



Where Automation Connects.



ProSoft i-View

Mobile Process Monitoring and Control
Application

Version 2.0.2

September 29, 2011

USER MANUAL

Your Feedback Please

We always want you to feel that you made the right decision to use our products. If you have suggestions, comments, compliments or complaints about our products, documentation, or support, please write or call us.

How to Contact Us

ProSoft Technology

5201 Truxtun Ave., 3rd Floor

Bakersfield, CA 93309

+1 (661) 716-5100

+1 (661) 716-5101 (Fax)

www.prosoft-technology.com

support@prosoft-technology.com

Copyright © 2011 ProSoft Technology, Inc., all rights reserved.

ProSoft i-View User Manual

September 29, 2011

ProSoft Technology[®], ProLinx[®], inRAx[®], ProTalk[®], and RadioLinx[®] are Registered Trademarks of ProSoft Technology, Inc. All other brand or product names are or may be trademarks of, and are used to identify products and services of, their respective owners.

ProSoft Technology[®] Product Documentation

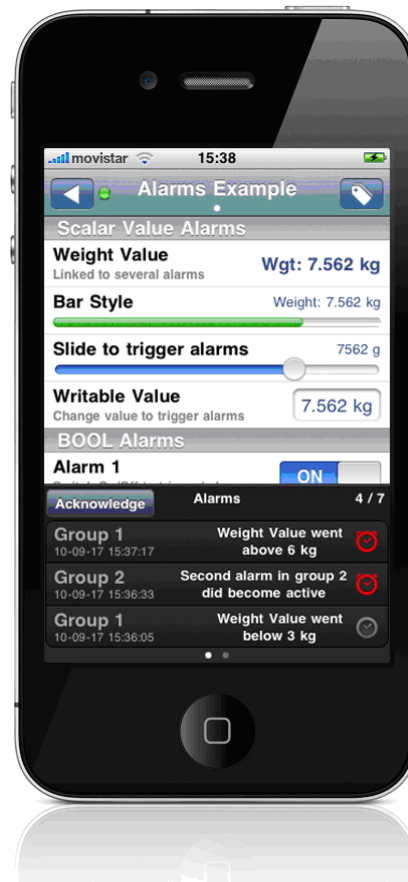
In an effort to conserve paper, ProSoft Technology no longer includes printed manuals with our product shipments. User Manuals, Datasheets, Sample Ladder Files, and Configuration Files are provided on the enclosed CD-ROM in Adobe[®] Acrobat Reader file format (.PDFs). These product documentation files may also be freely downloaded from our web site: www.prosoft-technology.com

Quick Start

What is ProSoft i-View?

ProSoft i-View is a native iPhone® and iPod touch® application for real-time monitoring of industrial PLC-based systems and processes.

It can be applied to building automation, industrial process control, mini-hydro power plants, water plants, and applications where reliable, instant access to real-time remote data is essential.



Main Features

- Local or remote access, and any number of concurrent PLCs
- Very fast, independent of project size, immediate connection and display
- Integer, floating-point, and boolean values
- Advanced String and Arrays Support
- Direct file import from Excel
- Configurable accounts with access levels
- Alarms, trend graphs
- Lookup texts, expressions
- Direct connection to PLCs and RTUs without servers
- TCP/IP-based security

How to Use ProSoft i-View in Five Simple Steps

- 1** Create an Excel sheet that specifies parameters for the variables to be controlled.
See Data Source Files (page 15).
- 2** Export or save into a CSV file.
- 3** Import the file created in Step 2 into ProSoft i-View. See File Import (page 63).
- 4** Set the PLC IP address in ProSoft i-View if you did not do so in Step 1.
See Network Settings (page 75).
- 5** Monitor process variable states and values from anywhere with mobile or WiFi access.

Contents

Your Feedback Please	2
How to Contact Us	2
ProSoft Technology® Product Documentation	2
Quick Start	3
What is ProSoft i-View?	3
Main Features	4
How to Use ProSoft i-View in Five Simple Steps.....	4
1 General Aspects	7
1.1 Supported Protocols	8
2 User Interface Elements	9
2.1 Tabbed Interface	10
2.1.1 Settings.....	10
2.1.2 File Server	10
2.1.3 Files	10
2.1.4 Connections	10
2.1.5 Home	10
2.2 Variables in ProSoft i-View.....	10
2.3 Home Tab Bar and Navigation Bar	12
2.4 Bottom Panel	13
3 Data Source Files	15
3.1 Data Sources Created in Excel	15
3.1.1 Specification of Variable Names (Column A)	16
3.1.2 Variable Types (Column B)	18
3.1.3 Variable Addresses (Column C).....	20
3.1.4 Attributes (Column D).....	24
3.1.5 Attribute Scope and Type.....	25
3.1.6 Tag Attributes	25
3.1.7 Global Attributes.....	33
3.1.8 Pages, Sections, Rows and Data Sources	36
3.1.9 Lookup Tables	37
3.1.10 Alarms	39
3.1.11 Comments in Data Sources	40
3.1.12 Specification of Communication Protocol.....	40
3.1.13 Expressions	46
3.2 Rockwell RSLogix 5000 as a Data Source Generator	60
3.2.1 Building a Project in RSLogix 5000.....	60
3.2.2 Exporting Controller Tags from RSLogix 5000.....	60
3.3 Editing Source Files in a Text Editor	61

4	File Import into ProSoft i-View	63
4.1	Source Files Supported by ProSoft i-View	65
4.2	Other Files Supported by ProSoft i-View	66
4.3	Custom Company Logo	67
5	Connections	69
6	User Accounts	71
6.1	Restrictions for Non-Administrator Users	72
6.2	Managing Accounts	73
7	Network Settings for Local Access	75
7.1	PLC Settings for Local Access	76
7.2	ProSoft i-View Settings for Local PLC Access	77
8	Network Settings for Remote Access	79
9	Security	81
9.1	Validation Codes	81
9.1.1	Custom Validation Tag	83
9.2	Background Task Processing	84
9.2.1	Keep Connected	84
9.3	Performance	86
10	Pre-installed Examples	89
10.1	DataTypesModbus.csv	91
10.2	DataTypesModbusWSource.csv	92
10.3	EIP_TAG_Examples.csv	93
10.4	EIP_PCCC_Examples.csv	94
10.5	PagesExampleModbus.csv	95
10.6	PagesExampleEIP.csv	96
10.7	StylesExampleModb.csv	97
10.8	StylesExampleEIP_PCCC.csv	98
10.9	AlarmsModbus.csv	99
10.10	AlarmsEIP_PCCC.csv	100
10.11	ColorfulControlsModbus.csv	101
10.12	ColorfulControlsEIP_PCCC.csv	102
10.13	Formula-ONE.csv	103
10.14	Document Revision History	104
10.14.1	Version 2.0.0	104
Index		105

1 General Aspects

In This Chapter

- ❖ Supported Protocols.....8

ProSoft i-View is presented in a tabbed interface. Every tab has its own role within the application and allows for different functions. With ProSoft i-View you can monitor data coming from various PLCs.

Generally, you will use the **SETTINGS**, **FILE SERVER**, **FILES**, and **CONNECTIONS** tabs during deployment stages.

The **HOME** tab shows real-time values of process variables in PLCs, and is the one you will use for normal monitoring. ProSoft i-View automatically goes to this tab on launch.

ProSoft i-View uses the concept of *Data Sources* (see *Data Sources* (page 15)) and *Connections* (see *Connections* (page 69)) to do its job. *Data Sources* contain the variable definitions and *Connections* represent links with PLCs. A *Source* is always associated with a *Connection*, but a single *Connection* can belong to several *Sources*.

ProSoft i-View also supports user accounts with access levels that can limit the ability to perform certain operations

1.1 Supported Protocols

ProSoft i-View gets variable values from industrial PLCs by polling them using TCP/IP industrial protocols. The following protocols are supported.

PROTOCOL NAME	SUPPORTED PLCs or Brands (Not exhaustive)	REMARKS
Modbus TCP	Schneider [®] Electric, Automation Direct [®] , Phoenix Contact [®] , Wago [®] ...	For communication with PLCs and RTUs using the Modbus TCP/IP specification
EIP/Native	Allen Bradley [®] ControlLogix [®] and CompactLogix [™]	Native CIP [®] communications using EtherNet/IP [™] Explicit Messaging
EIP/PCCC	Allen Bradley SLC [™] 500 and MicroLogix [™] controllers	PCCC commands (DF1) encapsulated in EtherNet/IP.

2 User Interface Elements

In This Chapter

❖ Tabbed Interface	10
❖ Variables in ProSoft i-View	10
❖ Home Tab Bar and Navigation Bar.....	12
❖ Bottom Panel.....	13

This section provides an overview of the main aspects of the ProSoft i-View user interface. It is not an exhaustive explanation. Most interface elements will be described as needed in later sections.

2.1 Tabbed Interface

ProSoft i-View uses the typical iPhone® tabbed interface to organize several aspects of the application. On each tab, a navigation interface is usually presented. Not all tabs are available to all user accounts and not all the options in a tab are accessible to all users. Five tabs are available.

2.1.1 Settings

Available to all users, but with restricted options for non-Administrator users. From this tab, Administrator users can create accounts, log in to a particular account, set several user interface behaviors, and specify default communication settings.

Non-Administrator users have all options disabled except the log in feature.

2.1.2 File Server

Only available to the Administrator user. Allows for managing, uploading, and downloading files into ProSoft i-View through an embedded Web Server.

2.1.3 Files

Only available to the Administrator user. Presents a list of source files containing tag definitions, and allows for placing a selection of the relevant ones for an application.

2.1.4 Connections

Available to all users, but with restricted options for non-Administrator users. Presents a series of entries, called connections, that represent actual links to PLCs. Within each connection, you can check the communication status and set validation codes. Relevant communication settings and parsing information is presented for source files related to each connection. You can also switch monitoring **ON** or **OFF** from this section.

Non-Administrator users are allowed only to switch monitoring **ON** or **OFF** from this tab. Connections are hidden for these users.

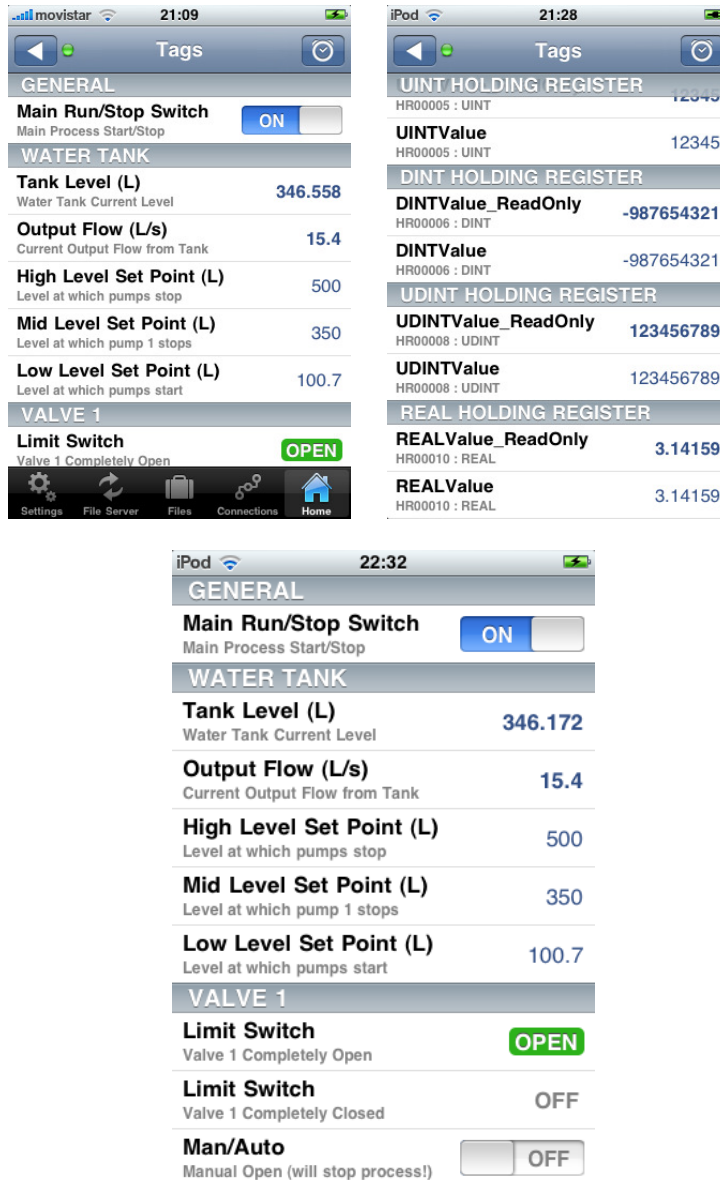
2.1.5 Home

This is the main view and the place where actual tags and tag values are presented. It is available to all users, but access levels on tags are applied depending on the current source files configuration, so what each user views may vary. Trend graphs and alarms are also displayed in this tab.

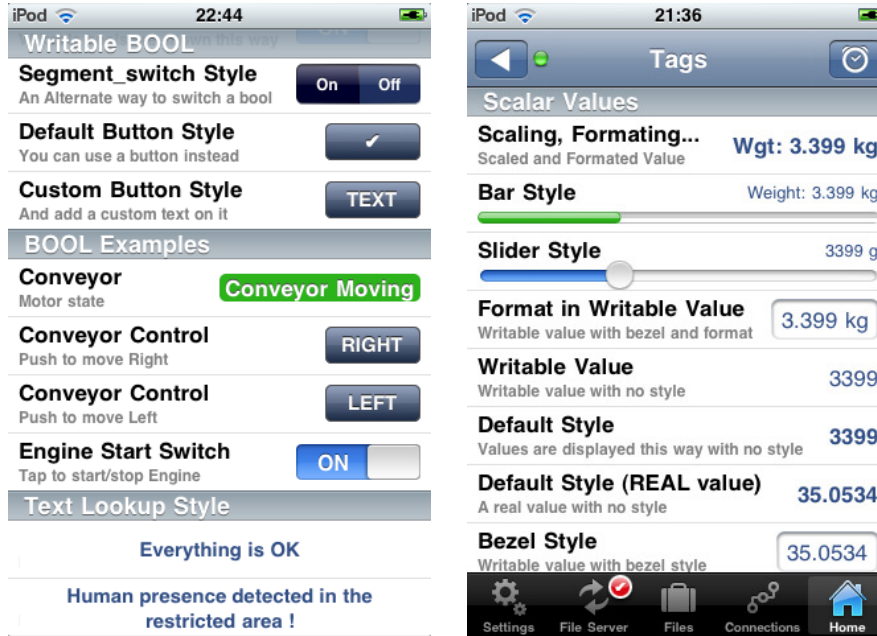
2.2 Variables in ProSoft i-View

Process variables or tag values coming from PLCs are organized in ProSoft i-View as a list with sections in a way similar to the iPhone Contacts Application. The specific display may vary depending on variable type, style, and other characteristics.

On the left side of each row, the list shows relevant information for identifying variables, such as their name, or their particular role in the monitored process. Real-time values of variables or suitable controls for interacting with them are shown on the right side.

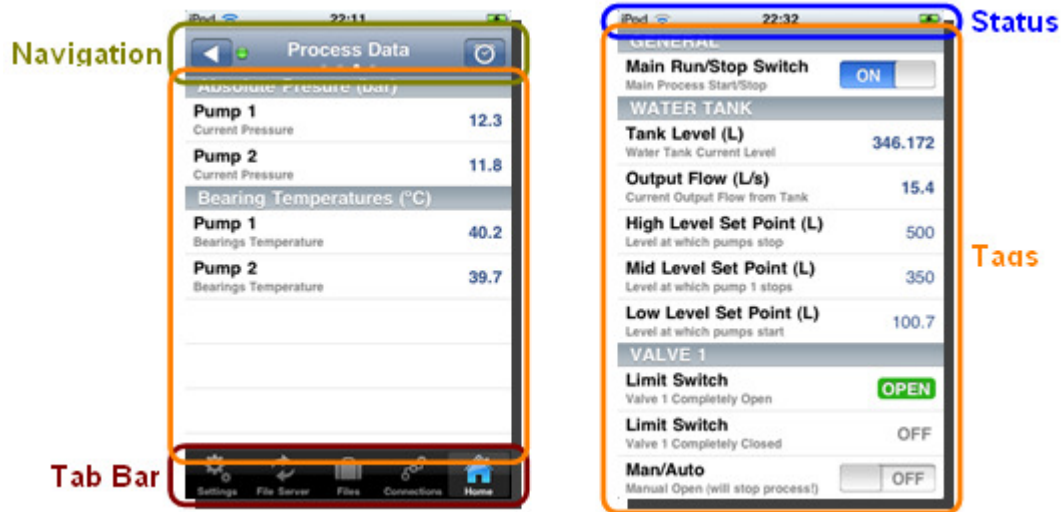


The following examples demonstrate several ways of displaying elements on the *Home* view list.



2.3 Home Tab Bar and Navigation Bar

The *Home* tab is where tags are displayed and where users can interact with their PLC variables. Like most iPhone productivity applications, the interface consists of a navigation bar on top, and a tab bar at the bottom of the screen. The standard iPhone status bar is always visible in ProSoft i-View.



Tab Bar and Navigation Bar Hidden Tab Bar and Navigation Bar

To optimize the available space on the *Home* tab view, users can choose to hide the tab bar as well as the navigation bar as shown in the above screenshots.

You can perform the following interactions on these basic interface elements.

- To hide or show the tab bar on the *Home* view, use the **HIDE BOTTOM BAR** option on the *Settings* tab.
- To scroll to the top of the *Tags* table, tap on the status bar at the top of the iPhone screen.
- To hide or show the navigation bar on the *Home* view, use a scroll down motion while on top of the *Tags* table.
- To switch from one page to another, use a scroll left or scroll right motion on the page title.
- To hide or show the page control on the navigation bar, use the **PAGE DETENTS** option on the *Settings* tab.
- To navigate to the list of pages, tap on the navigation bar's left button.
- To see alarms or trend graphs, tap on the navigation bar's right button.

2.4 Bottom Panel

Tapping on the navigation bar's right button in *Home* view will cause the bottom panel to appear.

The various user interface elements in it allow you to view and acknowledge alarms, as well as to create any number of trend graphs with any number of plots in them.

Just scroll the panel left and right to move to various pages on it. The page control on the bottom also provides a way to switch between bottom panel pages



3 Data Source Files

In This Chapter

- ❖ Data Sources Created in Excel 15
- ❖ Rockwell RSLogix 5000 as a Data Source Generator 60
- ❖ Editing Source Files in a Text Editor 61

Source files contain information that ProSoft i-View needs for visualizing PLC process variables as required by users.

ProSoft i-View accepts csv files created in Excel or Open Office. For some PLC brands, it also supports files directly created from PLC vendors' development tools, such as Allen Bradley RSLogix.

Source files imported into ProSoft i-View are shown in the *Files* tab under the *Sources* section. One or more sources can be selected and variables from all of them will be displayed depending on current user access level.



Each source refers to a single PLC, but several sources can refer to the same PLC. All sources that point to the same PLC are automatically joined into a single connection.

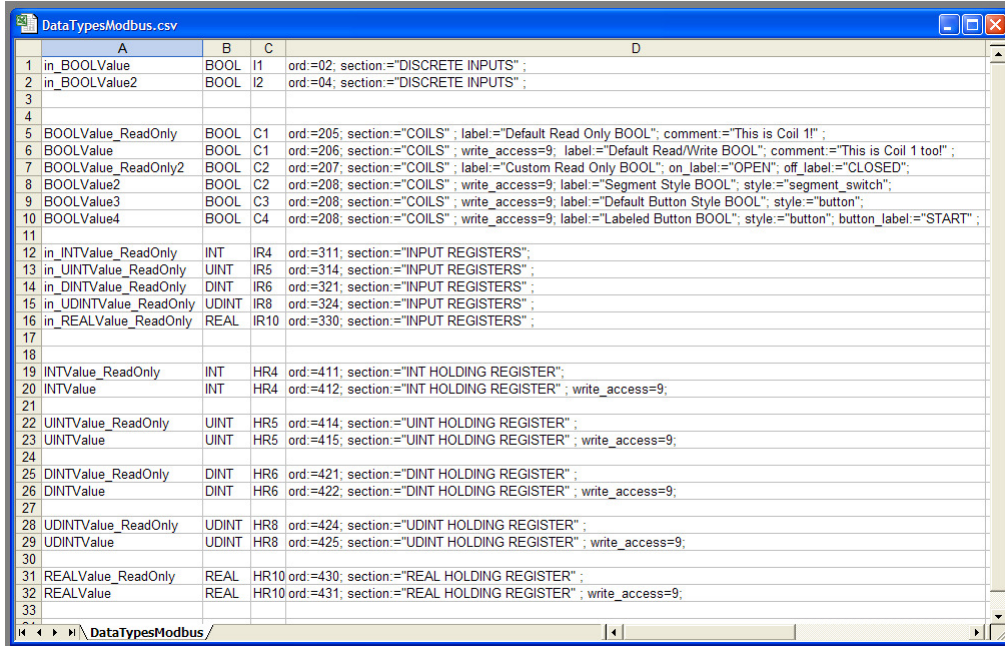
Note: Be sure you have at least one selected source or ProSoft i-View will not display any variables.

3.1 Data Sources Created in Excel

You can use Excel or Open Office to compose a data source file for ProSoft i-View. The file must be exported or saved in CSV format.

Note: CSV file format is not identical for all language localizations due to different use of delimiters. ProSoft i-View accepts any csv in any of the supported languages, but unfortunately this is not the case for MS Excel. Consequently, Excel may not correctly open the provided examples in the wrong language.

An Excel spreadsheet with variable specifications has four (4) columns, and one row per variable.



ProSoft i-View looks for the following information in each column.

Column A : variable name

Column B : data type

Column C : variable address

Column D : tag or global attributes including display and communication attributes

3.1.1 Specification of Variable Names (Column A)

Tag names are entered in column A, they must begin with a letter and should not contain spaces or special characters. For Register or Memory Area based protocols, tag names are only informative or for documentation purposes. Variables can hold numeric values, strings or arrays.

For EIP/Native, variable names are symbolic names that are actually sent to remote devices for communications. In this case, any valid reference to a scalar type (see Variable Types (page 18)), including members of structures or array elements, is allowed.

EIP/Native protocol (AB Logix controllers) Legacy Note

Prior to ProSoft i-View version 1.5, EIP/Native tag names used to be specified in Column A. However, with the incorporation of expressions and in order to favor a general syntax for all protocols, this has been deprecated. From ProSoft i-View version 1.5, it is obligatory to adhere to the General Rule if you want to use tags in expressions. In this case you must specify EIP/Native as communication protocol and you have to set the actual PLC tag name in Column C instead of column A. The name that you enter in Column A can be used then in expressions. If you are not going to use expressions you can still enter a EIP/Native PLC tag name in Column A and write "TAG" in column C as in pre 1.5 versions. Although this is still allowed and existing source files will continue to work, it is recommended for integrators to switch to the new syntax for new projects and to start a transition plan for existing ones.

For example, *myStructData[2,3].intMember* may refer to an integer value referenced by the *intMember* member of element (2,3) of an array of structures. Look at the *EIP_TAG_Examples.csv* template for more examples of how to specify tag names for Allen Bradley Logix controllers.

As a general rule, any *Tag* name path that refers to a scalar value (BOOL, SINT, INT, DINT, REAL) in a Logix Controller will be accessed by ProSoft i-View.

Note that ProSoft i-View performs a Validation Code security check before any other attempt to access any tags is made; therefore, it is mandatory to have a tag named *SMValidationCode* of type INT in your PLC. For more information, see Validation Codes (page 81).

Tag Scope

Tags can be defined to have a **local** or **global** scope

Local tags are identified as per the general specification of variable names, that is beginning with a letter. Global tags are identified by placing a \$ sign before their actual name.

Examples:

```
local_var  
$global_var
```

Local tags have a scope limited to the data source file they are in. When a local name is found in an expression its definition is looked for only in the same file the expression appears, therefore you can use the same names and expressions in several source files without conflicts.

Global tags have an application wide scope so you can only have single instances of them across all selected source files. The advantage of using them is that you can access to their values from anywhere in your project even if the project is made of several source files.

3.1.2 Variable Types (Column B)

Data types determine in part how variables will be displayed. Non-boolean scalar types can be represented in various ways such as by a number, a slider control, or a progress bar. Boolean values can also be displayed in several formats depending on attributes.

Arrays of values can also be stored in variables and transferred from/to PLCs. By properly using array expressions you can retrieve individual values as desired. To indicate that a variable holds an array you append [n] to its data type. In such case, 'n' indicates the total number of elements that the array will hold.

The following types are supported.

DATA TYPE	REMARKS
BOOL[n]	Value that can adopt one of two states
SINT[n]	8-bit signed integer value (-128 ... +127)
INT[n]	16-bit signed integer value (-32768 ... +32767)
UINT[n]	16-bit unsigned integer value (0 ... 65535)
UINT_BCD[n]	4-digit BCD value stored in a 16-bit register using 4 bits per digit (0 ... 9999)
DINT[n]	32-bit signed integer value (-2147483648 ... +2147483647)
UDINT[n]	32-bit unsigned integer value (0 ... 4294967295)
UDINT_BCD[n]	8-digit BCD value stored in two 16-bit registers using 4 bits per digit (0 ... 99999999)
REAL[n]	32-bit floating-point value (IEEE 754) (approx -1e38 ... +1e38)
CHANNEL[n]	Same as UINT
WORD[n]	Same as UINT
DWORD[n]	Same as UDINT
STRING[n]	<p>Type containing a characters string. Actual representation depends on protocol, for example Allen Bradley controllers can hold up to 82 character bytes. Strings on controllers are interpreted by default as per the WINDOWS-LATIN1 encoding, but other encodings are possible if a Explicitly Encoding or a UTF-16 file is given. The STRING type should be used with the appropriate string memory area or string tag type in the controller when available.</p> <p>On protocols with no explicitly support for strings ProSoft i-View uses a generic string representation consisting on a leading word (16 bit register) containing the length of the string followed by a 82 byte long string buffer.</p> <p>Note that STRING[n] does not indicate a string containing <i>n</i> characters but an array holding <i>n</i> strings with the default capacity. Particularly do not confuse with CHAR[n]</p>
CHAR[n]	<p>Similar to STRING except that it does not insert a leading length word. It can be used on protocols with no specific support for strings such as Modbus. In this case 'n' indicates the string buffer length, i.e. the number of character bytes that should be allocated in the PLC for the string, starting from the address specified in column C.</p> <p>Note that CHAR[20] would mean an array of 20 character bytes, however in all cases it will be treated as a single string with a capacity of 20 bytes.</p> <p>Keep in mind that if you use a string encoding other that the default, you must expect the string to hold less than <i>n</i> characters. This is because on some encodings a single character may require multiple bytes to be represented.</p>
LOOKUP	<p>This is a special-purpose type used to create a text entry on a table, which is referred to by tags having the <i>Lookup</i> style attribute. Rows with the <i>Lookup</i> data type in column B will not cause any read or write operations to PLCs. For more information, see Lookup Tables (page 37).</p>
<p>When specifying the tag type, you can optionally define an array size for it as shown above in italics. When you do so, the related variable will hold an array of values of the relevant type instead of a single value. See Memory Arrays for more information.</p> <p>Size definition is obligatory for CHAR types.</p>	

3.1.3 Variable Addresses (Column C)

A variable address represents a memory location or a register in a PLC to which a variable refers. Addresses are specified in different ways depending on which communications protocol they belong to. Therefore, each protocol has its own set of valid addresses. Addresses belonging to different protocols cannot be mixed in a particular source file. Use a separate source file for each PLC and communication protocol. The relevant communication protocol for a source file is uniquely determined by the kind of variable addresses included in it.

The particular protocol to use can be specified by means of a comment on the first line as described in Specification of Communication Protocol. This comment may look something like this:

```
# %protocol eip/native
```

For protocols based on registers or memory locations, Variable Addresses are specified by a prefix referring to the appropriate memory area followed by a numeric value indicating the position in that area. Allen Bradley's Ethernet/IP for Logix Controllers is based on symbolic names. Write PLC Tag symbolic names in Column C

The following memory areas and prefixes are supported.

PROTOCOL	AREA PREFIX	REMARKS
Modbus TCP/IP	I: Input Discrete (read only) C: Coil IR: Input Register (read only) HR: Holding Register	To access Coil number 10, specify C10. To access Holding register 1, specify HR1. Individual bits in HRs can be accessed for reading or writing using a dot notation. For example, HR1.3 would refer to bit 3 in HR1.
EIP/Native (Allen Bradley)	TAG (deprecated, do not use)	Actual symbolic tag name is given in column A. See Variable Names (page 16). However, "TAG" must still be specified in column C.
EIP/PCCC (Allen Bradley)	O0: Outputs I1: Inputs S2: Status B3: Binary T4: Timer C5: Counter R6: Control Nn: Integer Fn: Floating Point STn: String	Tags are specified by file type, file number and offset in the regular way. Individual bits in words can be accessed using the usual slash notation. Examples: B3:5 would access word 5 on file 3 of type B N7:0 would access value at position 0 in N7 file N7:0/3 would access bit 3 in N7:0

Accessing Data Types Longer Than One Register

For data types requiring more than one register or memory location, the lower address in their range must be specified. For example, a variable of type DINT addressed by HR100 will use HR100 and HR101 because 2 Modbus registers (16 bits) are required to accommodate the complete variable (32 bits). Integrators must be aware of this to avoid overlapping tag values. This applies to all protocols except EIP/Native.

Accessing a Register as a BOOL

It is possible to specify a BOOL type for a register or memory location even if it is not meant to hold a BOOL. For example, you specify on a row that HR1 is a writable BOOL. In this case, ProSoft i-View will use a switch control for that row, and will write a value of one (1) or zero (0) to the register depending on user interaction on the switch. This applies to all protocols except EIP/Native. EIP/Native does not allow a non BOOL PLC Tag to be treated as BOOL due to the strict type checking that this protocol encourages. You can use the 'bool' style instead to force the same effect.

Accessing Individual Bits in a Register

Individual bits in registers can be accessed by using the BOOL type and by specifying a bit address using dot (.) or slash (/) notation, depending on protocol (see table above). ProSoft i-View will use the appropriate protocol command to avoid overwriting bits on the register. Note that this feature does not apply to EIP/Native. On EIP/Native you can still use the dot notation to access individual bits on variables, but due to strict type checking, you must set the correct variable type on column B. In order to force ProSoft i-view to display such values as bools you can use the 'bool' style.

Note on EIP/Native Communication Protocol

EIP/Native communications do not rely on particular memory locations or positions, but on symbolic names. With this protocol, the user is relieved of the responsibility of assigning memory addresses or registers, as well as the need to take tag sizes into account for storage. Additionally, EIP/Native tags hold data information such as type and size, which ProSoft i-View uses to check against type mismatches on PLC returned values. As a result, it is not possible to store values that differ in type or size from the values uniquely defined in the PLC. Any attempt to do so will result in a 'type mismatch' error for the offending tag. For **EIP/Native** symbolic names any valid reference to an existing scalar or array type tag including structure members or array elements is allowed.

For example "myStructData[2,3].intMember" may refer to an integer value referenced by the intMember member of element (2,3) of an array of structures. Look at 'EIP_TAG_Examples.csv' template for more examples of how to specify tag names for Allen Bradley Logix controllers. As a general rule, any Tag name path referring to an existing scalar value (BOOL, SINT, INT, DINT, REAL, STRING) or array of such elements in a Logix Controller can be accessed by ProSoft i-View.

To access arrays as a whole you need to define an array size next to the type, as discussed on the previous and following sections.

You can also access program tags by using the following syntax

Program:<program_name>.<tag_name>

Note that 'Program' is literal. <program_name> and <tag_name> identify just what they suggest.

Note also that ProSoft i-View performs a Validation Code security check before any other attempt to access other tags is made, therefore, it is mandatory to have a tag named "SMValidationCode" of type INT in your PLC for communications to work.

Internal Tags

Internal tags are stored and managed inside the ProSoft i-View app. Internal tags mostly behave as actual PLC tags except that they are not linked to an actual PLC address. Thus, internal tags do not require an active TCP connection to display a value. In combination with expressions, internal tags are a powerful feature that allows for presenting calculated values to the user or holding intermediate values for subsequent use.

Internal tags support most of the available attributes except the ones specifically targeted at PLC tags such as the 'scale' attribute.

To specify that a Tag is internal use the word 'INTERNAL' instead of a PLC address or Symbolic Tag

Protocol	AREA PREFIX or TAG	REMARKS
All	INTERNAL LOCAL (deprecated)	Indicates that this tag does not have a link to an actual PLC tag. Instead, it exists only in the app. Internal tags can be used to store and represent intermediate values, or expression results.

Just as regular PLC tags, internal tags can have *Local* or *Global* scope. To identify an Internal tag to have a *Global* scope just prefix it with a \$ sign.

3.1.4 Attributes (Column D)

Process variables in ProSoft i-View are represented in 'cells.'

You can configure the behavior and display of the monitored variables by setting the appropriate attributes. For example, some of the available attributes are "section," "label" and "comment."



An attribute description in column D of a source file follows this general pattern:

attribute := value;

For example, a boolean process variable like the one represented in the first cell on the figure above could have the following attribute description:

```
ord := 1 ; section := "GENERAL" ; label := "Main Run/Stop Switch" ;  
comment := "Main Process Start/Stop" ; access := 3 ; write_access  
:= 5;
```

The way the cell is actually displayed in the figure implies that the current user has an access level of 5 or above, because a selectable switch is provided as the process variable value instead of static text. See User Accounts (page 71).

3.1.5 Attribute Scope and Type

Attributes by Scope

Most attributes apply to a single tag. They are referred to as **tag attributes**. Others have a global scope within a source file, and we refer to them as **global attributes**. In the following sections, all attributes are discussed individually.

Attributes by Type

Attributes can hold a **numeric value**, a **text**, a **special text** or list of values depending on their meaning or purpose.

- **Numeric values** are expressed as decimal numbers with optional decimal point and decimal digits if applicable to the attribute.

Several special values are provided for convenience.

true : equals 1.0

false : equals 0.0

-inf : represents a very large negative number

+inf : represents a very large positive number

Numeric values are expressed directly after the equal sign without quotation marks, including the convenience values. Examples:

```
write_access := 3 ;
```

```
word_swap := true ;
```

- **Text** usually represent text labels or fields in the application interface. They are only required to be enclosed in quotation marks if they contain spaces or the semicolon character. However, to maintain readability, it is advisable to always use quotation marks.

```
label := "Main Run/Stop Switch" ;
```

```
suffix := " %" ;
```

- Attributes requiring a **special text** only accept specific, pre-defined text strings. Valid texts vary depending on the particular attribute.

```
style := "bezel" ;
```

```
format := "4.2" ;
```

- **Value Lists** are used in cases where a single numeric value is not enough to provide the information required by the attribute. The general format is a list of numbers separated by colons and enclosed between '{' and '}' characters.

```
scale := { 0, 360, 0, 100 } ;
```

```
bounds := { 0, 100 } ;
```

```
color_bounds := { -inf, 50 } ;
```

3.1.6 Tag Attributes

A tag attribute is applied to a tag at the same row and exclusively affects that tag. All attributes are optional, and the specified attributes can be in any order.

The following attributes are supported.

TAG ATTRIBUTE	TYPE	DESCRIPTION
Page	text	Text indicating which page the variable belongs to. Text values must be enclosed in quotation marks. For example, page:="FIRST PAGE";
Section	text	Text indicating which section the variable belongs to. Text values must be enclosed in quotation marks. For example, section:="SECTION ONE";
Label	text	Main text in the cell representing a variable. If not specified, the variable name in column A will be used instead.
Comment	text	Secondary text in the cell representing a variable. If not specified, a text comprising the variable address and its type will be displayed instead.
Style	special text	Text attribute allowing for additional ways to represent the value of a variable. "switch" - The default style for writable booleans. It displays a boolean variable using a "switch" control. The variable will change its state on each touch. Will be ignored if the variable is not writable. "segment_switch" - Displays an alternative to the "switch" style for writable booleans "button" - Presents a push button for writing of boolean values. Contrary to the "switch" style, the related process variable does not change permanently but goes to 1 (true) on button press and goes to 0 (false) on button release. "bezel" - Draws a bezel line around a text field for highlighting writable numeric values. Note that it is only applicable to writable tags "slider" - Presents a writable tag as a slider control. Min and Max values are given with the <i>Bounds</i> attribute. "bar" - Presents a read-only tag as a bar. Min and Max values are given with the <i>Bounds</i> attribute. "lookup" - Displays text from a lookup table instead of the actual tag value. The tag value is used to determine which entry in the table is shown. Note that this attribute will only affect read-only tags. "alarm" - Tags with this style are not shown in the main tag table. Instead they are treated as alarm conditions. They may be combined with the <i>Bounds</i> attribute. See Alarms (page 39).
Ord	number	Numeric value determining the order in which pages, sections and tags are displayed. For more information, see Pages, Sections, Rows and Data Sources (page 36).
Access	number	Indicates the minimum access level a user needs in order to view a variable. By default, access:=9 ; is set.

TAG ATTRIBUTE	TYPE	DESCRIPTION
Write_access	number	Indicates the minimum access level an user needs in order to trigger changes (write) to a process variable value. Omitting this attribute forbids any change to the associated variable.
On_label	Expression(string)	String expression displayed for a read-only boolean variable when its value is 1 (true).
Off_label	Expression(string)	String expression for a read-only boolean variable when its value is 0 (false).
Button_label	Expression(string)	String expression In combination with the "button" style, it defines the text for the button.
Scale	array	A four-element array in the form {x1,x2,y1,y2} where x1, x2 represent a pair of numeric values in raw units as present in the PLC and y1, y2 represent the same values in engineering units as will be displayed on ProSoft i-View. By setting this tag attribute, you can convert (scale) raw values to engineering values on the display by applying a linear transformation. This attribute can be specified for any read-only or writable tag. Example: scale:={0,100,0,1} ;
Bounds	array	A two-element array in the form {min,max} where min and max are numeric values used to indicate a range expressed in engineering units. This attribute can have several meanings depending on other attributes, particularly the <i>Style</i> and the <i>Write_access</i> attributes. On writable tags, it determines and limits the available range of values that users will be able to enter. On read/only tags, its meaning depends on the particular style of the tag. It is currently supported by the "bar", "slider", and "alarm" styles. Example: bounds:={-100,100} ;
Format	special text	Text in the form "m.n" where m represents the minimum number of characters to be displayed. If the value to be displayed is shorter than this number, the result is padded with blank spaces (or zeros if the number starts with zero). The value is not truncated even if the result is larger. n represents the number of digits to be displayed after the decimal point. Example: format:="07.2" ; will display the REAL value 12.345 as 0012.34
Prefix	text	Text to be prepended just before the variable value. Example: prefix:="\$" ;
Suffix	text	Text to be appended to the variable value. Example: suffix:=" %" ;

TAG ATTRIBUTE	TYPE	DESCRIPTION
Color	special text	Indicates a color to apply to a variable value or control. Colors can be specified by name as listed in http://www.w3schools.com/css/css_colornames.asp . Colors can also be given by RGB value in hexadecimal format. The <i>Color_bounds</i> attribute determines when the color will be applied. Supported values also include "TextDefault", "BarDefault" and "DefaultGreen," which are the colors used by default on texts, bars and boolean tags. Examples: color := "red"; color := "olive" ; color :=#FF3300;
Color	Expression Numeric	Numeric Expression. The color attribute used in this way expects a 32 bit integer value containing the RGB color coordinates in the three lower significance bytes, with the B value in the least significant byte. The method SM.color can be used for convenience to generate a color from its RGB coordinates or name.
Tint_color	special text	Indicates an alternate color to apply. The <i>Color_bounds</i> and <i>Tint_color_bounds</i> attributes determine color application ranges. Refer to the <i>Color</i> attribute for valid color values. Example: tint_color := "red";
Color_bounds	Value list	A two-element array in the form {low,high} where low and high are numeric values determining the range of values for the tag where the color will NOT be applied, also known as an exclusion range. Examples: color_bounds := {-inf,50}; This will exclude tag values below 50 from displaying in the color set in the <i>Color</i> attribute. Therefore, tag values will be displayed in the specified color when they are above 50. color_bounds := {-50,50}; Tag values from -50 to 50 will be excluded from displaying in the color set in the <i>Color</i> attribute. Therefore tag values will be displayed in the specified color only when they are below -50 or above +50, effectively enabling coloring for edge conditions. For boolean tags the color_bounds attribute has a predetermined value that will always make the tag appear in the color color for the On state.

TAG ATTRIBUTE	TYPE	DESCRIPTION
Tint_color_bounds	Value list	<p>A two-element array in the form {low,high} where low and high are numeric values determining the range of values for the tag where the tint color will NOT apply, also known as an exclusion range. Refer to the <i>Color_bounds</i> attribute for more information.</p> <p>When both <i>Tint_color_bounds</i> and <i>Color_bounds</i> attributes are specified, <i>Color_bounds</i> takes preference over <i>Tint_color_bounds</i>. However, <i>Tint_color_bounds</i> still has the chance to act in the range that has been excluded by <i>Color_bounds</i>, allowing for further color customization based on tag value.</p> <p>Combined examples: color_bounds := {-inf,50}; tint_color_bounds := {-inf,0}; Will display all values above 50 in the <i>Color</i> color; positive values up to 50 will be displayed in the <i>Tint_color</i> color. color_bounds := {-50,50}; tint_color_bounds := {-20,20}; Will display all values below -50 and above 50 in the <i>Color</i> color. Values from -20 to 20 will be excluded from the <i>Tint_color</i> color, so the tint color will show for values from -50 to -20 and from +20 to +50.</p> <p>For boolean tags <i>color_bounds</i> and <i>tint_color_bounds</i> have predetermined values. You can simply use <i>tint_color</i> to display a color for both Off and On states, while <i>color</i> will supersede the color for the On state.</p>
blink	Expression boolean	Numeric Expression. Will cause any non writable tag to blink on the interface when the value assigned to the attribute is not zero (true)
blink_bounds	Value list	<p>A two elements array in the form {low,high} where low and high are numeric values determining the range of tag values where blinking will NOT be applied. If specified on a read only tag, the tag will visually blink with a period of 1 second for the values not in the specified range.</p> <p>Example (to set a tag to blink always) : blink_bounds:=-{inf,-inf}; Example (to set a tag to blink when its value goes below -10 or above +10) : blink_bounds:=-{10,10}; Example (to set a boolean tag to blink for the On state) : blink_bounds:={0,0};</p>
plot_color	special text	Determines the color to be used in trend graphs when plotting this tag. If omitted a sequential color is chosen automatically.

value	expression	<p>Specifies a value to write on the related tag based on the result of an expression (see Expressions)</p> <p>Note that the 'value' attribute has a slightly different behavior for Internal tags than for regular PLC tags:</p> <p>For PLC tags the result of the expression is first written to the PLC already converted to the relevant type and appropriately limited and descaled according to the 'bounds' and 'scale' attributes. On reading, the tag value is scaled back to the engineering unit. As a consequence a tag may end having a slightly different value after that. For example a Tag of type INT having scale={0,10,0,1}; will become 1.2 when assigning 1.234 to it through a 'value' change. Of course, the value in the PLC will become 12.due to the scaling and type conversion.</p> <p>On PLC writes involving STRINGS implicit conversions will also be performed if possible. For example, the STRING "2.3" will be converted to the number 2.3 upon writing a REAL tag. Similarly, the number 2.3 will result to the sequence of characters "2.3" when writing to a STRING tag.</p> <p>For INTERNAL tags no type or scale conversion is performed on the result of the 'value' expressions. Internal tags are just given the result of the expression as it evaluates. Thus, the 'scale' and 'bounds' attributes will have no effect. Even the tag type is ignored to the effects of the assignment and the variable may hold a completely different type.</p> <p>The following considerations also apply:</p> <p>INTERNAL tags containing tag references in its 'value' expression can not be user writable and will ignore an eventual 'write_access' attribute (NOTE: this restriction was removed after version 1.5).Despite the preceding consideration, INTERNAL tags will preserve the meaning of the 'write_access' attribute if their 'value' expression represents a constant value, in such case the expression will provide the initial value for the tag.</p> <p>Writable INTERNAL tags will simulate a write-read round on a virtual PLC taking into account tag type, 'scale' and 'bounds' attributes with the same effects as a real write-read round as described above.</p>
hidden	expression (boolean)	<p>Numeric expression. Will make the row hidden when the result of the expression is non zero. Hiding all rows in a Section will remove the entire section including the section title. Hiding all rows/sections on a page will remove that page from the interface.</p> <p>Hiding/showing interface elements is fully animated and dynamic on evaluation of the 'hidden' expression. Dynamic hiding is useful to switch among several rows that selectively meet an arbitrary condition, or to force display parts of the interface depending on user or PLC triggered conditions.</p>
style	special text	<p>Attribute containing one of the texts below determining ways to represent or display the value of a variable.</p>

		"switch"	It is the default style for writable booleans. It displays a boolean variable using a "Switch" control. The variable will change its state on each touch. Will be ignored if the variable is not writable.
		"segment_switch"	Displays an alternative to the "switch" style for writable booleans.
		"button"	Presents a Push Button for writing of boolean values. Contrary to the "switch" style, the related process variable does not change permanently but it goes to 1 (true) on button press and goes to 0 (false) on button release.
		"bezel"	Draws a bezel line around a text field for highlighting writable numeric values. Note that it is only applicable to writable tags.
		"bool"	Presents the tag as a boolean even if it is not of type BOOL, limiting the possible values to 0 or 1. It also activates related attributes such as <i>on_label</i> and <i>off_label</i> and brings the special meanings of some attributes for bool types.
		"slider" "bar"	Presents a tag as a slider control or a bar depending on whether it is writable. Min and Max values are given with the <i>bounds</i> attribute. Both are identical and can be used indistinctly.
		"lookup"	Displays a text from a lookup table instead of the actual tag value. The tag value is used to determine which entry in the table is shown. This attribute will work both on read only tags and writable tags. On writable tags the bounds attribute is used to determine which range on the lookup table will be made available to the user for selection. See Lookup Table .
		"picker"	It is like the "lookup" style except that it will draw a bezel around the displayed lookup text when used on writable tags.

		"alarm"	Tags with this style are referred to as Alarm Tags and they are not shown on the main table but on the Alarms Panel. They can be combined with the <i>bounds</i> attribute. See Alarms .
		"barcode"	Activates the barcode reader for this tag. The tag must be writable (usually of type STRING). After reading a barcode with the device camera, the tag will be updated with the scanned code.

Tag attributes specific to Modbus devices:

MODBUS TAG ATTRIBUTE	TYPE	MEANING
Slave_id	number	This attribute identifies the associated tag as belonging to a particular Modbus device. Specifically, Modbus serial devices with the specified ID that are connected through a TCP gateway will be accessed. The default value is 1.

Note that the *Slave_id* attribute for Modbus is a tag attribute, so it applies only to the register it is next to. This is in contrast to the *Controller_slot* attribute, which is global and applies to a source file.

3.1.7 Global Attributes

Global attributes are modifiers that apply to all tags or have a meaning in the context of the whole source file. Global attributes can be placed anywhere in column D of source files. So they can be specified next to any variable and they will still have a global scope.

Note: If a particular global attribute is included more than once in a source file, only its first occurrence will take effect.

Communication settings are global and therefore need to be specified only once in a source file.

COMMUNICATION ATTRIBUTES	TYPE	DESCRIPTION
Local_ip	text	Source address in text format for local access (LAN). Example: "192.168.1.40"
Local_port	number	TCP port used for local connections (LAN) to this source. If it is not specified, ProSoft i-View will use the standard port for the protocol of the current source file. (For example, 502 for Modbus. Note that this can differ from the default port specified in the ProSoft i-View <i>Settings</i> tab.)
Remote_host	text	Source address or host name for remote connections. Example: "remote.remotehost.com"
Remote_port	number	TCP port used for remote connections to this source (WAN-Internet). If it is not specified ProSoft i-View will use the standard port for the protocol of the current source file.
Ssl	number (boolean)	Specifies whether TLS/SSL encryption is required for a remote connection. Nonzero numeric values are considered true. Default value is 'false'.
Validation_tag	Special text	Allows for using a custom validation tag on protocols supporting it. For EIP/PCCC use validation_tag = "Nx:y"; only N files can be used and the code is stored as an INT (default is N98:0). For FINS/TCP use validation_tag = "Dx"; only DM area can be used and the code is stored as a WORD (default is D19998). For EIP/NATIVE the validation tag name is always SMValidationTag, it can not be changed. For MODBUS/TCP no validation tag is available.
If no attribute from this group is specified, ProSoft i-View will dynamically use values from the application <i>Settings</i> tab view.		

The Modbus specification does not define exactly how the data is stored in registers or the order in which bytes or words are sent. The following global attributes help to deal with this. Swapped words/bytes options for Modbus are global.

MODBUS GLOBAL ATTRIBUTES	TYPE	DESCRIPTION
rtu_mode	number (boolean)	ProSoft i-View will use "Modbus/RTU over TCP" instead of "Modbus/TCP". This will allow for accessing serial modbus/RTU devices behind an Ethernet-to-serial gateway not supporting MBAP. Use the 'slave_id' attribute on tags to route commands to the right modbus slave node.
Word_swap	number (boolean)	Swaps words for 32-bit data (such as DINT or REAL) before sending to or upon receiving from a Modbus device. Nonzero values are 'true.' Default value is 'false.'
Byte_swap	number (boolean)	Swaps bytes for 16-bit or 32-bit data before sending to or upon receiving from a Modbus device. Nonzero values are 'true.' Default value is 'false.'
<p>The combined effect swap attributes is as follows: Assuming a default of 'ABCD' for a byte order where 'A' is the Most Significant Byte (MSB) and 'D' is the Least Significant Byte (LSB), you can combine <i>Word_swap</i> and <i>Byte_swap</i> with the following results:</p> <ol style="list-style-type: none"> 1- word_swap:=false; byte_swap:=false; will give 'ABCD' for 32-bit values and 'AB' for 16-bit values. 2- word_swap:=false; byte_swap:=true; will give 'BADC' for 32-bit values and 'BA' for 16-bit values. 3- word_swap:=true; byte_swap:=false; will give 'CDAB' for 32-bit values and 'AB' for 16-bit values. 4- word_swap:=true; byte_swap:=true; will give 'DCBA'. for 32-bit values and 'BA' for 16-bit values. <p>Note: These attributes are only meant for Modbus communications and are ignored for the rest of the supported protocols.</p>		

Allen Bradley ControlLogix controllers can be plugged into any slot on the backplane. Ethernet/IP messages can be sent "connected" or "unconnected". The following attributes can be used to determine these characteristics. These are global attributes.

EIP/NATIVE GLOBAL ATTRIBUTES	TYPE	DESCRIPTION
controller_slot	number	Identifies the slot where the Logix controller is located. Default value is 0.
connected_mode	number (boolean)	When true, ProSoft i-View will use "connected messaging" instead of the default "unconnected messaging" for retrieving data from Ethernet/IP enabled PLCs. Look below for a discussion on what possible effects you might expect. Default value is 'false'.
<p>ProSoft i-View currently supports two EIP mechanisms to send commands to AB PLCs:</p> <p>(1) For a Micrologix or SLC it will send PCCC commands (DF1) embedded in EIP using a direct path.</p> <p>(2) For a ControlLogix/CompactLogix it will send native CIP commands using a Backpane, Slot-Number path. The Backpane defaults to 1 and the Slot number is given in controller_slot.</p> <p>ProSoft i-View uses CIP Explicit Messages to retrieve and send data from/to Ethernet/IP enabled PLCs. Explicit messages can be sent "unconnected" or "connected". "Connected" messages require a Connection ID which is first asked to the PLC before sending other messages, while "unconnected" messages identify the specific path to the destination in the same message. Connected messaging is generally considered to be more reliable than unconnected because it reserves buffer space for the message, and is therefore less likely to be blocked by other message traffic. However, if the TCP link between the message originator and the receiver is weak or prone to fail, unconnected messaging may be a better choice. Wireless spots or carrier networks can easily drop due to lack of coverage or weak signal, in these cases connected messaging communications may take longer to reestablish after a fault, resulting in less overall reliability and more user perceived delays than unconnected messaging. ProSoft i-View uses unconnected messaging by default, but you can set it to use connected messaging for a source file by setting the connected_mode attribute to true.</p>		
<p>Note that since the <i>Controller_slot</i> attribute is a global one, it applies to the whole source file. This is in contrast to the <i>Slave_id</i> attribute for Modbus, which only applies to the tag next to it. Also, note that since different protocols cannot be mixed in the same source file, the two attributes are incompatible between them, and the non-relevant one will be ignored.</p>		

3.1.8 Pages, Sections, Rows and Data Sources

Page, Section and Ord Attributes

The ProSoft i-View interface for the iPhone and iPod Touch consists of a set of cells or rows representing PLC variable values, which in turn are arranged in pages containing section headers for groups or rows.

Variables are optionally assigned a page and section through the *Page* and *Section* attributes. All variables having exactly the same *Page* name will belong and be added to a page with such name. Similarly, the *Section* name attribute is used to put all variables belonging to the same section under a heading with that name.

A single attribute, the *Ord* attribute, is used to force pages, sections and rows to be placed in specific locations.

Pages and sections are arranged according to their first appearance in the source files based on global order of all variables. This is used to force sections and pages to appear in the desired place. Variables without an order number will be considered to fall below or after ordered variables, but they will maintain their relative positions in the source file.

How It Works

First, rows from *all* selected data source files are sorted from low to high using their *Ord* attribute. As stated above, variables with no *Ord* attribute are moved to the end of the list, always keeping their original relative order.

As sorting is performed, the variables' *Ord* numbers are used to determine their page and section position order.

Pages are arranged in position orders determined by the **smaller** *Ord* number of all variables belonging to each page. In other words, a particular page will be shown in the order given by the *Ord* number of the **first** variable belonging to that page.

Within each page, sections are then arranged according to the **smaller** (or **first**) *Ord* number of all variables belonging to that section and page.

Finally, variables are placed within their page and section according to their *Ord* number.

Example

The following *Page / Section / Ord* attributes in a series of tags from two sources will be arranged as shown below.

Source 1		Source 2	
Tag	Attributes	Tag	Attributes
TAG A	page := "Page 1"; section := "Section 1"; ord := 1 ;	TAG D	page := "Page 1"; section := "Section 2"; ord := 4 ;
TAG B	page := "Page 2"; section := "Section 1"; ord := 2 ;	TAG E	page := "Page 2"; section := "Section 2"; ord := 5 ;
TAG C	page := "Page 3"; section := "Section 1"; ord := 3 ;	TAG F	page := "Page 3"; section := "Section 2"; ord := 6 ;

Resulting arrangement of tags in Pages and Sections

Page 1	Page 2	Page 3
Section 1	Section 1	Section 1
TAG A	TAG B	TAG C
Section 2	Section 2	Section 2
TAG D	TAG E	TAG F

3.1.9 Lookup Tables

The Lookup table feature allows a tag to pick up a text string from a previously entered list based on current tag value, and display this text instead of the tag value.

To have a tag display a lookup text instead of the usual value, use the *Style* attribute with "lookup" as value.

style := "lookup" ;

If you would like to show the lookup text alone as shown in the *StylesExampleModb* example, you might consider setting the tag label to an empty text.

label := "" ;

The actual table is given in additional rows in the CSV file by using special tags of type LOOKUP (set in column B) and by specifying the related indexes in column C. The text or the comment attribute value on column D is the Lookup Text.

As an example, this is what you could do to create a two-entry table with indexes 1 and 2.

Column A	Column B	Column C	Column D
entry1	LOOKUP	1	This is the lookup table at index 1
entry2	LOOKUP	2	This is the second entry in the table so it will display if a lookup tag value is 2

Lookup numbers do not need to be ordered or contiguous. They can be any number that fits in 16 bits (0 to 65535).

Multiple User Lookup Tables

You can specify an access level for entries in column D. In this case, you must set the lookup text in the *Comment* attribute. This allows for having different texts depending on user level. For example, consider the following:

Column A	Column B	Column C	Column D
entry1_boss	LOOKUP	1	access:=9; comment:= "Only I will see this message. I'm the boss!";
entry1_worker	LOOKUP	1	access:=3; comment:= "I can see this with my user level of 3 or more";

Multiple Range Lookup Tables

Lookup styled tags use the engineering unit as the index to the table. Therefore you can use the *scale* attribute to make a lookup styled tag access to determinate portions or ranges of the table.

For example, the setting: **scale:={0,10,100,110}**; will forward any raw value coming from the PLC in the range from 0 to 10 to table text entries from 100 to 110.

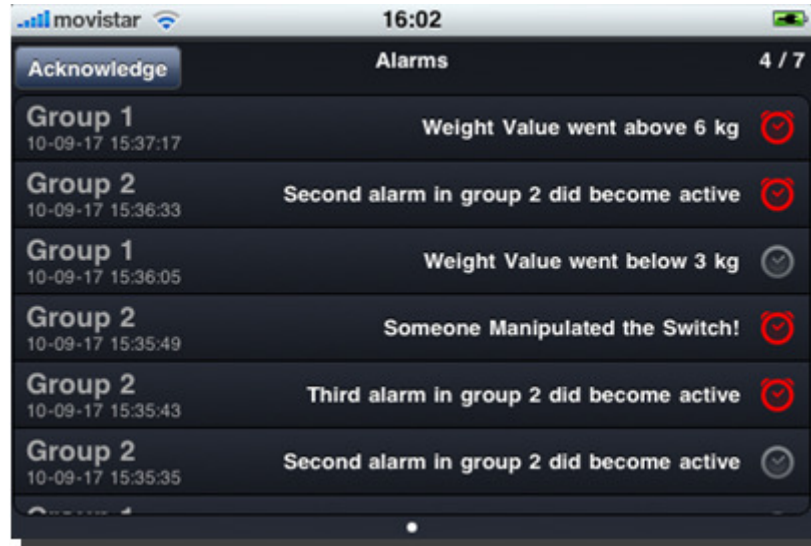
You can use this feature to effectively have several virtual tables simultaneously in use. The *StylesExampleModb* example shows how this concept works.

The global lookup table is available to expressions with the *SM.lookup* method.

3.1.10 Alarms

An alarm tag is a special kind of tag that is not displayed in the main tags table. An alarm tag is characterized by its *Style* attribute being set to "alarm." Alarm conditions can be specified with the *Style* attribute in combination with the *Bounds* attribute attached to the tag.

Alarms are tracked and listed on a separate table as shown in the screenshot below. When an alarm condition is triggered, the tag *Label* attribute is used to display the text on the left, which may refer to an alarm group, and the *Comment* attribute contains the alarm message displayed on the right.



Alarms will remain on the list as long as they remain active or have not been acknowledged. Their current state will be shown by small icons next to the alarm text.



Bright Red Alarm Clock icon means active and not acknowledged



Dark Red icon means active and acknowledged



Gray Clock icon means inactive and not acknowledged

For scalar type variables such as INT or REAL, the bounds attribute determines the exclusion range of the alarm condition. In other words, an alarm styled tag will be considered active if its current value is NOT in the specified range. For example consider the following:

```
style:="alarm"; bounds:={0,100};
```

The tag holding these attributes will be treated as an alarm, which will become active when its value is below 0 or above 100.

To support simple min or max condition alarms, the special numeric values "-inf" and "inf" can be used. For more information, refer to Attribute Scope and Kind (page 25). For example,

```
style:="alarm"; bounds:={-inf,100};
```

will trigger the alarm only when the value on the tag is above 100.

The default value for the *Bounds* attribute on alarm tags is `bounds:={0, 0}`; Therefore, any non-zero value in a tag will trigger an alarm. This is especially relevant for BOOL tag alarms, because by default they will become active when the tag goes to 1 (true) and inactive otherwise, just as you would expect.

Example

The following rows in a source file represent the settings for the alarm tags that generated the first and fourth rows on this page:

```
Alarm1 INT    HR1    label:="Group 1"; comment:="Weight Value went above 6
kg"; style:=alarm; bounds:={-inf,6};
Alarm4 BOOL   C1     label:="Group 2"; comment:="Someone Manipulated the
Switch!"; style=alarm;
```

Performance Considerations

Unlike regular tags, alarm tags are continuously polled from PLCs even if they are not shown on the screen. Also, ProSoft i-View may continue polling them while running in the background. So it is recommended that special care be taken when deciding what tags will be reserved for alarms. Particularly, it is highly recommended to group alarms in tags as contiguously as possible, and to use boolean alarms instead of scalar value alarms as much as possible. Arrays of BOOL are the best choice, if supported by the protocol. Observing this recommendation will lead to shorter communication patterns and less network overhead than if no special care was taken, ultimately improving the end user experience.

3.1.11 Comments in Data Sources

You can comment Rows on Source Files for documental purposes or while you are testing your project. To do so just start the row with the '#' character. Examples:

```
# These are the lookup table entries for the style selection texts we are using
# This alarm checks for a value being too high on the statistics page
```

3.1.12 Specification of Communication Protocol

Starting with version 1.5 you can explicitly set the communication protocol on source files, instead of having ProSoft i-View determining it by the kind of addresses used in Column C as before. This feature has been provided to grant future compatibility with a larger number of communication protocols, where some addressing naming conventions could conflict or overlap existing ones.

To tell the actual communication protocol on a source file you must insert the following comment as the first line.


```
# %protocol <protocol_string>
```

As <protocol_string> you can use one of the following;

```
eip/native  
eip/pccc  
modbus/tcp  
fins/tcp
```

For example to communicate to an Allen Bradley ControlLogix you would have the following as the first line on your source file:

```
# %protocol eip/native
```

NOTE: Protocol Specification in this way will be made obligatory in the future, so it is recommended for integrators and advanced users to use the new syntax for new projects and to set a plan to edit their source files to conform with it.

3.1.13 International Languages Support and String Encodings

The ProSoft i-View iOS app includes localizations for English and Spanish for its user interface interface. However, International Characters and Strings in any language are fully supported. Integrators can therefore chose to present their project interface in any language.

To represent strings the concept of *String Encodings* is used. String Encodings are international conventions that determine how characters representing particular languages are stored into files and device memory.

English and Western European Languages

By default ProSoft i-View assumes source files and Strings to conform to the **Windows Latin1 encoding**. This is adequate for English and most Western European languages such as German, French, Spanish, Portuguese, and many others. It does not require any particular setting. English and Western European versions of Excel on Windows also use the Latin1 encoding for CSV files. On Apple Mac computers you must save your CSVs as 'csv-windows' format to conform to this encoding.

The Latin1 encoding is backward compatible with old plain ASCII, meaning that ASCII characters share the same codes when represented in Latin1 encoding.

String Encoding for International Languages

If you use a language that can not be represented with the Latin1 encoding, you are still able to import your files containing international characters in text attributes. Additionally, international characters are supported in Strings and they can be written and read back from PLCs without breaking their particular encoding.

There are two ways for using international characters in ProSoft i-View:

Unicode UTF-16 encoding (16 bit multibyte encoding)

The UTF-16 encoding is capable for representing any character from virtually any language in the world. To make use of this encoding you can do the following:

- Create a source file with texts in your language using your localized version of Excel as usual.
- Export the file as Unicode UTF-16 text.
- Import the file as usual in ProSoft i-View. ProSoft i-View will automatically detect files encoded as UTF-16. Unicode strings will be converted to UTF-8 before storing them on PLC memory.

NOTE: Strings in source files containing UTF-16 characters will be converted automatically to UTF-8 regardless of the Explicit Encoding specified on the source file as defined in the next sub heading. Any Explicit string Encoding given in a source file will be thus ignored if the file was already UTF-16 encoded.

Explicit Localized encodings (8 bit or multibyte encodings)

Instead of using UTF-16 you may chose to use one of the 8 bit or multibyte encodings that are commonly used by default on localized versions of Windows and Excel. To do so follow the next steps:

- Create a source file with Strings on your language using your default localized version of Excel as usual.
- You must explicitly tell ProSoft i-View the right string encoding of your file. See below.
- Export the file as usual. Excel will use the default encoding for your localization.
- Import the file in ProSoft i-View. ProSoft i-View will use the explicit encoding to open the file.

Setting a Explicit String Encoding

To set a explicit *String Encoding* for your file place the following command embedded in a comment on top of your file

```
# %encoding <string_encoding>
```

.

For example to set Japanese you can write

```
# %encoding Japanese/Win
```

The following explicit *string encodings* are supported:

WindowsLatin1	Identifies the ISO Latin 1 encoding (ISO 8859-1). This is the default.
UTF-8	Identifies the Unicode UTF 8 encoding.
MacRoman	Identifies the Mac Roman encoding. Used on western localizations of Mac OS. Useful when you use diacritic characters (Spanish, French, German, the degree ° symbol...) but you do not want to export your file as csv-windows.
Cyrillic/Mac	Identifies the Mac Cyrillic encoding
Cyrillic/Win	Identifies the Windows Code page 1251 Slavic Cyrillic encoding
Cyrillic/ISO	Identifies the ISO 8859-5 Cyrillic encoding
Japanese/Mac	Identifies the Mac Japanese encoding
Japanese/Win	Identifies the Windows Code page 932 Japanese encoding
Japanese/JIS	Identifies the Shift-JIS format encoding of JIS X0213
Chinese/Mac	Identifies the Mac Simplified Chinese encoding
Chinese/Win	Identifies the Windows Simplified Chinese encoding
Chinese/GB2312	Identifies the GB_2312 Chinese encoding
<p>Strings in source files containing UTF-16 characters will be converted automatically to UTF-8 regardless of the encoding specified on the source file.</p> <p>The UTF-8 encoding is a multibyte character encoding derived from UTF-16. Like UTF-16 it can represent every character of all languages, but unlike UTF-16, it is backward compatible with ASCII, using only one byte for representing ASCII characters.</p>	

3.1.14 Use of International Characters in PLC Strings

You can store international Strings in PLCs with ProSoft i-View just as easily as you do with ASCII strings. ProSoft i-View will use the source file encoding identification to decode/encode strings onto raw bytes in the PLC.

When storing international Strings into PLCs you must expect the number of bytes used, and thus the PLC string length, to be larger than the number of characters the string actually holds. This is particularly notorious when storing Chinese or Japanese strings in PLCs.

The UTF-8 encoding, for instance, can use up to 6 bytes per character in a PLC. However, this does not affect how strings are allocated in ProSoft i-View or the behavior of String methods and operators in expressions, since these always refer to actual characters and actual character lengths regardless of encoding.

Of course, if you only use English or ASCII characters with an encoding that is backward compatible with ASCII, or you use the Latin1 encoding, only one byte per character will be allocated in your PLC to store strings.

3.1.15 Expressions

Expressions are allowed on several attributes and provide an advanced way to customize various aspects of the interface and behavior of ProSoft i-View projects.

Expressions allow for combining tags with operators to produce custom results. Tags are referenced in expressions by using their names as entered in Column A. There are also several system variables that can be used for special uses.

Event Driven Architecture

Expressions in ProSoft i-View are stored in a compiled form and are executed by an event-driven engine. The execution engine keeps expression reference information in a way that value changes trigger expression evaluation. The engine is not endlessly executing 'for' loops but only change events.

Expressions keep references to referring expressions to create a tree like network where all expressions have links to other expressions. When a PLC tag changes, or an user interacts on a control, a change event is triggered. This event, that occurs at some point in the expressions network, is propagated through the relevant links to reach only the expressions that need to know about it, generally only a few.

The result is that expressions execution time is basically independent of project sizes or the total number of expressions defined in projects. The Event Driven Architecture is specially suitable for running ProSoft i-View in the constrained environment of a mobile device and still be able to support a very large number of tags with no noticeable performance penalties.

Using Expressions

ProSoft i-View expression syntax is based on the open source Ruby scripting language syntax. For basic operations this syntax is similar to that of the 'C' programming language and virtually identical to all modern scripting languages.

The Ruby language was chosen because it features a clean, easy to learn, object-oriented syntax with a particular focus on expressions allowing for practical ways to represent and dealt with several data types and formats with great flexibility. ProSoft i-View supports most operators including all common Logical, Arithmetic and Comparison operators, as well as commonly used Ruby functions and methods.

Support of Ruby expressions in ProSoft i-View is a subset of the Ruby language. Expressions are not, and do not pretend to be a complete implementation of Ruby. In some cases we provided a single way to accomplish something that can be done in several ways on Ruby, and in other cases we integrated several functionalities in single methods instead of implementing all of them. So it is important to refer to this manual if you are also using a Ruby tutorial to determine what it is actually supported on ProSoft i-View and which behavior differences may apply.

For those who already used Ruby, one of the most obvious differences between 'pure' Ruby and ProSoft i-View is treatment of boolean values. Ruby treats everything as objects, including numbers, while ProSoft i-View keeps the traditional 'C' like behavior. For example, in Ruby any number used in a boolean expression is a *true* value even if it holds zero. ProSoft i-View, on the other hand, will still interpret 0 (zero) as *false* and non-zero as *true*, in the traditional sense of earlier programming languages, and hopefully in accordance to what PLC programmers would expect or feel more natural.

NOTE: If you are not familiarized with Ruby and want to test some ProSoft i-View expressions before you start using them in your project you can try them on any of the available Ruby interpreters to help you understand how they work.

On any Mac OS X or Linux computer open the Terminal and type **irb**. This will launch the interactive Ruby interpreter. On the >> prompt type or paste any expression you would like to try.

Ruby can be installed on Windows as well. Refer to this to begin with:

<http://www.ruby-lang.org/en/downloads/>.

You can even type Ruby expressions right on your Internet Browser!. Go here:

<http://TryRuby.org/>

Just type or paste the expression examples in this manual on any of the mentioned tools and see whether they give the expected results.

Values used in expressions or assigned to tag attributes using expressions are represented in engineering units. ProSoft i-View does not have a notion of PLC raw values except for writing or reading values from PLCs. This is important in order to understand why assigning a 'value' to a INTERNAL tag may not give the same result than writing it to a PLC tag and reading it back. Differences are due to the implicit scaling/descaling and type conversion performed on PLC tags. See the 'value' attribute description for a discussion on this subject.

When using expressions in your project you must know the following:

- Tag names in expressions are case sensitive. This means that a tag named int_value will not be the same as a tag named Int_value.
- Logical or Comparison operators assume non-zero values to be true and zero values to be false. The result of a Logical or Comparison operator is always a value of 0 or 1.
- Comparison operators are non-associative. This means that expressions such as value=a<b<c; are not valid. You must use value=a<b && b<c; instead.
- Assignments in expressions are not supported except for assigning values to attributes. Therefore expressions such as value = condition && (int_tag = 3); will cause a syntax error on the second assignment operator. Do not confuse the assignment operator = with the the equality operator == which is fully supported.
- An expression is executed only when at least one of the referred variables or tags change. The process is totally transparent and integrators might not need to know about it. However, keeping the event driven nature of ProSoft i-View in mind can help integrators to understand why and when PLC tags will be written or alarms will trigger as a consequence of an user interaction or PLC Tag change that originated a cascade of change events.
- Expressions containing Logical operators are no exception to the event driven design. They will be executed even if a change occurs on the right side of the Logical operator. For example the expression value = condition1 && condition2; will be always false if condition1 is false, however it will execute anyway as a consequence of a change on condition2. The expression result may not change (false), but the engine will still send a change event to any referring expressions, which could potentially cause other effects such as a PLC tag rewrite if the expression was linked to a PLC tag.
- Expressions are not allowed to create circular or recursive references. This means that a result of an expression can not be refitted to another expression that ultimately would send a change event to the originating expression. This is not allowed either on the same expression or through intermediate expressions. For example the following row int_value INT LOCAL "value=int_value+1;" is not valid because the int_value tag creates a circular reference around the 'value' expression.
- Expression values are internally stored as double float values (64 bits) and all arithmetic and logical operations are performed as double float operations. Values read from PLCs including BOOL and INT tags are scaled and converted to double float type before they are used in expressions. This has no other relevance than you do not have to care about types, type matching or type conversions while using expressions in ProSoft i-View.

Data types in Expressions

Variables and Expressions in ProSoft i-View support three basic data types, **Numbers**, **Strings** and **Arrays**. Appropriate *operators* and *methods* allow for conversion among types and to perform custom operations with great flexibility. See the following sections for a discussion on methods and operators.

Mixing Numbers, Strings or Arrays is only possible through the use of the appropriate *operators* and *methods* that result in compatible types. An immediate consequence is, for instance, that you are not allowed to add directly a number to a string unless you convert the string to a number first. Thus, some operators take particular significance depending on type as it is commonly accepted on most modern scripting languages including Ruby. See next sections for a further discussion on this and other subjects.

Numeric values.

Numeric values in expressions and variables are internally stored as Double Float values (64 bits) Values read from PLCs including BOOL and INT tags are scaled and converted to Double Float type before they are used in expressions. All Arithmetic, Logical and Comparison operations are performed as Double Float operations, so you may never expect to obtain truncated values from arithmetic calculations.

The above statement may change in the future to give support for true integer arithmetic. Currently, an implicit conversion to an integer type is only performed for bit or bitwise operations on numbers, and indexed access to string or array elements. In other cases you can use the to_i method to explicitly get the integral part of a numeric value according to your needs.

Constant numbers can be represented with optional decimal point and a base 10 exponent. Additionally, hexadecimal and binary notations are supported by using the *0x* or *0b* prefixes. The special forms *true*, *false*, *+inf* and *-inf* are supported as well.

Examples:

-1.42 (decimal representation)
1.1666e+2 (decimal representation with exponent)
0xe0af (hexadecimal representation)
0b011011101 (binary representation)
true (same as 1)
false (same as 0)
-inf (very big negative number)
+inf (very big positive number)

Strings.

Strings are arbitrary sequences of characters that are manipulated as a whole. Strings can be read from or written to PLCs,. Several operations can be performed on strings such as concatenating, splitting or substring extraction by using the appropriate *operators* or *methods*. String literals are represented enclosed in double quotes.

Example :

"This is a literal string"

"Дискретные датчики"

"ピーエルシーのアラーム表示"

Arrays.

Arrays are indexed collections of objects. Each element in an array is associated with and referred to by an index.

Array indexing starts at 0. A negative index is assumed relative to the end of the array, that is, an index of -1 indicates the last element of the array, -2 is the next to last element in the array, and so on.

Arrays can hold any objects such as Numbers, Strings or other Arrays. Arrays can be read from or written to PLC or can be created on demand in expressions by just enclosing their elements in square brackets.

Example:

["element at index 0", 123.4, [33, temperature]]

The above expression represents an array of three elements.

At index 0 we have a literal string: "element at index 0".

At index 1 we have a numeric value: 123.4.

At index 2 we have an array of 2 elements with the number 33 and the variable 'temperature' as their components

Elements of the referred array can be accessed by index as shown next:

["element at index 0", 123.4, [33, temperature]][1] would return 123.4

["element at index 0", 123.4, [33, temperature]][-3] would return "element at index 0"

["element at index 0", 123.4, [33, temperature]][-1][0] would return 33

["element at index 0", 123.4, [33, temperature]][2][1] would return the value of *temperature*

Operators and Operator Precedence

The following table shows the available operators and its precedence. The table lists all operators from highest precedence to lowest.

()	Parentheses (grouping).	from inner to outer
! + -	Logical NOT, unary plus, unary minus.	left-to-right
. () []	Method selection, Method/Function call, Array or String subscript	left-to-right
* /%	Multiply, divide, Modulo	left-to-right
+ -	Addition, subtraction	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise XOR, Bitwise OR	left-to-right
< <= > >= != ==	Comparison operators	not associative
&&	Logical AND	left-to-right
	Logical OR	left-to-right
?:	Ternary if then else	right-to-left
=	Assignment	not associative
If than else	If than else clause	right-to-left

Operators are used in the usual way as per the Ruby or “C” language. Depending on data types involved the same operator may have a different meaning.

System Variables

There are a number of system variables that can be used in expressions as if they would be regular tags. The following variables are provided.

\$SMPulse10s \$SMPulse30s \$SMPulse60s	Number (read only)	They generate a square wave signal with the period implicit on the variable name. They can be used to implement a Keep-Alive tag, to write periodically a value on a PLC, or to trigger periodic events for any purpose.
\$SMAckButton	Number (read only)	Variable linked to the Acknowledge button on the alarms panel. Goes to 1 when the button is pushed and 0 when it is released.
\$SMCommState	Number (read only)	A value indicating the current communication state of ProSoft i-View. Possible values are the following: 0 - Communications running with all PLC connections linked. 1 - Monitor is switched off. 2 - One or more PLC are not linked or a new connection is in course. Partial link state. 3 - General communications error. No communication is established. This variable can be used to implement alarms related to PLC reachability or to show/hide interface elements depending on PLC availability.
\$SMCurrentPage-Name	String (read/write)	Name of currently displayed page
\$SMDate	String (read only)	Twext representation of the current date and time in the following format “yyyy-mm-dd-hh-mm-ss”

Functions Methods, and more about Operators

Methods can be applied to intermediate expressions or variables to perform type conversions or to achieve particular requirements. They are like computer language functions that perform particular tasks. Not all *methods* are applicable to all types and their meaning can vary depending on type. *Methods* are invoked by appending a dot (*method selector operator*) followed by its name to the variable or subexpression they apply to.

Operators can also have a different meaning depending on the type of data they are applied to.

In the following tables we represent the meaning of the applicable operators and methods depending on data type.

Numeric operators and methods.

num operator num2	Arithmetic, comparison, logical operators produce the expected results. Available operators are listed on the operators precedence table shown earlier. The bitwise and complement operators extract the integral part of the operands before computing the result Example: 2+2 returns 4
num[n]	Returns bit n from the integral part of num. Bit 0 is the least significant bit. The result can be only 0 or 1. Example: 3[0] returns 1 Example: 3[1] returns 1 Example: 3[2] returns 0
num.to_i	Returns the integral part of num. Example: 3.666.to_i returns 3
num.to_f	Returns num
num.to_s num.to_s(fmt)	Returns a string representation of num optionally formatted according to fmt. For a description of possible format specifiers refer to the format function. Example: 3.666.to_s("%02d") results in "03" Example: 3.666.to_s("%02f") results in "04" Example: 3.666.to_s results in "3.666" Example: 25.to_s("%02.1f °C") results in "25.0 °C" (Note that specifying a format in to_s is not a standard feature of Ruby)
num.chr	Returns a string containing a single character represented by the character code num. Example: 72.chr would return "H"
num.abs	Returns the absolute value of num. Example: (-3.66).abs would return 3.66
num.round	Returns num rounded to the nearest integer. Example: 3.66.round would return 4

The Format Function

The built-in function *format* returns a string formatted according to a format string like the usual printf conventions of the C language. In addition, *format* accepts %b for binary. ProSoft i-View. *format* specifiers adopt the following form:

%<flags><width><.precision>specifier

Where *specifier* is the most significant one and defines the type and the interpretation of the value of the corresponding argument ('<' and '>' denote optional fields).

Note that the string provided for the *format tag attribute* or the format string passed to the *to_s* method have a slightly different purpose and may have less available options.

The following format conversion specifiers are available:

FORMAT SPECIFIER	Description	<i>format</i> attribute support	<i>to_s</i> method support	<i>format</i> function support
b	Binary integer	YES	YES	YES
c	Single character	NO	YES	YES
d,i	Decimal integer	YES	YES	YES
e	Exponential notation (e.g., 2.44e6)	YES	YES	YES
E	Exponential notation (e.g., 2.44E6)	YES	YES	YES
f	Floating-point number (e.g., 2.44)	YES	YES	YES
g	Use the shorter of e or f	YES	YES	YES
G	Use the shorter of E or f	YES	YES	YES

FORMAT SPECIFIER	Description	<i>format</i> attribute support	<i>to_s</i> method support	<i>format</i> function support
o	Octal integer	NO	YES	YES
s	String or any object converted using <i>to_s</i>	NO	YES	YES
u	Unsigned decimal integer	YES	YES	YES
x	Hexadecimal integer (e.g., 39ff)	YES	YES	YES
X	Hexadecimal integer (e.g., 39FF)	YES	YES	YES

For the meaning and possible contents of the optional *flags*, *width*, and *precision* fields refer to the `sprintf` specification:

<http://www.cplusplus.com/reference/clibrary/cstdio/sprintf/>

Note that since there is no need for it the *length* field is not available

The if then else clause and the ternary conditional operator

Two forms of conditional execution are provided:

The ternary conditional operator provide conditional execution of expressions. Its syntax is the following:

```
expr ? expr1 : expr2
```

Returns *expr1* if *expr* is not zero (true) or *expr2* otherwise.

The *if then else* clause provide conditional choice of expressions. It is used as follows:

```
if expr [then] expr1 [else expr2] [end]
```

Executes *expr1* if *expr* is not zero (true). If *expr* is zero (false) *expr2* is executed instead. Items between brackets are optional.

The *if then else* clause (2) differs from the ternary conditional operator (1) in the following ways:

(1) executes when any of *expr*, *expr1*, *expr2* generate a change event. The result is always updated and will be consistent with the values of *expr*, *expr1* and *expr2* at all times. The execution will in turn trigger relevant change events up the expressions tree just as any expression would do.

(2) only attends to *expr* change events. Any changes in *expr1* or *expr2* will not have an effect until *expr* executes. Furthermore, if *expr* is *false* and *expr2* was not specified, the execution tree is trimmed at this stage and no further execution up the expression tree will happen.

is useful in cases where you want to achieve a differential effect, for example to trigger an event when a condition goes from *false* to *true* but not the oposite. This is not possible with (1) because it will always execute both ways. Consider the following:

```
start    BOOL    INTERNAL    label = "Start Button"; style = "Button";  
write_access=0;  
stop    BOOL    INTERNAL    label = "Stop Button"; style = "Button";  
write_access=0;  
motor   BOOL    C1          label = "Motor State"; value = if start  
then 1 else (if stop then 0);
```

The previous lines will display a Start and a Stop button. When the Start button is touched 1 will be written to C1. When the stop button is touched 0 will be written to C1.

Another use of the *if then else* clause is the implementation of a counter:

```
reset    BOOL    INTERNAL    label = "Reset Button"; style =  
"Button"; write_access=0;  
increment  BOOL    INTERNAL    label = "Tick Button"; style =  
"Button"; write_access=0;  
counter   DINT    HR1          label = "Counter"; value = if  
reset then 0 else (if increment then counter+1);
```

The Reset and Tick buttons will provide in this case the interface for a counter value written to the PLC.

Now consider the following case

```
color1          UDINT    INTERNAL    label    =    "Color    1";  
write_access=0;  
color2          UDINT    INTERNAL    label    =    "Color    2";  
write_access=0;  
colorSelection  BOOL     INTERNAL    label = "Select Color"; style  
= "Switch"; write_access=0;  
coloredValue   DINT     HR1          label = "Colored Value";  
color = colorSelection ? color1 : color2 ;
```

The coloredValue color will be always updated according to colorSelection after any change of color1 or color2.

Putting it all together. Advanced Expressions Examples

Converting an arbitrary number of seconds to hh:mm:ss format

The following expression shows how to get a string in the form 'hh:mm:ss' from a numeric value containing seconds. In this example *x* contains the total number of seconds to be converted to the desired format.

```
value = [(x/3600).to_s("%02d"), ((x%3600)/60).to_s("%02d"), (x%60).to_s("%02d")].join(":") ;
```

The operators % and / are used to calculate hours, minutes, seconds as numeric values. These are then truncated to integer with the *to_i* method and successively converted to formatted strings with *to_s*. The resulting individual strings are embedded into an array and then joined by means of the *join* method using ':' as separator.

Given the following two rows:

```
x      DINT      HR1      label = "Total Seconds";  
t      STRING  INTERNAL label = "Time"; value =  
                               [(x/3600).to_s("%02d"),  
                               ((x%3600)/60).to_s("%02d"),  
                               (x%60).to_s("%02d")].join(":") ;
```

when HR1 holds **3661**, which is 3600 seconds (1 hour) + 60 seconds (1 minute) + 1 second,

the second row will display **01:01:01**

Instead of using the *join* method we could have used the *format* function in a possibly more convenient way. Consider the following:

```
value = format("%02d:%02d:%02d", x/3600, (x%3600)/60, x%60) ;
```

in this case the format specifiers in the format string are just replaced with the relevant time values.

Calculating seconds from a string having the hh:mm:ss format.

Just to illustrate what expressions allow to do let's try now to get the original seconds value from a string already in the hh:mm:ss format. To do so we can use the following expression:

```
value=3600*t.split(":").fetch(-3,0).to_i + 60*t.split(":").fetch(-2,0).to_i + t.split(":").fetch(-1,0).to_i;
```

In this case we extract separately the hours, minutes and seconds as numeric values from the string, we multiply them by 3600, 60 and 1 respectively and then sum them to get the total number of seconds. The extraction of each value from the original string is performed by the *split* method using ':' as delimiter. The relevant element from the split array is obtained with the *fetch* method. We use 0 as the default value for fetching.

Note that we could have used simple array indexing such as `t.split(":")[-3]` to get each part of the original string but this would lead to potential out of bound errors if the original string had some missing part. Particularly, if the original string did only contain minutes and seconds, such as "50:30" (50 minutes, 30 seconds) the referred indexed expression would give an out of bounds error as it would attempt to access a non existing element (the one before the first one). Note also that in all cases we use negative indexing because we interpret that the last part is always meant to be the seconds, the previous to the last one the minutes and so on.

The proposed expression can be optionally optimized by storing the split string in a temporary variable so that the splitting is only performed once. If we apply this optimization the final solution would look as follows:

```
tspt      STRING[3]  INTERNAL  value = t.split(":") ;
seconds  DINT       INTERNAL  value = 3600*tspt.fetch(-
3,0).to_i + 60*tspt.fetch(-2,0).to_i + tspt.fetch(-1,0).to_i;
```

Creating a row that alternates between displaying the current time and an arbitrary value

In this example we will create a row that shows a living digital clock showing the current time. Every 5 seconds the time is alternated with a temperature value given in a variable named *temp*. In order to achieve this we enter the following expression in a row.

```
value = $SMPulse10s ? "Time: "+$SMDDate.split(" ")[1] : "Temperature: "+temp.to_s("%3.1f")+ " °C" ;
```

We use the ternary operator to switch between the time and the temperature depending on the \$SMPulse10 system pulse variable. For the clock we take \$SMDDate and discard the date portion by splitting it out. The temperature is presented formatted with a custom prefix and suffix appended to the actual value. We can alternatively use the format function to simplify a bit some portions of the expression

```
value = $SMPulse10s ? format("Time: %s", $SMDDate.split(" ")[1]) : format("Temp: %3.1f F",
9/5*celsius+32) ;
```

3.2 Rockwell RSLogix 5000 as a Data Source Generator

ProSoft i-View accepts files generated directly from Rockwell's RSLogix development environment as data sources. To import RSLogix files, follow the following steps.

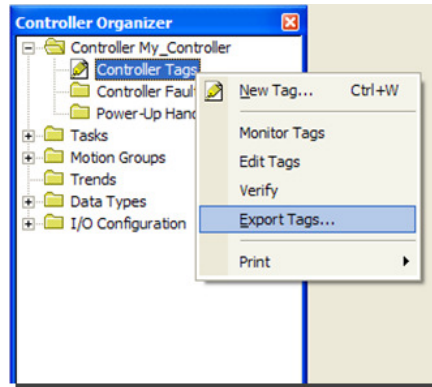
3.2.1 Building a Project in RSLogix 5000

Create a new RSLogix project or open an existing one. Instructions for programming a PLC are beyond the scope of this manual. Please refer to your RSLogix documentation.

3.2.2 Exporting Controller Tags from RSLogix 5000

Unfortunately, RSLogix does not provide a way to select and export selected tags to a file. However it is possible to do a bulk export containing all global controller tags.

In RSLogix, right-click **CONTROLLER TAGS**. From the dropdown menu, select **EXPORT TAGS**.



RSLogix will generate a special CSV file containing all global tags present in the project. Tag source files generated in this way can contain attributes as described in Attributes (page 24). If you need to specify attributes for ProSoft i-View, you can put them in the *Description* column of the *Controller Tags* view in RSLogix. This is equivalent to filling column D on an Excel-generated source.

Note: Files with a CSV extension created by RSLogix as described in this section are structured differently from CSV files created in Excel as described earlier. However, ProSoft i-View will recognize them as well. Since RSLogix does not provide a convenient way to select a particular set of tags before exporting, you may need to edit the exported file with Excel or a Text Editor to adapt it to your needs. In this case you do not need to change its overall structure and format - just eliminate redundant rows and you will be done.

3.3 Editing Source Files in a Text Editor

Since source files created in any of the ways described above are simply CSV files, they are essentially text files as well. Integrators and advanced users can conveniently edit them with any text editor such as WordPad or NotePad, and save them in plain text format. It is also possible to create either CSV or TXT files from scratch in a text editor.

If you choose to use a text editor to create or edit source files, you need to be aware of the double-quoting that Excel applies to text columns containing quotes. So you will need to add them manually where necessary. This is not anything particular to ProSoft i-View, it is just how CSVs are.

Double-quoting is particularly relevant for column D of a ProSoft i-View source file, because many attributes expect a text string as their value, which may be enclosed in quotation marks. When entering attributes in Excel, you do not need to take any special care on this. However, the same file edited in NotePad will reveal the existence of extra quoting that Excel automatically inserts to avoid text conflicts and to meet the CSV spec.

As an example, assuming an Allen Bradley controller tag name is *main_switch*, the row referred to in the Attributes section (page 24) would look like this in the text editor:

```
main_switch, BOOL, TAG, "ord := 1 ; section := ""GENERAL"" ; label :=  
""Main Run/Stop Switch"" ;  
comment := ""Main Process Start/Stop"" ; access := 3 ; write_access := 5;"
```

If you choose to create and edit your files using only a text editor, the recommended way is to use tab-delimited CSV formatting, in which commas are replaced by tabs, which ProSoft i-View also supports. This format is much more convenient to use in text editors because it is more readable and you can avoid all the double-quoting hassle. The row shown above will look as follows if tabs are used as delimiters instead of commas:

```
main_switch    BOOL    TAG    ord := 1; section := "GENERAL"; label  
:= "Main Run/Stop Switch"; comment := "Main Process Start/Stop"; access := 3;  
write_access := 5;
```


4 File Import into ProSoft i-View

In This Chapter

❖ Source Files Supported by ProSoft i-View.....	65
❖ Other Files Supported by ProSoft i-View.....	66
❖ Custom Company Logo.....	67

ProSoft i-View features an embedded Web server to enable users to access to their file contents and to perform various operations. Follow the steps below to use the integrated Web server.

- 1 Go to the **FILE SERVER** tab.

Note: In order to access files you must log in to the "administrator" account. The initial password is "admin." ProSoft i-View sets itself to this account on first launch. See User Accounts (page 71).

- 2 Switch the File Server **ON**.
- 3 With ProSoft i-View's File Server started, copy the displayed IP address to your PC's Web browser. If you use Apple's Safari browser, you can simply click on the relevant 'Bonjour' link.

- 4 Now use the available options on the displayed Web page to move files to and from ProSoft i-View. You can upload any files and store them in your iPhone/iPod.



4.1 Source Files Supported by ProSoft i-View

ProSoft i-View supports the following source file formats for import:

- csv - generated by Excel or Open Office
- txt - any text file with tab, comma or semicolon field delimiters generated by Excel, OpenOffice, NotePad etc.
- csv - generated by RSLogix

Note: The recommended way to generate source files is to begin with one of the provided examples, and then continue by editing what is missing.

4.2 Other Files Supported by ProSoft i-View

ProSoft i-View is able to store and display a number of file formats in addition to source files. File types supported for display include but are not limited to:

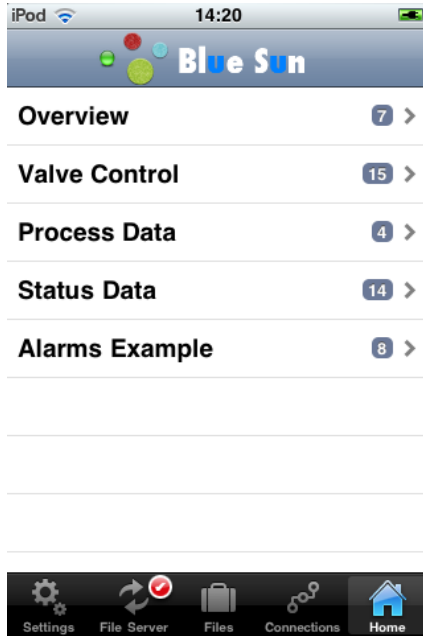
- Excel (.xls)
- PDF (.pdf)
- Powerpoint (.ppt)
- Word (.doc)
- Rich Text Format (.rtf)
- Image files (.png, .jpg, .gif, ...)

The capability to store and view files can be useful for document-specific configurations, or simply for moving documents between computers.

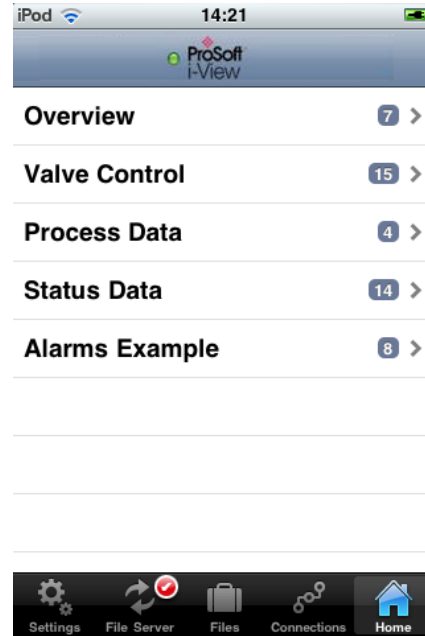
ProSoft i-View is able to store any file type, even those that are not supported for display.

4.3 Custom Company Logo

You can place a custom logo on top of the *Home* view when the list of pages is seen. You do this by using the available options on the Web page provided by the integrated Web server. Below are two screenshots of the same screen - one with an imaginary company logo and one with the default logo.



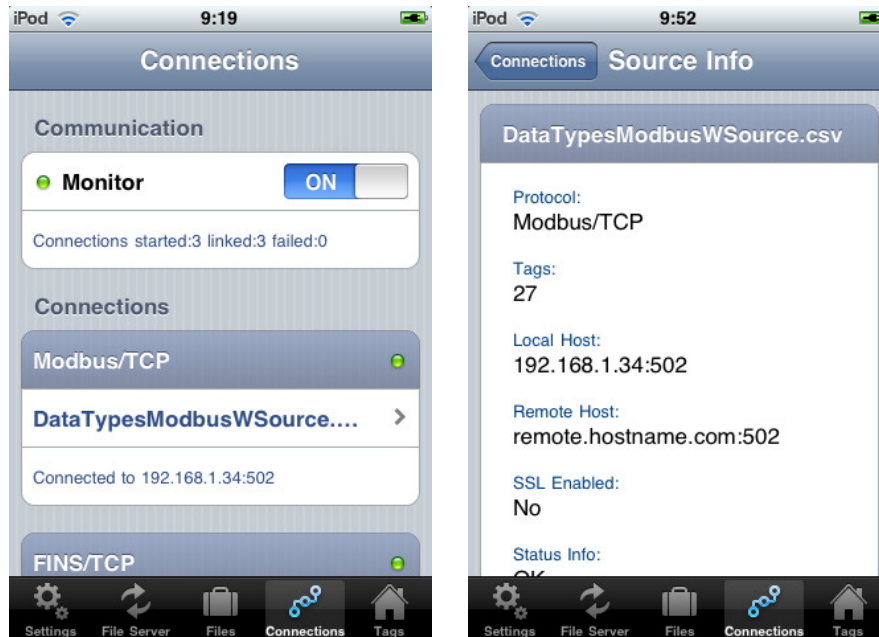
Imaginary Company Logo



Default Logo

5 Connections

Connections represent links between ProSoft i-View and PLCs. Each connection can have one or more associated sources. Sources that share the same communication attributes are automatically grouped in a single connection. You can look at connection states and their associated sources in the *Connections* tab view. Source information, including potential parsing errors, is also displayed.



6 User Accounts

In This Chapter

- ❖ Restrictions for Non-Administrator Users 72
- ❖ Managing Accounts 73

User accounts allow the administrator to determine which process variables will be visible or editable for a particular use or user depending on specific variable attributes. Relevant attributes for this purpose are *Access* and *Write_access*. See *Attributes* (page 24).

Only users who have an access level equal to or greater than the one set for a particular variable will be able to view or edit it.

The purpose of user accounts is to make it possible for a PLC program developer or integrator to create a pre-configured solution for a particular customer and limit that customer's ability to change the application. An end user does not usually need to change communication settings or modify attributes. This is normally a developer's responsibility. Therefore, by setting user accounts and keeping the administrator password private, a developer can choose to provide customized, safe access to a monitored process.

User accounts can also help prevent unauthorized persons from interacting with remote processes by getting physical access to an iPhone with ProSoft i-View installed.

Two accounts are set up by default with the following names, passwords and access levels:

USER	PASSWORD	ACCESS
administrator	admin	9
nobody		0

The administrator account allows you to create additional accounts with intermediate access levels, as well as to select active accounts.

It is strongly recommended that you change the default passwords once you have a running configuration in place. Note that there is no way to retrieve a lost password. The only way to recreate the original passwords is to remove ProSoft i-View from your iPhone or iTunes and reinstall from the App Store. Be sure your source files are conveniently backed up, in case you should ever need to do this.

6.1 Restrictions for Non-Administrator Users

Only the administrator account has full access to all the application. Non-administrator accounts have restricted access to deployment features, and not all tabs are available to them. The following screenshots show the available tabs for the Administrator (left) and for a regular user (right).



administrator user



regular user

6.2 Managing Accounts

Accounts are managed in a navigation interface similar to that of the Contacts application. The screenshot below shows that a new account named "Tommy" has been created, which has an access level of 3 and is active.



To log in as a particular user, tap on **CURRENT ACCOUNT**.

You can also make ProSoft i-View ask for the current account password at launch. To do so, set the **AUTOMATIC LOGIN** switch in *Settings* to **OFF**.

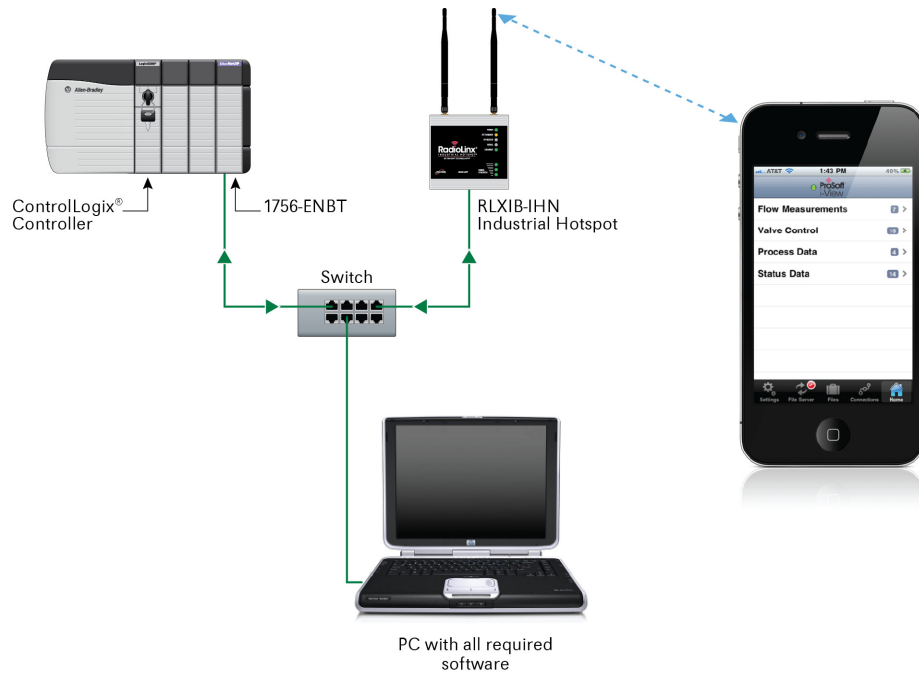
7 Network Settings for Local Access

In This Chapter

- ❖ PLC Settings for Local Access 76
- ❖ ProSoft i-View Settings for Local PLC Access..... 77

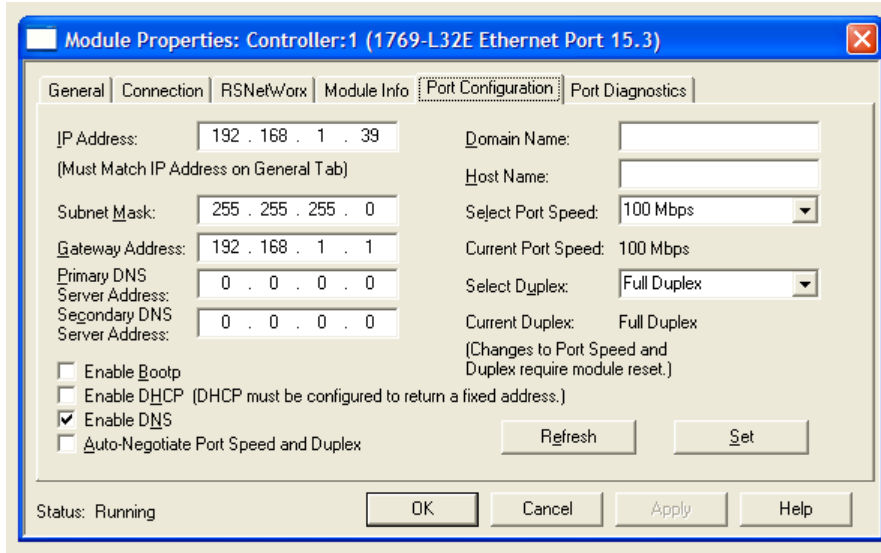
ProSoft i-View uses wireless TCP/IP technology to connect to and communicate with PLCs. Direct access from a local network requires both devices to be on the same subnet. The PLC acts as the communications server, and the iPhone or iPod Touch is the Client.

The following picture shows a typical setup using the recommended industrial wireless hardware, but basically any WiFi router will do it.



7.1 PLC Settings for Local Access

For EIP/Native protocol and Allen Bradley controllers, use the RSLogix 5000 tool to set a fixed local IP for the PLC in the ethernet *Module Properties* dialog box.



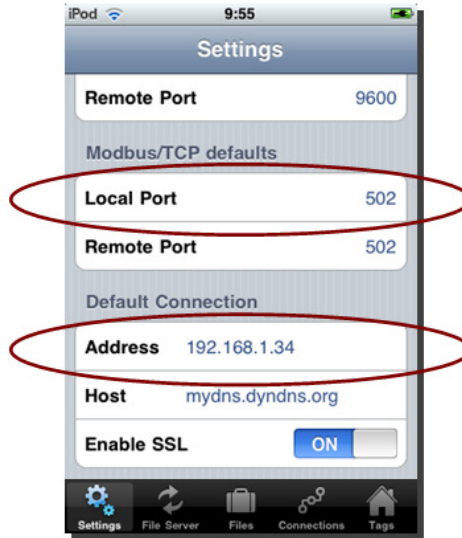
For EIP/PCCC protocol, use Allen Bradley's RSLogix 500 tool to set a fixed local IP for the PLC on the *Channel Configuration* panel

For PLCs or devices based on the Modbus TCP/IP protocol, consult the relevant vendor documentation to know how to set ports and addresses.

7.2 ProSoft i-View Settings for Local PLC Access

ProSoft i-View configuration depends on whether you have specified communication attributes for source files. For more information, see *Attributes* (page 24).

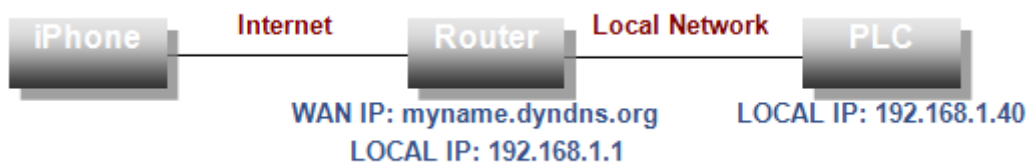
For sources without communication attributes, ProSoft i-View uses the default ports and addresses entered into the relevant fields in the *Settings* tab view. Fill in the values you used for the PLC settings.



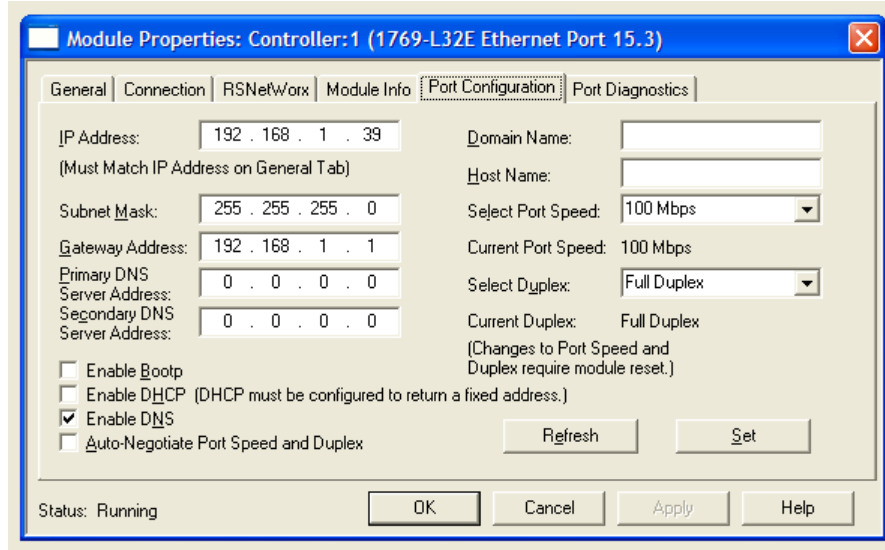
8 Network Settings for Remote Access

ProSoft i-View is designed to communicate with PLCs without using dedicated servers or any specific software installed on a PC. ProSoft i-View communicates with PLCs using industrial protocol commands.

For remote connection, a GPRS or DSL router is needed at the PLC site. The router acts as a bridge between the LAN (Local Network) where the PLC is installed and the WWAN or WAN (Internet) that a remote iPhone or iPod Touch will have access to. This figure shows a standard setup.



- 1 Determine the LOCAL IP address of the GPRS or ADSL router. PLCs need to know the router address as it is the gateway to the internet.
For EIP/Native protocol and Allen Bradley controllers, use the RSLogix tool to set the fixed local router IP (gateway) in the ethernet *Module Properties* dialog box.

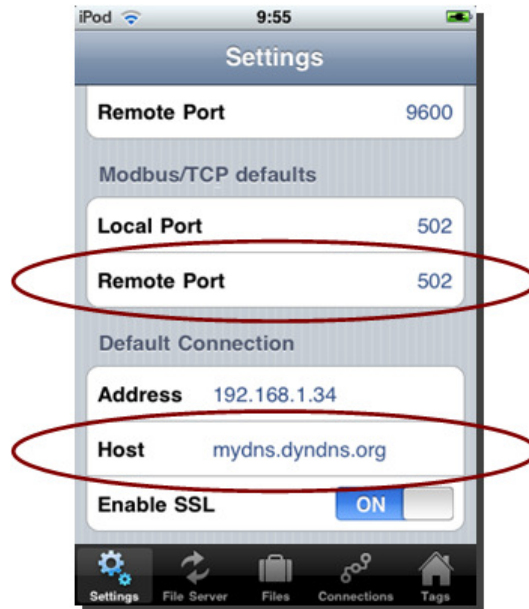


For EIP/PCCC protocol, use Allen Bradley's RSLogix 500 tool and set the gateway IP on the *Channel Configuration* panel.

For Modbus TCP/IP based devices, refer to the vendor's documentation.

- 2 Log on to the GPRS or DSL router and configure NAT options to set up a bridge between the WAN and your PLC local address and port. Note that the default port number is **44818** for Ethernet/IP and **502** for Modbus TCP/IP. The protocol on the router must be set to **TCP/IP**. Refer to your router documentation for details.

- 3 If you have a fixed IP address, enter it as such in ProSoft i-View, either as a *Settings* default, or embedded in source files. .
- 4 If your router accesses the WAN through a dynamic IP then you must create an account with a dynamic DNS services provider such as www.dyndns.org, and configure your router to notify of IP changes. In this case, enter in ProSoft i-View the name you choose for your dynamic DNS. The port number must still be the one configured in the NAT section of your router.



9 Security

In This Chapter

- ❖ Validation Codes 81
- ❖ Background Task Processing..... 84

ProSoft i-View networking security is based on TCP/IP technology and depends in part on the security features available in the router installed at the PLC location.

For local connections through WiFi, security is provided by the wireless network security protocol in use. WPA and WPA2 with a strong password is the recommended security protocol.

For remote connections, an iPhone or iPod touch is able to make use of secure data tunnels by enabling VPN. If your router supports L2TP/IPSEC or PPTP, then you will be able to create this kind of connection. Most medium to high-end DSL or Cable routers support at least PPTP. VPNs Client connections are configured on the iPhone with the General Settings App.

Some routers can be loaded with a SSL certificate and be configured to bridge incoming SSL requests from the WAN to unencrypted TCP on the LAN side. ProSoft i-View supports TLS-SSL encryption. You can activate TLS-SSL in ProSoft i-View to provide communications confidentiality if your router supports SSL/TCP bridging.

For most protocols, ProSoft i-View provides an independent way to protect users from undesired access by persons using uncontrolled ProSoft i-View copies. This is done by setting a Validation Code both in the PLCs and ProSoft i-View which will prevent ProSoft i-View from accessing PLCs unless both codes match. The next section describes validation codes and how you can set them up.

Finally, physical access can compromise security. It is relatively easy for an unauthorized user to gain physical access to a device and run a remote monitoring application. To prevent this from happening, ProSoft i-View's user accounts provide password-based security. If you turn **OFF** the **AUTOMATIC LOGIN** switch in the ProSoft i-View *Settings* tab, a password key will be requested each time the application is launched. Furthermore, Apple provides a service for blocking lost or stolen devices so that no one is able to access data or execute apps in them until the real owner reactivates them.

9.1 Validation Codes

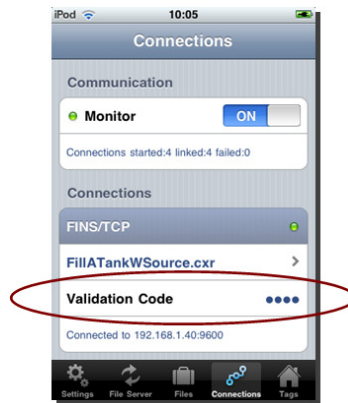
For Allen Bradley EIP/Native and EIP/PCCC, ProSoft i-View requires a password code to be held by the PLC, which is queried on each connection. This password must be stored in your PLC as a 16-bit hexadecimal value (0 to FFFF) and must match the value specified in *Validation Code* for connections to that PLC. In most cases, this security measure alone is enough for simple applications.

Validation Codes are stored in PLCs in the following memory address or tag depending on protocol.

PROTOCOL	Default Tag	REMARKS
Modbus TCP/IP	(Not Available)	See note below.
EIP/Native	SMValidationTag	Any INT value. This tag must be present in order for ProSoft i-View to communicate. Set to 0 initially to avoid having to enter it in ProSoft i-View during development stages.
EIP/PCCC	N98:0	Any INT value. This tag must be present in order for ProSoft i-View to communicate. You may need to create a Data File number 98 of type Integer with at least 1 element.

The Validation Code feature is not available for Modbus TCP/IP due to the great number of Industrial devices supporting this protocol, which makes it impractical to establish a general way to implement such a feature.

Validation codes are entered in the relevant fields of ProSoft i-View's *Connections* view. ProSoft i-View will store and remember codes between connections but it will reset them to **0** for connections that changed as a result of source change.



Note that ProSoft i-View will always perform this security check. There is no way to disable or avoid it. However you can set a custom validation tag.

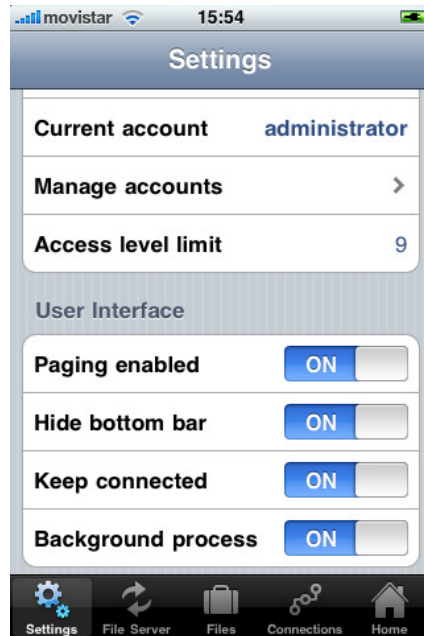
9.1.1 Custom Validation Tag

If the default validation tag interferes with your project you can set a custom one with the *validation_tag* attribute. When you use custom validation tags you must follow the following points.

- Be sure to have the same validation tag in all files sharing a single connection, (i.e with identical communication settings), otherwise one or more files will appear as dissociated on the Connections view.
- If you explicitly set the *validation_tag* attribute to the default one, you will still have to do the same in all your files sharing a connection.
- When you explicitly set a validation tag, you will have to explicitly set a non zero value for the validation code, as using 0 as validation code is explicitly forbidden in this case and doing so will always fail the validation check.

9.2 Background Task Processing

You can tell ProSoft i-View to keep connections alive even when the device (iPhone, iPod) is locked. Additionally, ProSoft i-View supports running in the background during a finite period of time on multitasking enabled devices. On the *Settings* tab, there are two switches that allow for enabling such options.



9.2.1 Keep Connected

When ON, ProSoft i-View will keep any open connection as is and will continue polling PLCs even when the device is locked. This is intended to help keep alarms and graphs updated, but it may have a negative impact on battery life.

When OFF, ProSoft i-View will close any open connection when another application becomes active or the device goes to the locked state, either by the Auto-Lock timeout or by pushing the bottom on the edge. This will prevent excessive battery drain and may reduce carrier fees in case of cellular remote connections. When the device is unlocked and ProSoft i-View becomes active again, communications with PLCs will be restarted.

Background Process

When ON, the application will continue running in the background for a finite period of time while other applications are active, making it possible for alarms and graphs to keep updating, and enabling local alarm notifications while the user is performing other tasks.

When OFF, the application will not explicitly start a procedure to run in the background, and local notifications will be disabled. However, given how the iPhone iOS behaves, the application may still be running in the background for an undetermined period of time if enough resources are available or the user does not explicitly switch to a different app.

The maximum time applications are allowed to run in the background or in the lock state is an Apple choice, and it is currently set at 10 minutes. ProSoft i-View, like any other application, will be switched off or suspended after this time has expired. Therefore, end users are advised to put ProSoft i-View in the foreground as soon as they have completed other tasks and not allowing the device to go to sleep, if it is essential to keep alarms and graphs updated at all times.

9.3 Performance

ProSoft i-View is designed to offer great performance and a satisfactory user experience with good interface responsiveness. Integrators do not usually need to take particular actions to improve performance. The app carries out a series of optimizations on tags and PLC communications that in most cases are enough to relieve integrators from having to adopt particular actions or design patterns.

Still, understanding what optimizations ProSoft i-View performs can be useful to find out why, if ever, a performance issue arose and what you can do to improve on it.

Performance matters tend to be a complex subject and extensive discussions can be made; ultimately, testing is the only way to have a trustable answer. We will just enumerate some of the techniques used in ProSoft i-View to maximize performance and will give some tips to help you getting the best of the app for your particular project.

ProSoft i-View automatically performs the following optimizations:

- Only the minimum set of tags necessary to display the interface is polled at a given time. Particularly, tags that are not used in expressions and which values are not shown on the interface at a given time are not polled at all.
- *Therefore, the less tags you link with expressions or use in expressions the better. In fact, having a large number of tags for the single purpose of showing their PLC value will not incur in any performance penalty because most of the time they will not be polled out from the PLC anyway.*
-
- If a particular PLC address needs to be read at a given time, it will only be read once per polling cycle regardless of how many instances of tags tied to that address are found in the project.
- *Consequently, you can repeat the same PLC tag address as many times as you need in your project without having to worry about any performance penalty.*
-
- The list of candidate tags to be read at a given time is applied a minimal cost algorithm to determine the less and shorter available communication frames for the relevant protocol which fulfill the entire request. The optimal solution may include reading unsolicited tags if this justify a better performance figure. The simpler scenario is reading several, almost contiguous tags, with a small gap in between. Reading all of them and then discarding the unused ones may be faster than only reading the solicited ones.
- *To help on this, you can use tags as contiguous as possible and leave the shorter possible gaps between them, however note that if you have spared tags that are used in expressions, or your users make extensive use of the plotting feature, this measure may not be as effective as you might expect or even contraindicated, because basically you may not always beat the already optimal set of commands that ProSoft i-View would generate anyway.*
-
- Tags used in alarms or used in expressions are permanently polled to guarantee interface consistency. These tags are also candidates to automatic optimization and applied all the above techniques, but since they are read very often they offer the best opportunity for integrators to optimize their project in an user noticeable way.
- *As said, this is by far the best thing you can do to optimize your project. The key is to have as few as possible tags tied to expressions and to keep them in contiguous PLC memory addresses, specially with no intercalations of tags that are not used in expressions (i.e. tags that are candidates to be stripped off from readings). In other words, keep the tags used in expressions or alarms as packed as possible in PLC memory and do not interlace them with the remaining tags.*

- *If you use a lot of boolean tags consider boolean arrays on EIP/Native or bit access on registers on other protocols. Bit readings on registers are faster because you get several boolean values with a single register read.*

Finally, if you are concerned about performance or you are planning a really big project, do some planning following the above recommendations before even starting your project. Then profile periodically your project for performance as you are developing it and adding more rows to it.

10 Pre-installed Examples

In This Chapter

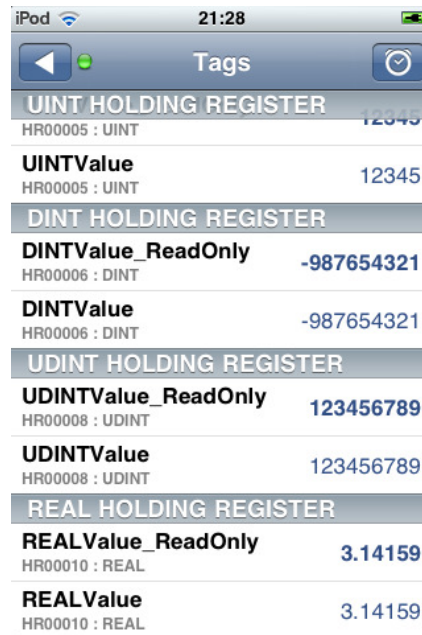
❖ DataTypesModbus.csv	91
❖ DataTypesModbusWSource.csv	92
❖ EIP_TAG_Examples.csv	93
❖ EIP_PCCC_Examples.csv	94
❖ PagesExampleModbus.csv	95
❖ PagesExampleEIP.csv	96
❖ StylesExampleModb.csv	97
❖ StylesExampleEIP_PCCC.csv	98
❖ AlarmsModbus.csv	99
❖ AlarmsEIP_PCCC.csv	100
❖ ColorfulControlsModbus.csv	101
❖ ColorfulControlsEIP_PCCC.csv	102
❖ Formula-ONE.csv	103

The ProSoft i-View application comes with pre-installed examples that can be used directly or downloaded to a PC to be used as templates for your own development. The following examples are provided.

EXAMPLE	CREATED WITH	PROTOCOL	COMMUNICATION ATTRIBUTES	STYLES	ALARMS	EXPRESSIONS
DataTypesModbus.csv	MS Excel	Modbus TCP/IP	NO	YES	NO	NO
DataTypesModbusWSource.csv	MS Excel	Modbus TCP/IP	YES	YES	NO	NO
EIP_TAG_Examples.csv	MS Excel	EIP/Native	NO	NO	NO	NO
EIP_PCCC_Examples.csv	MS Excel	EIP/PCCC	NO	NO	NO	NO
PagesExampleModbus.csv	MS Excel	Modbus TCP/IP	NO	NO	NO	NO
PagesExampleEIP.csv	MS Excel	EIP/Native	NO	NO	NO	NO
StylesExampleModbus.csv	MS Excel	Modbus TCP/IP	NO	YES	NO	NO
StylesExampleEIP_PCCC.csv	MS Excel	EIP/PCCC	NO	YES	NO	NO
AlarmsModbus.csv	MS-Excel	Modbus TCP/IP	NO	YES	YES	NO
AlarmsEIP_PCCC.csv	MS Excel	EIP/PCCC	NO	YES	YES	NO
ColorfulControlsModbus.csv	MS-Excel	Modbus TCP/IP	NO	YES	NO	NO
ColorfulControlsEIP_PCCC.csv	MS Excel	EIP/PCCC	NO	YES	NO	NO
Formula-ONE.csv	MS Excel	none	NO	YES	YES	YES

10.1 DataTypesModbus.csv

This example shows the supported data types with a focus on Modbus devices. This file was created in MS Excel and does not require any program in the PLC.



The screenshot shows the 'Tags' screen of the ProSoft i-View mobile application. It displays four sections of Modbus data types, each with a 'ReadOnly' and a standard 'Value' field. The data is as follows:

Register Name	Value
UINT HOLDING REGISTER	
HR00005 : UINT	12345
UINTValue	12345
HR00005 : UINT	12345
DINT HOLDING REGISTER	
HR00006 : DINT	-987654321
DINTValue_ReadOnly	-987654321
HR00006 : DINT	-987654321
UDINT HOLDING REGISTER	
HR00008 : UDINT	123456789
UDINTValue_ReadOnly	123456789
HR00008 : UDINT	123456789
REAL HOLDING REGISTER	
HR00010 : REAL	3.14159
REALValue_ReadOnly	3.14159
HR00010 : REAL	3.14159

10.2 DataTypesModbusWSource.csv

Same as the previous example except that it contains communication attributes.

10.3 EIP_TAG_Examples.csv

The purpose of this example is to demonstrate a set of valid symbolic tag names used in Allen Bradley Logix controllers.

Tag Name	Tag Type	Value / State
bool_value	TAG : BOOL	OFF
sint_value	TAG : SINT	122
int_value	TAG : INT	13
dint_value	TAG : DINT	67
timer1.PRE	TAG : DINT	1000
timer1.ACC	TAG : DINT	223
timer1.EN	TAG : BOOL	ON
timer1.TT	TAG : BOOL	ON
timer1.DN	TAG : BOOL	OFF
indx_var[1,3,2]	TAG : INT	31
struct_var.member1	TAG : INT	32
struct_var.realMember	TAG : REAL	2.71828
struct_var.boolMember	TAG : BOOL	OFF
struct_var.indxMember[0]	TAG : INT	35
struct_var.indxMember[1]	TAG : INT	36
struct_var.indxMember[2]	TAG : INT	37
struct_var.indxMember[3]	TAG : INT	38
indx_struct_var[3].memb...	TAG : INT	39

10.4 EIP_PCCC_Examples.csv

The purpose of this example is to demonstrate a set of valid symbolic tag names used in Allen Bradley Micrologix controllers.

10.5 PagesExampleModbus.csv

This example shows how to arrange tags in several pages. It uses arbitrary Modbus Coils and Registers.

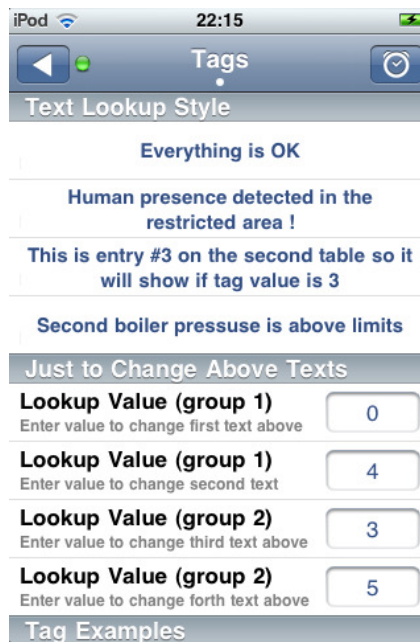


10.6 PagesExampleEIP.csv

Similar to the previous example except that it uses EIP/Native arbitrary tag names.

10.7 StylesExampleModb.csv

This example shows the available styles and some related tag attributes such as *Scale*, *Format*, *Bounds*, *Prefix*, *Suffix*, as well as the use of lookup tables.

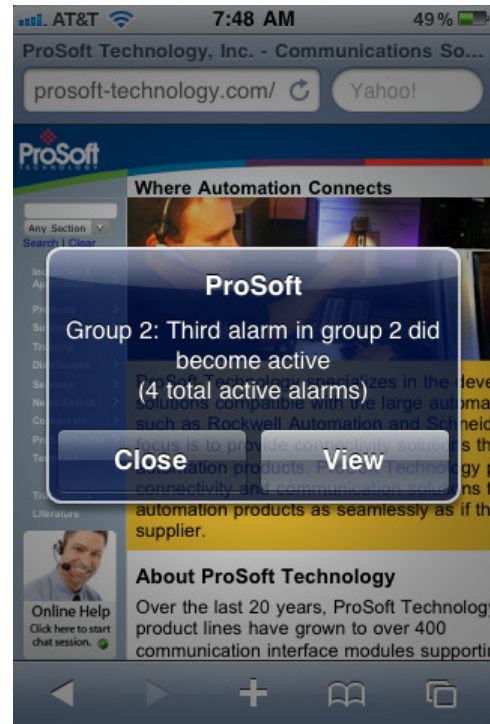
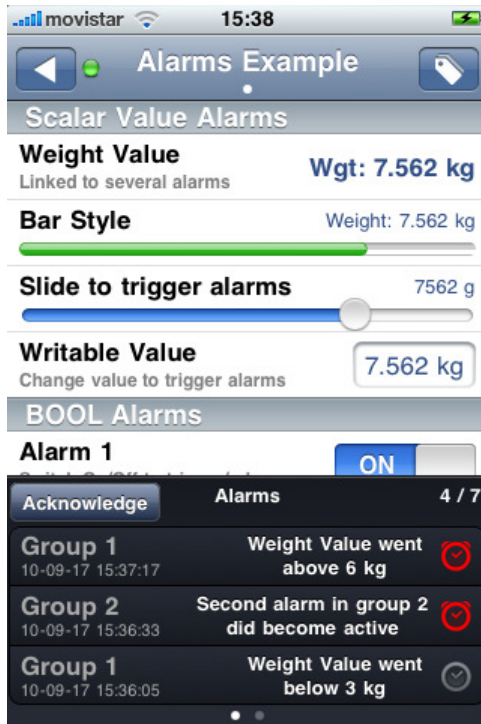


10.8 StylesExampleEIP_PCCC.csv

This example shows the available styles and some related tag attributes such as *Scale*, *Format*, *Bounds*, *Prefix*, *Suffix*, as well as the use of lookup tables using EIP/PCCC protocol. It will run on an AB Micrologix Controller.

10.9 AlarmsModbus.csv

Demonstrates alarm management on a Modbus configuration. The example includes both alarms based on tag value and discrete alarms. The screenshot on the left shows alarming within the application. The screenshot on the right shows an alarm notification from ProSoft i-View while it was running in the background, which happened while the user was browsing the internet.

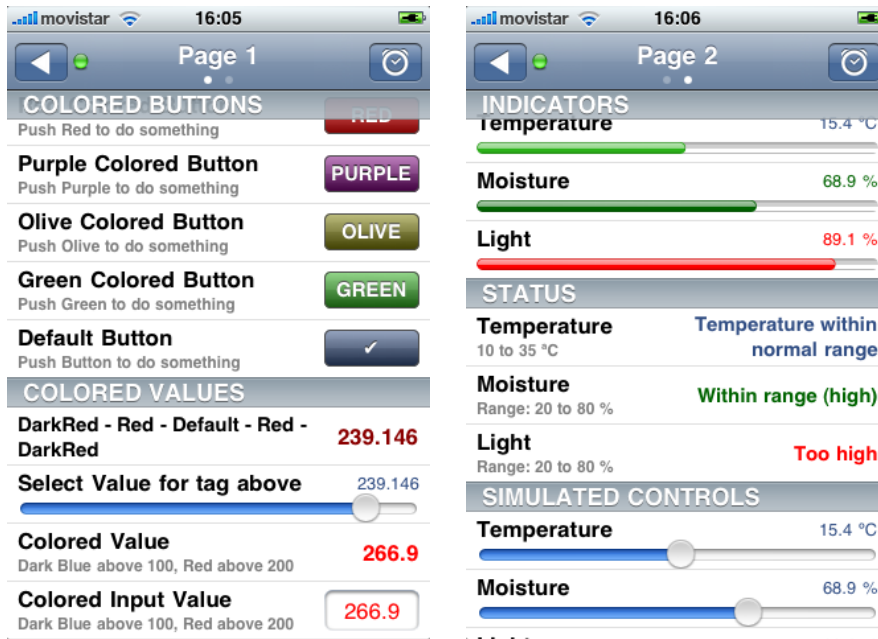


10.10 AlarmsEIP_PCCC.csv

Identical to the previous one but based on the EtherNet/IP PCCC protocol.

10.11 ColorfulControlsModbus.csv

Presents ways to specify colors in tags and controls based on value. The example is based on the Modbus TCP/IP protocol.



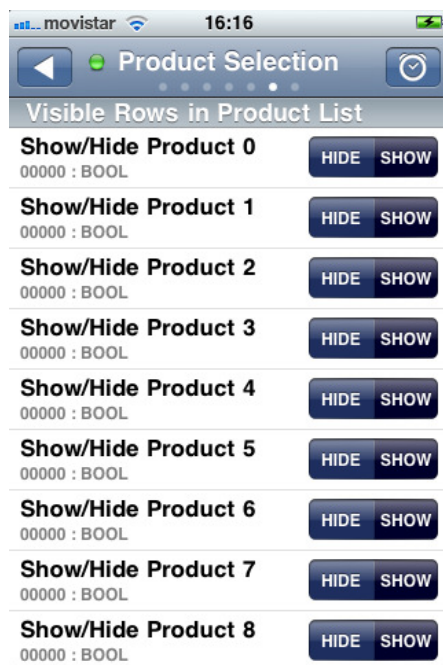
10.12 ColorfulControlsEIP_PCCC.csv

Identical to the previous one but based on the EtherNet/IP PCCC protocol.

10.13 Formula-ONE.csv

This example shows several aspects and possibilities of using expressions on attributes. The example is self contained by using only LOCAL tags and it does not need an actual PLC connection to run. It is structured in pages and fully commented to demonstrate:

- Switching and displaying/hiding of rows on the interface based on user selectable style.
- Use of the 'picker' attribute to allow users to retrieve a value based on a picker wheel control.
- Complete or partial removal of rows, sections and pages based on simple boolean states.
- Arithmetic and boolean calculations based on tag values
- Implementation of controls to enable/disable particular alarms
- Use of system variables in expressions to achieve special effects.



10.14 Document Revision History

Refer to this section to look at changes on this document over different versions.

10.14.1 Version 2.0.0.

- New styled table of contents.
- Added section: “Tag Scope”.
- Added memory arrays definition in section “Specification of Variable Types”.
- Mention of how to specify communication protocol in section “Specification of Variable Addresses”.
- Incorporated references to the new strings type in section “Specification of Variable Addresses”.
- Added reference for accessing program tags in Logix controllers in section “Specification of Variable Addresses”.
- Added section: “Internal Tags”.
- Replaced all previous references to Local tags by “Internal tags”.
- Replaced all previous references to arrays by “value lists” and text strings by “text”
- Added section: “PLC Memory Arrays”.
- New ‘bool’, ‘barcode’ and ‘validation_tag’ attributes or styles added to the “Tag Attributes” section.
- Changed placement of sections: “Comments in Data Sources” and “Specification of Communication Protocol”.
- Extended section “Expressions” to cover Strings and Arrays.
- Extended and renamed section: “Supported Operators and Operator Precedence”.
- Added section “Data types in Expressions”, extending and replacing some subheadings in the old “Expressions” section.
- Updated section “System Variables”.
- Added section “Functions Methods and more about Operators”.
- Added section “Putting it all together. Advanced Expressions Examples”.
- Added chapter “Performance”.
- Added “Document Revision History”.

Index

A

Accessing a Register as a BOOL • 21
Accessing Data Types Longer Than One Register • 21
Accessing Individual Bits in a Register • 22
Alarms • 26, 39
AlarmsEIP_PCCC.csv • 96
AlarmsModbus.csv • 95
Attribute Scope and Type • 25, 40
Attributes (Column D) • 24, 56, 57, 67, 73
Attributes by Scope • 25
Attributes by Type • 25

B

Background Process • 81
Background Task Processing • 80
Bottom Panel • 13
Building a Project in RSLogix 5000 • 56

C

ColorfulControlsEIP_PCCC.csv • 98
ColorfulControlsModbus.csv • 97
Connections • 7, 10, 65
Custom Company Logo • 63

D

Data Source Files • 4, 7, 15
Data Sources Created in Excel • 16
DataTypesModbus.csv • 87
DataTypesModbusWSource.csv • 88

E

Editing Source Files in a Text Editor • 57
EIP_PCCC_Examples.csv • 90
EIP_TAG_Examples.csv • 89
Event Driven Architecture • 42
Example • 40
Exporting Controller Tags from RSLogix 5000 • 56
Expressions • 42

F

File Import into ProSoft i-View • 4, 59
File Server • 10
Files • 10
Formula-ONE.csv • 99

G

General Aspects • 7
Global Attributes • 33

H

Home • 10
Home Tab Bar and Navigation Bar • 13

How It Works • 36
How to Contact Us • 2
How to Use ProSoft i-View in Five Simple Steps • 4

K

Keep Connected • 80

L

Lookup Tables • 19, 37

M

Main Features • 4
Managing Accounts • 69
Multiple Range Lookup Tables • 38
Multiple User Lookup Tables • 38

N

Network Settings for Local Access • 4, 71
Network Settings for Remote Access • 75
Note on EIP/Native Communication Protocol • 22

O

Operators and Operator Precedence • 47
Other Files Supported by ProSoft i-View • 62

P

Page, Section and Ord Attributes • 36
Pages, Sections, Rows and Data Sources • 27, 36
PagesExampleEIP.csv • 92
PagesExampleModbus.csv • 91
Performance Considerations • 40
PLC Settings for Local Access • 72
Pre-installed Examples • 85
ProSoft i-View Settings for Local PLC Access • 73
ProSoft Technology® Product Documentation • 2

Q

Quick Start • 3

R

Restrictions for Non-Administrator Users • 68
Rockwell RSLogix 5000 as a Data Source Generator • 56

S

Security • 77
Settings • 10
Source Files Supported by ProSoft i-View • 61
StylesExampleEIP_PCCC.csv • 94
StylesExampleModb.csv • 93
Supported Protocols • 8
System Variables • 47

T

Tabbed Interface • 10
Tag Attributes • 26

U

User Accounts • 24, 59, 67
User Interface Elements • 9
Using Expressions • 43

V

Validation Codes • 17, 78
Variable Addresses (Column C) • 20
Variable Names (Column A) • 17, 21
Variable Types (Column B) • 17, 18
Variables in ProSoft i-View • 11

W

What is ProSoft i-View? • 3

Y

Your Feedback Please • 2