table* for editable cells:

- **ID**
  This column contains a worksheet function with an edit ID. To create a new editable cell, a new edit ID should be created using this function. The ID is automatically generated in the table by **eXLerate** by means of the `exEditID(…)` worksheet function. There are two arguments to this function: the cell address, e.g. "Tanks!D36", and a *recalculation trigger*, which causes Excel to recalculate the current worksheet function at startup.

- **Target**
  The target column contains optionally the `exEditTarget(…)` worksheet function. The content of a cell may be copied to a target using this worksheet function. Arguments to this function are the object ID (from the previous column), the Target, Target Type and a Group. If a group is specified, the value will be copied to the target after a call to the function `exAcceptEditGroup(…)`.

- **Type**
  The type column contains optionally the `exEditType(…)` worksheet function. If no type is specified, the cell is assumed to be a text string (*xText*). Arguments to this function are the object ID, the Type and the Edit Type Alert. If the type is a *xList*, the fourth parameter refers to a cell range containing a list. See **Edit Lists** for a detailed description on this subject. If the type is a *xDate* or *xTime*, the optional fourth parameter refers to a date or time format. See **Date and Time Edit Formats** for a detailed description on this subject.

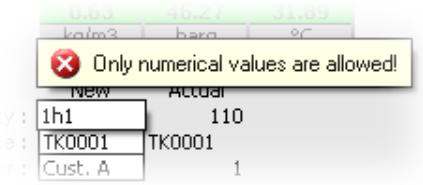


Figure 10-10: Type validation during operation

- **Min / Max**
  The min and max columns are validation columns. Both the columns contain optionally the `exEditValidation(…)` worksheet function. Arguments to this function are the object ID, Validation Type (e.g. *xMinimum* or *xMaximum*), the Validation Value and the Validation Alert Text. By default both the min and max validations use the same alert text. It is however possible to configure different validation texts for the min and max validations.

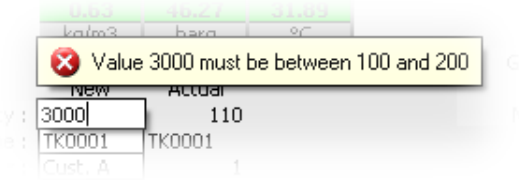


Figure 10-11: Min / max validation during operation

- **Access**
  The access column can be used to disable editing for a single cell. The worksheet function `exEditAccess(…)` can be used to set the access rights for the cell. Arguments to this function are the object ID, Access Level and

the Enabled status. This function extends the edit level security defined in the *Worksheet Table* so that it is possible to define security for a single editable cell. This function may be omitted when the edit level security defined in the *Worksheet Table* is sufficient.

Edit Lists

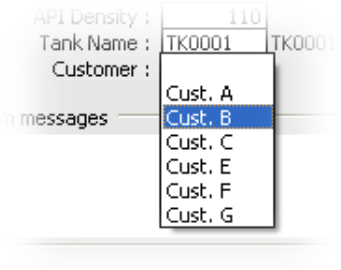


Figure 10-12: Edit Lists

Instead of letting the user type the specific value, it is also possible to present the user with a list of items as shown above.

The contents of a list can be configured in the Lists Table in the *xEditing* worksheet. The List Table is located on the **right**-side of the *xEditing* worksheet.

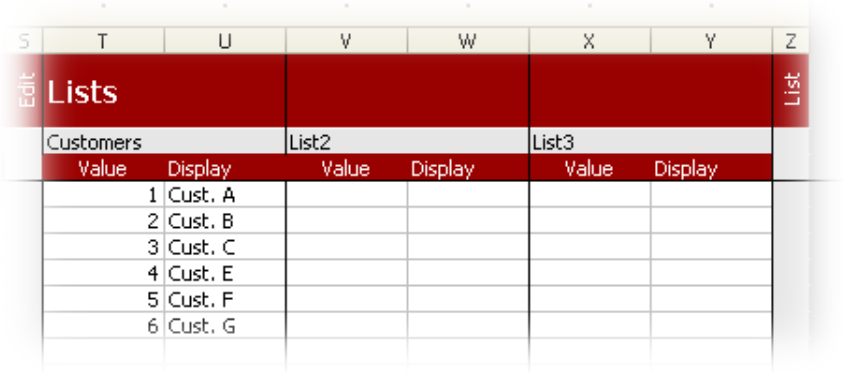


Figure 10-13: Edit List Table

New list entries can be made by inserting new columns into the list table. It is not obligated to define lists specifically in the list table, it is simply provided as a location for storing lists.

A list always consists of two columns and one or more rows. The first column consists of value items and the second column consists of display items. The display items are shown to the user when he or she tries to pick an item from the list. The value that is copied to the target when the cell is accepted is the item value.

For the next example, assume that the list is used, which is mentioned in the figure above. If the user selects "Cust. A" from the list, the target would receive the value "1"

The following actions are required for using lists:

- **Set the edit type to '*xList*'**

Figure 10-14: Edit Type *xList*

- **Configure a list in the List Table or somewhere else**

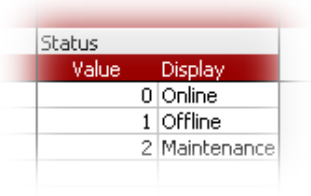


Figure 10-15: Configure list

- **Modify the *exEditType* worksheet function so it uses the list**

  Add a fourth argument to the `exEditType(…)` worksheet function which refers to the list. The range should consist only of the actual values, not the column headers.

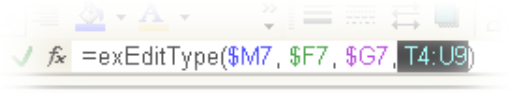


Figure 10-16: Modify exEditType worksheet function

Date and Time Edit Formats

By default, editable cells of type *xDate* or *xTime* use the Windows Regional Settings formats when entering new values. For instance, when the Windows

Regional Settings format for short-dates is "yyyy/mm/dddd", the user should enter the date in that particular format and the following mask is displayed:

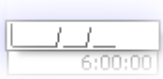


Figure 10-17: Example date mask

It is however also possible to specify a custom date or time format. A date or time format can be configured by modifying the `exEditType(…)` worksheet function. The fourth argument of this function is the custom format. If this argument is omitted the Windows Regional Settings format is used.

Figure 10-18: Custom date/time format

The following format specifiers can be used for date formats:

Specifier	Description
y, yy	Year without century, as a decimal number (00-99)
yyy, yyyy	Year with century, as a decimal number
m, mm, mmm	Month as a decimal number (01-12)
d, dd	Day of the month, as decimal number (01-31)

Table 10-5: Date format specifiers

The following format specifiers can be used for time formats:

Specifier	Description
h, hh	Hour in 24-hour format (00-23)
m, mm	Minute as decimal number (00-59)
s, ss	Second as decimal number (00-59)

Table 10-6: Time format specifiers

In some cases, it will be preferable to use the format of the cell itself. In that case the worksheet function `exCellProperties(…)` can be used to obtain the format of the cell. The format of a cell can be obtained in the following manner:

Example:

=exCellProperties(Run_1!S33, xFormat, xAutoRecalc)

results in:
"m/d/yyyy"

Chapter 11 -Reporting

In this chapter, you will learn how to add reports to your application, and how reports are generated in an application.

Reports may be generated manually, or automatically at predefined intervals and at your own events.

Reports and display pages may be also published as HTML files, so a user is able to monitor your application in a web browser.

Finally, reporting in a real-time HMI application isn't a nightmare anymore, it's fun! Ever dreamt of creating a beautiful & advanced report in less than 5 minutes? Continue reading!

Report generation

Introduction

In **eXLerate** you are able to automatically generate a report. A report in **eXLerate** is a worksheet containing data references to the application to which the report is attached.

The report with the references to the application is called a report *template*, because when the report is 'generated', a copy of this worksheet template, containing only –updated- values without references to the application is saved to an external workbook.

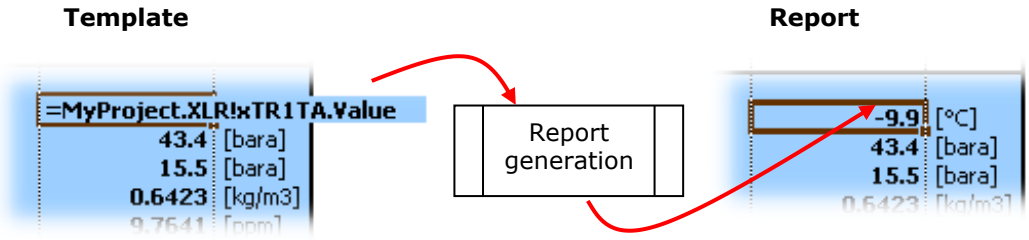


Figure 11-1: References converted to values during report generation

The generated values-only copy of the report worksheet template, which in fact is the report *output* worksheet, may be copied to a specific workbook, to a specific worksheet.

Example

The 'MyReport' workbook contains three worksheets, setup as three report templates: a daily report (worksheet tab: 'DailyReport'), a monthly report (worksheet tab 'MonthlyReport'), and a snapshot report (worksheet tab 'SnapShot'). The daily report will be automatically printed, every day, at 08:00 in the morning. The daily reports are stored in a monthly workbook ('DayReports

YYYYMM.XLS'), with up to 31 worksheets ('DD') per month workbook. The monthly reports, which are also automatically printed on the first day of each month, are stored in a yearly workbook ('MonthReports YYYY.XLS'), where each month is stored as a worksheet ('MM').

In the report output files, no functions or references to the original project workbook are present, just the numbers.

Report locations

For every application shortcut, the (base) report output path can be specified. Whenever a report is generated, it is placed in this location.

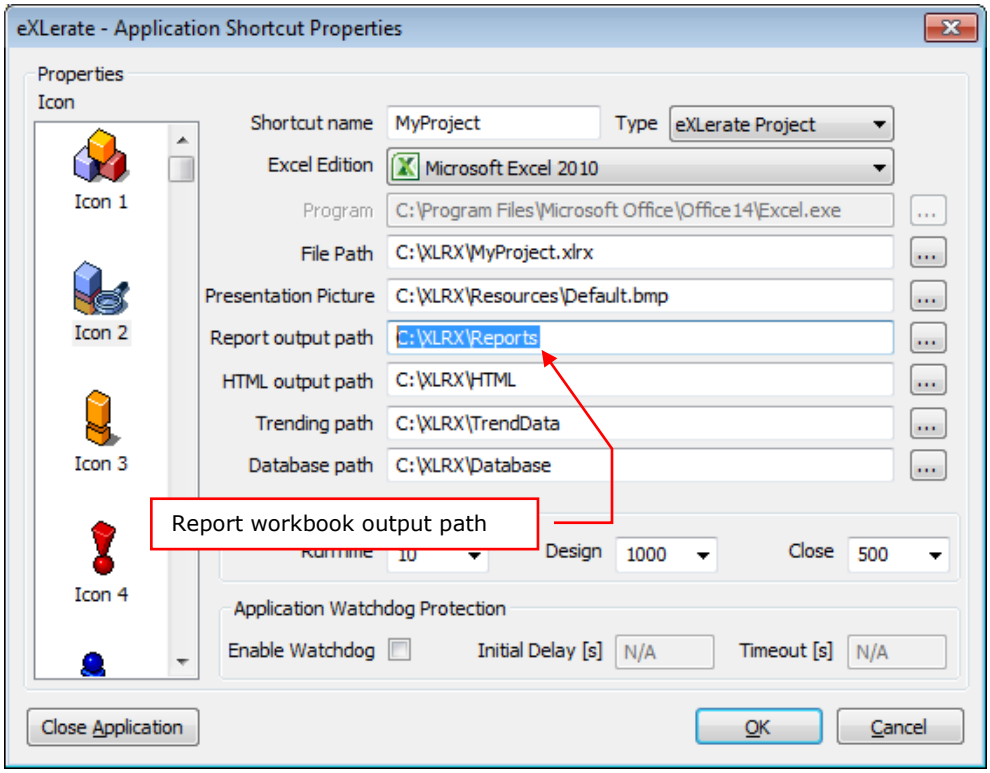


Figure 11-2: Report directory specification

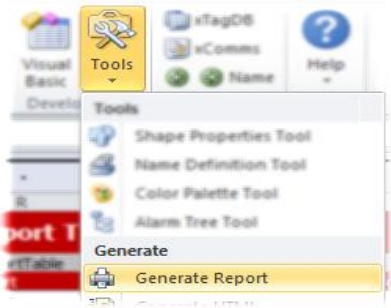
Report generation

Reports may be generated manually, by operator request, or automatically, for example every day, or every hour, or at a specific event, controlled from VBA. The way report generation takes place is called a *report event*.

When the report is printed automatically, it may be optionally marked as 'Original'. Copies of the report are marked as 'Reprint'.

Multiple copies of a report may be printed. Report output workbooks may be optionally set to Read-only. All report events of the application are defined in the *Report Table*.

Reports may be manually generated from the **eXLerate** ribbon, after which the following dialog appears:



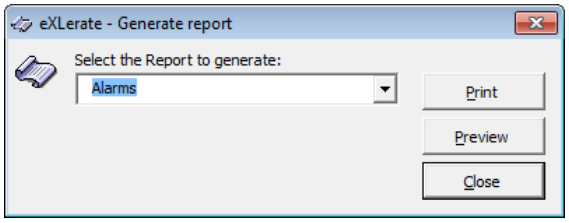


Figure 11-3: Generate report pop-up dialog

In the pull-down list box, the report to be generated and printed may be specified. It is called a snapshot report, because it is not generated automatically by **eXLerate**.

Whenever a report is being generated or printed, a progress window is shown:

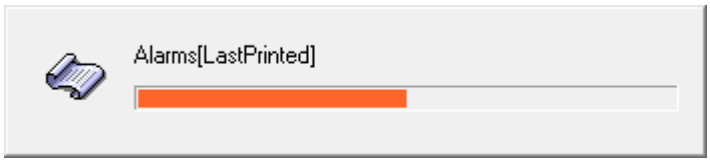
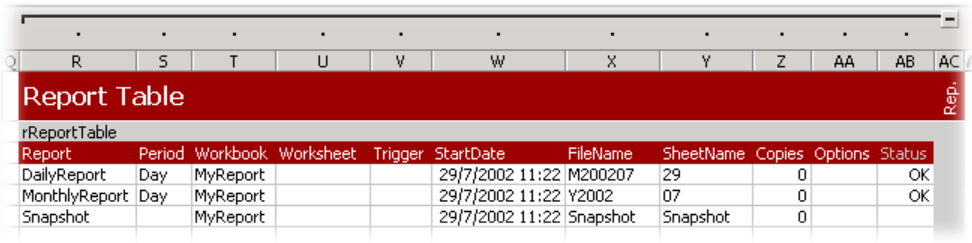


Figure 11-4: Report progress dialog

When using the 'exGenerateReport(…)' and 'exReprintReport(…)' functions it is also possible to 'silently' generate/print reports in order to hide the progress window. Automatically generated reports are always 'silent' and thus don't show the progress dialog.

The Report Table

Report events of the application are defined in the *Report Table*, which resides in the 'xEvents' worksheet. The *Report Table* looks as follows:



Report Table

rReportTable

Report	Period	Workbook	Worksheet	Trigger	StartDate	FileName	SheetName	Copies	Options	Status
DailyReport	Day	MyReport			29/7/2002 11:22	M200207	29	0		OK
MonthlyReport	Day	MyReport			29/7/2002 11:22	Y2002	07	0		OK
Snapshot		MyReport			29/7/2002 11:22	Snapshot	Snapshot	0		

Figure 11-5: Report Table layout

As with most configuration tables in **eXLerate**, the table starts with a table identifier ('rReportTable') 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 Report Table has as many rows as there are reports for the application, and various columns, each with a specific function.

The following fields are supported:

- **Report**
  This is the logical name of the report.

- **Period**
  This is the name of the interval event, which is associated to the report. For example, a daily report is linked to the 'Day' event.

- **Worksheet**
  This is the name of the worksheet containing the report template. The template is a single worksheet, and consists of one or more pages.

- **Trigger**
  This field contains the *Period* number of the corresponding interval event, at which the report is generated. When *Trigger* is left empty, or contains an expression yielding to a value < 0, it is generated at every interval event. When the expression at this field yields to a number >=0, it is assumed to be the corresponding *Period* at which the report is to be generated.

- **StartDate**
  This field is automatically filled in by **eXLerate**, and contains the date of the next time a report should be generated. It may be used to construct the file- and worksheet name of the report output. When the report is generated, it is updated automatically with the current date.

- **FileName**
  The name of the report output file is specified with this field, for example: ="MyReport" & YEAR(*StartDate*) The current file will be overwritten in case the target file already exists. You may also specify sub-directories here like this: "Daily\MyReport".

- **SheetName**
  This field contains the worksheet tab name under which the generated report in the workbook is stored at. *StartDate* may be used to construct the sheetname.

- **Copies**
  The number of printed copies is specified with this field, starting from '0'. When left empty, 1 copy is printed to the report printer.

- **Options**
  This field is used to specify additional printing options. Currently, only a '1' may be used to make the report output workbook read-only. Other values are reserved for future usage. If not specified, the report output workbook file is read/write.

- **Password**
  This field is used to specify an optional password which is used to protect the sheets in the output report.

- **Status**
  When the report is generated, it status is set, being: '0', when no errors have been encountered during the report generation, or '1', when the report generation has failed.

- **Workbook (Obsolete)**
  Older versions of eXLerate supported storing reports in external workbooks. This feature is no longer supported and therefore this field no longer has any purpose.

Creating report contents

When you have created a report template, you may create the report as if it were a regular display page. A report may contain graphical objects, shapes, bitmaps, values, expressions, constants, and everything else that Microsoft Excel supports.

To insert a value from the application into your report, you might want to use the simple '=' sign, and point to the value in your application, as in the example below:

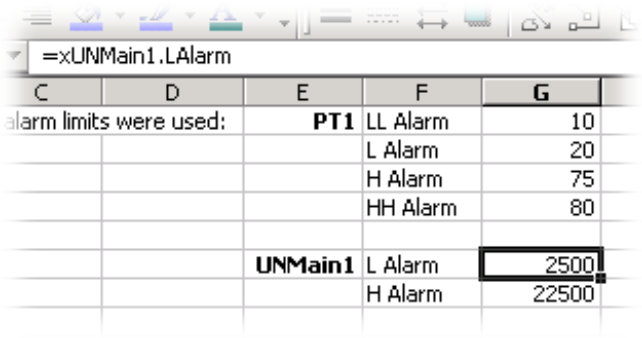


Figure 11-6: Inserting a parameter from the application in the report

When your have created periods in an application, for example an hour period containing 24 periods, for a daily report, you have done so using the *Interval Table*. In the tag database, for each tag value to be reported in that hour, you have created a *latch* and/or *average* in the 'P_Hour' column. If this is not the case, or you have skipped reading section '*Intervals and Periods*' of in this volume, you might want to add such definitions to your application at this time.

In the 'rDaily' worksheet, a report is worked out for you containing 24 periods of an hour. The 'rDaily' report contains the following section:

Hour	TT1 [°C]	PT1 [bara]	PT2 [bara]	DT1 [kg/m3]	AT1 [ppm]	USLVFR1 [m³/h]	USNVFR1 [Nm³/h]	UPHNF1 [Nm³]
07:00	24.9	44.6	29.7	0.6	0.6	54.6	98470.8	203610.0
08:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
09:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10:00	2.1	42.9	4.1	1.0	1.0	54.5	97587.7	208732.6
11:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12:00	-3.2	47.7	41.5	0.8	0.8	50.1	92906.9	207470.3
13:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
18:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
20:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
21:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
00:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
02:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
03:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
04:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
05:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
06:00	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 11-7: Report section containing 24 hours of data

In this report, at the left '**Hour**' column, the hours are given using the array formula '`{=xPeriod.Hour.rTimeScale}`' for 24 rows.

The data columns in this table section of the report are created using the array formula: '`{=xTT1.Hour.rPeriods}`', for example for tag 'TT1'.

The Tag & Object wizard automatically creates these arrays for you when the *Interval Table* and the tag database are processed (which takes automatically place once you run the Tag & Object wizard).

All you need to do is setup the *Interval Table* properly, and define for which tags you need (averaged or latched) data, and **eXLerate** generates all required data for you!

Report print date

When a report is generated, all formulas are replaced by values. Thus when using the worksheet function '=NOW()' on a report, it will result in the date/time of when the report was generated.

In order to put the print date/time on a report, the following syntax can be used:

="=NOW()"

When this formula is converted into a value, the end result is:

=NOW()

When a report is re-printed or previewed, the NOW() formula is re-calculated and the current date/time is placed on the report.

Advanced reporting in your application

There is a VBA interface available with which reports may be generated from user-defined dialogs, for example to allow an operator to print certain specific reports. An example of such a user-defined dialog is given below:

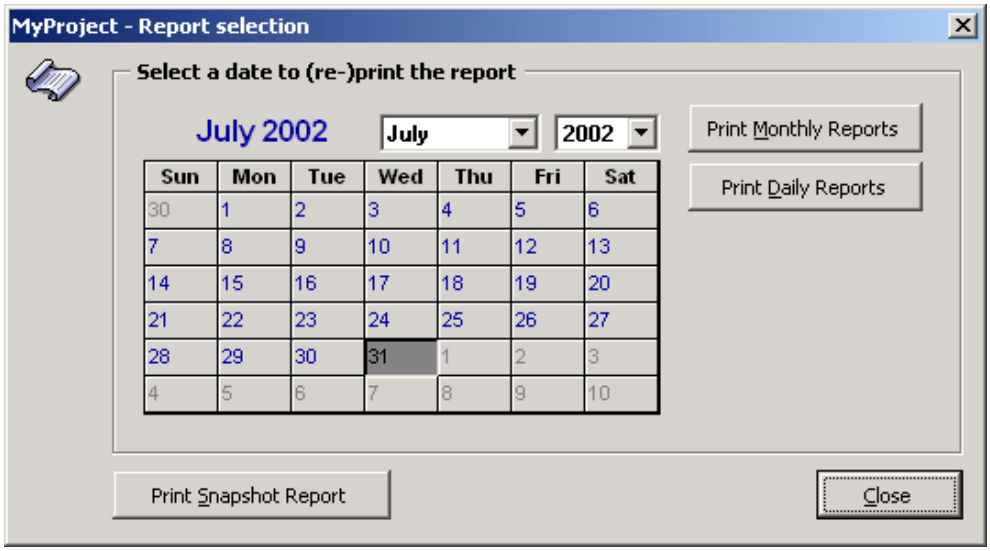


Figure 11-8: Sample of a user-defined print dialog

The VBA-code 'behind' the '**Print Daily Reports**'-button looks as follows:

```
Private Sub btnDaily_Click()
  'Local data
  Dim strReport As String
  Dim strTab As String

  Cal1.ValueIsNull = False

  'Current day?
  If (Cal.Year = Year(Now)) And (Cal.Month = Month(Now)) And
     (Cal.Day = Day(Now)) Then

    exGenerateReport "DailyReport", nCopies:=1,
                                PrintMode:=exPrintModeSnapshot
    Exit Sub
  End If

  strRep = "M" & Cal.Year & Format(Cal.Month, "00") & ".XLS"
  strTab = Format(Cal.Day, "00")
  exReprintReport strReport, strTab, False
End Sub
```

Figure 11-9: Code fragment for a dedicated user interface

The VBA sample code above demonstrates two API functions that are available for report generation, `exGenerateReport(…)`, and `exReprintReport(…)`:

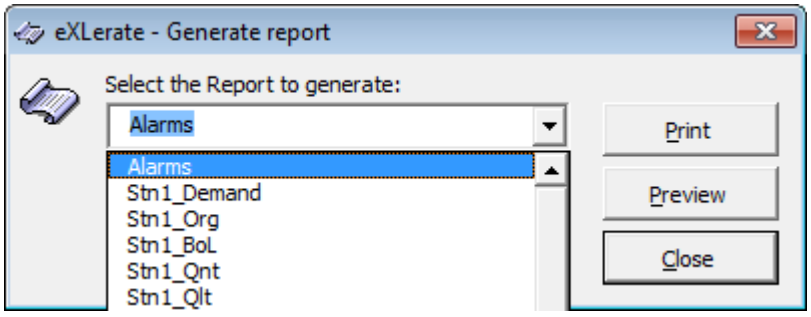
- **`exGenerateReport(…)`**
  This function generates a report output, from the report template, and prints 1 copy of the report. The *PrintMode* flag in the example above sets the variable: '*xPrintMode*' in the report template. The '*xPrintMode*' variable contains either values '–1', '0', or '1', for 'Snapshot', 'Original', and 'Reprint', which may be printed on the report for fiscal integrity. In Excel, you can format these three values as a text, simply by its Cell Number format.

- **`exReprintReport(…)`**
  This function in the example above reprints an existing report output worksheet ('*strTab*') from the report workbook ('*strReport*'). Reports may be optionally previewed before actually reprinted, which is disabled in the example above with the *False* value of the last argument.

The VBA sample code above contains a few lines of code to detect if the selected date on the dialog's calendar control, represented with the variable: '*Cal*' in the sample code is the current date. If this is the case, then a report is actually generated, because it was not automatically generated just yet. When the date from '*Cal*' points to another date, it is assumed to be an already existent report, after which a reprint of the selected report is issued. Other buttons on the dialog are processed likewise. In the sample project, the dialog is included, which may be used for further analysis.

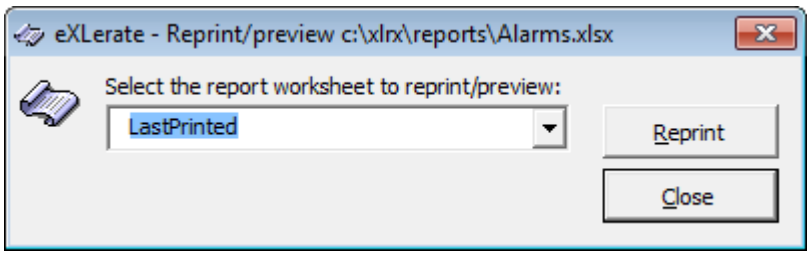
Other available API functions for reporting include:

- **exShowPrintDialog(…)**
  This function programmatically displays the pop-up dialog as displayed in: *Figure 11-3: Generate report pop-up dialog*, on page 11-197.



- **exPrintCurrent(…)**
  This function prints the current worksheet to the printer. The entire worksheet is printed, or just the 'Print_Area' range when defined for the worksheet. In the sample application, this function is called when the user presses <Ctrl-P>.

- **exShowFileReprintDialog(…)**
  This function displays a dialog with which the user is able to select one of the available worksheet tabs of the specified workbook for reprinting.



  Example: `exShowFileReprintDialog "C:\XLRX\Reports\Alarms.xlsx"`

- **exShowBackupDialog(…)**
  This function displays a combined dialog which may be used for reprinting of an existing worksheet tab, or to additionally create a backup of the selected report files.

Restrictions & Guidelines

When creating a report template, there are some criteria which affect report generation.

Object positioning

When an image, chart, shape or any other type of object is placed on a report, the "Object positioning" property must be set to 'Move and size with cells'. Furthermore, the 'Print object' option must be enabled. If these settings are not correct, the object may not appear correctly or not appear at all on the generated report.

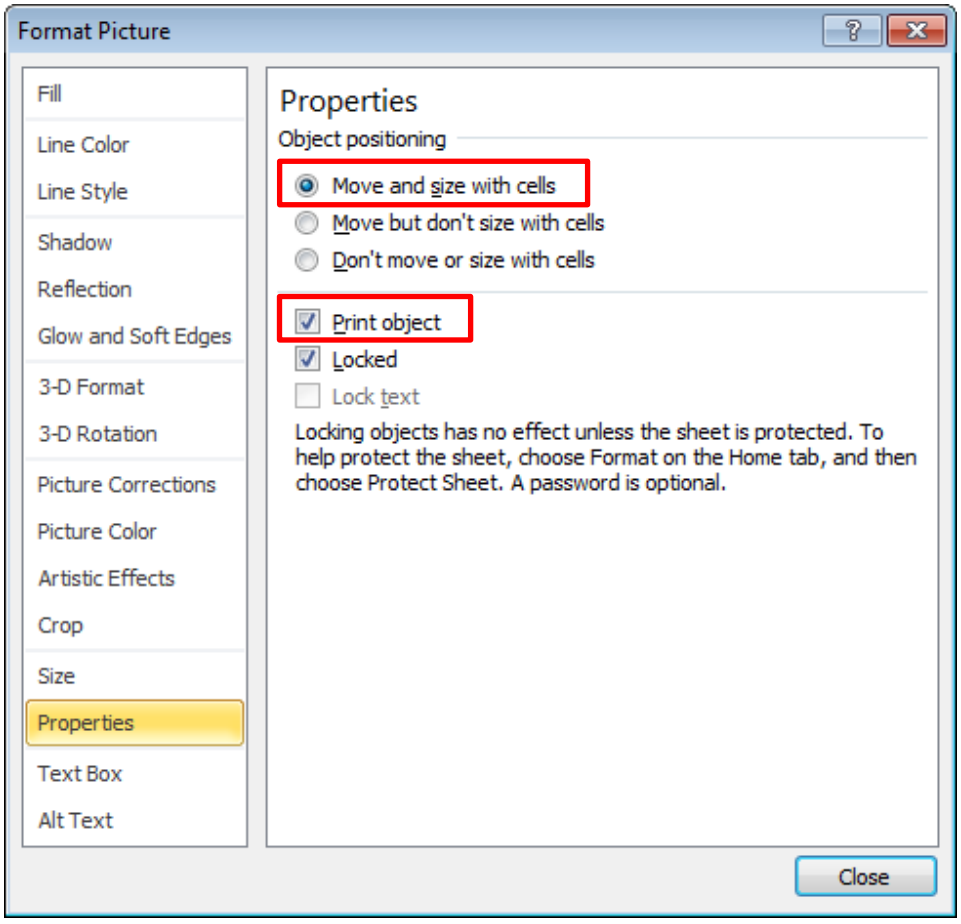


Figure 11-10: Report object properties

Hidden rows & columns

Although having hidden rows & columns is fully supported, it can have a negative effect on report file-size and performance. In general the following situation should be prevented:

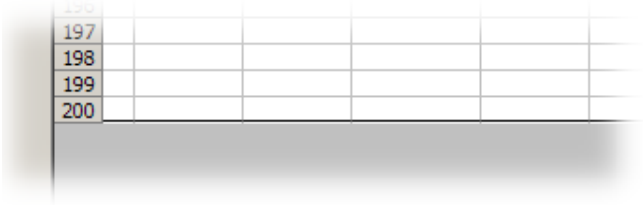


Figure 11-11: Hidden rows on reports

This would hide all rows below 200, but it also causes the report to have a minimal size of 200 rows. If for instance, only 80 rows are used for report data, the other 120 empty rows are still saved into the report file making it larger than necessary and it also takes longer to generate/reprint the report.

Sub-cell formatting (superscript, subscript)

A cell may only contain a single font or text color. For instance, the following cell contains multiple fonts and colors:

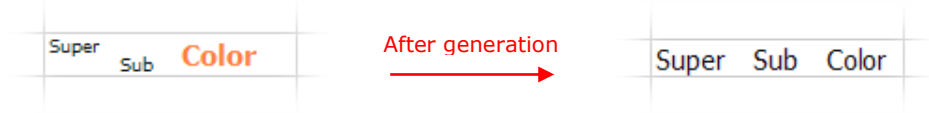


Figure 11-12: Unsupported cell formatting

After report generation the sub-cell formatting is removed.

For certain units like m^3, °F, etc… special symbols may be used to represent them properly. Use ribbon option "Insert->Symbol" to open the following "Symbol" dialog:

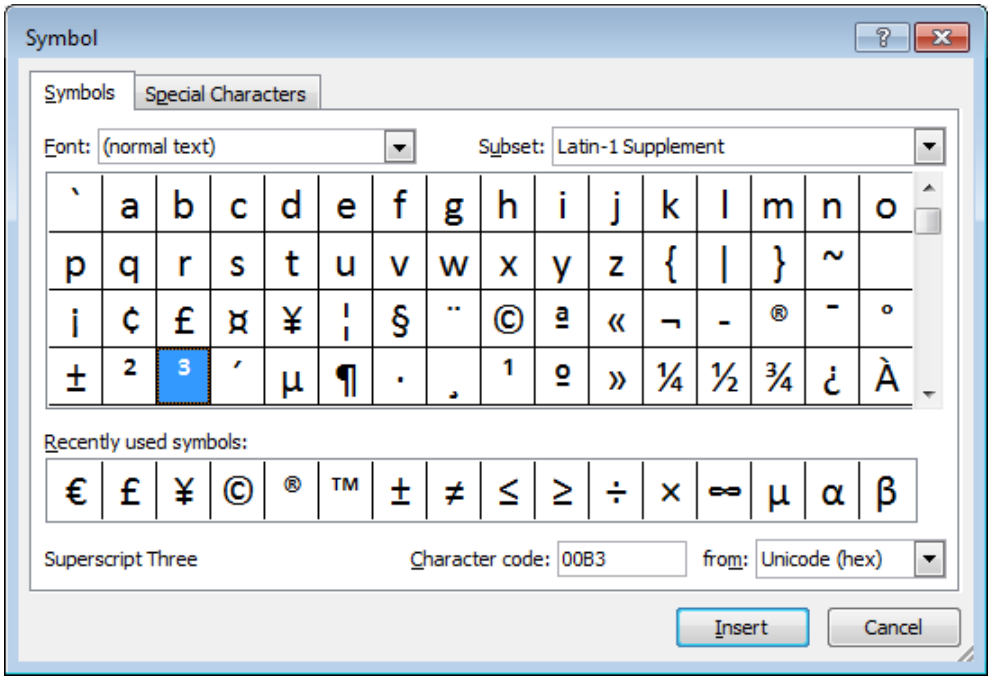


Figure 11-13: Symbol dialog

Select the symbol and press the "Insert" button to insert the symbol into your application.

Chart source data

Charts should always refer to source data on the same worksheet. If the source data is on an external worksheet, the following technique may be used for accessing source data on other worksheets.

1. Explicitly specify the print-area of a report.
2. Create cell references to the external source data on a place the report which is outside the print-area.
3. Change the source data of the chart so that it refers to the cells created in step 2.

Background images

Sheet background images are not supported.

Embedded objects

Embedded objects such as Visio drawings are not supported. These objects are automatically converted into shapes when placed on a report.

This page is intentionally left blank.

HTML page support

Introduction

Microsoft Excel has built-in support for HTML. A worksheet may be 'saved-as' HTML page, to be re-opened with the Internet Explorer.

No need to explain why eXLerate utilizes this basic feature to support automated generation of HTML pages in your application.

HTML formatted display pages allow programs like Internet Explorer and Netscape to view your application in web-pages, for example over a company's intranet.

HTML options

It depends on your own usage of HTML pages how eXLerate should be configured, if at all, for the automated generation of HTML pages.

eXLerate currently supports two options:

- **HTML Template generation**
  You are able to manually generate a HTML template from a worksheet from the eXLerate ribbon: 'Tools', 'Generate HTML…'.

- **Periodically updated HTML pages**
  eXLerate is able to periodically create and update HTML 'operator' pages from worksheets that have been configured for HTML support.

eXLerate filters the HTML output as generated by Excel, to remove the many unused Office based markup tags. This is additionally done to keep the generated HTML file size as small as possible.

Web options

There are many options for the generated pages, which can be found under the 'File', 'Save As', 'Tools', 'Web options' menu of Excel:

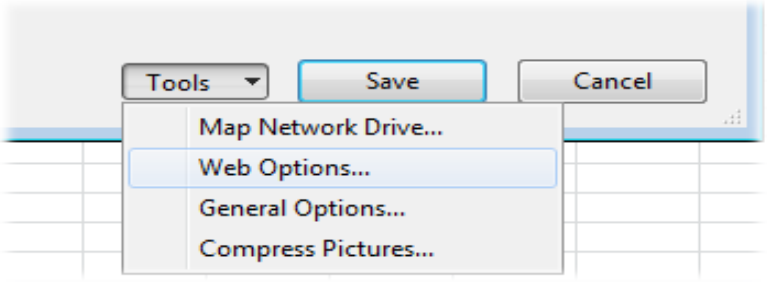


Figure 11-14: HTML file format options selection

When selected, the following dialog is presented:

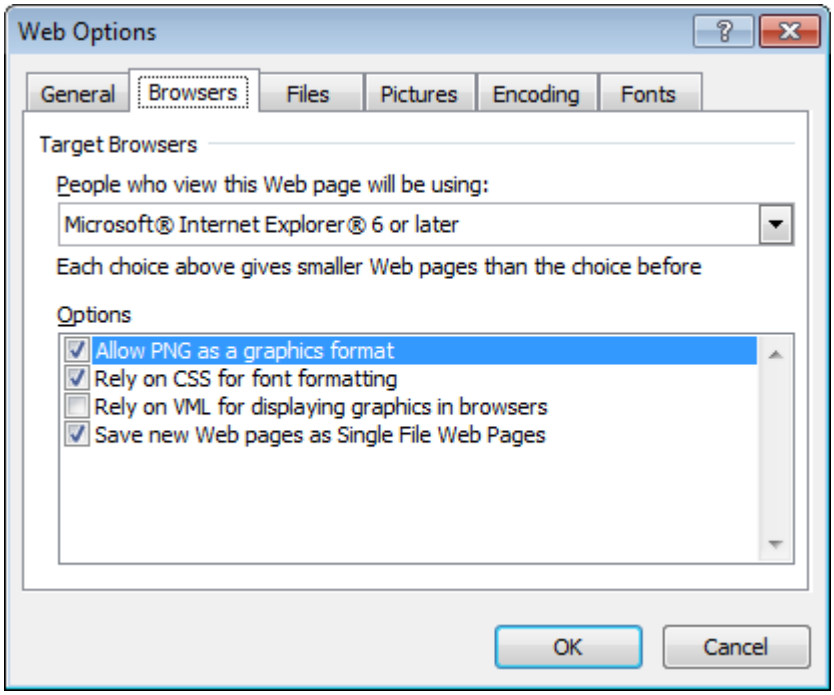


Figure 11-15: Excel Web Options dialog

Make sure that 'Allow PNG as a graphics format' is **enabled** and that 'Rely on VML for displaying graphics in browsers' is **disabled**, otherwise graphics may not appear correctly on most modern browsers.

The settings may be entered using this dialog, or alternatively from VBA, by settings properties and methods of the **DefaultWebOptions** object and the **WebOptions** object. You may want to check the Microsoft Excel documentation for detailed help on using and configuring these objects from VBA.

HTML templates

A HTML template file is a standard HTML file optionally containing special support tags and variables, which are to be used for subsequent processing of web-servers.

These support variables are defined at the tag database, using the '*.HTML*' field, which may be defined for each tag.

An example of a Web page in the 'MyProject' application is worksheet: 'WEB'.

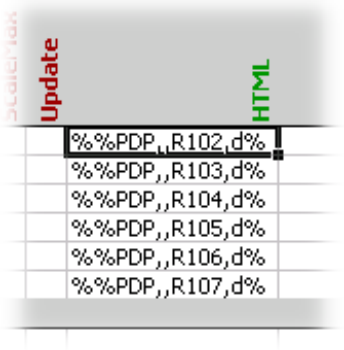


Figure 11-16: HTML fields in the tag database with HTML variables

In the WEB-page references may be made to all object names in the application, including the '.HTML' fields. The current value of these objects is saved in the HTML page output when the output is generated.

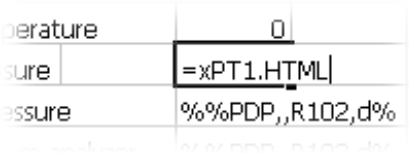


Figure 11-17: Using HTML fields in the 'WEB' worksheet

To tell eXLerate that a worksheet is a HTML page, the *Worksheet Table* must be properly extended with the tow fields 'HTML range', and 'HTML period':

Worksheet Table

Sheets

rWorksheetTable

Worksheet	Visible level	Edit level	UIRange	HTML range	HTML period
Help	0	3000			
AlarmSummary	0	1000			
AlarmHistory	0	3000			
TagView	0	3000			
Sample	0	3000			
Trending	0	3000			
Overview	0	3000			
Receipe	0	3000			
rDaily				$A$1:$L$58	4

Figure 11-18: Defining a worksheet as HTML page

The two fields: *HTML range*, as well as the *HTML period* define which and how HTML is added to your application. The first column contains the range, which is to be saved as HTML page, in this case $A1:$L58 of display sheet 'rDaily'.

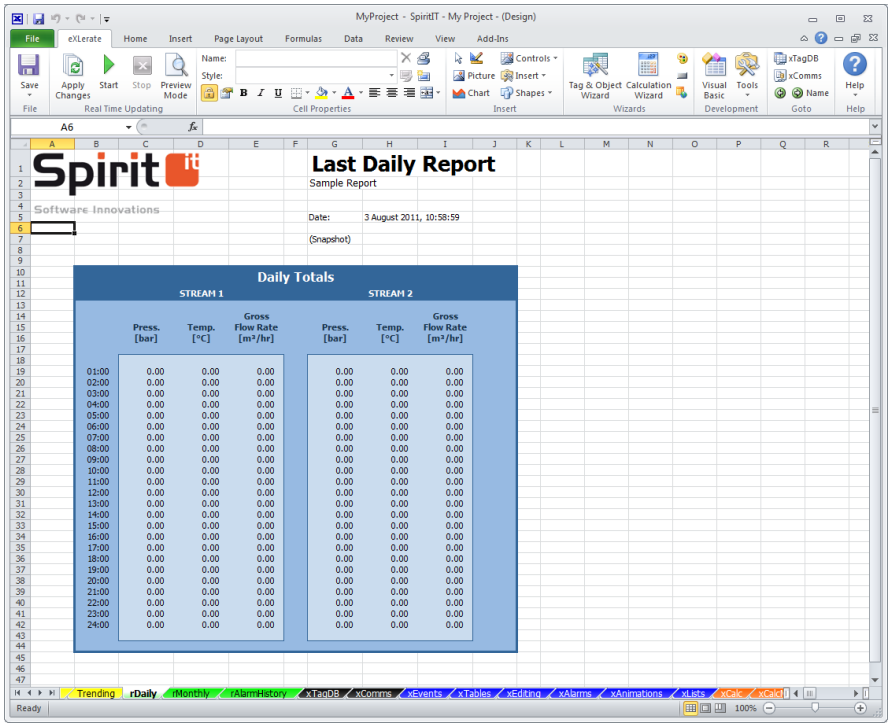


Figure 11-19: Sample worksheet: 'rDaily'

When the *Worksheet Table* is defined for the 'rDaily' display page, first select 'Apply Worksheet Changes', then 'eXLerate', 'Tools', 'Generate HTML…' to create a HTML page from the worksheet.

The resulting page is generated at the designated HTML output path, which is defined at the application shortcut in the Control Center.

When the file: "rDaily.HTM" is opened with a web browser, the following is displayed:

Figure 11-20: Sample of a generated web page

Periodically updated HTML pages

The above procedure generates a HTML page simply by filling in a worksheet name, and a corresponding range at the *Worksheet Table*, and a user may generate the Web page (template).

eXLerate is also capable of periodically updating the HTML file by entering the update interval event ID at the 'HTML Period' column of the *Worksheet Table*.

This period ID is an interval ID from the *Interval Table*. In the example, '4' is used, which corresponds with an update every hour. The updates take automatically place after start of real-time updating.