

CitectSCADA

v7.20

Cicode Reference Guide

October 2010

Legal Notice

DISCLAIMER

Schneider Electric (Australia) Pty. Ltd. makes no representations or warranties with respect to this manual and, to the maximum extent permitted by law, expressly limits its liability for breach of any warranty that may be implied to the replacement of this manual with another. Further, Schneider Electric (Australia) Pty. Ltd. reserves the right to revise this publication at any time without incurring an obligation to notify any person of the revision.

COPYRIGHT

© Copyright 2010 Schneider Electric (Australia) Pty. Ltd. All rights reserved.

TRADEMARKS

Schneider Electric (Australia) Pty. Ltd. has made every effort to supply trademark information about company names, products and services mentioned in this manual.

Citect, CitectHMI, and CitectSCADA are registered trademarks of Schneider Electric (Australia) Pty. Ltd.

IBM, IBM PC and IBM PC AT are registered trademarks of International Business Machines Corporation.

MS-DOS, Windows, Windows NT, Microsoft, and Excel are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

DigiBoard, PC/Xi and Com/Xi are trademarks of Digi International Inc.

Novell, Netware and Netware Lite are either registered trademarks or trademarks of Novell, Inc. in the United States and other countries..

dBASE is a trademark of dataBased Intelligence, Inc.

All other brands and products referenced in this document are acknowledged to be the trademarks or registered trademarks of their respective holders.

GENERAL NOTICE

Some product names used in this manual are used for identification purposes only and may be trademarks of their respective companies.

October 2010 edition for CitectSCADA Version v7.20

Manual Revision Version v7.20.

Contact Schneider Electric (Australia) Pty. Ltd. today at www.Citect.com/citecscada

Contents

Legal Notice	2
Contents	3
Introduction	11
Safety Information	13
Chapter: 1 Introducing Cicode	15
Getting Started.....	15
Using Cicode Files.....	16
Using Cicode	17
Chapter: 2 Using Cicode Commands	19
Setting Variables.....	19
Performing Calculations.....	20
Using Multiple Command Statements.....	21
Using Include (Text) Files.....	21
Getting Runtime Operator Input.....	23
Chapter: 3 Using Cicode Expressions	25
Displaying Data Using Expressions.....	25
Decision-Making.....	26
Logging Expression Data.....	26
Triggering Events Using Expressions.....	27
Chapter: 4 Using Cicode Functions	29

Contents

Calling Functions from Commands and Expressions.....	29
Triggering Functions via Runtime Operator Input.....	29
Evaluating Functions.....	30
Combining Functions with Other Statements.....	30
Passing Data to Functions (Arguments).....	31
Using String Arguments.....	31
String assignment.....	32
Using the Caret Escape Sequence Character.....	32
Using Multiple Arguments.....	33
Using Numeric Arguments.....	33
Using Variable Arguments.....	33
Using Operator Input in Functions.....	34
Returning Data from Functions.....	34
Chapter: 5 Working with Commonly Used Functions.....	35
Alarm Functions.....	35
Page Functions.....	36
Keyboard Functions.....	37
Report Functions.....	37
Time/date Functions.....	37
Miscellaneous Functions.....	37
Chapter: 6 Writing Functions.....	39
Cicode Function Structure.....	39
Function Uses.....	40
Writing Groups of Functions.....	41
Cicode Function Libraries.....	41
Creating a Function Outline.....	42
Pseudocode.....	42
Using Comments in Cicode.....	43
Using Comments for Debugging Functions.....	44
Tag Reference /TagReadEx() behavior in Cicode Expressions.....	44
Following Cicode Syntax.....	45
Cicode Function Syntax.....	46
End of line markers.....	48
Function Scope.....	48
Declaring the Return Data Type.....	49
Declaring Functions.....	50
Naming Functions.....	51
Function Argument Structure.....	52
Declaring Argument Data Type.....	54
Naming Arguments.....	55
Setting Default Values for Arguments.....	56
Returning Values from Functions.....	58
Chapter: 7 Using Variables.....	61
Declaring Variable Properties.....	61

Declaring the Variable Data Type.....	62
QUALITY Data Type.....	62
TIMESTAMP Data Type.....	63
Naming Variables.....	63
Setting Default Variable Values.....	64
Using Variable Scope.....	64
Using Database Variables.....	66
Chapter: 8 Using Arrays.....	67
Declaring Array Properties.....	67
Declaring the Array Data Type.....	68
Naming Arrays.....	68
Declaring the Variable Array Size.....	68
Setting Default (Initial) Array Values.....	69
Passing Array Elements as Function Arguments.....	70
Using One-dimensional Arrays.....	70
Using Two-dimensional Arrays.....	70
Using Three-dimensional Arrays.....	71
Using Array Elements in Loops.....	72
Using the Table (Array) Functions.....	72
Chapter: 9 Using Cicode Macros.....	73
IFDEF.....	73
IFDEFAdvAlm.....	74
IFDEFAnaAlm.....	75
IFDEFDigAlm.....	77
Macro Arguments.....	78
Chapter: 10 Converting and Formatting Cicode Variables.....	79
Converting Variable Integers to Strings.....	79
Converting Real Numbers to Strings.....	80
Converting Strings to Integers.....	81
Converting Strings to Real Numbers.....	81
Formatting Text Strings.....	81
Escape Sequences (String Formatting Commands).....	83
Chapter: 11 Working with Operators.....	85
Using Mathematical Operators.....	85
Using Bit Operators.....	87
Using Relational Operators.....	88
Using Logical Operators.....	89
Order of Precedence of Operators.....	90
Chapter: 12 Working with Conditional Executors.....	91
Setting IF ... THEN Conditions.....	91

Contents

Using FOR ... DO Loops	92
Using WHILE ... DO Conditional Loops	93
Nested Loops	93
Using the SELECT CASE statement	94
Chapter: 13 Performing Advanced Tasks.....	97
Handling Events	97
How Cicode is Executed	98
Multitasking	99
Foreground and background tasks	99
Controlling tasks	100
Pre-emptive multitasking	100
Chapter: 14 Editing and Debugging Code.....	103
The Cicode Editor	103
Starting the Cicode Editor	104
Changing the default Cicode Editor	105
Creating Cicode files	106
Creating functions	106
Saving files	106
Opening Cicode files	107
Deleting Cicode files	108
Finding text in Cicode files	108
Compiling Cicode files	108
Viewing errors detected by the Cicode Compiler	109
Cicode Editor Options	109
Docking the Windows and Toolbars	109
Displaying the Editor Options Properties dialog	110
Windows and Bars Tab	111
Toolbar options	111
Window options	111
Viewing Editor windows	112
Options Properties Tab	117
Language Formatter Properties Tab	119
Debugging Cicode	120
Using debug mode	120
Debugging functions	120
Debugging functions remotely	121
Using breakpoints	122
Inserting or removing breakpoints	122
Enabling/disabling breakpoints	122
Stepping through code	123
Chapter: 15 Using Cicode Programming Standards.....	125
Variable Declaration Standards	126
Variable Scope Standards	126
Variable Naming Standards	127

Standards for Constants, Variable Tags, and Labels.....	128
Formatting Simple Declarations.....	129
Formatting Executable Statements.....	130
Formatting Expressions.....	131
Cicode Comments.....	132
Formatting Functions.....	132
Format Templates.....	134
Function Naming Standards.....	136
Modular Programming.....	138
Defensive Programming.....	141
Function Error handling.....	142
Debug Error Trapping.....	145
Functions Reference.....	147
Chapter: 16 Accumulator Functions.....	149
Accumulator Functions.....	149
Chapter: 17 ActiveX Functions.....	157
ActiveX Functions.....	157
Chapter: 18 Alarm Functions.....	171
Alarm Functions.....	171
Chapter: 19 Clipboard Functions.....	281
Clipboard Functions.....	281
Chapter: 20 Cluster Functions.....	285
Cluster Functions.....	285
Chapter: 21 Color Functions.....	293
Color Functions.....	293
Chapter: 22 Communication Functions.....	299
Communication Functions.....	299
Chapter: 23 Dynamic Data Exchange Functions.....	307
DDE Functions.....	307
Chapter: 24 Device Functions.....	323

Contents

Device Functions.....	323
Chapter: 25 Display Functions.....	353
Display Functions.....	353
Chapter: 26 DLL Functions.....	447
DLL Functions.....	447
Equipment Database Functions.....	452
Chapter: 27 Error Functions.....	461
Error Functions.....	461
Chapter: 28 Event Functions.....	475
Event Functions.....	475
Chapter: 29 File Functions.....	493
File Functions.....	493
Chapter: 30 Form Functions.....	513
Form Functions.....	513
Chapter: 31 Format Functions.....	553
Format Functions.....	553
Chapter: 32 FTP Functions.....	563
FTP Functions.....	563
Chapter: 33 FuzzyTech Functions.....	569
FuzzyTech Functions.....	569
Chapter: 34 Group Functions.....	577
Group Functions.....	577
Chapter: 35 I/O Device Functions.....	587
I/O Device Functions.....	587
Chapter: 36 Keyboard Functions.....	599
Keyboard Functions.....	599
Chapter: 37 Mail Functions.....	615

Mail Functions.....	615
Chapter: 38 Math and Trigonometry Functions.....	621
Math/Trigonometry Functions.....	621
Chapter: 39 Menu Functions.....	639
Menu Functions.....	640
Chapter: 40 Miscellaneous Functions.....	657
Miscellaneous Functions.....	657
Chapter: 41 Page Functions.....	711
Page Functions.....	711
Chapter: 42 Plot Functions.....	751
Plot Functions.....	751
Chapter: 43 Process Analyst Functions.....	773
Process Analyst Functions.....	773
Chapter: 44 Quality Functions.....	781
Quality Functions.....	781
Chapter: 45 Report Functions.....	795
Report Functions.....	795
Chapter: 46 Security Functions.....	801
Security Functions.....	801
Chapter: 47 Server Functions.....	827
Server Functions.....	827
Chapter: 48 Statistical Process Control Functions.....	843
SPC Functions.....	843
Chapter: 49 SQL Functions.....	859
SQL Functions.....	859
Chapter: 50 String Functions.....	885

Contents

String Functions.....	885
Chapter: 51 Super Genie Functions.....	915
Super Genie Functions.....	915
Chapter: 52 Table (Array) Functions.....	951
Table (Array) Functions.....	951
Chapter: 53 Tag Functions.....	957
Tag Functions.....	957
Chapter: 54 Task Functions.....	999
Task Functions.....	999
Chapter: 55 Time and Date Functions.....	1041
Time/Date Functions.....	1041
Chapter: 56 Timestamp Functions.....	1063
Timestamp Functions.....	1063
Chapter: 57 Trend Functions.....	1079
Trend Functions.....	1079
Chapter: 58 Window Functions.....	1177
Window Functions.....	1177
Technical Reference.....	1215
Chapter: 59 Cicode Errors.....	1217
Hardware/Cicode Errors.....	1217
Cicode and General Errors.....	1218
MAPI Errors.....	1238
Chapter: 60 Browse Function Field Reference.....	1241
Server Browse Function Fields.....	1253
Index.....	1255

Part: 1

Introduction

This section provides some introductory material for CitectSCADA:

[Introducing Cicode](#)

Safety Information

Hazard categories and special symbols

The following symbols and special messages may appear in this manual or on the product to warn of potential hazards or to call attention to information that clarifies or simplifies a procedure.

A lightning bolt or ANSI man symbol in a "Danger" or "Warning" safety label on the product indicates an electrical hazard which, as indicated below, can or will result in personal injury if the instructions are not followed.

The exclamation point symbol in a safety message in a manual indicates potential personal injury hazards. Obey all safety messages introduced by this symbol to avoid possible injury or death.

Symbol	Name
	Lightning Bolt
	ANSI man
	Exclamation Point

DANGER

DANGER indicates an imminently hazardous situation, which, if not avoided, will result in death or serious injury.

WARNING

WARNING indicates a potentially hazardous situation, which, if not avoided, can result in death or serious injury.

CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, can result in minor or moderate injury.

CAUTION

CAUTION used without the safety alert symbol, indicates a potentially hazardous situation which, if not avoided, can result in property damage.

Please Note

Electrical equipment should be installed, operated, serviced, and maintained only by qualified personnel. No responsibility is assumed by Schneider Electric (Australia) Pty. Ltd. for any consequences arising out of the use of this material.

Before You Begin

CitectSCADA is a Supervisory Control and Data Acquisition (SCADA) solution. It facilitates the creation of software to manage and monitor industrial systems and processes. Due to CitectSCADA's central role in controlling systems and processes, you must appropriately design, commission, and test your CitectSCADA project before implementing it in an operational setting. Observe the following:

WARNING

UNINTENDED EQUIPMENT OPERATION

Do not use CitectSCADA or other SCADA software as a replacement for PLC-based control programs. SCADA software is not designed for direct, high-speed system control.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

WARNING

LOSS OF CONTROL

- The designer of any control scheme must consider the potential failure modes of control paths and, for certain critical control functions, provide a means to achieve a safe state during and after a path failure. Examples of critical control functions are emergency stop and overtravel stop.
- Separate or redundant control paths must be provided for critical control functions.
- System control paths may include communication links. Consideration must be given to the implications of unanticipated transmission delays or failures of the link.*
- Each implementation of a control system created using CitectSCADA must be individually and thoroughly tested for proper operation before being placed into service.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

* For additional information, refer to NEMA ICS 1.1 (latest edition), "Safety Guidelines for the Application, Installation, and Maintenance of Solid State Control".

Chapter: 1 Introducing Cicode

Cicode is a programming language designed for use in CitectSCADA to monitor and control plant equipment. It is a structured language similar to Visual Basic or 'C'. You need no previous programming experience to use it.

Using Cicode, you can access real-time data (variables) in the CitectSCADA project, and CitectSCADA facilities: variable tags, alarms, trends, reports, and so on. You can use Cicode to interface to various facilities on the computer, such as the operating system and communication ports. Cicode supports advanced features including pre-empted multitasking, multi threads, and remote procedure calls.

Getting Started

Use the following sections as a quick start to using Cicode in your CitectSCADA projects:

- Cicode can be stored in procedures called functions for multiple reuse and centralized maintenance. For details, see [Using Cicode Files](#).
- Cicode can be typed directly into command fields in online CitectSCADA forms. For details, see [Using Cicode Commands](#).
- Cicode expressions are used to display and log data for monitoring and analysis, and to trigger various elements in your system, such as alarms, events, reports, and data logging. For information on using expressions, see [Using Cicode Expressions](#).
- A Cicode function is a small program, a collection of statements, variables, operators, conditional executors, and other functions. A Cicode function can perform complex tasks and give you access to CitectSCADA graphics pages, alarms, trend data, and so on. For information on using functions, see the section titled [Using Cicode Functions](#). Cicode has many pre-defined functions that perform a variety of tasks. For details on commonly used functions, see the section titled [Working with Commonly Used Functions](#). Where system functionality cannot be achieved with built-in functions, you can write your own functions. See [Writing Functions](#).
- The Cicode Editor is the code editing tool provided with CitectSCADA for the writing, editing and debugging of your Cicode code. For details, see [The Cicode Editor](#).

See Also

[Performing Advanced Tasks](#)

[Using Cicode Programming Standards](#)

Using Cicode Files

You write your Cicode functions in Cicode source files, stored on your hard disk. Cicode files are identified by having a *.CI extension.

To minimize potential future difficulties with maintaining your Cicode files, adopt a programming standard as early as possible (see [Using Cicode Programming Standards](#)).

Maintain structured Cicode files, by logically grouping your Cicode functions within the files, and by choosing helpful descriptive names. For details about modular programming methods, see [Modular Programming](#). For details about using and debugging Cicode functions, see [Formatting Functions](#) and [Debugging Cicode](#) respectively.

When you compile your CitectSCADA project, the compiler reads the functions in your Cicode source files. Your system can then use these functions in the same way as it uses built-in functions. You can use as many Cicode files as required. Cicode files reside in the same directory as your CitectSCADA project. When you back up your project, the Cicode source files in the project directory are also backed up.

See Also

[The Cicode Editor](#)

[Creating Cicode files](#)

[Opening Cicode files](#)

Part: 2

Using Cicode

This section contains information for Users and describes the following:

<u>Using Cicode Commands</u>	<u>Using Cicode Macros</u>
<u>Using Cicode Expressions</u>	<u>Converting and Formatting Cicode Variables</u>
<u>Using Cicode Functions</u>	<u>Working with Operators</u>
<u>Working with Commonly Used Functions</u>	<u>Working with Conditional Executors</u>
<u>Writing Functions</u>	<u>Performing Advanced Tasks</u>
<u>Using Variables</u>	<u>Editing and Debugging Code</u>
<u>Using Arrays</u>	<u>Using Cicode Programming Standards</u>

Chapter: 2 Using Cicode Commands

Cicode commands extend the control element of a CitectSCADA control and monitoring system. You use commands to control your CitectSCADA system and therefore the processes in your plant.

Each command has a mechanism to activate it. Commands can be issued manually, through an operator typing a key sequence, or by clicking on a button (or object) on a graphics page. You can also configure commands to execute automatically:

- When an operator logs into or out of the runtime system
- When a graphics page is displayed or closed
- When an alarm is triggered
- In a report
- When an event is triggered

To define a Cicode command, you enter a statement (or group of statements) in the command field (Input category) for an object.

Each statement in a command usually performs a single task, such as setting a variable to a value, calculating a value, displaying a message on the screen, or running a report. For information on using variables, see the section titled [Using Variables](#).

If you want to evaluate a condition, like checking the state of your plant rather than perform an action or command upon your plant, use an expression instead. See the section titled [Using Cicode Expressions](#).

See Also

[Using Cicode Programming Standards](#)

Setting Variables

You can set a Variable in CitectSCADA within a Command field, an Expression field, or in a Cicode Function, by using the mathematical 'equals' sign (=) assignment operator. The value on the right is assigned (set) to the variable on the left, as shown in the following Cicode example :

```
<VAR_TAG> = Val;
```

where:

<VAR_TAG> is the name of the variable, and val is the value being assigned to the variable.

Examples

To set a digital variable (named BIT_1) to ON (1), use the command:

```
BIT_1 = 1;
```

To set a digital variable (named BIT_1) to OFF (0), use the command:

```
BIT_1 = 0;
```

To set a digital variable (named B1_PUMP_101_M) to ON (1), use the command:

```
B1_PUMP_101_M = 1;
```

To set a digital variable (named B1_PUMP_101_M) to OFF (0), use the command:

```
B1_PUMP_101_M = 0;
```

To set an analog variable (named B1_TIC_101_SP) to a value of ten (10), use the command:

```
B1_TIC_101_SP = 10;
```

You can copy a variable to another by assigning (setting) the value of a variable to the value of another variable, for example:

```
B1_PUMP_101_COUNT = B1_PUMP_101_CLIMIT;
```

The value of B1_PUMP_101_COUNT is set to the value of B1_PUMP_101_CLIMIT only when that command is issued.

Note: The value of B1_PUMP_101_CLIMIT could change immediately after, but B1_PUMP_101_COUNT remains unchanged and storing the original value, until this command is issued again.

Performing Calculations

Mathematical calculations can be performed between variables in a Cicode statement. For example:

```
B1_TIC_101_SP = B1_TIC_101_PV + B1_TIC_102_PV - 100;
```

When this command is executed, the variable B1_TIC_101_SP is set to a value that is the sum of variables B1_TIC_101_PV and B1_TIC_102_PV minus 100.

Using Multiple Command Statements

A single statement in a Cicode command usually performs a single task. When the CitectSCADA runtime system is in operation, the statement executes whenever the command is requested. For example, if the statement is linked to a keyboard command, the task is performed when an operator presses the keyboard key defined as that command.

To perform several tasks at the same time, you combine statements in a command property:

```
B1_PUMP_101_COUNT = B1_PUMP_101_CLIMIT;
BATCH_NAME = "Bread";
B1_TIC_101_SP = 10;
```

The example above uses three statements, separated by semi-colons (;). The first statement sets the variable B1_PUMP_101_COUNT to the value of the variable B1_PUMP_101_CLIMIT; the second statement sets the variable BATCH_NAME to the string "Bread"; and the third statement sets the variable B1_TIC_101_SP to 10. Each statement is executed in order.

Note: Separate each statement in a command with a semicolon (;). If you don't, CitectSCADA will not recognize the end of a statement, and errors will result when the project is compiled.

The number of statements you can enter in a command property is limited only by the size of the field. However, for clarity, don't use too many statements; enter the statements into an Include File or write a Cicode Function. You then refer to the include file or call the function in the command property field.

Using Include (Text) Files

There is a maximum number of characters that you can type in a Command or Expression field (usually 128). If you need to include many commands (or expressions) in a property field, you can define a separate include file that contains the commands or expressions.

An include file is a separate and individual ASCII text file containing only one sequence of CitectSCADA commands or expressions that would otherwise be too long or complicated to type into the Command or Expression field within CitectSCADA. The include file name is entered instead, and the whole file is activated when called.

When you compile the project, the commands (or expressions) in the include file are substituted for the property field, just as if you had typed them directly into the field.

Use a text editor such as Notepad to create the text file.

Enter the name of the include file (either upper- or lower case) in the property, in the following format:

```
@<filename>
```

where *<filename>* is any valid DOS file name. Be aware that the bracket characters (< >) are part of the syntax.

You can use include files with many properties (except record names), but they are commonly used for commands and expressions, for example:

- Key sequence: F5 ENTER
- Command: @<setvars.cii>

In the above example, the *setvars.cii* include file would contain commands to be substituted for the Command property when you compile your project, for example:

```
PV12 = 10;
PV22 = 20;
PV13 = 15;
PV23 = 59;
PageDisplay("Mimic");
```

Notes

- Include files can not be used for genie properties.
- Do not confuse *include files* and *included projects*. Include files contain CitectSCADA commands and/or expressions and are used as substitutions in a CitectSCADA command or expression property field. Included projects are separate (usually smaller) CitectSCADA projects that can be included in another CitectSCADA project so that they appear together as one project.
- The include file name can contain a maximum of 64 characters, or 253 characters including a path, and can consist of any characters other than the semi-colon (;) or the single quote('). There is no need to include the .cii extension, but if the file is not in the project directory, you need to enter the full path to the file. If the file is not in the project directory, it will not be backed up with the Backup facility.

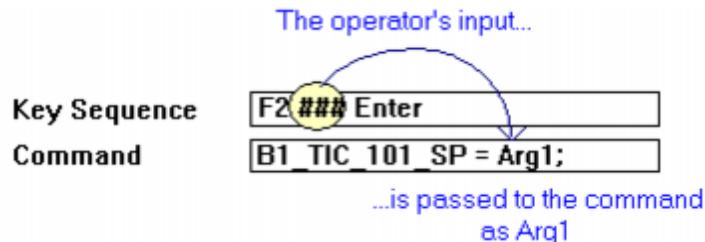
- If modifying an Include file with the Cicode Editor, when you save your changes a .ci file extension will be appended to the file name. Change this to a .cii file extension in Windows Explorer.

Getting Runtime Operator Input

You can define a keyboard command as a key sequence, to perform a specific task each time the key sequence is pressed, for example:

- Key sequence: F2 ENTER
- Command: B1_TIC_101_SP = 10;

A key sequence can include provision for the operator to enter data. In the following example, the operator can set the value of the variable B1_TIC_101_SP:



The operator sends out the command by pressing the F2 key, up to three characters, and the Enter key. The three character sequence (identified by the three hash (#) characters) is called an *argument*. The argument is passed into the command (as Arg1) when the command is completed (when the operator presses the Enter key).

The operator might type:



The value 123 is passed to the command, and B1_TIC_101_SP is set to 123.

It is recommended that you use a specific key (for example, Enter) to signal the end of a key sequence. If, for example, you use the key sequence F2 ####, the operator **needs to** enter 4 characters for the command to be executed - CitectSCADA waits for the fourth character. But if you use F2 #### Enter, the operator can enter between one and four characters as necessary. The command executes as soon as the Enter key is pressed.

To use more than one argument in a command, separate the arguments with commas (,):

- Key sequence: F2 ###### Enter
- Command: B1_TIC_101_SP = Arg1; B1_TIC_101_PV = Arg2;

To set both variables, the operator can type:



The values 123 and 18 are passed to the command. B1_TIC_101_SP is set to 123 and B1_TIC_101_PV is set to 18.

Chapter: 3 Using Cicode Expressions

Cicode expressions are the basic elements of the Cicode language. An expression can be a constant, the value of a variable tag, or the result of a complex equation. You can use expressions to display and log data for monitoring and analysis, and to trigger various elements in your system, such as alarms, events, reports, and data logging.

You can enter a Cicode expression in any CitectSCADA editor form or graphic object that contains an expression property. Unlike a command, an expression does not execute a specific task - it is evaluated. The evaluation process returns a value that you can use to display information on the screen (for example, as a bar graph) or to make decisions.

The following expression returns a result of 12:

- Numeric expression: $8 + 4$

In the above example, the value of the expression is a constant (12) because the elements of the expression are constants (8 and 4).

See Also

[Displaying Data Using Expressions](#)

[Logging Expression Data](#)

[Triggering Events Using Expressions](#)

[Using Cicode Programming Standards](#)

[Using Cicode Files](#)

Displaying Data Using Expressions

In the following example, the value of the expression is the value of the variable B1_TIC_101_PV. As its value changes, the value of the expression also changes. You can use this expression to display a number on a graphics page.

- Numeric expression: B1_TIC_101_PV

As the expression changes, the number also changes.

Expressions can also include mathematical calculations. For example, you can add two variables together and display the combined total:

- Numeric expression: B1_TIC_101_PV + B1_TIC_102_PV

In this case, the value of the expression is the combined total. As the value of one variable (or both variables) changes, the value of the expression changes.

See Also

[Using Cicode Expressions](#)

Decision-Making

Some expressions return only one of two logical values, either TRUE(1) or FALSE(0). You can use these expressions to make decisions, and to perform one of two actions, depending on whether the return value is TRUE or FALSE. For example, you can configure a text object with appearance as follows:

- On text when: B1_PUMP_102_CMD
- ON text: Pump Running
- OFF text: "Pump Stopped"

In this example, if B1_PUMP_102_CMD is a digital tag (variable), it can only exist in one of two states (0 or 1). When your system is running and the value of B1_PUMP_102_CMD changes to 1, the expression returns TRUE and the message "Pump Running" is displayed. When the value changes to 0, the expression returns FALSE and the message "Pump Stopped" is displayed.

See Also

[Using Cicode Expressions](#)

Logging Expression Data

You can log the value of an expression to a file for trending, by defining it as a trend tag:

Trend Tag Name	B1_TIC
Expression	B1_TIC_101_PV + B1_TIC_102_PV
File Name	[log]:B1_TIC

When the system is running, the value of the expression B1_TIC_101_PV + B1_TIC_102_PV is logged to the file [log]:B1_TIC.

See Also

[Using Cicode Expressions](#)

Triggering Events Using Expressions

Logical expressions - those that return either TRUE (1) or FALSE (0) -can be used as triggers.

For example, you might need to log the expression in the above example only when an associated pump is running.

Trend Tag Name	B1_TIC
Expression	B1_TIC_101_PV + B1_TIC_102_PV
File Name	[log]:B1_TIC
Trigger	B1_PUMP_101_CMD

In this example, the trigger is the expression B1_PUMP_101_CMD (a digital variable tag). If the pump is ON, the result of the trigger is TRUE, and the value of the expression (B1_TIC_101_PV + B1_TIC_102_PV) is logged. If the pump is OFF, the result is FALSE, and logging ceases.

See Also

[Using Cicode Expressions](#)

Chapter: 4 Using Cicode Functions

A Cicode function can perform more complex tasks than a simple command or expression allows. Functions give you access to CitectSCADA graphics pages, alarms, trend data, and so on.

CitectSCADA has several hundred built-in functions that display pages, acknowledge alarms, make calculations, and so on. You can also write your own functions to meet your specific needs.

See Also

[Working with Commonly Used Functions](#)

[Writing Functions](#)

Calling Functions from Commands and Expressions

You can call a function by entering its name in any command or expression property. The syntax is as follows:

Command	FunctionName (Arg1, Arg2, ...);
----------------	-----------------------------------

where:

FunctionName is the name of the function

Arg1, Arg2, ... are the arguments you pass to the function

Triggering Functions via Runtime Operator Input

In the following command, the `PageNext()` function displays the next graphics page when the Page Down keyboard key is pressed by the Runtime operator.

Key Sequence	Page_Down
---------------------	-----------

Command	PageNext();
----------------	-------------

Evaluating Functions

You can use a function in any expression. For example, the AlarmActive() function returns TRUE (1) if any alarms are active, and FALSE (0) if no alarms are active. In the following text object, either "Alarms Active" or "No Alarms Active" is displayed, depending on the return value of the expression.

ON text when	AlarmActive(0)
ON Text	"Alarms Active"
OFF Text	"No Alarms Active"

Note: Functions return a value that indicates the success of the function, or provides information on an error that has occurred. In many cases (for example, when used in a command) the return value can be ignored. You need to use the parentheses () in the function name, even if the function uses no arguments. Function names are not case-sensitive: `PageNext()`, `pagenext()` and `PAGENEXT()` call the same function.

Combining Functions with Other Statements

In expressions and commands you can use functions alone or in combination with other functions, operators, and so on.

The following example uses three statements:

Command	<code>Report("Shift"); B1_TIC_101_PV = 10; PageDisplay("Boiler 1")</code>
----------------	---

Each statement is executed in order. The "Shift" report is started first, the variable B1_TIC_101_PV is set to 10 next, and finally, the "Boiler 1" page is displayed.

Functions combine with operators and conditional executors to give you specific control over your processes, for example, you can test for abnormal operating conditions and act on them.

Passing Data to Functions (Arguments)

The parentheses () in the function name identify the statement as a function and enclose its arguments. Arguments are the values or variables that are passed into the function when it executes.

Note: Some functions (such as PageNext()) have no arguments. However you need to include the parentheses () or CitectSCADA will not recognize that it is a function, and an error could result when the project is compiled.

Using String Arguments

Functions can require several arguments or, as in the following example, a single argument:

Command	PageDisplay("Boiler 1");
---------	--------------------------

This function displays the graphics page called "Boiler 1". Be aware that when you pass a string to a function, you need to always enclose the string in double quotes.

You can use the `PageDisplay()` function to display any graphics page in your system - in each case, only the argument changes. For example, the following command displays the graphics page "Boiler 2":

Command	PageDisplay("Boiler 2");
---------	--------------------------

You can use the `Report()` function to run a report (for example, the "Shift" report) when the command executes:

Command	Report("Shift");
---------	------------------

The following example uses the `Prompt()` function to display the message "Press F1 for Help" on the screen when the command executes:

Command	Prompt("Press F1 for Help");
---------	------------------------------

String assignment

You can also assign string variables in commands. For example, if BATCH_NAME is a variable tag defined as a string data type, you can use the following command to set the tag to the value "Bread":

```
BATCH_NAME = "Bread";
```

Note: you need to enclose a string in double quotation marks (").

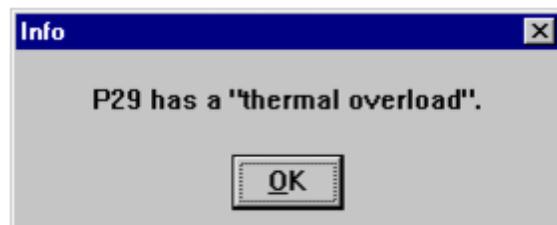
Using the Caret Escape Sequence Character

The caret character (^) signifies a special instruction in Cicode, called an escape sequence, primarily used in the formatting of text strings. Escape sequences include formatting instructions such as new line, form feed, carriage return, backspace, horizontal and vertical tab-spaces, single and double quote characters, the caret character, and hexadecimal numbers.

Strings are commonly represented in Cicode between double quote characters (") known as delimiters. If you want the string to contain a double quote character itself as part of the string, you need to precede the double quote character with the caret character (^") so that Cicode doesn't interpret the double quote in the string as the delimiter indicating the end of the string. The caret character is interpreted as a special instruction, and together with the characters immediately following it, are treated as an escape sequence instruction. See the section titled [Formatting Text Strings](#) for the list of escape sequences used in Cicode.

In the following Cicode example, both of these message functions will display the following message.

```
Message("Info", "P29 has a ^"thermal overload^".", 0);
sCurrentAlmText = "Thermal Overload";
Message("Info", "P29 has a ^""+sCurrentAlmText+"^.", 0);
```



Using Multiple Arguments

Some functions require several arguments. You need to list arguments between the parentheses, and separate each argument with a comma (,) as in the following example:

Command	Login("Manager", "ABC");
----------------	--------------------------

The order of the arguments affects the operation of any function. The `Login()` function logs a user into your runtime system. The first argument ("Manager") indicates the name of the user, and the second argument ("ABC") is the user's password. If you reverse the order of the arguments, the function would attempt to login a user called "ABC" - if a user by this name does not exist, an error message displays.

Using Numeric Arguments

You can pass **numbers** (integers and floating point numbers) directly to a function, for example:

Command	AlarmAck(2, 35);
----------------	------------------

Using Variable Arguments

When variables (such as real-time data) are used as arguments, the value of the variable is passed, not the variable itself. The following example uses the `DspStr()` function to display the value of a process variable at AN25:

Command	DspStr(25, "TextFont", B1_TIC_101_PV);
----------------	--

In this instance, the **value** of `B1_TIC_101_PV` displays. If it is a real-time variable, the number that displays depends on its value at the time.

Note: If you use double quotes around variables, for example, "B1_TIC_101_PV", the text string `B1_TIC_101_PV` displays, rather than the value of the variable.

Using Operator Input in Functions

You can pass operator input to functions at runtime. For example, you can define a System Keyboard Command to let the operator select a page:

Key Sequence	F10 ##### Enter
Command	PageDisplay(Arg1);

When the command executes, the page name is passed to the function as Arg1. The operator can then display any page, for example:



Returning Data from Functions

Functions return data to the calling statement (a command or expression). Some functions simply return a value that indicates whether the function was successful. For example, both the PageNext() and PageDisplay() functions return 0 (zero) if the page displays successfully, otherwise they return an error number. For a large number of simple applications, you can ignore this return value.

Some functions return data that you can use in an expression or command. For example, the Date() function returns the current date as a string. To display the current date on a graphics page, use the following expression in a text object display value property:

Numeric expression	Date();
---------------------------	---------

The following example shows an entry command event for a graphics page, using a combination of two functions. The FullName() function returns the name of the user who is currently logged in to the run-time system, passing this name to the calling function, Prompt(). When the page is opened, a welcome message displays in the prompt line.

On page entry	Prompt("Hello, " + FullName())
----------------------	--------------------------------

For example, if the current user is John Citizen, the message "Hello, John Citizen" displays.

Chapter: 5 Working with Commonly Used Functions

Cicode has many functions that perform a variety of tasks. Many of these are used for building complex CitectSCADA systems. The functions you will often use are divided into six categories:

- [Alarm Functions](#)
- [Page Functions](#)
- [Keyboard Functions](#)
- [Report Functions](#)
- [Time/date Functions](#)
- [Miscellaneous Functions](#)

See Also

[Functions Reference](#)

Alarm Functions

You can use alarm functions to display alarms and their related alarm help pages, and to acknowledge, disable, and enable alarms. You can assign a privilege to each command that uses an alarm function, so that only an operator with the appropriate privilege can perform these commands. However, you should assign privileges to commands only if you have not assigned privileges to individual alarms.

- [AlarmAck](#): Acknowledges an alarm. The alarm where the cursor is positioned (when the command is executed) is acknowledged. You can also use this function to acknowledge multiple alarms.
- [AlarmComment](#): Adds a comment to the alarm summary entry at run time. The comment is added to the alarm where the cursor is positioned when the command is executed. A keyboard argument passes the comment into the function. Verify that the length of the comment does not exceed the length of the argument, or an error results.
- [AlarmDisable](#): Disables an alarm. The alarm where the cursor is positioned (when the command is executed) is disabled. You can also use this function to disable multiple alarms.

- [AlarmEnable](#): Enables an alarm. The alarm where the cursor is positioned (when the command is executed) is enabled. You can also use this function to enable multiple alarms.
- [AlarmHelp](#): Displays an alarm help page for the alarm. Each alarm in your system can have an associated help page. The help page for the alarm at the position of the cursor (when the command is executed) is displayed.
- [AlarmSplit](#):Duplicates an entry in the alarm summary display. You can use this function to add additional comments to the alarm entry.

Page Functions

With the page functions, you can display your graphics pages and the standard alarm pages.

Note: The following page functions are not supported in the server process in a multiprocessor environment. Calling page functions from the server process results in a hardware alarm being raised.

- [PageAlarm](#): Displays current alarms on the alarm page configured in the project.
- [PageDisabled](#): Displays disabled alarms on the alarm page configured in the project.
- [PageDisplay](#): Displays a new page on the screen. The Page name or number is required as an argument. (Use the PageLast() function to go back to the last page - the page that this new page replaced).
- [PageFile](#): Displays a file on the file page configured in the project.
- [PageGoto](#): Displays a new page on the screen. This function is similar to the PageDisplay() function, except that if PageLast() is called, it does not return to the last page.
- [PageHardware](#): Displays hardware alarms on the alarm page configured in the project.
- [PageLast](#): Displays the graphics page that was displayed before the current one. You can use this function to 'step back' through the last ten pages.
- [PageNext](#): Displays the next graphics page (defined in the Next Page property of the Pages form).
- [PagePrev](#): Displays the previous graphics page (defined in the Prev Page property of the Pages form).
- [PageSummary](#): Displays summary alarm information on the alarm page configured in the project.
- [PageTrend](#): Displays a standard trend page.

Keyboard Functions

Keyboard functions control the processing of keyboard entries and the movement of the keyboard cursor on the graphics page.

- [KeyBs](#): Backspaces (removes) the last key from the key command line. Use this function with a 'Hotkey' command. It is normally used to erase keyboard characters during runtime command input.
- [KeyDown](#): Moves the cursor down the page to the closest animation point number (AN).
- [KeyLeft](#): Moves the cursor left (across the page) to the closest animation point number (AN).
- [KeyRight](#): Moves the cursor right (across the page) to the closest animation point number (AN).
- [KeyUp](#): Moves the cursor up the page to the closest animation point number (AN).

Report Functions

To run a report by operator action, use the following function:

- [Report](#): Runs the report on the report server.

Time/date Functions

The following functions return the current date and time:

- [Date](#): Returns the current date as a string.
- [Time](#): Returns the current time as a string.

Miscellaneous Functions

- [Beep](#): Beeps the speaker on the CitectSCADA computer.
- [FullName](#): Returns the full name of the user who is currently logged in to the system.
- [InfoForm](#): Displays the animation information form. This form displays the real-time data that is controlling the current animation.
- [Login](#): Allows a user access to the CitectSCADA system.
- [LoginForm](#): Displays a dialog box to allow a user to log in to the system.
- [Logout](#): Logs the current user out of the CitectSCADA system.

- [Name](#): Returns the user name of the user who is currently logged in to the system.
- [Prompt](#): Displays a message on the screen. The message **String** is supplied as an argument to the function.
- [Shutdown](#): Terminates CitectSCADA. Use this function, or the ShutdownForm() function, to shut down your system. Otherwise buffered data may be lost.
- [ShutdownForm](#): Displays a dialog box to allow a user to shut down your Citect-SCADA system.

Chapter: 6 Writing Functions

CitectSCADA is supplied with over 600 built-in functions. One of these functions (or several functions in combination) can usually perform the required tasks in your system. However, where system functionality cannot be achieved with built-in functions, you can write your own functions.

A Cicode function is a small program: a collection of statements, variables, operators, conditional executors, and other functions.

While it is not necessary to be an experienced programmer to write simple Cicode functions, it is strongly recommended not to attempt to write large, complex functions unless you are familiar with computer programming, and have experience with Cicode. Functions are equivalent to the subroutines of BASIC and assembly language, and the subroutines and functions used in Pascal and C.

Note: The Cicode Editor is designed specifically for editing and debugging Cicode functions.

See Also

[The Cicode Editor](#)

[Using Cicode Files](#)

Cicode Function Structure

A function in Cicode can be described as a collection or list of sequential statements that CitectSCADA can perform (execute) in the logical order that they exist within the function.

A Cicode function starts with the FUNCTION statement and finishes with the END statement. Every statement that lie between the FUNCTION and END statements, will be executed by the function, when called to do so.

A typical Cicode function is structured like the following example:

```
FUNCTION
FunctionName ( )
    ! The exclamation point indicates that the rest of this line contains a comment.
    ! Further Cicode statements go here, between the function name and the END.
END
```

The line immediately following the FUNCTION statement, contains the name of the function, which is used to identify the function to CitectSCADA. This name is referred to when the function is called upon (called) to be executed (perform the statements it contains) by some other event, action, or function in CitectSCADA.

Note: Functions can contain statements that call other functions. These functions are then executed before returning to the rest of the statements within the calling function.

The function name has to end with parentheses (), which may or may not contain one or more arguments required by the function. Arguments are explained in the section titled [Function Argument Structure](#).

Every line between the function name line and the END statement line contain the statements that will be executed when the function is called in CitectSCADA. These statements are executed one at a time in logical order from top to bottom within the function. For details about function structure, see [Formatting Functions](#). For details about Cicode function syntax, see [Following Cicode Syntax](#).

For details about using comments in Cicode and in Cicode functions, see [Using Comments in Cicode](#).

Function Uses

Cicode functions can have many purposes. Quite often, functions are used to store a common set of commands or statements that would otherwise require repetitious typing and messy command or expression fields.

Some functions are simple, created to avoid a long command or expression. For example, the following command increments the variable tag COUNTER:

Com- mand	IF COUNTER < 100 THEN COUNTER = COUNTER + 1; ELSE COUNTER = 0; END;
----------------------------	--

This command would be easier to use (and re-use) if it was written as a function that can be called in the command:

Command	IncCounter ();
----------------	-----------------

To be able to use the function like this, you need to write it in a Cicode file, and declare it with the FUNCTION keyword:

```
FUNCTION
IncCounter ( )
    IF COUNTER < 100 THEN
        COUNTER = COUNTER + 1;
    ELSE
        COUNTER = 0;
    END
END
```

Be aware that the indented code is identical in functionality to the long command above.

By placing the command code inside a function, and using the function name in the command field as in the previous example, this function need only to be typed once. It can then be called any number of times, from anywhere in CitectSCADA that requires this functionality. Because the code exists in the one location, rather than repeated wherever needed (in potentially many places), it can be easily maintained (altered if necessary).

Writing Groups of Functions

To perform complex tasks you need careful design. Large, complex functions are not only more difficult to understand and debug than simple functions, but they can also hide tasks that are common to other activities.

Cicode functions allow a modular approach - complex tasks can be organized into small functions, each with a single, clear purpose. These small functions can then be called by other functions, or called directly in commands and expressions. In fact, any function can call - and be called by - any other function.

For example, you might need to write a set of functions for handling alarms. To perform any action on an alarm, you first need to know which alarm. You would identify the alarm in a separate function, and call this function from the other functions.

Cicode Function Libraries

Cicode functions are stored within Cicode files. You can use a separate file for each stand-alone function, or group several functions together into a common file. For easy maintenance, store functions that perform related tasks in the same file - for example, store functions that act on alarm data in an **Alarms.CI** file.

Note: Every Cicode file in your project directory will be included when you compile

your project.

Creating a Function Outline

First, define the purpose of the function group, and create an outline of the tasks to be performed. The following example shows an outline for a group of functions that change the threshold values of analog alarms during run time. The outline describes the workings of the function group, and is written in pseudocode (also called Program Design Language).

```
/*
This file contains functions to allow the operator to make runtime
changes to Analog Alarm thresholds.
This file has 4 functions. The master function calls the other
functions.
ChangeAnalogAlarmThresholds ( )
This calls in turn:
1:GetVariableTag ( )
Argument:    cursor position
Return:      name of variable tag at cursor
2:GetAlarmThresholds ( )
Argument:    tag name
Return:      threshold value of alarm
3:DisplayAlarmThresholds ( )
Argument:    threshold value of alarm
Displays threshold values in prompt line
Return:      success or error code
*/
```

Pseudocode

The pseudocode above is a Cicode comment, enclosed between the comment markers /* and */, and is ignored by the compiler. With pseudocode, you can get the logic of the function correct in a more readable structure, before you write it in Cicode syntax, leaving the pseudocode within the finished code as comments.

It is good practice to use comments as file headers at the start of each Cicode file, to describe the functions in the file - their common purpose, a broad description of how they achieve that purpose, special conditions for using them, and so on. You can also use the header to record maintenance details on the file, such as its version number and date of revision. For example:

```

/*
**     FILE:          Recipe Download.Ci
**
**     AUTHOR:        AJ Smith
**
**     DATE:          March 2008
**
**     REVISION:      1.0 for CitectSCADA v7.1
**
**     This file contains functions to allow the operator to load the
**     recipe data from the SQL server to the PLC.
*/

```

Following the file header are the functions in series:

```

/*
**     Main function
*/
FUNCTION
RecipeDownload ( )
! {body of function}
!
END
/*
**     Function to open the SQL connection.
*/
FUNCTION
RecipeConnectSQL ( )
! {body of function}
!
END
! (and so on)

```

Using Comments in Cicode

It is good programming practice to include comments in your Cicode files. Comments allow you to quickly understand how a function works next time you (or another designer) need to modify it.

The Cicode compiler recognizes the following single line, C style, and C++ style comments:

```

! A single line comment
WHILE DevNext ( hDev ) DO
    Counter = Counter + 1 ;      ! An in-line comment
END
/* A block comment is a C-style comment, and can
extend over several lines. Block comments need to
finish with a delimiter, but delimiters at the

```

```
start of each line are optional only. */
// A double-slash comment is a C++ style comment, for example:
Variable = 42;           // This is a comment
```

Single line (!) and C++ style (//) comments can have a line of their own, where they refer to the block of statements either before or after it. It is good practice to set a convention for these comments. These comments can also be on the same line as a statement, to explain that statement only. Any characters after the ! or // (until the end of the line) are ignored by the compiler.

Block (C style) comments begin with /* and end with */. These C style comments need no punctuation between the delimiters.

Using Comments for Debugging Functions

You can use comments to help with the debugging of your functions. You can use comments to temporarily have the compiler ignore blocks of statements by changing them to comments. C style and C++ style comments can be nested, for example.

```
FUNCTION
IncCounter ( )
    IF COUNTER < 100 THEN
        COUNTER = COUNTER + 1 ;
    /*
        ELSE                      // Comment about statement
        COUNTER = 0;              // Another comment
    */
    END
END
```

The complete ELSE condition of the IF conditional executor will be ignored (and not execute) so long as the block comment markers are used in this example.

Note: The inline (//) comments have no effect within the block /* and */ comments (as the whole section is now one big comment), and should remain unchanged, so that when you do remove the block comments, the inline comments will become effective again.

Tag Reference /TagReadEx() behavior in Cicode Expressions

The following table describes the tag reference and [TagReadEx\(\)](#) behavior in a Cicode expression if the quality of the tag is BAD:

Tag Reference / TagReadEx syntax	Error Mode/Citect.ini settings	Cicode Expression behavior
"Tag1"	ErrSet(0) [Code]HaltOnInvalidTagData = 0	Tag ref returns a BAD quality value, Cicode expression continues, Error is set.
TagReadEx("Tag1")	ErrSet(0) [Code]HaltOnError = 0	Function returns a BAD quality value, Cicode expression continues, Error is set.
"Tag1"	ErrSet(0) [Code]HaltOnInvalidTagData = 1	Tag ref returns a BAD quality value, Cicode expression stops.
TagReadEx("Tag1")	ErrSet(0) [Code]HaltOnError = 1	Function returns a BAD quality value, Cicode expression stops.
"Tag1"	ErrSet(1)	Tag ref returns a BAD quality value, Cicode expression continues, Error is set.
TagReadEx("Tag1")	ErrSet(1)	Function returns a BAD quality value, Cicode expression continues, Error is set.
"Tag1.V"	ErrSet(0) or ErrSet(1)	Tag ref returns a GOOD quality value, Cicode expression continues, No error is set.
TagReadEx("Tag1.V")	ErrSet(0) or ErrSet(1)	Function returns a GOOD quality value, Cicode expression continues, No error is set.

See Also

[TagReadEx\(\)](#)

[Tag Functions](#)

Following Cicode Syntax

Some programming languages have strict rules about how the code needs to be formatted, including the indenting and positioning of the code structure. Cicode has no indenting or positioning requirements, allowing you to design your own format - provided only that you follow the correct syntax order for each statement. However, it is a good idea to be consistent with your programming structure and layout, so that it can be easily read and understood.

For details about programming standards, see the section titled [Using Cicode Programming Standards](#), which includes sections on:

- Standards for constants, variable tags, and labels
- Standards variables: declaration, scope, and naming
- Standards for functions: naming , file headers, headers

- Formatting of: declarations, statements, expressions, and functions
- Use of comments

For information on problem solving, see the sections on [Modular Programming](#), [Defensive Programming](#), [Function Error handling](#), or [Debugging Cicode](#).

The following is an example of a simple Cicode function:

```
/*
This function is called from a keyboard command. The operator
presses the key and enters the name of the page to be displayed. If
the page cannot be displayed, an error message is displayed at the
prompt AN.
*/
INT
FUNCTION
MyPageDisplay ( STRING sPage ) ! pass in the name of the page to be displayed
    ! declare a local integer to hold the results of the pagedisplay function
    INT Status;
    ! call the page Cicode pagedisplay function and store the result
    Status = PageDisplay ( sPage ) ;
    ! determine if the page display was successful
    IF Status < > 0 THEN ! error was detected
        ! display an error message at the prompt AN
        DspError ( "Cannot Display " + sPage ) ;
    END
    ! return the status to the caller
    RETURN Status;
END
```

The rules for formatting statements in Cicode functions are simple, and help the compiler in interpreting your code.

It is good practice to use white space to make your code more readable. In the example above, the code between the FUNCTION and END statements is indented, and the statement within the IF THEN conditional executor is further indented to make the conditions and actions clear. Develop a pattern of indentation - and stick to it. Extra blank lines in the code make it easier to read (and understand).

Cicode Function Syntax

Note: In the following function syntax example:

- Every placeholder shown inside arrow brackets (<placeholder>) should be replaced in any actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown here only for your information.

- Statements shown between square brackets ([]) are optional. The square brackets should not be included in the statement, and are shown here only for your information.

Cicode functions have the following syntax:

```
[ <Scope> ]
[ <ReturnDataType> ]
FUNCTION
<FunctionName> ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
    RETURN <ReturnValue> ;
END
```

where:

- <Scope> = Scope Statement: optional, PRIVATE or PUBLIC, default PUBLIC, no semicolon. See the section titled [Function Scope](#).
- <ReturnDataType> = Return Data Type Statement: optional and one of INT, REAL, STRING, OR OBJECT. No default, no semicolon. If no return type is declared, the function cannot return any data. See the section titled [Declaring the Return Data Type](#).
- FUNCTION = FUNCTION Statement: required, indicates the start of the function, keyword, no semicolon. See the section titled [Declaring Functions](#).
- <FunctionName> = Name statement: required, up to 32 ASCII text characters, case insensitive, no spaces, no reserved words, no default, no semicolon. See the section titled [Naming Functions](#).
- (<Arguments>) = Argument statement: surrounding brackets required even if no arguments used, if more than one argument - each need to be separated by a comma, can contain constants or variables of INT or REAL or STRING or QUALITY or TIMES-TAMP data type, default can be defined in declaration, can be spread over several lines to aid readability, no semicolon. See the section titled [Function Argument Structure](#).
- <Statement> = Executable Statement: required, one or more executable statements that perform some action in CitectSCADA, often used to manipulate data passed into the function as arguments, semicolon required.
- RETURN = RETURN Statement: optional, used to instruct Cicode to return a value to the caller of the function - usually a manipulated result using the arguments passed in to the function by the caller, need to be followed by Return Value Statement, keyword, no semicolon.
- <ReturnValue> = Return Value Statement; required if RETURN Statement used in function, need to be either a constant or a variable, the data type need to have been

previously declared in the function Return Data Type Statement - or does not return a value, semicolon required. See the section titled [Returning Values from Functions](#).

- END = END Statement: required, indicates the end of the function, keyword, no semicolon. See the section titled [Declaring Functions](#).

End of line markers

Most statements within the function are separated by semicolons (;) but some exceptions exist. The FUNCTION and END Statements (the start and end of the function) have no semicolons, nor does the Scope or Return Data Type Statements, nor any statement that ends with a reserved word.

Where a statement is split over several lines (for example, within the IF THEN conditional executor), each line ends with a semicolon - unless it ends in a reserved word.

Function Scope

The optional Scope Statement of a function (if used), precedes all other statements of a function declaration in Cicode, including the FUNCTION Statement.

The scope of a function can be either PRIVATE or PUBLIC, and is declared public by default. That is, if no Scope Statement is declared, the function will have public scope.

Both PRIVATE and PUBLIC are Cicode keywords and as such, are reserved.

A private scope function is only accessible (can be called) within the file in which it is declared.

Public scope functions can be shared across Cicode files, and can be called from pages and CitectSCADA databases (for example, Alarm.dbf).

Because functions are public by default, to make a function public requires no specific declaration. To make a function private however, you need to prefix the FUNCTION Statement with the word **PRIVATE**.

```
PRIVATE
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

Declaring the Return Data Type

For information about the RETURN Statement, see the section titled [Returning Values from Functions](#).

The optional Return Data Type Statement of a function (if used), follows the optional Scope Statement (if used), and precedes the FUNCTION Statement declaration in Cicode.

The return data type of a function can be only one of six possible data types: INT (32 bits), REAL (64 bits), STRING (255 bytes), OBJECT (32 bits), QUALITY or TIMESTAMP (64 bits). If no Return Data Type Statement is declared, the function will not be able to return any type of data.

INT, REAL, STRING, OBJECT, QUALITY and TIMESTAMP are Cicode keywords and as such, are reserved.

Note: In the following function syntax example, every placeholder shown inside arrow brackets (<placeholder>) should be replaced in the actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown only for your information.

To declare the data type that will be returned to the calling code, prefix the FUNCTION Statement with one of the Cicode data type keywords, in the <ReturnDataType> placeholder in the following example.

```
<ReturnDataType>
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

The following example returns an integer of value 5:

```
INT
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    INT Status = 5;
    <Statement> ;
    RETURN Status;
END
```

If the RETURN Statement within the function encounters a different data type to that declared in the return data type statement, the value is converted to the declared return data type.

In the example below, the variable `Status` is declared as a real number within the function. However, `Status` is converted to an integer when it is returned to the caller, because the data type of the return was declared as an integer type in the return data type statement:

```
INT          ! declare return value as integer
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    REAL Status = 5;           ! declare variable as a REAL number
    <Statement> ;
    RETURN Status;           ! returned as an integer number
END
```

If you omit the return data type, the function does not return a value.

Declaring Functions

The required FUNCTION Statement follows the optional Scope Statement (if used) and the optional Return Data Type Statement (if used), and precedes any other statements of a function declaration in Cicode. Everything between it and the END Statement, contains the function.

Both FUNCTION and END are Cicode keywords and, as such, are reserved.

You declare the start of a function with the FUNCTION Statement, and declare the end of a function with the END Statement:

```
FUNCTION
<FunctionName> ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

The FUNCTION Statement needs to be followed by the Name Statement, then the Argument Statement, before any code statements that will be processed by the function.

For information on the Name and Argument Statements, see the sections titled [Naming Arguments](#) and [Function Argument Structure](#).

The code (as represented by the <Statement> placeholders) located between the FUNCTION and END Statements, will be executed (processed by the function) when called to do so.

Functions can execute a large variety of statements, and are commonly used to process and manipulate data, including the arguments passed when the function was called, plant-floor and other CitectSCADA data, Windows data, and so on. CitectSCADA provides many built-in functions. For more information, see the section titled [Working with Commonly Used Functions](#).

Naming Functions

The required name statement follows the FUNCTION Statement and precedes the arguments statement in a CitectSCADA function. The function name is used elsewhere in CitectSCADA to activate (call) the function to have it perform the statements it contains.

Replace the <FunctionName> placeholder in the following function example with an appropriate name for your function. See the section [Function Naming Standards](#) for details.

```
FUNCTION
<FunctionName> ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

You can use up to 32 ASCII text characters to name your functions. You can use any valid name except for a reserved word. The case is ignored by the CitectSCADA compiler, so you can use upper and lower case to make your names clear. For example, MixerRoomPageDisplay is easier to read than mixerroompagedisplay or MIXERROOMPAGEDISPLAY.

```
FUNCTION
MixerRoomPageDisplay ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

Your functions take precedence over any other entity in CitectSCADA with the same name:

- **Variable tags.** When you call a function by the same name as a variable tag, the function has precedence. The variable tag can not be referred to because the function executes each time the name is used.
- **Built-in functions.** You can give your function the same name as any built-in Cicode function. Your function takes precedence over the built-in function - the built-in function cannot be called. Because built-in Cicode functions cannot be changed, this

provides a method of 'modifying' any built-in function to suit an application. For example, you might want to display the message "Press F1 for Help" whenever you display a page. You could simply write a new function called PageDisplay(). The body of the function would be the statements that display the page and prompt message:

```
Prompt ( "Press F1 for Help" ) ;PageDisplay ( <Arguments> ) ;
```

Your function is invoked whenever you use the function name in CitectSCADA.

Function Argument Structure

The optional Arguments Statement follows the required FUNCTION Statement and precedes the executable statements of a function in Cicode.

Note: The maximum number of arguments you can have in a function is 128.

When you call a function, you can pass one or more arguments to the function, enclosed within the parentheses () located after the function name statement. Replace the <Arguments> placeholder in the following function example with your Argument Statement.

```
FUNCTION  
FunctionName ( <Arguments> )  
    <Statement> ;  
    <Statement> ;  
    <Statement> ;  
END
```

For your function to perform tasks with data, it requires accessibility to the data. One way to achieve this, is to pass the data directly to the function when the function is being called. To enable this facility, Cicode utilizes arguments in its function structure. An argument in Cicode is simply a variable that exists in memory only as long as its function is processing data, so the scope of an argument is limited to be local only to the function. Arguments cannot be arrays.

Arguments are variables that are processed within the body of the function only. You cannot use an argument outside of the function that declares it.

As arguments are variables used solely within functions, they need to be declared just as you would otherwise declare a variable in Cicode. See the section titled [Declaring Variable Properties](#). An argument declaration requires a data type, a unique name, and may contain an initial value which also behaves as the default value for the argument.

Notes: In the following function syntax example:

- Every placeholder shown inside arrow brackets (`<placeholder>`) should be replaced in any actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown here only for your information.
- Statements shown between square brackets (`[]`) are optional. The square brackets should not be included in the statement, and are shown here only for your information.

Cicode function argument statements have the following syntax:

```
<ArgumentDataType>
<ArgumentName>
[ = <InitialDefaultValue> ]
```

where:

- `<ArgumentDataType>` = Argument Data Type Statement: required, INT or REAL or STRING. See the section titled [Declaring Argument Data Type](#).
- `<ArgumentName>` = Argument Name Statement: required, up to 32 ASCII text characters, case insensitive, no spaces, no reserved words. See the section titled [Naming Arguments](#).
- `<InitialDefaultValue>` = Argument Initialization Statement: optional, preceded by equals (=) assignment operator, a value to assign to the argument variable when first initialized, needs to be the same data type as that declared in the argument `<ArgumentDataType>` parameter, defaults to this value if no value passed in for this argument when the function was called.
See the section titled [Setting Default Values for Arguments](#).

The Argument Statement in a Cicode function can have only one set of surrounding parentheses (), even if no arguments are declared in the function.

If more than one argument is used in the function, each needs to also be separated by a comma.

Argument Statements can be separated over several lines to aid in their readability.

When you call a function, the arguments you pass to it are used within the function to produce a resultant action or return a value. For information on passing data to functions, see the section titled [Passing Data to Functions \(Arguments\)](#). For information on returning results from functions, see the section titled [Returning Data from Functions](#).

Arguments are used in the function and referred to by their names. For instance, if we name a function AddTwoIntegers, and declare two integers as arguments naming them FirstInteger and SecondInteger respectively, we would end up with a sample function that looks like the following:

```
INT
FUNCTION
AddTwoIntegers ( INT FirstInteger, INT SecondInteger )
    INT Solution ;
    Solution = FirstInteger + SecondInteger ;
    RETURN Solution ;
END
```

In this example, the function would accept any two integer values as its arguments, add them together, and return them to the caller as one integer value equal to the summed total of the arguments values passed into the function.

This functionality of passing values into a function as arguments, manipulating the values in some way, then being able to return the resultant value, is what makes functions potentially very powerful and time saving. The code only needs to be written once in the function, and can be utilized any number of times from any number of locations in CitectSCADA. Write once, use many.

Declaring Argument Data Type

If an argument is listed in a Cicode function declaration, the Argument Data Type Statement is required, and is listed first before the required Argument Name Statement and the optional Argument Initialisation Statement.

The argument data type of a function can be only one of six possible data types: INT (32 bits), REAL (32 bits), STRING (255 bytes), OBJECT (32 bits), QUALITY or TIMESTAMP (64 bits).

INT, REAL, STRING, OBJECT, QUALITY and TIMESTAMP are Cicode keywords and as such, are reserved.

Note: In the following function syntax example:

- Every placeholder shown inside arrow brackets (**<placeholder>**) should be replaced in any actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown here only for your information.
- Statements shown between square brackets ([]) are optional. The square brackets should not be included in the statement, and are shown here only for your information.

To declare the argument data type that will be used in the function, you need to prefix the Argument Name Statement with one of the Cicode data type keywords, in the **<ArgumentDataType>** placeholder in the following example.

```

FUNCTION
FunctionName ( <ArgumentDataType> <ArgumentName> [ =
<InitialDefaultValue> ] )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END

```

The Argument Statement in a Cicode function needs to have only one set of surrounding parentheses () brackets, even if no arguments are declared in the function.

If more than one argument is used in the function, each needs to also be separated by a comma.

Argument Statements can be separated over several lines to aid in their readability.

Naming Arguments

If an argument is listed in a Cicode function declaration, the Argument Name Statement is required, and is listed second, after the required Argument Data Type Statement, and before the optional Argument Initialization Statement.

The argument name is used only within the function to refer to the argument value that was passed into the function when the function was called. The name of the argument variable should be used in the executable statements of the function in every place where you want the argument variable to be used by the statement.

Note: In the following function syntax example:

- Every placeholder shown inside arrow brackets (<placeholder>) should be replaced in any actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown here only for your information.
- Statements shown between square brackets ([]) are optional. The square brackets should not be included in the statement, and are shown here only for your information.

Replace the <ArgumentName> placeholder in the following function example with an appropriate name for your Argument variable. See the section titled [Function Argument Structure](#) for details.

```

FUNCTION
FunctionName ( <ArgumentDataType> <ArgumentName> [ = <InitialDefaultValue> ] )
    <Statement> ;
    <Statement> ;
    <Statement> ;

```

```
END
```

You can use up to 32 ASCII text characters to name your arguments. You can use any valid name except for a reserved word. The case is ignored by the CitectSCADA compiler, so you can use upper and lower case to make your names clear. For example, iPacketQty is easier to read than ipacketqty or IPACKETQNTY .

```
FUNCTION
FunctionName ( INT iPacketQty )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

To refer to the argument (in the body of your function) you use the name of the argument in an executable statement:

```
INT
FUNCTION
AddTwoIntegers ( INT FirstInteger, INT SecondInteger )
    INT Solution ;
    Solution = FirstInteger + SecondInteger ;
    RETURN Solution ;
END
```

Setting Default Values for Arguments

If an argument is listed in a Cicode function declaration, the Argument Initialisation Statement is optional, and if used, is listed last in the Argument Statement after the required Argument Data Type and the Argument Name Statements. The Argument Initialization Statement needs to be preceded by an equals (=) assignment operator.

Note: In the following function syntax example:

- Every placeholder shown inside arrow brackets (<placeholder>) should be replaced in any actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown here only for your information.
- Statements shown between square brackets ([]) are optional. The square brackets should not be included in the statement, and are shown here only for your information.

Replace the <InitialDefaultValue> placeholder in the following function example with an appropriate value for your Argument variable.

```
FUNCTION
FunctionName ( <ArgumentDataType> <ArgumentName> [ =
<InitialDefaultValue> ] )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

The default value for an argument needs to be of the same data type as declared for the argument in the Argument Data Type Statement.

You assign a default argument variable value in the same manner that you assign a Cicode variable value, by using the equals (=) assignment operator. For example:

```
FUNCTION
PlotProduct ( INT iPackets = 200 , STRING sName = "Packets" )
    <Statement> ;
    <Statement> ;
    <Statement> ;
END
```

If you assign a default value for an argument, you may omit a value for that argument when you call the function, (because the function will use the default value from the declaration.) To pass an empty argument to a function, omit any value for the argument in the call. For example, to call the PlotProduct function declared in the previous example, and accept the default string value of "Packets", a Cicode function call would look like:

```
PlotProduct ( 500 , )
```

Be aware that the second argument for the function was omitted from the calling code. In this instance, the default value for the second argument ("Packets") would remain unchanged, and so would be used as the second argument value in this particular function call.

If you do call that function and pass in a value for that argument in the call, the default value is replaced by the argument value being passed in. However, the arguments are reinitialized every time the function is called, so each subsequent call to the function will restore the default values originally declared in the function.

If more than one argument is used in a function, each needs to also be separated by a comma. Equally, if a function containing more than one argument is called, each argument needs to be accounted for by the caller. In this case, if an argument value is to be omitted from the call, (to utilise the default value), comma placeholders need to be used appropriately in the call to represent the proper order of the arguments.

For more information on function calls, callers, and calling, see the section titled [Calling Functions from Commands and Expressions](#).

Argument Statements can be separated over several lines to aid in their readability.

Returning Values from Functions

Many of the built-in Cicode functions supplied with CitectSCADA return a data value to their calling statement. Mathematical functions return a calculated value. The Date () and Time () functions return the current date and time. Other functions, like Page-Display (), perform an action, and return a value indicating either the success of the action or the type of error that occurred.

You can also use return values in your own functions, to return data to the calling statement. The return value is assigned in the RETURN Statement:

The optional RETURN Statement of a function (if used), needs to be placed in the executable Statements section of a Cicode function between the FUNCTION and END Statements. Because the RETURN Statement is used to return data values that have usually been manipulated by the function, they are usually placed last just before the END Statement.

```
<ReturnDataType>
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    <Statement> ;
    <Statement> ;
    RETURN <ReturnValue> ;
END
```

The RETURN Statement consists of the RETURN keyword followed by a value to be returned and finished with the semicolon (;) end-of-line marker.

The RETURN value needs to be of the same data type as was declared in the Return Data Type Statement at the start of the function declaration. The return data type of a function can be only one of six possible data types: INT (32 bits), REAL (64 bits), STRING (255 bytes), OBJECT (32 bits), QUALITY or TIMESTAMP (64 bits). If no Return Data Type Statement is declared, the function will not be able to return any type of data.

If the RETURN Statement within the function encounters a different data type to that declared in the Return Data Type Statement, the value is converted to the declared return data type. For information about the Return Data Type Statement, see the section titled [Declaring the Return Data Type](#).

FUNCTION, INT, REAL, STRING, and OBJECT are Cicode keywords and as such, are reserved.

Note: In the following function syntax example every placeholder shown inside arrow brackets (<placeholder>) should be replaced in any actual code with the value of the item that it describes. The arrow brackets and the word they contain should not be included in the statement, and are shown here only for your information.

To declare the value that will be returned to the calling code, you need to replace the <ReturnValue> placeholder in the following example with an appropriate data value to match the Return Data Type as declared in the function.

```
<ReturnDataType>
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    <Statement> ;
    RETURN <ReturnValue> ;
END
```

The following example returns an integer of value 5:

```
INT
FUNCTION
FunctionName ( <Arguments> )
    <Statement> ;
    INT Status = 5;
    <Statement> ;
    RETURN Status;
END
```

The RETURN statement passes a value back to the calling procedure (either another function, command or expression). Outside of the function, the return value can be read by the calling statement. For example, it can be used by the caller as a variable (in a command), or animated (in an expression).

Chapter: 7 Using Variables

A variable is a named location in the computer's memory where data can be stored. Cicode variables can store the basic data types (such as strings, integers, and real numbers) and each variable is specific for its particular data type. For example, if you set up a Cicode variable to store an integer value, you cannot use it for real numbers or strings.

Note: Each data type uses a fixed amount of memory: integers use 4 bytes of memory, real numbers use 4 bytes, and strings use 1 byte per character. PLC INT types use only 2 bytes.

The computer allocates memory to variables according to the data type and the length of time you need the variable to be stored.

Real-time variables (such as PLC variables) are already permanently stored in database files on your hard disk. Any variable you use in a database field command or expression needs to be defined as a variable tag, or the compiler will report an error when the system is compiled.

Note: Cicode variables can handle a wide range of CitectSCADA variable tag data types. For example, a Cicode variable of INT data type can be used to store I/O device data types: BCD, BYTE, DIGITAL, INT, LONG, LONGBCD, and UINT.

See Also

[Using Arrays](#)

[Variable Declaration Standards](#)

[Variable Naming Standards](#)

[Variable Scope Standards](#)

[Using Cicode Files](#)

Declaring Variable Properties

You need to declare each variable used in your functions (except for variables that are configured as variable tags). In the declaration statement, you specify the name and data type of the variable. You can also set a default value for the variable.

Declaring the Variable Data Type

You can use variables of the following data types:

INT	Integer (32 bits)	-2,147,483,648 to 2,147,483,647
REAL	Floating point (64 bits)	-3.4E38 to 3.4E38
STRING	Text string (128 bytes maximum, including null termination character)	ASCII (null terminated)
OBJECT	ActiveX control	
QUALITY	Represents the CitectSCADA quality	QUAL_GOOD, QUAL_BAD, QUAL_UNCR
TIMESTAMP	64-bit value representing the number of 100-nano-second intervals since January 1, 1601	

If you want to specify a digital data type, use the integer type. Digital types can either be TRUE(1) or FALSE(0), as can integer types.

Note: Cicode may internally store floating point values as 64 bit to minimize rounding errors during floating point calculations.

QUALITY Data Type

The QUALITY data type is a new data type in Cicode which incorporates the CitectSCADA quality. The QUALITY data type and the Cicode quality labels can be used in Cicode expressions.

The operators allowed for the QUALITY data type are:

- Assignment operator: =.
- Relational operators: =, <>.

The assignment operation also allows for the QUALITY data type.

Example:

```
QUALITY q1;
QUALITY q2;

q1 = q2;
q1 = Tag1.Field.Q;
```

```
//the following expression will generate a compiler error as a tag //element can be modified only
as a whole
Tag1.Field.Q = q1;
```

A set of Cicode functions are provided which allow quality fields to be initialized, a specific quality field to be extracted, and other operations on the QUALITY data type. Conversion between the QUALITY data type and other Cicode data types is not allowed. Direct conversion from Quality to string will return an empty string.

Example:

```
//this will generate a compiler error
INT n = Tag1.Q;
```

TIMESTAMP Data Type

The TIMESTAMP data type is a new data type in Cicode which represents the date and time as a 64-bit value by specifying the number of 100-nanosecond intervals since January 1, 1601 .

The operators allowed for the TIMESTAMP data type are:

- Assignment operator: =.
- Relational operators: =, <>, <, >, <=, >=.

Example:

```
TIMESTAMP t1;
TIMESTAMP t2;

t1 = Tag1.T;
t1 = t2;

IF      t1 < Tag2.T THEN
// insert code here
END
```

A set of Cicode functions are provided which allow initialization, conversion and other operations on the TIMESTAMP data type. Implicit conversion between the TIMESTAMP data type and other Cicode data types is not allowed.

Naming Variables

Throughout the body of the function, the variable is referred to by its name. You can name a variable any valid name except for a reserved word, for example:

```
STRING      sStr;
REAL       Result;
```

```
INT          x, y;
OBJECT      hObject;
```

The first 32 characters of a variable name needs to be unique.

See Also

[Variable Naming Standards](#)

Setting Default Variable Values

When you declare variables, you can set them to an initial (startup) value; for example:

```
STRING      Str = "Test";
REAL        Result = ;
INT         x = 20, y = 50;
```

Using Variable Scope

Scope refers to the accessibility of a function and its values. A Cicode variable can be defined as any one of three types of scope - global, module, and local. By default, Cicode variables are module scope, unless they are declared within a function.

Variables have the following format:

```
DataType Name [=Value];
```

Global variables

A global Cicode variable can be shared across all Cicode files in the system (as well as across include projects). They cannot be accessed on pages or databases (for example, Alarm.dbf).

Global Cicode variables are prefixed with the keyword **GLOBAL**, and needs to be declared at the start of the Cicode file. For example:

```
GLOBAL STRING sDefaultPage = "Mimic";
INT
FUNCTION
MyPageDisplay(STRING sPage)
    INT iStatus;
    iStatus = PageDisplay(sPage);
    IF iStatus <> 0 THEN
```

```

        PageDisplay(sDefaultPage);
    END
    RETURN iStatus;
END

```

The variable **sDefaultPage** could then be used in any function of any Cicode file in the system.

Note: Use global variables sparingly if at all. If you have many such variables being used by many functions, finding bugs in your program can become time consuming. Use local variables wherever possible. Global Cicode STRING types are only 128 bytes, instead of 256 bytes.

Module variables

A module Cicode variable is specific to the file in which it is declared. This means that it can be used by any function in that file, but not by functions in other files.

By default, Cicode variables are defined as module, therefore prefixing is not required (though a prefix of **MODULE** could be added if desired). Module variables should be declared at the start of the file. For example:

```

STRING sDefaultPage = "Mimic";
INT
FUNCTION
MyPageDisplay(STRING sPage)
    INT Status;
    Status = PageDisplay(sPage);
    IF Status <> 0 THEN
        PageDisplay(sDefaultPage);
    END
    RETURN Status;
END
INT
FUNCTION
DefaultPageDisplay()
    PageDisplay(sDefaultPage);
END

```

Note: Use module variables sparingly if at all. If you have many such variables being used by many functions, finding bugs in your program can become time-consuming. Use local variables wherever possible.

Local variables

A local Cicode variable is only recognized by the function within which it is declared, and can only be used by that function. You need to declare local variables before you can use them.

Any variable defined within a function (that is, after the function name) is a local variable, therefore no prefix is needed. Local variables are destroyed when the function exits.

Local variables take precedence over global and module variables. If you define a local variable in a function with the same name as a global or module variable, the local variable is used; the global/module variable is unaffected by the function. This situation should be avoided, however, as it is likely to cause confusion.

Local Variables and Variable Tags

Local variables have limited functionality compared with variable tags. Limitations are:

- Qualities of Override, OverrideMode, ControlMode and Status elements are showing Bad with extended substatus QUAL_EXT_INVALID_ARGUMENT. Writing to the elements returns error CT_ERROR_INVALID_ARG.
- Values of Override, OverrideMode, ControlMode and Status elements are showing 0.
- Respective timestamps and quality of Field, Valid and default elements are the same.
- Field, Valid and default elements can be read.
- Field and default elements can be written.

See Also

[Variable Scope Standards](#)

Using Database Variables

You can use any variable that you have defined in the database (with the Variable Tags form) in your functions. To use a database variable, specify the tag name:

```
<Tag>
```

where `Tag` is the name of the database variable. For example, to change the value of the database variable "LT131" at run time, you would use the following statement in your function:

```
LT131=1200;      !Changes the value of LT131 to 1200
```

Chapter: 8 Using Arrays

A Cicode variable array is a collection of Cicode variables of the same data type, in the form of a list or table. You name and declare an array of variables in the same way as any other Cicode variable. You can then refer to each element in the array by the same variable name, with a number (index) to indicate its position in the array.

See Also

[Variable Declaration Standards](#)

[Declaring Array Properties](#)

[Declaring the Array Data Type](#)

[Naming Arrays](#)

[Declaring the Variable Array Size](#)

[Setting Default \(Initial\) Array Values](#)

[Passing Array Elements as Function Arguments](#)

[Using One-dimensional Arrays](#)

[Using Two-dimensional Arrays](#)

[Using Three-dimensional Arrays](#)

[Using Array Elements in Loops](#)

[Using the Table \(Array\) Functions](#)

[Using Cicode Files](#)

Declaring Array Properties

Arrays have several properties that you need to declare to the compiler along with the array name: data type, size and dimension. You can also set default values for individual elements of the array. An array declaration has the following syntax:

```
DataType Name[Dim1Size,{Dim2Size},{Dim3Size}]{=Values};
```

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Declaring the Array Data Type

As with any other Cicode variable, arrays can have four Data Types:

INT	Integer (32 bits)
REAL	Floating point (32 bits)
STRING	Text string (255 bytes)
OBJECT	ActiveX object (32 bits)
QUALITY	CitectSCADA Quality
TIMESTAMP	Date and Time (64 bits)

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Naming Arrays

Throughout the body of a Cicode function, a Cicode variable array is referred to by its name, and individual elements of an array are referred to by their index. The index of the first element of an array is 0 (that is a four element array has the indices 0,1,2, and 3). You can name a variable any valid name except for a reserved word; for example:

```
STRING      StrArray[5];      ! list
REAL       Result[5][2];      ! 2-D table
INT        IntArray[4][3][2];  ! 3-D table
```

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Declaring the Variable Array Size

You need to declare the size of the array (the number of elements the array contains), for example:

```
STRING      StrArray[5];
```

This single dimension array contains 5 elements. The compiler multiplies the number of elements in the array by the size of each element (dependent upon the Data Type), and allocates storage for the array in consecutive memory locations.

You cannot declare arrays local to a function. However, they can be declared as Module (that is at the beginning of the Cicode file), or Global. When referring to the array within your function, take care to remain within the size you set when you declared the array. The example below would cause an error:

```
STRING  StrArray[5];
...
StrArray[10] = 100;
...
```

The compiler allows storage for 5 strings. By assigning a value to a 10th element, you cause a value to be stored outside the limits of the array, and you could overwrite another value stored in memory.

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Setting Default (Initial) Array Values

When you declare an array, you can (optionally) set the individual elements to an initial (or start-up) value within the original declaration statement. For instance, naming a string array "ArrayA", sizing it to hold 5 elements, and initializing the array with string values, would look like the following example:

```
STRING ArrayA[5] = "This", "is", "a", "String", "Array";
```

This array structure would contain the following values:

```
ArrayA[0] = "This"
ArrayA[1] = "is"
ArrayA[2] = "a"
ArrayA[3] = "String"
ArrayA[4] = "Array"
```

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Passing Array Elements as Function Arguments

To pass a Cicode variable array element to a Cicode function, you need to provide the element's address; for example:

```
/* Pass the first element of ArrayA. */
MyFunction (ArrayA[0])
/* Pass the second element of ArrayA. */
MyFunction (ArrayA[1])
/* Pass the fifth element of ArrayA. */
MyFunction (ArrayA[4])
```

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Using One-dimensional Arrays

To use a one-dimensional array:

```
STRING ArrayA[5] = "This", "is", "a", "String", "Array";
This array sets the following values:
ArrayA[0] = "This"
ArrayA[1] = "is"
ArrayA[2] = "a"
ArrayA[3] = "String"
ArrayA[4] = "Array"
```

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Using Two-dimensional Arrays

To use a two-dimensional array:

```
REAL ArrayA[5][2] = 1, 2, 3, 4, 5, 6, 7, 8.3, 9.04, 10.178;
```

This array sets the following values:

ArrayA[0][0]=1	ArrayA[0][1]=2
ArrayA[1][0]=3	ArrayA[1][1]=4
ArrayA[2][0]=5	ArrayA[2][1]=6
ArrayA[3][0]=7	ArrayA[3][1]=8.3
ArrayA[4][0]=9.04	ArrayA[4][1]=10.178

See Also[Using Arrays](#)[Using Cicode Files](#)

Using Three-dimensional Arrays

To use a three-dimensional array:

```
INT ArrayA[4][3][2]=1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,
16,17,18,19,20,21,22,23,24;
```

This array sets the following values:

ArrayA[0][0][0]=1	ArrayA[0][0][1]=2	ArrayA[0][1][0]=3
ArrayA[0][1][1]=4	ArrayA[0][2][0]=5	ArrayA[0][2][1]=6
ArrayA[1][0][0]=7	ArrayA[1][0][1]=8	ArrayA[1][1][0]=9
ArrayA[1][1][1]=10	ArrayA[1][2][0]=11	ArrayA[1][2][1]=12
ArrayA[2][0][0]=13	ArrayA[2][0][1]=14	ArrayA[2][1][0]=15
ArrayA[2][1][1]=16	ArrayA[2][2][0]=17	ArrayA[2][2][1]=18
ArrayA[3][0][0]=19	ArrayA[3][0][1]=20	ArrayA[3][1][0]=21
ArrayA[3][1][1]=22	ArrayA[3][2][0]=23	ArrayA[3][2][1]=24

You use arrays in your functions in the same way as other variables, but arrays have special properties that, in many situations, reduce the amount of code you need to write.

See Also

[Using Arrays](#)

[Using Cicode Files](#)

Using Array Elements in Loops

You can set up loops that deal efficiently with arrays by incrementing the index number. The following example shows a method of initializing an array:

```
REAL Array[10]
:
FOR Counter = 0 TO 9 DO
    Array[Counter] = 0
END
RETURN Total
:
```

See Also

[Working with Conditional Executors](#)

[Using Arrays](#)

[Using Cicode Files](#)

Using the Table (Array) Functions

Cicode has built-in functions for processing Cicode variable arrays:

- To perform calculations (max, min, total, etc.) on array elements.
- To look up the index number of an array element.
- To shift the elements of an array left or right.

See Also

[Table \(Array\) Functions](#)

[Using Arrays](#)

[Using Cicode Files](#)

Chapter: 9 Using Cicode Macros

Cicode has the following macros:

- [**IFDEF**](#): Determines one of two possible outcomes based on the existence of a specified non-alarm tag at compile time. Use one of the macros below for alarm tags.
- [**IFDEFAdvAlm**](#): Determines one of two possible outcomes based on the existence of a specified advanced alarm tag at compile time.
- [**IFDEFAnaAlm**](#): Determines one of two possible outcomes based on the existence of a specified analog alarm tag at compile time.
- [**IFDEFDigAlm**](#): Determines one of two possible outcomes based on the existence of a specified digital alarm tag at compile time.

IFDEF

The IFDEF macro allows you to define two possible outcomes based on whether or not a specified tag exists within a project at the time of compiling. The macro can be implemented anywhere a simple expression is used, including fields within relevant Citect-SCADA dialogs.

The macro was primarily created to avoid the "Tag not found" compile error being generated whenever a genie was missing an associated tag. By allowing a "0" or "1" to be generated within the **Hidden When** field of a Genie's properties, elements could simply be hidden if a required tag was missing, allowing the genie to still be pasted onto a graphics page.

The macro accepts three arguments: the first specifies the tag that requires confirmation, the second defines the outcome if the tag exists, the third defines the outcome if it does not exist. In the case of a genie being pasted on a graphics page, the IFDEF function would be configured as follows in the **Hidden When** field of the object properties dialog:

```
IFDEF("Bit_1",0,1)
```

If the tag "Bit_1" is defined in the tag database, the value in the **Hidden When** field will be 0. If Bit_1 is undefined, the value will be 1. Since the object is hidden when the value is TRUE (1), the object will be hidden when Bit_1 is undefined. See Hiding Graphics Objects for details.

Beyond this purpose, the IFDEF macro can be broadly used as a conditional variable. The [<value if defined>] and <value if not defined> arguments can support any variable, expression, or constant. The [<value if defined>] argument is optional; if you leave it blank it will generate the current variable. You can also use nested IFDEF macros.

Note: As different types of alarms can share the same name, you have to use a variation of IFDEF to check for the existence of alarm tags. See IFDEFAnaAlm for analog alarms, IFDEFDigAlm for digital alarms, or IFDEFAdvAlm for advanced alarms.

Syntax

```
IFDEF(TagName, [<value if defined>], <value if not defined>)
```

Return Value

If the tag specified in the first argument exists, the value defined by the second argument is returned. This could be a variable, expression, or constant, or the current tag value if the argument has been left blank. If the specified tag does not exist, the variable, expression, or constant defined by the third argument is returned.

Example

```
! Generate the tag value if tag "Bit_1" is defined  
! Generate an empty string if "Bit_1" is not defined  
IFDEF("Bit_1","", "")  
! Generate a zero value (0) if tag "Bit_1" is defined  
! Generate a true value (1) if "Bit_1" is not defined  
IFDEF("Bit_1",0,1)
```

For more examples of how to implement the IFDEF macro, see the CitectSCADA Knowledge Base article Q3461.

See Also

[IFDEFAnaAlm](#), [IFDEFDigAlm](#), [IFDEFAdvAlm](#), Hiding Graphics Objects, IFDEF macro

IFDEFAdvAlm

Based on the IFDEF macro, IFDEFAdvAlm allows you to define two possible outcomes based on whether or not a specified advanced alarm tag exists within a project at the time of compiling. The macro can be implemented anywhere a simple expression is used, including fields within relevant CitectSCADA dialogs.

The macro accepts three arguments: the first specifies the advanced alarm tag that requires confirmation, the second defines the outcome if the alarm exists, the third defines the outcome if it does not exist.

Note: As different types of alarms can share the same name, you have to use a variation of IFDEF to check for the existence of alarm tags. See [IFDEFAnaAlm](#) for analog alarms, or [IFDEFDigAlm](#) for digital alarms.

Syntax

```
IFDEFAdvAlm(TagName, [<value if defined>], <value if not defined>)
```

Return Value

If the advanced alarm tag specified in the first argument exists, the value defined by the second argument is returned. This could be a variable, expression, or constant, or the current tag value if the argument has been left blank. If the specified alarm does not exist, the variable, expression, or constant defined by the third argument is returned.

Example

```
! Generate tag value if advanced alarm "AdvAlarm_1" is defined
! Generate an empty string if "AdvAlarm_1" is not defined
IFDEFAdvAlm("AdvAlarm_1","", "")
! Generate a zero value (0) in Hidden When field if advanced alarm
"AdvAlarm_1" is defined
! Generate a true value (1) in Hidden When field if "AdvAlarm_1"
is not defined
IFDEFAdvAlm("AdvAlarm_1",0,1)
```

For more examples of how to implement the IFDEF macro, see the CitectSCADA Knowledge Base article Q3461.

See Also

[IFDEFAnaAlm](#), [IFDEFDigAlm](#), [IFDEF](#)

IFDEFAnaAlm

Based on the IFDEF macro, IFDEFAnaAlm allows you to define two possible outcomes based on whether or not a specified analog alarm tag exists within a project at the time of compiling. The macro can be implemented anywhere a simple expression is used, including fields within relevant CitectSCADA dialogs.

The macro accepts three arguments: the first specifies the analog alarm tag that requires confirmation, the second defines the outcome if the alarm exists, the third defines the outcome if it does not exist.

Note: As different types of alarms can share the same name, you have to use a variation of IFDEF to check for the existence of alarm tags. See [IFDEFDigAlm](#) for digital alarms, or [IFDEFAdvAlm](#) for advanced alarms.

Syntax

`IFDEFAnaAlm(TagName, [<value if defined>], <value if not defined>)`

Return Value

If the analog alarm tag specified in the first argument exists, the value defined by the second argument is returned. This could be a variable, expression, or constant, or the current tag value if the argument has been left blank. If the specified alarm does not exist, the variable, expression, or constant defined by the third argument is returned.

See Also

[IFDEF](#), [IFDEFDigAlm](#), [IFDEFAdvAlm](#)

Example

```
! Generate tag value if analog alarm "AnaAlarm_1" is defined
! Generate an empty string if "AnaAlarm_1" is not defined
IFDEFAnaAlm("AnaAlarm_1","", "")
! Generate a zero value (0) in Hidden When field if analog alarm
"AnaAlarm_1" is defined
! Generate a true value (1) in Hidden When field if "AnaAlarm_1"
is not defined
IFDEFAnaAlm("AnaAlarm_1",0,1)
```

For further examples of how to implement the IFDEF macro, see the CitectSCADA Knowledge Base article Q3461.

See Also

[IFDEF](#), [IFDEFDigAlm](#), [IFDEFAdvAlm](#)

IFDEFDigAlm

Based on the IFDEF macro, IFDEFDigAlm allows you to define two possible outcomes based on whether or not a specified digital alarm tag exists within a project at the time of compiling. The macro can be implemented anywhere a simple expression is used, including fields within relevant CitectSCADA dialogs.

The macro accepts three arguments: the first specifies the digital alarm tag that requires confirmation, the second defines the outcome if the alarm exists, the third defines the outcome if it does not exist.

Note: As different types of alarms can share the same name, you have to use a variation of IFDEF to check for the existence of alarm tags. See [IFDEFAnaAlm](#) for analog alarms or [IFDEFAdvAlm](#) for advanced alarms.

Syntax

```
IFDEFDigAlm(TagName, [<value if defined>], <value if not defined>)
```

Return Value

If the digital alarm tag specified in the first argument exists, the value defined by the second argument is returned. This could be a variable, expression, or constant, or the current tag value if the argument has been left blank. If the specified alarm does not exist, the variable, expression, or constant defined by the third argument is returned.

Example

```
! Generate tag value if digital alarm "DigAlarm_1" is defined
! Generate an empty string if "DigAlarm_1" is not defined
IFDEFDigAlm("DigAlarm_1","", "")
! Generate a zero value (0) in Hidden When field if digital alarm
"DigAlarm_1" is defined
! Generate a true value (1) in Hidden When field if "DigAlarm_1"
is not defined
IFDEFDigAlm("DigAlarm_1",0,1)
```

For more examples of how to implement the IFDEF macro, see the CitectSCADA Knowledge Base article Q3461.

Related macros

[IFDEFAnaAlm](#), [IFDEFAdvAlm](#), [IFDEF](#)

Macro Arguments

The Cicode macros use the following arguments.

- TagName
- [<value if defined>]
- <value if not defined>

TagName

The name of the tag you would like the IFDEF macro to confirm the existence of. The CitectSCADA compiler will check the current project database for a tag matching this name.

[<value if defined>]

Defines the outcome of the macro if the specified tag exists in the current project. This argument is optional, which means you can:

- Generate any variable, constant, or expression.
- Generate the current value for the specified tag by leaving the argument blank.

<value if not defined>

Defines the outcome of the macro if the specified tag does not exist in the current project. This will generate any variable, constant, or expression, including a blank string (" ") if you want nothing to be presented.

Chapter: 10 Converting and Formatting Cicode Variables

CitectSCADA provides four functions for converting integers and real numbers into strings, and vice versa.

- [IntToStr](#): converts an integer variable into a string
- [RealToStr](#): converts a floating-point variable into a string
- [StrToInt](#): converts a string into an integer variable
- [StrToReal](#): converts a string into a floating-point variable

You can convert data types without using these Cicode functions, but the result of the format conversion might not be what you expect. If you want more control over the conversion process, use the appropriate Cicode functions.

Note: Variables of type *object* cannot be converted to any other type.

When variables are automatically converted, or when the return value from a function call is converted, specific rules apply.

See Also

[Converting Variable Integers to Strings](#)

[Converting Real Numbers to Strings](#)

[Converting Strings to Integers](#)

[Converting Strings to Real Numbers](#)

[Formatting Text Strings](#)

[Escape Sequences \(String Formatting Commands\)](#)

[Using Cicode Files](#)

Converting Variable Integers to Strings

To convert an integer variable to a string:

```
IntVar=5;  
StringVar=IntVar;
```

The value of StringVar is set to "5".

The format of the string is specified when the variable is defined in the database. However you can override this default format with the string format (:) operator, and use the # format specifier to set a new format. For example:

```
IntVar=5;  
StringVar=IntVar:###
```

The value of StringVar = " 5 ". (The '#' formatting characters determine the size and number of decimal places contained in the string, that is a length of 4 with no decimal places.)

See Also

[Converting and Formatting Cicode Variables](#)

[Using Cicode Files](#)

Converting Real Numbers to Strings

To convert a real number variable to a string:

```
RealVar=5.2;  
StringVar=RealVar;
```

The value of StringVar is set to "5.2".

Note: Unpredictable results may occur if you use large numbers with a large number of decimal places.

The format of the string is specified when the variable is defined in the database. However you can override this default format with the string format (:) operator, and use the # format specifier to set a new format. For example:

```
StrTag1=RealTag1:#####.###
```

The value of StringVar = " 5.200 ". (The '#' formatting characters determine the size and number of decimal places contained in the string, that is a length of 10 including a decimal point and three decimal places.)

See Also

[Converting and Formatting Cicode Variables](#)

[Using Cicode Files](#)

Converting Strings to Integers

To convert a string variable to an integer:

```
StringVar="50.25";
IntVar=StringVar;
```

The value of IntVar is set to 50. If StringVar contains any characters other than numeric characters, IntVar is set to 0.

See Also

[Converting and Formatting Cicode Variables](#)

[Using Cicode Files](#)

Converting Strings to Real Numbers

To convert a string variable to a real number:

```
StringVar="50.25";
RealVar=StringVar;
```

The value of RealVar is set to 50.25. If StringVar contains any characters other than numeric characters, RealVar is set to 0.

See Also

[Converting and Formatting Cicode Variables](#)

[Using Cicode Files](#)

Formatting Text Strings

A string in Cicode is represented as text positioned between double quote (") delimiters. For example:

```
"This is my text string."
```

A string value can be assigned to a string variable. For example:

```
STRING sMyStringVariable;
sMyStringVariable = "This is my text string.;"
```

More than one string can be joined together (concatenated) using the Cicode 'plus' mathematical operator (+). For example:

```
STRING sMyStringVariable;
sMyStringVariable = "This is my text string." + "This is my second
text string.;"
```

The two strings would be joined together and assigned to the string variable `sMyStringVariable`. However, if subsequently displayed somehow, like in the following MESSAGE example, the concatenated string would look wrong because there is no space character positioned between the string sentences.

```
STRING sMyStringVariable;
sMyStringVariable = "This is my text string." + "This is my second
text string.;"
```

MESSAGE("String Concatenation Example",sMyStringVariable,32);



To overcome this potential formatting problem, you could include an extra space as the last character in the strings, or include the space as a third string in the concatenation. For example:

```
sMyStringVariable = "This is my text string. " + "This is my
second text string. ";
```

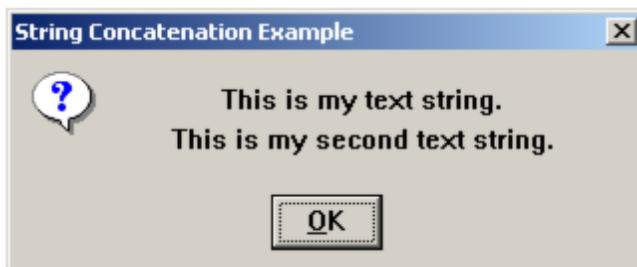
or

```
sMyStringVariable = "This is my text string." + " " + "This is my
second text string. ";
```

However, these are considered poor programming practices and not recommended. Instead, you can use special string formatting commands known as *escape sequences*.

If the two strings (as used in the previous example), were formatted using appropriate escape sequences positioned within the strings, and subsequently displayed somehow, like in the following MESSAGE example, the concatenated string would look different, For example:

```
STRING sMyStringVariable;
STRING s.NewLine = "\n";
sMyStringVariable = "This is my text string." + s.NewLine + "This
is my second text string.";
MESSAGE("String Concatenation Example",sMyStringVariable,32);
```



Strings and string variables can also be concatenated as in the previous example. Be aware of how the newline escape sequence (^n) was assigned to the string variable `s.NewLine`, and how this value was concatenated between the other strings and assigned to the string variable `sMyStringVariable` for display in the MESSAGE function.

See Also

[Converting and Formatting Cicode Variables](#)

[Using Cicode Files](#)

Escape Sequences (String Formatting Commands)

Cicode supports several escape sequences that you can use in text strings for custom formatting of the string. By using the appropriate Cicode escape sequences listed below you can format the string display to do such things as divide onto separate lines at specific positions, insert tab spaces, insert quotes, or to display Hexadecimal numbers.

Cicode escape sequences are preceded by a caret (^) character. The caret character is interpreted as a special instruction, and together with the characters immediately following it, are treated as an Cicode escape sequence formatting command. The escape sequences used in Cicode are:

<code>^b</code>	backspace
<code>^f</code>	form feed

<code>^n</code>	new line
<code>^t</code>	horizontal tab
<code>^v</code>	vertical tab
<code>^'</code>	single quote
<code>^"</code>	double quote
<code>^~</code>	caret
<code>^r</code>	carriage return
<code>^0xhh</code>	where <i>hh</i> is a hexadecimal number (for example, <code>^0x1A</code>)

See Also

[Converting and Formatting Cicode Variables](#)

[Using Cicode Files](#)

Chapter: 11 Working with Operators

With Cicode, you can use the data operators that are standard in a large number of programming languages: mathematical, bit, relational, and logical operators.

See Also

[Using Mathematical Operators](#)

[Using Bit Operators](#)

[Using Relational Operators](#)

[Using Logical Operators](#)

[Order of Precedence of Operators](#)

Using Mathematical Operators

Standard mathematical operators allow you to perform arithmetic calculations on numeric variables - integers and floating point numbers.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
MOD	Modulus (Remainder)

Example

The following are examples of mathematical operators

Com-	mand	PV12 = PV10 + PV11;
Com-		PV12 is the sum of PV10 and PV11

ment	
Com- mand	Counter = Counter - 1;
Com- ment	The value of Counter is decreased by 1
Com- mand	PV12 = Speed * Counter;
Com- ment	PV12 is the product of Speed and Counter
Com- mand	Average = Total / ShiftHrs;
Com- ment	Average is Total divided by ShiftHrs
Com- mand	Hold = PV12 MOD PV13;
Com- ment	If PV12 = 10 and PV13 = 8, Hold equals 2 (the remainder when PV12 is divided by PV13)
Com- mand	Hold = PV12 MOD PV13;
Com- ment	If PV12 = 10 and PV13 = 8, Hold equals 2 (the remainder when PV12 is divided by PV13)

Note: Cicode uses the standard order of precedence, that is multiplication and division are calculated before addition and subtraction. In the statement $A=1+4/2$, 4 is divided by 2 before it is added to 1, and the result is 3. In the statement $A=(1+4)/2$, 1 is first added to 4 before the division, and the result is 2.5.

You can also use the addition operator (+) to concatenate (join) two strings.

Operator	Description
+	Concatenate

Command	Message = "For info see " + "Supervisor";
Comment	Message now equals "For info see Supervisor"

For example:

See Also

[Working with Operators](#)

[Using Cicode Files](#)

Using Bit Operators

With a bit operator, you can compare the corresponding bits in two numeric expressions. (A bit is the smallest unit of data a computer can store.)

Operator	Description
BITAND	AND
BITOR	OR
BITXOR	Exclusive OR

For example

Command	Tag3 = Tag1 BITAND Tag2;
Command	Tag3 = Tag1 BITAND 0xFF;
Command	Tag3 = Tag1 BITOR Tag2;
Command	Tag3 = Tag1 BITXOR Tag2;

See Also

[Working with Operators](#)

[Using Cicode Files](#)

Using Relational Operators

Relational operators describe the relationship between two values. The relationship is expressed as one value being larger than, the same as, or smaller than another. You can use relational operators for both numeric and string variables, however you can only test variables of the same type. A numeric variable cannot be compared with a string variable.

Operator	Description
=	Is equal to
<>	Is not equal to
<	Is less than
>	Is greater than
<=	Is less than or equal to
>=	Is greater than or equal to

For example:

Command	IF Message = "Alarm Active" THEN ...
Expression	PV12 <> PV10;
Command	IF (Total + Count) / Avg < 10 THEN ...
Expression	Counter > 1;
Command	IF PV12 <= PV10 THEN ...
Expression	Total >= Shift * Hours;

See Also

[Working with Operators](#)

[Using Cicode Files](#)

Using Logical Operators

With logical operators, you can test several conditions as either TRUE or FALSE.

Operator	Description
AND	Logical AND
OR	Logical OR
NOT	Logical NOT

Examples:

Command	Result = (PV12 = 10 AND PV13 = 2);
Comment	If PV12 equals 10 and PV13 equals 2 then Result is TRUE(1)
Expression	Motor_1 AND Motor_2;
Comment	If both Motor_1 and Motor_2 are TRUE, that is Digital bits are 1 or ON, then the expression is TRUE
Expression	PV12 = 1 OR PV13 > 2 OR Counter <> 0;
Comment	If either PV12 equals 1 or PV13 is greater than 2 or Counter is not equal to 0, then the expression is TRUE
Command	Result = (Motor1_OI OR Motor2_OI);
Comment	If either Motor1_OI or Motor2_OI is TRUE, that is Digital bit is 1 or ON, then Result is TRUE (1)
Command	IF NOT PV12 = 10 THEN ...
Comment	If PV12 does not equal 10 then the result is TRUE. This is functionally identical to IF PV12 <> 10 THEN ...

Expression NOT Tag_1;

Comment This expression is TRUE if Tag_1 = 0. This is commonly used for testing digital variables

See Also

[Working with Operators](#)

[Using Cicode Files](#)

Order of Precedence of Operators

The table below shows the order of precedence of operators.

Operators have a set of rules that govern the order in which operations are performed. These rules are called the order of precedence. The precedence of Cicode operators from highest to lowest is:

1.	()
2.	NOT
3.	*, /, MOD
4.	:
5.	+, -
6.	>, <, <=, >=
7.	=, <>
8.	AND
9.	OR
10.	BITAND, BITOR, BITXOR

See Also

[Working with Operators](#)

[Using Cicode Files](#)

Chapter: 12 Working with Conditional Executors

The statements that control decisions and loops in your functions are called conditional executors. Cicode uses four conditional executors: If, For, While, and select case.

See Also

[Formatting Executable Statements](#)

[Setting IF ... THEN Conditions](#)

[Using FOR ... DO Loops](#)

[Using WHILE ... DO Conditional Loops](#)

[Using the SELECT CASE statement](#)

[Using Cicode Files](#)

Setting IF ... THEN Conditions

The IF statement executes one or more statements based on the result of an expression. You can use If in one of two formats: If Then and If Then Else.

```
If Expression Then
    Statement(s);
END
-or-
If Expression Then
    Statement(s);
Else
    Statement(s);
END
```

When you use the **If Then** format, the statement(s) following are executed only if the expression is TRUE, for example:

```
INT Counter;
IF PV12 = 10 THEN
    Counter = Counter + 1;
END
```

In this example, the Counter increments only if the tag PV12 is equal to 10, otherwise the value of Counter remains unchanged. You can include several statements (including other IF statements), within an IF statement, for example:

```
INT Counter;
IF PV12 = 10 THEN
    Counter = Counter + 1;
    IF Counter > 100 THEN
        Report("Shift");
    END
END
```

In this example, the report runs when the Counter increments, that is when PV12 = 10, and the value of the counter exceeds 100.

You can use the **If Then Else** format for branching. Depending on the outcome of the expression, one of two actions are performed, for example:

```
INT Counter;
IF PV12 = 10 THEN
    Report("Shift");
ELSE
    Counter = Counter + 1;
END
```

In this example, the report runs if PV12 is equal to 10 (TRUE), or the counter increments if PV12 is anything but 10 (FALSE).

See Also

[Working with Conditional Executors](#)

Using FOR ... DO Loops

A For loop executes a statement or statements a specified number of times.

```
FOR Variable=Expression To Expression DO
    Statement(s);
END
```

The following function uses a For loop:

```
STRING ArrayA[5] = "This", "is", "a", "String", "Array";
INT
FUNCTION
DisplayArray()
    INT Counter;
    FOR Counter = 0 TO 4 DO
        Prompt(ArrayA[Counter]);
        Sleep(15);
    END
END
```

This function displays the single message "This is a String Array" on the screen one word at a time pausing for 15 seconds between each word.

See Also

[Working with Conditional Executors](#)

Using WHILE ... DO Conditional Loops

A While loop executes a statement or statements in a loop as long as a given condition is true.

```
WHILE Expression DO  
    Statement(s);  
END
```

The following code fragment uses a WHILE loop:

```
INT Counter;  
WHILE DevNext(hDev) DO  
    Counter = Counter + 1;  
END  
/* Count the number of records in the device (hDev) */
```

Be careful when using WHILE loops in your Cicode functions: WHILE loops can cause excessive loading of the CPU and therefore reduce system performance. If you use a WHILE loop to loop forever, you should call the Cicode function `Sleep()` so that Citect-SCADA can schedule other tasks. The `Sleep()` function increases the performance of your CitectSCADA system if you use many WHILE loops.

See Also

[Working with Conditional Executors](#)

Nested Loops

You can "nest" one loop inside the other. That is, a conditional statement can be placed completely within (nested inside) a condition of another statement.

See Also

[Working with Conditional Executors](#)

Using the SELECT CASE statement

The select case statement executes on several groups of statements, depending on the result of an expression. SELECT CASE statements are a more efficient way of writing code that would otherwise have to be done with nested IF THEN statements.

```
SELECT CASE Expression
CASE CaseExpression1,CaseExpression2
    Statement(s);
CASE CaseExpression3 TO CaseExpression4
    Statement(s);
CASE IS >CaseExpression5,IS<CaseExpression6
    Statement(s);
CASE ELSE
    Statement(s);
END SELECT
```

Where **CaseExpressionn** is any one of the following forms:

- expression
- expression TO expression

Where the TO keyword specifies an inclusive range of values. The smaller value needs to be placed before TO.

- **IS <relop> expression.**

Use the IS keyword with relational operators (**<relop>**). Relational operators that may be used are **<, <=, =, >, >=**.

If the Expression matches any CaseExpression, the statements following that CASE clause are executed up to the next CASE clause, or (for the last clause) up to the END SELECT. If the Expression matches a CaseExpression in more than one CASE clause, only the statements following the first match are executed.

The CASE ELSE clause is used to indicate the statements to be executed if no match is found between the Expression and any of the CaseExpressions. When there is no CASE ELSE statement and no CaseExpressions match the Expression, execution continues at the next Cicode statement following END SELECT.

You can use multiple expressions or ranges in each CASE clause. For example, the following line is valid:

```
CASE 1 To 4, 7 To 9, 11, 13, Is > MaxNumber
```

You can also specify ranges and multiple expressions. In the following example, CASE matches strings that are exactly equal to "everything", strings that fall between "nuts" and "soup" in alphabetical order, and the current value of "TestItem":

```
CASE "everything","nuts" To "soup",TestItem
```

SELECT CASE statements can be nested. Each SELECT CASE statement needs to have a matching END SELECT statement.

For example, if the four possible states of a ship are Waiting, Berthed, Loading, and Loaded, the Select Case statement could be run from a button to display a prompt detailing the ship's current state.

```
select case iStatus
CASE    1
    Prompt("Waiting");
CASE    2
    Prompt("Berthed");
CASE    3
    Prompt("Loading");
CASE    4
    Prompt("Loaded");
CASE Else
    Prompt("No Status");
END SELECT
```

See Also

[Working with Conditional Executors](#)

Chapter: 13 Performing Advanced Tasks

This section introduces and explains event handling, CitectSCADA tasks, CitectSCADA threads, how CitectSCADA executes, and multitasking - including foreground and background tasks, controlling tasks, and pre-emptive multitasking.

See Also

[Handling Events](#)

[How CitectSCADA Executes](#)

[Multitasking](#)

[Foreground and background tasks](#)

[Controlling tasks](#)

[Pre-emptive multitasking](#)

Handling Events

Cicode supports event handling. You can define a function that is called only when a particular event occurs. Event handling reduces the overhead that is required when event trapping is executed by using a loop. The following example illustrates the use of the `OnEvent()` function:

```
INT
FUNCTION MouseCallback()
    INT x, y;
    DspGetMouse(x,y);
    Prompt("Mouse at "+x:"####+", "y:####");
    RETURN 0;
END
OnEvent(0,MouseCallback);
```

The function `MouseCallBack` is called when the mouse is moved - there is no need to poll the mouse to check if it has moved. CitectSCADA watches for an event with the `OnEvent()` function.

Because these functions are called each time the event occurs, you should avoid complex or time consuming statements within the function. If the function is executing when another call is made, the function can be blocked, and some valuable information may be lost. If you do wish to write complex event handling functions, you should use the queue handling functions provided with Cicode.

See Also

[Performing Advanced Tasks](#)

How Cicode is Executed

Your multi-tasking operating system gives CitectSCADA access to the CPU through threads. However, this access time is not continuous, as CitectSCADA needs to share the CPU with other applications and services.

Note: Be careful when running other applications at the same time as CitectSCADA. Some applications place high demands on the CPU and reduce the execution speed of CitectSCADA.

The CitectSCADA process has many operations to perform, including I/O processing, alarm processing, display management, and Cicode execution - operations that are performed continuously. And, because CitectSCADA is a real-time system, it needs to perform the necessary tasks within a minimum time - at the expense of others. For this reason, CitectSCADA is designed to be multitasking, so it can efficiently manage its own tasks.

CitectSCADA performs its tasks in a specific order in a continuous loop (cycle). CitectSCADA's internal tasks are scheduled at a higher priority than that of Cicode and have access to the CPU before the Cicode. For example, the Alarms, Trends, and I/O Server tasks all get the CPU before any of your Cicode tasks. The reports are scheduled at the same priority as your Cicode. CitectSCADA background spoolers and other idle tasks are lower priority than your Cicode.

For Cicode, which consists of many tasks, CitectSCADA uses round-robin single priority scheduling. With this type of scheduling each task has the same priority. When two or more Cicode tasks exist, they each get a CPU turn in sequence. This is a simple method of CPU scheduling.

Note: If a Cicode task takes longer than its designated CPU time to execute, it is preempted until the next cycle - continuing from where it left off.

See Also

[Performing Advanced Tasks](#)

Multitasking

Multitasking is when you can run more than one task at the same time. Windows supports this feature at the application level. For example you can run MS-Word and MS-Excel at the same time.

CitectSCADA also supports multitasking internally; that is you can tell CitectSCADA to do something, and before CitectSCADA has completed that task you can tell CitectSCADA to start some other task. CitectSCADA will perform both tasks at the same time. CitectSCADA automatically creates the tasks, leaving you to call the functions.

Multitasking is a feature of CitectSCADA not the operating system. Many applications cannot do this, for example if you start a macro in Excel, while that macro is running you cannot do any other operation in Excel until that macro completes.

A multitasking environment is useful when designing your Cicode. It allows you to be flexible, allowing the operator to perform one action, while another is already taking place. For example, you can use Cicode to display two different input forms at the same time, while allowing the operator to continue using the screen in the background.

See Also

[Performing Advanced Tasks](#)

Foreground and background tasks

Cicode tasks (or threads) can be executing in either foreground or background mode. A foreground task is one that displays and controls animations on your graphics pages. Any expression (not a command) entered in a property field (that is Text, Rectangle, Button, etc.) is executed as a foreground task. Any other commands and expressions are executed in background mode.

The difference between a background and foreground task is that a background task can be pre-empted. That is, if system resources are limited, the task (for example, the printing of a report) can pause to allow a higher priority task to be executed. When the task is completed (or when system resources become available) the original task resumes. Foreground tasks are the highest priority and can not be pre-empted.

See Also

[Performing Advanced Tasks](#)

Controlling tasks

You can use the Task functions to control the execution of Cicode tasks, and use the CitectSCADA Kernel at runtime to monitor the tasks that are executing. Since CitectSCADA automatically creates new tasks (whenever you call a keyboard command, etc.), schedules them, and destroys them when they are finished, users rarely need to consider these activities in detail.

Sometimes it is desirable to manually 'spawn' a new task. For example, suppose your Cicode is polling an I/O Device (an operation which need to be continuous), but a situation arises that requires operator input. To display a form would temporarily halt the polling. Instead you can spawn a new task to get the operator input, while the original task continues polling the device.

Note: The TaskNew Cicode function is used to spawn new tasks.

See Also

"Using the CitectSCADA Kernel" in the CitectSCADA User Guide

[Performing Advanced Tasks](#)

[Task Functions](#)

Pre-emptive multitasking

Cicode supports pre-empted multitasking. If a Cicode task is running, and a higher priority task is scheduled, CitectSCADA will suspend the original task, complete the higher priority task and return to the original task.

Preemption is supported between Cicode threads and other internal processes performed by CitectSCADA. You can, therefore, write Cicode that runs forever (for example, a continuous while loop) without halting other Cicode threads or CitectSCADA itself. For example:

```
INT FUNCTION MyLoopFunction()
    WHILE TRUE DO
        // Whatever is required in the continuous loop
        Sleep(1); // Optional
    END
END
```

In the above example, the function Sleep() is used to force preemption. The Sleep() function is optional, however it will reduce the load on the CPU, because the loop is suspended each second (it will not repeat at a high rate).

See Also

[Performing Advanced Tasks](#)

Chapter: 14 Editing and Debugging Code

This section describes how to edit and debug your Cicode using the Cicode Editor.

The Cicode Editor

You use the Cicode Editor to write, edit, and debug your Cicode code. The Cicode Editor behaves similarly to other code editing tools like Microsoft Dev Studio, and contains many advanced editing features such as:

- Dockable windows and toolbars.
- Syntax highlighting - color highlighting of syntax functions.
- IntelliSense AutoPrompt - function definition tooltips.
- IntelliSense AutoComplete - automatic inline prompting and completion of functions with their parameters.
- AutoCaseCorrect - automatic case correction of function keywords.
- AutoIndent - automatic indent alignment of code.
- AutoScroll - automatic mouse middle button support.
- Drag and Drop - copy or move of selected text.
- Bookmark and Breakpoint indicator bar - single click set and reset of bookmarks and breakpoints.
- Keyboard Shortcuts support.

Cicode Editor starts automatically when you double-click a Cicode file object in Citect Explorer, or click the Cicode Editor button in Citect Explorer. See the topic [Starting the Cicode Editor](#).

Cicode files are stored as text files. For more information see the [Introducing Cicode](#) and the section [Using Cicode Files](#).

Note: Be careful not to confuse a Cicode file (*.ci) with an **Include file** (*.cii).

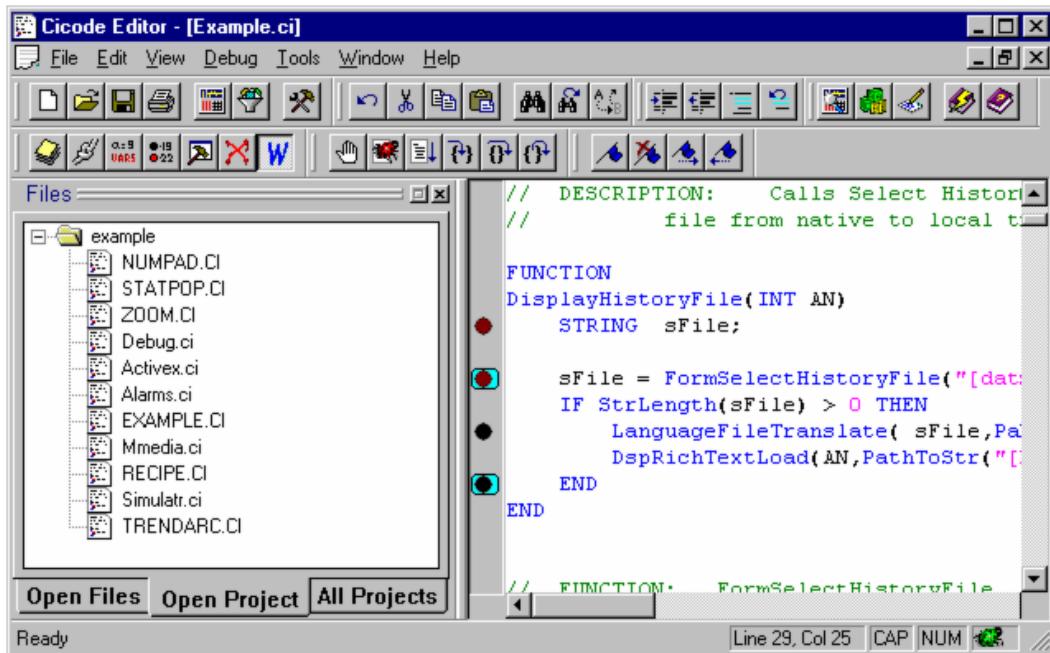
You could use any text editor to view or edit the Cicode files, however, the Cicode Editor provides integrated views specific to Cicode. As well as the features listed above, it includes:

- Breakpoint window
- Output window,
- Global Variable Window
- Stack window
- Thread window
- Compile Errors window
- CitectVBA Watch window
- Files window

To minimize potential future problems with maintaining your Cicode files, you should adopt a programming standard as early as possible, as discussed in the section [Using Cicode Programming Standards](#). Maintain structured Cicode files, by logically grouping your Cicode functions within the files, and by choosing helpful descriptive names.

Modular programming methods are discussed in the section [Modular Programming](#).

Cicode functions are introduced in the section titled [Using Cicode Functions](#). Suggestions for debugging your Cicode is included in the section titled [Debugging Cicode](#).



Starting the Cicode Editor

To start the Cicode Editor:

1. Click the Citect Explorer button.
2. Open the Cicode Files folder in the project list area of your project.

3. Do either of the following:
 - Double click a Cicode file (*.ci).
 - Choose **Tools | Cicode Editor** in a CitectSCADA application.
 - Click the **Cicode Editor** button.

Changing the default Cicode Editor

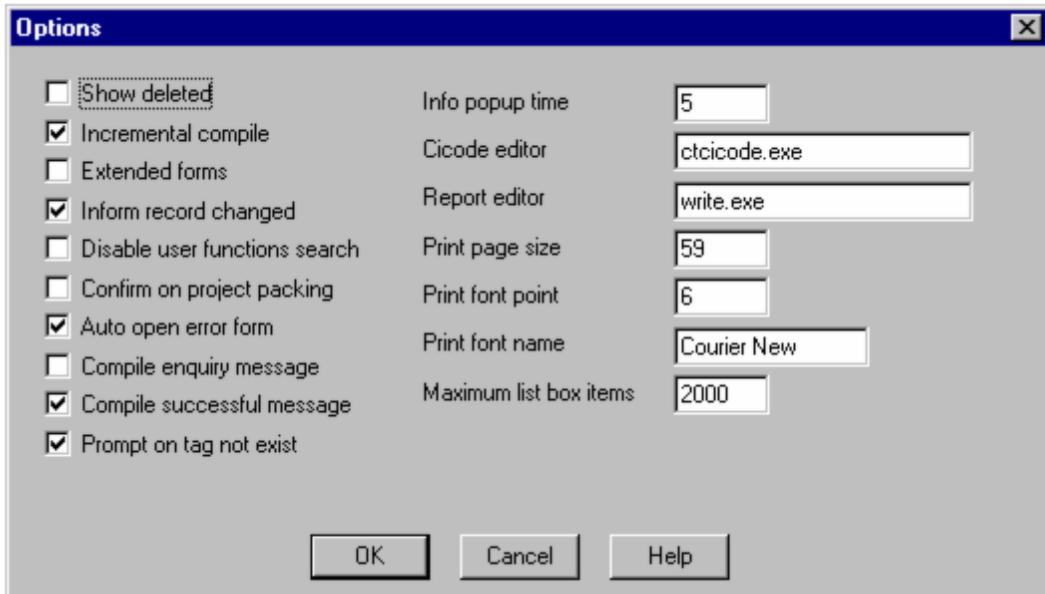
CitectSCADA allows you to use any text editor supported by Windows (for example, ED for Windows, Windows Notepad, or Microsoft Word), instead of the default Cicode Editor.

To change the default Cicode Editor:

1. Click the Project Editor button.
2. Choose **Tools | Options**.
3. Enter the editor application file name in the **Cicode Editor** field.

Note: The application name of the default Cicode Editor is `ctcicode.exe` located in the CitectSCADA`bin` folder. The application name for Notepad is `notepad.exe`, located in the Microsoft Windows `c:\windows\` folder. The relative path to the editor application need to be included if the application is not stored in the CitectSCADA`bin` folder.

4. Click **OK** to save the changes and close the form, or **Cancel** to abort changes without saving.



Creating Cicode files

To create a new Cicode file:

1. Start the Cicode Editor.
2. Choose **File** | **New**, or click **New**.

Save the Cicode file after creating it. The file is only stored on disk after you save it.

Creating functions

To create a new Cicode function:

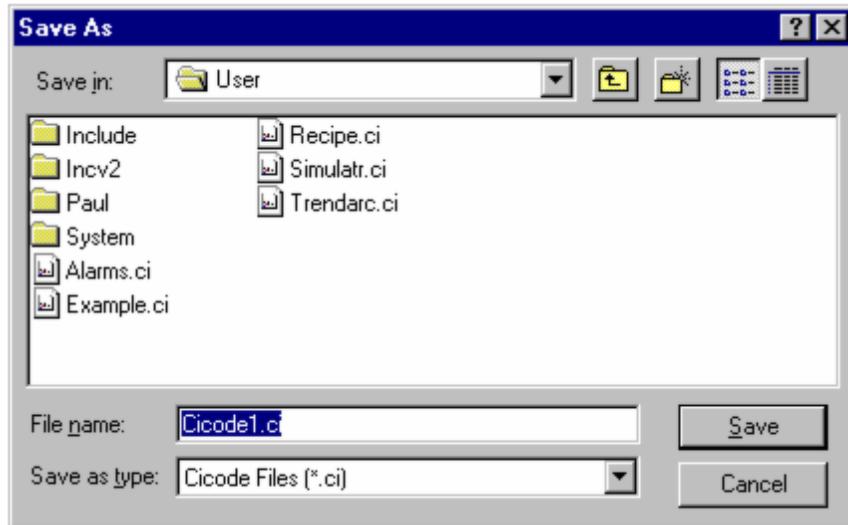
1. Start the Cicode Editor.
2. Choose **File** | **New**, or click **New**.
3. Type in your new Cicode function in the blank space, or at the end of the file. Format the Cicode function correctly, following the documented syntax.
4. Save the Cicode file.

Saving files

To save a Cicode file:

1. Choose **File** | **Save**, or click **Save**.
2. If the file is new, you will be prompted by the Save as dialog. CitectSCADA automatically suggests a name.
3. Type in a new name in the **File name** field.
4. Click **Save** to save the file, or **Cancel** to abort the save.

To save your Cicode file under a new name, choose **Save as** instead of **Save**. The original file will remain in your project under the original filename, until you delete it. All source files in your project directory will be included when you compile your project.

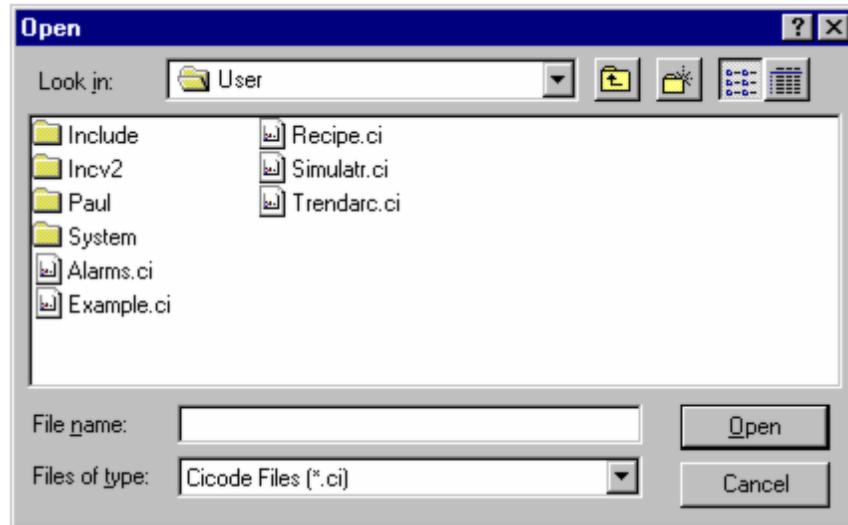


Opening Cicode files

To open a Cicode file:

1. Start the Cicode Editor.
2. Choose **File | Open**, or click **Open**.
3. Select a file from the list. You can use the dialog controls to open other projects and directories.
4. Click the **Open** button to open the file, or **Cancel** to abort.

Note: Double clicking on any Cicode file (*.ci) in the Citect Explorer will launch the Cicode Editor and open the Cicode file. However, be careful not to confuse a



Deleting Cicode files

To delete a Cicode file:

1. Run the Cicode Editor.
2. Choose **File | Open**, or click the Open button.
3. Select the target file from the list. You can use the dialog controls to open other projects and directories.
4. Press the **Delete** key.
5. Click the **Yes** button to confirm delete, or **No** to abort.
6. Click the **Cancel** button to close the Open form.

Finding text in Cicode files

To find text in a Cicode file:

1. Choose **Edit | Find**, or click the **Find** button.
2. Complete the Find dialog, filling in the **Find what** field.
3. Click the **Find Next** button to begin searching, or **Cancel** to abort. The search is performed down the file from the cursor. Hits are highlighted.

Compiling Cicode files

To compile Cicode:

1. Run the Cicode Editor.
2. Choose **File** | **Compile**, or click the Compile button.

Note: You cannot compile Cicode functions individually. When you compile Citect-SCADA, it automatically compiles the entire contents of the project.

Viewing errors detected by the Cicode Compiler

To view errors detected by the Cicode compiler:

Do either of the following:

- From the **Compile Errors** in the **File** menu of the Project Editor, click **Goto**. This launches the Cicode Editor and opens the appropriate file at the correct line.
- Choose **View** | **Compile Errors**, then double-click the compile error you want to view.

Cicode Editor Options

Cicode error handling behavior is controlled through the Cicode Editor Options Properties Dialog. These allow you to set (and change) what should happen when errors occur in running Cicode, under which circumstances the debugger should be started, and how the debugger behaves when in debug mode.

There are three tabbed property pages of options within the Debugger Options Properties dialog:

- View Windows and ToolBars tab
- Options Properties tab
- Language Formatter Properties tab

See Also

[Debugging Cicode](#)

Docking the Windows and Toolbars

The view windows and toolbars of the Cicode Editor can be docked or free floating within the editing and debugging environment.

Toolbars are docked by default within the toolbar area at the top of the Cicode Editor. Windows are docked by default in the document display area at the lower portion of the Cicode Editor, beneath the toolbar area.

Docked windows are those that resize themselves to fit totally within the Cicode Editor display area, by docking (attaching) themselves to an internal edge of the display area. Docked windows cannot be resized manually, and will share the display space with the Editor toolbars and other docked windows. Docked windows and toolbars share the display space side-by-side, without obscuring the view of each other.

Free floating windows are those that are not docked to the editor, nor are necessarily constrained by the editor boundaries. Free floating windows can be resized manually, and are subject to layering (Z-order), in which they can be partly or wholly obscured by another window, and they could partly or wholly obscure the view of another window themselves. The window or toolbar with the current focus, is the one completely visible at the top of all other display window layers, partly or wholly obscuring any beneath it in the Z-order.

Windows and toolbars can be moved about in the Cicode Editor environment by clicking and dragging the titlebar of a window, or non-button area of a button bar. Docking behaviour is by default, and can be overridden by holding down the CTRL key during the drag-and-drop to force the window or bar to be free floating.

The position of the mouse during the drop action determines which side the window or toolbar docks to. Docking outlines of the window or toolbar are displayed with gray lines during the drag action to indicate the potential docked position.

[Debugging Cicode](#)

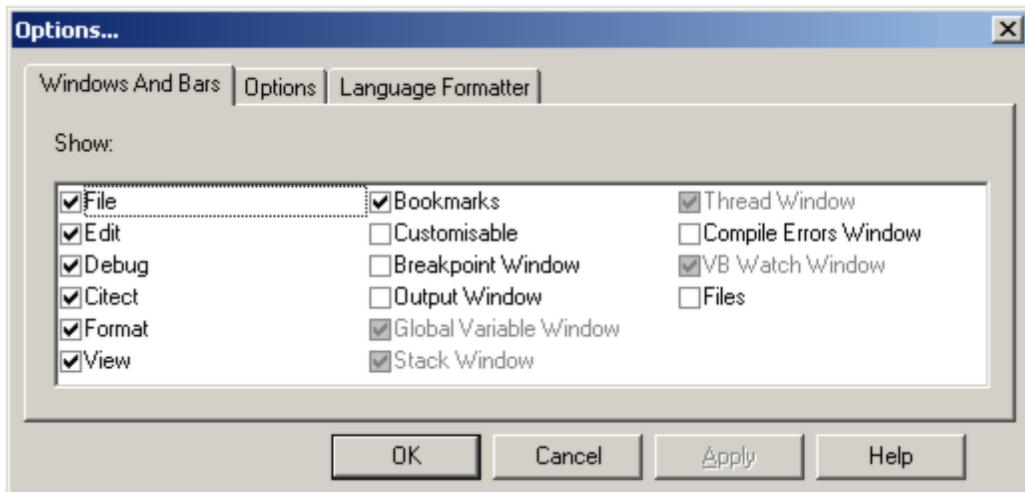
Displaying the Editor Options Properties dialog

To view/hide the Editor Options properties dialog:

1. Run the Cicode Editor.
2. Choose **Debug | Options**, or press Ctrl + T and then select the appropriate Window from the dialog.

Note: You can also choose **View | Options**, and then select the appropriate Window from the dialog.

Windows and Bars Tab



The Windows and Bars tab displays the current display state of the listed Toolbars and Debug Windows within the Cicode Editor. A check mark in the checkbox next to the Window or Toolbar name enables the display of that Window or Toolbar in the Cicode Editor. A grayed-out checkbox indicates that the window is disabled (presently unable to be displayed). For example: Many of the debug windows which display the active state of project Cicode variables are disabled when a Cicode project is not running, and therefore the Cicode Editor cannot be in debug mode).

Note: Right-click in the toolbar area to view a menu of available toolbars and debug windows. For a description the buttons, see [The Cicode Editor](#).

Toolbar options

Click the button on the toolbar to display the tool bar you want; for example, click Edit to display the Edit tool bar.

Window options

The Cicode Editor has several editing and debug windows that you can use to display information about running Cicode and CitectVBA.

The Cicode Editor windows available are:

- Breakpoint window
- Output window
- Global Variable window
- Stack window
- Thread window
- Compile Errors window
- CitectVBA Watch window
- Files window

Viewing Editor windows

You can choose to view Editor windows or hide them to give you more room on your screen.

To view/hide an Editor Window:

1. Run the Cicode Editor.
2. From the **View** menu, select the appropriate Window, or click the toggle button you want in the View toolbar.

Breakpoint window

Displays the Breakpoint Window, which is used to list all breakpoints that are currently set within the project. Double clicking an item in the list loads the file into the editor and jumps to the breakpoint position. Right-clicking an item allows the enable/disable/removal of the list item.

Breakpoint Window		
File	Line	Enabled
L:\Citect542125\User\Example\Activex.ci	18	Yes

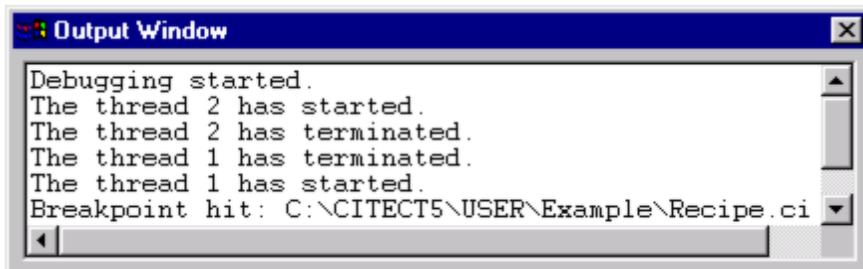
The Breakpoint Window has the following fields:

- **File:** the full name and location of the code file in which the breakpoint exists.
- **Line:** the line number (in the code file) where the breakpoint is located.
- **Enabled:** indicates if the breakpoint is enabled or not. **Yes** indicates it is active, **No** indicates it is not.

Output window

Displays the Output Window, which lists the output messages sent by CitectSCADA during debugging. It states when threads start and terminate, and if a break occurs. This window will show messages sent by the `TraceMsg()` function.

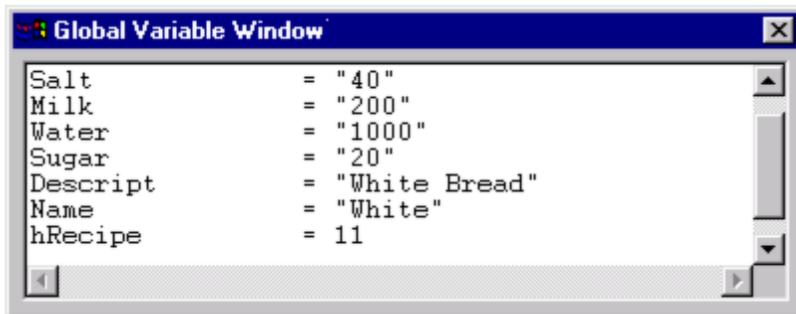
The Output window shows entries in the order that they occur:



Note: you need to be in debug mode to view the messages.

Global Variable Window

Displays the Global Variables Window, which lists the names and values of all global variables processed to date in the running project during debugging. A global variable is added to the list when it is first assigned a value. Each time the Global variable is processed, its value will be updated in the Global Variable Window.



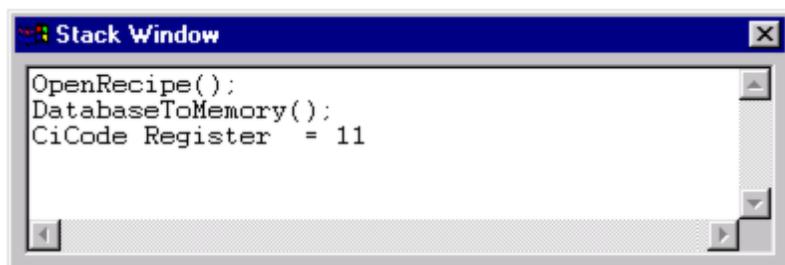
Note: you need to be in debug mode to view global variable values in this window.

Stack window

Displays the Call Stack window, which lists the stack values of the current thread. The stack consists of the functions called (including the arguments), any variables used in the functions, and return values. This is especially useful during debugging to trace the origin of the calling procedures.

A stack is a section of memory that is used to store temporary information. For example, when you call a Cicode function, the variables used inside the function exist only as long as the function runs.

To view the values of arguments and variables in a procedure, place a breakpoint within the procedure under watch. When that breakpoint is reached, the Stack Window will display the current call stack of the procedure containing the breakpoint. The values of the stack are updated as the values change.



Note: you need to be in debug mode to view this window.

Thread window

Displays the Threads History window.

A screenshot of a Windows-style application window titled "Thread Window". The window contains a table with the following data:

Name	HND	CPU	State	CPU_Time
PID	15	00	Ready	00:00:00.000
PID	16	00	Ready	00:00:00.111
PID	17	00	Ready	00:00:00.020
WaterTankLevel	18	00	Sleep	00:00:00.000
Always	19	00	Sleep	00:00:00.000

The Thread Window has the following fields:

- **Name:** The name of the Cicode thread. This is the name of the function called to start the thread (from the TaskNew() function for example).

If you click on the Name of the Cicode thread, you will make the selected thread the current focus of the Debugger. The Debugger will change the display to show the source of the new thread.

Note: If the thread was not started from TaskNew(), the Name shown will be **Command**.

- **Hnd:** The Cicode thread handle.
- **CPU:** The amount of CPU the Cicode thread is currently using, as a percentage of the total CPU usage. Cicode is efficient and this value should be quite small (0-25%). If this value is large it can indicate a problem with the Cicode program you have created. For example, values over 60% can indicate that your thread is running in 'hard' loops, and needs a Sleep() function inserted.
- **State:** The state of the Cicode thread. The states are defined as follows:
 - **Ready:** The Cicode is ready to be run.
 - **Sleep:** Suspended using the Sleep() function.
 - **Run:** The thread is running.
- **CPU_Time:** The total amount of CPU time that the Cicode thread has consumed. This tracks how much CPU time the thread has used over its lifetime.

Note: you need to be in debug mode to view this window.

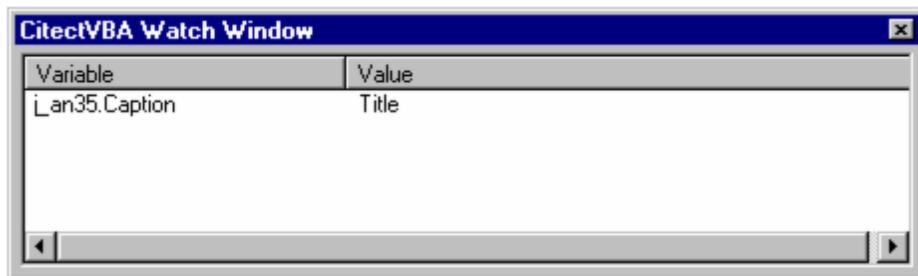
Compile Errors window

Displays the Compile Errors window, which lists any code errors that have occurred during compile. You can double-click on the file name in the list, to open that code file in the Cicode Editor, and jump to the line of code that caused the compile error.

File	Line	Error	Context
L:\Citect54212...	12	error...	FALSE {OBJECTz} CallStack;
L:\Citect54212...	12	error...	OBJECTz (CallStack); OBJECT

CitectVBA Watch window

Displays the CitectVBA Watch window. During debugging mode, you can use the CitectVBA Watch window to watch the value of any CitectVBA variables in the current scope. Click in the **Variable** column and type in the name of the variable under watch. As it comes into scope, its value is updated and appears in the **Value** column.

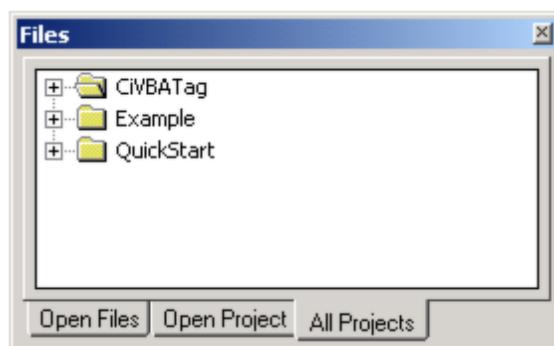


Note: You need to be in debug mode to view this window.

Files window

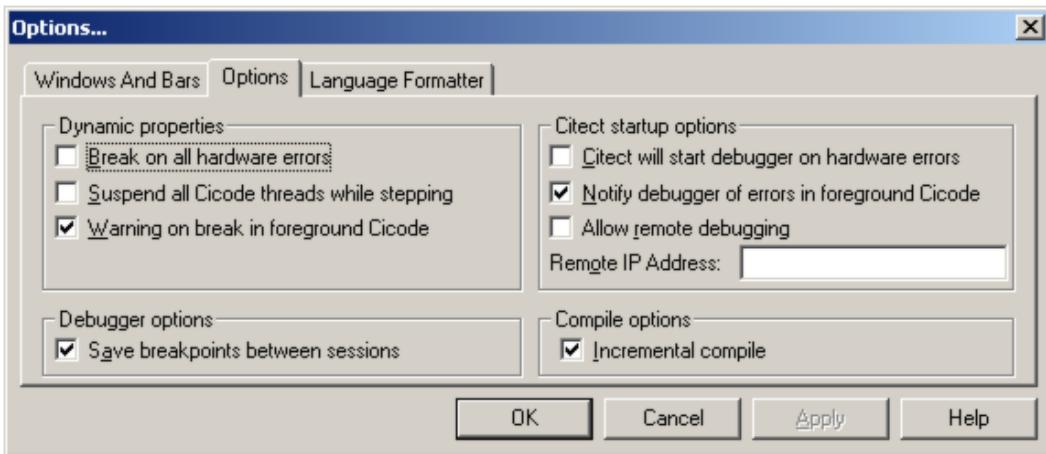
Displays the Files window containing three tabs.

- The 'All Projects' tab displays a tree hierarchy view of all projects and their Cicode and CitectVBA files available within Citect Explorer.
- The 'Open Project' tab displays a tree hierarchy view of the currently selected project, and all included projects. The currently selected project will be the top entry.
- The 'Open Files' tab lists the names of all files currently open for editing in the Cicode Editor.



Note: Clicking any of the file names displayed in the tree will open that file in the editor and give it the focus.

Options Properties Tab



The Options properties tab has the following features:

[Dynamic properties] Break on all hardware errors

Stops a Cicode thread if a hardware error is detected. A Cicode error will be generated and the thread will terminate (without executing the rest of the function).

[Dynamic properties] Suspend all Cicode threads while stepping

All Cicode threads will be suspended while the debugger is stepping (or when the debugger reaches a breakpoint, or the user performs a manual break). If you try to run any Cicode thread at such a time (by pressing a button at runtime, and so on), the **Command paused while in debug mode** message will display in the runtime prompt line.

This option allows better isolation of any software errors that are detected, especially those that occur when your Cicode thread interacts with other threads. Foreground Cicode cannot be suspended and will continue running when this option is set.

Note: This option will help prevent all new Cicode threads from running (including keyboard and touch commands), and should not be used on a running plant.

[Dynamic properties] Warning on break in foreground Cicode

If a break point is 'hit' in a foreground Cicode task, the **Foreground Cicode cannot break (343)** error message is generated, and will be displayed on the Hardware Alarm page. Disable this option to stop the alarm message from displaying.

[CitectSCADA startup options] CitectSCADA will start debugger on hardware errors

CitectSCADA will automatically start the debugger when a Cicode generated hardware error is detected. The debugger will display the Cicode source file, and mark the location of the error.

Note: This option will interrupt normal runtime operation, and should only be used during testing and commissioning of systems.

WARNING

UNINTENDED EQUIPMENT OPERATION

Do not use the following options during normal plant or process operations:

- Suspend all Cicode threads while stepping.
- CitectSCADA will start debugger on hardware errors.

These options are only for use during system testing and commissioning.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

[CitectSCADA startup options] Notify debugger of errors in foreground Cicode

CitectSCADA will automatically start the debugger if an error is detected in a foreground task. The debugger will display the Cicode source file, and mark the location of the error.

This option is overridden by the **CitectSCADA will start debugger on hardware errors** option. That is, if the above option is disabled, then this option is disabled also.

Note: Foreground Cicode cannot be suspended. The break point will be marked, but you will not be able to step through the function.

[CitectSCADA startup options] Allow remote debugging

Allows debugging of Cicode on this computer from a remote CitectSCADA computer.

[CitectSCADA startup options] Remote IP Address

The Windows Computer Name or TCP/IP address of the remote CitectSCADA computer.

The Windows Computer Name is the same as specified in the Identification tab, under the Network section of the Windows Control Panel. You specify this name on the computer from which you are debugging.

The TCP/IP address (for example, 10.5.6.7 or plant.yourdomain.com) can be determined as follows:

- Go to the Command Prompt, type **IPCONFIG**, and press **Enter**.

[Debugger options] Save breakpoints between sessions

Save the location and states of breakpoints between running sessions of the Cicode Editor and Debugger. This means breakpoints inserted using the Cicode Editor can later be recalled when an error is detected - even though the file (and application) has been closed.

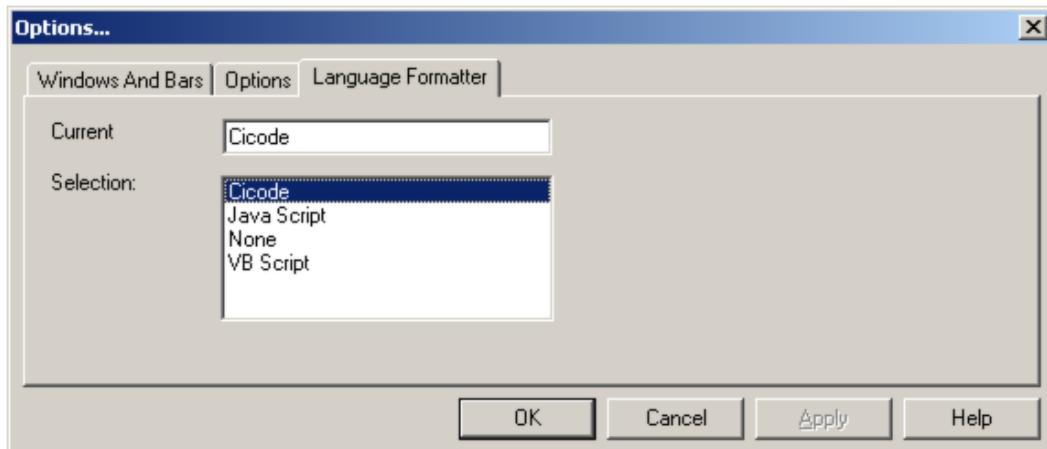
[Compile options] Incremental compile

Enables the incremental compilation of the project.

See Also

[Debugging Cicode](#)

Language Formatter Properties Tab



This dialog displays the currently selected programming language that the editor will use to format the syntax of the file being edited in the code window. If you open a Cicode file (with a .Ci extension), the current language formatter changes to Cicode. If you open a CitectVBA file (with a .bas extension), the current language formatter changes to CitectVBA.

Similarly, if you open a file with neither a Cicode nor a CitectVBA extension, say a text file (with a .txt extension), the editor will not know which language type you want to use, and will not apply any formatting to the file. You can use this dialog to select which programming language the file contains, and it will format the file appropriately for display in the code window.

Note: The Cicode Editor can be used to edit any ASCII text based file, including Microsoft JScript. The editor recognizes JScript files (with a .jav extension) and will change the current language formatter to JScript. CitectSCADA does not support JScript, and will not compile it into your project. However, the editor can still be used

separately to edit or create a JScript file or any other ASCII text based file.

Current

Displays the currently selected programming language formatter for appropriate syntax coloring of the file displayed in the code window.

Selection

Displays the range of possible programming languages that can be chosen as the current language for formatting and display in the code window.

Debugging Cicode

To help you locate Cicode errors, you can switch the Cicode Editor to debug mode to analyze running Cicode. You can toggle debugging on and off as required, but Citect-SCADA needs to be running for the debugger to work.

Note: The Cicode Editor cannot debug foreground Cicode. A break in a foreground Cicode will result in the **Foreground Cicode cannot break** message.

See Also

[Cicode Editor Options](#) | [Function Error handling](#) | [Debug Error Trapping](#)

Using debug mode

To switch to debug mode:

1. Run the Cicode Editor.
2. Choose **Debug | Start Debugging**, or click the Toggle Debug button.

Note: If the current project is not running, CitectSCADA compiles and runs it automatically. The bug in the bottom right-hand corner is green when debugging.

Debugging functions

To debug a function:

1. Run the Cicode Editor.
2. Open the file containing the function you wish to debug.
3. Click the Toggle Debug button, or choose **Debug | Start Debugging**.

Note: If the current project is not running, CitectSCADA compiles and runs it automatically. The bug in the bottom right-hand corner is green when debugging.

4. Insert a breakpoint where you want to start debugging.
5. From the **View** menu, select any debug windows you want to use. If you are unsure, you can use them all.
6. Initiate the thread by calling the function. You can do this directly from the Cicode window in the Kernel, or by using a function, etc.
7. The function will break at the specified breakpoint. You can then use the step tools to step through and trace your function.
8. Click the Toggle Debug button to stop debugging, or choose **Debug | Stop Debugging**.

Debugging functions remotely

You can debug functions remotely if both computers are running identical projects and the CitectSCADA Path is the same on both machines.

To remotely debug Cicode:

1. Click the **Cicode Editor** button on the computer that will be running CitectSCADA (the remote).
2. Choose **Debug | Options**.
3. Check (tick) the **Allow remote debugging** option.
4. Click **OK**.
5. Click the Run button (you can close the Cicode Editor first), or choose **File | Run**.
6. On the computer that will be debugging CitectSCADA, click the **Cicode Editor** button.
7. Choose **Debug | Options**.
8. Enter the Windows Computer Name or TCP/IP address of the remote CitectSCADA computer.

The Windows Computer Name is specified on the Computer Name tab of the System Properties dialog (go to **Control Panel** and select **System**).

The TCP/IP address (for example, 10.5.6.7 or plant.yourdomain.com) can be determined by going to the Command Prompt, typing **IPCONFIG**, and pressing **Enter**.

9. Click **OK**.
10. Click the debug button to start remote debugging.

Note: CitectSCADA uses Named Pipes for remote debugging. To enable the Windows

Named Pipe service, you need to enable the Server service. Select **Administrative Tools** from **Control Panel**, then select **Services**. Locate the **Server** service in the list that appears, and confirm that it is running. You can start and stop a service by right-clicking on it.

Using breakpoints

There are three ways for a processing thread to halt:

- Manually inserting a breakpoint.
- Using the `DebugBreak()` Cicode function.
- If a hardware error is detected.

To debug a function, you need to first be able to stop it at a particular point in the code. You can place a breakpoint on any line in the source code functions. Breakpoints may be inserted or removed while editing or debugging without the need for them to be saved with the file.

For a detected hardware error to halt a function, you need to have either the **Break on all hardware errors** or **Break on hardware errors in active thread** option set (**Debug menu - Options**). When the break occurs, the default Cicode Editor will be launched (if it is not open already), with the correct code file, function, and break point line displayed. To launch the debugger in this case, you need to have the **CitectSCADA will start debugger on hardware errors** option set.

Inserting or removing breakpoints

You can insert and remove breakpoints to halt processing.

To insert/remove a breakpoint:

1. Open the Cicode Editing window.
2. Position the cursor on the line where you want the breakpoint to be placed or removed.
3. Click the Debug indicator bar. Alternatively, you can press **F9** or choose **Debug | Insert/Remove Breakpoint**.

The breakpoint appears as a large red dot at the beginning of the line.

Enabling/disabling breakpoints

You can enable or disable breakpoints you have inserted into your Cicode.

To enable/disable a breakpoint:

1. Open the Cicode Editing Window.
2. Position the cursor on the line where the breakpoint is located.
3. Press **Ctrl + F9**, or choose **Debug | Enable/Disable Breakpoint**.

Note: A disabled breakpoint appears as a large dark gray (disabled) dot at the beginning of the line.

Stepping through code

Once you have halted a thread, the debugger marks the position in the code with an arrow. Now you can step through the function, line by line, and watch what happens in the debug window (see below). The following tools are provided in the Cicode Editor, to control stepping through functions.

Step Into	Advances the current Cicode thread by one statement. If the statement is a user defined function, the debugger steps into it (the pointer jumps to the first line of the source code).
Step Over	Advances the current Cicode thread by one statement. If the statement is a user defined function, the debugger steps over it (the function is not expanded).
Step Out	Advances to the end of the current function and return. If there is no calling function, the thread terminates.
Continue	Re-starts normal execution of the current Cicode thread. If there are no more breaks, the thread terminates normally.

Chapter: 15 Using Cicode Programming Standards

Implementing programming practices results in Cicode that is more robust, manageable, and predictable in execution, regardless of the author. Using programming standards entails:

- Adopting modular programming techniques.
- Helping to ensure that programs are adequately described by suitable module headers.
- Formatting code to improve readability.

The following topics are presented as a suggested means of achieving good programming standards:

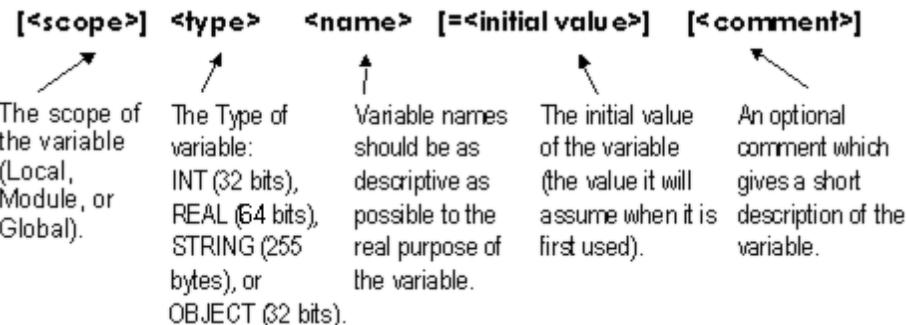
- [Variable Declaration Standards](#)
- [Variable Scope Standards](#)
- [Variable Naming Standards](#)
- [Standards for Constants, Variable Tags, and Labels](#)
- [Formatting Simple Declarations](#)
- [Formatting Executable Statements](#)
- [Formatting Expressions](#)
- [Cicode Comments](#)
- [Formatting Functions](#)
- [Modular Programming](#)
- [Defensive Programming](#)
- [Function Error handling](#)
- [Debug Error Trapping](#)

See Also

[Using Cicode Functions](#)

Variable Declaration Standards

When declaring variables you should use consistent formatting. A variable declaration has up to five parts. Each part is separated by at least one tab stop:



Note: Parts contained within square brackets are optional. For example, you may omit the variable scope (it defaults to local). Parts contained within greater than (<) and less than (>) signs should be replaced with the relevant text/value. For example, you would replace <initial value> with an actual value. (You would not bracket your value with greater than and less than signs.)

When declaring your variables, the parts of each should align vertically (the scope part of each should be vertically aligned, the type part of each should be aligned, etc.). Each part of the declaration is allotted a set amount of space. If one part is missing, its space should be left blank. The missing part should not affect the positioning of the next part:

```
Module      int      miRecipeMax=100;
int        iRecipeMax;
string    sRecipeDefault  ="Tasty";
```

See Also

[Using Cicode Programming Standards](#)

Variable Scope Standards

Local variable standards

Local Variables should be initialized, for example:

```
INT      iFile    = 0;
STRING   sName   = "";
```

```
INT          bSuccess = FALSE;
```

Module variable standards

Module Variables should be initialized, for example:

```
MODULE      INT          mhForm = -1;
MODULE      STRING       msPageName = "Loop";
```

Global variable standards

Global variables should be initialized, for example:

```
GLOBAL      INT          ghTask = -1;
GLOBAL      STRING       gsLastPage = "Menu";
```

Variable Naming Standards

The following naming conventions should be applied to variables:

- Variable names should have a small case letter prefix as follows:

Type	Pre-fix	Used for
INT (32 bits)	i	index, loop counter
INT (32 bits) and OBJECT (32 bits)	h	handle
INT (32 bits)	b	boolean (TRUE/FALSE)
REAL (64 bits)	r	real type variables
STRING (255 bytes)	s	string type variables

- Variable names typically consist of up to three words. Each word in a variable name should start with a capital letter, for example:

iTrendType, rPeriod, sFileName

- Module variable names should be prefixed with an "m", for example:

miTrendType, mrPeriod, msFileName

- Global variable names should be prefixed with a "g", for example:

giTrendType, grPeriod, gsFileName

- Local variable names should not be prefixed (when you declare a variable without specifying the scope, it is considered a Local variable by default):

iTrendType, rPeriod, sFileName

See Also

[Using Cicode Programming Standards](#)

Standards for Constants, Variable Tags, and Labels

When coding constants, variable tags and labels in Cicode you should try to use the following standards:

- [Constants](#)
- [Variable tags](#)
- [Labels](#)

Constants

In Cicode there is no equivalent of `#defines` of C language, or a type that will force variables to be constants (read-only variables). However, the variable naming convention makes constants easily identifiable so developers will treat those variables as read-only variables.

- Constants are recommended to have the prefix 'c'.
- Constants need to be declared and initialized at the beginning of the Cicode file and under no circumstances assigned a value again.

For example:

```
INT      ciTrendTypePeriodic = 1;
INT      ciTrendTypeEvent = 2;
STRING    csPageName = "Mimic";
```

Variable tags

Variable tags that have been defined in the database (with the Variable Tags form) can be used in all functions in the Cicode files. Variable tags are identifiable because they will not have a prefix (also, they are generally in uppercase letters).

Labels

Labels, like variable tags, can be used in all functions in the Cicode files. They can be either all upper case letters or mixed case. In order to differentiate them from the variable tags and other Cicode variables they should have an '_' (underscore) in front of them. For example:

```
_BILLING_EVENT, _UNIT_OFFLINE, _AfterHoursEvent
```

Note: There are a few labels without an underscore defined in the Labels form in the INCLUDE project. Although they do not follow the guidelines set in this document their wide usage makes changing those labels impractical. These labels are: TRUE, FALSE, BAD_HANDLE, XFreak, XOutsideCL, XAboveUCL, XBelowLCL, XOutsideWL, XUpTrend, XDownTrend, XGradualUp, XGradualDown, XErratic, XStratification, XMixture, ROutsideCL, RAboveUCL, RBelowLCL

See Also

[Using Cicode Programming Standards](#)

Formatting Simple Declarations

The following conventions should be observed when formatting simple Cicode declarations:

- Only one item should be declared per declaration; there should be no comma separated list of variables.
- Tab stops should be used for declarations and indentation.

For example:

```
INT      hFile,hForm;      // WRONG
INT      hFile;    // RIGHT
INT      hForm;    // RIGHT
```

The reasons for this are:

- Only the first identifier in the WRONG case is obvious and the others are often missed in a quick glance;
- It is harder to add a comment or initialization to an item in the WRONG case.
- Types, items, and initialization within a group of declarations should be vertically aligned.

For example:

```
STRING sFileName = "temp.dat"; // WRONG
INT iOffset = -1;           // WRONG
INT iState = 3;             // WRONG
STRING sFileName          = "temp.dat"; // RIGHT
INT    iOffset = -1;        // RIGHT
INT    iState  = 3;         // RIGHT
```

See Also

[Using Cicode Commands](#)

[Using Cicode Programming Standards](#)

Formatting Executable Statements

The following conventions should be observed when formatting executable statements:

- Statements are placed on new lines, indented one tab stop from the level of their surrounding block.

Note: Do not place more than one statement on a single line. While this practice is permitted in other programming languages, in Cicode the subsequent statements will not be interpreted and will effectively be lost, potentially affecting your program's runtime behavior.

WARNING

UNINTENDED EQUIPMENT OPERATION

Do not place more than one statement per line.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Although it may be argued that some statements are logically related, this is not sufficient justification. If they are logically related, place the statements on consecutive lines and separate the statements by a blank line before and after. For example:

```
hFile = -1; hForm = -1; // WRONG
hFile = -1;           // RIGHT
hForm = -1;           // RIGHT
```

- **IF statements** can be used in one of the formats below. When indenting the IF statements, a tab stop should be used. For example:

- Simple IF block

```
IF <expression> THEN
    ...
END
```

- IF-THEN-ELSE block

```
IF <expression> THEN
    ...
ELSE
    ...
END
```

- To simulate **ELSEIF blocks**, use nested statements. For example:

```
IF <expression> THEN
    ...
ELSE
    IF <expression> THEN
        ...
    ELSE
        IF <expression> THEN
            ...
        ELSE
            ...
    END
END
```

- For **WHILE** and **FOR** statements see [Working with Conditional Executors](#).

See Also

[Using Cicode Commands](#)

[Working with Conditional Executors](#)

[Using Cicode Programming Standards](#)

Formatting Expressions

The following conventions should be observed when formatting Cicode functions:

- When an expression forms a complete statement, it should, like any other statement, occupy one or more lines of its own and be indented to the current level. Operators should be surrounded by spaces. For example:

```
i= i*10+c-'0';      // WRONG
```

```
i = i * 10 + c - '0'; // RIGHT
```

- When a sub-expression is enclosed in brackets, the first symbol of the sub-expression should be placed hard against the opening bracket. The closing bracket should be placed immediately after the last character for the sub-expression. For example:

```
a = b * ( c - d ); // WRONG  
a = b * (c - d); // RIGHT
```

- The round brackets which surround the arguments of a function attract no spaces, for example:

```
DisplayText( "hello" ); // WRONG  
DisplayText("hello"); // RIGHT
```

- Commas, whether used as operators or separators, would be placed hard against the previous symbol and followed by a space. For example:

```
DevSeek(hDev ,Offset); // WRONG  
DevSeek(hDev, Offset); // RIGHT
```

See Also

[Using Cicode Expressions](#)

[Using Cicode Commands](#)

[Using Cicode Programming Standards](#)

Cicode Comments

Comments are designed to aid understanding and maintenance of code. You should place comments in the notes of the function header so as not to clutter up the code. Small one-line comments are acceptable to explain some small part of the code which may be hard to understand in the normal header comment.

See Also

[Using Cicode Programming Standards](#)

Formatting Functions

Cicode functions have up to seven parts: Scope, Type, Keyword, Name, Argument(s), Statement(s), Keyword.

[Scope]

The scope of the function. If the scope is omitted, the function will be Public by default. That is, it will be available to all Cicode files, pages, and CitectSCADA databases (for example, Alarm.dbf). To make a function Private (that is only available within the file in which it is declared), you need to prefix it with the word PRIVATE.

[Type]

The return type of the function. This should be on a separate line.

Keyword

The keyword **FUNCTION**. This should be on a separate line.

Name

The function name. Function names should follow the Function Naming Standards. This should be on a separate line.

Argument(s)

The argument list. The arguments are separated by commas and they can have default values. The argument list is normally on the same line as the function name but multiple line argument list is also acceptable if it improves readability.

Statement(s)

The statements. Each statement should be on a separate line.

Keyword

The keyword **END** which marks the end of the function. This should be on a separate line.

Note: Parts contained within square brackets - [] - are optional. For example, the scope may be omitted and if so, it will default to Public. Parts contained within greater than & less than signs - < > - should be replaced with the relevant text/value. For example, you would replace <initial value> with an actual value. You would not bracket your value with greater than & less than signs.

For example:

```

FUNCTION
PlotInit()
    <statements>
END
INT
FUNCTION
PlotOpen(STRING sName, INT nMode)
    INT      hPlot = _BAD_HANDLE;
    ...

```

```
    hPlot = ....;
    ...
    RETURN hPlot;
END
PRIVATE
STRING
FUNCTION
WasteInfoName(INT nType, INT nMode)
    STRING sName = "Sydney";
    ...
    sName = ....;
    ...
    RETURN sName;
END
```

See Also

[Writing Functions](#)

[Using Cicode Functions](#)

[Using Cicode Programming Standards](#)

Format Templates

The format of a format template string

```
[text]{<name>[,width[,justification]]}[text]...
```

Rules for valid format template display

1. If the "width" value is not present then the width is set to the length of the number of characters inclusive between '{' and '}'. This means that the field value may be truncated or padded depending on the name value length.
2. If the "width" value is specified then that is the length of the field. This means that the name value length may be truncated or padded.
3. The justification is made up of a single character with the following behaviours as specified:
 - 'R' or 'r' will align the field on the right hand side. If the width is longer than the name value length then the left hand side of the name value is padded with spaces.
 - 'L' or 'l' will align the field on the left hand side. If the width is longer than the name value length then the right hand side of the name value is padded with spaces.
 - 'Z' or 'z' will align the field on the right hand side. If the width is longer than the name value length then the left hand side of the value is padded with zeros.

- 'N' or 'n' will remove any extra padding that is used. Essentially any padding of the name value is trimmed.
4. If a justification is not specified then the name value is assumed to be left justified.
 5. Any spaces appearing after the first comma onwards in the format template will be stripped out at no penalty to the user.

Malformed format template display

There are two types of malformed templates and below are examples of each and the resulting output.

Internal malformation

This is when there is a correct open and close bracer '{' and '}' but inside the format template there is a malformation. For example there may be a space not a comma separating the name and the width. In this case the whole field is ignored which means nothing between and including '{' and '}' is displayed.

For example:

Take the following string

```
< { LocalTimeDate , 20 , R } > TagLabel < { Tag , 20 L } > DescriptionLabel < {
Desc , 20 , L } >
```

The output would as follows:

```
< 2009-07-17 11:13:17 > TagLabel < > DescriptionLabel < ValidAlarm1Desc >
```

Note: The "Tag" name value is not outputted as the field has no ',' between the width and justification.

Bracer malformation

This is when there is an open bracer '{' but no closing bracer '}'. In this case the malformation is printed as a string literal.

For example:

Take the following string:

```
< { LocalTimeDate , 20 , R } > TagLabel < { Tag , 20 , L > DescriptionLabel < {
Desc , 20 , L } >
```

The output would be as follows:

```
< 2009-07-17 11:31:44 > TagLabel < { Tag , 20 , L > DescriptionLabel <
ValidAlarm1Desc >
```

Note: The "Tag" name value is outputted as a literal as no closing brace '}' is detected.

Functions Reference

Function Naming Standards

Function names should contain at least the following information:

- A three-to-five letter description of the function type (Trend, Plot, Win).
- A one or two word description of the data to be operated on (Info, ClientInfo, Mode).
- A one word action to be taken (Get, Set, Init, Read).

For example:

```
PlotInit();TrendClientOpen();TrendClientInfoRead();
```

See Also

Naming Functions

Source file headers

Source files (the files that contain your Cicode) should have a header to provide a basic overview of the file. This header should be formatted as follows:

```
/** FILE:      <name of file.CI>
 /**
 /** DESCRIPTION:    <description of basically what is in the file>
 /**
 /** FUNCTIONS:    PUBLIC
 /**
 <list of the PUBLIC functions contained
 /**
 in this file>
 /**
 /**
 PRIVATE
 /**
 <list of the PRIVATE functions contained
 /**
 in this file>
 /**
 /**
 **** MODULE CONSTANTS*****
 <module constants>    //<comments (optional)>
 /**
 **** MODULE VARIABLES ****
 <module variables>    //<comments (optional)>
 /**
 ****
```

Note: Declare all module variables at the MODULE VARIABLES section at the

beginning of the file and initialize the module variables.

For example:

```
/** FILE: Recipe.CI
/** DESCRIPTION: Contains all functions for gathering recipe data.
/** FUNCTIONS: PUBLIC
/** OpenRecipeDatabase
/** CloseRecipeDatabase
/** ReadRecipeData
/** WriteRecipeData
/** GatherRecipeData
/** RecipeForm
/** OpenRecipeDatabase
/** PRIVATE
/** ButtonCallback
*****
***** MODULE CONSTANTS*****
module int cmiRecipeMax =100; //Maximum number of recipes
***** MODULE VARIABLES *****
module int miRecipeNumber =0; //Minimum number of recipes
*****
```

Function headers

Functions should have a descriptive header, formatted as follows:

```
/** FUNCTION : <name of function>
/** DESCRIPTION : <suggests the operation, application source and
/** multi-tasking issues>
/** REVISION DATE BY COMMENTS
/** <revision number> <date> <author> <comments about the change>
/** ARGUMENTS: <argument description>
/** RETURNED VALUE: < description of possible return values>
/** NOTES:
```

The order of functions in a file is important for efficient operation.

Initialization and shutdown functions should be placed at the top of the file. Command functions should be next. Local utility functions should be at the bottom of the file.

For example:

```
/***      FUNCTION :      OpenRecipeDatabase
/***
/***      DESCRIPTION :      Opens the specified database.
/***
/***      REVISION      DATE      BY      COMMENTS
/***      1      28/09/97      BS      Original
/***      2      05/10/97      SFA      Added INI checking
/***
/***      ARGUMENTS:
/***
/***          STRING sName      Name of the recipe database.
/***
/***          INT dwMode      Mode to open the recipe database.
/***                      0 for read only, 1 for read/write.
/***
/***          RETURNED VALUE:      Handle if successful, otherwise -1.
/***
/***      NOTES:
INT
FUNCTION
OpenRecipeDatabase(STRING sName, INT dwMode)
...
END
```

Modular Programming

One of the more effective programming practices involves partitioning large, complex programming challenges into smaller and more manageable sub-tasks and reusable functions. A similar approach should be taken when using a programming language like Cicode to complete a task. Reducing the task to smaller tasks (or functions) has the following advantages:

- **Reduced Complexity** - Once the function is created and tested, the detailed operation about how it works need not be revisited. Users need only focus on the results produced by the function.
- **Avoids Duplicate Code** - Creating a generic function instead of copying similar code reduces the total amount of code in the system. It also means the function can be reused by separate code areas. This makes the code more maintainable because it is smaller in size, and only one instance needs to be modified.
- **Hides Information** - Information can be in the form of operations, data, or resources. Access to information can be controlled when functions are written that provide a limited set of actions to be performed on the information. For example, if a user wishes to log a message to a database, he or she should only send the message to a function, say **LogDBaseMessage("hello world")**, and the function should control the database resource. The function then becomes the single interface to the database resource. Resources that have multiple interfaces to them are harder to control. This

is because in a multitasking environment, the user cannot control or even know in advance the order of code execution, and hence a resource may be modified at the same time by different tasks. Information hiding can also smooth out any wrinkles in standard functions, minimizing possible misuse of resources such as semaphores, queues, devices, and files. Functions that do this are often called 'wrapper' functions as they add a protective shell to existing functions.

- **Improves Performance** - Optimizing code that resides in one place immediately increases the performance of code that calls this function. Scattered code will require multiple areas to be modified should any optimization be necessary.
- **Isolates Complex Code** - Code that requires complex operations such as communications protocols, complex algorithms, boolean logic, or complex data manipulation is susceptible to errors. Placing this code in a separate function reduces the possibility of this code corrupting or halting other code.
- **Improves Readability** - A small function with meaningful parameter names assists readability as it is a step towards self-documenting code and reduces the need to scan multiple pages of code to establish what the operation is meant to achieve.

Modular programming has a few rules that define how functions should be structured - Cohesion - and how they are related to other functions - Coupling.

See Also

[Defensive Programming](#)

[Using Cicode Programming Standards](#)

Cohesion

A goal of modular programming is to create simple functions that perform a single task, but perform that task well. This can be described as how 'cohesive' a function is.

Two factors that affect the level of cohesion are:

- Number of tasks the function performs.
- Similarity of the tasks.

The following table illustrates the different levels of cohesion:

Number of tasks	Sim- ilar- ity	Cohesion level	Example
1	Not applic- able	High	$\text{Sin}(x)$
More than 1	Similar	Mod- erate	$\text{SinAndTan}(x)$

Number of tasks	Sim- ilarity	Cohesion level	Example
More than 1	Dis- similar	Low	SinAndLength(x)
Many	Radically different	None	SinAnd- DateAndTimeAndSQLNext(x)

For example, the function **Sin(x)** will perform one task - return the trigonometric Sine value of x. This is an example of a highly cohesive function. The function **SinAndTan(x)** performs two tasks - calculate the trigonometric Sine and Tan of the value X. This function has lower cohesion than **Sin(x)** because it performs two tasks.

Highly cohesive functions are more dependable, easier to modify, and easier to debug than functions that have lower levels of cohesion and are hence acceptable and encouraged.

Low cohesion functions are typically complex, prone to errors, and are more costly to fix. Low cohesion functions are regarded as poor programming practice and discouraged.

Coupling

Another rule of modular programming is to reduce the number of relationships between functions. This is referred to as function coupling. Functions that have few, or no, relationships between them are loosely coupled. Loosely coupled functions provide simple, visible interfaces to the function. This makes the functions easier to use and modify. For example, the Cicode function **TimeCurrent()** is a loosely coupled function. To use this function, a user need only call its name, and the function will return with the desired result. The user does not need to be aware of any relationships because there are no parameters passed to the function, and it does not read from, or write to, any global data. There is very little likelihood of error with this function; it only returns a time/date variable and does not support error codes. In the unlikely event that the function did not return the time/date variable, it would be through no error of the calling function because it has no relationship to it.

Functions that have many relationships between them are tightly coupled. For example, a user written function like **AddCustomerRecord(hDatabase, sFirstName, sSurname, sAddress, sAge, sPhone)** has a higher level of coupling than the function **TimeCurrent()**. Tightly coupled functions are inflexible in their use, less robust, and harder to maintain. The **AddCustomerRecord()** function is less robust because it could experience an error of its own accord, or if the function calling it passes bad data to it. Tightly coupled functions are harder to maintain because modifying a function with many relationships in it may result in modifications to other functions to accept the data.

The different types of function relationships are listed below:

- **Passed parameters.** A simple, visible form of loose coupling that is encouraged. Once the number of parameters passed to a function exceeds seven, you should consider partitioning the function into two smaller functions. These types of relationships are acceptable.
- **Control information.** Control information causes the called function to behave in a different way. For example, the function **ChangeData(iMode)**, behaves differently depending on the value of the variable **iMode** that is passed into it. It may be responsible for deleting, inserting, or updating data. In addition to being a tightly coupled function, it has low cohesion because it performs multiple tasks. This function could be divided into three separate functions to perform the separate tasks. These types of relationships are moderately acceptable.
- **Shared common data.** This is often referred to as global variable data. This is an invisible form of tight coupling that, particularly in pre-emptive, multi-tasking environments, can result in an unpredictable, hard-to-maintain program. When functions write to the global variable data there is no monitoring of or restriction on who is writing to the variable. Hence the value can be indeterminate. Global variables are acceptable when they are used for read-only purposes, otherwise their use is discouraged. Similarly, module variable data in CitectSCADA should be treated the same way. The use of local function variables is encouraged to decrease function coupling.

Defensive Programming

Defensive programming is an approach to improve software and source code. It aims to improve general quality by reducing the number of software bugs. It promotes making the source code readable and understandable. It aims to make your code behave in a predictable manner despite unexpected input or user actions.

You should try to:

- Verify that your code does not produce unexplained hardware alarms.
- Check that denominators in division are not zero.
- Check that array indexes cannot go out of range.
- Check that arguments from external sources are valid.
- Check that loop terminations are obvious and achievable.
- Only write code once. If you find that two sections of code look identical or almost identical it is worth spending the time to re-write or re-design it. This will generally reduce the size of the code in question by a third or more, which reduces complexity

and therefore maintenance and debugging time. An effective method to achieve this is to convert the identical code to a new function.

- Do not access the module data in any function other than the member functions.
- Write the member functions whenever an array is defined. Do not try to pass arrays between functions, make the arrays the centre piece of the object.
- Cicode is a multitasking language. Several tasks (commands, expressions and tasks created by TaskNew function) can be executed concurrently. This powerful feature of Cicode should be used with care as some of the functions may be modifying module data. It is essential that the number of tasks running at any point in time be minimized. This may require the use of semaphores to help protect the module data from interference and corruption. (For the use of semaphores, refer to SemOpen, SemClose, SemSignal and SemWait functions in on-line help or the Cicode Reference manual).

WARNING

UNINTENDED EQUIPMENT OPERATION

- Write your Cicode programs with the minimum number of concurrent instructions suitable to your application.
- Use semaphores or some related means to coordinate program flow if your program will execute a high number of concurrent instructions.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

See Also

[Using Cicode Programming Standards](#)

[Modular Programming](#)

[Function Error handling](#)

Function Error handling

Errors are handled by examining the return values of the functions. The Cicode functions can be classified as regards their return value as follows:

Functions return-ing	Calling functions should check for
Error code only	0 no error (success) > 0 error code
Handles	-1 bad handle

Functions returning	Calling functions should check for
	>= 0 valid handle
Random values	the return of IsError()

The following Cicode functions can halt the current task:

DevOpen, DevHistory, DevNext, DevPrev, DevSeek, DevFind, DevFlush, DevRecNo, DevRead, DevReadLn, DevAppend, DevDelete, DevZap, DevControl, DevPrint, DevModify, ErrTrap; FileOpen, FileClose, FileReadBlock, FileWriteBlock, FileSeek, FileDelete, FileReName, FileSize, FileReadLn, FileCopy; FormNew; SQLConnect, SQLTraceOn, SQLTraceOff, SQLErrMsg.

If an error is detected in one of these functions, your Cicode task will generate a hardware error and be halted. You may stop your Cicode task from being halted by using the ErrSet() function and checking for errors using IsError().

The parameter [Code]HaltOnError allows you to stop any errors detected in these functions from halting your Cicode. If you set...

```
[code]
HaltOnError=0
```

then your Cicode will continue to run after a hardware error is detected in these functions.

For example:

- **Example of error code only**

```
INT
FUNCTION
ExampleInit()
    INT      nError    = 0;
    nError = ExampleOpen("MyForm");
    IF nError = 0 THEN
        ...
    END
END
INT
FUNCTION
ExampleOpen(STRING sName)
    INT      nError    = 0;
    ...
    IF <an error has been detected> THEN
        nError = 299;
    END
    RETURN nError;
END
```

- **Example of handles**

```
INT
FUNCTION
ExampleInit()
    INT      hFile     = BAD_HANDLE;
    ...
    hFile = ExampleFileOpen("MyFile.txt");
    IF hFile <> BAD_HANDLE THEN
        ...
    END
END
FUNCTION
ExampleFileOpen(STRING sName)
    INT      hFile = BAD_HANDLE;
    hFile = FileOpen(sName, "r+");
    IF hFile = BAD_HANDLE THEN
        hFile = FileOpen(sName, "r");
    END
    RETURN hFile;
END
```

- **Example of random values**

```
INT
FUNCTION
ExampleInit()
    INT      nSamples      = 0;
    ...
    ErrSet(1);
    nSamples = ExampleSamples();
    IF IsError() = 0 THEN
        ...
    END
    ErrSet(0);
END
INT
FUNCTION
ExampleSamples()
    INT      nSamples = 0;
    INT      nError   = 0;
    ...
    ErrTrap(nError);
    RETURN nSamples;
END
```

See Also

[Debugging Cicode](#)

Debug Error Trapping

The functions listed below can also be used during normal project execution to trap run-time problems:

- [DebugMsg function](#)
- [Assert function](#)

DebugMsg function

`DebugMsg()` internally calls the `TraceMsg()` function if the debug switch is on. The implementation of this function can be found in `DEBUG.CI` in the `INCLUDE` project. You can turn the debug switch on or off by doing any of the following:

- Calling `DebugMsgSet(INT bDebugMsg)` on the Kernel Cicode window. (Or, this function can be called from a keyboard command or something similar.)
- Changing the `[Code]DebugMessage` parameter in the INI file.

For example:

```

INT
FUNCTION
FilePrint(STRING sDeviceName, STRING sFileName)

    INT      hFile;
    INT      hDev;
    STRING   Str1;

    hDev = DevOpen(sDeviceName, 0);
    IF (hDev = -1) THEN
        DebugMsg("Invalid arg to FilePrint - 'DeviceName'");
        RETURN 261; /* File does not exist */
    END
    hFile = FileOpen(sFileName, "r");
    IF (hFile = -1) THEN
        DebugMsg("Invalid arg to FilePrint - 'FileName'");
        DevClose(hDev);
        RETURN 261; /* File does not exist */
    END
    WHILE NOT FileEof(hFile) DO
        Str1 = FileReadLn(hFile);
        DevWriteLn(hDev, Str1);
    END
    FileClose(hFile);
    DevClose(hDev);
    RETURN 0;
END

```

Assert function

Assert reports an error if the test passed by the argument does not return the expected value. The implementation of this function can be found in DEBUG.CI in the INCLUDE project.

For example:

```
INT
FUNCTION
FileDisplayEx(STRING sFileName)

    INT      hFile;

    hFile = FileOpen(sFileName, "r");
    ASSERT(hFile > -1);
    ...
    FileClose(hFile);
    RETURN 0;
END
```

See Also

[Debugging Cicode](#)

Part: 3

Functions Reference

This section describes Cicode functions, and provides syntax and use examples.

Note: In some examples, lines of code might wrap due to page size limitations. Cicode does not support code written over more than one line and has no line continuation character. Cicode uses the semicolon (;) as the end-of-line character. If you copy these examples into your project, reassemble any lines that have wrapped and place them back onto the one line in your code.

Cicode includes the following function categories:

Accumulator Functions	I/O Device Functions
ActiveX Functions	Keyboard Functions
Alarm Functions	Mail Functions
Clipboard Functions	Math/Trigonometry Functions
Cluster Functions	Miscellaneous Functions
Color Functions	Page Functions
	Plot Functions

Communication Functions	Quality Functions
DDE Functions	Report Functions
Device Functions	Security Functions
Display Functions	SPC Functions
DLL Functions	SQL Functions
Error Functions	String Functions
Event Functions	Super Genie Functions
File Functions	Table (Array) Functions
Form Functions	Tag Functions
Format Functions	Task Functions
FTP Functions	Time/Date Functions
FuzzyTech Functions	Trend Functions
Group Functions	Window Functions

Chapter: 16 Accumulator Functions

Accumulator functions enable you to programmatically browse and control Accumulators and retrieve information from them.

Accumulator Functions

Following are functions relating to accumulators.

AccControl	Controls accumulators for example, motor run hours.
AccumBrowseClose	Closes an accumulator browse session.
AccumBrowseFirst	Gets the oldest accumulator entry.
AccumBrowseGetField	Gets the field indicated by the cursor position in the browse session.
AccumBrowseNext	Gets the next accumulator entry in the browse session.
Accum-BrowseNumRecords	Returns the number of records in the current browse session.
AccumBrowseOpen	Opens an accumulator browse session.
AccumBrowsePrev	Gets the previous accumulator entry in the browse session.

See Also

[Functions Reference](#)

AccControl

Controls accumulators, for example, motor run hours. You can reset the values of Run Time, Totalizer Inc, and No. of Starts (defined in the Accumulator database), re-read these values from the I/O device, or flush pending writes of these values to the I/O device.

Syntax

AccControl(*sName*, *nMode* [, *ClusterName*])

sName:

The name of the accumulator or a mask for the names of accumulators. You can use the following wildcards:

- * matches all following characters, for example, "Motor*" matches all accumulators starting with the word "Motor"
- ? matches any single character, for example, "Motor?10" matches "MotorA10" and "MotorB10"

This argument can be prefixed by the name of the cluster for example ClusterName.AccumulatorName.

nMode:

The mode of the control:

- 1 - Reset Run Time and Totalizer value
- 2 - Reset No. of Starts
- 3 - Reset Run Time, Totalizer value, and No. of Starts
- 4 - Flush pending writes to the I/O device
- 5 - Re-read Run Time, Totalizer value, and No. of Starts from the I/O device

ClusterName:

Name of the cluster in which the accumulator resides. This is optional if you have one cluster or are resolving the reports server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Example

```
! Reset all accumulator variables for accumulator "MCC123".
AccControl("MCC123", 3, "ClusterXYZ");
```

See Also

[Accumulator Functions](#)

AccumBrowseClose

The AccumBrowseClose function terminates an active data browse session and cleans up all resources associated with the session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AccumBrowseClose(*iSession*)

iSession:

The handle to a browse session previously returned by a AccumBrowseOpen call.

Return Value

0 (zero) if the accumulator browse session exists, otherwise an [error](#) is returned.

Related Functions

[AccumBrowseFirst](#), [AccumBrowseGetField](#), [AccumBrowseNext](#), [AccumBrowseNumRecords](#), [AccumBrowseOpen](#), [AccumBrowsePrev](#)

See Also

[Accumulator Functions](#)

AccumBrowseFirst

The AccumBrowseFirst function places the data browse cursor at the first record.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AccumBrowseFirst(*iSession*)

iSession:

The handle to a browse session previously returned by a AccumBrowseOpen call.

Return Value

0 (zero) if the accumulator browse session exists, otherwise an [error](#) is returned.

Related Functions

[AccumBrowseClose](#), [AccumBrowseGetField](#), [AccumBrowseNext](#), [AccumBrowseNumRecords](#), [AccumBrowseOpen](#), [AccumBrowsePrev](#)

See Also

[Accumulator Functions](#)

AccumBrowseGetField

The AccumBrowseGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AccumBrowseGetField(*iSession*, *sFieldName*)

iSession:

The handle to a browse session previously returned by a AccumBrowseOpen call.

sFieldName:

The name of the field that references the value to be returned. Supported fields are:

NAME, CLUSTER, PRIV, AREA, TRIGGER, VALUE, RUNNING,
STARTS, TOTALISER

See [Browse Function Field Reference](#) for information about fields.

Return Value

The value of the specified field as a string. An empty string may or may not be an indication that an error has been detected. The last error should be checked in this instance to determine if an error has actually occurred.

Related Functions

[AccumBrowseClose](#), [AccumBrowseFirst](#), [AccumBrowseNext](#), [AccumBrowseNumRecords](#),
[AccumBrowseOpen](#), [AccumBrowsePrev](#)

Example

```
STRING fieldValue = "";
STRING fieldName = "TYPE";
INT errorCode = 0;
...
fieldValue = AccumBrowseGetField(iSession, sFieldName);
IF fieldValue <> "" THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Accumulator Functions](#)

[AccumBrowseNext](#)

The AccumBrowseNext function moves the data browse cursor forward one record. If you call this function after you have reached the end of the records, error 412 is returned (Databrowse session EOF).

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AccumBrowseNext(*iSession*)

iSession

The handle to a browse session previously returned by a AccumBrowseOpen call.

Return Value

0 (zero) if the accumulator browse session exists, otherwise an [error](#) is returned.

Related Functions

[AccumBrowseClose](#), [AccumBrowseFirst](#), [AccumBrowseGetField](#), [AccumBrowseNumRecords](#), [AccumBrowseOpen](#), [AccumBrowsePrev](#)

See Also

[Accumulator Functions](#)

AccumBrowseNumRecords

The AccumBrowseNumRecords function returns the number of records that match the filter criteria.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AccumBrowseNumRecords(*iSession*)

iSession:

The handle to a browse session previously returned by a AccumBrowseOpen call.

Return Value

The number of records that have matched the filter criteria. A value of 0 denotes that no records have matched. A value of -1 denotes that the browse session is unable to provide a fixed number. This may be the case if the data being browsed changed during the browse session.

Related Functions

[AccumBrowseClose](#), [AccumBrowseFirst](#), [AccumBrowseGetField](#), [AccumBrowseNext](#),
[AccumBrowseOpen](#), [AccumBrowsePrev](#)

Example

```
INT numRecords = 0;  
...  
numRecords = AccumBrowseNumRecords(iSession);  
IF numRecords <> 0 THEN  
    // Have records  
ELSE  
    // No records  
END  
...
```

See Also

[Accumulator Functions](#)

AccumBrowseOpen

The AccumBrowseOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AccumBrowseOpen(*sFilter*, *sFields* [, *sClusters*])

sFilter:

A filter expression specifying the records to return during the browse. An empty string indicates that all records will be returned. Where a fieldname is not specified in the filter, it is assumed to be tagname. For example, the filter "AAA" is equivalent to "name=AAA".

Note: Use the following fields with care in filters since they return the actual value of the variable tag which they refer to.

RUNNING, STARTS, TOTALISER, TRIGGER, VALUE.

sFields:

Specifies via a comma delimited string the columns to be returned during the browse. An empty string indicates that the server will return all available columns. Supported fields are:

COMMENT, TAGGENLINK.

See [Browse Function Field Reference](#) for information about fields.

sClusters:

An optional parameter that specifies via a comma delimited string the subset of the clusters to browse. An empty string indicates that the connected clusters will be browsed.

Return Value

Returns an integer handle to the browse session. Returns -1 when an error is detected.

The returned entries will be ordered alphabetically by name. After a reload of the Report Server, any new records may be added at the end.

Related Functions

[AccumBrowseClose](#), [AccumBrowseFirst](#), [AccumBrowseGetField](#), [AccumBrowseNext](#),
[AccumBrowseNumRecords](#), [AccumBrowsePrev](#)

Example

```
INT iSession;
...
iSession = AccumBrowseOpen ("NAME=ABC*", "NAME,AREA",
"ClusterA,ClusterB");
IF iSession <> -1 THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Accumulator Functions](#)

AccumBrowsePrev

The AccumBrowsePrev function moves the data browse cursor back one record. If you call this function after you have reached the beginning of the records, error 412 is returned (Databrowse session EOF).

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AccumBrowsePrev(*iSession*)

iSession:

The handle to a browse session previously returned by a AccumBrowseOpen call.

Return Value

0 (zero) if the accumulator browse session exists, otherwise an [error](#) is returned.

Related Functions

[AccumBrowseClose](#), [AccumBrowseFirst](#), [AccumBrowseGetField](#), [AccumBrowseNext](#),
[AccumBrowseNumRecords](#), [AccumBrowseOpen](#)

See Also

[Accumulator Functions](#)

Chapter: 17 ActiveX Functions

ActiveX functions enable you to create and interact with ActiveX objects, using Citect-SCADA as an ActiveX container.

ActiveX Functions

Following are functions relating to ActiveX objects:

ObjectCallMethod	Calls a specific method for an ActiveX object.
ObjectGetProperty	Retrieves a specific property of an ActiveX object.
ObjectSetProperty	Sets a specific property of an ActiveX object.
AnByName	Retrieves the animation point number of an ActiveX object.
CreateControlObject	Creates a new instance of an ActiveX object.
CreateObject	Creates the automation component of an ActiveX object.
ObjectAssociateEvents	Allows you to change the ActiveX object's event class.
Objec-tAssociatePropertyWithTag	Establishes an association between a variable tag and an ActiveX object property.
ObjectByName	Retrieves an ActiveX object.
ObjectHasInterface	Queries the ActiveX component to determine if its specific interface is supported.
ObjectIsValid	Determines if the given handle for an object is valid.
ObjectToStr	Converts an object handle to a string.

See Also

[Functions Reference](#)

[ObjectCallMethod](#)

Calls a specific method for an ActiveX object. (See the documentation for your ActiveX object for details on methods and properties.)

Note: The parameter list passed to the control can only have Cicode variables or variable tags; it cannot use values returned directly from a function because an ActiveX control may modify parameters passed to it.

For example:

```
//Calculate a value and pass to ActiveX control  
_ObjectCallMethod(hControl, "DoSomething" CalcValue());
```

is not allowed because the return value of a function cannot be modified. The following should be used instead:

```
INT nMyValue;  
//Calculate Value  
nMyValue = CalcValue();  
//Pass Value to ActiveX control  
_ObjectCallMethod(hControl, "DoSomething" nMyValue);
```

Syntax

_ObjectCallMethod(*hObject*, *sMethod*, *vParameters*)

hObject:

The handle for the object (as returned by the ObjectByName() function).

sMethod:

The name of the method.

vParameters:

A variable length parameter list of method arguments. The variables will be passed however you enter them, and will then be coerced into appropriate automation types. Likewise, any values modified by the automation call will be written back - with appropriate coercion - into the passed Cicode variable.

Return Value

The return value from the method - if successful, otherwise an [error](#) is returned.

Related Functions

[ObjectByName](#), [DspAnCreateControlObject](#), [CreateObject](#), [CreateControlObject](#)

Example

See [CreateControlObject](#).

See Also

[ActiveX Functions](#)

_ObjectGetProperty

Gets a specific property of an ActiveX object.

Syntax

_ObjectGetProperty(*hObject*, *sProperty*)

hObject:

The handle for the object (as returned by the ObjectByName() function).

sProperty:

The name of the property you want to get.

Return Value

The value of the property - if successful, otherwise an [error](#) is returned.

Related Functions

[ObjectByName](#), [DspAnCreateControlObject](#), [CreateObject](#), [CreateControlObject](#)

Example

See [CreateControlObject](#)

See Also

[ActiveX Functions](#)

_ObjectSetProperty

Sets a specific property of an ActiveX object.

Syntax

_ObjectSetProperty(*hObject*, *sProperty*, *vValue*)

hObject:

The handle for the object (as returned by the ObjectByName() function).

sProperty:

The name of the property you want to set.

vValue:

The value to which the property will be set. This value can be of any data type. Appropriate coercion will take place when creating the equivalent automation parameter.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ObjectByName](#), [DspAnCreateControlObject](#), [CreateObject](#), [CreateControlObject](#)

Example

See [CreateControlObject](#)

See Also

[ActiveX Functions](#)

AnByName

Retrieves the animation point number of an ActiveX object.

Syntax

AnByName(*sName*)

sName:

The name for the object in the form of "AN" followed by its AN number, for example, "AN35". This name is used to access the object.

Return Value

The animation point number of the object - if successful, otherwise an [error](#) is returned.

Related Functions

[CreateControlObject](#)

See Also

[ActiveX Functions](#)

CreateControlObject

Creates a new instance of an ActiveX object.

An object created using this function remains in existence until the page is closed or the associated Cicode Object is deleted. This function does not require an existing animation point. When the object is created, an animation point is created internally. This animation point is freed when the object is destroyed.

Syntax

CreateControlObject(*sClass*, *sName*, *x1*, *y1*, *x2*, *y2*, *sEventClass*)

sClass:

The class of the object. You can use the object's human readable name, its program ID, or its GUID. If the class does not exist, the function will return an error message.

For example:

- "Calendar Control 8.0" - human readable name
- "MSCAL.Calendar.7" - Program ID
- "{8E27C92B-1264-101C-8A2F-040224009C02}" - GUID

sName:

The name for the object in the form of "AN" followed by its AN number, for example, "AN35". This name is used to access the object.

x1:

The x coordinate of the object's top left hand corner as it will appear in your CitectSCADA window.

y1:

The y coordinate of the object's top left hand corner as it will appear in your CitectSCADA window.

x2:

The x coordinate of the object's bottom right hand corner as it will appear in your CitectSCADA window.

y2:

The y coordinate of the object's bottom right hand corner as it will appear in your CitectSCADA window.

sEventClass:

The string you would like to use as the event class for the object.

Return Value

The newly created object, if successful, otherwise an [error](#) is generated.

Related Functions

[DspAnCreateControlObject](#), [CreateObject](#), [AnByName](#)

Example

```
// This function creates a single instance of the calendar control
// at the designated location with an object name of "CalendarEvent"
// and an event class of "CalendarEvent"
FUNCTION
CreateCalendar()
    OBJECT Calendar;
    STRING sCalendarClass;
    STRING sEventClass;
    STRING sObjectName;
    sCalendarClass = "MSCal.Calendar.7";
    sEventClass = "CalendarEvent";
    sObjectName = "MyCalendar";
    Calendar = CreateControlObject(sCalendarClass, sObjectName, 16,
    100, 300, 340, sEventClass);
END
// This function shows how to change the title font of the
// calendar
FUNCTION
CalendarSetFont(STRING sFont)
    OBJECT Font;
    OBJECT Calendar;
    Calendar = ObjectByName("MyCalendar");
    Font = _ObjectGetProperty(Calendar, "TitleFont");
    _ObjectSetProperty(Font, "Name", sFont);
END
// This function shows how to change the background color of the
// calendar
FUNCTION
CalendarSetColor(INT nRed, INT nGreen, INT nBlue)
    OBJECT Calendar;
    Calendar = ObjectByName("MyCalendar");
    _ObjectSetProperty(Calendar, "BackColor",
    PackedRGB(nRed,nGreen,nBlue));
END
// This function shows how to call the NextDay method of the
// calendar
FUNCTION
CalendarNextDay()
    OBJECT Calendar;
    Calendar = ObjectByName("MyCalendar");
    _ObjectCallMethod(Calendar, "NextDay");
END
// This function shows you how to write a mouse click event
// handler for the calendar
FUNCTION
CalendarEvent_Click(OBJECT This)
    INT nDay;
    INT nMonth;
```

```

INT nYear;
nDay = _ObjectGetProperty(This, "Day");
nMonth = _ObjectGetProperty(This, "Month");
nYear = _ObjectGetProperty(This, "Year");
...
Your code goes here...
...
END

```

See Also[ActiveX Functions](#)**CreateObject**

Creates a new instance of an ActiveX object. If you use this function to create an ActiveX object, it will have no visual component (only the automation component will be created).

If you assign an object created with the CreateObject() function to a local variable, that object will remain in existence until the variable it is assigned to goes out of scope. This means that such an object will only be released when the Cicode function that created it ends.

If you assign an object created with the CreateObject() function to a module or global scope variable, then that object will remain in existence until the variable either has another object assigned or is set to NullObject, *provided the CreateObject() call is not made within a loop*.

Objects created by calls to CreateObject() within WHILE or FOR loops are only released on termination of the Cicode function in which they are created, regardless of the scope of the variable to which the object is assigned. The use of CreateObject() within a loop may therefore result in the exhaustion of system resources, and is not generally recommended unless performed as shown in the examples below.

 WARNING**UNINTENDED EQUIPMENT OPERATION**

Do not use the CreateObject() function within a loop except in strict accordance with the following instructions.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax**CreateObject(sClass)**

sClass:

The class of the object. You can use the object's human readable name, its program ID, or its GUID. If the class does not exist, the function will return an error.

For example:

- "Calendar Control 8.0" - human readable name
- "MSCAL.Calendar.7" - Program ID
- "{8E27C92B-1264-101C-8A2F-040224009C02}" - GUID

Return Value

The newly created object, if successful, otherwise an [error](#) is generated.

Related Functions

[DspAnCreateControlObject](#), [CreateControlObject](#)

Example

The following examples show correct techniques for calling CreateObject() within a loop.

```
/* In the example below, the variable objTest is local. Resources
associated with calls to ProcessObject() will be released each
time that function ends. */
FUNCTION Forever()
    WHILE 1 DO
        ProcessObject();
        Sleep(1);
    END
END
FUNCTION ProcessObject()
    .OBJECT objTest;
    objTest=CreateObject("MyObject");
    - do something
END
/* In the example below, the variable objTest is global. Resources
associated with calls to ProcessObject() will be released when
objTest is set to NullObject. */
FUNCTION Forever()
    WHILE 1 DO
        ProcessObject();
        Sleep(1);
    END
END
FUNCTION ProcessObject()
    objTest=CreateObject("MyObject");
    - do something
    objTest=NullObject;
END
```

See Also

[ActiveX Functions](#)

ObjectAssociateEvents

Allows you to change the ActiveX object's event class. If you have inserted an object on a graphics page using Graphics Builder, it allows you to change the event class to something other than the default of **PageName_ObjectName**

Syntax

ObjectAssociateEvents(*sEventClass*, *hSource*)

sClass:

The class of the object. You can use the object's human readable name, its program ID, or its GUID.
If the class does not exist, the function will report an error.

hSource:

The source object firing the events which are to be handled by the event handler.

For example:

- "Calendar Control 8.0" - human readable name
- "MSCAL.Calendar.7" - Program ID
- "{8E27C92B-1264-101C-8A2F-040224009C02}" - GUID

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspAnCreateControlObject](#), [CreateControlObject](#)

Example

Inserting ActiveX objects using Cicode

If you have created an ActiveX object using Cicode (for example, by calling the function **CreateControlObject()**), the parameter 'sEventClass' would have required you to define an event class for the object to enable event handling. If you want to change the class you used, you can call **ObjectAssociateEvents()**.

Inserting ActiveX objects via Graphics Builder

If you have inserted an ActiveX object in Graphics Builder, runtime will automatically create an event class for the object in the form **PageName_ObjectName**. If this is the case, you may want to change the object's event class.

Using the example of an ActiveX sludge tank controller, the default event class for the object could be "PageName_AN35". This means any events handlers for the object would take the form "PageName_AN35_Click" (presuming this example relates to a click event). You may want to define this more clearly, in which case you can call the following:

```
// This function redefines the event class for the ActiveX sludge
tank controller at AN35 to "SludgeTank". //
ObjectAssociateEvents ("SludgeTank", ObjectByName("AN35"));
..
```

With the event class for the object now defined as "SludgeTank", the event handlers can take the form "SludgeTank_Click".

This would be useful if you define event handlers in relation to an object that will eventually be copied to other graphics pages, as it will reduce the need to redefine the event handlers to identify the default event class associated with each new placement of the object.

See Also

[ActiveX Functions](#)

ObjectAssociatePropertyWithTag

Establishes an association between an ActiveX property and a variable tag. This means that any changes made to an ActiveX object property will be mirrored in the variable tag.

Generally, ActiveX objects issue "property change notifications" to CitectSCADA whenever a change occurs to a specific property value. This notification tells CitectSCADA when to read the property value.

Note: An association will not succeed if property change notifications are not supported and the OnChangeEvent argument is left blank. Verify that the scaling and units of the associated tag are compatible with the ActiveX property values. However, some property changes do not trigger property change notifications. If this is the case, you need to choose an appropriate "on change" event instead - an event fired by the ActiveX object in response to a change. An "appropriate" event is one that happens to be fired whenever the property value changes. For example, the MS Calendar Control fires an AfterUpdate event whenever a day button is pressed.

Syntax

ObjectAssociatePropertyWithTag(*sObject*, *sPropertyName*, *sTagName* [, *sOnChangeEvent*])

sObject:

The object instance that associates a property with a tag.

sPropertyName:

The name of the ActiveX property to associate with the tag.

sTagName:

The name of the CitectSCADA variable tag to associate with the property.

sOnChangeEvent:

The name of the "on change" event that informs CitectSCADA of a change to the ActiveX object. This is required where the ActiveX object does not automatically generate a property change notification. Choose an event that happens to be fired whenever the ActiveX object property changes, for example, the MS Calendar Control fires an AfterUpdate event whenever a day button is pressed.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspAnCreateControlObject](#), [CreateObject](#), [CreateControlObject](#)

See Also

[ActiveX Functions](#)

ObjectByName

Retrieves an ActiveX object. This is useful when you know the object by name only (this will often be the case for objects created during configuration, rather than those created at runtime using a Cicode function).

Syntax

ObjectByName(sName)

sName:

The name for the object in the form of "AN" followed by its AN number, for example, "AN35". This name is used to access the object. The sName argument should be enclosed in quotes "".

Return Value

The requested object, if successful, otherwise an [error](#) is generated.

Related Functions

[DspAnCreateControlObject](#), [CreateObject](#), [CreateControlObject](#)

Example

See [CreateControlObject](#)

See Also

[ActiveX Functions](#)

ObjectHasInterface

Queries the ActiveX component to determine if its specific interface is supported. (Refer to the ActiveX object's documentation for details of its interfaces.)

Syntax

ObjectHasInterface(*hObject*, *sInterface*)

hObject:

The handle for the object (as returned by the ObjectByName() function).

sInterface:

The name of the interface (case sensitive).

Return Value

0 if the interface is not supported, otherwise 1.

Related Functions

[ObjectByName](#), [CreateObject](#), [CreateControlObject](#)

Example

```
hPen = _ObjectGetProperty(hControl, "Pen");
IF ObjectHasInterface(hPen, "IDigitalPen") THEN
    //Fill is only supported on digital pen
    _ObjectSetProperty(hPen, "Fill", 0)
END
```

See Also

[ActiveX Functions](#)

ObjectIsValid

Determines if the given handle for an object is a valid handle. This function is useful for programmatically checking that an object was returned for a call.

Syntax

ObjectIsValid(hObject)

hObject:

The handle for the object (as returned by the ObjectByName() function).

Return Value

0 if the handle is not valid, otherwise 1.

Related Functions

[ObjectByName](#), [CreateObject](#), [CreateControlObject](#)

Example

```
hFont = _ObjectGetProperty(hControl, "Font");
IF ObjectIsValid(hFont) THEN
    _ObjectSetProperty(hFont, "Size", 22)
END
```

See Also

[ActiveX Functions](#)

ObjectToStr

Converts an object handle to a string.

Syntax

ObjectToStr(hObject)

hObject:

The handle for the object (as returned by the ObjectByName() function).

Return Value

A string containing the converted object handle

Related Functions

[ObjectByName](#), [CreateObject](#), [CreateControlObject](#)

See Also

[ActiveX Functions](#)

Chapter: 18 Alarm Functions

Alarm functions display alarms and their related alarm help pages, and acknowledge, disable, and enable alarms. They provide information about alarms and allow your operators to add comments to alarm records. You can also access alarms at the record level (on the alarms server) for more complex operations.

Alarm Functions

Following are functions relating to alarms:

AlarmAck	Acknowledges alarms.
AlarmAckRec	Acknowledges alarms by record number.
AlarmActive	Determines if any alarms are active in the user's area.
AlarmCatGetFormat	Returns the display format string of the specified alarm category.
AlarmClear	Clears acknowledged, inactive alarms from the active alarm list.
AlarmClearRec	Clear an alarm by its record number.
AlarmComment	Allows users to add comments to alarm summary entries.
AlarmDelete	Deletes alarm summary entries.
AlarmDisable	Disables alarms.
AlarmDisableRec	Disables alarms by record number.
AlarmDsp	Displays alarms.
AlarmDspClusterAdd	Adds a cluster to a client's alarm list.
AlarmDspClusterInUse	Determines if a cluster is included in a client's alarm list.
AlarmDspClusterRemove	Removes a cluster from a client's alarm list.

AlarmDspLast	Displays the latest, unacknowledged alarms.
AlarmDspNext	Displays the next page of alarms.
AlarmDspPrev	Displays the previous page of alarms.
AlarmEnable	Enables alarms.
AlarmEnableRec	Enables alarms by record number.
AlarmEventQue	Opens the alarm event queue.
AlarmFirstCatRec	Searches for the first occurrence of an alarm category and type.
AlarmFirstPriRec	Searches for the first occurrence of an alarm priority and type.
AlarmFirstTagRec	Searches for the first occurrence of an alarm tag, name, and description.
AlarmGetDelay	Gets the delay setting for an alarm.
AlarmGetDelayRec	Gets the delay setting for an alarm via the alarm record number.
AlarmGetDsp	Gets field data from the alarm record that is displayed at the specified AN.
AlarmGetFieldRec	Gets alarm field data from the alarm record number.
AlarmGetInfo	Gets information about an alarm list displayed at an AN.
AlarmGetOrderbyKey	Retrieves the list of key(s) used to determine the order of the alarm list.
AlarmGetThreshold	Gets the thresholds of analog alarms.
AlarmGetThresholdRec	Gets the thresholds of analog alarms by the alarm record number.
AlarmHelp	Displays the help page for the alarm where the cursor is positioned.
AlarmNextCatRec	Searches for the next occurrence of an alarm category and type.

AlarmNextPriRec	Searches for the next occurrence of an alarm priority and type.
AlarmNextTagRec	Searches for the next occurrence of an alarm tag, name, and description.
AlarmNotifyVarChange	Activates a time-stamped digital or time-stamped analog alarm
AlarmQueryFirstRec	Searches for the first occurrence of an alarm category (or priority) and type.
AlarmQueryNextRec	Searches for the next occurrence of an alarm category (or priority) and type.
AlarmSetDelay	Changes the delay setting for an alarm.
AlarmSetDelayRec	Changes the delay set for an alarm via the alarm record number.
AlarmSetInfo	Changes the display parameters for the alarm list displayed at an AN.
AlarmSetQuery	Specifies a query .to be used in selecting alarms for display.
AlarmSetThreshold	Changes the thresholds of analog alarms.
AlarmSetThresholdRec	Changes the thresholds of analog alarms by the alarm record number.
AlarmSplit	Duplicates an alarm summary entry where the cursor is positioned.
AlarmSumAppend	Appends a new blank record to the alarm summary.
AlarmSumCommit	Commits the alarm summary record to the alarm summary device.
AlarmSumDelete	Deletes alarm summary entries.
AlarmSumFind	Finds an alarm summary index for an alarm record and alarm on time.
AlarmSumFirst	Gets the oldest alarm summary entry.
AlarmSumGet	Gets field information from an alarm summary entry.

AlarmSumLast	Gets the latest alarm summary entry.
AlarmSumNext	Gets the next alarm summary entry.
AlarmSumPrev	Gets the previous alarm summary entry.
AlarmSumSet	Sets field information in an alarm summary entry.
AlarmSumSplit	Duplicates an alarm summary entry.
AlarmSumType	Retrieves a value that indicates a specified alarm's type.
AlmSummaryAck	Acknowledges the alarm at the current cursor position in an active data browse session.
AlmSummaryClear	Clears the alarm at the current cursor position in an active data browse session.
AlmSummaryClose	Closes an alarm summary browse session.
AlmSummaryCommit	Commits the alarm summary record to the alarm summary device.
AlmSummaryDelete	Deletes alarm summary entries from the browse session.
AlmSummaryDeleteAll	Deletes all alarm summary entries from the browse session.
AlmSummaryDisable	Disables the alarm at the current cursor position in an active data browse session.
AlmSummaryEnable	Enables the alarm at the current cursor position in an active data browse session.
AlmSummaryFirst	Gets the oldest alarm summary entry.
AlmSummaryGetField	Gets the field indicated by the cursor position in the browse session.
AlmSummaryLast	Places the data browse cursor at the latest summary record from the last cluster of the available browsing cluster list.
AlmSummaryNext	Gets the next alarm summary entry in the browse session.
AlmSummaryOpen	Opens an alarm summary browse session.
AlmSummaryPrev	Gets the previous alarm summary entry in the browse ses-

	sion.
Alm-SummarySetFieldValue	Sets the value of the field indicated by the cursor position in the browse session.
AlmTagsAck	Acknowledges the alarm tag at the current cursor position in an active data browse session.
AlmTagsClear	Clears the alarm tag at the current cursor position in an active data browse session.
AlmTagsClose	Closes an alarm tags browse session.
AlmTagsDisable	Disables the alarm tag at the current cursor position in an active data browse session.
AlmTagsEnable	Enables the alarm tag at the current cursor position in an active data browse session.
AlmTagsFirst	Gets the oldest alarm tags entry.
AlmTagsGetField	Gets the field indicated by the cursor position in the browse session.
AlmTagsNext	Gets the next alarm tags entry in the browse session.
AlmTagsNumRecords	Returns the number of records in the current browse session.
AlmTagsOpen	Opens an alarm tags browse session.
AlmTagsPrev	Gets the previous alarm tags entry in the browse session.
HwAlarmQue	Returns the handle of the hardware alarm queue.

See Also[Functions Reference](#)**AlarmAck**

Acknowledges alarms. You can acknowledge the alarm where the cursor is positioned, one or more alarm lists on the active page, a whole category of alarms, or alarms of a particular priority.

This command takes the currently logged in user into account. In other words, only the alarms that the user can see are acknowledged.

You would normally call this function from a keyboard command. No action is taken if the specified alarms have already been acknowledged.

Syntax

AlarmAck(*Mode, Value [, ClusterName]*)

Mode:

The type of acknowledgment:

0 - Acknowledge a single alarm where the cursor is positioned. Set Value to 0 (zero) - it is not used.

1 - Acknowledge a page of alarms. AN alarm page can contain more than one alarm list:

- Set Value to the AN where the alarm list is displayed.
- Set Value to 0 to acknowledge the (displayed) alarm list (on the active page) where the cursor is positioned.
- Set Value to -1 to acknowledge the (displayed) alarm lists on the active page.

2 - Acknowledge a category of alarms:

- Set Value to the alarm category (0 to 16375) of the alarms to be acknowledged. Please be aware that Alarm category 0 indicates all categories; alarm category 255 indicates hardware alarms.
- Set Value to the group number to acknowledge a group of categories.

3 - Acknowledge alarms of a specific priority.

- Set *Value* to the alarm priority (0-255) of the alarms to be acknowledged. Alarm priority 0 indicates all priorities.
Hardware alarms are not affected by priority.
Set *Value* to the group handle to acknowledge a group of alarms of different priorities.

Value:

Used with Mode 1 and 2 to specify which alarms to acknowledge.

ClusterName:

Used with Mode 2 or 3 to specify the name of the cluster in which the alarms being acknowledged reside. This argument is optional if the client is connected to only one cluster containing an Alarm Server or are resolving the alarm server via the current cluster context.

This argument is not required where:

- the *mode* is 2 and the value is 255 (hardware alarm category).

This argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) code will return

Note: In some cases an error code is not returned. This function is non-blocking, and as such, any error that is detected when the alarm server processes the command will not be returned.

Related Functions

[GrpOpen](#)

Example

System Keyboard	
Key Sequence	LeftButton
Command	AlarmAck(0, 0)
Comment	Acknowledge the alarm where the cursor is positioned
System Keyboard	
Key Sequence	ShiftLeftButton
Command	AlarmAck(1, -1)
Comment	Acknowledge a page of alarms
System Keyboard	
Key Sequence	AlarmAck # ## Enter
Command	AlarmAck(2, Arg1, "clusterXYZ")
Comment	Acknowledge alarms of a specified category in cluster XYZ
System Keyboard	
Key Sequence	AckPri ##### ###### Enter
Command	AlarmAck(3,Arg1, "clusterXYZ")
Comment	Acknowledge alarms of a specific priority in cluster XYZ

```
! Acknowledge alarms of the specified group of categories.  
FUNCTION  
AckGrp(STRING CategoryGroup)  
    INT hGrp;  
    hGrp=GrpOpen("CatGroup",1);  
    StrToGrp(hGrp,CategoryGroup);  
    AlarmAck(2,hGrp, "clusterXYZ");  
    GrpClose(hGrp);  
END
```

See Also

[Alarm Functions](#)

AlarmAckRec

Acknowledges alarms by record number on both the Primary and Standby Alarm Servers. This function can be called from Alarm Server or Client and should not be used with a [MsgRPC\(\)](#) call to the Alarm Server.

Syntax

AlarmAckRec(*Record* [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- **AlarmFirstCatRec()** or **AlarmNextCatRec()**: used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- **AlarmFirstPriRec()** or **AlarmNextPriRec()**: used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- **AlarmFirstTagRec()** or **AlarmNextTagRec()**: used to search for a record by alarm tag, name, and description.
- **AlarmGetDsp()**: used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the sField argument in **AlarmGetDsp()** to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmFirstCatRec](#), [AlarmFirstTagRec](#), [AlarmNextTagRec](#), [AlarmGetDelayRec](#), [MsgRPC](#)

Example

```
/* Acknowledge all unacknowledged (Type 1) alarms of the specified
alarm category. */
FUNCTION
AutoAccept(INT Category)
    INT Current;
    INT Next;
    Current=AlarmFirstCatRec(Category,1);
    WHILE Current<>-1 DO
        Next=AlarmNextCatRec(Current,Category,1);
        AlarmAckRec(Current);
        Current=Next;
    END
END
```

See Also

[Alarm Functions](#)

AlarmActive

Determines if any alarms are active in the user's area. Call this function from the Page Strings database, to display an alarm message at a specified AN on a graphics page. You can specify the type of alarms, for example, active hardware alarms or disabled non-hardware alarms.

Syntax

AlarmActive(*Type* [, *ClusterName*])

Type:

The type of alarms to check:

Non-hardware alarms

- 0 - Active alarms
- 1 - Unacknowledged alarms, ON and OFF
- 2 - Highest priority unacknowledged alarm
- 3 - Disabled alarms

Hardware alarms

- 5 - Active alarms
- 6 - Unacknowledged alarms, ON and OFF

ClusterName:

The name of the cluster to check for active alarms. If this argument is blank or empty, the function will check the connected clusters.

Return Value

- 1 or 0 for Non-hardware alarms (modes 0, 1, and 3).
- The priority of the highest priority unacknowledged alarm (mode 2).
- The number of active alarms for Hardware alarms (modes 5 and 6).

Example

Strings	
AN	9
Expression	AlarmActive(5)
True Text	"Hardware Alarms Active"
False Text	"No Active Hardware Alarms"
Comment	Display the alarm status at AN 9

See Also

[Alarm Functions](#)

AlarmCatGetFormat

Returns the display format string of the specified alarm category.

Syntax

AlarmCatGetFormat(*Category* [*Type*])

Category:

The alarm category.

Type:

The type of display format string:

- 0 - Alarm format. Default value.
- 1 - Summary format.

Return Value

The display format string of the specified category. If the alarm category is not specifically defined or it has no format string specified in your project, the format string of category 0 will be returned.

Example

```
sFormat = AlarmCatGetFormat(0, 0);
! sFormat is assigned to the format string as defined in the Alarm Format field of
the Alarm Categories form for category 0 in your project.
```

See Also

[Alarm Functions](#)

AlarmClear

Clears an acknowledged (and off) alarm from the active alarm list. You can clear the alarm where the cursor is positioned, one or more alarm lists on the active page, a whole category of alarms, or alarms of a particular priority.

If you set the [Alarm]AckHold parameter to 1, alarms that go off and have been acknowledged are not removed from the active list until this function is called.

Syntax

AlarmClear(*Mode*, *Value* [, *ClusterName*])

Mode:

The type of clear:

0 - Clear a single alarm where the cursor is positioned:

- Set *Value* to 0 (zero) - it is not used.

1 - Clear a page of alarms. AN alarm page can contain more than one alarm list:

- Set *Value* to the AN where the alarm list is displayed.
- Set *Value* to 0 to clear the (displayed) alarm list (on the active page) where the cursor is positioned.
- Set *Value* to -1 to clear every (displayed) alarm list on the active page.

2 - Clear a category of alarms:

- Set *Value* to the alarm category (0 to 16375) of the alarms to clear. Please be aware that alarm category 0 indicates all categories; alarm category 255 indicates hardware alarms.
- Set *Value* to the group number to clear a group of categories.

3 - Clear alarms of a specific priority.

- Set *Value* to the alarm priority (0-255) of the alarms to be cleared. Alarm priority 0 indicates all priorities. Hardware alarms are not affected by priority. Set *Value* to the group handle to clear a group of alarms of different priorities.

Value:

Used with Mode 1 or 2 to specify which alarms to clear.

ClusterName:

Used with Mode 2 or 3 to specify the name of the cluster in which the alarms being cleared reside. This argument is optional if the client is connected to only one cluster containing an Alarm Server or you are resolving the alarm server via the current cluster context.

This argument is not required where:

- the *mode* is 2 and the value is 255 (hardware alarm category).

This argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmAck](#)

Example

System Keyboard	
Key Sequence	Clear
Command	AlarmClear(0, 0)
Comment	Clear the alarm where the cursor is positioned
System Keyboard	
Key Sequence	ClearAll
Command	AlarmClear(1, -1)
Comment	Clear a page of alarms

System Keyboard	
Key Sequence	AlarmClear # ## Enter
Command	AlarmClear(2, Arg1, "clusterXYZ")
Comment	Clear alarms of a specified category in cluster XYZ
System Keyboard	
Key Sequence	CtrlClear
Command	AlarmClear(2, 0, "clusterXYZ")
Comment	Clear categories of inactive alarms in cluster XYZ
System Keyboard	
Key Sequence	ClearPri ##### Enter
Command	AlarmClear(3,Arg1, "clusterXYZ")
Comment	Clear alarms of a specific priority in cluster XYZ

See Also

[Alarm Functions](#)

AlarmClearRec

Clears an alarm by its record number on both the Primary and Standby Alarms Servers. This function can be called from Alarm Server or Client and should not be used with a [MsgRPC\(\)](#) call to the Alarm Server.

Syntax

AlarmClearRec(*Record* [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).

- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.
- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the `sField` argument in `AlarmGetDsp()` to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmFirstCatRec](#), [AlarmAck](#), [AlarmFirstTagRec](#), [AlarmNextTagRec](#), [AlarmGetDelayRec](#), [MsgRPC](#)

See Also

[Alarm Functions](#)

AlarmComment

Allows an operator to add a comment to a selected alarm summary entry during run-time. You would normally call this function from a keyboard command.

Comments can only be added to alarm summary (Alarm Type 10) alarms.

Syntax

AlarmComment(*sComment*)

sComment:

The comment to add to the alarm summary entry. Comments have a maximum of 128 characters.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDsp](#)

Example

System Keyboard	
Key Sequence	Com ##### Enter
Command	AlarmComment(Arg1)
Comment	Add an alarm comment to the alarm where the cursor is positioned

See Also

[Alarm Functions](#)

AlarmDelete

Deletes alarm summary entries that are currently displayed. You can delete the alarm where the cursor is positioned, one or more alarm lists on the active page, a whole category of alarms, or alarms of a particular priority.

You would normally call this function from a keyboard command.

Syntax

AlarmDelete(*Mode*, *Value* [, *ClusterName*])

Mode:

The type of deletion:

0 - Delete a single alarm where the cursor is positioned.

- Set *Value* to 0 (zero) - it is not used.

1 - Delete a page of alarms. AN alarm page can contain more than one alarm list:

- Set *Value* to the AN where the alarm list is displayed.
- Set *Value* to 0 to delete the (displayed) alarm list (on the active page) where the cursor is positioned.
- Set *Value* to -1 to delete every (displayed) alarm list on the active page.

2 - Delete a category of alarms.

- Set *Value* to the alarm category (0-16375) of the alarms to delete. Please be aware that alarm category 0 indicates all categories; alarm category 255 indicates hardware alarms.

- Set *Value* to the group number to delete a group of categories.

3 - Delete alarms of a specific priority.

- Set *Value* to the alarm priority (0-255) of the alarms to be deleted.

Alarm priority 0 indicates all priorities. Hardware alarms are not affected by priority. Set *Value* to the group handle to delete a group of alarms of different priorities.

Value:

Used with Mode 1 or 2 to specify which alarms to delete.

ClusterName:

Used with Mode 2 or 3 to specify the name of the cluster in which the alarms being deleted reside. This argument is optional if the client is connected to only one cluster containing an Alarm Server or you are resolving the alarm server via the current cluster context. This argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#)

Example

System Keyboard	
Key Sequence	DelSum
Command	AlarmDelete(0, 0)
Comment	Delete the alarm summary entry where the cursor is positioned
System Keyboard	
Key Sequence	ShiftDelSum
Command	AlarmDelete(1, -1)
Comment	Delete a page of alarm summary entries
System Keyboard	
Key Sequence	SumDelete # ## Enter

Command	AlarmDelete(2, Arg1, "clusterXYZ")
Comment	Delete alarm summary entries of a specified category in cluster XYZ
System Keyboard	
Key Sequence	DelSum ##### Enter
Command	AlarmDelete(3,Arg1, "clusterXYZ")
Comment	Delete alarm summary entries of a specified priority in cluster XYZ

See Also

[Alarm Functions](#)

AlarmDisable

Disables alarms. You can disable the alarm where the cursor is positioned, one or more alarm lists on the active page, a whole category of alarms, or alarms of a particular priority.

You would normally call this function from a keyboard command. No action is taken if the alarms are already disabled. Use the `AlarmEnable()` function to re-enable an alarm.

After you disable an alarm, it does not display on the alarm page, alarm summary page, or alarm log. If you set the `[Alarm]DisplayDisable` parameter to 1, logging of disabled alarms continues, but the disabled alarms are not displayed on the alarm display or alarm summary pages.

Syntax

AlarmDisable(*Mode*, *Value* [, *ClusterName*])

Mode:

The type of disable:

0 - Disable a single alarm where the cursor is positioned.

- Set *Value* to 0 (zero) - it is not used.

1 - Disable a page of alarms. AN alarm page can contain more than one alarm list:

- Set *Value* to the AN where the alarm list is displayed.
- Set *Value* to 0 to disable the (displayed) alarm list (on the active page) where the cursor is positioned.
- Set *Value* to -1 to disable the (displayed) alarm list on the active page.

2 - Disable a category of alarms.

- Set *Value* to the alarm category (0-16375) of the alarms to be disabled. Please be aware that alarm category 0 indicates all categories; alarm category 255 indicates hardware alarms.
- Set *Value* to the group number to disable a group of categories.

3 - Disable alarms of a specific priority.

- Set *Value* to the alarm priority (0-255) of the alarms to be disabled.

Alarm priority 0 indicates all priorities. Hardware alarms are not affected by priority. Set *Value* to the group handle to disable a group of alarms of different priorities.

Value:

Used with Mode 1 and 2 to specify which alarms to disable.

ClusterName:

Used with Mode 2 or 3 to specify the name of the cluster where the alarms being disabled reside in. This argument is optional if the client is connected to only one cluster containing an Alarm Server or are resolving the alarm server via the current cluster context.

This argument is not required where:

- the *mode* is 2 and the value is 255 (hardware alarm category).

This argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#), [AlarmEnable](#), [AlarmDisableRec](#)

Example

System Keyboard	
Key Sequence	Disable
Command	AlarmDisable(0, 0)
Comment	Disable the alarm where the cursor is positioned
System Keyboard	

Key Sequence	ShiftDisable
Command	AlarmDisable(1, -1)
Comment	Disable a page of alarms
System Keyboard	
Key Sequence	AlarmDisable # ## Enter
Command	AlarmDisable(2, Arg1, "clusterXYZ")
Comment	Disable alarms of a specified category in cluster XYZ
System Keyboard	
Key Sequence	DisPri ##### Enter
Command	AlarmDisable(3,Arg1,"clusterXYZ")
Comment	Disable alarms of a specific priority in cluster XYZ

See Also

[Alarm Functions](#)

AlarmDisableRec

Disables alarms by record number on both the Primary and Standby Alarms Servers. This function can be called from Alarm Server or Client and should not be used with a [MsgRPC\(\)](#) call to the Alarm Server.

Syntax

AlarmDisableRec(*Record* [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.

- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the `sField` argument in `AlarmGetDsp()` to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmFirstTagRec](#), [AlarmNextTagRec](#), [AlarmDisable](#), [MsgRPC](#)

Example

```
/* Disable/enable the specified "Pump" alarm. Flag determines
whether the alarm is disabled (Flag=0) or enabled (Flag=1). */
FUNCTION
DisablePumps(STRING sTag, INT Flag)
    INT Current;
    INT Next;
    Current=AlarmFirstTagRec(sTag,"Pump","");
    WHILE Current<>-1 DO
        Next=AlarmNextTagRec(Current,sTag,"Pump","");
        IF Flag=0 THEN
            AlarmDisableRec(Current);
        ELSE
            AlarmEnableRec(Current);
        END
        Current=Next;
    END
END
```

See Also

[Alarm Functions](#)

AlarmDsp

Displays an alarm list, starting at a specified AN and then on subsequent ANs. You specify the number of alarms to display, the type of alarms and the name of the cluster the alarms belong to, for example, active hardware alarms or disabled non-hardware alarms in cluster XYZ. Before you call this function, you need to first add animation points to the graphics page for each alarm to be displayed.

If you only need to display the standard alarm page, use the [PageAlarm](#) function - it uses this `AlarmDsp()` function to display alarms. If you need more control over the display of alarms you can use this function, but only to display alarms on the alarm page. Use the [AlarmDspLast](#) function to display alarms on another graphics page (it uses less memory).

Syntax

AlarmDsp(*AN*, *Count* [, *Type*] [, *ClusterName*] [, *iNoDraw*] [, *sCallbackFunc*])

AN:

The AN where the first alarm is to display.

Note: The [Animator]MaxAn citect ini parameter sets the maximum AN which `AlarmDsp` will work with.

Count:

The number of alarms to display.

Type:

The type of alarms to display:

Non-hardware alarms

- 0 - All active alarms, that is Types 1 and 2
- 1 - All unacknowledged alarms, ON and OFF
- 2 - All acknowledged ON alarms
- 3 - All disabled alarms
- 4 - All configured (non-hardware) alarms, that is Types 0 to 3, plus acknowledged OFF alarms.

Hardware alarms

- 5 - All active alarms, that is Types 6 and 7
- 6 - All unacknowledged alarms, ON and OFF
- 7 - All acknowledged ON alarms
- 8 - All disabled alarms
- 9 - All configured alarms, that is Types 5 to 8

Alarm Summary

- 10 - All summary alarms

Alarm General

- 11 - All ON alarms
- 12 - All OFF alarms
- 13 - All ON hardware alarms
- 14 - All OFF hardware alarms

If you omit the Type, the default is 0.

ClusterName:

The cluster name to which the alarms belong. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

If the client is connected to only one cluster containing an Alarms Server then this argument is optional, the list returned will be limited to alarms within this cluster.

If the client is connected to clusters containing more than one Alarms Server then the Cluster Name needs to be specified. If a cluster name is not specified, alarms are returned for all clusters.

iNoDraw:

Makes call to Alarm Server to update the ALMCB but does not automatically perform the animation of the data when the result is returned.

sCallbackFunc:

Callback function to associate with the return of the ALMCB data from the Alarm Server.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDspNext](#), [AlarmDspPrev](#), [AlarmDspLast](#), [AlarmGetInfo](#), [PageAlarm](#)

Example

Advanced Animation	
Command	AlarmDsp(20,15,3)
Comment	Display 15 disabled alarms at AN 20

See Also

[Alarm Functions](#)

AlarmDspClusterAdd

Adds a cluster to a client's alarm list. Alarms in the specified cluster (that correspond to the mode set in [AlarmDsp](#)) will be added to the alarm list at the AN number.

Syntax

AlarmDspClusterAdd(AN, ClusterName)

AN:

The AN used in the original AlarmDsp() call.

ClusterName:

The name of the cluster to be used for this alarm list. The argument is enclosed in quotation marks ("").

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDspClusterRemove](#) [AlarmDSPClusterInUse](#) [AlarmDsp](#)

See Also

[Alarm Functions](#)

AlarmDspClusterInUse

Determines if a cluster is included in a client's alarm list.

Syntax

AlarmDspClusterInUse(AN, ClusterName)

AN:

The AN used in the original AlarmDsp() call.

ClusterName:

The name of the cluster to query an alarm list for to determine if it's included. The argument is enclosed in quotation marks ("").

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDspClusterAdd](#), [AlarmDSPClusterRemove](#), [AlarmDsp](#)

See Also

[Alarm Functions](#)

AlarmDspClusterRemove

Removes a cluster from a client's alarm list. Alarms for the specified cluster will be removed from the alarm list at the AN number.

Syntax

AlarmDspClusterRemove(*AN*, *ClusterName*)

AN:

The AN used in the original AlarmDsp() call.

ClusterName:

The name of the cluster to remove from this alarm list. The argument is enclosed in quotation marks ("").

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDspClusterAdd](#), [AlarmDSPClusterInUse](#), [AlarmDsp](#)

See Also

[Alarm Functions](#)

AlarmDspLast

Displays the latest unacknowledged alarms, at a specified AN with the cluster named. Use this function to display the last alarms. You can specify the number of alarms to display of a specified type, for example, active hardware alarms or disabled non-hardware alarms.

Syntax

AlarmDspLast(*AN* [, *Count*] [, *Type*] [, *ClusterName*] [, *iNoDraw*] [, *sCallbackFunc*])

AN:

Note: The [Animator]MaxAn citect ini parameter sets the maximum AN which AlarmDspLast will work with.

The AN where the last alarms are to be displayed.

Count:

The number of alarms to display. If you omit the Count, the default is 1.

Type:

The type of alarms to display:

Non-hardware alarms

- 0 - All active alarms, that is Types 1 and 2
- 1 - All unacknowledged alarms, ON and OFF
- 2 - All acknowledged ON alarms
- 3 - All disabled alarms
- 4 - All configured (non-hardware) alarms, that is Types 0 to 3, plus acknowledged OFF alarms.

Hardware alarms

- 5 - All active alarms, that is Types 6 and 7
- 6 - All unacknowledged alarms, ON and OFF
- 7 - All acknowledged ON alarms
- 8 - All disabled alarms
- 9 - All configured alarms, that is Types 5 to 8

Alarm Summary

- 10 - All summary alarms

Alarm General

- 11 - All ON alarms
- 12 - All OFF alarms
- 13 - All ON hardware alarms
- 14 - All OFF hardware alarms

If you omit the Type, the default is 1.

ClusterName:

The cluster name to which the alarms belong. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

If a cluster name is not specified, alarms are returned for all clusters.

iNoDraw:

Makes call to Alarm Server to update the ALMCB but does not automatically perform the animation of the data when the result is returned.

sCallbackFunc:

Callback function to associate with the return of the ALMCB data from the Alarm Server.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDsp](#)

Example

Advanced Animation

Command `AlarmDspLast(11, 'ClusterXYZ')`

Comment Display the last alarm at AN 11

Advanced Animation

Command `AlarmDspLast(21,3, 'ClusterXYZ')`

Comment Display the last 3 alarms at AN 21

See Also

[Alarm Functions](#)

AlarmDspNext

Displays the next page of alarms. This function pages down (scrolls) the alarms displayed by the `AlarmDsp()` function. You would normally call this function from a keyboard command.

Syntax

`AlarmDspNext(AN)`

AN:

The AN where the alarm list is displayed, or:

-1 - Scroll every alarm list displayed on the page.

0 - Scroll the alarm list where the cursor is positioned.

Note: An alarm page can contain more than one alarm list.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDsp](#), [AlarmDspPrev](#)

Example

System Keyboard	
Key Sequence	NextAlarm
Command	AlarmDspNext(20)
Comment	Display the next page of alarms (from the alarm list) at AN20

See Also

[Alarm Functions](#)

AlarmDspPrev

Displays the previous page of alarms. This function pages up (scrolls) the alarms displayed by the AlarmDsp() function. You would normally call this function from a keyboard command.

Syntax

AlarmDspPrev(AN)

AN:

The AN where the alarm list is displayed, or:

- 1 - Scroll every alarm list displayed on the page.
- 0 - Scroll the alarm list where the cursor is positioned.

Note: An alarm page can contain more than one alarm list.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmDsp](#), [AlarmDspNext](#)

Example

System Keyboard	
Key Sequence	PrevAlarm
Command	AlarmDspPrev(20)
Comment	Display the previous page of alarms (from the alarm list) at AN20

See Also

[Alarm Functions](#)

AlarmEnable

Enables an alarm on the active alarm list. You can enable the alarm where the cursor is positioned, one or more alarm lists on the active page, a whole category of alarms, or alarms of a particular priority.

No action is taken if the alarms are already enabled. You would normally call this function from a keyboard command.

Syntax

AlarmEnable(*Mode*, *Value* [, *ClusterName*])

Mode:

The type of enable:

0 - Enable a single alarm where the cursor is positioned.

- Set *Value* to 0 (zero) - it is not used.

1 - Enable a page of alarms. AN alarm page can contain more than one alarm list:

- Set *Value* to the AN where the alarm list is displayed.
- Set *Value* to 0 to enable the (displayed) alarm list (on the active page) where the cursor is positioned.
- Set *Value* to -1 to enable every (displayed) alarm list on the active page.

2 - Enable a category of alarms.

- Set *Value* to the alarm category (0-16375) of the alarms to be enabled. Please be aware that alarm category 0 indicates all categories; alarm category 255 indicates hardware alarms.
 - Set *Value* to the group number to enable a group of categories.
- 3 - Enable alarms of a specific priority.
- Set *Value* to the alarm priority (0-255) of the alarms to be enabled. Alarm priority 0 indicates all priorities. Hardware alarms are not affected by priority. 3) Set *Value* to the group handle to enable a group of alarms of different priorities.

Value:

Used with Mode 1 and 2 to specify which alarms to enable.

ClusterName:

Used with Mode 2 or 3 to specify the name of the cluster where the alarms being enabled reside in. This argument is optional if the client is connected to only one cluster containing an Alarm Server or are resolving the alarm server via the current cluster context.

This argument is not required where:

- the *mode* is 2 and the value is 255 (hardware alarm category).

This argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#), [AlarmDisable](#), [AlarmEnableRec](#)

Example

System Keyboard	
Key Sequence	Enable
Command	AlarmEnable(0, 0)
Comment	Enable the alarm where the cursor is positioned
System Keyboard	
Key Sequence	ShiftEnable

Command	AlarmEnable(1, -1)
Comment	Enable a page of alarms
System Keyboard	
Key Sequence	AlarmEnable # ## Enter
Command	AlarmEnable(2, Arg1, "clusterXYZ")
Comment	Enable alarms of a specified category in cluster XYZ
System Keyboard	
Key Sequence	EnPri ##### Enter
Command	AlarmEnable(3,Arg1, "clusterXYZ")
Comment	Enable alarms of a specific priority in cluster XYZ

See Also

[Alarm Functions](#)

AlarmEnableRec

Enables alarms by record number on both the Primary and Standby Alarms Servers. This function can be called from Alarm Server or Client and should not be used with a [MsgRPC\(\)](#) call to the Alarm Server.

Syntax

AlarmEnableRec(*Record* [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.

- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the *sField* argument in `AlarmGetDsp()` to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmFirstTagRec](#), [AlarmNextTagRec](#), [AlarmEnable](#), [AlarmDisableRec](#), [MsgRPC](#)

Example

See [AlarmDisableRec](#)

See Also

[Alarm Functions](#)

AlarmEventQue

Opens the alarm event queue. The Alarms Server writes events into this queue as they are processed. These events include activated, reset, acknowledged, enabled and disabled alarms. To read events from this queue, use the `QueRead()` or `QuePeek()` functions. The data put into the queue is the alarm record identifier (into the **Type** field) and the alarm event format (into the **Str** field). The function puts every state change into the queue and CitectSCADA does not use this queue for anything.

To use this function, you need to enable the alarm event queue with the `[Alarm]EventQue` parameter. This parameter will tell the Alarms Server to start placing events into the queue. The `[Alarm]EventFmt` parameter defines the format of the data placed into the string field. You can enable the `EventQue` parameter without setting the event format so that the Alarms Server does not place a formatted string into the queue.

Enabling this formatting feature can increase CPU loading and reduce performance of the Alarms Server as every alarm is formatted and placed in the queue. You should reconsider using this feature if a decrease in performance is noticeable.

The maximum length of each queue is controlled by the [Code]Queue parameter. You may need to adjust this parameter so as not to miss alarm events. When the queue is full, the Alarms Server will discard events.

WARNING

UNINTENDED EQUIPMENT OPERATION

You may need to adjust the [Code]Queue parameter so as not to miss alarm events. When the queue is full, the Alarms Server will discard events.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax

AlarmEventQue()

Return Value

The handle of the alarm event queue, or -1 if the queue cannot be opened.

Related Functions

[QueRead](#), [QuePeek](#), [TagWriteEventQue](#)

Example

```
hQue = AlarmEventQue()
WHILE TRUE DO
    QueRead(hQue, nRecord, sAlarmFmt, 1);
    /* do what ever with the alarm event */
    ...
    Sleep(0);
END
```

See Also

[Alarm Functions](#)

AlarmFirstCatRec

Searches for the first occurrence of an alarm category and type. You can search all areas, the current area only, or specify an area to limit the search. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

This function returns an alarm record identifier that you can use in other alarm functions, for example, to acknowledge, disable, or enable the alarm, or to get field data on that alarm.

Note: Record numbers obtained from AlarmGetDsp are not valid for this function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmFirstCatRec(*Category*, *Type* [, *Area*] [, *ClusterName*])

Category:

The alarm category or group number to match. Set Category to 0 (zero) to match all alarm categories.

Type:

The type of alarms to find:

Non-hardware alarms

0 - All active alarms, that is Types 1 and 2.

1 - All unacknowledged alarms, ON and OFF.

2 - All acknowledged ON alarms.

3 - All disabled alarms.

4 - All configured alarms, that is, types 0 to 3, plus acknowledged OFF alarms.

If you do not specify a type, the default is 0.

Area:

The area in which to search for alarms. If you do not specify an area, or if you set Area to -1, only the current area will be searched.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[GrpOpen](#), [AlarmNextCatRec](#), [AlarmFirstPriRec](#), [AlarmNextPriRec](#), [AlarmGetFieldRec](#), [AlarmAckRec](#), [AlarmDisableRec](#), [AlarmEnableRec](#), [AlarmGetThresholdRec](#), [AlarmSetThresholdRec](#), [MsgRPC](#)

Example

See [AlarmAckRec](#)

See Also

[Alarm Functions](#)

AlarmFirstPriRec

Searches for the first occurrence of an alarm priority and type. You can search all areas, the current area only, or specify an area to limit the search. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

This function returns an alarm record identifier that you can use in other alarm functions, for example, to acknowledge, disable, or enable the alarm, or to get field data on that alarm.

Note: Record numbers obtained from [AlarmGetDsp](#) are not valid for this function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Note: This function will return a match for an Acknowledge Off alarm with [Alarm]AckHold=1 even after it has been cleared using [AlarmClear](#) or [AlarmClearRec](#).

Syntax

AlarmFirstPriRec(*Priority*, *Type* [, *Area*] [, *ClusterName*])

Priority:

The alarm Priority or group handle of a group of alarm priorities. Set Priority to 0 (zero) to match all alarm priorities.

Type:

The type of alarms to find:

Non-hardware alarms

- 0 - All active alarms, that is Types 1 and 2.
 - 1 - All unacknowledged alarms, ON and OFF.
 - 2 - All acknowledged ON alarms.
 - 3 - All disabled alarms.
 - 4 - All configured alarms, that is types 0 to 3, plus acknowledged OFF alarms.
- If you do not specify a type, the default is 0.

Area:

The area in which to search for alarms. If you do not specify an area, or if you set Area to -1, only the current area will be searched.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found. If you do not specify an area, only alarms in the current area on the alarms server are searched.

Related Functions

[GrpOpen](#), [AlarmNextCatRec](#), [AlarmFirstPriRec](#), [AlarmNextPriRec](#), [AlarmGetFieldRec](#), [AlarmAckRec](#), [AlarmDisableRec](#), [AlarmEnableRec](#), [AlarmGetThresholdRec](#), [AlarmSetThresholdRec](#), [MsgRPC](#)

Example

```
/* Acknowledge all unacknowledged (Type 1) alarms of the specified
alarm priority. */
FUNCTION
AutoAccept(INT iPriority)
    INT iCurrent;
    INT iNext;
    iCurrent=AlarmFirstPriRec(iPriority,1,-1);
    WHILE iCurrent <>-1 DO
        iNext=AlarmNextPriRec(iCurrent,iPriority,1,-1);
        AlarmAckRec(iCurrent);
        iCurrent=iNext;
    END
END
```

See Also

[Alarm Functions](#)

[AlarmFirstTagRec](#)

Searches for the first occurrence of an alarm tag, name, and description. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Note: Record numbers obtained from `AlarmGetDsp` are not valid for this function.

This function returns an alarm record identifier that you can use in other alarm functions, for example, to acknowledge, disable, or enable the alarm, or to get field data on that alarm.

Note: This function will return a match for an Acknowledge Off alarm with `[Alarm]AckHold=1` even after it has been cleared using `AlarmClear` or `AlarmClearRec`.

Syntax

`AlarmFirstTagRec(Tag, Name, Description [, ClusterName])`

Tag:

The alarm tag to be matched. Specify an empty string (" ") to match all alarm tags.

Name:

The alarm name to be matched. Specify an empty string (" ") to match all alarm names.

Description:

The alarm description to be matched. Specify an empty string (" ") to match all alarm descriptions.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[AlarmNextTagRec](#), [AlarmGetFieldRec](#), [AlarmAckRec](#), [AlarmDisableRec](#), [AlarmEnableRec](#), [AlarmGetThresholdRec](#), [AlarmSetThresholdRec](#), [MsgRPC](#)

Example

See [AlarmDisableRec](#)

See Also

[Alarm Functions](#)

AlarmGetDelay

Gets the delay setting for the alarm the cursor is currently positioned over.

Syntax

AlarmGetDelay(*Type*)

Type:

The type of delay:

- 0 - Delay (digital alarm/advancedalarm)
- 1 - High high delay (analog alarm)
- 2 - High delay (analog alarm)
- 3 - Low delay (analog alarm)
- 4 - Low low delay (analog alarm)
- 5 - Deviation delay (analog alarm)

Return Value

The alarm delay if successful, otherwise -1 is returned. Use IsError() to retrieve extended [error](#) information.

Related Functions

[AlarmNotifyVarChange](#), [AlarmSetDelayRec](#), [AlarmGetDelayRec](#)

See Also

[Alarm Functions](#)

AlarmGetDelayRec

Gets the delay setting for an alarm via the alarm record number.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmGetDelayRec(Record, Type [, ClusterName])

Record:

The alarm record number, returned from any of the following alarm functions:

- AlarmFirstCatRec() or AlarmNextCatRec() - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- AlarmFirstPriRec() or AlarmNextPriRec() - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- AlarmFirstTagRec() or AlarmNextTagRec() - used to search for a record by alarm tag, name, and description.
- AlarmGetDsp() - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the sField argument in AlarmGetDsp() to "RecNo".

Type:

The type of delay:

- 0 - Delay (digital alarm/advancedalarm)
- 1 - High high delay (analog alarm)
- 2 - High delay (analog alarm)
- 3 - Low delay (analog alarm)
- 4 - Low low delay (analog alarm)
- 5 - Deviation delay (analog alarm)

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm delay if successful, otherwise -1 is returned. Use IsError() to retrieve extended [error](#) information.

Related Functions

[AlarmNotifyVarChange](#), [AlarmSetDelayRec](#), [AlarmGetDelay](#)

See Also

[Alarm Functions](#)

AlarmGetDsp

Gets field data from the alarm record that is displayed at the specified AN. You can use this function for both Alarm Pages and Alarm Summaries (an Alarm Page or Alarm Summary needs to be displayed before this function can be used).

You can call this function on an Alarms Server or a client to get the contents of any field in the alarm record at that AN.

You can return the record number of the alarm record for use in other alarm functions, for example, to acknowledge, disable, or enable an alarm (on an Alarms Server).

The `AlarmGetDsp()` function does not support hardware alarms.

Syntax

`AlarmGetDsp(AN, sField)`

AN:

AN number of an ALMCB Alarm record. Equal to AN where actual ALMCB resides + Offset into the list of ALMCB records.

sField:

The name of the field from which the data is retrieved. The contents of the following fields can be retrieved when the Alarm Page is displayed:

Field	Description
Area	The area to which the alarm belongs. The user needs to have access to this area to access this alarm data.
AlmComment	The text entered into the Comment field of the alarm properties dialog.
Category	Alarm category
Comment	Operator comments attached to the Alarm Log entry (if any)
Custom1..8	Custom Filter Fields
Date	The date that the alarm changed state (mm/dd/yyyy)
DateExt	The date that the alarm changed state in extended format
Deadband	Deadband (Only Valid on Analog Alarms)
Deviation	Deviation Alarm trigger value (Only Valid on Analog Alarms)
Desc	Alarm description
Font	Font of alarm.
Format	Format of alarm.
High	High Alarm trigger value (Only Valid on Analog Alarms)
HighHigh	High High Alarm trigger value (Only Valid on Analog Alarms)

Field	Description
Help	Alarm help page
LogState	The last state that the alarm passed through
Low	Low Alarm trigger value (Only Valid on Analog Alarms)
LowLow	Low Low Alarm trigger value (Only Valid on Analog Alarms)
Name	Alarm name
Priority	The alarm priority
Rate	Rate of change trigger value (Only Valid on Analog Alarms)
RecNo	The alarm record number
State	The current state of the alarm
State_desc	The configured description (for example, healthy or stopped) of a particular state
Tag	Alarm tag
Time	The time that the alarm changed state (hh:mm:ss)
Type	The type of alarm or condition
Value	The current value of the alarm variable

The contents of the any of the above fields (except for State) and the following fields can be retrieved when the Alarm Summary is displayed:

Field	Description
UserName	The name of the user (User Name) who was logged on and performed some action on the alarm (for example, acknowledging the alarm or disabling the alarm, etc.). Be aware that when the alarm is first activated, the user name is set to "system" (because the operator did not trip the alarm).
FullName	The full name of the user (Full Name) who was logged on and performed some action on the alarm (for example, acknowledging the alarm or disabling the alarm, etc.). Be aware that when the alarm is first activated, the full name is set to "system" (because the operator did not trip the alarm).
UserDesc	The text related to the user event
OnDate	The date when alarm was activated
OnDateExt	The date (in extended format) when the alarm was activated (dd/mm/yyyy)
OffDate	The date when the alarm returned to its normal state
OffDateExt	The date (in extended format) when the alarm returned to its normal state (dd/mm/yyyy)

Field	Description
OnTime	The time when the alarm was activated
OffTime	The time when the alarm returned to its normal state
DeltaTime	The time difference between OnDate/OnTime and OffDate/OffTime, in seconds
OnMilli	Adds milliseconds to the time the alarm was activated.
OffMilli	Adds milliseconds to the time the alarm returned to its normal state.
AckTime	The time when the alarm was acknowledged
AckDate	The date when the alarm was acknowledged
AckDateExt	The date (in extended format) when the alarm was acknowledged (dd/mm/yyyy)
SumState	Describes the state of the alarm when it occurred
SumDesc	A description of the alarm summary
Native_SumDesc	A description of the alarm summary, in the native language
Native_Comment	Native language comments the operator adds to an Alarm Summary entry during runtime.

Return Value

Field data from the alarm entry (as a string). Current font handle for alarm record depending on state.

Related Functions

[AlarmDsp](#)

Example

```

! Display the tag and category for the alarm at the specified AN.
FUNCTION
AlarmData(INT AN)
    STRING Category;
    STRING Tag;
    Category=AlarmGetDsp(AN,"Category");
    Tag=AlarmGetDsp(AN,"Tag");
    Prompt("Alarm "+Tag+" is Category "+Category);
END

```

See Also

[Alarm Functions](#)

AlarmGetFieldRec

Gets the contents of the specified field in the specified alarm record. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

Note: Record numbers obtained from AlarmGetDsp are not valid for this function. Instead use AlarmFirstTagRec() to get the record. This should be called from the server side using MsgRPC.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmGetFieldRec(Record, sField [, nVer] [, ClusterName])

Record:

The alarm record number, returned from any of the following alarm functions:

- AlarmFirstCatRec() or AlarmNextCatRec() - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- AlarmFirstPriRec() or AlarmNextPriRec() - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- AlarmFirstTagRec() or AlarmNextTagRec() - used to search for a record by alarm tag, name, and description.

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

sField:

The name of the field from which the data is retrieved.

Field	Description
Category	Alarm category
Desc	Alarm description
Help	Alarm help page
Name	Alarm name
Tag	Alarm tag
Time	The time that the alarm changed state (hh:mm:ss)
Comment	Operator comments attached to the Alarm Log entry (if any)
Date	The date that the alarm changed state (mm/dd/yyyy)

Field	Description
DateExt	The date that the alarm changed state in extended format
Type	The type of alarm or condition
State	The current state of the alarm
Value	The current value of the alarm variable
High	High Alarm trigger value (Only Valid on Analog Alarms)
HighHigh	High High Alarm trigger value (Only Valid on Analog Alarms)
Low	Low Alarm trigger value (Only Valid on Analog Alarms)
LowLow	Low Low Alarm trigger value (Only Valid on Analog Alarms)
Rate	Rate of change trigger value (Only Valid on Analog Alarms)
Deviation	Deviation Alarm trigger value (Only Valid on Analog Alarms)
Deadband	Deadband (Only Valid on Analog Alarms)
LogState	The last state that the alarm passed through
AlmComment	The text entered into the Comment field of the alarm properties dialog.
Custom1..8	Custom Filter Fields
State_desc	The configured description (for example, healthy or stopped) of a particular state
UserName	The name of the user (User Name) who was logged on and performed some action on the alarm (for example, acknowledging the alarm or disabling the alarm, etc.). Be aware that when the alarm is first activated, the user name is set to "system" (because the operator did not trip the alarm).
FullName	The full name of the user (Full Name) who was logged on and performed some action on the alarm (for example, acknowledging the alarm or disabling the alarm, etc.). Be aware that when the alarm is first activated, the full name is set to "system" (because the operator did not trip the alarm).
UserDesc	The text related to the user event
OnDate	The date when alarm was activated
OnDateExt	The date (in extended format) when the alarm was activated (dd/mm/yyyy)
OffDate	The date when the alarm returned to its normal state
OffDateExt	The date (in extended format) when the alarm returned to its normal state (dd/mm/yyyy)
OnTime	The time when the alarm was activated
OffTime	The time when the alarm returned to its normal state

Field	Description
DeltaTime	The time difference between OnDate/OnTime and OffDate/OffTime, in seconds
OnMilli	Adds milliseconds to the time the alarm was activated.
OffMilli	Adds milliseconds to the time the alarm returned to its normal state.
AckTime	The time when the alarm was acknowledged
AckDate	The date when the alarm was acknowledged
AckDateExt	The date (in extended format) when the alarm was acknowledged (dd/mm/yyyy)
SumState	Describes the state of the alarm when it occurred
SumDesc	A description of the alarm summary
Native_SumDesc	A description of the alarm summary, in the native language
Native_Comment	Native language comments the operator adds to an Alarm Summary entry during runtime.

nVer:

The version of an alarm.

If an alarm has been triggered more than once in a given period, the version lets you distinguish between different instances of the alarm's activity.

The version is used in filtering alarms for display. A query function passes a value to this parameter in order to get field information for a particular alarm.

This parameter is not needed when you use AlarmGetFieldRec() for purposes other than filtering. It will default to 0 if omitted.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm field data (as a string).

Related Functions

[AlarmFirstTagRec](#), [AlarmNextTagRec](#), [MsgRPC](#)

Example

```

FUNCTION
GetNameFromTag(STRING sTag)
    INT record;
    STRING sName
    record = AlarmFirstTagRec(sTag, "", "");
    IF record <> -1 THEN
        sName = AlarmGetFieldRec(record, "NAME");
    ELSE
        sName = "";
    END
    RETURN sName;
END

```

See Also

[Alarm Functions](#)

AlarmGetInfo

Gets data on the alarm list displayed at a specified AN. Use this function to display the current alarm list information on an alarm page. If only one alarm list has been configured on an alarm page, modes 2 and 3 of this function return the current alarm page information.

Note: You cannot retrieve the order by key setting for an alarm list using this function, as it can only returns numeric values. To retrieve this information, use the function `AlarmGetOrderbyKey`

Syntax

AlarmGetInfo(AN, Type)

AN:

The AN where the alarm list (with the required information) is displayed. Set the AN to 0 (zero) to get information on the alarm list where the cursor is positioned.

Type:

The type of data:

0 - Alarm page number. The vertical offset (in pages) from the AN where the alarm list commenced. The alarm list need to have scrolled off the first page for this type to return a non-zero value.

1 - Alarm list offset. The vertical offset (in lines) from the AN where the alarm list commenced. You need to have scrolled off the first page of alarms for this type to return a non zero value.

- 2 - Category of alarms displayed on the alarm list. You can use a group number to display a group of categories.
- 3 - Type of alarms displayed on the alarm list. See `AlarmDsp()` for a list of these types.
- 7 - Priority of alarms displayed on the alarm list. The return value may be a group number if the alarm list contains alarms of more than one priority.
- 8 - Display mode of the alarm list.
- 9 - Sorting mode of the alarm list.
- 10 - Retrieves the error code for the last alarm summary request that was not able to be processed due to a buffer overflow. The last request error value will be reset on the next successful response from the servers.

Return Value

Alarm list data as a numeric value.

Related Functions

[AlarmDsp](#), [AlarmSetInfo](#), [AlarmGetOrderbyKey](#).

Example

```
/* In the following examples, data is returned on the alarm
list where the cursor is positioned. */
page = AlarmGetInfo(0,0);
! returns the alarm page number.
offset = AlarmGetInfo(0,1);
! returns the alarm list offset.
cat = AlarmGetInfo(0,2);
! returns the alarm category displayed.
type = AlarmGetInfo(0,3);
! returns the type of alarms displayed.
```

See Also

[Alarm Functions](#)

AlarmGetOrderbyKey

Retrieves the list of key(s) that are used to determine the order of the alarm list. These keys can be set by the `AlarmSetInfo()` function.

Syntax

AlarmGetOrderbyKey(AN)

AN:

The AN where the alarm list (with the required information) is displayed.

Return Value

Order-by key (as a string).

Example

```
page = AlarmGetOrderbyKey(21);
! returns the order-by key string of the alarm list at AN '21'.
```

See Also

[Alarm Functions](#)

AlarmGetThreshold

Gets the threshold of the analog alarm where the cursor is positioned.

Syntax

AlarmGetThreshold(*Type*)

Type:

The type of threshold:

0 - High high

1 - High

2 - Low

3 - Low low

4 - Deadband

5 - Deviation

6 - Rate of change

Return Value

The alarm threshold.

Related Functions

[AlarmGetThresholdRec](#), [AlarmSetThreshold](#), [AlarmSetThresholdRec](#)

See Also

[Alarm Functions](#)

AlarmGetThresholdRec

Gets the threshold of analog alarms by the alarm record number. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

Note: Record numbers obtained from `AlarmGetDsp` are not valid for this function. Instead use `AlarmFirstTagRec()` to get the record. This should be called from the server side using `MsgRPC`.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmGetThresholdRec(Record, Type [, ClusterName])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

Type:

The type of threshold:

- 0 - High high
- 1 - High
- 2 - Low
- 3 - Low low
- 4 - Deadband
- 5 - Deviation
- 6 - Rate of change

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm threshold.

Related Functions

[AlarmGetThreshold](#), [AlarmSetThreshold](#), [AlarmSetThresholdRec](#), [MsgRPC](#)

See Also

[Alarm Functions](#)

AlarmHelp

Displays the alarm help page (associated with the alarm) where the cursor is positioned. You can assign a help page to each alarm when you define it (using the Digital Alarms or the Analog Alarms database, depending on the type of alarm). You need to also define the help page in the Pages database.

Syntax

AlarmHelp()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageAlarm](#)

Example

System Keyboard	
Key Sequence	AlmHelp
Command	AlarmHelp()
Comment	Display the alarm help page

See Also

[Alarm Functions](#)

AlarmNextCatRec

Searches for the next occurrence of an alarm category and type, commencing with the specified alarm record identifier (returned from the previous search through the [AlarmFirstCatRec](#) function). You can search all areas, the current area only, or specify an area to limit the search. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

This function returns an alarm record identifier that you can use in other alarm functions, for example, to acknowledge, disable, or enable the alarm, or to get field data on that alarm.

Note: Record numbers obtained from `AlarmGetDsp` are not valid for this function. Instead use `AlarmFirstTagRec()` to get the record. This should be called from the server side using `MsgRPC`.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmNextCatRec(*Record*, *Category*, *Type* [, *Area*] [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.

Category:

The alarm category or group number to match. Set Category to 0 (zero) to match all alarm categories.

Type:

The type of alarms to find:

Non-hardware alarms

- 0 - Active alarms, that is Types 1 and 2.
- 1 - Unacknowledged alarms, ON and OFF.
- 2 - Acknowledged ON alarms.
- 3 - Disabled alarms.

4 - Every configured alarms, that is Types 0 to 3, plus acknowledged OFF alarms. If you choose to omit the *Type*, the default is 0.

Area:

The area in which to search for alarms. If you choose to omit the area, or if you set Area to -1, only the current area will be searched.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[GrpOpen](#), [AlarmFirstCatRec](#), [AlarmFirstPriRec](#), [AlarmNextPriRec](#), [AlarmGetFieldRec](#), [AlarmAckRec](#), [AlarmDisableRec](#), [AlarmEnableRec](#), [AlarmGetThresholdRec](#), [AlarmSetThresholdRec](#), [MsgRPC](#)

Example

See [AlarmAckRec](#)

See Also

[Alarm Functions](#)

AlarmNextPriRec

Searches for the next occurrence of an alarm of a specified priority and type, commencing with the specified alarm record identifier (returned from the previous search through the [AlarmFirstPriRec\(\)](#) function). You can search all areas, the current area only, or specify an area to limit the search. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

This function returns an alarm record identifier that you can use in other alarm functions, for example, to acknowledge, disable, or enable the alarm, or to get field data on that alarm.

Note: Record numbers obtained from [AlarmGetDsp](#) are not valid for this function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmNextPriRec(Record, Priority, Type [, Area] [, ClusterName])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.
- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the `sField` argument in `AlarmGetDsp()` to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

Priority:

The alarm Priority or group handle of a group of alarm priorities. Set Priority to 0 (zero) to match all alarm priorities.

Type:

The type of alarms to find:

Non-hardware alarms

- 0 - All active alarms, that is Types 1 and 2.
- 1 - All unacknowledged alarms, ON and OFF.
- 2 - All acknowledged ON alarms.
- 3 - All disabled alarms.
- 4 - All configured alarms, that is Types 0 to 3, plus acknowledged OFF alarms.
If you do not specify a *Type*, the default is 0.

Area:

The area in which to search for alarms. Set Area to -1 to search all areas. If you do not specify an area, only alarms in the current area on the Alarms Server are searched.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[GrpOpen](#), [AlarmFirstCatRec](#), [AlarmFirstPriRec](#), [AlarmNextCatRec](#), [AlarmGetFieldRec](#), [AlarmAckRec](#), [AlarmDisableRec](#), [AlarmEnableRec](#), [AlarmGetThresholdRec](#), [AlarmSetThresholdRec](#), [AlarmSetInfo](#), [MsgRPC](#)

See Also

[Alarm Functions](#)

AlarmNextTagRec

Searches for the next occurrence of an alarm tag, name, and description, starting with the alarm record identifier (returned from the previous search through the [AlarmFirstTagRec\(\)](#) function). If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

This function returns an alarm record identifier that you can use in other alarm functions, for example, to acknowledge, disable, or enable the alarm, or to get field data on that alarm.

Note: Record numbers obtained from `AlarmGetDsp` are not valid for this function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmNextTagRec(*Record*, *Tag*, *Name*, *Description* [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.

- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the `sField` argument in `AlarmGetDsp()` to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

Tag:

The alarm tag to be matched. Specify an empty string (" ") to match all alarm tags.

Name:

The alarm name to be matched. Specify an empty string (" ") to match all alarm names.

Description:

The alarm description to be matched. Specify an empty string (" ") to match all alarm descriptions.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[AlarmFirstTagRec](#), [AlarmGetFieldRec](#), [AlarmAckRec](#), [AlarmDisableRec](#),
[AlarmEnableRec](#), [AlarmGetDelayRec](#), [AlarmGetThresholdRec](#), [AlarmSetThresholdRec](#),
[MsgRPC](#)

Example

See [AlarmDisableRec](#).

See Also

[Alarm Functions](#)

AlarmNotifyVarChange

This function is used to provide time-stamped digital and time-stamped analog alarms with data. When called, it notifies the alarm server that the specified variable tag has changed.

The alarm server will then check all time-stamped digital and time-stamped analog alarms that use the variable tag to see if their alarm states need to be updated as a result of the change. Any alarm state changes that result from this check will be given the timestamp passed into this function as their time of occurrence.

Note: Although you can hardcode a value into the setpoint when using analog alarms, you cannot use hardcoded values with time-stamped analog alarms. If the setpoint is hardcoded, this function cannot be used to notify the alarm when the variable changes.

Syntax

AlarmNotifyVarChange(*Tag*, *Value*, *Timestamp* [, *TimestampMS*] [, *ClusterName*])

Tag:

Name of the variable tag that has changed as a string. This name may include the name of the tag's cluster in the form cluster.tagname. This cluster name may be different from the cluster of the alarm server indicated by ClusterName below.

The Tag parameter is resolved on the alarm server, so the alarm server should be configured to connect to the tag's cluster.

Value:

Value of the variable tag at the time of the change as a floating-point number

Timestamp:

Time/date at which the variable tag changed in the standard CitectSCADA time/date variable format (Seconds since 1970).

TimestampMS:

Millisecond portion of the time at which the variable tag changed.

ClusterName:

Name of the cluster of the alarm server. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The [error](#) that was detected when the function was called.

Example

```
AlarmNotifyVarChange("LOOP_1_SP", 50.0, TimeCurrent() - 10, 550,
```

```
"ClusterXYZ");  
This will tell the alarm server in cluster XYZ that the value of  
variable tag LOOP_1_SP changed to 50.0 at 9.450 seconds ago.
```

See Also

Time-stamped Digital Alarm Properties, Time-stamped Analog Alarm Properties

[Alarm Functions](#)

AlarmQueryFirstRec

Searches for the first occurrence of an alarm category (or priority) and type. This is a wrapper function of [AlarmFirstCatRec](#) and [AlarmFirstPriRec](#).

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmQueryFirstRec(*Group*, *Type*, *Area*, *QueryType* [*ClusterName*])

Group:

Alarm category if QueryType is 0 or alarm priority if QueryType is 1.

Type:

Type of alarms to find:

Non-hardware alarms

- 0 - All active alarms; that is, types 1 and 2.
- 1 - All unacknowledged alarms, ON and OFF.
- 2 - All acknowledged ON alarms.
- 3 - All disabled alarms.
- 4 - All configured alarms; that is, types 0 to 3, plus acknowledged OFF alarms.

Area:

Area in which to search for alarms. Set Area to -1 to search all areas.

QueryType:

Query type.

- 0 - Search by category.
- 1 - Search by priority.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[AlarmQueryNextRec](#)

See Also

[Alarm Functions](#)

AlarmQueryNextRec

Searches for the next occurrence of an alarm category (or priority) and type, commencing with the specified alarm record identifier (returned from the previous search through the alarm query functions).

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

This is wrapper function of [AlarmNextCatRec](#) and [AlarmNextPriRec](#).

Syntax

AlarmQueryNextRec(*Record*, *Group*, *Type*, *Area*, *QueryType* [, *ClusterName*])

Record:

Alarm record number.

Group:

Alarm Category if QueryType is 0 or alarm priority if QueryType is 1.

Type:

Type of alarms to find:

Non-hardware alarms

0 - All active alarms; that is, types 1 and 2.

1 - All unacknowledged alarms, ON and OFF.

2 - All acknowledged ON alarms.

3 - All disabled alarms.

4 - All configured alarms; that is, types 0 to 3, plus acknowledged OFF alarms.

Area:

Area in which to search for alarms. Set Area to -1 to search all areas.

QueryType:

Query type.

0 - Search by category.

1 - Search by priority.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The alarm record identifier or -1 if no match is found.

Related Functions

[AlarmQueryFirstRec](#)

See Also

[Alarm Functions](#)

AlarmSetDelay

Changes the delay setting for an alarm (that is Delay, High High Delay, Deviation Delay, etc.). This function acts on the alarm that the cursor is positioned over. Use this function during runtime to change the delay values that were specified in the alarms database. Delay changes made using this process are persistent (that is they are saved to the project).

Syntax

AlarmSetDelay(*Type*, *Value*)

Type:

The type of delay:

0 - Delay (digital alarm/advanced alarm)

1 - High high delay (analog alarm)

2 - High delay (analog alarm)

3 - Low delay (analog alarm)

4 - Low low delay (analog alarm)

5 - Deviation delay (analog alarm)

Value:

The new value for the delay. Enter a blank value " " to remove the delay setting.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmGetDelay](#), [AlarmSetDelayRec](#), [AlarmGetDelayRec](#)

See Also

[Alarm Functions](#)

AlarmSetDelayRec

Changes the delay setting for an alarm (that is Delay, High High Delay, Deviation Delay, etc.) by the alarm record number. You can only call this function on an alarms server for local alarms, or on a redundant server if one has been configured. However, a client can call this function remotely by using the MsgPRC() function.

Syntax

AlarmSetDelayRec(*Record*, *Type*, *Value*)

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.
- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the *sField* argument in `AlarmGetDsp()` to "RecNo".

Type:

The type of delay:

- 0 - Delay (digital alarm/advanced alarm)
- 1 - High high delay (analog alarm)
- 2 - High delay (analog alarm)

- 3 - Low delay (analog alarm)
- 4 - Low low delay (analog alarm)
- 5 - Deviation delay (analog alarm)

Value:

The new value for the delay. Enter a blank value " " to remove the delay setting.

Related Functions

[AlarmGetDelay](#), [AlarmNotifyVarChange](#), [AlarmGetDelayRec](#)

See Also

[Alarm Functions](#)

AlarmSetInfo

Controls different aspects of the alarm list displayed at a specified AN.

Syntax

AlarmSetInfo(AN, Type, Value)

AN:

The AN where the alarm list originally commenced. (AN alarm page can contain more than one alarm list). You can also specify:

- 1 - Change the display parameters of all alarm lists displayed on the page.
- 0 - Change the display parameters of the alarm list where the cursor is positioned.

Type:

The type of data. The aspects and related types are listed below:

Display aspect	Types
Change display line and page offset	0, 1
Formatting of alarms in the alarm list	4, 5, 6
Filtering of alarms	2, 3, 7, 8
Sorting of alarms - to control the sorting aspect of the alarm list, type 9 and 10 should be used together.	9, 10

0 - Alarm page number. The vertical offset (in pages) from the AN where the alarm list commenced.

1 - Alarm list offset. The vertical offset (in lines) from the AN where the alarm list commenced.

2 - Category of alarms displayed on the alarm list. To specify all categories use a value of 0.

You can use a group handle to display a group of categories. (A group can be defined using Groups - from the Project Editor System menu - or by using the GrpOpen() function.) Before you can display a group of categories, you need to first open the group using the GrpOpen() function. You would usually do this by entering the GrpOpen() function as the Page entry command for your alarm page (set using Page Properties). Be aware, however, that you should not close the group until you close the display page. If you do, the group will be lost and the group handle will become invalid. The page would then be unable to continue displaying the desired group. The handle may be reused for another group, which means the page may display a different category, or display all alarms.

You would normally close the group by entering the GrpClose() function as the Page exit command.

3 - Type of alarms displayed on the alarm list. See AlarmDsp() for a list of these types.

4 - Display all alarms according to the format and fonts specified for one category (specified in Value).

5 - The display format for all alarms specified by a format handle. All of the alarm categories will display in the same format.

6 - The display font for all user alarms specified by a font handle. All of the user alarms will appear in the same font and color.

7 - The priority of the alarms to be displayed in the alarm list. You can use a group number to display a group of priorities.

You can use a group handle to display a group of priorities. (A group can be defined using Groups - from the Project Editor System menu - or by using the GrpOpen() function.) Before you can display a group of priorities, you need to first open the group using the GrpOpen() function. You would usually do this by entering the GrpOpen() function as the Page entry command for your alarm page (set using Page Properties). Be aware, however, that you should not close the group until you close the display page. If you do, the group will be lost and the group handle will become invalid. The page would then be unable to continue displaying the desired group. You would normally close the group by entering the GrpClose() function as the Page exit command.

- 8 - Use the Value argument of the AlarmSetInfo() function to specify whether the display mode of the alarm list is based on Alarm Category or Priority:
 - Set the Value argument to 0 (zero) to display by Category.
 - Set the Value argument to 1 to display by Priority.
- 9 - Use the Value argument of the AlarmSetInfo() function to specify the sorting mode of the alarm list:
 - Set the Value argument to 0 (zero) to display alarms sorted by ON time within their groups.
 - Set the Value argument to 1 to display alarms sorted by the order-by keys. Please be aware that this option will only be meaningful if you have already called the AlarmSetInfo() function with a Type of 10 to set the order-by keys.
- 10 - Use the Alarm Order-by key specified in the Value argument of the AlarmSetInfo() function to determine the order in which the alarm list will be displayed.

The AlarmSetInfo() function should then be called again using a Type of 9 and a Value of 1 for CitectSCADA to sort the alarms in the order specified.

Be aware that you cannot sort Summary Alarms using this mode. Summary Alarms can only be sorted with the citect.ini settings [\[Alarm\]SummarySort](#) and [\[Alarm\]SummarySortMode](#).

Value:

The meaning of the Value argument depends on the data type specified in the Type argument.

- If you set *Type* = 8, the *Value* argument determines whether alarms are displayed by category or priority:
 - 0 - Alarm list displayed by Category.
 - 1 - Alarm list displayed by Priority.
- If you set *Type* = 10, the *Value* argument specifies the order-by keys to be used in sorting. Up to sixteen keys may be specified:

`{KeyName [SortDirection]}[{KeyName [SortDirection]}]`

The Keyname argument specifies the name of the pre-defined order-by key to be used. The valid options are a subset of the alarm display fields: Tag, Name, Category, Priority, Area, Priv, Time, State.

The SortDirection argument is optional, and indicates whether the sort will be ascending or descending. Valid options are: 0 Descending (default), 1 Ascending.

For example:

```
{Time,0} : sorts by <Time> (descending)  
{Tag,1} : sorts by <Tag> (ascending)  
{Tag,1}{Time} : sorts by <Tag> (ascending), then <Time> (descending)
```

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#), [AlarmDsp](#), [AlarmGetInfo](#)

Examples

In the following example, the alarm list is set to display in the order of the order-by key. Please be aware that this is a two-step process requiring two calls to the `AlarmSetInfo()` function, and that it applies only to non-hardware alarm lists.

```
! Set the order-by key.
AlarmSetInfo(21,10,"{Time}");
! Set the sorting mode.
AlarmSetInfo(21,9,1);
```

Type 8 of the function is used to set the display mode to either category or priority. This is helpful when filtering based on either of these fields. So In order to filter on category 2 we should use:

```
AlarmSetInfo(21, 8, 0);
AlarmSetInfo(21, 2, 2);
```

Once we do this the alarms with category 2 will be displayed in the alarm list and remaining although active will not be displayed.

Similarly if we want to filter on priority we set the mode to priority and then use type 7. For example to filter on priority 4 we should use:

```
AlarmSetInfo(21, 8, 1); ! priority mode
AlarmSetInfo(21, 7, 4); ! apply filter
```

In the following examples, the display parameters of the alarm list where the cursor is positioned are changed.

```
! Change the vertical offset (pages) to 2.
AlarmSetInfo(0,0,2);
! Change the vertical offset (lines) to 15.
AlarmSetInfo(0,1,15);
```

Change the alarm category to 10.

```
AlarmSetInfo(0,2,10);
```

Change the type of alarms displayed to type 5 (hardware alarms).

```
AlarmSetInfo(0,3,5);
```

In the following examples, the display parameters of the alarm list at AN 20 are changed.

```
! Display alarms with category 120 format and fonts
AlarmSetInfo(20, 4, 120);
! Display alarms with a new format
hFmt=FmtOpen("MyFormat","{Name}{Desc,20}",0);
AlarmSetInfo(20, 5, hFmt);
! Display alarms with a new font
hFont = DspFont("Times",-60,black,gray);
AlarmSetInfo(20, 6, hFont);
```

The following example displays alarms with categories 1-10, 20, or 25. Before AlarmSetInfo() is run, the page entry command for the alarm display page is configured as follows:

On page entry command: `hGrp=GrpOpen("MyGrp",1); StrToGrp(hGrp,"1..10,20,25");`

The page exit command for the alarm display page is configured as follows:

On page exit command: `GrpClose(hGrp); AlarmSetInfo(20, 2, hGrp);`

Note: `hGrp` is defined in the variables database.

See Also

[Alarm Functions](#)

AlarmSetQuery

Allows you to choose which alarms display on a page, by calling a user-defined query function to filter the alarms on specific criteria. The query function is called for each alarm, and only alarms matching the criteria are displayed on the page. AlarmSetQuery() only runs on the alarm server and returns to the display client. If you call a query function inAlarmSetQuer from a display then this function needs to exist in the project on the alarm server.

There are two steps involved in using a query to display alarms:

1. Write the Cicode function that will be used as the query function.
2. Specify the query function and its arguments in a call to AlarmSetQuery().

Note: You can also use AlarmSetQuery() to remove filtering from an alarm list. AlarmSetQuery(-1, "", "") stops the query function filtering the display of alarms.

Syntax

AlarmSetQuery(AN, QueryFunction [, sArgs] [, iAlways])

AN:

The AN where the alarm list originally commenced. (AN alarm page can contain more than one alarm list). You can also specify:

- 1 - Change the display parameters of every alarm list displayed on the page.
- 0 - Change the display parameters of the alarm list where the cursor is positioned.

QueryFunction:

The name of the Cicode query function written by the user. Once this function has been specified, it is called for each alarm, and determines whether or not the alarm should be displayed.

The QueryFunction returns an INT value of 1 (TRUE) or 0 (FALSE). If a value of TRUE is returned, the alarm will be displayed. If the query function returns FALSE, the alarm will be ignored and not displayed.

The query function's first parameter needs to be an INT. This parameter is initialized with the record ID of the current alarm, providing the query function with information about the alarm.

The query function's second parameter needs to also be an INT. It represents the instance or event of an alarm, and is used in filtering the alarms for display.

sArgs:

A list of arguments to be passed to the Cicode query function. The arguments are enclosed by double quotes ("") and separated by commas. This parameter is optional. If the query function does not require parameters other than the default INT parameter, then the list of arguments may be left out as follows:

```
AlarmSetQuery(0, "AlarmQueryDate");
```

In this case, the default value of an empty string will be used for the third parameter.

If the query function requires values to be passed in by the user, the following rules apply to determine the types of arguments:

- Digits are interpreted as INT
- Digits with decimals are interpreted as REAL
- Anything enclosed by ^" ^" is interpreted as a STRING

For example, to pass an INT of 23, a string of "23/12/1999", and a REAL value of 23.45 to the query function MyQueryDate(), AlarmSetQuery() should be invoked in the following way:

```
AlarmSetQuery(0, "MyQueryDate", "23, ^"23/12/1999^", 23.45);
```

The query function MyQueryDate() would be defined as follows:

```
INT
FUNCTION
MyQueryDate(INT nRID, INT nVer, INT iOne, STRING sOne, REAL rOne)
    ..
    ..
END
```

The types of the arguments listed in AlarmSetQuery() should match the types of the arguments defined in the query function.

iAlways:

Set to TRUE to so that the query is performed whenever it is called (no optimization). Default value is 0 (FALSE).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmGetFieldRec](#), [AlarmSetInfo](#), [QueryFunction](#)

Example

```
!Sets MyQueryDate() as the query function and provides the
arguments 23, 23/12/1999, and 23.45
AlarmSetQuery(0, "MyQueryDate", "23, ^"23/12/1999^", 23.45");
!Removes filtering by the current query function from all alarm
lists on the page
AlarmSetQuery(-1, "", "");
```

See Also

[Alarm Functions](#)

AlarmSetThreshold

Changes the thresholds (that is High High, Low etc.) of analog alarms. This function acts on the analog alarm where the cursor is positioned. Use this function to change (at run time) the threshold values that were specified in the Analog Alarms database. Threshold changes made using this function are permanent (that is they are saved to the project). The display format currently specified for the record (in the Analog Alarms form) will be applied to these values.

Syntax

AlarmSetThreshold(*Type*, *Value*)

Type:

The type of threshold:

- 0 - High high
- 1 - High
- 2 - Low
- 3 - Low low
- 4 - Deadband
- 5 - Deviation
- 6 - Rate of change

Value:

The new value of the threshold. Enter a blank value "" to remove the threshold.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmSetThresholdRec](#)

Example

System Keyboard	
Key Sequence	SetHighHigh # ## Enter
Command	AlarmSetThreshold(0, Arg1)
Comment	Change the threshold of a high high alarm
System Keyboard	
Key Sequence	SetHigh # # # Enter
Command	AlarmSetThreshold(1, Arg1)

Comment	Change the threshold of a high alarm
System Keyboard	
Key Sequence	SetLow # ## Enter
Command	AlarmSetThreshold(2, Arg1)
Comment	Change the threshold of a low alarm
System Keyboard	
Key Sequence	SetlowLow # ## Enter
Command	AlarmSetThreshold(3, Arg1)
Comment	Change the threshold of a low low alarm

See Also

[Alarm Functions](#)

AlarmSetThresholdRec

Changes the threshold (that is High High, Low etc.) of analog alarms by the alarm record number. You can call this function only on an Alarms Server for alarms on that server, or on the redundant server (if a redundant server is configured). If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

Threshold changes made using this function are permanent (that is they are saved to the project). The display format currently specified for the record (in the Analog Alarms form) will be applied to these values.

Note: Record numbers obtained from `AlarmGetDsp` are not valid for this function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

`AlarmSetThresholdRec(Record, Type, Value)`

Record:

The alarm record number, returned from any of the following alarm functions:

- `AlarmFirstCatRec()` or `AlarmNextCatRec()` - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstPriRec()` or `AlarmNextPriRec()` - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- `AlarmFirstTagRec()` or `AlarmNextTagRec()` - used to search for a record by alarm tag, name, and description.
- `AlarmGetDsp()` - used to find the record that is displayed at a specified AN, for either an alarm list or alarm summary entry. Set the `sField` argument in `AlarmGetDsp()` to "RecNo".

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

Type:

The type of threshold:

- 0 - High high
- 1 - High
- 2 - Low
- 3 - Low low
- 4 - Deadband
- 5 - Deviation
- 6 - Rate of change

Value:

The new value of the threshold. Enter a blank value "" to remove the threshold.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmSetThreshold](#), [MsgRPC](#)

See Also

[Alarm Functions](#)

AlarmSplit

Duplicates an entry (where the cursor is positioned) in the alarm summary display. You can use this function to add another comment to an alarm summary entry. You would normally call this function from a keyboard command.

Syntax

`AlarmSplit()`

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmSumSplit](#)

Example

System Keyboard	
Key Sequence	Split
Command	<code>AlarmSplit()</code>
Comment	Duplicates an alarm summary entry

See Also

[Alarm Functions](#)

AlarmSumAppend

Appends a new blank record to the alarm summary. Use this function to add new alarm summary entries, either for actual alarms or as special user summary entries.

If you specify a valid alarm tag in the *sTag* field, the summary entry is linked to the actual alarm. If you specify an asterisk '*' as the first letter of the tag, the summary entry becomes a user event.

User events are not attached to alarm records, so their status will not change. Manually change the status of the user event, by calling the `AlarmSumSet()` function with the index returned by `AlarmSumAppend()`. As user events are not attached to alarms, they don't have the alarm fields - so the `AlarmSumGet()` function will not return any field data.

The latest entry in the Alarm summary will reflect the events of the alarm whether the alarm summary entry was appended created by an actual alarm event.

You can use user events to keep a record of logins, or control operations that you need to display in the alarm summary etc. The fields of UserEvents needs to be set immediately after creation using the AlarmSumSet() function or AlmSummarySetFieldValue() function. These entries in the Alarm Summary cannot be filtered from the summary in an AlmSummaryOpen() browse session.

To give an appended alarm summary entry the appearance of having been Acknowledged set the Acknowledge time of the summary entry. When the summary entry is linked to an actual alarm the function AlmSummaryAck() can be used to acknowledge the actual alarm linked to the summary entry. However the AlmSummaryAck function will not directly affect the alarm summary entry.

AlarmSumAppend() can only be used if the Alarm Server is on the current machine. When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumAppend(sTag [, ClusterName, OnTime, onMilli, bRedundant=true])

sTag:

The alarm tag to append. Use an asterisk '*' as the first letter to append a user event to the alarm summary. Please be aware that if you using this 'user event mode' the AlarmSumAppend function returns the alarm summary index - not the error code.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

OnTime:

The alarm OnTime

OnMilli:

Milliseconds part of the alarm's ON time

bRedundant:

New alarm record is created on both redundant server instances. For backward compatibility with existing projects, redundancy can be switched off by calling AlarmSumAppend with *bRedundant = false*

Return Value

The index of the alarm summary entry, or -1 if the record could not be appended.

Related Functions

[AlarmSumSet](#), [AlmSummarySetFieldValue](#), [AlmSummaryAck](#)

Example

```
! Append alarm to summary display
AlarmSumAppend("CV101");
! Append user event
iIndex = AlarmSumAppend("*MyEvent");
AlarmSumSet(iIndex, "Comment", "My event comment");
AlarmSumSet(iIndex, "OnTime", TimeCurrent());
```

See Also

[Alarm Functions](#)

AlarmSumCommit

Commits the alarm summary record to the alarm summary device. Alarm summaries are normally written to the alarm summary device just before they are deleted from the summary queue. The length of time that alarm summary entries remain in the alarm summary queue is controlled by [Alarm]SummaryTimeout parameter.

This function allows you to commit the alarm summary records now, rather than when they are deleted from the queue.

This function can only be used if the Alarm Server is on the current machine. When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumCommit(*Index [, ClusterName]*)

Index:

The alarm summary index (returned from the `AlarmSumFirst()`, `AlarmSumNext()`, `AlarmSumLast()`, `AlarmSumPrev()`, `AlarmSumAppend()`, or `AlarmSumFind()` function).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumGet](#), [AlarmSumFind](#)

Example

```
/* This function commits alarm summary entries that match the
specified tag. */
FUNCTION
SumCommitTag(STRING sTag)
    INT Next;
    INT Index;
    STRING Name;
    Index=AlarmSumFirst();
    WHILE Index<>-1 DO
        Name=AlarmSumGet(Index,"Tag");
        Next=AlarmSumNext(Index);
        IF Name=sTag THEN
            AlarmSumCommit(Index);
        END
        Index=Next;
    END
END
```

See Also

[Alarm Functions](#)

AlarmSumDelete

This command is deprecated in this version of CitectSCADA. Use the [Alm-SummaryDelete](#) command instead.

Deletes an alarm summary entry. You identify the alarm summary entry by the *Index*, returned by one of the alarm summary search functions.

By embedding this function in a loop, you can delete a series of alarm summary entries. To start deleting from the oldest entry, call the `AlarmSumFirst()` function to get the index, and then call `AlarmSumNext()` in a loop. To delete back from the most recent entry, call `AlarmSumLast()` and then `AlarmSumPrev()` in a loop.

You can also get the *Index* from the `AlarmSumFind()` function, which finds an alarm summary entry by its alarm record identifier and time of activation.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumDelete(*Index* [, *ClusterName*])

Index:

The alarm summary index (returned from the `AlarmSumFirst()`, `AlarmSumNext()`, `AlarmSumLast()`, `AlarmSumPrev()`, `AlarmSumAppend()`, or `AlarmSumFind()` function).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if the specified alarm entry exists, otherwise an [error](#) is returned.

Related Functions

[AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumGet](#), [AlarmSumFind](#)

Example

```
/* This function deletes all alarm summary entries that match the
specified tag. */
FUNCTION
SumDelTag (STRING sTag)
    INT Next;
    INT Index;
    STRING Name;
    Index=AlarmSumFirst () ;
    WHILE Index<>-1 DO
        Name=AlarmSumGet (Index, "Tag") ;
        Next=AlarmSumNext (Index) ;
        IF Name=sTag THEN
            AlarmSumDelete (Index) ;
        END
        Index=Next;
    END
END
```

See Also

[Alarm Functions](#)

AlarmSumFind

This command is deprecated in this version of CitectSCADA. Use the AlmSummary commands instead.

Finds the alarm summary index for an alarm that you specify by the alarm record identifier and alarm activation time (OnTime). You can use this index in the AlarmSumGet() function to get field data from an alarm record, in the AlarmSumSet() function to change the existing data in that record, or in the AlarmSumDelete() function to delete the record. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

To work with a series of alarm summary records, call this function to get the index, and then call either AlarmSumNext() to move forwards in the summary, or AlarmSumPrev() to move backwards in the summary.

Note: Record numbers obtained from AlarmGetDsp are not valid for this function. Instead use AlarmFirstTagRec() to get the record. This should be called from the server side using MsgRPC.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumFind(*Record*, *OnTime* [, *ClusterName*])

Record:

The alarm record number, returned from any of the following alarm functions:

- AlarmFirstCatRec() or AlarmNextCatRec() - used to search for a record by alarm category, area, and type (acknowledged, disabled, etc.).
- AlarmFirstPriRec() or AlarmNextPriRec() - used to search for a record by alarm priority, area, and type (acknowledged, disabled, etc.).
- AlarmFirstTagRec() or AlarmNextTagRec() - used to search for a record by alarm tag, name, and description.

To store this value, use data type Int in Cicode or Long for variable tags (Long needs 4 bytes).

OnTime:

The ON time of the alarm associated with the Record, that is, the time that the alarm was activated.

AlarmSumFind() requires that the OnTime argument contains the number of seconds from Midnight, so the formulation:

```
iOnTime = StrToTime(AlarmSumGet(iIndex, "OnTime"));
```

will NOT yield the correct result. The correct formulation for this calculation is:

```
OnTime = StrToTime(AlarmSumGet(iIndex, "OnTime")) + Time-
Midnight(TimeCurrent());
```

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The index of the alarm summary entry, or -1 if no alarm summary entry is found.

Related Functions

[AlarmSumNext](#), [AlarmSumSet](#), [AlarmSumDelete](#), [AlarmSumFirst](#), [AlarmSumNext](#),
[AlarmSumLast](#), [AlarmSumPrev](#), [MsgRPC](#)

Example

```
/* This function sets the summary comment from the alarm record
number and the ontime of the summary event. */
FUNCTION
SumSetComment(INT AN, STRING sComment)
    INT nRecord;
    INT iOnTime;
    INT hAlarm1;
    STRING AlmTag;

    AlmTag = AlarmGetDsp(AN, "Tag");
    iOnTime = StrTo-
Date(AlarmGetDsp(AN, "OnDate"))+StrToTime(AlarmGetDsp(AN, "OnTime"));
    hAlarm1 = MsgOpen("Alarm", 0, 0);
    MsgRPC(hAlarm1, "AlmSvrSumSetComment", "^^" + AlmTag + "^," + IntToStr(iOnTime)
+ ",^" + sComment + "^", 1);
    MsgClose("Alarm", hAlarm1);
END

FUNCTION
AlmSvrSumSetComment(STRING AlmTag, INT iOnTime, STRING sComment)
    INT nRecord = AlarmFirstTagRec(AlmTag, "", "");
    INT Index = AlarmSumFind(nRecord, iOnTime);
    IF Index <> -1 THEN
        AlarmSumSet(Index, "Comment", sComment);
    END
END
```

See Also

[Alarm Functions](#)

[AlarmSumFirst](#)

This command is deprecated in this version of CitectSCADA. Use the [AlmSummaryFirst](#) command instead.

Gets the index of the oldest alarm summary entry. You can use this index in the `AlarmSumGet()` function to get field data from an alarm record, in the `AlarmSumSet()` function to change the existing data in that record, or in the `AlarmSumDelete()` function to delete the record.

To work with a series of alarm summary records, call this function to get the index, and then call `AlarmSumNext()` within a loop, to move forwards in the alarm summary.

This function can only be used if the Alarm Server is on the current machine. When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumFirst([ClusterName])

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The index of the oldest alarm summary entry, or -1 if no alarm summary entry is found.

Related Functions

[AlarmSumGet](#), [AlarmSumSet](#), [AlarmSumDelete](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#)

Example

```
/* This function finds all alarm summary entries that match the
specified tag and sets the "OffTime" to the time specified. The
alarm entry is not acknowledged or set to the off state, the alarm
summary "OffTime" field is all that is affected. */
FUNCTION
SumSetTime(STRING sTag, INT Time)
    INT Index;
    STRING Name;
    Index=AlarmSumFirst();
    WHILE Index<>-1 DO
        Name=AlarmSumGet(Index,"Tag");
        IF Name=sTag THEN
```

```
    AlarmSumSet(Index, "OffTime", Time);
END
Index=AlarmSumNext (Index);
END
END
```

See Also

[Alarm Functions](#)

AlarmSumGet

This command is deprecated in this version of CitectSCADA. Use the [Alm-SummaryGetField](#) command instead.

Gets field data from an alarm summary entry. The data is returned as a string. You identify the alarm summary entry by the *Index*, returned by one of the alarm summary search functions. If calling this function from a remote client, use the [MsgRPC\(\)](#) function.

By embedding this function in a loop, you can get data from a series of alarm summary entries. To start from the oldest entry, call the `AlarmSumFirst()` function to get the index, and then call `AlarmSumNext()` in a loop. To work back from the most recent entry, call `AlarmSumLast()` and then `AlarmSumPrev()` in a loop.

You can also get the *Index* from the `AlarmSumFind()` function, which finds an alarm summary entry by its alarm record identifier and time of activation.

Note: Record numbers obtained from `AlarmGetDsp` are not valid for this function.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

`AlarmSumGet(Index, sField [, ClusterName])`

Index:

The alarm summary index (returned from the `AlarmSumFirst()`, `AlarmSumNext()`, `AlarmSumLast()`, `AlarmSumPrev()`, `AlarmSumAppend()`, or `AlarmSumFind()` function).

sField:

The name of the field from which to extract the data:

Tag	Alarm tag
-----	-----------

AckDate	Alarm acknowledged date
AckTime	Alarm acknowledged time
Category	Alarm category
Comment	Alarm comment
DeltaTime	Alarm active time
Desc	Alarm description
Help	Help page
Name	Alarm name
OffDate	Alarm OFF date
OffTime	Alarm OFF time
OnDate	Alarm ON date
OnTime	Alarm ON time
State	Alarm state

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

Field data from the alarm summary entry (as a string).

Related Functions

[AlarmSumSet](#), [AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumFind](#), [MsgRPC](#)

Example

See [AlarmSumFirst](#)

See Also

[Alarm Functions](#)

AlarmSumLast

This command is deprecated in this version of CitectSCADA. Use the [AlmSummaryLast](#) command instead.

Gets the index of the most recent alarm summary entry. You can use this index in the AlarmSumGet() function to get field data from an alarm record, in the AlarmSumSet() function to change the existing data in that record, or in the AlarmSumDelete() function to delete the record.

To work with a series of alarm summary records, call this function to get the index, and then call AlarmSumPrev() within a loop, to move backwards in the alarm summary.

This function can only be used if the Alarm Server is on the current machine. When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumLast([ClusterName])

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The index of the most recent alarm summary entry, or -1 if no alarm summary entry is found.

Related Functions

[AlarmSumGet](#), [AlarmSumSet](#), [AlarmSumDelete](#), [AlarmSumPrev](#), [AlarmSumFirst](#), [AlarmSumNext](#)

Example

```
/* This function finds all alarm summary entries that match the
specified tag and sets the "OffTime" to the time specified. The
alarm entry is not acknowledged or set to the off state, the alarm
summary "OffTime" field is all that is affected. */
FUNCTION
```

```

SumSetTime(STRING sTag, INT Time)
    INT Index;
    STRING Name;
    Index=AlarmSumLast();
    WHILE Index<>-1 DO
        Name=AlarmSumGet(Index, "Tag");
        IF Name=sTag THEN
            AlarmSumSet(Index, "OffTime", Time);
        END
        Index=AlarmSumPrev(Index);
    END
END

```

See Also

[Alarm Functions](#)

AlarmSumNext

This command is deprecated in this version of CitectSCADA. Use the [AlmSummaryNext](#) command instead.

Gets the index of the next alarm summary entry, that is, the entry that occurred later than the entry specified by Index. You can use this index in the AlarmSumGet() function to get field data from an alarm record, in the AlarmSumSet() function to change the existing data in that record, or in the AlarmSumDelete() function to delete the record.

You can use this function to work with a series of alarm summary records. Call the AlarmSumFirst() or AlarmSumFind() function to get the index, and then call AlarmSumNext() within a loop, to move forwards in the alarm summary.

You can also get the index of an entry as soon as it displays on the alarm summary. Alarm summary entries are recorded with the most recent entry at the end of the list. Call AlarmSumLast() to get the index for the most recent entry, and then call AlarmSumNext() to get the index for the next entry that occurs.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumNext(Index [, ClusterName])

Index:

The alarm summary index (returned from the AlarmSumFirst(), AlarmSumNext(), AlarmSumLast(), AlarmSumPrev(), AlarmSumAppend(), or AlarmSumFind() function).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The index of the next alarm summary entry or -1 if no more alarm summary entries are found.

Related Functions

[AlarmSumGet](#), [AlarmSumSet](#), [AlarmSumDelete](#), [AlarmSumFirst](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumFind](#)

Example

See [AlarmSumFirst](#)

See Also

[Alarm Functions](#)

AlarmSumPrev

This command is deprecated in this version of CitectSCADA. Use the [AlmSummaryPrev](#) command instead.

Gets the index of the previous alarm summary entry, that is, the entry that occurred before the entry specified by *Index*. You can use this index in the AlarmSumGet() function to get field data from an alarm record, in the AlarmSumSet() function to change the existing data in that record, or in the AlarmSumDelete() function to delete the record.

You can use this function to work with a series of alarm summary records. Call the AlarmSumLast() or AlarmSumFind() function to get the index, and then call AlarmSumPrev() within a loop, to move backwards in the alarm summary.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumPrev(*Index* [, *ClusterName*])

Index:

The alarm summary index (returned from the AlarmSumFirst(), AlarmSumNext(), AlarmSumLast(), AlarmSumPrev(), AlarmSumAppend(), or AlarmSumFind() function).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if the alarm summary entry exists, otherwise an [error](#) is returned.

Related Functions

[AlarmSumGet](#), [AlarmSumSet](#), [AlarmSumDelete](#), [AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumFind](#)

Example

See [AlarmSumLast](#).

See Also

[Alarm Functions](#)

AlarmSumSet

This command is deprecated in this version of CitectSCADA. Use the [Alm-SummarySetFieldValue](#) command instead.

Sets field information in an alarm summary entry. You identify the alarm summary entry by the *Index*, returned by one of the alarm summary search functions.

By embedding this function in a loop, you can change field data in a series of alarm summary entries. To start from the oldest entry, call the `AlarmSumFirst()` function to get the index, and then call `AlarmSumNext()` in a loop. To work back from the latest entry, call `AlarmSumLast()` and then `AlarmSumPrev()` in a loop.

You can also get the *Index* from the `AlarmSumFind()` function, which finds an alarm summary entry by its alarm record identifier and time of activation.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Fields of an appended alarm can only be set using this function or the replacement function `AlmSummarySetField()`.

You can use user events to keep a record of logins, or control operations that you need to display in the alarm summary etc. The fields of UserEvents need to be set immediately after creation using this function. These entries in the Alarm Summary cannot be filtered from the summary in an `AlmSummaryOpen()` browse session.

Syntax

AlarmSumSet(Index, sField, sData [, ClusterName])

Index:

The alarm summary index (returned from the AlarmSumFirst(), AlarmSumNext(), AlarmSumLast(), AlarmSumPrev(), AlarmSumAppend(), or AlarmSumFind() function).

sField:

The name of the field in which data is to be set:

AckTime	Alarm acknowledged time
Comment	Alarm comment
OffMilli (for time stamped alarms only)	Alarm millisecond off time
OffTime	Alarm OFF time
OnMilli (for time stamped alarms only)	Alarm millisecond on time
OnTime	Alarm ON time
State	Alarm state

sData:

The new value of the field.

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if the alarm summary entry exists, otherwise an [error](#) is returned.

Related Functions

[AlarmSumGet](#), [AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumFind](#)

Example

See [AlarmSumFirst](#)

See Also

[Alarm Functions](#)

AlarmSumSplit

Duplicates the alarm summary entry identified by *Index*. You can use this function to add another comment to an alarm summary entry.

When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

To duplicate an alarm summary entry on a Control Client, use the `AlarmSplit()` function - the entry at the cursor position is duplicated.

Syntax

`AlarmSumSplit(Index [, ClusterName])`

Index:

The alarm summary index (returned from the `AlarmSumFirst()`, `AlarmSumNext()`, `AlarmSumLast()`, `AlarmSumPrev()`, `AlarmSumAppend()`, or `AlarmSumFind()` function).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The Index of the new entry, or -1 on [error](#).

Related Functions

[AlarmSumGet](#), [AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumFind](#), [AlarmSplit](#)

Example

```
/* This function finds the first alarm summary entry that matches
the specified tag, splits that entry and then adds the specified
comment to the new entry. */
```

```
FUNCTION
AlarmSplitAdd(STRING Tag, STRING Comment)
    INT Index;
    STRING Name;
    Index=AlarmSumFirst();
    WHILE Index<>-1 DO
        Name=AlarmSumGet(Index,"Tag");
        IF Name=sTag THEN
            AlarmSumSplit(Index);
            Index=AlarmSumFirst();
            AlarmSumSet(Index,"Comment",Comment);
            Index=-1;
        ELSE
            Index=AlarmSumNext(Index);
        END
    END
END
```

See Also

[Alarm Functions](#)

AlarmSumType

This command is deprecated in this version of CitectSCADA. Use the AlmSummary commands instead.

Retrieves a value that indicates a specified alarm's type, that is whether it's a digital alarm, an analog alarm, hardware alarm, etc.

This function can only be used if the Alarm Server is on the current machine. When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

AlarmSumType(*Index [, ClusterName]*)

Index:

The alarm summary index (returned from the AlarmSumFirst(), AlarmSumNext(), AlarmSumLast(), AlarmSumPrev(), AlarmSumAppend(), or AlarmSumFind() function).

ClusterName:

Specifies the name of the cluster in which the Alarm Server resides. This is optional if you have one cluster or are resolving the alarm server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

A number that represents one of the following alarm types:

- **0** = digital alarm
- **1** = analog alarm
- **2** = advanced alarm
- **3** = Multi-Digital alarm
- **4** = Argyle analog alarm
- **5** = user-generated event
- **6** = high resolution alarm
- **8** = time-stamped digital alarm
- **9** = time-stamped analog alarm
- **-1** indicates an invalid response to the request.

Related Functions

[AlarmSumGet](#), [AlarmSumFirst](#), [AlarmSumNext](#), [AlarmSumLast](#), [AlarmSumPrev](#), [AlarmSumFind](#), [AlarmSplit](#)

See Also

[Alarm Functions](#)

AlmSummaryAck

The AlmSummaryAck function acknowledges the alarm in the active alarm list which is linked to the current entry of the alarm summary browse session.

If the current alarm summary browse session entry is a user event the function will have no effect. To change the appearance of a summary entry to acknowledged without sending an acknowledge message to the active alarm list set the acknowledged time using the AlmSummarySetFieldValue() function.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmSummaryAck(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryClear

The AlmSummaryClear function clears the alarm at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmSummaryClear(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryClose

The AlmSummaryClose function terminates an active data browse session and cleans up all resources associated with the session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmSummaryClose(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryCommit

The AlmSummaryCommit function triggers the actual write of the value for the field previously specified by AlmSummarySetFieldValue.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryCommit(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

Example

```
INT errorCode = 0;
...
errorCode = AlmSummaryCommit(iSession);
IF errorCode = 0 THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Alarm Functions](#)

AlmSummaryDelete

The AlmSummaryDelete function deletes the record in the filtered list that the cursor is currently referencing.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryDelete(*iSession*)

iSession:

The handle to a filtered list previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

Example

```
INT errorCode = 0;
...
errorCode = AlmSummaryDelete(iSession);
```

```

IF errorCode = 0 THEN
    // Successful case
ELSE
    // Function returned an error
END
...

```

See Also[Alarm Functions](#)**AlmSummaryDeleteAll**

The AlmSummaryDeleteAll function deletes every record from the filtered list source.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax**AlmSummaryDeleteAll(*iSession*)**

iSession:

The handle to a filtered list previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm filtered list session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

Example

```

INT errorCode = 0;
...
errorCode = AlmSummaryDeleteAll(iSession);
IF errorCode = 0 THEN
    // Successful case
ELSE
    // Function returned an error
END
...

```

See Also

[Alarm Functions](#)

AlmSummaryDisable

The AlmSummaryDisable function disables the alarm at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmSummaryDisable(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryEnable

The AlmSummaryEnable function enables the alarm at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmSummaryEnable(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryFirst

The AlmSummaryFirst function places the data browse cursor at the first record.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryFirst(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryGetField

The AlmSummaryGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmSummaryGetField(*iSession*, *sFieldName*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

sFieldName:

The name of the field that references the value to be returned. Supported fields are:

ACQERROR, CUSTOM1, CUSTOM2, CUSTOM3, CUSTOM4, CUSTOM5, CUSTOM6, CUSTOM7, CUSTOM8, DATEEXT, ERRDESC, ERRPAGE, FORMAT, GROUP, LOCALTIMEDATE, LOGSTATE, NATIVE_DESC, NATIVE_NAME, NODE, OFFTIMEDATE, ONTIMEDATE, ORATODATE, ORATOFFDATE, ORATOONDATE, PRIV, SUMTYPE, TAGEX, TIMEDATE, TYPE, TYPENUM.

See [Browse Function Field Reference](#) for information about fields.

Return Value

The value of the specified field as a string. An empty string may or may not be an indication that an error has been detected. The last error should be checked in this instance to determine if an error has actually occurred.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

Example

```
STRING fieldValue = "";
STRING fieldName = "TYPE";
INT errorCode = 0;
...
fieldValue = AlmSummaryGetField(iSession, sFieldName);
IF fieldValue <> "" THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Alarm Functions](#)

AlmSummaryLast

The AlmSummaryLast function places the data browse cursor at the most recent summary record from the last cluster of the available browsing cluster list.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryLast(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryNext

The AlmSummaryNext function moves the data browse cursor forward one record. If you call this function after you have reached the end of a summary, error 412 is returned (Databrowse session EOF).

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryNext(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummaryOpen

The AlmSummaryOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryOpen(*sFilter*, *sFields* [, *sClusters*])

sFilter:

A filter expression specifying the records to return during the browse. An empty string indicates that all records will be returned. Where a fieldname is not specified in the filter, it is assumed to be tagname. For example, the filter "AAA" is equivalent to "TAG=AAA".

sFields:

Specifies via a comma delimited string the columns to be returned during the browse. An empty string indicates that the server will return all available columns. Supported fields are:

ACKDATE, ACKDATEEXT, ACKTIME, ACQERROR, ALARMTYPE, ALM-COMMENT, AREA, CATEGORY, CLUSTER, COMMENT, CUSTOM1, CUSTOM2, CUSTOM3, CUSTOM4, CUSTOM5, CUSTOM6, CUSTOM7, CUSTOM8, DATE, DATEEXT, DEADBAND, DELTATIME, DESC, DEVIATION, ERRDESC, ERRPAGE, FORMAT, FULLNAME, GROUP, HELP, HIGH, HIGHHIGH, LOCALTIMEDATE, LOGSTATE, LOW, LOWLOW, MILLISEC, NAME, NATIVE_COMMENT, NATIVE_DESC, NATIVE_NAME, NATIVE_SUMDESC, NODE, OFFDATE, OFFDATEEXT, OFF-MILLI, OFFTIME, OFFTIMEDATE, OLD_DESC, ONDATE, ONDATEEXT, ONMILLI, ONTIME, ONTIMEDATE, ORATODATE, ORATOFFDATE, ORATOONDATE, PAGING, PAGINGGROUP, PRIORITY, PRIV, RATE, STATE, STATE_DESC, STATE_DESC0, STATE_DESC1, STATE_DESC2, STATE_DESC3, STATE_DESC4, STATE_DESC5, STATE_DESC6, STATE_

DESC7, SUMDESC, SUMSTATE, SUMTYPE, TAG, TAGEX, TIME, TIME-DATE, TYPE, TYPENUM, USERDESC, USERNAME, VALUE.

See [Browse Function Field Reference](#) for information about fields.

sClusters:

An optional parameter that specifies via a comma delimited string the subset of the clusters to browse. An empty string indicates that the connected clusters will be browsed.

Return Value

Returns an integer handle to the browse session. Returns -1 on error.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryPrev](#), [AlmSummarySetFieldValue](#)

Example

```
INT iSession;
...
iSession = AlmSummaryOpen ("NAME=ABC*", "NAME,TYPE",
"ClusterA,ClusterB");
IF iSession <> -1 THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Alarm Functions](#)

AlmSummaryPrev

The AlmSummaryPrev function moves the data browse cursor back one record. If you call this function after you have reached the beginning of a summary, error 412 is returned (Databrowse session EOF).

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummaryPrev(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummarySetFieldValue](#)

See Also

[Alarm Functions](#)

AlmSummarySetFieldValue

The AlmSummarySetFieldValue function sets a new value for the specified field for the record the data browse cursor is currently referencing. The value is not committed until a call to AlmSummaryCommit is made.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmSummarySetFieldValue(*iSession*, *sFieldName*, *sFieldValue*)

iSession:

The handle to a browse session previously returned by a AlmSummaryOpen call.

sFieldName:

The name of the field whose value is to be updated. Supported fields are:

AckTime	Alarm acknowledged time
Comment	Alarm comment
OffMilli (for time stamped alarms only)	Alarm millisecond off time
OffTime	Alarm OFF time

OnMilli (for time stamped alarms only)	Alarm millisecond on time
OnTime	Alarm ON time

See [Browse Function Field Reference](#) for additional information about fields.

sFieldValue:

The field value to update.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmSummaryAck](#), [AlmSummaryClear](#), [AlmSummaryClose](#), [AlmSummaryCommit](#), [AlmSummaryDelete](#), [AlmSummaryDeleteAll](#), [AlmSummaryDisable](#), [AlmSummaryEnable](#), [AlmSummaryFirst](#), [AlmSummaryGetField](#), [AlmSummaryLast](#), [AlmSummaryNext](#), [AlmSummaryOpen](#), [AlmSummaryPrev](#)

Example

```
STRING sFieldValue = "NEW_COMMENT";
STRING sFieldName = "COMMENT";
INT errorCode = 0;
...
errorCode = AlmSummarySetFieldValue(iSession, sFieldName,
sFieldValue);
IF errorCode = 0 THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Alarm Functions](#)

AlmTagsAck

The AlmTagsAck function acknowledges the alarm tag at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmTagsAck(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also

[Alarm Functions](#)

AlmTagsClear

The AlmTagsClear function clears the alarm tag at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmTagsClear(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also

[Alarm Functions](#)

AlmTagsClose

The AlmTagsClose function terminates an active data browse session and cleans up all resources associated with the session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmTagsClose(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also

[Alarm Functions](#)

AlmTagsDisable

The AlmTagsDisable function disables the alarm tag at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmTagsDisable(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also

[Alarm Functions](#)

AlmTagsEnable

The AlmTagsEnable function enables the alarm tag at the current cursor position in an active data browse session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmTagsEnable(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also

[Alarm Functions](#)

AlmTagsFirst

The AlmTagsFirst function places the data browse cursor at the first record.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmTagsFirst(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also[Alarm Functions](#)**AlmTagsGetField**

The AlmTagsGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax**AlmTagsGetField(*iSession*, *sFieldName*)***iSession*:

The handle to a browse session previously returned by a AlmTagsOpen call.

sFieldName:

The name of the field that references the value to be returned. Supported fields are:

ACQERROR, CUSTOM1, CUSTOM2, CUSTOM3, CUSTOM4, CUSTOM5, CUSTOM6, CUSTOM7, CUSTOM8, DATEEXT, ERRDESC, ERRPAGE, FORMAT, GROUP, LOCALTIMEDATE, LOGSTATE, NATIVE_DESC, NATIVE_NAME, NODE, OFFTIMEDATE, ONTIMEDATE, ORATODATE, ORATOOFFSETDATE, ORATOONDATE, PRIV, SUMTYPE, TAGEX, TIMEDATE, TYPE, TYPENUM.

See [Browse Function Field Reference](#) for information about fields.

Return Value

The value of the specified field as a string. An empty string may or may not be an indication that an error has been detected. The last error should be checked in this instance to determine if an error has actually occurred.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

Example

```
STRING fieldValue = "";
STRING fieldName = "TYPE";
INT errorCode = 0;
...
fieldValue = AlmTagsGetField(iSession, sFieldName);
```

```
IF fieldValue <> "" THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Alarm Functions](#)

AlmTagsNext

The AlmTagsNext function moves the data browse cursor forward one record. If you call this function after you have reached the end of the records, error 412 is returned (Dat-abrowse session EOF).

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

AlmTagsNext(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the browse has successfully been moved to the next record, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

See Also

[Alarm Functions](#)

AlmTagsNumRecords

The AlmTagsNumRecords function returns the number of records that match the filter criteria.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

AlmTagsNumRecords(*iSession*)

iSession:

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

The number of records that have matched the filter criteria. A value of 0 denotes that no records have matched. A value of -1 denotes that the browse session is unable to provide a fixed number. This may be the case if the data being browsed changed during the browse session.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsOpen](#), [AlmTagsPrev](#)

Example

```
INT numRecords = 0;
...
numRecords = AlmTagsNumRecords(iSession);
IF numRecords <> 0 THEN
    // Have records
ELSE
    // No records
END
...
```

See Also

[Alarm Functions](#)

AlmTagsOpen

The AlmTagsOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Note: After calling AlmTagsOpen() it is necessary to call AlmTagsFirst() in order to place the cursor at the beginning of the browse session, otherwise a hardware alarm is invoked.

Syntax

AlmTagsOpen(*sFilter*, *sFields* [, *sClusters*])

sFilter:

A filter expression specifying the records to return during the browse. An empty string indicates that all records will be returned. Where a fieldname is not specified in the filter, it is assumed to be tagname. For example, the filter "AAA" is equivalent to "name=AAA".

sFields:

Specifies via a comma delimited string the columns to be returned during the browse. An empty string indicates that the server will return all available columns. Supported fields are:

ACKDATE, ACKDATEEXT, ACKTIME, ACQERROR, ALARMTYPE, ALM-COMMENT, AREA, CATEGORY, CLUSTER, COMMENT, CUSTOM1, CUSTOM2, CUSTOM3, CUSTOM4, CUSTOM5, CUSTOM6, CUSTOM7, CUSTOM8, DATE, DATEEXT, DEADBAND, DELTATIME, DESC, DEVIATION, ERRDESC, ERRPAGE, FORMAT, FULLNAME, GROUP, HELP, HIGH, HIGHHIGH, LOCALTIMEDATE, LOGSTATE, LOW, LOWLOW, MILLISEC, NAME, NATIVE_COMMENT, NATIVE_DESC, NATIVE_NAME, NATIVE_SUMDESC, NODE, OFFDATE, OFFDATEEXT, OFF-MILLI, OFFTIME, OFFTIMEDATE, OLD_DESC, ONDATE, ONDATEEXT, ONMILLI, ONTIME, ONTIMEDATE, ORATODATE, ORATOFFDATE, ORATOONDATE, PAGING, PAGINGGROUP, PRIORITY, PRIV, RATE, STATE, STATE_DESC, STATE_DESC0, STATE_DESC1, STATE_DESC2, STATE_DESC3, STATE_DESC4, STATE_DESC5, STATE_DESC6, STATE_DESC7, SUMDESC, SUMSTATE, SUMTYPE, TAG, TAGEX, TIME, TIME-DATE, TYPE, TYPENUM, USERDESC, USERNAME, VALUE.

See [Browse Function Field Reference](#) for information about fields.

sClusters:

An optional parameter that specifies via a comma delimited string the subset of the clusters to browse. An empty string indicates that the connected clusters will be browsed.

Return Value

Returns an integer handle to the browse session. Returns -1 when an error is detected.

The returned entries will be ordered alphabetically by name.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsPrev](#)

Example

```

INT iSession;
...
iSession = AlmTagsOpen("NAME=ABC*", "NAME,TYPE",
"ClusterA,ClusterB");
IF iSession <> -1 THEN
    // Successful case
ELSE
    // Function returned an error
END
...

```

See Also[Alarm Functions](#)**AlmTagsPrev**

The AlmTagsPrev function moves the data browse cursor back one record. If you call this function after you have reached the beginning of the records, error 412 is returned (Databrowse session EOF).

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax**AlmTagsPrev(*iSession*)***iSession:*

The handle to a browse session previously returned by a AlmTagsOpen call.

Return Value

0 (zero) if the alarm browse session exists, otherwise an [error](#) is returned.

Related Functions

[AlmTagsAck](#), [AlmTagsClear](#), [AlmTagsDisable](#), [AlmTagsEnable](#), [AlmTagsClose](#), [AlmTagsFirst](#), [AlmTagsGetField](#), [AlmTagsNext](#), [AlmTagsNumRecords](#), [AlmTagsOpen](#)

See Also[Alarm Functions](#)**HwAlarmQue**

Returns the handle of the hardware alarm queue. The Alarms Server writes hardware alarm information into this queue as each hardware alarm occurs. To read events from this queue, use the QueRead() or QuePeek() functions. The data written into the queue is the hardware alarm format, and is stored in the Str field.

To use this function, you need to enable the hardware alarm queue by specifying the [Alarm]HwAlarmQueMax parameter. This parameter specifies the maximum length that the queue can grow to. The [Alarm]HwAlarmFmt parameter defines the format of the data placed into the string field. If HwAlarmFmt is not specified then the format defaults to "Time: {Time,12} Date:{Date,11} Desc:{Desc,40}".

The following format fields are relevant to hardware alarms:

- {Tag,n}
- {TagEx,n}
- {Cluster,n}
- {Name,n}
- {State,n}
- {Time,n}
- {Date,n}
- {Desc,n}
- {ErrDesc,n}
- {ErrPage,n}

For a description of the fields see the "["Alarm Display Fields"](#)" help page.

The number of buffers available for all user queues combined is controlled by the [Code]Queue parameter. Each entry in any user queue consumes one buffer. When all buffers have been used the Alarms Server will not be able to add new hardware alarms to the queue, and the error message "Out Of Buffers Usr.Que" will be written to sys-log.dat.

Syntax

HwAlarmQue()

Return Value

The handle of the hardware alarm queue, or -1 if the queue cannot be opened.

Related Functions

[QueRead\(\)](#), [QuePeek\(\)](#)

Example

```
hQue = HwAlarmQue()
```

```
WHILE TRUE DO
```

```
QueRead(hQue, nAlarmType, sHwAlarmString, 1);
```

```
/* do what ever with the alarm information */
```

```
....
```

```
Sleep(0);
```

```
END
```

See Also

[Alarm Functions](#)

QueryFunction

The user-defined query function set in **AlarmSetQuery**. Called for each active alarm, the query function can be written to display an alarm based on specific information (for example, OnDate). To examine the information in an alarm field, call the function **AlarmGetFieldRec** from within your query function.

Note: The function name "QueryFunction" can be any valid Cicode function name specified by the user.

Syntax

QueryFunction(*nRID*, *nVer* [, *Arg01*, *Arg02*,])

nRID:

The record number of the alarm currently being filtered. This provides the query function with access to information about the alarm. This parameter is represented with an INT, and needs to be the first parameter of your query function.

nVer:

The version of an alarm.

If an alarm is triggered more than once in a given period, the version lets you distinguish between different instances of the alarm's activity.

Since you may wish to display on a page alarms which have more than one instance, this parameter needs to be passed to AlarmGetFieldRec in order to correctly filter the alarms.

The version is represented with an INT, and needs to be the second parameter of your query function.

Arg01, Arg02:

A list of arguments, separated by commas.

The query function is passed the arguments specified in the call to AlarmSetQuery(). For this reason, the arguments listed in AlarmSetQuery() needs to be of the same type as those defined in the query function.

Return Value

The return value needs to be defined as an INT with a value of either 1 (TRUE) or 0 (FALSE). If the function returns a value of TRUE, the alarm being filtered is displayed, otherwise it is excluded from the alarms list.

Related Functions

[AlarmSetQuery](#), [AlarmGetFieldRec](#), [AlarmSetInfo](#)

Example

```
! The query function AlarmQueryDate() compares sDate with the
OnDate of each alarm.AlarmGetFieldRec() is used to check the
contents of the "OnDate" field for each alarm.
! If they are the same, the alarm is displayed.
INT
FUNCTION
AlarmQueryDate(INT nRID, INT nVer, STRING sDate)
    INT bResult;
    IF sDATE = AlarmGetFieldRec(nRID, "OnDate", nVer) THEN
        bResult = TRUE;
    ELSE
        bResult = FALSE;
    END
    RETURN bResult;
END
```

See Also

[Alarm Functions](#)

Chapter: 19 Clipboard Functions

With the Clipboard functions, you can copy data to, and paste data from, the Windows Clipboard.

Clipboard Functions

Following are functions relating to the Windows clipboard:

ClipCopy	Copies a string to the Windows clipboard.
ClipPaste	Pastes a string from the Windows clipboard.
ClipReadLn	Reads a line of text from the Windows clipboard.
ClipSetMode	Sets the format of data sent to the Windows clipboard.
ClipWriteLn	Writes a line of text to the Windows clipboard.

See Also

[Functions Reference](#)

ClipCopy

Copies a string to the Windows clipboard. When the string is in the clipboard, you can paste it to any Windows program.

Syntax

ClipCopy(*sText*)

sText:

The string to copy to the clipboard.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ClipWriteLn](#)

Example

```
ClipCopy("put this in clipboard");
```

See Also

[Clipboard Functions](#)

ClipPaste

Pastes a string from the Windows clipboard.

Syntax

`ClipPaste()`

Return Value

The contents of the clipboard (as a string). If the clipboard is empty, an empty string is returned.

Related Functions

[ClipReadLn](#)

Example

```
/* Get string from clipboard into sText. */
sText = ClipPaste();
```

See Also

[Clipboard Functions](#)

ClipReadLn

Reads a single line of text from the Windows clipboard. With this function, you can read a block of text from the clipboard - line by line. Call the function once to read each line of text from the clipboard. When the end of the clipboard is reached, an empty string is returned.

Syntax

ClipReadLn()

Return Value

One line of text from the clipboard (as a string). If the clipboard is empty, an empty string is returned.

Related Functions

[ClipPaste](#)

Example

```
/* Get first line of text from clipboard. */
sText = ClipReadLn();
WHILE StrLength(sText) > 0 DO
    ! Do something with text
    ...
    ! Read next line of clipboard
    sText = ClipReadLn();
END
```

See Also

[Clipboard Functions](#)

ClipSetMode

Sets the format of data sent to the Windows clipboard.

Syntax

ClipSetMode(*nMode*)

nMode:

The mode of the data:

1 - ASCII Text

2 - CSV (Comma separated values) format

You can select multiple modes by adding modes together.

Return Value

The value of the previous mode.

Related Functions

[ClipCopy](#), [ClipWriteLn](#)

Example

```
/* Set the clipboard to CSV mode, write two values, and reset the
clipboard to the original mode. */
nOldMode = ClipSetMode(2);
ClipCopy("100,200");
ClipSetMode(nOldMode);
```

See Also

[Clipboard Functions](#)

ClipWriteLn

Writes a line of text to the Windows clipboard. With this function, you can write any amount of text to the clipboard. Call this function once for each line of text. To terminate the block of text, call this function and pass an empty string.

Syntax

ClipWriteLn(*sText*)

sText:

The line of text to write to the clipboard, or an empty string ("") to end the write operation.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ClipCopy](#)

Example

```
ClipWriteLn("first line of text");
ClipWriteLn("second line of text");
ClipWriteLn(""); ! End of write operation
```

See Also

[Clipboard Functions](#)

Chapter: 20 Cluster Functions

This section describes functions for manipulating clusters, checking their status, getting information, activating and deactivating them.

Cluster Functions

Following are functions relating to clusters:

ClusterActivate	Allows the user to activate an inactive cluster.
ClusterDeactivate	Allows the user to deactivate an active cluster.
ClusterFirst	Allows the user to retrieve the first configured cluster in the project.
ClusterGetName	Deprecated in this version
ClusterIsActive	Allows the user to determine if a cluster is active.
ClusterNext	Allows the user to retrieve the next configured cluster in the project.
ClusterSetName	Connects to a specific cluster server.
ClusterServerTypes	Allows the user to determine which servers are defined for a given cluster.
ClusterStatus	Allows the user to determine the connection status from the client to a server on a cluster.
ClusterSetName	Deprecated in this version
ClusterSwapActive	Allows the user to deactivate an active cluster at the same time as activating a deactive cluster.

See Also

[Functions Reference](#)

[ClusterActivate](#)

This function allows the user to activate an inactive cluster. When a cluster is made active, all data associated with that cluster is available to the client, and hardware alarms will occur if no connections can be made to the servers in the cluster.

Syntax

ClusterActivate(ClusterName)

ClusterName:

The name of the cluster to activate enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterDeactivate

This function allows the user to deactivate an active cluster. When a cluster is made inactive, no data associated with that cluster is available to the client, and hardware alarms will not occur if no connections can be made to the servers in the cluster.

Syntax

ClusterDeactivate(ClusterName)

ClusterName:

The name of the cluster to deactivate enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[ClusterActivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterFirst

This function allows the user to retrieve the first configured cluster in the project.

Syntax

ClusterFirst()

Return Value

The name of the first configured cluster.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterGetName

ClusterGetName is deprecated in this version of CitectSCADA.

Syntax

ClusterGetName(*sPrimary*, *sStandby*, *nMode*)

sPrimary:

The variable containing the name of the cluster's primary server (that is that which was set as *sPrimary* using the ClusterSetName() function).

sStandby:

The variable containing the name of the cluster's standby server (that is that which was set as *sStandby* using the ClusterSetName() function).

nMode:

The mode is for future expansion of the function - set to 0 (zero).

Return Value

The status of the get name.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

Example

```
// Return and display the server names.//  
ClusterGetName(sPrimary, sStandby, 0);  
Prompt("Name of Cluster" + sPrimary);
```

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterIsActive

This function allows the user to determine if a cluster is active.

Syntax

ClusterIsActive(ClusterName)

ClusterName:

The name of the cluster to query enclosed in quotation marks "".

Return Value

TRUE if active, FALSE otherwise. If the cluster name was invalid, this function will return FALSE and a hardware alarm will be generated.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterNext

This function allows the user to retrieve the next configured cluster in the project.

Syntax

ClusterNext(ClusterName)

ClusterName:

Any configured cluster name enclosed in quotation marks "", this will usually be the name of the previous cluster as returned from [ClusterFirst](#), or a previous call to ClusterNext.

Return Value

The name of the next configured cluster or an empty string if there is no more clusters.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterServerTypes

This function allows the user to determine which servers are defined for a given cluster.

Syntax

ClusterServerTypes(ClusterName)

ClusterName:

The name of the cluster to query enclosed in quotation marks "".

Return Value

Logical OR of the following server flags:

- 0001 - 1st bit set means an Alarm Server is configured
- 0010 - 2nd bit set means a Trend Server is configured
- 0100 - 3rd bit set means a Report Server is configured
- 1000 - 4th bit set means an IO Server is configured

For example, a return value of 14 indicates an IO Server, a Report Server, and a Trend Server are configured.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterSetName

ClusterSetName is deprecated in this version of CitectSCADA.

Syntax

ClusterSetName(*sPrimary*, *sStandby*, *nMode*)

sPrimary:

The name of the cluster's primary server (Reports Server, Alarms Server etc.), as defined using the Computer Setup Wizard. When the ClusterSetName() function is used, CitectSCADA will attempt to connect to this server.

sStandby:

The name of the cluster's standby server (Reports Server, Alarms Server etc.), as defined using the Computer Setup Wizard. If the *sPrimary* server is unavailable when the ClusterSetName() function is used, CitectSCADA will attempt to connect to this server.

If there is no standby server, enter an empty string for *sStandby*.

nMode:

The mode of the connection:

0 - If you select this mode, CitectSCADA will renew the last connection. If it was connected to the *sPrimary* server, when this function was last used, it will attempt to connect to it again. If it was last connected to the *sStandby* server, it will attempt to connect to it again.

This mode is useful when a server is known to be unavailable, as it facilitates faster cluster switching.

1 - CitectSCADA will attempt to connect to the *sPrimary* server first, each time this function is used. If the *sPrimary* server is unavailable, CitectSCADA will try the *sStandby* server.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskCluster](#)

Example

```
// Connect to Cluster A, with server CITECTA1 as primary server,
// and CITECTA2 as standby.//
ClusterSetName("CITECTA1", "CITECTA2", 0);
// Display the menu page for Cluster A Project.//
PageDisplay("MenuA");
```

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterStatus

This function allows the user to determine the connection status from the client to a server on a cluster.

Syntax

ClusterStatus(*clusterName*, *serverType*)

clusterName:

The name of the cluster to query enclosed in quotation marks "".

serverType:

The type of server (not a bit mask):

- 1 - Alarm Server
- 2 - Trend Server
- 4 - Report Server
- 8 - IO Server

Return Value

One of the following values:

- -1 - if the cluster does not contain a server of the given type.
- -2 - if the cluster does not exist"
- 0 - if the cluster contains the server but the cluster is inactive.

- 1 - if the cluster is active but the connection to the server is offline.
- 2 - if the cluster is active and the connection to the server is online.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterSwapActive](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

ClusterSwapActive

This function allows the user to deactivate an active cluster at the same time as activating an inactive cluster. The arguments may be passed in any order, but one cluster needs to be active and the other needs to be inactive.

Syntax

ClusterSwapActive(*clusterNameA*, *clusterNameB*)

clusterNameA:

The name of the cluster to activate or deactivate enclosed in quotation marks "".

clusterNameB:

The name of the cluster to activate or deactivate enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [TaskCluster](#)

See Also

[Cluster Functions](#)

"About cluster context" in the CitectSCADA User Guide

Chapter: 21 Color Functions

Allow manipulation of colors (for example, to convert CitectSCADA colors to the format required by ActiveX objects).

Color Functions

Following are functions relating to colors:

Citect-ColourToPackedRGB	Converts a CitectSCADA color into a packed RGB color value that can be used by an ActiveX object.
GetBlueValue	Returns the Blue component of a packed RGB color.
GetGreenValue	Returns the Green component of a packed RGB color.
GetRedValue	Returns the Red component of a packed RGB color.
MakeCitectColour	Creates a color from red, green and blue component parts.
PackedRGB	Returns a packed RGB color based on specified red, green, and blue values.
PackedRGB-ToCitectColour	Converts a packed RGB color into the nearest equivalent Citect-SCADA color.

See Also

[Functions Reference](#)

CitectColourToPackedRGB

Converts a CitectSCADA color value into a packed RGB color value that can be understood by an ActiveX object.

Syntax

CitectColorToPackedRGB(*nCitectColor*)

nCitectColor:

The CitectSCADA color value to be converted into a packed RGB color. CitectSCADAcolors are defined in the labels database, or calculated by the function [MakeCitectColour](#)

Return Value

The packed RGB color value - if successful, otherwise an [error](#) is returned.

Related Functions

[PackedRGBToCitectColour](#)

See Also

[Color Functions](#)

GetBlueValue

Returns the Blue component of a packed RGB color.

Syntax

GetBlueValue(*nPackedRGB*)

nPackedRGB:

The packed RGB color.

Return Value

The red value (0-255) - if successful, otherwise an [error](#) is returned.

Related Functions

[GetRedValue](#), [GetGreenValue](#)

See Also

[Color Functions](#)

GetGreenValue

Returns the green component of a packed RGB color.

Syntax

GetGreenValue(*nPackedRGB*)

nPackedRGB:

The packed RGB color.

Return Value

The red value (0-255) - if successful, otherwise an [error](#) is returned.

Related Functions

[GetRedValue](#), [GetBlueValue](#)

See Also

[Color Functions](#)

GetRedValue

Returns the red component of a packed RGB color.

Syntax

GetRedValue(*nPackedRGB*)

nPackedRGB:

The packed RGB color.

Return Value

The red value (0-255) - if successful, otherwise an [error](#) is returned.

Related Functions

[GetGreenValue](#), [GetBlueValue](#)

See Also

[Color Functions](#)

MakeCitectColour

Creates a color from red, green and blue component parts.

Note: To define a transparent color, use the label TRANSPARENT.

Syntax

MakeCitectColour(*nRed*,*nGreen*,*nBlue*)

nRed:

The color value for red, from 0-255

nGreen:

The color value for green, from 0-255

nBlue:

The color value for blue, from 0-255

Return Value

An integer that is an encoded representation of the color created.

Examples

```
! creates the color red
MakeCitectColor(255,0,0)
! creates the color white
MakeCitectColor(255,255,255)
```

See Also

[Color Functions](#)

PackedRGB

Returns a packed RGB color based on specified red, green, and blue values.

Syntax

PackedRGB(*nRed*, *nGreen*, *nBlue*)

nRed:

The red component of the desired packed RGB color.

nGreen:

The green component of the desired packed RGB color.

nBlue:

The blue component of the desired packed RGB color.

Return Value

The packed RGB color value - if successful, otherwise an [error](#) is returned.

Related Functions

[CitectColourToPackedRGB](#)

See Also

[Color Functions](#)

PackedRGBToCitectColour

Converts a packed RGB color into a calculated CitectSCADA color value.

Syntax

PackedRGBToCitectColour(*nPackedRGB*)

nPackedRGB:

The packed RGB color.

Return Value

The CitectSCADA color value if successful; otherwise an [error](#) is returned.

Related Functions

[CitectColourToPackedRGB](#)

See Also

[Color Functions](#)

Chapter: 22 Communication Functions

The communication functions give you direct access to the communication ports on your computer(s). You can use these functions to communicate with external equipment, such as low speed devices (e.g. bar code readers), serial keyboards, and dumb terminals.

You should not use these functions to communicate with high speed PLCs, as they are designed for low-level communication on a COM port and the performance may not be adequate. To communicate with a PLC, a standard I/O device setup should be configured using the required driver.

Note: The Communication functions can only be called from an I/O server.

Communication Functions

Following are functions relating to communications:

<u>ComClose</u>	Closes a communication port.
<u>ComOpen</u>	Opens a communication port for access.
<u>ComRead</u>	Reads characters from a communication port.
<u>ComReset</u>	Resets the communication port.
<u>ComWrite</u>	Writes characters to a communication port.
<u>SerialKey</u>	Redirects serial characters from a port to the keyboard.

See Also

[Functions Reference](#)

ComClose

Closes a communication port. Any Cicode tasks that are waiting for a read or write operation to complete (or that are retrying to read or write) return with a range error. Citect-SCADA automatically closes all communication ports at shutdown.

This function can only be called from an I/O Server.

Syntax

ComClose(*hPort*)

hPort:

The communication port handle, returned from the ComOpen() function. This handle identifies the table where all data on the associated communication port is stored.

Return Value

0 if the port is successfully closed, or an [error](#) if the port is already closed or if the port number is invalid.

Related Functions

[ComOpen](#), [ComRead](#), [ComWrite](#)

Example

See [ComOpen](#)

See Also

[Communication Functions](#)

ComOpen

Opens a communication port for access. The board and port need to both be defined in the database (using the Boards and Ports forms from the Communication menu).

If you try to open the same COM port twice with ComOpen(), the second open will not succeed and return -1. If this is passed without checking other Com functions, the COM port may not do anything. For this reason, do not open COM ports twice, and always check the return value from ComOpen().

The communication system should be used for low speed communications only. You should not use the communication functions to communicate with high speed PLCs - the performance may not be adequate. If you need high speed communication (for communicating with PLCs, etc.), you should write a protocol driver. Refer to the Citect-SCADA "Driver Development Kit".

This function can only be called from an I/O Server.

Syntax

ComOpen(*sPort*, *iMode*)

sPort:

The port name as specified in the Ports database.

iMode:

The mode of the open:

0 - Take control of the port from CitectSCADA. In this non-shared mode, you have complete access to the port - CitectSCADA cannot use the port. Communication will be restored when the port is closed.

1 - Share the port with CitectSCADA. In this mode, you can write to the port, and CitectSCADA can also use it. Please be aware that ComRead will be unreliable if the communication port is opened as shared.

Return Value

A communication port handle if the communication system is opened successfully, otherwise -1 is returned. The handle identifies the table where all data on the associated port is stored. You can use the handle in the other communication functions, to send and receive characters from the port.

Related Functions

[ComClose](#), [ComRead](#), [ComWrite](#)

Example

```

INT
FUNCTION
StartSerial(STRING sPort)
    INT     hPort;
    hPort = ComOpen(sPort, 0);
    IF hPort < 0 THEN
        Prompt("Cannot open port " + sPort);
        RETURN -1;
    END
    TaskNew("SerialRead", hPort, 0);
    TaskNew("SerialWrite", hPort, 0);
    ComClose(hPort);
    RETURN 0;
END
INT
FUNCTION
SerialWrite(INT hPort)
    STRING buffer;
    INT SerialWriteError;
    INT length;
    WHILE 1 DO
        ! put data into buffer and set length
        .
        .
        SerialWriteError = ComWrite(hPort, buffer, length, 2);
        IF SerialWriteError THEN

```

```
Prompt("Error Writing port");
ComReset(hPort);
END
END
RETURN 0;
END
INT
FUNCTION
SerialRead(INT hPort)
    STRING    buffer;
    INT      length;
    INT      total;
    INT      SerialReadError;
    total = 0;
    WHILE 1 DO
        length = 128; ! need to set length as read modifies
        SerialReadError = ComRead(hPort, buffer, length, 2);
        IF SerialReadError THEN
            Prompt("Error from port " + SerialReadError : #####);
            ComReset(hPort);
        ELSE
            ! get data from buffer, length is set to number read
            .
            .
            .
        END
    END
    RETURN 0;
END
```

See Also

[Communication Functions](#)

ComRead

Reads characters from a communication port. The characters are read from the communication port into a string buffer. If no characters have arrived after the specified timeout, the function returns with a timeout error. If the timeout is 0, the function gets any characters that have arrived from the last call, and returns immediately.

You use the iLength variable to specify the length of the buffer, or the maximum number of characters to read when ComRead() is called. When ComRead() returns, iLength is set to the actual number of characters read. Because iLength is modified by this function, you need to reset it before each call.

You should not treat the string buffer as a normal string - it has no string terminator. Use the StrGetChar() function to extract characters from the buffer.

It is strongly recommended not to call ComRead() while another ComRead() is still pending on the same port, because it can produce unexpected results.

 **WARNING**
UNINTENDED EQUIPMENT OPERATION

Do not call ComRead() if another instance of ComRead() is still pending on the same port.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete. This function can only be called from an I/O Server.

Syntax

ComRead(*hPort*, *sBuffer*, *iLength*, *iTimeOut*)

hPort:

The communication port handle, returned from the ComOpen() function. This handle identifies the table where the data on the associated communication port is stored.

sBuffer:

The buffer into which to put the characters. The actual number of characters read is returned in *iLength*.

iLength:

The number of characters to read into the buffer. The maximum length you may read in one call is 128 characters. When the function returns, this variable is set to the actual number of characters read.

iTimeOut:

The timeout for the read to complete:

- If *iTimeOut* = 0 (zero), the function checks for characters in the buffer and returns.
- If *iTimeOut* > 0, the function returns after this number of seconds - if no characters have been received.
- If *iTimeOut* < 0, the function waits forever for characters.

Return Value

0 (zero) if the read is successful, otherwise an [error](#) is returned.

Related Functions

[ComOpen](#), [ComClose](#), [ComWrite](#), [StrGetChar](#)

Example

See [ComOpen](#)

See Also

[Communication Functions](#)

ComReset

Resets the communication port. This function can only be called from an I/O Server.

Syntax

ComReset(*hPort*)

hPort:

The communication port handle, returned from the ComOpen() function. This handle identifies the table where all data on the associated communication port is stored.

Return Value

0 (zero) if the write is successful, otherwise an [error](#) is returned.

Related Functions

[ComOpen](#), [ComClose](#), [ComRead](#), [StrGetChar](#)

Example

See [ComOpen](#)

See Also

[Communication Functions](#)

ComWrite

Writes characters to a communication port. The characters are written from the string buffer to the port. If the characters have not been transmitted after the specified timeout, the function returns with a timeout error. If the timeout is 0, the function returns immediately and the characters are transmitted in the background.

ComWrite() does not treat the buffer as a true string, but rather as an array of characters of the length specified - you can send any character to the communication port. Use the StrSetChar() function to build the buffer. Do not call ComWrite() while another ComWrite() is still pending on the same port, because it can produce unexpected results.

You use the *iLength* variable to specify the length of the buffer, or the maximum number of characters to write when ComWrite() is called. When ComWrite() returns, *iLength* is reset to zero.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

This function can only be called from an I/O Server.

Syntax

ComWrite(*hPort*, *sBuffer*, *iLength*, *iTimeOut*)

hPort:

The communication port handle, returned from the ComOpen() function. This handle identifies the table where all data on the associated communication port is stored.

sBuffer:

The buffer from which to write the characters.

iLength:

The number of characters to write from the buffer. The maximum number of characters you can write is 128.

iTimeOut:

The timeout for the write to complete.

- If *iTimeOut* = 0 (zero), the characters are copied to the communication buffer and the function returns immediately - the characters are transmitted in the background.
- If *iTimeOut* > 0, the function returns after this number of seconds - if the characters cannot be transmitted.
- If *iTimeOut* < 0, the function waits forever to transmit the characters.

Return Value

0 (zero) if the write is successful, otherwise an [error](#) is returned.

Related Functions

[ComOpen](#), [ComClose](#), [ComRead](#), [StrGetChar](#)

Example

See [ComOpen](#)

See Also

[Communication Functions](#)

SerialKey

Redirects all serial characters from a port to the keyboard. If using a keyboard attached to a serial port, you should call this function at startup, so that CitectSCADA copies all characters (read from the port) to the keyboard. The Port needs to be defined in the Ports database.

If the port is not on an I/O server, you need to create a dummy I/O server record (for example, name the server DServer1). Complete the Boards and Ports records. Set the following parameters in the CITECT.INI file:

```
[IOServer]Name to the server name (for example, DServer1)
[IOServer]Server to 0
```

This method enables the port without making the computer an I/O server. (If the I/O server is enabled (and not required as an I/O server), extra overhead and memory are used.)

This function can only be called from an I/O server.

Syntax

SerialKey(*sPort*)

sPort:

The name of the port connected to the serial keyboard.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ComOpen](#)

Example

```
SerialKey("Port1"); ! enable the serial keyboard
```

See Also

[Communication Functions](#)

Chapter: 23 Dynamic Data Exchange Functions

The Cicode DDE (Dynamic Data Exchange) functions permit you to exchange data between CitectSCADA and other Windows applications running on the same computer in real time, continuously, and with no operator intervention. For example, you can send your run-time data to a DDE compliant spreadsheet or word processing application, either by posting the data to memory for DDE access by other applications, or by writing the data directly into another application. Conversely, you could read data from a DDE compliant application like a spreadsheet or document directly into a CitectSCADA variable.

You could also run processes in any DDE compliant Windows application running on the same computer by using the Cicode DDEExec() function to send commands to that application. Similarly, you can call any Cicode function (built-in or user-written) in CitectSCADA from any Windows application (running on the same computer), that supports a DDE Execute command.

The DDERead(), DDEPost(), DDEWrite(), and DDEExec() functions each perform a single exchange of data. Each of these functions starts a DDE conversation with the external application, sends or receives the data (or command), and ends the conversation - effectively treated as one operation.

The DDE handle (*DDEh...*) functions return a handle to the conversation - a DDE channel number. You should use the DDE handle functions for Network DDE, in particular for Access DDE.

Note: CitectSCADA runtime automatically behaves as a DDE Server and makes its variable tag database available for DDE Client applications to link with.

DDE Functions

Following are functions relating to Dynamic Data Exchange:

<u>DDEExec</u>	Executes a command in an external DDE compliant Windows application.
<u>DDEPost</u>	Makes a CitectSCADA variable available for DDE linking by other DDE compliant Windows applications.

DDERead	Reads a variable from a DDE compliant Windows application.
DDEWrite	Writes a variable to a DDE compliant Windows application.
DDEhExecute	Executes a command in an external DDE compliant Windows application.
DDEhGetLastError	Gets the latest Windows DDE error code.
DDEhInitiate	Starts a DDE conversation with an external DDE compliant Windows application.
DDEhPoke	Writes data to a DDE compliant Windows application.
DDEhReadLn	Reads a line of text from a DDE Conversion.
DDEhRequest	Requests data from a DDE compliant Windows application.
DDEhSetMode	Set the mode of a DDE conversation.
DDEhTerminate	Closes a DDE conversation with a Windows application.
DDEhWriteLn	Writes a line of text to the DDE conversation.

See Also

[Functions Reference](#)

DDEExec

Executes a command in an external Windows application running on the same computer. With this function, you can control other applications that support DDE. Refer to the documentation provided with the external Windows application to determine if DDE is supported and what functions can be called.

You cannot use DDEExec() to call macros on a remote computer or to call Access SQLs. For these calls, Network DDE needs to pass the *sDocument* argument, so you need to use the *DDEh...* functions, passing *sDocument* in the DDEhInitiate() function.

Syntax

DDEExec(*sApplication*, *sCommand*)

sApplication:

Application name (.EXE filename), for example, "WinWord".

sCommand:

The command that the application will execute.

Return Value

1 (one) if successful, otherwise an [error](#) is returned.

Related Functions

[DDEPost](#), [DDERead](#), [DDEWrite](#), [DDEhExecute](#)

Example

```
/* Instruct the Excel application to recalculate its spreadsheet
immediately. */
DDEExec("Excel", "[Calculate.Now()]");
```

See Also

[DDE Functions](#)

DDEhExecute

Executes a command in an external Windows application. You need to first start a conversation with the DDEhInitiate function, and use the handle returned by that function to identify the conversation.

With this function, you can control other applications that support DDE. Refer to the documentation provided with your other Windows application to determine if DDE is supported and what functions can be called.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

DDEhExecute(*Handle*, *sCommand*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

sCommand:

The command that the application will execute.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DDEhInitiate](#), [DDEhRequest](#), [DDEhPoke](#), [DDEhTerminate](#), [DDEhGetLastError](#)

Example

See [DDEhInitiate](#)

See Also

[DDE Functions](#)

DDEhGetLastError

Gets the latest error code issued from Windows for the conversation identified by the handle.

Syntax

DDEhGetLastError(*Handle*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

Return Value

The [error](#) code last issued from Windows DDEML (for that conversation):

DMLERR_ADVACKTIMEOUT	0x4000
DMLERR_BUSY	0x4001
DMLERR_DATAACKTIMEOUT	0x4002
DMLERR_DLL_NOT_INITIALIZED	0x4003
DMLERR_DLL_USAGE	0x4004
DMLERR_EXECACKTIMEOUT	0x4005
DMLERR_INVALIDPARAMETER	0x4006
DMLERR_LOW_MEMORY	0x4007
DMLERR_MEMORY_ERROR	0x4008

DMLERR_NOTPROCESSED	0x4009
DMLERR_NO_CONV_ESTABLISHED	0x400a
DMLERR_POKEACKTIMEOUT	0x400b
DMLERR_POSTMSG_FAILED	0x400c
DMLERR_REENTRANCY	0x400d
DMLERR_SERVER_DIED	0x400e
DMLERR_SYS_ERROR	0x400f
DMLERR_UNADVACKTIMEOUT	0x4010
DMLERR_UNFOUND_QUEUE_ID	0x4011

Related Functions

[DDEhInitiate](#), [DDEhExecute](#), [DDEhRequest](#), [DDEhPoke](#), [DDEhTerminate](#)

Example

See [DDEhInitiate](#)

See Also

[DDE Functions](#)

DDEhInitiate

Starts a conversation with an external Windows application. When the data exchange is complete, you should terminate the conversation to free system resources.

Syntax

DDEhInitiate(*sApplication*, *sDocument*)

sApplication:

The application name (.EXE filename), for example, "WinWord".

sDocument:

The document, topic, or file name.

Return Value

An integer handle for the conversation between CitectSCADA and the other application, or -1 if the conversation is not started successfully. The handle is used by the other *DDEh...* functions, to identify the conversation.

Related Functions

[DDEhExecute](#), [DDEhRequest](#), [DDEhPoke](#), [DDEhTerminate](#), [DDEhGetLastError](#)

Example

```
! Read from Excel spreadsheet
STRING FUNCTION GetExcelData();
    INT hChannel;
    STRING sData;
    hChannel = DDEhInitiate("EXCEL", "DATA.XLS");
    IF hChannel > -1 THEN
        sData = DDEhRequest(hChannel, "R1C1");
        DDEhTerminate(hChannel);
        hChannel = -1;
    END;
    RETURN sData;
END

! Write to Excel spreadsheet
FUNCTION SetExcelData(STRING sData);
    INT hChannel;
    hChannel = DDEhInitiate("EXCEL", "DATA.XLS");
    IF hChannel > -1 THEN
        DDEhPoke(hChannel, "R1C1", sData);
        DDEhTerminate(hChannel);
        hChannel = -1;
    END;
END

! Execute Excel Macro
FUNCTION DoExcelMacro();
    INT hChannel;
    hChannel = DDEhInitiate("EXCEL", "DATA.XLS");
    IF hChannel > -1 THEN
        DDEhExecute(hChannel, "[RUN(^\"TestMacro^\") ]");
        DDEhTerminate(hChannel);
        hChannel = -1;
    END;
END
```

See Also

[DDE Functions](#)

DDEhPoke

Writes a value to an external Windows application, for example, an Excel spreadsheet. The value is written once to the application. (To write the value dynamically, you need to call this function at the rate at which the data needs to be updated.)

You need to first start a conversation with the DDEhInitiate function, and use the handle returned by that function to identify the conversation.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

DDEhPoke(*Handle*, *sItem*, *sValue*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

sItem:

A unique name for the item; for example, the variable name, field name, or spreadsheet cell position.

sValue:

The value of the item.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DDEhInitiate](#), [DDEhExecute](#), [DDEhRequest](#), [DDEhTerminate](#), [DDEhGetLastError](#)

Example

See [DDEhInitiate](#)

See Also

[DDE Functions](#)

DDEhReadLn

Reads a line of text from a DDE Conversion, for example, from an Excel spreadsheet.

You need to first start a conversation with the DDEhInitiate function, and use the handle returned by that function to identify the conversation. This function allows you to read a large amount of data via DDE. Keep calling the function until an empty string is returned to verify that all the data has been read.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

DDEhReadLn(*Handle*, *sTopic*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

sTopic:

A unique topic name for the item; for example, the variable name, field name, or spreadsheet cell position.

Return Value

A line of data, or an empty string when all data has been read.

Related Functions

[DDEhSetMode](#), [DDEhWriteLn](#), [DDEhInitiate](#), [DDEhExecute](#), [DDEhRequest](#), [DDEhTerminate](#), [DDEhGetLastError](#)

Example

See [DDEhWriteLn](#)

See Also

[DDE Functions](#)

DDEhRequest

Reads a value from an external Windows application, for example, from an Excel spreadsheet. You need to first start a conversation with the DDEhInitiate function, and use the handle returned by that function to identify the conversation.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

DDEhRequest(*Handle*, *sItem*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

sItem:

A unique name for the item; for example, the variable name, field name, or spreadsheet cell position.

Return Value

A string of data, or an empty string if the function cannot read the value.

Related Functions

[DDEhInitiate](#), [DDEhExecute](#), [DDEhPoke](#), [DDEhTerminate](#), [DDEhGetLastError](#)

Example

See [DDEhInitiate](#)

See Also

[DDE Functions](#)

DDEhSetMode

Set the mode of the DDE conversation. The default mode of a DDE conversation is to use TEXT data format - a simple string of data. This function allows you to set the mode to CSV (Comma Separated Values). Some Windows applications support this mode of data as it helps them to separate the data. For example, when you send CSV format to Excel, each value will be placed into a unique cell. If you use TEXT mode all the data will be placed into the same cell.

Syntax

DDEhSetMode(*Handle*, *sMode*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

sMode:

The mode of the DDE conversation:

- 1 - Text (default)
- 2 - CSV

Return Value

The [error](#) code.

Related Functions

[DDEhInitiate](#), [DDEhExecute](#), [DDEhRequest](#), [DDEhTerminate](#), [DDEhGetLastError](#),
[DDEhPoke](#), [DDEhReadLn](#), [DDEhWriteLn](#), [DDEhSetMode](#)

Example

See [DDEhWriteLn](#)

See Also

[DDE Functions](#)

DDEhTerminate

Closes the conversation identified by the handle, and frees the resources associated with that conversation. After you call this function, the handle is no longer valid.

With Network DDE, you might need to terminate and re-initiate a conversation. For example, if you delete rows on an MS Access sheet, the deleted rows display as #DELETED until you terminate and re-initiate the conversation.

Syntax

DDEhTerminate(*Handle*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DDEhInitiate](#), [DDEhExecute](#), [DDEhPoke](#), [DDEhRequest](#), [DDEhGetLastError](#)

Example

See [DDEhInitiate](#)

See Also

[DDE Functions](#)

DDEhWriteLn

Writes a line of text to the DDE conversation. With this function, you can write any amount of text to the DDE conversation. Call this function once for each line of text. To terminate the block of text, call this function and pass an empty string.

Syntax

DDEhWriteLn(*Handle*, *sTopic*, *sData*)

Handle:

The integer handle that identifies the DDE conversation, returned from the DDEhInitiate function.

sTopic:

A unique name for the topic the data will be written to; for example, the spreadsheet cell position. The topic is only used when you complete the write by passing an empty string for data.

sData:

The line of data to write. To terminate the data and make CitectSCADA send the data, set the data to an empty string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DDEhInitiate](#), [DDEhExecute](#), [DDEhRequest](#), [DDEhTerminate](#), [DDEhGetLastError](#),
[DDEhPoke](#), [DDEhReadLn](#), [DDEhWriteLn](#), [DDEhSetMode](#)

Example

```

!     Write to Excel spreadsheet
!     write the numbers 1..8 into 8 unique cells in Excel.
FUNCTION WriteExcelData(STRING sData);
    INT hChannel;
    hChannel = DDEhInitiate("EXCEL", "DATA.XLS");
    IF hChannel > -1 THEN
        // set to CSV mode so EXCEL will put each value in a cell
        DDEhSetMode(hChannel, 2);
        DDEhWriteLn(hChannel, "", "1,2,3,4");
        DDEhWriteLn(hChannel, "R1C1:R2C4", "5,6,7,8");
        DDEhWriteLn(hChannel,"R1C1:R2C4","");
        DDEhTerminate(hChannel);
        hChannel = -1;
    END;
END

```

See Also

[DDE Functions](#)

DDEPost

Makes a CitectSCADA variable value available for DDE linking (that is posts a DDE link so that it can be read by other DDE compliant applications running on the same computer). This sets-up CitectSCADA to behave as a DDE Server for this DDE channel.

After a value is posted, other Windows applications running on the same computer can read the value by using their own DDE Client functions. If the value of the posted variable changes, any linked applications are informed of the new value.

To link to this value from any DDE Client applications running on the same computer, they need to appropriately use the DDE Client syntax with:

- "Citect" as the **<DDE Server application name>**
- "Data" as the **<DDE Topic name>**
- The name used for the first parameter **sItem** in this DDEPost() function as the **<DDE data item name>**.

Unlike the DDERead() and DDEWrite() Cicode functions which are static, the DDEPost() function can be used to create a dynamic DDE link, providing the DDE Client applications appropriately set their side of the DDE channel to be automatically updated.

Syntax

DDEPost(sItem, sValue)

sItem:

A unique name for the item; for example, the variable name, field name, or spreadsheet cell position.

sValue:

The value of the item.

Return Value

The value that is posted, or 0 (zero) if the function does not succeed in posting the link.

Related Functions

[DDEExec](#), [DDERead](#), [DDEWrite](#)

Example

```
! In Citect Project Editor, create a variable tag named PV1  
! In Cicode, post a link to the tag PV1 for external DDE
```

```

applications to connect with DDEPost("TAGONE",PV1);
/* To link to this posted tag from a cell in Excel, set the cell to
=Citect!Data!TAGONE. This will set the value of the Excel cell to
the value of tag PV1. */
/* To link to this posted tag from a field in Word, set the field
to{DDEAuto Citect Data TAGONE}. This will set the value of the
field link to the value of tag PV1. */

```

See Also

[DDE Functions](#)

DDERead

Reads values from an external DDE compliant Windows application running on the same computer, (for example, from an Excel spreadsheet cell or a Word document).

This is a one-way static communication which is read once from the application per call. To read the value dynamically, call this function at the rate at which the data is required to be updated.

Use this function when you want precise control over exactly what you want from the DDE exchange.

Syntax

DDERead(*sApplication*, *sDocument*, *sItem* [, *Mode*])

sApplication:

The application name (.EXE filename), for example, "WinWord".

sDocument:

The document, topic, or file name.

sItem:

A unique name for the item; for example, the variable name, field name, or spreadsheet cell position.

Mode:

A flag that tells the application whether or not to set up an advise loop:

0 - Do not set up advise loop.

1 - Set up advise loop (default).

Return Value

The value (from the external application) as a string, or an empty string if the function cannot read the desired values.

Related Functions

[DDEExec](#), [DDEPost](#), [DDEWrite](#)

Example

```
/* Read the value from R1C1 (Row1,Column1) of an Excel spreadsheet
named "Sheet1". */
DDERead("Excel","Sheet1","R1C1");
/* Read the value from the Item1 bookmark of the Word document
named "Recipes.doc". */
DDERead("Winword","Recipes","Item1");
```

See Also

[DDE Functions](#)

DDEWrite

Writes a value to an external Windows application, for example, to an Excel spreadsheet. The value is written once to the application. To write the value dynamically, you need to call this function at the rate at which the data needs to be updated.

Use DDEWrite() to cause CitectSCADA runtime to initiate the DDE conversation with a DDE compliant application running on the same computer.

Syntax

DDEWrite(*sApplication*, *sDocument*, *sItem*, *sValue*)

sApplication:

The application name (.EXE filename), for example, "WinWord".

sDocument:

The document, topic, or file name.

sItem:

A unique name for the item; for example, the variable name, field name, or spreadsheet cell position.

sValue:

The value of the item.

Return Value

The value that is sent to the other application, or an empty string if the function does not successfully write the value.

Related Functions

[DDEExec](#), [DDEPost](#), [DDERead](#)

Example

```
/* Write the value of a CitectSCADA variable named  
TAGONE to R1C1 (Row1,Column1) of an Excel spreadsheet named  
"Sheet1". The value is in string format. */  
DDEWrite("Excel","Sheet1","R1C1",TAGONE);
```

See Also

[DDE Functions](#)

Chapter: 24 Device Functions

The device functions provide access to devices. They allow access to SQL, dBASE, and ASCII files through database-like operations, and provide more control over output to printers.

With these functions you can open and close any device, and read from and write to any record or field in the device. You can store recipes or any other data in a database, and then down-load or up-load the data as required to an I/O device on the plant floor, or to the operator. You can also update the database with real-time data for data exchange with other applications.

Device Functions

Following are functions relating to devices:

DevAppend	Appends a blank record to the end of a device.
DevClose	Closes a device.
DevControl	Controls a dBASE or SQL device.
DevCurr	Gets the current device number.
DevDelete	Deletes the current record in a database device.
DevDisable	Disables (and re-enables) a device from any access.
DevEOF	Checks for the end of a file.
DevFind	Finds a record in a device.
DevFirst	Finds the first record in a device.
DevFlush	Flushes buffered data to a device.
DevGetField	Gets field data from the current record.
DevHistory	Renames a device file and any subsequent history files.

DevInfo	Gets device information.
DevModify	Modifies the attributes of a device.
DevNext	Gets the next record in a device.
DevOpen	Opens a device for access.
DevOpenGrp	Opens a group of devices.
DevPrev	Gets the previous record in a device.
DevPrint	Prints free-format data to a group of devices.
DevRead	Reads characters from a device.
DevReadLn	Reads a line of characters from a device.
DevRecNo	Gets the current record number of a device.
DevSeek	Moves to any record in a device.
DevSetField	Sets new field data in the current record.
DevSize	Gets the size of a device.
DevWrite	Writes a string to a device.
DevWriteLn	Writes a string with a newline character to a device.
DevZap	Zaps a device.
Print	Prints a string in a report.
PrintLn	Prints a string with a newline character in a report.
PrintFont	Changes the printing font on the current device.

See Also

[Functions Reference](#)

DevAppend

Appends a blank record to the end of a device. After the record is appended, you can use the DevSetField() function to add data to fields in the record.

You need to first call the DevOpen() function to get the device handle (hDev).

Syntax

DevAppend(hDev)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

0 (zero) if the record is successfully appended, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevSetField](#)

Example

```
INT
FUNCTION WriteAlarmCount( INT hDevice, STRING sAlarm,
    INT iCount, INT iTime )
    DevAppend(hDevice);
    DevSetField(hDevice, "ALARM", sAlarm);
    DevSetField(hDevice, "TIME", IntToStr(iTime));
    DevSetField(hDevice, "COUNT", IntToStr(iCount));
END
```

See Also

[Device Functions](#)

DevClose

Closes a device. Any data in the buffer is flushed to the device before it is closed. After a device is closed, its device handle becomes invalid and cannot be used.

Syntax

DevClose(hDev, Mode)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where data on the associated device is stored.

Mode:

The mode of the close:

0 - Close the device in user mode - the default mode if none is specified. A device opened by Cicode function DevOpen() need to be closed in this mode.

1 - Close the device in remove logging mode - under this mode, the current device will be rolled over to history files immediately. You should only use this mode in a report.

2 - Close the device in keep logging mode - under this mode, the current device will not be rolled over to history files. This allows subsequent messages to be written to the same file. This mode is used internally in a report written in rich text format (rtf).

Note:Do not call DevClose() to the current device in an rtf report. This may make the output file unreadable.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#)

Example

```
DevClose(hDev);
```

See Also

[Device Functions](#)

DevControl

Controls a dBASE or SQL device. You can pack a dBASE device to physically remove deleted records, or re-index a dBASE device to regenerate the keys. You can issue queries to an SQL device, or get the error status of the last SQL query.

Syntax

DevControl(*hDev*, *Type* [, *sData*])

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Type:

The type of command:

- 0 - Re-index the device based on the key defined in the device record (dBASE devices only).
- 1 - Pack the database file - all deleted records are removed (dBASE devices only).
- 2 - Issue a direct SQL query to the device (SQL devices only).
- 3 - Get error status of the last SQL query (SQL devices only).

Note: ASCII files and printers are not supported.

sData:

The command data, that is the SQL query to be issued. Used only for Type 2 commands.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevZap](#)

Example

```
! pack a dBASE file device
DevControl(hDev, 1, "");
```

See Also

[Device Functions](#)

DevCurr

Gets the current device handle. You can only call this function in a report, to get the handle of the device where the report is logging. You can then use the other device functions (for example, DevPrint()) to access that logging device. (To get the handle of a device other than a logging device, you need to use the DevOpen() function.)

If the report is logging to a group of devices, this function will return the group handle. However, not all device functions support group handles, for example, you cannot read from a group of devices.

Syntax

DevCurr()

Return Value

The current device handle or group handle. If no device is configured, -1 is returned.

Related Functions

[DevOpen](#), [DevPrint](#)

Example

```
! Get the report device number.  
hDev=DevCurr();
```

See Also

[Device Functions](#)

DevDelete

Deletes the current record in a dBASE database device. The record is not physically deleted, but is marked for deletion. You can physically delete the record by packing the database with the DevControl() function.

Syntax

DevDelete(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

0 (zero) if the record is successfully deleted, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevClose](#), [DevControl](#)

Example

```
! Delete the current record.  
DevDelete(hDev);
```

See Also

[Device Functions](#)

DevDisable

Disables (and re-enables) a device from all access, and discards any data written to the device. When a device is disabled, it cannot be opened, and data cannot be read from the device. Use this function to disable logging to a database or printer.

The *State* argument is a toggle. A *State* of 1 disables the device(s), but you can then re-enable the device(s) by repeating the function with *State* = 0.

Syntax

DevDisable(*sName*, *State*)

sName:

The device name, or * (asterisk) for all devices.

State:

The disable state:

- 0 - Enable the device.
- 1 - Disable the device.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#)

Example

```
! Disable the AlarmLog device.
DevDisable("AlarmLog",1);
:
DevDisable("AlarmLog",0);           ! Re-enable the device.
```

See Also

[Device Functions](#)

DevEOF

Gets the status of the end of file (EOF) flag for a device. When you use the DevPrev(), DevNext(), or DevSeek() function, the start or end of the device will eventually be reached, and the EOF flag will be set. Use this function to test the EOF flag.

Syntax

DevEOF(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

1 if the EOF flag has been set, otherwise 0 (zero).

Related Functions

[DevOpen](#), [DevPrev](#), [DevNext](#), [DevSeek](#), [DevReadLn](#)

Example

```
hDev = DevOpen("Log", 0);
WHILE NOT DevEOF(hDev) DO
    Prompt(DevGetField(hDev, "Tag"));
    DevNext(hDev);
END
DevClose(hDev);
```

See Also

[Device Functions](#)

DevFind

Searches a device for a record that contains specified data in a specified field. The search starts at the current record and continues forward until the matched data is found or the end of the database is reached. If the file has a keyed index, an indexed search is used.

Syntax

DevFind(*hDev*, *sFind*, *sField*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

sFind:

The data to find in *sField*, as a string.

For SQL devices: The DevFind() function can distinguish between numbers, strings, and dates, so you do not need to enclose the data in quote marks. Dates and times need to be in the correct format:

- Date: YYYY-MM-DD
- Time: HH:MM:SS
- DateTime: YYYY-MM-DD HH:MM:SS[.F...] (The fraction .F... is optional.)

sField:

The field name to match.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevSeek](#)

Example

```

! Find the Ice cream recipe.
DevNotFound=DevFind(hDev,"Ice cream","Recipe");
IF DevNotFound=0 THEN
    ! Get the recipe values.
    ..
ELSE
    Prompt("Ice cream not found");
END

```

See Also

[Device Functions](#)

DevFirst

Finds the first record in a device.

Syntax

DevFirst(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

The first indexed record (if the device is an indexed database), otherwise the first record in the device.

Related Functions

[DevOpen](#), [DevClose](#)

Example

```
! Find the first record.  
FirstRec = DevFirst(hDev);
```

See Also

[Device Functions](#)

DevFlush

Flushes buffered data to the physical device. CitectSCADA normally optimizes the writing of data for maximum performance, so use this function only if it is really necessary.

Syntax

DevFlush(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where data on the associated device is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevClose](#)

Example

```
! Flush device to disk.  
DevFlush(hDev);
```

See Also

[Device Functions](#)

DevGetField

Gets field data from the current record in a device.

Syntax

DevGetField(*hDev*, *Field*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Field:

The field name, as a string of up to 10 characters. (The dBASE file format limits all field names to a maximum of 10 characters.)

Return Value

The field data (as a string). If the field is not found an empty string is returned.

Related Functions

[DevOpen](#), [DevSetField](#)

Example

```

INT
FUNCTION
GetRecipe (STRING sName)
    INT hDev;
    hDev = DevOpen("Recipe", 0);
    IF hDev >= 0 THEN
        DevSeek(hDev, 1);
        IF DevFind(hDev, sName, "NAME") = 0 THEN
            PLC_FLOUR = DevGetField(hDev, "FLOUR");
            PLC_WATER = DevGetField(hDev, "WATER");
            PLC_SALT = DevGetField(hDev, "SALT");
            PLC_MILK = DevGetField(hDev, "MILK");
        ELSE
            DspError("Cannot Find Recipe " + sName);
        END
        DevClose(hDev);
    ELSE
        DspError("Cannot open recipe database");
    END
END

```

See Also

[Device Functions](#)

DevHistory

Renames a device file and any subsequent history files. The current device is closed and renamed as the first history file. For example, the device file 'Templog.txt' is renamed as 'Templog.001'. If a history file 'Templog.001' already exists, it is renamed as 'Templog.002', and so on. The next time data is written to the device, a new device file is created.

Note: If the device file has not been created (that is data has not been written to the device), only existing history files are renamed. Use this function for direct control of the device history process.

Syntax

DevHistory(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevControl](#)

Example

```
! Create history file  
DevHistory(hDev);
```

See Also

[Device Functions](#)

DevInfo

Gets information on a device.

Syntax

DevInfo(*hDev, Type*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Type:

Type of information:

-n: Name of field n (where n is any number up to the total number of fields). For example, if there are 10 fields, -7 will return the name of field 7.

-(Total no. of fields + n): Length of field n (where n is any number up to the total number of fields). For example, if there are 10 fields, -15 will return the length of field 5.

0: Device Name

1: Format

2: Header

3: File Name

4: Number of history files

5: Form length

6: Number of fields

7: Disable flag

8: Device type

9: Record size

10: Format number

11: Type of history schedule:

0: Event triggered

1: Daily

2: Weekly

3: Monthly

4: Yearly

12: The history period, in seconds, or week day, month or year, for example, if history is weekly then this is the day of the week, that is 1 to 7

13: Synchronisation time of day of the history in seconds, for example, 36000 (that is, 10:00:00)

14: The time the next history file will be created in seconds

Return Value

The device information as a string if successful, otherwise an empty string is returned.

Related Functions

[DevControl](#)

Example

```
! Get the number of fields in a device.  
NoFields=DevInfo(hDev, 6);  
FOR I=1 TO NoFields DO  
    ! Get and display the name of each field.  
    sField=DevInfo(hDev,-I);  
    nLength=DevInfo(hDev,-I - NoFields);  
    Prompt("Field Name "+sField + "Length " + nLength:##);  
END
```

See Also

[Device Functions](#)

DevModify

Modifies the attributes of a device. The device needs to be closed before you can modify a device.

This function allows you to dynamically change the file name or other attributes of a device at run time. You can use a single device to access many files. For example, you can create a device called Temp with a file name of TEMP.DBF. Using this function you could dynamically change the file name to access any dBASE file.

This function is useful in conjunction with the FormOpenFile() or FormSaveAsFile() functions. (These functions allow the operator to select file names easily.)

When using this function, you should be careful that no other Cicode function is already using the same device. Check the return value of this function before opening the device or you will destroy the data in the device to which it is already attached. If the device is already open, calling DevModify will return an error (and raise a hardware alarm to notify user).

If DevModify returns error, it means it has not modified the device and the device parameters will remain as they were before the call to DevModify.

Use a semaphore to help protect your Cicode.

Syntax

DevModify(*Name*, *Format*, *Header*, *FileName*, *Type*)

Name:

The name of the device.

Format:

A new format for the device or "*" to use the existing format. See [Format Templates](#) for more information.

Header:

A new header for the device or "*" to use the existing header.

FileName:

A new file name for the device or "*" (asterisk) to use the existing filename.

Type:

A new device type.

Device Type	Device
ASCII_DEV	ASCII file
PRINTER_DEV	Printer
dBASE_DEV	dBASE file
SQL_DEV	SQL database

or -1 to use the existing device type.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevClose](#), [DevSetField](#), [DevInfo](#), [DevAppend](#), [FormOpenFile](#)

Example

```

! change the file name of MyDev
DevModify("MyDev", "", "*", "c:\data\newfile.dbf", -1);
! change the fields and file name of MyDev
DevModify("MyDev", "{time}{date}{tags}", "*",
"C:\DATA\OLDFILE.DBF", -1);
! change the device to TXT file
DevModify("MyDev", "*", "*", "C:\DATA\OLDFILE.TXT", ASCII_DEV);

```

See Also

[Device Functions](#)

DevNext

Gets the next record in a device. If the end of the database is reached, the EOF flag is set and an error code is returned.

Syntax

DevNext(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

0 if the next record is read, or an [error](#) if the end of the database is reached.

Related Functions

[DevEOF](#), [DevPrev](#)

Example

```
Status=0;
I = 0;
hDev = DevOpen("Log", 0);
WHILE Status = 0 DO
    DspText(20 + I, 0, DevGetField(hDev,"Tag"));
    I = I + 1;
    Status = DevNext(hDev);
END
DevClose (hDev);
```

See Also

[Device Functions](#)

DevOpen

Opens a device and returns the device handle. The device needs to be defined in the CitectSCADA database. If the device cannot be opened, and user error checking is not enabled, the current Cicode task is halted.

You can use this function to return the handle of a device that is already open. The DevOpen() function does not physically open another device - it returns the same device handle as when the device was opened. The mode of the second open call is ignored. To re-open an open device in a different mode, you need to first close the device and then re-open it in the new mode.

When using an ODBC driver to connect to an SQL server or database, experience has shown that connecting only once on startup and not closing the device yields the best performance. ODBC connection is slow and if used on demand may affect your system's performance. Also, some ODBC drivers may leak memory on each connection and may cause errors after a number of re-connects.

Note: If you are opening a database device in indexed mode (nMode=2), an index file will automatically be created by CitectSCADA if one does not already exist. If you feel a device index has become corrupt, delete the existing index file and a new one will be created the next time the DevOpen function is run.

Syntax

DevOpen(*Name* [, *nMode*])

Name:

The name of the device.

nMode:

The mode of the open:

0 - Open the device in shared mode - the default mode when opening a device if none is specified.

1 - Open the device in exclusive mode. In this mode only one user can have the device open. The open will return an error if another user has the device open in shared or exclusive mode.

2 - Open the device in indexed mode. In this mode the device will be accessed in index order. This mode is only valid if the device is a database device and has an index configured in the Header field at the Devices form.
Please be aware that specifying mode 2 when opening an ASCII device is ignored internally.

4 - Open the device in 'SQL not select' mode. If opened in this mode, you need to not attempt to read from an SQL device.

8 - Open the device in logging mode. In this mode the history files will be created automatically.

16 - Open the device in read only mode. In this mode data can be viewed, but not written. This mode is supported only by DBF and ASCII files - it is ignored by printers and SQL/ODBC databases.

Return Value

The device handle. If the device cannot be opened, -1 is returned. The device handle identifies the table where all data on the associated device is stored.

Related Functions

[DevClose](#), [DevOpenGrp](#)

Example

```
INT
FUNCTION
PrintRecipe(STRING sCategory)
    STRING sRecipe;
    INT hRecipe, hPrinter;
    ErrSet(1); ! enable user error checking
    hRecipe = DevOpen("Recipe", 0);
    IF hRecipe = -1 THEN
        DspError("Cannot open recipe");
        RETURN FALSE;
    END
    hPrinter = DevOpen("Printer1", 0);
    IF hPrinter = -1 THEN
        DspError("Cannot open printer");
        RETURN FALSE;
    END
    ErrSet(0); ! disable user error checking
    WHILE NOT DevEof(hRecipe) DO
        sRecipe = DevReadLn(hRecipe);
        DevWriteLn(hPrinter, sRecipe);
    END
    DevClose(hRecipe);
    DevClose(hPrinter);
    RETURN TRUE;
END
```

See Also

[Device Functions](#)

DevOpenGrp

Opens a group of devices.

Syntax

DevOpenGrp(*hGrp* [, *nMode*])

hGrp:

The handle to a database containing a group of devices.

nMode:

The mode of the open:

0 - Open the device in shared mode - the default mode when opening a device.

1 - Open the device in exclusive mode. In this mode only one user can have the device open. The open will return an error if another user has the device open in shared or exclusive mode.

2 - Open the device in indexed mode. In this mode the device will be accessed in index order. This mode is only valid if the device is a database device and has an index configured in the Header field at the Devices form. Please be aware that specifying mode 2 when opening an ASCII device is ignored internally.

4 - Open the device in 'SQL not select' mode. If opened in this mode, you need to not attempt to read from an SQL device.

8 - Open the device in logging mode. In this mode the history files will be created automatically.

16 - Open the device in read only mode. In this mode data can be viewed, but not written. This mode is supported only by DBF and ASCII files - it is ignored by printers and SQL/ODBC databases.

Return Value

Returns 0 if successful or -1 if the function is provided with a bad handle and cannot open the group.

Related Functions

[DevClose](#), [DevOpen](#)

DevPrev

Gets the previous record in a device. If the start of the database is reached, the EOF flag is set and an error code is returned.

Syntax

DevPrev(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

0 if the record is read successfully, or an [error](#) if the start of the database is reached.

Related Functions

[DevOpen](#), [DevEOF](#), [DevNext](#)

Example

```
Status=0;
I = 0;
hDev = DevOpen("Log", 0);
iError = DevSeek(hDev, DevSize(hDev)); ! seek to end
WHILE iError = 0 DO
    DspText(20 + I, 0, DevGetField(hDev,"Tag"));
    I = I + 1;
    iError = DevPrev(hDev);
END
DevClose(hDev);
```

See Also

[Device Functions](#)

DevPrint

Prints free-format data to groups of devices. Using this function, you can write data to many devices at the same time. You would normally use this function in a report.

Syntax

DevPrint(*hGrp*, *sData*, *NewLine*)

hGrp:

The device handle, or the group handle for a group of devices.

sData:

The data to print to the group of devices.

NewLine:

The newline flag:

0 - Do not insert a newline character.

1 - Insert a newline character.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevWriteLn](#), [DevCurr](#)

Example

```
! Get the report device number or group number (for a group of
devices).
hGrp=DevCurr();
! Print PV123 to a group of devices.
DevPrint(hGrp,"PV123="+PV123:###,1);
```

See Also

[Device Functions](#)

DevRead

Reads characters from a device. If the device is record-based, the current field is read. If the device is free-format, the specified number of characters is read. If the number of characters specified is greater than the number of characters remaining in the device, only the remaining characters are read.

Syntax

DevRead(*hDev*, *Length*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Length:

The number of characters to read.

Return Value

The data (in string format). If the end of the device is found, an empty string is returned.

Related Functions

[DevOpen](#), [DevReadLn](#), [DevFind](#)

Example

```
! Read 20 characters from a device.  
Str=DevRead(hDev, 20);
```

See Also

[Device Functions](#)

DevReadLn

Reads data from the current record of a device until the end of the line, or end of the record. If the device is record-based, the record number is incremented. The carriage return and newline characters are not returned.

Syntax

DevReadLn(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

The data (in string format). If the end of the device is found, an empty string is returned and the EOF flag is set.

Related Functions

[DevOpen](#), [DevRead](#), [DevEOF](#), [DevFind](#)

Example

```
Str=DevReadLn (hDev);
```

See Also

[Device Functions](#)

DevRecNo

Gets the current record number of a device. If the device is record-based, the record number ranges from 1 to the maximum size of the file. If the device is free-format, the record number ranges from 0 to the maximum byte size -1.

Syntax

DevRecNo(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

The record number. If an [error](#) is detected while getting the record number, -1 is returned.

Related Functions

[DevOpen](#), [DevSeek](#)

Example

```
! Get the current record number.  
Rec=DevRecNo(hDev);
```

See Also

[Device Functions](#)

DevSeek

Moves the device pointer to a specified position in the device. If the device is a database, and it is opened in indexed mode, DevSeek will seek to the record number - not through the index. To locate the first record in an indexed device, call the DevFirst() function.

Syntax

DevSeek(*hDev, Offset*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Offset:

The offset in the device. If the device is a database device, the offset is the record number. If the device is a binary device, the offset is in bytes (from 0 to the maximum file size -1).

Note: If offset causes a seek past the end of the file, DevSeek returns no error, but sets the EOF flag (that is, a subsequent DevEOF() call will return true).

Return Value

0 (zero) if the seek was successful, otherwise an [error](#) code is returned.

Related Functions

[DevOpen](#), [DevEOF](#), [DevRecNo](#), [DevFirst](#)

Example

```
hDev=DevOpen ("Log", 0);
DevSeek(hDev,100);
DevGetField(hDev,"Tag");
! Gets the value of the "Tag" field at record 100.
```

See Also

[Device Functions](#)

DevSetField

Sets new field data in the current record in a device.

Syntax

DevSetField(*hDev*, *Field*, *sData*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Field:

The field name, as a string of up to 10 characters. (The dBASE file format limits all field names to a maximum of 10 characters.)

sData:

New field data, in string format. CitectSCADA converts any other data type into a string before setting the data.

Return Value

0 (zero) if the data is successfully set, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevAppend](#), [DevGetField](#)

Example

```

! Set the fields in the "Recipe" device.
hDev=DevOpen("Recipe", 0);
DevSeek(hDev, 1);
DevSetField(hDev,"Name", "WhiteBread");
DevSetField(hDev,"Flour", IntToStr(iFlour));
DevSetField(hDev,"Water", iWater:####);
DevSetField(hDev,"Salt", iSalt);
DevClose(hDev);

```

See Also[Device Functions](#)**DevSize**

Gets the size of a physical device.

Syntax**DevSize(*hDev*)**

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

If the device is a database device, the number of records is returned. If the device is a binary device, the number of bytes in the file is returned. If an [error](#) is detected, -1 is returned.

Related Functions[DevRecNo](#), [DevSeek](#)**Example**

```

INT NoRec;
NoRec=DevSize(hDev);
! Seek to the last record.
DevSeek(hDev,NoRec);

```

See Also[Device Functions](#)**DevWrite**

Writes a string to a device. If the device is free-format, the data is written to the device as specified. If the device is record-based, the data is written to the current field, and the field pointer is moved to the next field.

Writing to a DBF device appends the data to the device.

DevWrite(*hDev*, *sData*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

sData:

The data to write, as a string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevWriteLn](#)

Example

```
! Write PV123 to the device.  
DevWrite(hDev, "PV123="+PV123:###.#);
```

For SQL devices: The DevWrite() function can distinguish between numbers, strings, and dates, so you do not need to enclose the data in quote marks. Dates and times need to be in the correct format:

- Date: YYYY-MM-DD
- Time: HH:MM:SS
- DateTime: YYYY-MM-DD HH:MM:SS[F...] (The fraction .F... is optional.)

See Also

[Device Functions](#)

DevWriteLn

Writes a string to a device. If the device is free-format, the data is written to the device, followed by a newline character. If the device is record-based, a new record is appended to the device and the data is written to this record. The record pointer is then moved to the next record.

Syntax

DevWriteLn(*hDev*, *sData*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

sData:

The data to write, as a string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevOpen](#), [DevWrite](#)

Example

```
/* Write PV123 to the device followed by a newline character */
DevWriteLn(hDev,"PV123="+PV123:###.#);
```

See Also

[Device Functions](#)

DevZap

Zaps a device. If a database device is zapped, all records are deleted. If an ASCII file is zapped, the file is truncated to 0 (zero) length. Use this function when you want to delete all records in a database or file without deleting the actual file.

Syntax

DevZap(*hDev*)

hDev:

The device handle, returned from the DevOpen() function. The device handle identifies the table where all data on the associated device is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DevDelete](#)

Example

```
! Delete all records in the alarm log database.  
hDev = DevOpen("AlarmLog", 0);  
DevZap(hDev);
```

See Also

[Device Functions](#)

Print

Prints a string on the current device. You should call this function only in a report. The output is sent to the device (or group of devices) defined in the Reports database (in the output device field).

Note: To print a new line in an RTF report, use the "\par" special character. For example, Print("String" + "\par").

Syntax

Print(*String*)

String:

The string (data) to print.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PrintLn](#)

Example

```
! Print "Testvar" and stay on the same line.  
Print("Value of Testvar="+Testvar:##.#);
```

See Also

[Device Functions](#)

PrintFont

Changes the printing font on the current device. You should call this function only in a report. It will change the font style for the device (or group of devices) defined in the Reports database (output device field). It has effect only on reports being printed to a PRINTER_DEV - it has no effect on other types of devices, such as ASCII_DEV and dBASE_DEV.

Syntax

PrintFont(*Font*)

Font:

The CitectSCADA font (defined in the Fonts database).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Print](#)

Example

The following report file...

```
{! example.rpt }
-----
AN example Report
-----
{CICODE}
    PrintFont ("HeadingFont");
{END}
    Plant Area 1
{CICODE}
    PrintFont ("ReportFont");
{END}
{Time(1) }      {Date(2) }
PV_1           {PV_1:#####.##}
PV_2           {PV_2:#####.##}
-----End of Report-----
```

...will print as...

```
-----
AN example Report
-----
```

```
Plant Area 1
04:41:56      19-10-93
PV_1          49.00
PV_2          65.00
-----End of Report-----
```

See Also

[Device Functions](#)

PrintLn

Prints a string on the current device, followed by a newline character. You should call this function only in a report. The output will be sent to the device or group of devices defined in the Reports database (in the output device field).

Note: To print a new line in an RTF report, use the "\par" special character. For example, `PrintLn("String" + "\par")`.

Syntax

PrintLn(*String*)

String:

The string (data) to print.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Print](#)

Example

```
! Print "Testvar" followed by a new line.
PrintLn("Value of Testvar="+Testvar:##.#);
```

See Also

[Device Functions](#)

Chapter: 25 Display Functions

Display functions control the display and processing of graphics pages and objects. You can use these functions to display graphics pages, print them on your printer, send them to a file, or copy them to the Windows Clipboard. You can also display text files on screen.

Note: The properties defined for an object will override any conflicting Cicode Display functions.

You can create and move ANs (animation-point numbers), and obtain runtime information about graphics pages and their associated ANs.

Display Functions

Following are functions relating to the display of graphics pages and objects:

<u>DspAnCreateControlObject</u>	Creates a new instance of an ActiveX object. If the object already exists for the given Animation Point Number, then that object will be used (a new object is not created).
<u>DspAnFree</u>	Frees (removes) an AN from the current page.
<u>DspAnGetArea</u>	Gets the area configured for an object at a specific AN (animation-point number).
<u>DspAnGetMetadata</u>	Retrieves the field value of the specified metadata entry.
<u>DspAnGetMetadataAt</u>	Retrieves metadata information at the specified index.
<u>DspAnGetPos</u>	Gets the x and y coordinates of an AN (animation-point number).
<u>DspAnGetPrivilege</u>	Gets the privileges configured for an object at a specific AN (animation-point number).
<u>DspAnInfo</u>	Gets information on the state of the animation at an AN.
<u>DspAnInRgn</u>	Checks if an AN is within a specified region.

DspAnMove	Moves an AN.
DspAnMoveRel	Moves an AN relative to its current position.
DspAnNew	Creates an AN.
DspAnNewRel	Creates an AN relative to another AN.
DspAnSetMetadata	Non-blocking function, that sets the value of the specified metadata entry.
DspAnSetMetadataAt	Sets the value of a metadata entry.
DspBar	Displays a bar graph at an AN.
DspBmp	Displays a bitmap at a specified AN.
DspButton	Displays a button at an AN and puts a key into the key command line (when the button is selected).
DspButtonFn	Displays a button at an AN and calls a function when the button is selected.
DspChart	Displays a chart at an AN.
DspCol	DspCol is deprecated in this version.
DspDel	Deletes the objects at an AN.
DspDelayRenderBegin	Delays screen updating until DspDelayRenderEnd() is called.
DspDelayRenderEnd	Ends the screen update delay set by DspDelayRenderBegin().
DspDirty	Forces an update to an AN.
DspError	Displays an error message at the prompt AN.
DspFile	Defines the screen attributes for displaying a text file.
DspFileGetInfo	Gets the attributes of a file to screen display.
DspFileGetName	Gets the name of the file being displayed in the display "window".
DspFileScroll	Scrolls a file (displayed in the display "window") by a number of characters.

DspFileSetName	Sets the name of the file to display in the display "window".
DspFont	Creates a font.
DspFontHnd	Gets a font handle.
DspFullScreen	Enables or disables the fullscreen mode of the active window.
DspGetAnBottom	Gets the bottom extent of the object at the specified AN.
DspGetAnCur	Gets the current AN.
DspGetAnExtent	Gets the extent of the object at a specified AN.
DspGetAnFirst	Returns the first AN on the current page.
DspGetAnFromPoint	Gets the AN of the object at a specified set of screen coordinates.
DspGetAnHeight	Gets the height of the object at a specified AN.
DspGetAnLeft	Gets the left extent of the object at the specified AN.
DspGetAnNext	Returns the AN following a specified AN.
DspGetAnRight	Gets the right extent of the object at the specified AN.
DspGetAnTop	Gets the top extent of the object at the specified AN.
DspGetAnWidth	Gets the width of the object at a specified AN.
DspGetEnv	Gets a page environment variable.
DspGetMouse	Gets the mouse position.
DspGetMouseOver	Determines if the mouse is within the boundaries of a given AN.
DspGetNearestAn	Gets the nearest AN.
DspGetParentAn	Gets the parent animation number (if any), for the specified AN.
DspGetSlider	Gets the current position (value) of a slider at an AN.

DspGetTip	Gets the tool tip text associated with an AN.
DspGrayButton	Greys and disables a button.
DspInfo	Gets object display information from an AN.
DspInfoDestroy	Deletes an object information block created by DspInfoNew().
DspInfoField	Gets stored and real-time data for a variable tag.
DspInfoNew	Creates an object information block for an AN.
DspInfoValid	Checks if an object information block is still valid.
DspIsButtonGray	Gets the current status of a button.
DspKernel	Displays the Kernel window.
DspMarkerMove	Moves a trend or chart marker to a specified position.
DspMarkerNew	Creates a new trend marker.
DspMCI	Controls a multimedia device.
DspPlaySound	Plays a waveform (sound).
DspPopUpConfigMenu	Displays the contents of a menu node as a pop-up (context) menu, and runs the command associated with the selected menu item.
DspPopupMenu	Creates a menu consisting of a number of menu items.
DspRichText	Creates a Rich Text object at the animation point.
DspRichTextEdit	Enables/disables editing of the contents of a rich text object.
DspRichTextEnable	Enables/disables a rich text object.
DspRichTextGetInfo	Returns size information about a rich text object.
DspRichTextLoad	Loads a copy of a rich text file into a rich text object.
DspRichTextPgScroll	Scrolls the contents of a rich text object by one page length.

<u>DspRichTextPrint</u>	Prints the contents of a rich text object.
<u>DspRichTextSave</u>	Saves the contents of a rich text object to a file.
<u>DspRichTextScroll</u>	Scrolls the contents of a rich text object by a user defined amount.
<u>DspRubEnd</u>	Ends a rubber band selection.
<u>DspRubMove</u>	Moves a rubber band selection to the new position.
<u>DspRubSetClip</u>	Sets the clipping region for the rubber band display.
<u>DspRubStart</u>	Starts a rubber band selection (used to rescale a trend with the mouse).
<u>DspSetSlider</u>	Sets the current position of a slider at the specified AN.
<u>DspSetTip</u>	Sets tool tip text associated with an AN.
<u>DspSetTooltipFont</u>	Sets the font for tool tip text.
<u>DspStatus</u>	Sets the communication status error for a specified animation number.
<u>DspStr</u>	Displays a string at an AN.
<u>DspSym</u>	Displays a symbol at an AN.
<u>DspSymAnm</u>	Displays a series of animated symbols at an AN.
<u>DspSymAnmEx</u>	Displays a series of animated symbols at an AN.
<u>DspSymAtSize</u>	Displays a symbol at a scale and offset from an AN.
<u>DspText</u>	Displays text at an AN.
<u>DspTipMode</u>	Switches the display of tool tips on or off.
<u>DspTrend</u>	Displays a trend at an AN.
<u>DspTrendInfo</u>	Gets information on a trend definition.

See Also[Functions Reference](#)

DspAnCreateControlObject

Creates a new instance of an ActiveX object. If the object already exists for the given Animation Point Number, then that object will be used, that is a new object will not be created, the existing object will merely be refreshed.

AN object created using this function remains in existence until the page is closed or the associated Cicode Object is deleted.

Syntax

DspAnCreateControlObject(AN, sClass, Width, Height [, sEventClass])

AN:

The animation-point number.

sClass:

The class of the object. You can use the object's human readable name, its program ID, or its GUID. If the class does not exist, the function will return an error.

For example:

- "Calendar Control 8.0" - human readable name
- "MSCAL.Calendar.7" - Program ID
- "{8E27C92B-1264-101C-8A2F-040224009C02}" - GUID

Width:

The width of the ActiveX object.

Height:

The height of the ActiveX object.

sEventClass:

The string you would like to use as the event class for the object.

Return Value

The newly created object, if successful, otherwise an [error](#) is generated.

Related Functions

[CreateObject](#), [CreateControlObject](#)

Example

See [CreateControlObject](#)

See Also[Display Functions](#)**DspAnFree**

Note: This function is only used for V3.xx and V4.xx animations, and will be superseded in future releases.

Frees (removes) an AN from the current page. If an animation exists at the animation number, it is deleted before the AN is freed. Use this function to free existing ANs or ANs created with the DspAnNew() function. Please be aware that the ANs are only freed in memory - the change is not persistent. The next time the page is opened it will display the AN.

Syntax**DspAnFree(AN)***AN:*

The animation-point number.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[DspAnNew](#)**Example**

```
/* Remove AN20 from the current page. */
DspAnFree(20);
```

See Also[Display Functions](#)**DspAnGetArea**

Gets the area configured for an object at a specific AN (animation-point number). The area is returned as an integer.

Note: This function does not return the areas of keyboard commands associated with

the object.

Syntax

DspAnGetArea(AN)

AN:

The animation-point number.

Return Value

The area if successful, otherwise an [error](#) is returned. If the object is configured with 'Same area as page' checked, the area of the page will be returned. AN area of 0 (zero) means no areas are configured for the object.

Related Functions

[DspAnGetPrivilege](#)

Example

```
/* Get the area configured for the object at AN60. /
DspAnGetArea(60);
```

See Also

[Display Functions](#)

DspAnGetMetadata

Retrieves the field value of the specified metadata entry.

Syntax

DspAnGetMetadata(nAn, sMetaName)

nAn:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. When -2 is specified, it is equivalent to using DspGetAnCur(). (See [DspGetAnCur](#) for usage and limitations.)

sMetaName:

The name of the metadata entry for which to search.

Note: Before calling this function, it may be worthwhile to call ErrSet(1) to disable error checking as this function will generate a hardware error for any object that does not have a metadata entry 'sMetaName', and the cicode task will stop executing.

Return Value

Value for the specified metadata. Returns empty string if a matching metadata entry is not defined and error code CT_ERROR_OBJECT_NOT_FOUND is set.

Related Functions

[DspAnGetMetadataAt](#), [DspAnSetMetadata](#), [DspAnSetMetadataAt](#)

See Also

[Display Functions](#)

DspAnGetMetadataAt

Retrieves metadata information at the specified index.

Syntax

DspAnGetMetadataAt(*nAN*,*nIndex*,*sField*)

nAn:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. When -2 is specified, it is equivalent to using DspGetAnCur(). (See [DspGetAnCur](#) for usage and limitations.)

nIndex:

The index of the metadata in the animation point. The index is 0-based; i.e. the first metadata entry has an index of 0, the next 1, and so on.

sField:

The name of the field from which to retrieve the information for the metadata. Supported fields are:

- Name
- Value

Return Value

The field value string. If there is an error, an empty string is returned. The error code can be obtained by calling the [IsError](#) Cicode function.

Related Functions

[DspAnGetMetadata](#), [DspAnSetMetadata](#), [DspAnSetMetadataAt](#)

See Also

[Display Functions](#)

DspAnGetPos

Gets the x and y coordinates of an AN, in pixels, relative to the top-left corner of the window.

Syntax

DspAnGetPos(AN, X, Y)

AN:

The animation-point number.

X, Y:

The variables used to store the x and y pixel coordinates of the AN, returned from this function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned. The *X* and *Y* variables are set to the AN's position if successful, or to -1 if an error has been detected.

Related Functions

[DspAnMove](#), [DspAnInRgn](#), [DspGetAnCur](#), [DspGetMouse](#), [DspGetNearestAn](#)

Example

```
/* Get the position of AN20 into X and Y. /
DspAnGetPos(20,X,Y);
```

See Also

[Display Functions](#)

DspAnGetPrivilege

Gets the privileges configured for an object at a specific AN (animation-point number). The privilege is returned as an integer.

Note: This function does not return the privileges of keyboard commands associated

with the object.

Syntax

DspAnGetPrivilege(AN)

AN:

The animation-point number.

Return Value

The privilege if successful, otherwise an [error](#) is returned. A privilege of 0 (zero) means no privileges are configured for the object.

Related Functions

[DspAnGetArea](#)

Example

```
/* Get the privileges of the object at AN45. /
DspAnGetPrivilege(45);
```

See Also

[Display Functions](#)

DspAnInfo

Note: This function is only used for V3.xx and V4.xx animations, and has been superseded by future releases.

Gets information on an AN - the type or state of the animation that is currently displayed.

Syntax

DspAnInfo(AN, Type)

AN:

The animation-point number.

Type:

The type of information:

0 - The type of animation currently displayed at the AN. The following is returned:

0 - No animation is displayed.

1 - Color is displayed.

2 - A bar graph is displayed.

3 - Text is displayed.

4 - A symbol is displayed.

5 - AN animation symbol is displayed.

6 - A trend is displayed.

7 - A button is displayed.

8 - A slider is displayed.

9 - A plot is displayed.

1 - The state of the animation currently displayed. If color is displayed, the color is returned. If a bar graph, trend, or symbol is displayed, the bar, trend, or symbol name is returned. If text is displayed, the font handle is returned.

2 - The value of the text or the name of a button at the given AN point is returned.

Return Value

The animation information, which depends on the type passed argument, as described above, as a string.

Related Functions

[DspGetAnCur](#)

Example

```
IF DspAnInfo(25,0) = "1" THEN
    /* If color on AN 25, then get the color */
    col = DspAnInfo(25,1);
END
```

See Also

[Display Functions](#)

DspAnInRgn

Checks if an AN is within a region bounded by two ANs.

Syntax

pAnInRgn(*AN, One, Two*)

AN:

The animation-point number.

One, Two:

One - the AN at a corner of the region; two - the AN at the opposite corner of the region.

Return Value

1 if the AN is within the region, or 0 (zero) if it is not.

Example

```
DspGetMouse (X, Y) ;
DspAnMove (250, X, Y) ;
IF DspAnInRgn (250, 20, 30) THEN
    Prompt ("Mouse in region bounded by AN20 and AN30") ;
ELSE
    Prompt ("Mouse not in region") ;
END
```

See Also

[Display Functions](#)

DspAnMove

Note: This function is only used for V3.xx and V4.xx animations, and was superseded by future releases.

Moves an AN to a new position. Any animation at this AN is also moved.

Syntax

DspAnMove(AN, X, Y)

AN:

The animation-point number.

X:

The x pixel coordinates of the new position.

Y:

The y pixel coordinates of the new position.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspAnMoveRel](#)

Example

```
DspAnMove (25,100,200);  
! Moves AN25 to pixel location 100,200.
```

See Also

[Display Functions](#)

DspAnMoveRel

Note: This function is only used for V3.xx and V4.xx animations, and was superseded by future releases.

Moves an AN relative to its current position. Any animation at this AN is also moved.

Syntax

DspAnMoveRel(AN, X, Y)

AN:

The animation-point number.

X:

The number of pixels to move the AN in the x plane.

Y:

The number of pixels to move the AN in the y plane.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspAnMove](#)

Example

```
DspAnMoveRel(25,10,20);
/* Moves AN25 by 10 pixels to the right and 20 pixels downward,
relative to its current position. */
```

See Also[Display Functions](#)**DspAnNew**

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Creates an AN at the specified x and y coordinates.

Syntax**DspAnNew(X, Y)**

X:

The x pixel coordinate where the new AN is created.

Y:

The y pixel coordinate where the new AN is created.

Return Value

If successful, the new AN is returned. If the AN cannot be created, -1 is returned. If an AN already exists at this location, that AN is returned.

Related Functions[DspAnNewRel](#), [DspAnFree](#)**Example**

```
AN=DspAnNew(100,200);
DspSym(AN,20);
/* Displays symbol 20 at pixel location 100,200 */
```

See Also[Display Functions](#)

DspAnNewRel

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Creates an AN at a distance of x,y pixels from a specified AN.

Syntax

DspAnNewRel(AN, X, Y)

AN:

The AN used as a reference for the new AN.

X:

The distance in the x plane (in pixels) from the reference AN to the new AN.

Y:

The distance in the y plane (in pixels) from the reference AN to the new AN.

Return Value

If successful, the new AN is returned. If the AN cannot be created, -1 is returned. If an AN already exists at this location, that AN is returned.

Related Functions

[DspAnNew](#), [DspGetAnCur](#)

Example

```
AN=DspAnNewRel(20,100,200);
/* Creates an AN at 100x and 200y pixels from AN20 */
```

See Also

[Display Functions](#)

DspAnSetMetadata

Non-blocking function, that sets the value of the specified metadata entry.

Note: Metadata items can only be set using Cicode if the name is configured in the object properties -metadata tab and saved with the page.

Syntax

DspAnSetMetadata(*nAn*, *sMetaName*, *sValue*)

nAn:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. When -2 is specified, it is equivalent to using DspGetAnCur(). (See [DspGetAnCur](#) for usage and limitations.)

sMetaName:

The name of metadata entry for which to search.

Note: Before calling this function, it may be worthwhile to call ErrSet(1) to disable error checking as this function will generate a hardware error for any object that does not have a metadata entry called 'sMetaName', and the cicode task will stop executing

sValue:

The value for the metadata to be set.

Return Value

0 if successful, error code if unsuccessful

Related Functions

[DspAnSetMetadataAt](#), [DspAnGetMetadata](#), [DspAnGetMetadataAt](#)

See Also

[Display Functions](#)

DspAnSetMetadataAt

Non-blocking function, that sets the value of a metadata entry.

Note: Metadata items can only be set using Cicode if the name is configured in the object properties -metadata tab and saved with the page.

Syntax

DspAnSetMetadataAt(*nAN*, *nIndex*, *sField*, *sFieldValue*)

nAn:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. When -2 is specified, it is equivalent to using DspGetAnCur(). (See [DspGetAnCur](#) for usage and limitations.)

nIndex:

The index of the metadata in the animation point.

sField:

The name of the field in which to set the information for the metadata. Supported fields are:

- Name
- Value

sFieldValue:

The value to set in the specified field of the metadata entry.

Note: Clusters should be configured either directly by specifying a full tag name such as C1.TagA or indirectly via the function calls (such as WinNewAt(...)) or via the page configuration parameter.

Return Value

0 if successful, error code if unsuccessful

Related Functions

[DspAnSetMetadata](#), [DspAnGetMetadata](#), [DspAnGetMetadataAt](#)

See Also

[Display Functions](#)

DspBar

Displays a bar graph (on a graphics page) at a specified AN. To scale a tag into the correct range, use the EngToGeneric() function.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspBar(AN, Bar, Value)

AN:

The AN where the bar graph will be displayed.

Bar:

The name of the bar graph to display in the format <[LibName.]BarName>. If you do not specify the library name, a bar graph from the Global library displays (if it exists). To display a Version 1.xx bar graph, specify the bar definition (1 to 255). For example, if you specify bar 1, CitectSCADA displays the bar graph Global.Bar001.

Value:

The value to display on the bar graph. The value needs to be from 0 to 32000 to give 0 to full-scale range on the bar.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[EngToGeneric](#)

Example

```
DspBar(25, "Bars.Loops", 320);
/* Displays a value of 320 (that is 10%) on the loops bar (from the
bars library) at AN25. */
DspBar(25, 3, 320);
/* Displays a value of 320 (that is 10%) on bar definition 3
(CitectSCADA Version 1.xx) at AN25. */
DspBar(26, "Loops_Bar", EngToGeneric(Tag1, 0, 100));
/* Displays Tag1 on the loops_bar (from the global library) at
AN26. Tag1 has an engineering scale of 0 to 100. */
```

See Also

[Display Functions](#)

DspBmp

Displays a bitmap at a specified AN. This function allows you to display any bitmap file at run time. (You can get a new bitmap file from operator input or from the plant, and display it dynamically.)

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspBmp(*AN*, *sFile*, *Mode*)

AN:

The animation-point number.

sFile:

The name of the bitmap (.BMP) file. The file needs to be in the user project path. (Does not support 1 bit, 24 bit or OS/2 bitmaps.)

Mode:

The mode of bitmap display:

0 - Erase the existing bitmap and display this bitmap.

1 - Do not erase the existing bitmap, just draw the new bitmap. (This mode provides smoother animation than Mode 0, but the bitmaps needs to be the same size).

2 - Do not erase the existing bitmap, just draw the new bitmap. This mode is similar to mode 1, but it displays the bitmap about 3 times faster. However, the bitmap should not contain any transparent color, or it will display as a random color. Use this mode for fast, smooth animation.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspDel](#)

Example

```
// Display the bitmap "MyImage.bmp" at AN 60
DspBMP(60, "MyImage.bmp", 0)
```

See Also

[Display Functions](#)

DspButton

Displays a button at a specified AN. When the button is selected, the key definition is put into the key command line. The font, width, height, and down and repeat keys of the button are optional. If you do not specify a width and height, the button adjusts to the size of the button *Name*.

Note: This function is only used for V3.xx and V4.xx animations, and was

superseded in later releases.

Syntax

DspButton(*AN*, *UpKey*, *Name* [, *hFont*] [, *Width*] [, *Height*] [, *DownKey*] [, *RepeatKey*] [, *Style*])

AN:

The animation-point number.

UpKey:

The key generated when the command button is selected (when the mouse button is released after being clicked down). This is the default operation for commands activated by a button.

Name:

The name to display on the button.

hFont:

The handle of the font used to display the button name. Use the DspFont() function to create a new font and return the font handle. Use the DspFontHnd() function to return the font handle of an existing font. The Windows button font is used if the font is omitted or is not defined in the database.

Width:

The width of the button in pixels.

Height:

The height of the button in pixels.

DownKey:

The key generated when the mouse button is clicked down (over the command button). Normally this parameter is not used, because most buttons are configured to activate a command when the mouse button is released (returning to the 'up' position).

RepeatKey:

The key generated repetitively, while the mouse button is being held down (over the command button).

Style:

A number indicating the visibility style of the button:

- 0 - NORMAL: The button appears as a standard button.
- 1 - BORDER_3D: The button is drawn with only the 3-D border (transparent face).
- 2 - BORDER: The button is drawn with only a thin line border.

- 3 - TARGET: The button is totally transparent - this constitutes a screen target.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspButtonFn](#), [KeySetSeq](#), [DspFont](#), [DspFontHnd](#)

Example

```
/* Display a self-sizing button at AN20 using the default font.  
The button is named "Help". When selected, the Key Code "KEY_F1"  
is put into the key command line. */  
DspButton(20,KEY_F1,"Help");  
/* Display the same button at AN20, but in an existing font called  
"BigFont". */  
DspButton(20,KEY_F1,"Help",DspFontHnd("BigFont"));
```

See Also

[Display Functions](#)

DspButtonFn

Displays a button at a specified AN. When the button is selected, a user function is called. If the width and height are 0 (zero), then the button adjusts to the size of the button *Name*.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspButtonFn(AN, UpFunction, Name [, hFont] [, Width] [, Height] [, DownFunction] [, RepeatFunction])

AN:

The animation-point number.

UpFunction:

The user function called when the command button is selected (when the mouse button is released after being clicked down). This is the default operation for commands activated by a button. This callback function can have no arguments, so specify the function with no parentheses (). The callback function needs to return INT as its return data type. You cannot specify a CitectSCADA built-in function for this argument.

Name:

The name to display on the button.

hFont:

The handle of the font used to display the button name. Use the DspFont() function to create a new font and return the font handle. Use the DspFontHnd() function to return the font handle of an existing font. The Windows button font is used if the font is omitted or is not defined in the database.

Width:

The width of the button in pixels.

Height:

The height of the button in pixels.

DownFunction:

The user function called when the mouse button is clicked down (over the command button). Normally this parameter is not used, because most buttons are configured to activate when the mouse button is released (returning to the 'up' position). The callback function needs to have no arguments, so specify the function with no parentheses (). The callback function needs to return INT as its return data type. You cannot specify a CitectSCADA built-in function for this argument.

RepeatFunction:

The user function called repetitively, while the mouse button is being held down (over the command button). The callback function needs to have no arguments, so specify the function with no parentheses (). The callback function needs to return INT as its return data type. You cannot specify a CitectSCADA built-in function for this argument.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspButton](#), [DspFont](#), [DspFontHnd](#)

Example

```
DspButtonFn(20,MyFunc,"Help",0,50,10);
! Call this function when the button is selected.
```

```
INT
FUNCTION
MyFunc()
    PageDisplay("Help");
    RETURN 0;
END
```

See Also

[Display Functions](#)

DspChart

Displays a chart at an AN. Charts are trend lines with markers on them. Values are plotted on the chart pens. You need to specify *Value1*, but *Value2* to *Value8* are optional.

If more values (than the configured pens) are specified, the additional values are ignored. If fewer values (than the configured pens) are specified, the pens that have no values are not displayed.

You should use this function only if you want to control the display of charts directly.

Syntax

DspChart(*AN*, *Chart*, *Value1* [, *Value2* ... *Value8*])

AN:

The AN where the chart will be displayed.

Chart:

The chart to display.

Value1:

The value to display on Pen 1 of the chart.

Value2 ... *8*:

The values to display on Pen 2...Pen 8 of the chart. These values are optional.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspDel](#), [DspTrend](#)

Example

```
/* Using chart definition 5 at AN25,      display a value of 10 on
Pen1, 20 on Pen2,      30 on Pen3 and 40 on Pen4 of the chart. */
DspChart(25,5,10,20,30,40);
/* Using chart definition 6 at AN26, display a value of 100 on Pen1
and 500 on Pen2 of the chart. */
DspChart(26,6,100,500);
```

See Also[Display Functions](#)**DspCol**

DspCol is deprecated in this version of CitectSCADA.

Syntax**DspCol**(*AN*, *Color*)*AN*:

The animation-point number.

Color:

The color to display at the AN.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[DspDel](#)**Example**

```
DspCol(25,RED);
/* Displays the color red at AN25. */
```

See Also[Display Functions](#)**DspDel**

Deletes all objects from a specified AN.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspDel(AN)

AN:

The animation-point number.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspDirty](#)

Example

```
DspDel(25);  
! Deletes all animation at AN25.
```

See Also

[Display Functions](#)

DspDelayRenderBegin

Delays screen updating until DspDelayRenderEnd is called. This function should be used with DspDelayRenderEnd() to "sandwich" Cicode that will modify the appearance of a page. The code should be preceded by DspDelayRenderBegin(), and followed by DspDelayRenderEnd(). This will reduce screen update times, because the modifying code is given time to execute before the page is updated with the changes, and the changes are all made in a single re-draw.

Note: If you have not changed the [Page]DelayRenderAll parameter from its default (TRUE), then you do not need to use this function.

You can call this function as many times in a row as you like, as long as each is ended with a call to DspDelayRenderEnd.

Because your display will stop updating while the "sandwiched" code runs, you should try to make that code as efficient as possible. Do not call Sleep() or any other Cicode functions that will take a long time to run.

Do not call WinSelect within the "sandwiched" code. Do not call this function directly from the Kernel.

Syntax

DspDelayRenderBegin()

Related Functions

[DspDelayRenderEnd](#)

Example

```
/* Begin delay so the following code can be executed before the
images are re-drawn. */
DspDelayRenderBegin();
DspBMP(50, "Image1.bmp", 0) ! Display the bitmap "Image1.bmp"
at AN 50
DspBMP(100, "Image2.bmp", 0) ! Display the bitmap "Image2.bmp"
at AN 100
DspBMP(150, "Image3.bmp", 0) ! Display the bitmap "Image3.bmp"
at AN 150
DspBMP(200, "Image4.bmp", 0) ! Display the bitmap "Image4.bmp"
at AN 200
DspBMP(250, "Image5.bmp", 0) ! Display the bitmap "Image5.bmp"
at AN 250
/* End delay so the images can be re-drawn. */
DspDelayRenderEnd();
```

See Also

[Display Functions](#)

DspDelayRenderEnd

Ends the screen update delay set by DspDelayRenderBegin. This function should be used with DspDelayRenderBegin() to "sandwich" Cicode that will modify the appearance of a page. The code should be preceded by DspDelayRenderBegin(), and followed by DspDelayRenderEnd(). This will reduce screen update times, because the modifying code is given time to execute before the page is updated with the changes, and the changes are all made in a single re-draw.

Because your display will stop updating while the "sandwiched" code runs, you should try to make that code as efficient as possible. Do not call Sleep() or any other Cicode functions that will take a long time to run.

Do not call WinSelect within the "sandwiched" code. Do not call this function directly from the Kernel.

Note: If you have not changed the [Page]DelayRenderAll parameter from its default (TRUE), then you do not need to use this function.

Syntax

DspDelayRenderEnd()

Return Value

No value is returned.

Related Functions

[DspDelayRenderBegin](#)

Example

```
/* Begin delay so the following code can be executed before the
images are re-drawn. */
DspDelayRenderBegin();
DspBMP(50, "Image1.bmp", 0) ! Display the bitmap "Image1.bmp"
at AN 50
DspBMP(100, "Image2.bmp", 0) ! Display the bitmap "Image2.bmp"
at AN 100
DspBMP(150, "Image3.bmp", 0) ! Display the bitmap "Image3.bmp"
at AN 150
DspBMP(200, "Image4.bmp", 0) ! Display the bitmap "Image4.bmp"
at AN 200
DspBMP(250, "Image5.bmp", 0) ! Display the bitmap "Image5.bmp"
at AN 250
/* End delay so the images can be re-drawn. */
DspDelayRenderEnd();
```

See Also

[Display Functions](#)

DspDirty

Forces CitectSCADA to update an AN. Normally, CitectSCADA updates the animation on the AN only if the data has changed. This function tells CitectSCADA to update the AN the next time it animates the AN - even if the data has not changed.

Use this function when you have complex animations that overlap. If two or more animations overlap, you should use the DspDel() or DspDirty() function on their ANs, and then display them in the same order (when they need to be updated).

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspDirty(AN)

AN:

The animation-point number.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspDel](#)

Example

```
DspDirty(20);
! Forces an update of AN20.
```

See Also

[Display Functions](#)

DspError

Displays an error message at the prompt AN on the operator's computer. You can disable the error message display (of this function) by setting the Cicode execution mode in the CodeSetMode() function.

Syntax

DspError(String)

String:

The message to be displayed.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[CodeSetMode](#), [Prompt](#)

Example

```
DspError("Error found");
! Displays "Error found" at the prompt AN.
```

See Also

[Display Functions](#)

DspFile

Defines the screen attributes for displaying a text file. This function defines a "window" where the file will be displayed. You should call this function before any file-to-screen function.

you need to define sequential ANs for each line of text in the display. The file is displayed starting at the specified AN, then the next (highest) AN, and so on. You should not use proportionally-spaced fonts, because the columns of text might not be aligned.

You would normally call this function as the entry function for a graphics page. Use the DspFileSetName() function to specify the file to be displayed. This function is a low level animation function - it controls exactly how the file is to display. If you just want to display a file, use the PageFile() function.

Syntax

DspFile(*AN*, *hFont*, *Height*, *Width*)

AN:

The AN where the file display window will be positioned. When this is set to -2, the window will be created in the Citect Kernel. However, the *hFont* argument is ignored.

hFont:

The handle for the font that is used to display the file, returned from the DspFont() or DspFontHnd() function. The font handle identifies the table where all data on the associated font is stored.

Height:

The maximum number of lines to display on one page of the file display window.

Width:

The width of the file display window, in characters.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageFile](#), [DspFileGetInfo](#), [DspFileGetName](#), [DspFileScroll](#), [DspFileSetName](#), [DspFont](#), [DspFontHnd](#)

Example

```
DspFile(20,0,20,80);
/* Defines the attributes of a screen display to start at AN20,
using the default font, with a window size of 20 lines x 80
columns. */
```

See Also

[Display Functions](#)

DspFileGetInfo

Gets the attributes of a file-to-screen display (used for displaying text files).

Syntax

DspFileGetInfo(AN, Type)

AN:

The AN where the file display window will be located. This AN needs to be the same as the AN specified with the DspFile() function.

Type:

The type of data required:

0 - The width of the file display window, in characters.

1 - The maximum number of lines that can display in one page of the file display window.

2 - The file-to-screen row offset number.

3 - The file-to-screen column offset number.

4 - The number of lines in the displayed file.

Return Value

The attributes of the "window" as an integer. If an incorrect AN is specified, an [error](#) is returned.

Related Functions

[DspFile](#), [DspFileGetName](#), [DspFileScroll](#), [DspFileSetName](#)

Example

```
! Display the page number of the file display.  
PageNumber=IntToStr(DspFileGetInfo(20,2)/DspFileGetInfo(20,1)+1);  
DspText(12,0,"Page No "+PageNumber);
```

See Also

[Display Functions](#)

DspFileGetName

Gets the name of the file being displayed in the display "window". You can use this function to display the file name on the screen.

Syntax

DspFileGetName(AN)

AN:

The animation-point number.

Return Value

The name of the file (as a string). If an incorrect AN is specified, an [error](#) is returned.

Related Functions

[DspFile](#), [DspFileGetInfo](#), [DspFileScroll](#), [DspFileSetName](#)

Example

```
DspText(11,0,DspFileGetName(20));  
! Displays the name of the file displayed at AN20.
```

See Also

[Display Functions](#)

DspFileScroll

Scrolls a file (displayed in the display "window") by a number of characters.

Syntax

DspFileScroll(AN, Direction, Characters)

AN:

The animation-point number.

Direction:

The direction in which to scroll:

1 - Left

2 - Right

3 - Up

4 - Down

Characters:

The number of characters to scroll. To page up or page down through the file, scroll by the height of the file-to-screen window (returned by DspFileGetInfo(AN, 1)).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspFile](#), [DspFileGetInfo](#), [DspFileSetName](#), [DspFileGetName](#)

Example

Page Keyboard	
Key Sequence	PgUp
Command	DspFileScroll(20,3,10)
Comment	Scroll up 10 lines

See Also

[Display Functions](#)

DspFileSetName

Sets the name of the file to display in the display "window". You should call the DspFile() function first (as the entry function for a graphics page) to define the attributes of the display. You can then use the DspFileSetName() function (as a keyboard command) to display a user-specified file. When you call this function, the specified file name is read from disk and displayed on the screen.

Syntax

DspFileSetName(AN, sName)

AN:

The animation-point number.

sName:

The name of the file to display.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspFile](#), [DspFileGetInfo](#), [DspFileGetName](#), [DspFileScroll](#)

Example

Pages	
Page Name	FilePage
Entry Command	DspFile(20,0,20,80)
Comment	Defines a file to screen display to commence at AN20
Page Keyboard	
Key Sequence	##### Enter
Command	DspFileSetName(20, Arg1)
Comment	Displays a specified file on the page

```
DspFile(20,0,20,80);
```

```
/* Defines the file-to-screen display to commence at AN20 using
the default font, with a window size of 20 lines x 80 columns. */
DspFileSetName(20, "C:\AUTOEXEC.BAT");
! Displays file C:\AUTOEXEC.BAT.
```

See Also

[Display Functions](#)

DspFont

Creates a font and returns a font handle. If the requested font already exists, its font handle is returned. You can use this font handle in the functions that display text, buttons, and text files.

If the exact font size does not exist, the closest font size is used.

Syntax

DspFont(*FontType*, *PixelSize*, *ForeOnColor*, *BackOnColor* [, *ForeOffColor*] [, *BackOffColor*])

FontType:

The font type, for example, "Helv".

PixelSize:

The font size, as a positive number for pixels, or a negative number for points.

ForeOnColor:

The foreground color used for the text. If implementing flashing color, this is the initial color that will be used. Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

BackOnColor:

The color used for the background of text. If implementing flashing color, this is the initial color that will be used. Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

ForeOffColor:

An optional argument only required if implementing flashing color for the font foreground. It represents the secondary color used. Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColour](#).

BackOffColor:

An optional argument only required if implementing flashing color for the font background. It represents the secondary color used. Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColour](#).

Return Value

The font handle as an integer. If the font cannot be created, -1 is returned. The font handle identifies the table where all data on the associated font is stored.

Related Functions

[DspFontHnd](#), [DspText](#), [DspButton](#), [DspButtonFn](#), [DspFile](#)

Example

```
Font = DspFont("Helv", -12, White, Red);
DspText(20, Font, "Text in Helv Font");
/* Displays "Text in Helv Font" in 12-point Helvetica font in
white on red at AN20. */
Font = DspFont("Helv", 24, White, Red, Black);
DspText(20, Font, "Text in Helv Font");
/* Displays "Text in Helv Font" in 24 pixel Helvetica font in
flashing black and white on red at AN20. */
```

See Also

[Display Functions](#)

DspFontHnd

Gets the font handle of a font that is defined in the Fonts database. You can use this font handle in the functions that display text, buttons, and text files.

Syntax

DspFontHnd(*Name*)

Name:

The font name in the fonts database.

Return Value

The font handle as an integer. If the font cannot be found, -1 is returned. The font handle identifies the table where the data on the associated font is stored.

Related Functions

[DspFont](#), [DspText](#), [DspButton](#), [DspButtonFn](#), [DspFile](#)

Example

Fonts

Font Name	BigFont
Font Type	Helv
Pixel Size	24
Foreground Color	Blue
Background Color	-1
Comment	Defines a font

```

hBigFont=DspFontHnd("BigFont");
DspText(20,hBigFont,"Text in Big Font");
/* Displays "Text in Big Font" in 24-point Helvetica font in blue
on an unchanged background at AN20. */

```

See Also

[Display Functions](#)

DspFullScreen

Disables or enables the fullscreen mode of the currently active window. This function does not resize the window when it is called; it merely sets the mode flag. The next time the window is displayed, its size (on screen) changes to reflect the setting of the flag. This function overrides the [Animator]FullScreen parameter setting.

If [Page]DynamicSizing is turned on, a page in fullscreen state takes up the entire display area (assuming this does not affect its aspect ratio), and it cannot be resized. Also, a fullscreen page will display without a title bar unless **Title Bar** is checked in Page Properties (or was checked when the page was created). Resizing pages can result in lower picture quality. If this is unacceptable, you should re-design the page using the desired resolution.

If [Page]DynamicSizing is turned off, fullscreen will have the same limitations as it had in versions of CitectSCADA prior to V5.10. In other words, for a page to be displayed in fullscreen, the size of the page needs to be the same size as the display (or bigger). If the page is smaller than the display, the title bar will still display even if fullscreen mode is enabled. Check the size of the graphic pages in CtDraw Tools | Page Attributes Dialog to verify that it is the same as the display resolution. For example 640x480 for VGA, 800x600 for SVGA and 1024x768 for XGA.

Syntax

DspFullScreen(*Mode*)

Mode:

Fullscreen mode:

0 - Disable fullscreen mode.

1 - Enable fullscreen mode without title bar

2 – Enable fullscreen mode with title bar.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinMode](#)

Example

```
/*Minimize the Window, Enable fullscreen mode and then maximize  
the window.*/  
WinMode(6);  
DspFullScreen(1);  
WinMode(3);
```

See Also

[Display Functions](#)

DspGetAnBottom

Gets the bottom extent of the object at the specified AN.

Syntax

DspGetAnBottom(*AN*)

AN:

The animation-point number.

Return Value

The y coordinate of the bottom extent of the object at the AN. If no object exists at the AN, -1 is returned.

Related Functions

[DspGetAnBottom](#), [DspGetAnWidth](#), [DspGetAnHeight](#), [DspGetAnLeft](#), [DspGetAnRight](#),
[DspGetAnTop](#), [DspGetAnNext](#), [DspGetAnExtent](#)

Example

```
nBottom = DspGetAnBottom(30);
```

See Also

[Display Functions](#)

DspGetAnCur

Gets the AN of the current graphics object. This function should only be used by expressions or Cicode functions called from the condition fields of a graphics object, excluding input/command fields. If you need to know the AN that triggered the input/command, the [KeyGetCursor](#) function may be used as it returns the AN where the cursor is currently positioned.

You cannot call this function from the Button or Keyboard forms.

Syntax

DspGetAnCur()

Return Value

The AN associated with the current graphics object. If this function is called outside the page animation system or from an input/command field, -1 will be returned.

Example**Numbers**

AN	20
Expression	MyFunc(PV_10)
Comment	Display the value of PV_10 at AN20

```
/* Function displays a number at the current AN and returns the
```

```
value supplied in the call */
INT
FUNCTION
MyFunc(INT value)
    INT AN, hNew;
    AN = DspGetAnCur();
    hNew = DspAnNewRel(AN, 0, 20);
    DspStr(hNew, "Default", VALUE:###.#);
    RETURN value;
END
```

See Also

[Display Functions](#)

DspGetAnExtent

Gets the extent of the object (the enclosing boundary) at the specified AN.

Syntax

DspGetAnExtent(AN, Top, Left, Bottom, Right)

AN:

The AN at which the object is positioned.

Top:

A buffer that contains the top-most extent of the object.

Left:

A buffer that contains the left-most extent of the object.

Bottom:

A buffer that contains the bottom-most extent of the object.

Right:

A buffer that contains the right-most extent of the object.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned. The *Top*, *Left*, *Bottom*, and *Right* arguments contain the extents of the object, in pixels.

Related Functions

[DspGetAnWidth](#), [DspGetAnHeight](#), [DspGetAnLeft](#), [DspGetAnRight](#), [DspGetAnBottom](#),
[DspGetAnTop](#)

Example

```
// Get extents at AN 25.
DspGetAnExtent(25, Top, Left, Bottom, Right);
```

See Also[Display Functions](#)**DspGetAnFirst**

Gets the first AN on the current page, based on the order in which the ANs were stored by Graphics Builder.

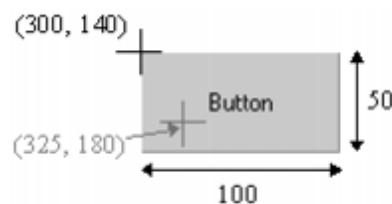
Syntax[DspGetAnFirst\(\)](#)**Return Value**

The value for the first AN, otherwise an [error](#) is returned.

Related Functions[DspGetAnNext](#)**See Also**[Display Functions](#)**DspGetAnFromPoint**

Gets the AN of the object at a specified set of screen coordinates. If the X and Y coordinates given are within the extents of an object, then the AN number of the object will be returned.

For example, if there is a button at coordinates (300, 140), and it is 100 wide, 50 high, this function would return the AN if it uses X between 300 & 400 and Y between 140 and 190, such as DspGetAnFromPoint(325,180).



Hint: If you are using groups and the specified coordinates point to an object that is

part of a group, the AN of the object is returned, not the AN of the group.

Syntax

DspGetAnFromPoint(X, Y [, PrevAN])

X:

The x coordinate of the screen point.

Y:

The y coordinate of the screen point.

PrevAN:

Retrieves the previous AN (in z-order) in situations where a number of objects overlap at the specified point. The default of 0 (zero) specifies no previous AN. A non-zero value should only ever be passed if it is the result of a previous call to DspGetAnFromPoint.

Return Value

The AN or 0 (zero) if no object exists at the point.

Example

```
DspGetMouse(X,Y);
// GetMouse position
AN = DspGetAnFromPoint(X,Y);
// Gets AN if mouse is over the object
Prompt("AN of object =" +AN:###);
!Displays the object's AN at the prompt line
```

If several objects overlap each other at the specified point, the PrevAN argument can be used to produce a list of the associated ANs. This is achieved by using PrevAN to pass the previous result into another call of the function until a zero return is given.

```
INT nAn;
nAn = DspGetAnFromPoint(100,100)
WHILE nAn <> 0 DO
    //Do Something
    nAn = DspGetAnFromPoint(100,100,nAn);
END
```

See Also

[Display Functions](#)

DspGetAnHeight

Gets the height of the object at a specified AN.

Syntax

DspGetAnHeight(AN)

AN:

The animation-point number.

Return Value

The height of the object (in pixels). If no object exists at the AN, -1 is returned.

Related Functions

[DspGetAnWidth](#), [DspGetAnLeft](#), [DspGetAnRight](#), [DspGetAnBottom](#), [DspGetAnTop](#)

Example

```
nHeight = DspGetAnHeight (30);
```

See Also

[Display Functions](#)

DspGetAnLeft

Gets the left extent of the object at the specified AN.

Syntax

DspGetAnLeft(AN)

AN:

The animation-point number.

Return Value

The x coordinate of the left extent of the object at the AN. If no object exists at the AN, -1 is returned.

Related Functions

[DspGetAnWidth](#), [DspGetAnHeight](#), [DspGetAnRight](#), [DspGetAnBottom](#), [DspGetAnTop](#),
[DspGetAnExtent](#)

Example

```
nLeft = DspGetAnLeft(30);
```

See Also

[Display Functions](#)

DspGetAnNext

Returns the AN that follows the specified AN, based on the order in which the ANs were stored on a page by Graphics Builder.

Syntax

DspGetAnNext(AN)

AN:

The animation-point number.

Return Value

The value for the next AN. If -1 is returned, it means the specified AN is invalid or it is the last AN on the page.

Related Functions

[DspGetAnFirst](#)

See Also

[Display Functions](#)

DspGetAnRight

Gets the right extent of the object at the specified AN.

Syntax

DspGetAnRight(AN)

AN:

The animation-point number.

Return Value

The x coordinate of the right extent of the object at the AN. If no object exists at the AN, -1 is returned.

Related Functions

[DspGetAnWidth](#), [DspGetAnHeight](#), [DspGetAnLeft](#), [DspGetAnBottom](#), [DspGetAnTop](#),
[DspGetAnExtent](#)

Example

```
nRight = DspGetAnRight(30);
```

See Also

[Display Functions](#)

DspGetAnTop

Gets the top extent of the object at the specified AN.

Syntax

DspGetAnTop(AN)

AN:

The animation-point number.

Return Value

The y coordinate of the top extent of the object at the AN. If no object exists at the AN, -1 is returned.

Related Functions

[DspGetAnWidth](#), [DspGetAnHeight](#), [DspGetAnLeft](#), [DspGetAnRight](#), [DspGetAnBottom](#),
[DspGetAnExtent](#)

Example

```
nTop = DspGetAnTop(30);
```

See Also

[Display Functions](#)

DspGetAnWidth

Gets the width of the object at a specified AN.

Syntax

DspGetAnWidth(AN)

AN:

The animation-point number.

Return Value

The width of the object (in pixels). If no object exists at the AN, -1 is returned.

Related Functions

[DspGetAnHeight](#), [DspGetAnLeft](#), [DspGetAnRight](#), [DspGetAnBottom](#), [DspGetAnTop](#),
[DspGetAnExtent](#)

Example

```
nWidth = DspGetAnWidth(30);
```

See Also

[Display Functions](#)

DspGetEnv

Gets a page environment variable.

Syntax

DspGetEnv(sName)

sName:

The name of the variable (set using the page environment dialog)

Return Value

The value of the variable (as a string).

Example

```
FUNCTION
PageGroup()
    PageDisplay(DspGetEnv("GroupMenu"));
END
```

See Also[Display Functions](#)**DspGetMouse**

Gets the x and y coordinates of the mouse position, relative to the top left corner of the window.

Syntax**DspGetMouse(X, Y)**

X:

The variables used to store the x pixel coordinate of the mouse position, returned from this function.

Y:

The variables used to store the y pixel coordinate of the mouse position, returned from this function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned. The X and Y variables are set to the mouse position.

Related Functions[KeyGetCursor](#), [DspAnGetPos](#), [DspGetMouseOver](#), [DspGetNearestAn](#)**Example**

```
! If the mouse cursor is at x,y pixel coordinate 43,20;
DspGetMouse(X,Y);
! Sets X to 43 and Y to 20.
```

See Also[Display Functions](#)**DspGetMouseOver**

Determines if the mouse is within the boundaries of a given AN.

Syntax

DspGetMouseOver(AN)

AN

The AN of the animation you wish to check, or -1 for the current AN. Defaults to -1.

Return Value

1 if within the specified AN, 0 if not.

Related Functions

[KeyGetCursor](#), [DspAnGetPos](#), [DspGetMouse](#), [DspGetNearestAn](#)

See Also

[Display Functions](#)

DspGetNearestAn

Gets the AN nearest to a specified x,y pixel location.

If using groups and the nearest object to the specified coordinates is part of a group, the AN of the object is returned, not the AN of the group.

Syntax

DspGetNearestAn(X, Y)

X:

The x coordinate (in pixels).

Y:

The y coordinate (in pixels).

Return Value

The animation point number (AN). A value of -1 is returned if no AN is found.

Related Functions

[DspGetMouse](#), [DspAnGetPos](#), [DspGetAnFromPoint](#)

Example

```
DspGetMouse(X,Y);
! Gets mouse position.
AN=DspGetNearestAn(X,Y);
! Gets AN nearest to the mouse.
Prompt("Mouse At AN"+AN:###);
! Displays AN nearest to the mouse.
```

See Also[Display Functions](#)**DspGetParentAn**

Gets the parent animation number (if any), for the specified animation number. AN animation point will have a parent animation point if it corresponds to an object in a group.

Syntax**DspGetParentAn(AN)***AN:*

The animation-point number.

Return Value

The parent animation point number (AN). If no parent animation exists or an invalid animation number is passed, 0 (zero) is returned.

Related Functions[DspGetAnCur](#)**Example**

```
// Get the parent animation for object 89 (part of a symbol set)
AN = DspGetParentAn(89);
```

See Also[Display Functions](#)**DspGetSlider**

Gets the current position (value) of a slider at an AN. You can call this function in the slider event to find the new position of the slider.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspGetSlider(AN)

AN:

The animation-point number.

Return Value

The value of the slider from 0 to 32000. If no animation exists at the AN, -1 is returned.

Related Functions

[DspSetSlider](#)

Example

```
// Get the position of the slider at AN 30
nPos = DspGetSlider(30);
```

See Also

[Display Functions](#)

DspGetTip

Gets the tool tip text associated with an AN.

Syntax

DspGetTip(AN, Mode)

AN:

The AN from which to get the tool tip text. If no object is configured at the AN, the function will return an empty string.

Mode:

0 - Tool tips from all animation records configured at the AN. Tips are concatenated with a newline character between each string. (This mode is only used for V3.xx and V4.xx animations, and has been subsequently superseded.)

1 - The tool tip from the object configured at the AN.

Return Value

The tool tip text (as a string). If no user tip is available, an empty string is returned.

Related Functions

[DspSetTip](#), [DspTipMode](#)

Example

```
!Display the tool tip text on AN19
DspText(19, 0, DspGetTip(KeyGetCursor(), 1));
```

See Also

[Display Functions](#)

DspGrayButton

Grays and disables a button. If the button is a symbol, the symbol is overwritten with a gray mask. (When a button is grayed, it cannot be selected.) If the **Disabled** field in the Buttons database is blank, the button is enabled unless you use this function. If the **Disabled** field in the Buttons database contains an expression, this function will not override the expression.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspGrayButton(*AN*, *nMode*)

AN:

The AN where the button is located.

nMode:

The mode of the operation:

0 - Ungray the button.

1 - (GRAY_SUNK) Recess the text or symbol (the text or symbol on the button is recessed and shadowed).

2 - (GRAY_PART) This mode is now obsolete - it now has the same effect as GRAY_ALL.

3 - (GRAY_ALL) - Mask the entire button (a gray mask displays over the face of the button).

Return Value

0 (zero) if successful, otherwise, -1 (if no AN is found).

Related Functions

[DspButton](#), [DspButtonFn](#), [DspIsButtonGray](#)

Example

```
! Disable button at AN21
DspGrayButton(21, GRAY_SUNK);
```

See Also

[Display Functions](#)

DspInfo

Extracts individual pieces of object information from an AN. Each AN can have multiple expressions associated with it, and each expression can have multiple variables associated with it. You use an index to refer to each individual expressions or variables. Typically, you would query the number of expressions, then the number of variables in a given expression, then the details of a given variable tag.

Note: Before calling this function you need to first use **DspInfoNew()** to create a handle to the information block from which you want to extract information.

Syntax

DspInfo(*hInfo*, *Type*, *Index*)

hInfo:

The object information block handle, as returned by DspInfoNew(). This handle identifies the table (or block) where all object data is stored.

Type:

The type of data to extract:

0 - Object title (the name of the object type)

1 - Object expression text

2 - Object expression result text

3 - The variable tag name

4 - Not supported.

Note: Getting the raw value using DspInfo is no longer supported. To get the raw value of a tag, use the [TagSubscribe](#) function, specifying a value of "Raw" for the *sScaleMode* parameter. When using TagSubscribe, you can either call [SubscriptionGetAttribute](#) to obtain the value whenever required or register a callback cicode function to run when the value changes. See [TagSubscribe](#) for more details.

5 - The engineering value associated with the variable

6 - The Cicode context. Calling DspInfo with this Type will return a string describing the context in which the Cicode expression is contained. For example, if it appears on the horizontal movement tab it would return "Move X".

7 - The number of Cicode expressions. Calling DspInfo with this Type will return the number of Cicode expressions associated with this animation point.

8 - The number of tags in the expression. Calling DspInfo with this Type will return the number of tags that appear in the given Cicode expression.

9 - Name of the cluster in which the variable tag resides.

10 - Full name of the variable tag in the form *cluster.tagname*.

Index:

An index to the variable within the information block. The required index changes according to the Type as follows:

- For Types 0 to 2, 6 and 8, the index needs to be set to the index of the expression that you wish to query.
- For Types 3 to 5, the index needs to be set to the index of the tag that you wish to query. When one of these types is used, DspInfo will query the tag in the most recently queried expression (otherwise expression 0).
- For Type 7, the index is ignored.

Return Value

The object information (as a string). A blank string is returned if you specify a non-existent expression or variable.

Related Functions

[DspInfoNew](#), [DspInfoField](#), [DspInfoDestroy](#), [TagSubscribe](#), [SubscriptionAddCallback](#), [SubscriptionGetAttribute](#)

Example

```
INT hInfo;
INT iEngineeringValue;
INT iNumberOfExpressions;
INT iNumberOfTags;
INT iExpressionIndex;
INT iTagIndex;
STRING sObjectType;
STRING sExpressionText;
STRING sExpressionResult;
STRING sExpressionContext;
STRING sTagName;
hInfo = DspInfoNew(AN);
IF (hInfo > -1) THEN
    sObjectType = DspInfo(hInfo, 0, 0);
    iNumberOfExpressions = StrToInt(DspInfo(hInfo, 7, 0));
    FOR iExpressionIndex = 0 TO iExpressionIndex < iNumberOfExpressions DO
        sExpressionText = DspInfo(hInfo, 1, iExpressionIndex);
        sExpressionResult = DspInfo(hInfo, 2, iExpressionIndex);
        sExpressionContext = DspInfo(hInfo, 6, iExpressionIndex);
        iNumberOfTags = StrToInt(DspInfo(hInfo, 8, iExpressionIndex));
        FOR iTagIndex = 0 TO iTagIndex < iNumberOfTags DO
            sTagName = DspInfo(hInfo, 3, iTagIndex);
            iEngineeringValue = StrToInt(DspInfo(hInfo, 5, iTagIndex));
        ..
    END
    ..
END
```

See Also

[Display Functions](#)

DspInfoDestroy

Destroys an object information block created by DspInfoNew(). You should destroy an object information block when you no longer need it, to free CitectSCADA resources.

When the page (with which the object is associated) is closed, CitectSCADA automatically destroys the object information block.

Syntax

DspInfoDestroy(*hInfo*)

hInfo:

The object information block handle, as returned by DspInfoNew(). This handle identifies the table (or block) where all object data is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspInfo](#), [DspInfoNew](#), [DspInfoField](#), [DspInfoValid](#)

Example

```
hInfo=DspInfoNew(20);
! Do animation operation
DspInfoDestroy(hInfo);
```

See Also

[Display Functions](#)

DspInfoField

Obtains static and real-time data from a variable tag. You get static data from the Variable Tags database. The additional field "Eng_Value", returns dynamic real-time data for the variable tag. To get this real-time data, you need to first call the DspInfoNew() function to get the information block handle *hInfo*.

Getting the raw value of a variable tag using DspInfoField is no longer supported. To get the raw value of a tag, use the TagSubscribe function, specifying a value of "Raw" for the *sScaleMode* parameter. When using TagSubscribe, you can either call [SubscriptionGetAttribute](#) to obtain the value whenever required or register callback cicode function to run when the value changes. See [TagSubscribe](#) for more details.

Syntax

DspInfoField(*hInfo*, *sTag*, *sField* [, *ClusterName*])

hInfo:

The object information block handle, as returned by DspInfoNew(). This handle identifies the table (or block) where all data on the object is stored. Set this handle to 0 (zero) if you do not require real-time data.

sTag:

The name of the variable tag. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag". This argument does not support arrays. If array syntax is used, the information will be retrieved for only the tag name.

sField:

The name of the field from which to extract the data:

Cluster - Name of the cluster in which the Tag resides

Comment - Variable tag comment

Eng_Full - Engineering Full Scale

Eng_Zero - Engineering Zero Scale

Eng_Units - Engineering Units

Eng_Value - Scaled engineering value - Dynamic

Field - Description

FullName - Full name of the tag in the form *cluster.tagname*.

Name - Variable Tag Name

Type - Data Type

Unit - I/O Device Name

ClusterName:

Specifies the name of the cluster in which the Tag resides. This is optional if you have one cluster or are resolving the tag via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The data (as a string).

Related Functions

[DspInfo](#), [DspInfoNew](#), [DspInfoDestroy](#), [SubscriptionGetAttribute](#), [SubscriptionAddCallback](#), [TagSubscribe](#)

Example

```
! Get the I/O device that Variable Tag "PV123" belongs to.  
IODev=DspInfoField(0,"PV123","Unit");  
! Get the real-time engineering value of a tag.  
hInfo=DspInfoNew(20);  
sTag=DspInfo(hInfo,3,0);  
EngValue=DspInfoField(hInfo,sTag,"Eng_Value");
```

See Also

[Display Functions](#)

DspInfoNew

Creates an object information block. Use this function with the associated low-level animation information functions to get and process object information on an AN.

Note: When you have finished with the object information block, you need to destroy

it with the DspInfoDestroy() function. There are limited number of info 383 blocks that can be allocated, if they are not freed properly DspInfoNew will return -1.

If you need simple animation help, use the InfoForm() or the InfoFormAn() functions.

Syntax

DspInfoNew(AN)

AN:

The AN for which object information is provided.

Return Value

The object information block handle. If no object data is available, then -1 is returned.

Related Functions

[DspInfo](#), [DspInfoField](#), [DspInfoDestroy](#), [InfoForm](#), [InfoFormAn](#)

Example

```
/*This example creates a form, with the title "Tag Info" and a
size of 25 x 5 characters. It creates an information block for the
AN closest to the mouse cursor and then extracts the name, I/O
device, and engineering value for the first tag in the object
expression.*/
INT hInfo;
STRING sTag;
hInfo=DspInfoNew(DspGetNearestAN());
IF hInfo>-1 THEN
    FormNew("Tag Info",25,5,2);
    sTag=DspInfo(hInfo,3,0);
    FormPrompt(0,0,sTag);
    FormPrompt(0,16,DspInfoField(hInfo,sTag,"Unit"));
    FormPrompt(0,32,DspInfoField(hInfo,sTag,"Eng_Value"));
    FormRead(0);
    DspInfoDestroy(hInfo);
END
```

See Also

[Display Functions](#)

[DsplInfoValid](#)

Checks if an object information block handle is valid. An object information block handle becomes invalid after it is destroyed, or if the user closes the page it is associated with. Use this function if background Cicode is using the object information block, and the operator closes the page.

Syntax

DspInfoValid(*hInfo*)

hInfo:

The object information block handle, as returned by DspInfoNew(). This handle identifies the table (or block) where all object data is stored.

Return Value

1 if the information block handle is valid, otherwise 0 (zero).

Related Functions

[DspInfoNew](#), [DspInfoField](#), [DspInfoDestroy](#)

Example

```
IF DspInfoValid(hInfo) THEN
    EngValue=DspInfoField(hInfo,sTag,"Eng_Value");
END
```

See Also

[Display Functions](#)

DspIsButtonGray

Gets the current status of a button.

Note: This function is only used for V3.xx and V4.xx animations, and has been superseded.

Syntax

DspIsButtonGray(*AN*)

AN:

The AN for which object information is provided.

Return Value

The current mode of the button:

- 0 - The button is active (not grayed).
- 1 - (SUNK_GRAY) The button is inactive (the text or symbol on the button is recessed).
- 2 - (PART_GRAY) This mode is now obsolete. The button will be inactive even if part_gray is returned.
- 3 - (ALL_GRAY) The button is inactive (the entire button is masked).

Related Functions

[DspButton](#), [DspButtonFn](#), [DspGrayButton](#)

Example

```
! Check the status of the button at AN21
status = DspIsButtonGray(21);
```

See Also

[Display Functions](#)

DspKernel

Displays the Kernel window. You should restrict the use of this function because once you are in the Kernel, you can execute any Cicode function with no privilege restrictions. You therefore have total control of CitectSCADA (and subsequently your plant and equipment). Please be aware that you can also open the Kernel by setting the Citect [Debug]Menu parameter to 1 and, when your system is running, selecting **Kernel** from the control-menu box.

Note: You should be experienced with CitectSCADA and Cicode before attempting to use the Kernel as these facilities are powerful, and if used incorrectly, can corrupt your system.

Note: You should only use the Kernel for diagnostics and debugging purposes, and not for normal CitectSCADA operation.

You should restrict access to the Kernel, because once you are in the Kernel, you can execute any Cicode function with no privilege restrictions. You therefore have total control of CitectSCADA (and subsequently your plant and equipment).

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

- Do not use the kernel for normal CitectSCADA operation. The kernel is only for diagnostics and debugging purposes.
- Configure your security so that only approved personnel can view or use the kernel.
- Do not view or use the kernel unless you are an expert user of CitectSCADA and Cicode, or are under the direct guidance of Schneider Electric (Australia) Pty. Ltd. Support.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax

DspKernel(*iMode*)

iMode:

The display mode of Kernel:

1 - Display the Kernel. If the Kernel is already displayed and *iMode*=1, the keyboard focus is changed to the Kernel.

0 - Hide the Kernel

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KerCmd](#), [TraceMsg](#)

Example

```
DspKernel(1);  
!Display the Citect Kernel window
```

See Also

[Display Functions](#)

DspMarkerMove

Moves a trend or chart marker to a specified position.

Syntax

DspMarkerMove(*AN*, *hMarker*, *Offset*)

AN:

The AN where the trend or chart is positioned.

hMarker:

The marker handle, as returned from the DspMarkerNew() function. The marker handle identifies the table where all data on the associated marker is stored.

Offset:

The offset by which to move the marker. Vertical markers have an offset from 0 (zero) to the maximum number of samples in the trend. Horizontal markers have a offset of 0 (zero) to 32000.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspMarkerNew](#), [OnEvent](#)

Example

See [DspMarkerNew](#)

See Also

[Display Functions](#)

DspMarkerNew

Creates a new trend marker. A trend marker is used to show cursor values or limits on a trend. You can use up to 10 markers on a single trend or chart.

If you add markers to a trend or chart that CitectSCADA is animating, you need to repaint them using the trend paint event (OnEvent(Window,22)). (Otherwise CitectSCADA will delete any markers displayed when the trend is updated.)

Syntax

DspMarkerNew(AN, Mode, Color)

AN:

The animation-point number.

Mode:

The mode of the marker:

0 - A vertical marker

1 - A horizontal marker

Color:

The color of the marker (flashing color not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB color using the function [MakeCitectColour](#).

Return Value

The marker handle, or -1 if the function is unsuccessful. The marker handle identifies the table where data on the associated marker is stored.

Related Functions

[DspMarkerMove](#), [OnEvent](#)

Example

```
INT offset; ! offset of marker
INT hMarker; ! marker handle
hMarker = DspMarkerNew(40, 0, WHITE);
! create a new marker, vertical WHITE
offset = 100;
DspMarkerMove(40, hMarker, offset);
! Moves marker to offset 100
OnEvent(22, MyTrendPaint);
! set trend paint event, needs to stop event when change pages
! this function is called when CitectSCADA updates the trend
INT
FUNCTION
MyTrendPaint()
    DspMarkerMove(40, hMarker, offset);
    RETURN 0;
END
```

See Also

[Display Functions](#)

DspMCI

Controls a multimedia device. The Media Control Interface (MCI) is a high-level command interface to multimedia devices and resource files. MCI provides applications with device-independent capabilities for controlling audio and visual peripherals (for example, for playing multimedia devices and recording multimedia resource files).

Using this function, you can control multimedia devices by using simple commands like open, play, and close. MCI commands are a generic interface to multimedia devices. You can control any supported multimedia device, including audio playback and recording. For a full overview of MCI, see the Windows *Multimedia Programmer's Guide*.

Syntax

DspMCI(*sCommand*)

sCommand:

The MCI command. See the Microsoft Windows Multimedia Programmer's Guide for details.

Return Value

A string message with the status of the MCI command.

Related Functions

[DspPlaySound](#)

Example

```
DspMCI("open cdaudio")
DspMCI("set cdaudio time format tmsf")
DspMCI("play cdaudio from 6 to 7")
DspMCI("close cdaudio")
/*Plays track 6 of an audio CD*/
DspMCI("open c:\mmdata\purplefi.wav type waveaudio alias finch")
DspMCI("set finch time format samples")
DspMCI("play finch from 1 to 10000")
DspMCI("close finch")
/*Plays the first 10,000 samples of a waveform audio file*/
```

See Also

[Display Functions](#)

DspPlaySound

Plays a waveform (sound). Wave form sound files *.WAV are provided with Windows and by third-party developers, or you can record them yourself to play long (and complex) sound sequences.

This function searches the [Sounds] section of the WIN.INI file for an entry with the specified name, and plays the associated waveform. If the name does not match an entry in the WIN.INI file, a waveform filename is assumed. The function will then search the following directories for the waveform file (directories are listed in search order):

1. The current directory
2. The Windows directory
3. The Windows system directory

4. The directories listed in the PATH environment variable
5. The list of directories mapped in the network.

If the file is not in one of the aforementioned directories, you need to include the full path to the sound file. If the file doesn't exist in one of the above directories or at a location specified with a full path, the sound will not be played.

Syntax

DspPlaySound(*sSoundname*, *nMode*)

sSoundname:

The waveform to be played. Predefined sounds (in the WIN.INI file) are:

- SystemAsterisk
- SystemExclamation
- SystemQuestion
- SystemDefault
- SystemHand
- SystemExit
- SystemStart

nMode:

Not used. Needs to be 0 (zero).

Return Value

TRUE if successful, otherwise FALSE (if an [error](#) is detected).

Related Functions

[Beep](#)

Example

```
DspPlaySound("C:\WINNT\MEDIA\Notify.wav",0);  
DspPlaySound("SystemStart",0);
```

See Also

[Display Functions](#)

DspPopupConfigMenu

Displays the contents of a menu node as a pop-up (context) menu, and run the command associated with the selected menu item. You can specify the contents of a menu using the menu configuration dialog at design time, or using the Menu family of Cicode functions at runtime.

Syntax

DspPopupConfigMenu(*hParent*, [, *bNonRecursive* [, *XPos* [, *YPos*]]])

hParent

The parent node of the menu tree returned from any of the following functions:

- `MenuGetGenericNode()`, `MenuGetPageNode()` or `MenuGetWindowNode()` - used to get the parent node of menu tree for a page.
- `MenuGetFirstChild()`, `MenuGetNextChild()`, `MenuGetPrevChild()`, `MenuGetParent()` - used to traverse to other nodes in a menu tree

bNonRecursive

Whether not to recursively transverse child tree nodes and list them as sub-menus in the pop-up menu. This parameter is optional. If it is left unspecified, its value will be defaulted to 0 (recursive). When it is set to 1, only the immediate child nodes of the specified menu handle will be listed. In this mode, tree nodes will be listed as normal menu items (instead of submenus) in the pop-up menu.

XPos

The x-coordinate (relative to the page) at which the menu will be displayed. This parameter is optional. If it is left unspecified, the menu will display at the cursor's current position.

YPos

The y-coordinate (relative to the page) at which the menu will be displayed. This parameter is optional. If it is left unspecified, the menu will display at the cursor's current position.

Return Value

0 if the selected menu command is run or error code if menu command cannot run.

Related Functions

[Display Functions](#)

DspPopupMenu

Creates a popup menu consisting of a number of menu items. Multiple calls to this function enable you to add new items and create sub menus, building a system of linked, Windows-style menus.

Menu items can be displayed as checked and/or disabled. You can also specify a bitmap to display as a menu icon.

This function is first called to build the menu's items and links, and then called again to display it on the screen. In this final call, you have the option to specify the coordinates at which the menu will display, or let it default to the current cursor position.

Syntax

DspPopupMenu(*iMenuNumber*, *sMenuItems* [, *XPos*] [, *YPos*])

iMenuNumber:

An integer representing the menu you are adding items to. The first menu created is Menu 0. If left unspecified, this parameter defaults to -1, causing the menu to be displayed on the screen.

Multiple function calls with the same *iMenuNumber* allow you to build up entries in a particular menu. For example, the following four function calls with *iMenuNumber* = 1 build up 8 entries in Menu 1:

- DspPopupMenu(1, "Selection A>2, Selection B>3");
- DspPopupMenu(1, "Selection C>2, Selection D");
- DspPopupMenu(1, "Selection E>2, Selection F>3");
- DspPopupMenu(1, "Selection G>2, Selection H");

sMenuItems:

A comma-separated string defining the items in each menu. The default value for this parameter is an empty string, which will get passed to the function in the call to display the menu.

The (!), (~), and (,) symbols control display options for menu items.

For example, !Item1 disables Item1; ~Item2 checks Item2; and ,Item3 inserts a separator above Item3. To insert a link from a menu item to a sub menu, use the (>) symbol. For example, : Item4>1 means Item4 links to menu 1.

To insert a bitmap to the left of a menu item as its icon, use the following notation: [Icon]Item5
Inserts the bitmap Icon.BMP to the left of Item5. [Icon] needs to be placed before the Item name, but after any disable (!) or check (~) symbols you may wish to specify.

Bitmap files used for menu icons need to be saved in the project directory so that they can be found by CitectSCADA.

XPos:

The x-coordinate (relative to the page) at which the menu will be displayed. This parameter is optional. If it is left unspecified, the menu will display at the cursor's current position.

YPos:

The y-coordinate (relative to the page) at which the menu will be displayed. This parameter is optional. If it is left unspecified, the menu will display at the cursor's current position.

Return Value

The selected menu item as an integer. This comprises the menu number (return value div 100), and the position of the item in the menu (return value mod 100). For example, a return value of 201 indicates that the first item in Menu 2 was selected, and a return value of 3 indicates that the third item in Menu 0 was selected.

The return value is limited to a maximum of 65535, that is 655 menus and 35 items on the menu. Above this limit the function returns 0.

Note: Links to sub menus are not counted as menu items. For example, if your menu consists of 10 links and one unlinked item, the function will return only when the unlinked item is selected.

Example1

```

!Example 1 illustrates one menu with three menu items.
FUNCTION BuildPopupMenu()
    INT iSelection;
    DspPopupMenu(0, "Item 1,!Item 2,~Item 3");
    iSelection = DspPopupMenu(-1, "", 150, 300);
    ! The above builds a menu with three items:
    ! 'Item 1' will be shown as normal, 'Item 2' will be shown as disabled,
    ! and 'Item 3' will be shown as checked.
    ! The menu will be displayed at position (150, 300).
END

```

Example 2

```

!Example 2 illustrates the creation of two menus which are linked.
FUNCTION BuildLinkedPopupMenu()
    INT iSelection;
    DspPopupMenu(0, "Item A,Item B>1,Item C");
    DspPopupMenu(1, "Item B1,,[Trend]Item B2,,Item B3");
    iSelection = DspPopupMenu();
    ! The above will build two menus - Menu 0 and Menu 1
    ! Item B on Menu 0 links to Menu 1.
    ! 'Item B2' will be shown with Trend.BMP at its left.
    ! The menu will be displayed at the cursor's position.
    ! If 'Item A' is selected, iSelection will equal 1
    ! If 'Item C' is selected, iSelection will equal 2
    ! If 'Item B1' is selected, iSelection will equal 101
    ! If 'Item B2' is selected, iSelection will equal 102
    ! If 'Item B3' is selected, iSelection will equal 103
END

```

See Also

[Display Functions](#)

DspRichText

Creates a Rich Text object of the given dimensions at the animation point *AN*. This object can then be used to display an RTF file (like an RTF report) called using the DspRichTextLoad function.

Syntax

DspRichText(AN, iHeight, iWidth, iMode)

AN:

The AN at which the rich text object will display when the DspRichText command is run.

iHeight:

The height of the rich text object in pixels. The height is established by measuring down from the animation point.

iWidth:

The width of the rich text object in pixels. The width is established by measuring across to the right from the animation point.

iMode:

The display mode for the rich text object. The mode can be any combination of:

0 - **Disabled** - should be used if the rich text object is to be used for display purposes only.

1 - **Enabled** - allows you to select and copy the contents of the RTF object (for instance an RTF report), but you will not be able to make changes.

2 - **Read/Write** - allows you to edit the contents of the RTF object. Remember, however, that the object needs to be enabled before it can be edited. If it has already been enabled, you can just enter Mode 2 as your argument. If it is not already enabled, you will need to enable it. By combining Mode 1 and Mode 2 in your argument (3), you can enable the object, and make it read/write at the same time.

Because the content of the rich text object is just a copy of the original file, changes will not affect the actual file, until saved using the DspRichTextSave function.

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[DspRichTextLoad](#), [PageRichTextFile](#)

Example

```
//This will produce a rich text object at animation point 57,
which is 200 pixels high, and 200 pixels wide. This object will be
for display purposes only (that is read only)//
DspRichText(57,200,200,0);
```

See Also[Display Functions](#)**DspRichTextEdit**

Enables editing of the contents of the rich text object at *AN* if *nEdit* = TRUE, and disables editing if *nEdit* = FALSE.

Syntax**DspRichTextEdit(*AN*, *bEdit*)***AN*:

The reference AN for the rich text object.

bEdit:

The value of this argument determines whether you will be able to edit the contents of the rich text object at AN. Enter TRUE to enable editing, or enter FALSE to make the contents read-only.

Changes made to the contents of the object will not be saved until the DspRichTextSave function is used.

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions[PageRichTextFile](#), [DspRichTextEnable](#), [DspRichTextSave](#)**Example**

```
// Enables editing of the rich text object at AN 25 - if one
exists. Otherwise an error will be returned to iResult //
iResult = DspRichTextEdit(25,TRUE);
```

See Also[Display Functions](#)

DspRichTextEnable

Enables the rich text object at *AN* if *nEnable* = TRUE, and disables the object if *nEnable* = FALSE. When the object is disabled, its contents cannot be selected or copied etc.

Syntax

DspRichTextEnable(AN, bEnable)

AN:

The reference AN for the rich text object.

bEnable:

The value of this argument determines whether the rich text object at *AN* will be enabled or disabled. Enter TRUE to enable the object (that is you can select and copy the contents of the RTF object, but you can't make changes). Enter FALSE to disable the object (that is make it display only).

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[DspRichTextEdit](#)

Example

```
// This line disables the rich text object at AN 25 - if one
// exists. Otherwise an error will be returned to iResult //
iResult = DspRichTextEnable(25, FALSE);
```

See Also

[Display Functions](#)

DspRichTextGetInfo

Retrieves size information about the rich text object at animation point *AN*.

Syntax

DspRichTextGetInfo(AN, iType)

AN:

The reference AN for the rich text object.

iType:

The following size information (in pixels) can be returned about the specified rich text object:

- 0 - Height
- 1 - Width

Return Value

The requested information as a string (units = pixels).

Related Functions

[PageRichTextFile](#)

Example

```
! Gets the height of the rich text object at AN 25 - if one exists.  
iHeight = DspRichTextGetInfo(25,0);  
! Gets the width of the rich text object at AN 423.  
iWidth = DspRichTextGetInfo(423,1);
```

See Also

[Display Functions](#)

DspRichTextLoad

Loads a copy of the file *Filename* into the rich text object at animation point *AN*. (The rich text object may have been created using either the DspRichTextLoad function or the PageRichTextFile function.)

Syntax

DspRichTextLoad(*AN*, *sFilename*)

AN:

The animation point at which a copy of the rich text file (for example, an RTF report) will display. This *AN* needs to match that of a rich text object (created using either the DspRichText function, or the PageRichTextFile function), or the copy of the file will not be loaded into anything, and will not display.

sFilename:

The name of the file to be copied and loaded into the rich text object at the specified animation point. The filename needs to be entered in quotation marks "".

The maximum file size that can be loaded is 512kb.

If you are loading a copy of an RTF report, the report needs to already have been run and saved to a file. Remember that the filename for the saved report comes from the File Name field in the Devices form. The location of the saved file needs to also be included as part of the filename. For example, if the filename in the Devices form listed [Data]\RepDev.rtf, then you would need to enter "[Data]\repdev.rtf" as your argument. Alternatively, you can manually enter the path, for example, "c:\MyApplication\data\repdev.rtf".

If you are keeping a number of history files for the report, instead of using the extension rtf, you need to change it to reflect the number of the desired history file, for example, 001.

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[DspRichText](#), [PageRichTextFile](#)

Example

```
// This will look in the [Data] path (as specified in the
// Citect.ini file), and load a copy of the file DayRep.rtf into the
// rich text object at animation point 57. //
DspRichTextLoad(57, "[Data]\DayRep.rtf");
// This will look in the [Data] path (as specified in the
// Citect.ini file), and load a copy of the history file DayRep.003
// into the rich text object at animation point 908. //
DspRichTextLoad(908, "[Data]\DayRep.003");
// This will load a copy of the history file
f:\MyApplication\data\DayRep.006, into the rich text object at animation
point 908. //
DspRichTextLoad(908, "f:\MyApplication\data\DayRep.006");
```

See Also

[Display Functions](#)

DspRichTextPgScroll

Scrolls the contents of the rich text object displayed at *AN*, by one page length in the direction given in *direction*.

Syntax

DspRichTextPgScroll(*AN*, *iDirection*)

AN:

The reference AN for the rich text object.

iDirection:

The direction in which you want to scroll each time this function is run. You can choose from the following:

- 1 - Left
- 2 - Right
- 3 - Up
- 4 - Down
- 8 - Scroll to top
- 16 - Scroll to bottom

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[PageRichTextFile](#), [DspRichTextEdit](#), [DspRichTextScroll](#)

Example

```
// This line scrolls the contents of the rich text object at AN 25
// down one page. Otherwise an error will be returned to iResult //
iResult = DspRichTextPgScroll(25,4);
// This line scrolls the contents of the rich text object at AN 423
// right one page. Otherwise an error will be returned to iResult //
iResult = DspRichTextPgScroll(423,2);
```

See Also

[Display Functions](#)

DspRichTextPrint

Prints the contents of the rich text object at animation point *AN*, to the port *PortName*.

Syntax

DspRichTextPrint(*AN*, *sPortName*)

AN:

The reference AN for the rich text object.

sPortName:

The name of the printer port to which the contents of the rich text object will be printed. This name needs to be enclosed within quotation marks "". For example "LPT1", to print to the local printer, or "\\Pserver\canon1" using UNC to print to a network printer.

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[DspRichText](#), [FileRichTextPrint](#)

Example

```
! This lines prints  
DspRichTextPrint(25,"LPT1:");
```

See Also

[Display Functions](#)

DspRichTextSave

Saves the contents of the rich text object at animation point *AN*, to the file *Filename*.

Syntax

DspRichTextSave(*AN*, *sFilename*)

AN:

The reference AN for the rich text object.

sFilename:

The name under which the contents of the rich text object will be saved. This name needs to be enclosed within quotation marks "", and needs to include the destination path. For example "[Data]\saved.rtf".

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[DspRichText](#), [PageRichTextFile](#), [DspRichTextLoad](#), [DspRichTextEdit](#)

Example

```
// These lines show two different ways of saving the contents of  
// the rich text object (at AN 25) to file DayRep.rtf//  
DspRichTextSave(25, "[Data]\DayRep.rtf");
```

```
DspRichTextSave(25, "c:\MyApplication\data\DayRep.rtf");
```

See Also

[Display Functions](#)

DspRichTextScroll

Scrolls the contents of the rich text object displayed at *AN*, in the direction given in *direction*, by the number of lines/units given in *amount*. Remember that the height of a line varies according to the font used, therefore if you need to scroll absolute distances, it might be advisable to use the DspRichTextPgScroll function.

Syntax

DspRichTextScroll(*AN*, *iDirection*, *iAmount*)

AN:

The reference AN for the rich text object.

iDirection:

The direction in which you want to scroll each time this function is run. You can choose from the following:

- 1 - Left
- 2 - Right
- 3 - Up
- 4 - Down
- 8 - Scroll to top
- 16 - Scroll to bottom

iAmount:

The amount by which you would like to scroll each time this function is run. Enter the number of lines (for a vertical direction) or units (for a horizontal direction) by which you would like to scroll.

Return Value

0 if successful, otherwise an [error](#) is returned.

Related Functions

[PageRichTextFile](#), [DspRichTextEdit](#), [DspRichTextPgScroll](#)

Example

```
DspRichTextScroll(25, 4, 8);  
DspRichTextScroll(423, 2, 1);
```

See Also

[Display Functions](#)

DspRubEnd

Ends the rubber band selection, and returns the coordinates of the rubber band selection. The meaning of the *cx* and *cy* values depend on the nMode you specify in the DspRubStart() function.

Syntax

DspRubEnd(*x, y, cx, cy*)

x,y:

The x and y coordinates of the start position.

cx,cy:

The x and y coordinates of the end position.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspRubStart](#), [DspRubMove](#), [DspRubSetClip](#)

Example

See [DspRubStart](#)

See Also

[Display Functions](#)

DspRubMove

Moves the rubber band selection to the new position. You need to first have defined a rubber band selection using the DspRubStart() and DspRubEnd() functions.

This function will erase the existing rubber band and then redraw it in the new position. You would normally move the rubber band by mouse input, but you can get input from the keyboard or any other Cicode to control the rubber band.

Syntax

DspRubMove(*x, y*)

x,y:

The x and y coordinates of the current position.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspRubStart](#), [DspRubEnd](#), [DspRubSetClip](#)

Example

See [DspRubStart](#)

See Also

[Display Functions](#)

DspRubSetClip

Sets the clipping region for the rubber band display. If you enable the clipping region, the rubber band will not move outside of the clip region. This allows you to restrict the rubber band to within some constrained region. (For example, to prevent an operator from dragging the rubber band outside of the trend display when zooming the trend.)
you need to call this function (to enable the clipping region) before you can start the rubber band selection (with the DspRubStart() function).

Syntax

DspRubSetClip(*x1, y1, x2, y2*)

x1,y1,x2,y2:

The x and y coordinates of the clipping region.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspRubStart](#), [DspRubEnd](#), [DspRubMove](#)

Example

```
// Set the clipping region to a rectangle starting at 100, 100 to  
200, 300  
DspRubSetClip(100, 100, 200, 300);  
// Start the rubber band display with clipping mode on  
DspRubStart(x, y, 4);
```

See Also

[Display Functions](#)

DspRubStart

Starts the rubber band selection. Call this function when the left mouse button is pressed - the rubber band is displayed at the starting position. Call the DspRubEnd() function to end the selection, when the mouse button is released. The DspRubMove() function moves the selection to the new position.

This function is used by the trend templates for the trend zoom function. Use the rubber band functions whenever you want the operator to select a region on the screen or display a dynamic rectangle on the screen.

You can only display one rubber band per page. If you display a second rubber band, the first rubber band is erased. To move a rubber band with the mouse, use the OnEvent() function to get notification of the mouse movement, and then the DspRubMove() function. Because these are generic rubber-band display functions, you can get input from the keyboard, Cicode variables, the I/O device, and the mouse.

Syntax

DspRubStart(*x, y, nMode*)

x,y:

The x and y coordinates of the current position.

nMode:

The mode of the rubber banding operation:

0 - cx,cy as absolute pixel positions

1 - cx,cy in pixels relative to x,y

2 - (x,y) the distance from top left to (cx,cy)

4 - enable the rubber band selection using the clipping region defined by DspRubSetClip().

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspRubEnd](#), [DspRubMove](#), [DspRubSetClip](#), [OnEvent](#)

Example

See also the ZOOM.CI file in the Include project for details.

```

INT xRub, yRub, cxRub, cyRub;
/* Call this function on left mouse button down. */
FUNCTION
StartSelection()
    INT x,y;
    DspGetMouse(x,y);           ! Get the current mouse position
    DspRubStart(x,y,0);         ! Start the rubber banding
    OnEvent(0,MouseEvent);     ! Attach mouse move event
END
/* Call this function on left mouse button up. */
FUNCTION
EndSelection()
    ! Stop the rubber banding and get sizes into the ..Rub variables
    DspRubEnd(xRub,yRub,cxRub,cyRub);
    OnEvent(0,0);              ! Stop mouse move event
END
INT
FUNCTION
MouseEvent()
    INT x,y;
    DspGetMouse(x,y);           ! Get mouse position
    DspRubMove(x,y);            ! Move the rubber band
    RETURN 0
END

```

See Also

[Display Functions](#)

DspSetSlider

Sets the current position of a slider at the specified AN. You can use this function to move a slider, and adjust the value of the variable associated with the slider.

Note: This function is only used for V3.xx and V4.xx animations, and was

superseded in later releases.

Syntax

DspSetSlider(AN, nPos)

AN:

The animation-point number.

nPos:

The position of the slider from 0 to 32000 where 0 is the zero position of the slider and 32000 if full position of the slider.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspGetSlider](#)

Example

```
// Set the position of the slider at AN 30 to 1/2 scale  
DspSetSlider(30, 16000);
```

See Also

[Display Functions](#)

DspSetTip

Sets tool tip text associated with an AN. Any existing text associated with the AN will be replaced with the new text.

Syntax

DspSetTip(AN, sText)

AN:

The animation-point number.

sText:

The tool tip text to set for the AN.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspGetTip](#), [DspTipMode](#)

Example

```
!Set a tool tip for AN19
DspSetTip(19, "Start Slurry Pump");
```

See Also

[Display Functions](#)

DspSetTooltipFont

Sets the font for tool tip text.

The parameter [Animator]TooltipFont also specifies the tool tip font. The parameter is checked at startup, and if it is set, the font is set accordingly. You can then use DspSetTooltipFont() to override the parameter until the next time you start CitectSCADA.

Syntax

DspSetTooltipFont(*sName* [, *nPointSize*] [, *sAttribs*])

sName:

The name of the Windows font to be used, enclosed by quotation marks ". A value for this parameter is required, however specifying an empty string "" will set the tooltip font to the default of MS Sans Serif.

nPointSize:

The size of the font in points. If you do not specify a value, the point size defaults to 12.

sAttribs:

A string specifying the format of the font. Use one or all of the following, enclosed by quotation marks " ":

- *B* to specify Bold
- *I* to specify Italics
- *U* to specify Underline

If you don't specify a value for this parameter, it will default to an empty string and no formatting will be applied.

Return Value

No return value.

Related Functions

[DspGetTip](#), [DspTipMode](#)

Example

```
!Set the tool tip font to Bold, Italic, Times New Roman, with a
point size of 12
DspSetTooltipFont("Times New Roman", 12, "BI");
```

See Also

[Display Functions](#)

DspStatus

Determines whether the object at the specified AN will be grayed (hatch pattern) in the event communication attempts are unsuccessful.

Syntax

DspStatus(*AN*, *nMode*)

AN:

The animation-point number.

nMode:

0 - Normal display when communication attempts are unsuccessful

1 - Gray the object (with a hatch pattern) when communication attempts are unsuccessful

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Example

```
DspStatus(67, 1)
// Disable the animation at AN 67
```

See Also

[Display Functions](#)

DspStr

Displays a string at a specified AN.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspStr(AN, sFont, sText [, iLength] [, iAlignMode] [, iLengthMode])

AN:

The AN where the text will be displayed.

sFont:

The name of the font that is used to display the text. The Font Name needs to be defined in the Fonts database. If the font is not found, the default font is used.

sText:

The text to display.

iLength:

Length of the Text to display, either in characters or pixels depending on Mode (default -1, no truncation)

iAlignMode:

The alignment of the text string:

- 0 - Left Justified (default)
- 1 - Right Justified.
- 2 - Center Justified.

iLengthMode:

The length mode of the text string:

- 0 - Length as pixels truncated (default)
- 1 - Length as pixels truncated with ellipsis
- 2 - Length interpreted as characters.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspText](#)

Example

```
DspStr(25,"RedFont","Display this text");
/* Displays "Display this text" using "RedFont" at AN25. "RedFont"
needs to be defined in the Fonts database. */
```

See Also

[Display Functions](#)

DspSym

Note: This function is only used for V3.xx and V4.xx animations. In later releases this function is redundant. The same functionality can be achieved using objects.

Displays a symbol at a specified AN. If the symbol number is 0, any existing symbol (at the AN) is erased.

Syntax

DspSym(*AN, Symbol [, Mode]*)

AN:

The AN where the symbol will be displayed.

Symbol:

The name of the symbol to display in the format <[LibName.]SymName>. If you do not specify the library name, a symbol from the Global library will display (if it exists).

Mode:

Not used. The mode is always set to 1, which means do not erase the existing symbol, just draw the new symbol.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspDel](#)

Example

```
! Display the centrifuge symbol (from the pumps library) at AN25.
```

```
DspSym(25,"Pumps.Centrifuge");
! Display the centrifuge symbol (from the global library) at AN26.
DspSym(26,"Centrifuge");
```

See Also[Display Functions](#)**DspSymAnm**

Animates a series of symbols at an AN. *Sym1* displays first, then *Sym2*, *Sym3* . . . *Sym8* and then *Sym1* displays again, etc. When the next symbol in the sequence is displayed, the current symbol is not erased, but is overwritten to provide a smoother animation. The symbols should all be the same size.

The frequency of changing the symbols is determined by the [Page]AnmDelay parameter. You only need to call this function once to keep the animation going. To stop the animation call the DspDel() function, or call this function again with different symbols (to change the animation).

Note: This function is only used for V3.xx and V4.xx animations. In later releases this function is redundant. The same functionality can be achieved using objects.

Syntax**DspSymAnm(AN, Sym1 [, Sym2 ... Sym8] [, iDisplayMode] [, sSym9])****AN:**

The AN where the animation will occur.

Sym1:

The name of the first symbol to animate in the format <[LibName.]SymName>. If you do not specify the library name, a symbol from the Global library will display (if it exists). Sym1 needs to be specified.

Sym2..Sym8:

The names of the symbols to animate in frames 2 to 8 in the format <[LibName.]SymName>. If you do not specify the library name, a symbol from the Global library will display (if it exists).

iDisplayMode:

Not used. Always set to -1, which means **Soft animation**. The background screen (a rectangular region beneath the symbol) is restored with the original image. Any objects that are within the rectangular region are destroyed when the background is restored. Use this mode when each animation symbol is a different size.

Sym9:

Not all symbols have to be specified. If for example, only two symbols are to display, specify Sym1 and Sym2.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspSym](#)

Example

```
DspSymAnm(25,"Pumps.Centrifuge","Pumps.Floatation");
! Alternately displays the centrifuge symbol and the flotation symbol
(from the pumps library) at AN25.
```

See Also

[Display Functions](#)

DspSymAnmEx

Animates a series of symbols at an AN. *Sym1* displays first, then *Sym2*, *Sym3* . . . *Sym9* and then *Sym1* displays again, etc. When the next symbol in the sequence is displayed, the current symbol is not erased, but is overwritten to provide a smoother animation. The symbols should all be the same size.

The frequency of changing the symbols is determined by the [Page]AnmDelay parameter.

You only need to call this function once to keep the animation going. To stop the animation call this function again with a different *Mode*.

Note: This function is only used for V3.xx and V4.xx animations. In later releases this function is redundant. The same functionality can be achieved using objects.

Syntax

DspSymAnmEx(AN, Mode, Sym1 [, Sym2 ... Sym9])

AN:

The AN where the animation will occur.

Mode:

Not used. Always set to -1, which means **Soft animation**. The background screen (a rectangular

region beneath the symbol) is restored with the original image. Any objects that are within the rectangular region are destroyed when the background is restored. Use this mode when each animation symbol is a different size.

Sym1:

The name of the first symbol to animate in the format <[LibName.]SymName>. If you do not specify the library name, a symbol from the Global library will display (if it exists). Sym1 needs to be specified.

Sym2..Sym9:

The names of the symbols to animate in frames 2 to 9 in the format <[LibName.]SymName>. If you do not specify the library name, a symbol from the Global library will display (if it exists).

Not all symbols have to be specified. If for example, only two symbols are to display, specify Sym1 and Sym2.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspSym](#)

Example

```
DspSymAnmEx(25,-1,"Pumps.Centrifuge","Pumps.Floatation");
! Alternately displays the centrifuge symbol and the flotation symbol
(from the pumps library) at AN25.
```

See Also

[Display Functions](#)

DspSymAtSize

Displays a symbol at the specified scale and offset from the AN position.

By calling this function continuously, you can move symbols around the screen and change their size and shape, to simulate trippers, elevators, and so on. You change the PositionX, PositionY values to change the position of the symbol, the SizeX, SizeY values to change its size, or the symbol itself to change its shape.

You can only use this function at a blank AN, or an AN with a symbol defined without symbols configured. The AN needs to not be attached to any other animation object.

Note: This function is only used for V3.xx and V4.xx animations, and was

superseded in later releases.

Syntax

DspSymAtSize(AN, sSym, PositionX, PositionY, SizeX, SizeY, Mode)

AN:

The AN where the symbol will be animated.

sSym:

The name of the symbol to display, move, or size. If sSym is 0 (zero), any existing symbol at the AN is erased.

PositionX:

The horizontal offset position (from the AN) of the symbol (in pixels).

PositionY:

The vertical offset position (from the AN) of the symbol (in pixels).

SizeX, SizeY:

The horizontal and vertical scaling factors for the symbol (0 - 32000). For example, if PositionX and PositionY are both 32000, the symbol is displayed at its normal size. Please be aware that symbols can only be reduced in size.

Mode:

The mode of the display:

-1 - **Soft animation.** The background screen (a rectangular region beneath the symbol) is restored with the original image. Any objects that are within the rectangular region are destroyed when the background is restored. Use this mode when each animation symbol is a different size.

0 - **Overlap animation.** The background screen (beneath the symbol) is not erased - the next symbol is displayed on top. Transparent color is supported in this mode, allowing for symbol overlap. For this mode to display correctly, each symbol needs to be the same size.

1 - **Animate animation.** The background screen (beneath the symbol) is not erased - the next symbol is displayed on top. This mode provides the fastest animation. For this mode to display correctly, each symbol needs to be the same size. Transparent color is not supported in this mode.

8 - **Stops animation at last symbol displayed.** Use this mode where you want to freeze your animation at the end of the sequence.

16 - Stops animation at current symbol displayed. Use this mode where you want to freeze your animation instantly.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspAnMove](#), [DspAnMoveRel](#), [DspSym](#)

Example

```
! Display tripper moving in x axis at normal size.
DspSymAtSize(21, "lib.tripper", x, 0, 32000, 32000, 0);
! Display elevator going up and down.
DspSymAtSize(22, "lib.elevator", 0, y, 32000, 32000, 0);
! Display can getting bigger and smaller.
DspSymAtSize(23, "lib.can", 0, 0, size, size, 0);
```

See Also

[Display Functions](#)

DspText

Displays text at a specified AN location. This function does the same operation as DspStr(), however it uses a font number rather than a font name.

Note: This function is only used for V3.xx and V4.xx animations, and was superseded in later releases.

Syntax

DspText(*hAN*, *iFont*, *sText* [, *iLength*] [, *iAlignMode*] [, *iLengthMode*])

hAN:

The AN where the text will be displayed.

iFont:

The font number that is used to display the text. (To use the default font, set to -1.)

sText:

The text to display.

iLength:

Length of the Text to display, either in characters or pixels depending on Mode (default -1, no truncation)

iAlignMode:

The alignment of the text string:

- 0 - Left Justified (default)
- 1 - Right Justified.
- 2 - Center Justified.

iLengthMode:

The length mode of the text string:

- 0 - Length as pixels truncated (default)
- 1 - Length as pixels truncated with ellipsis
- 2 - Length interpreted as characters.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspStr](#), [DspFont](#), [DspFontHnd](#)

Example

```
/* Displays "Display this text" at AN25 using the font defined as
BigFont. */
hBigFont=DspFontHnd("BigFont");
DspText(25,hBigFont,"Display this text");
```

See Also

[Display Functions](#)

DspTipMode

Switches the display of tool tips on or off. This function overrides the setting in the [Page]TipHelp parameter.

Syntax

DspTipMode(*nMode*)

nMode:

The display mode:

- 0 - Off

-
- 1 - On
 - 2 - Toggle the tool tip mode
 - 3 - Do not change the mode, just return the current value

Return Value

The old mode.

Related Functions

[DspSetTip](#), [DspGetTip](#)

Example

```
DspTipMode(1); //Switch on tool tips
```

See Also

[Display Functions](#)

DspTrend

Displays a trend at an AN. Values are plotted on the trend pens. You need to specify *Value1*, but *Value2* to *Value8* are optional. If more values (than configured pens) are specified, the additional values are ignored. If fewer values (than configured pens) are specified, the pens that have no values are not displayed.

DspTrend() is optimized so that it will not display the trend until a full set of samples has been collected. For example, if you have defined 100 samples for your trend, the trend will not display until value 100 is entered.

You should use this function only if you want to control the display of trends directly. If you use the standard Trends (defined in the Trends database) this function is called automatically.

Syntax

DspTrend(AN,Trend,Value1 [Value2 ... Value8])

AN:

The AN where the trend will be displayed.

Trend:

The name of the trend to display in the format <[LibName.]TrnName>. If you do not specify the library name, a trend from the Global library will display (if it exists).

To display a Version 1.xx trend, specify the trend number (0 to 255). For example, if you specify trend 1, CitectSCADA displays the trend Global.Trn001.

Value1:

The value to display on Pen 1 of the trend.

Value2...8:

The values to display on Pen 2...Pen 8 of the trend (optional).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspDel](#)

Example

```
/* Using the main_loop trend (from the trends library) at AN25,
display a value of 10 on Pen1, 20 on Pen2, 30 on Pen3 and 40 on
Pen4 of the trend. */
DspTrend(25,"Trends.Main_Loop",10,20,30,40);
/* Using trend definition 5 (CitectSCADA Version 1.xx)
at AN25, display a value of 10 on Pen1, 20 on Pen2, 30 on Pen3 and
40 on Pen4 of the trend. */
DspTrend(25,5,10,20,30,40);
/* Using the loops trend (from the global library) at AN26,
display a value of 100 on Pen1 and 500 on Pen2 of the trend. */
DspTrend(26,"Loops",100,500);
/* Display a trend configured with 100 samples immediately. The
data for the first 100 samples is stored in an array -
MyData[100]. On first display, grab all the data and call
DspTrend() .*/
FOR i = 0 to 100 DO
    DspTrend(AN, "Loops", MyData[i]);
END
// display new samples every 300ms
WHILE TRUE DO
    // Shift MyData down and grab new sample
    TableShift(MyData, 100, 1);
    MyData[99] = MyFastData;
    DspTrend(AN, "Loops", MyData[99]);
    SleepMS(300);
END
/* Display a trend configured with 100 samples immediately. Dummy
data is pushed into the first 100 samples to fill the trend. Once
these values are entered, the trend will be updated each time a
new sample value is entered.*/
// fill up the trend.
FOR i = 0 to 100 DO
```

```

    DspTrend(AN, "Loops", 0);
END
// display new samples every 300ms
WHILE TRUE DO
    DspTrend(AN, "Loops", MyFastData);
    SleepMS(300);
END

```

See Also[Display Functions](#)**DspTrendInfo**

Get information on a trend definition.

Syntax

DspTrendInfo(*hTrend*, *Type*, *AN*)

hTrend:

The name of the trend in the format <[LibName.]TrnName>. If you do not specify the library name, a trend from the Global library is assumed.

To get information on a Version 1.xx trend, specify the trend number (0 to 255). For example, if you specify trend 1, CitectSCADA obtains information from the trend Global.Trn001.

Type:

Type of trend info:

0 - Type of trend:

- 0 = line
- 1 = bar

1 - Number of samples in trend

2 - Height of trend (in pixels)

3 - Width of trend sample (in pixels)

4 - Number of trend pens

11 - Color of pen 1. If the pen uses flashing color, the initial color used. (Use type 19 to determine the secondary flashing color for pen 1.)

12 - Color of pen 2. If the pen uses flashing color, the initial color used. (Use type 20 to determine the secondary flashing color for pen 2.)

13 - Color of pen 3. If the pen uses flashing color, the initial color used. (Use type 21 to determine the secondary flashing color for pen 3.)

14 - Color of pen 4. If the pen uses flashing color, the initial color used. (Use type 22 to determine the secondary flashing color for pen 4.)

- 15 - Color of pen 5. If the pen uses flashing color, the initial color used. (Use type 23 to determine the secondary flashing color for pen 5.)
- 16 - Color of pen 6. If the pen uses flashing color, the initial color used. (Use type 24 to determine the secondary flashing color for pen 6.)
- 17 - Color of pen 7. If the pen uses flashing color, the initial color used. (Use type 25 to determine the secondary flashing color for pen 7.)
- 18 - Color of pen 8. If the pen uses flashing color, the initial color used. (Use type 26 to determine the secondary flashing color for pen 8.)
- 19 - The secondary color used for pen 1, if flashing color is used.
- 20 - The secondary color used for pen 2, if flashing color is used.
- 21 - The secondary color used for pen 3, if flashing color is used.
- 22 - The secondary color used for pen 4, if flashing color is used.
- 23 - The secondary color used for pen 5, if flashing color is used.
- 24 - The secondary color used for pen 6, if flashing color is used.
- 25 - The secondary color used for pen 7, if flashing color is used.
- 26 - The secondary color used for pen 8, if flashing color is used.

AN:

The AN where the trend is displayed.

Return Value

The trend information (as an integer). If Pen Color (*Types* 11 - 18) is requested from a bar trend, the return value is -1.

Related Functions

[DspTrend](#)

Example

```
! get the number of samples for the main_loop trend (from the
trends library).
nSamples = DspTrendInfo("Trends.Main_Loop", 1);
! get the number of samples for trend 3 (CitectSCADA
Version 1.xx).
nSamples = DspTrendInfo(3, 1);
```

See Also

[Display Functions](#)

Chapter: 26 DLL Functions

The DLL (Dynamic Link Library) functions allow you to call any DLL, including the Windows standard functions, any third-party library, or your own library.

With the DLL functions, you can write functions in 'C', Pascal, or any other language that supports DLLs, and then call those functions from CitectSCADA.

DLL Functions

Following are functions relating to DLLs:

DLLCall	Calls a DLL function.
DLLCal-IEx	Calls a DLL function, and passes the specified arguments to that function.
DLLClose	Closes a link to a DLL function.
DLLOpen	Opens a link to a DLL function.

See Also

[Functions Reference](#)

DLLCall

Calls a DLL function, and passes a string of arguments to that function. CitectSCADA converts these arguments (where required) into the type specified in the DLLOpen() call. If an argument cannot be converted, it is set to zero (0) or an empty string "".

You need to first open the DLL with the DLLOpen() function.

Only one call to the DLLCall() function can be made at a time, which means runtime will wait for the called function to return before doing anything else. If the called function takes too long to return, it won't let other tasks execute. Therefore, care needs to be taken so that one call returns before the next is made.

Good programming practice requires that functions which are not expected to complete in a short time are run as separate Windows threads and return a value immediately to CitectSCADA.

Syntax

DLLCall(*hFunction*, *sArgs*)

hFunction:

The DLL function handle, returned from `DLLOpen()`.

sArgs:

The string of arguments to pass to the DLL function. The argument string contains all the arguments for the function, separated by commas (,). Enclose string arguments in quote marks "", and use the string escape character (^) to put a string delimiter within a string. This syntax is the same as the syntax for the `TaskNew()` function

Return Value

The result of the function, as a string.

Related Functions

[DLLOpen](#), [DLLClose](#)

Example

See [DLLOpen](#)

See Also

[DLL Functions](#)

DLLCallEx

Calls a DLL function, and passes the specified arguments to that function.

You need to first open the DLL with the [DLLOpen](#) function.

Only one call to the `DLLCallEx()` function can be made at a time, which means runtime will wait for the called function to return before doing anything else. If the called function takes too long to return, it won't let other tasks execute. Therefore, care needs to be taken so that one call returns before the next is made.

Good programming practice requires that functions which are not expected to complete in a short time are run as separate Windows threads and return a value immediately to CitectSCADA.

Syntax

DLLCallEx(*hFunction*, *vParameters*)

hFunction:

The DLL function handle, returned from DLLOpen().

vParameters:

A variable length parameter list of method arguments. The parameters will be passed to the function in the order that you enter them. Specifying too few or too many parameters will generate an Invalid Argument hardware error. An Invalid Argument hardware error will also be generated if you specify a parameter to the DLL function with the wrong type.

Return Value

The result of the function. If the DLL function returns a string then your Cicode return variable should be of type STRING. All other types will be INT.

Related Functions

[DLLOpen](#), [DLLClose](#)

Example

```
/* This function is called when CitectSCADA starts up,
to initialize all the DLLs that are called */
INT hAnsiUpper;
INT hGlobalAlloc;
FUNCTION InitMyDLLs()
    ! Open DLL to AnsiUpper
    hAnsiUpper = DLLOpen("USER.DLL", "AnsiUpper", "CC");
    hGlobalAlloc = DLLOpen("Kernel", "GlobalAlloc", "IIJ");
END
/* This is the Cicode entry point into the DLL function call. This
function hides the DLL interface from the rest of CitectSCADA. *
STRING
FUNCTION AnsiUpper(STRING sString)
    STRING sResult;
    sResult = DLLCallEx(hAnsiUpper, sString);
    RETURN sResult;
END
/* Allocate memory and return memory handle */
INT
FUNCTION GlobalAlloc(INT Mode, INT Length)
    INT hMem;
    hMem = DLLCallEx(hGlobalAlloc, Mode, Length);
    RETURN hMem;
END
```

See Also

[DLL Functions](#)

DLLClose

Closes the link to a DLL function, and frees the memory allocated for that function link. When the link is closed, you cannot call the function. CitectSCADA automatically closes all function links at shutdown.

Syntax

DLLClose(*hFunction*)

hFunction:

The DLL function handle, returned from DLLOpen().

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DLLOpen](#), [DLLCall](#)

Example

See [DLLOpen](#)

See Also

[DLL Functions](#)

DLLOpen

Opens a link to a DLL function, by loading the specified DLL library into memory and attaching it to the named function. After you open the function link, you can call the function with the DLLCall() function. You pass the function number returned from the DLLOpen() function as an argument in the DLLCall() function.

One accepted method for interfacing with a DLL function is to write a Cicode function file. This file contains the DLLOpen() function to initialize the functions, and one Cicode function for each DLL function, as an interface. In this way, you can hide the DLL interface in this file. Any other Cicode function will call the Cicode interface, and the call to the DLL remains transparent.

Please be aware that DLLs need to be on the path. The file extension is not required.

Note: You need to specify the arguments to the function correctly. CitectSCADA has no way of checking the number and type of arguments to the function. If you specify the number of arguments incorrectly, your computer may display unexpected behavior. You should test your interface thoroughly before using it on a live system.

WARNING

UNINTENDED EQUIPMENT OPERATION

Ensure that you specify the arguments to the DLLOpen() function correctly according to the following list.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax

DLLOpen(*sLib*, *sName*, *sArgs*)

sLib:

The DLL library name.

sName:

The function name. An underscore (_) is required in the function name for a 'C' function, but not for a Pascal function. When you call a DLL from a Cicode function, *sName* needs to be the same as the name defined in the .DEF file used to link the DLL. The file extension is not required.

sArgs:

The string specifying the function arguments. The first character in the string is the return value of the function.

A - Logical.

B - IEEE 8 byte floating point number.

C - Null terminated string. Maximum string length 255 characters.

D - Byte counted string. First byte contains the length of the string, maximum string length 255 characters.

H - Unsigned 2 byte integer.

I - Signed 2 byte integer.

J - Signed 4 byte integer.

Return Value

The DLL function handle, or -1 if the library or function could not be found or loaded.

Related Functions

[DLLCall](#), [DLLClose](#)

Example

```
/* This function is called when CitectSCADA starts up,
to initialize the DLLs that are called */
INT hAnsiUpper;
INT hGlobalAlloc;
FUNCTION InitMyDLLs()
    ! Open DLL to AnsiUpper
    hAnsiUpper = DLLOpen("USER.DLL", "AnsiUpper", "CC");
    hGlobalAlloc = DLLOpen("Kernel", "GlobalAlloc", "IIJ");
END
/* This is the Cicode entry point into the DLL function call. This
function hides the DLL interface from the rest of CitectSCADA. */
STRING
FUNCTION AnsiUpper(STRING sString)
    STRING sResult;
    sResult = DLLCall(hAnsiUpper, "^^" + sString + "^^");
    RETURN sResult;
END
/* Allocate memory and return memory handle */
INT
FUNCTION GlobalAlloc(INT Mode, INT Length)
    STRING sResult;
    INT hMem;
    sResult = DLLCall(hGlobalAlloc, Mode : ##### + "," + Length : #####);
    hMem = StrToInt(sResult);
    RETURN hMem;
END
```

See Also

[DLL Functions](#)

Equipment Database Functions

Following are functions relating to the equipment database.

EquipBrowseClose	Closes an equipment database browse session.
EquipBrowseFirst	Gets the first equipment database entry in the browse session.
EquipBrowseGetField	Gets the field indicated by the cursor position in the browse session.
EquipBrowseNext	Gets the next equipment database entry in the browse session.
Equip-BrowseNumRecords	Returns the number of records in the current browse session.

EquipBrowseOpen	Opens an equipment database browse session.
EquipBrowsePrev	Gets the previous equipment database entry in the browse session.
EquipCheckUpdate	Checks if the equipment database file has been updated, and provides the facility to reload it.
EquipGetProperty	This function reads a property of an equipment database record from the EQUIP.DBF file.

See Also[Functions Reference](#)

EquipBrowseClose

The EquipBrowseClose function terminates an active data browse session and cleans up all resources associated with the session.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax**EquipBrowseClose(*iSession*)***iSession:*

The handle to a browse session previously returned by a EquipBrowseOpen call.

Return Value

0 (zero) if the equipment database browse session exists, otherwise an [error](#) is returned.

Related Functions

[EquipBrowseFirst](#), [EquipBrowseGetField](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#),
[EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

See Also[Equipment Database Functions](#)

EquipBrowseFirst

The EquipBrowseFirst function places the data browse cursor at the first record.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

EquipBrowseFirst(*iSession*)

iSession:

The handle to a browse session previously returned by a EquipBrowseOpen call.

Return Value

0 (zero) if the equipment database browse session exists, otherwise an [error](#) is returned.

Related Functions

[EquipBrowseClose](#), [EquipBrowseGetField](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#),
[EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

See Also

[Equipment Database Functions](#)

EquipBrowseGetField

The EquipBrowseGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

EquipBrowseGetField(*iSession*, *sFieldName*)

iSession:

The handle to a browse session previously returned by a EquipBrowseOpen call.

sFieldName:

The name of the field that references the value to be returned. Supported fields are:

Name, Cluster, Type, Area, Location, IODevice, Page, Help, Comment, Custom1, Custom2, Custom3, Custom4, Custom5, Custom6, Custom7, Custom8.

See [Browse Function Field Reference](#) for information about fields.

Return Value

The value of the specified field as a string. An empty string may or may not be an indication that an error has been detected. The last error should be checked in this instance to determine if an error has actually occurred.

Related Functions

[EquipBrowseClose](#), [EquipBrowseFirst](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#),
[EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

Example

```
STRING fieldValue = "";
STRING fieldName = "TYPE";
INT errorCode = 0;
...
fieldValue = EquipBrowseGetField(iSession, sFieldName);
IF fieldValue <> "" THEN
    // Successful case
ELSE
    // Function returned an error
END
...
```

See Also

[Equipment Database Functions](#)

EquipBrowseNext

The EquipBrowseNext function moves the data browse cursor forward one record. If you call this function after you have reached the end of the records, error 412 is returned (Databrowse session EOF).

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

EquipBrowseNext(*iSession*)

iSession

The handle to a browse session previously returned by a EquipBrowseOpen call.

Return Value

0 (zero) if the equipment database browse session exists, otherwise an [error](#) is returned.

Related Functions

[EquipBrowseClose](#), [EquipBrowseFirst](#), [EquipBrowseGetField](#), [EquipBrowseNumRecords](#),
[EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

See Also

[Equipment Database Functions](#)

EquipBrowseNumRecords

The EquipBrowseNumRecords function returns the number of records that match the filter criteria.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

EquipBrowseNumRecords(*iSession*)

iSession:

The handle to a browse session previously returned by a EquipBrowseOpen call.

Return Value

The number of records that have matched the filter criteria. A value of 0 denotes that no records have matched. A value of -1 denotes that the browse session is unable to provide a fixed number. This may be the case if the data being browsed changed during the browse session.

Related Functions

[EquipBrowseClose](#), [EquipBrowseFirst](#), [EquipBrowseGetField](#), [EquipBrowseNext](#), [EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

Example

```
INT numRecords = 0;  
...  
numRecords = EquipBrowseNumRecords(iSession);  
IF numRecords <> 0 THEN  
    // Have records  
ELSE  
    // No records  
END  
...
```

See Also

[Equipment Database Functions](#)

EquipBrowseOpen

The EquipBrowseOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

EquipBrowseOpen([sFilter] [, sFields] [, sClusters])

sFilter:

A filter expression specifying the records to return during the browse. An empty string indicates that all records will be returned. Where a fieldname is not specified in the filter, it is assumed to be tagname. For example, the filter "AAA" is equivalent to "name=AAA".

sFields:

Specifies via a comma delimited string the columns to be returned during the browse. An empty string indicates that the server will return all available columns. Supported fields are:

Name, Cluster, Type, Area, Location, IODevice, Page, Help, Comment, Custom1, Custom2, Custom3, Custom4, Custom5, Custom6, Custom7, Custom8.

See [Browse Function Field Reference](#) for information about fields.

sClusters:

An optional parameter that specifies via a comma delimited string the subset of the clusters to browse. An empty string indicates that all connected clusters will be browsed.

Return Value

Returns an integer handle to the browse session. Returns -1 when an error is detected.

Related Functions

[EquipBrowseClose](#), [EquipBrowseFirst](#), [EquipBrowseGetField](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

Example

```
INT iSession;
...
iSession = EquipBrowseOpen ("NAME=ABC*", "NAME, AREA",
"ClusterA, ClusterB");
IF iSession <> -1 THEN
    // Successful case
ELSE
    // Function returned an error
```

```
END
```

```
...
```

See Also

[Equipment Database Functions](#)

EquipBrowsePrev

The EquipBrowsePrev function moves the data browse cursor back one record. If you call this function after you have reached the beginning of the records, error 412 is returned (Databrowse session EOF).

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

EquipBrowsePrev(*iSession*)

iSession:

The handle to a browse session previously returned by a EquipBrowseOpen call.

Return Value

0 (zero) if the equipment database browse session exists, otherwise an [error](#) is returned.

Related Functions

[EquipBrowseClose](#), [EquipBrowseFirst](#), [EquipBrowseGetField](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#), [EquipBrowseOpen](#), [EquipCheckUpdate](#), [EquipGetProperty](#)

See Also

[Equipment Database Functions](#)

EquipCheckUpdate

The runtime environment will automatically check if the equipment database has been updated before each page open and before running an equipment browse open function ([EquipBrowseOpen](#)). It does this by checking the version of the database file. You may also update the database periodically in a background task to reload the database even if the page is not changed.

The EquipCheckUpdate function checks if the equipment database file has been updated, and provides the facility to reload it. The reload can only be performed if there are no open browse sessions.

Syntax

EquipCheckUpdate(*iReload*)

iReload:

Pass 1 (TRUE) if you want to reload the database, or 0 (FALSE) if you do not want to reload it and just want to check its status.

Return Value

0 (FALSE) if the equipment database has not changed, or 1 (TRUE) if it has been changed, otherwise an [error](#) is returned.

Related Functions

[EquipBrowseClose](#), [EquipBrowseGetField](#), [EquipBrowseFirst](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#), [EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipGetProperty](#)

See Also

[Equipment Database Functions](#)

EquipGetProperty

This function reads a property of an equipment database record from the EQUIP.DBF database file.

Syntax

EquipGetProperty(*sName*, *sField*)

sName:

The name of the equipment from which to get information. The name of the equipment can be prefixed by the name of the cluster that is "ClusterName.Equipment".

sField:

The field to read. Supported fields are:

Name, Cluster, Type, Area, Location, IODevice, Page, Help, Comment, Custom1, Custom2, Custom3, Custom4, Custom5, Custom6, Custom7, Custom8.

Name - The name of the equipment (254 characters).

Cluster - The cluster to which the equipment belongs (16 characters).

Type - The equipment-specific type of device (254 characters).

Area - Area number (integer) (16 characters).

Location - Equipment specific field (254 characters).

IODevice - I/O Device name(s) (254 characters).

Page - Page name (254 characters).

Help - Help context (254 characters).

Comment - User comment (254 characters).

Custom1..8 - User definable fields (254 characters each).

Return Value

String representation of the property of the equipment database entry. On error, an empty string and an [error](#) is set.

Related Functions

[EquipBrowseClose](#), [EquipBrowseFirst](#), [EquipBrowseGetField](#), [EquipBrowseNext](#), [EquipBrowseNumRecords](#), [EquipBrowseOpen](#), [EquipBrowsePrev](#), [EquipCheckUpdate](#)

See Also

[Equipment Database Functions](#)

Chapter: 27 Error Functions

The error functions trap and process errors. You can use these functions to check the status of any other function.

Error Functions

Following are functions relating to errors:

ErrCom	Gets the communication status for the current Cicode task.
ErrDrv	Gets a protocol-specific error message.
ErrGetHw	Gets a hardware error code.
ErrHelp	Displays help information about a hardware error.
ErrInfo	Gets error information.
ErrLog	Logs a system error.
ErrMsg	Gets the error message associated with a hardware error.
ErrSet	Sets the error mode.
ErrSetHw	Sets a hardware error.
ErrSetLevel	Sets the error level.
ErrTrap	Generates an error trap.
IsError	Checks for an error.

See Also

[Functions Reference](#)

ErrCom

Gets the communication status for the current Cicode task. You can call this function in reports, Cicode that is associated with an object, and in any Cicode task.

Syntax

ErrCom()

Return Value

0 (zero) if all I/O device data associated with the task is valid, otherwise an [error](#) is returned.

Related Functions

[CodeSetMode](#)

Example

```
IF ErrCom()<>0 THEN
    Prompt("I/O device data is bad");
END
```

In a report format:

```
{CICODE}
IF ErrCom()<>0 THEN
    PrintLn("This Report contains bad data");
END
{END}
```

See Also

[Error Functions](#)

ErrDrv

Gets a protocol-specific error message and native error code.

Syntax

ErrDrv(*sProtocol*, *sField*, *nError*)

sProtocol:

The CitectSCADA protocol.

sField:

The field in the PROTERR.DBF database:

- PROTOCOL
- MASK

- ERROR
- MESSAGE
- REFERENCE
- ACTION
- COMMENT

nError:

The protocol specific error code. This field needs to be a variable as it also the place where the returned error code is stored.

Since the first 34 specific error codes are standard for all protocols, CitectSCADA may add 'masking' to make the error code unique. For example, if an I/O device returns errors 1 to 10 (which are already used), the driver may add 0x100000 to its error codes. When this function is called, the mask will be removed before the result is returned to this variable.

Return Value

The error message (as a string), or an empty string ("") if the error is not found. The error code is returned into the *nError* variable.

Related Functions

[ErrInfo](#), [ErrHelp](#)

Example

```
// Get the error message and number associated with error 108
nError = 108;
sError = ErrDrv("TIWAY", "MESSAGE", nError);
```

See Also

[Error Functions](#)

ErrGetHw

Gets the current hardware error status for an I/O device.

I/O devices can be grouped into 2 distinct categories: Those that are created by the system engineer, and those that are created by CitectSCADA itself.

I/O devices that are created by the system engineer include any I/O device listed in the CitectSCADA I/O devices database, and any device visible as a record in the I/O Device form in the Project Editor.

I/O devices that are created by CitectSCADA include Generic, LAN, Cicode, Animation, Reports Server, Alarms Server, Trends Server, and I/O Server, and are those specifically not created by the system engineer.

The argument's values you supply in this function are used by CitectSCADA to determine which type of device hardware alarm you want to work with.

Note: Do not use this function if you have more than 511 I/O devices in your project and the flag [Code]BackwardCompatibleErrHw is set to 1. You may retrieve the hardware error status for the wrong I/O device.

Syntax

ErrGetHw(*Device*, *DeviceType*)

Device:

For I/O devices that are created by the system engineer, select the IODevNo as the argument value.

To determine the **IODevNo** of a physical I/O device in your project, use the I/O device record number from the I/O Device form in the Citect Project Editor. When using an **IODevNo**, the **DeviceType** argument needs to be set to 2.

For I/O devices that are created by CitectSCADA itself, select one of the following options as the argument value:

- Generic
- LAN
- Cicode
- Animation
- Reports Server
- Alarms Server
- Trends Server
- I/O Server

DeviceType:

Select a value from the following options to indicate the 'Type of Device' used in the **Device** argument:

0 - for I/O devices that are created by CitectSCADA itself (Generic, LAN, Cicode, Animation, etc).

2 - for I/O devices that are created by the system engineer.

The **DeviceType** argument was added to this function in V5.40 and later. Earlier versions did not pass a value for the **DeviceType** argument (as it did not exist). Versions prior to V5.40 identified an I/O device by passing the IODevNo (masked with the value of 8192) to the function as the **Device** argument, in the structure:

```
IODevNo + 8192
```

This was for versions of CitectSCADA permitting a maximum limit of 4095 I/O devices.

Versions prior to V5.20 masked the IODevNo with a value of 512. The backward compatibility flag for using this mask needs to be set in the Citect.INI file (see code parameter BackwardCompatibleErrHw.).

Return Value

The detected error.

Related Functions

[ErrHelp](#), [ErrInfo](#), [ErrMsg](#), [ErrSetHw](#)

Example

```
Error=ErrGetHw(3,0);
! Sets Error to the current error status for the animation device.
IF Error=0 THEN
    DspText(4,0,"");
ELSE
    DspText(4,0,"Hardware error");
END
```

See Also

[Error Functions](#)

ErrHelp

Displays information about a hardware error.

Syntax

ErrHelp(*Error*)

Error:

The Cicode hardware error string (as returned by ErrMsg()).

Return Value

0 (zero) if successful, otherwise an [error](#) (274) is returned.

Related Functions

[ErrInfo](#), [IsError](#), [ErrMsg](#)

Example

```
! Invokes the CitectSCADA Help with help on the
hardware alarm.
iResult = ErrHelp(ErrMsg(IsError()));
```

See Also

[Error Functions](#)

ErrInfo

Gets extended error information on the last error that was detected.

Syntax

ErrInfo(*Type*)

Type:

The type of error information. If type is 0 (zero), function returns the animation number where the error occurred.

Return Value

The [error](#) information.

Example

```
! Get the animation number where the last error occurred
AN = ErrInfo(0);
```

See Also

[Error Functions](#)

ErrLog

Logs a message to the CitectSCADA system log file.

This function is useful for logging errors in user functions, and for debugging user functions. The CitectSCADA system log file 'SYSLOG.DAT' is created in the local Windows directory of the computer, C:\Documents and Settings\All Users\Application Data\Citect\CitectSCADA 7.20\Logs.

Syntax

ErrLog(*Message*)

Message:

The message to log. This field can also contain control (such as /n) and formatting characters.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DebugMsg](#), [DebugMsgSet](#), [CodeTrace](#), [TraceMsg](#), [Halt](#)

Example

```
FUNCTION MyFunc(INT Arg)
    IF Arg<0 THEN
        ErrLog("Invalid arg in Myfunc");
        Halt();
    END
END
```

See Also

[Error Functions](#)

ErrMsg

Gets the error message associated with a detected hardware error.

Syntax

ErrMsg(*nError*)

nError:

The hardware error number returned from the `IsError()` function.

Return Value

The error message (as a string). A null value is returned if *nError* is not in the range of Cicode errors.

Related Functions

[IsError](#), [ErrHelp](#), [ErrInfo](#), [ErrTrap](#)

Example

```
//Get the message of the last hardware error  
sMsg = ErrMsg(IsError());
```

See Also

[Error Functions](#)

ErrSet

Sets the error-checking mode. When *Mode* is set to 0 and an error occurs that causes a component to stop executing, CitectSCADA halts the execution of the Cicode task that caused the error, and generates a hardware error.

You can perform error checking by setting *Mode* to 1 and using the IsError() function to trap errors. When the type of error is determined, you can control what happens under particular error conditions.

The operation of the ErrSet() function is unique to each Cicode task. If you enable user error checking for one task, it does not enable error checking for any other tasks.

Note: This has changed from previous versions of CitectSCADA where this feature used to affect all Cicode tasks.

Syntax

ErrSet(*Mode*)

Mode:

Error-checking mode:

0 - default - CitectSCADA will check for errors.

1 - The user needs to check for errors.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[IsError](#), [ErrSetHw](#), [ErrSetLevel](#)

Example

```
ErrSet(1);
Test=Var/0;
Error=IsError();
! Sets Error to 273 (divide by zero).
```

See Also

[Error Functions](#)

ErrSetHw

Sets the hardware error status for a hardware device. Call this function to generate a hardware [error](#).

I/O devices can be grouped into two distinct categories: those created by the system engineer, and those created by CitectSCADA itself.

I/O devices that are created by the system engineer, are any I/O device listed in the CitectSCADA I/O devices database, visible as records in the I/O Device form in the Project Editor.

I/O devices that are created by CitectSCADA, including Generic, LAN, Cicode, Animation, Reports Server, Alarms Server, Trends Server, and I/O Server (are those specifically not created by the system engineer).

The arguments values you supply in this function are used by CitectSCADA to determine the type of device hardware alarm you want to work with.

Note: To use this function, you need to set [Code]BackwardCompatibleErrHw to 1. You cannot use this function if you have more than 511 I/O devices in your project.

Syntax

ErrSetHw(*Device*, *Error*, *DeviceType*)

Device:

For I/O devices that are created by the system engineer, select the IODevNo as the argument value.

To determine the **IODevNo** of a physical I/O device in your project, use the I/O device record number from the I/O Device form in the Citect Project Editor. When using an **IODevNo**, the DeviceType argument needs to be set to 2.

For I/O devices that are created by CitectSCADA itself, select one of the following options as the argument value:

0 - Generic

- 1 - LAN
- 2 - Cicode
- 3 - Animation
- 4 - Reports Server
- 5 - Alarms Server
- 6 - Trends Server
- 7 - I/O Server

Error:

The error code.

DeviceType:

Select a value from the following options to indicate the 'Type of Device' used in the Device argument:

- 0 - For I/O devices that are created by CitectSCADA itself (Generic, LAN, Cicode, Animation, etc).
- 2 - For I/O devices that are created by the system engineer.

The **DeviceType** argument was added to this function in V5.40 and later. Earlier versions did not pass a value for the DeviceType argument (as it did not exist). Versions prior to V5.40 identified an I/O device by passing the IODevNo (masked with the value of 8192) to the function as the **Device** argument, in the structure:

```
IODevNo + 8192
```

This was for versions of CitectSCADA that permitted a maximum limit of 4095 I/O devices.

Versions prior to V5.20 masked the IODevNo with a value of 512. The backward compatibility flag for using this mask needs to be set in the Citect.INI file (see code parameter BackwardCompatibleErrHw).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ErrHelp](#) [ErrMsg](#) [ErrSet](#) [ErrSetHw](#)

Example

```
ErrSetHw(4,273,0);
! Generates a divide by zero error (273) on the report device.
ErrSetHw(3,0,0)
```

! Resets any error on the animation device.

See Also

[Error Functions](#)

ErrSetLevel

Sets the nesting error level to enable CitectSCADA error checking inside a nested function (when CitectSCADA error checking has been disabled). This function returns the old error level and sets a new error level.

The nesting error level is incremented every time the ErrSet(1) function is called.

Syntax

ErrSetLevel(*Level*)

Level:

The nesting error level.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ErrSet](#)

Example

```

! ErrorLevel 0 defaults to ErrSet(0) - enables CitectSCADA error-checking.
FUNCTION MainFn()
    ErrSet(1);
    ! ErrorLevel 1 - disables CitectSCADA error checking.
    Fn1();
    ErrSet(0);
    ! Enables CitectSCADA error checking.
END
FUNCTION Fn1()
    ErrSet(1);
    ! ErrorLevel 2 - disables CitectSCADA error checking.
    Test=Var/0;
    Error=IsError();
    ! Sets Error to 273 (divide by zero).
    Fn2();
    ErrSet(0);
    ! Enables CitectSCADA error checking.
END

```

```
FUNCTION Fn2()
    OldErrorLevel=ErrSetLevel(0);
    ! Sets nesting error level to 0 to enable CitectSCADA error-checking.
    Test=Var/0;
    ! Cicode halts and a hardware alarm is generated.
    ErrSetLevel(OldErrorLevel)
    ! Resets nesting error level to disable CitectSCADA error-checking.
END
```

See Also

[Error Functions](#)

ErrTrap

Generates an error trap. If CitectSCADA error checking is enabled, this function will generate a hardware error and may halt Cicode execution (see *bHalt* argument). If user error checking is enabled, the user function specified in OnEvent(2,Fn) is called.

Syntax

ErrTrap(*Error*, *bHalt*)

Error:

The error number to trap.

bHalt:

Determines whether the Cicode execution will be halted.

0 - Cicode execution is not halted

1 - Cicode execution is halted

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

[ErrSetHw](#), [ErrSet](#), [ErrSetLevel](#), [OnEvent](#)

Example

```
IF Tag=0 THEN
    ErrTrap(273);      ! Traps a divide by zero error.
ELSE
    Value=10/Tag;
END
```

See Also

[Error Functions](#)

IsError

Gets the current error value. The error value is set when any error is detected, and is reset after this function is called. You can call this function if user error-checking is enabled or disabled.

You should call this function as soon as possible after the operation to be checked, because the error code could be changed by the next error.

Syntax

IsError()

Return Value

The current [error](#) value. The current error is reset to 0 after this function is called.

Related Functions

[ErrSet](#)

Example

```
! Enable user error-checking.  
ErrSet(1);  
! Invalid ArcSine.  
Ac=ArcSin(20.0);  
! Sets ErrorVariable to 274 (invalid argument passed).  
ErrorVariable=IsError()
```

See Also

[Error Functions](#)

Chapter: 28 Event Functions

The event functions trap and process asynchronous events.

Event Functions

Following are functions used to trap and process asynchronous events:

CallEvent	Calls the event function for an event type.
ChainEvent	Calls an event function, by function number.
GetEvent	Gets the function number of the current callback event.
OnEvent	Sets an event callback function, by event type.
SetEvent	Sets an event callback function, by function number.

See Also

[Functions Reference](#)

CallEvent

Simulates an event, triggering any OnEvent() function that has the same Type argument specified.

CitectSCADA starts running the function immediately, without reading any data from the I/O devices. Any I/O device variable that you use will contain either 0 (zero) or the last value read.

Syntax

CallEvent(*Window*, *Type*)

Window:

The number of the window, returned from the WinNew(), WinNewAt(), or WinNumber() function.

Type:

The type of event:

0 - The mouse has moved. When the mouse moves the callback function is called. The return value must be 0.

1 - A key has been pressed. When the user presses a key, the callback function is called after CitectSCADA checks for hot keys. If the return value is 0, CitectSCADA checks for key sequences. If the return value is not 0, CitectSCADA assumes that you will process the key and does not check the key sequence. It is up to you to remove the key from the key command line.

If you are using a right mouse button click as an event, you should read about the `ButtonOnlyLeftClick` parameter.

2 - Error event. This event is called if an error is detected in Cicode, so you can write a single error function to check for your errors. If the return value is 0, CitectSCADA continues to process the error and generates a hardware error - it may then halt the Cicode task. If the return value is not 0, CitectSCADA assumes that you will process the error, and continues the Cicode without generating a hardware error.

3 - Page user communication error. A communication error has been detected in the data required for this page. If the return value is 0 (zero), CitectSCADA still animates the page. If the return value is not zero, it does not update the page.

4 - Page user open. A new page is being opened. This event allows you to define a single function that is called when all pages are opened. The return value must be 0.

5 - Page user close. The current page is being closed. This event allows you to define a single function that is called when all pages are closed. The return value must be 0.

6 - Page user always. The page is active. This event allows you to define a single function that is called when all pages are active. The return value must be 0.

7 - Page communication error. A communication error has been detected in the data required for this page. Reserved for use by CitectSCADA.

8 - Page open. This event is called each time a page is opened. Reserved for use by CitectSCADA.

9 - Page close. This event is called each time a page is closed. Reserved for use by CitectSCADA.

10 - Page always. This event is called while a page is active. Reserved for use by CitectSCADA.

11..17 - Undefined.

18 - Report start. The report server is about to start a new report. This event is called on the report server. The return value must be 0.

- 19 - Device history. A device history has just completed. The return value must be 0.
- 20 - Login. A user has just logged in.
- 21 - Logout. A user has just logged out.
- 22 - Trend needs repainting. This event is called each time CitectSCADA reanimates a real-time trend or scrolls an historical trend. You should use this event to add additional animation to a trend, because CitectSCADA deletes all existing animation when a trend is re-drawn. (For example, if you want to display extra markers, you must use this event.)
- 23 - Hardware error has been detected.
- 24 - Keyboard cursor moved. This event is called each time the keyboard command cursor moves. The cursor can be moved by the cursor keys, the mouse, or the Cicode function KeySetCursor(). Note that you can find where the keyboard command cursor is located by calling the function KeyGetCursor().
- 25 - Network shutdown. A Shutdown network command has been issued.
- 26 - Runtime system shutdown and restart. (Required because of configuration changes.)
- 27 - Event. An event has occurred.
- 28 - Accumulator. An accumulator has logged a value.
- 29 - Slider. A slider has been selected.
- 30 - Slider. A slider has moved.
- 31 - Slider. A slider has been released (that is stopped moving).
- While responding to slider events 29, 30, and 31, you can set any variables but you cannot call functions that cause immediate changes to animations on the page (for example, DspText() and DspSym()). Types 29, 30, & 31 relate only to V3.xx and V4.xx animations, and will be superseded in future releases.
- 32 - Shutdown. CitectSCADA is being shutdown.
- 33 - Reserved for CitectSCADA internal use.
- 34 - 41 - CitectSCADA Confirmation Events. Reserved for CitectSCADA internal use. For the confirmation events, two sets of event type code are defined. The runtime calls the CitectSCADA event handler first, and conditionally proceed to the user's event handler depending on the return value of the CitectSCADA event handler.
- 34 - CitectSCADA Event: Child Window Close Confirmation.
- 35 - CitectSCADA Event: Main Window Close Confirmation.
- 36 - CitectSCADA Event: Maximize Window Confirmation.
- 37 - CitectSCADA Event: Minimize Window Confirmation.

- 38 - CitectSCADA Event: Restore Window Confirmation.
- 39 - CitectSCADA Event: Move Window Confirmation.
- 40 - CitectSCADA Event: Size Window Confirmation.
- 41 - CitectSCADA Event: Shutdown Confirmation Confirmation.
- 42 to 49 - User Confirmation Events. These functions are called when a specific event (mainly from Window title bar) occur and before the runtime performs the intended action. This gives a chance for the user to decide what to do with the event. If the return value is 0, the event will be passed on to the default handler so the intended action will be performed. If the return value is not 0, the event will be ignored and no further action will be taken.
- 42 - Child Window Close Confirmation, when the close button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 43 - Main Window Close Confirmation, when close button of the windows' title bar is clicked which will cause the process to shutdown.
- 44 - Maximize Window Confirmation, when the maximize button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 45 - Minimize Window Confirmation, when the minimize button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 46 - Restore Window Confirmation, when the restore button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 47 - Move Window Confirmation, when the window is being dragged or an equivalent Windows' message is received.
- 48 - Size Window Confirmation, when the windows is being resized or an equivalent Windows' message is received.
- 49 - Shutdown Confirmation, when shutdown() function is called.
- 50 - 127 - Reserved for future CitectSCADA use.
- 128 - 256 - User-defined events. These events are for your own use.

Return Value

0 (zero) if successful, otherwise an [error](#) is set. To view the error, use the IsError() function.

Related Functions

[OnEvent](#), [GetEvent](#), [WinNew](#), [WinNewAt](#), [WinNumber](#), [IsError](#)

Example

```
! Call Event Type 1 - key has been pressed in the current window.  
Number=WinNumber();  
CallEvent(Number,1);
```

See Also

[Event Functions](#)

ChainEvent

Calls an event function using the function handle. This creates a chain of event handlers from a single event. Use the GetEvent() function to get the function number of the current event handler.

Syntax

ChainEvent(*hFn*)

hFn:

The function handle, as returned from the GetEvent() function.

Return Value

The return value of the called event function.

Related Functions

[OnEvent](#), [GetEvent](#)

Example

See [GetEvent](#)

See Also

[Event Functions](#)

GetEvent

Gets the function handle of the existing callback event handler. You can use this function handle in the ChainEvent() function to chain call the existing event function, or in the SetEvent() function to restore the event handler.

Syntax

GetEvent(*Type*)

Type:

The type of event:

- 0 - The mouse has moved. When the mouse moves the callback function is called. The return value must be 0.
- 1 - A key has been pressed. When the user presses a key, the callback function is called after CitectSCADA checks for hot keys. If the return value is 0, CitectSCADA checks for key sequences. If the return value is not 0, CitectSCADA assumes that you will process the key and does not check the key sequence. It is up to you to remove the key from the key command line.
If you are using a right mouse button click as an event, you should read about the `ButtonOnlyLeftClick` parameter.
- 2 - Error event. This event is called if an error is detected in Cicode, so you can write a single error function to check for your errors. If the return value is 0, CitectSCADA continues to process the error and generates a hardware error - it may then halt the Cicode task. If the return value is not 0, CitectSCADA assumes that you will process the error, and continues the Cicode without generating a hardware error.
- 3 - Page user communication error. A communication error has been detected in the data required for this page. If the return value is 0 (zero), CitectSCADA still animates the page. If the return value is not zero, it does not update the page.
- 4 - Page user open. A new page is being opened. This event allows you to define a single function that is called when all pages are opened. The return value must be 0.
- 5 - Page user close. The current page is being closed. This event allows you to define a single function that is called when all pages are closed. The return value must be 0.
- 6 - Page user always. The page is active. This event allows you to define a single function that is called when all pages are active. The return value must be 0.
- 7 - Page communication error. A communication error has been detected in the data required for this page. Reserved for use by CitectSCADA.
- 8 - Page open. This event is called each time a page is opened. Reserved for use by CitectSCADA.
- 9 - Page close. This event is called each time a page is closed. Reserved for use by CitectSCADA.
- 10 - Page always. This event is called while a page is active. Reserved for use by CitectSCADA.
- 11..17 - Undefined.

- 18 - Report start. The report server is about to start a new report. This event is called on the report server. The return value must be 0.
- 19 - Device history. A device history has just completed. The return value must be 0.
- 20 - Login. A user has just logged in.
- 21 - Logout. A user has just logged out.
- 22 - Trend needs repainting. This event is called each time CitectSCADA reanimates a real-time trend or scrolls an historical trend. You should use this event to add additional animation to a trend, because CitectSCADA deletes all existing animation when a trend is re-drawn. (For example, if you want to display extra markers, you must use this event.)
- 23 - Hardware error has been detected.
- 24 - Keyboard cursor moved. This event is called each time the keyboard command cursor moves. The cursor can be moved by the cursor keys, the mouse, or the Cicode function KeySetCursor(). Note that you can find where the keyboard command cursor is located by calling the function KeyGetCursor().
- 25 - Network shutdown. A Shutdown network command has been issued.
- 26 - Runtime system shutdown and restart. (Required because of configuration changes.)
- 27 - Event. An event has occurred.
- 28 - Accumulator. An accumulator has logged a value.
- 29 - Slider. A slider has been selected.
- 30 - Slider. A slider has moved.
- 31 - Slider. A slider has been released (that is stopped moving).
- While responding to slider events 29, 30, and 31, you can set any variables but you cannot call functions that cause immediate changes to animations on the page (for example, DspText() and DspSym()). Types 29, 30, & 31 relate only to V3.xx and V4.xx animations, and will be superseded in future releases.
- 32 - Shutdown. CitectSCADA is being shutdown.
- 33 - Reserved for CitectSCADA internal use.
- 34 - 41 - CitectSCADA Confirmation Events. Reserved for CitectSCADA internal use. For the confirmation events, two sets of event type code are defined. The runtime calls the CitectSCADA event handler first, and conditionally proceed to the user's event handler depending on the return value of the CitectSCADA event handler.
- 34 -CitectSCADA Event: Child Window Close Confirmation.
- 35 - CitectSCADA Event: Main Window Close Confirmation.

- 36 - CitectSCADA Event: Maximize Window Confirmation.
- 37 - CitectSCADA Event: Minimize Window Confirmation.
- 38 - CitectSCADA Event: Restore Window Confirmation.
- 39 - CitectSCADA Event: Move Window Confirmation.
- 40 - CitectSCADA Event: Size Window Confirmation.
- 41 - CitectSCADA Event: Shutdown Confirmation Confirmation.
- 42 to 49 - User Confirmation Events. These functions are called when a specific event (mainly from Window title bar) occur and before the runtime performs the intended action. This gives a chance for the user to decide what to do with the event. If the return value is 0, the event will be passed on to the default handler so the intended action will be performed. If the return value is not 0, the event will be ignored and no further action will be taken.
- 42 - Child Window Close Confirmation, when the close button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 43 - Main Window Close Confirmation, when close button of the windows' title bar is clicked which will cause the process to shutdown.
- 44 - Maximize Window Confirmation, when the maximize button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 45 - Minimize Window Confirmation, when the minimize button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 46 - Restore Window Confirmation, when the restore button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 47 - Move Window Confirmation, when the window is being dragged or an equivalent Windows' message is received.
- 48 - Size Window Confirmation, when the windows is being resized or an equivalent Windows' message is received.
- 49 - Shutdown Confirmation, when shutdown() function is called.
- 50 - 127 - Reserved for future CitectSCADA use.
- 128 - 256 - User-defined events. These events are for your own use.

Return Value

The function handle of the existing callback event handler, or -1 if there are no event handlers.

Related Functions

[OnEvent](#), [CallEvent](#), [ChainEvent](#), [SetEvent](#)

Example

```

! Get existing event handler.
hFn=GetEvent(0);
! Trap mouse movements.
OnEvent(0,MouseFn);

..
! Restore old event handler.
SetEvent(0,hFn);
INT
FUNCTION MouseFn()

..
! Chain call old event handler.
RETURN ChainEvent(hFn);
END

```

See Also

[Event Functions](#)

OnEvent

Sets an event callback function for an event type. The callback function is called when the event occurs.

Using callback functions removes the need for polling or checking for events. Callback functions have no arguments and needs to return an integer. They also need to be non-blocking.

CitectSCADA starts running the function immediately, without reading any data from the I/O devices. Any I/O device variable that you use will contain either 0 (zero) or bad quality. Only local variables are supported.

The return value of the callback will depend on the type of the event. Set the Fn argument to 0 (zero) to disable the event.

Notes

- For event type 42..49, a windows system event is received. When the user clicks any button of the Windows tile bar, or size/move the window, or shutting down a process, the callback function is called. If the return value is 0, the event will be processed by CitectSCADA in default mode which is the original behavior. If the return value is not 0, CitectSCADA assumes that you will process the event and discard the message internally.
- If the event handler is non-interactive with instant return value, it can be called directly.
- If the event handler is interactive or with big delay in processing the event, it needs to be called indirectly using the NewTask("EventHandler") function, and the actual

handler, EventHandler(), needs to call WinMode(), WinFree(), or Shutdown() from the handler if it decides the event should not be discarded.

- The "Shutdown Confirmation" event is raised in the following cases:
 - Calling shutdown() without setting the callEvent parameter to zero.
 - Closing the top level application window - after raising the event "Main Window Close Confirmation".
 - Receiving WM_QUIT message what can come from any source both internal and external.
- The "Shutdown Confirmation" event is not raised in other cases including, but not limited to, the following known scenarios:
 - The RuntimeManager is doing stop, restart the program, or doing shutdown all.
 - The Windows Task Manager is doing End Task, End Process, or End Process Tree.

Syntax

OnEvent(*Type*, *Fn*)

Type:

The type of event:

0 - The mouse has moved. When the mouse moves the callback function is called. The return value must be 0.

1 - A key has been pressed. When the user presses a key, the callback function is called after CitectSCADA checks for hot keys. If the return value is 0, CitectSCADA checks for key sequences. If the return value is not 0, CitectSCADA assumes that you will process the key and does not check the key sequence. It is up to you to remove the key from the key command line.

If you are using a right mouse button click as an event, you should read about the ButtonOnlyLeftClick parameter.

2 - Error event. This event is called if an error is detected in Cicode, so you can write a single error function to check for your errors. If the return value is 0, CitectSCADA continues to process the error and generates a hardware error - it may then halt the Cicode task. If the return value is not 0, CitectSCADA assumes that you will process the error, and continues the Cicode without generating a hardware error.

3 - Page user communication error. A communication error has been detected in the data required for this page. If the return value is 0 (zero), CitectSCADA still animates the page. If the return value is not zero, it does not update the page.

- 4 - Page user open. A new page is being opened. This event allows you to define a single function that is called when all pages are opened. The return value must be 0.
- 5 - Page user close. The current page is being closed. This event allows you to define a single function that is called when all pages are closed. The return value must be 0.
- 6 - Page user always. The page is active. This event allows you to define a single function that is called when all pages are active. The return value must be 0.
- 7 - Page communication error. A communication error has been detected in the data required for this page. Reserved for use by CitectSCADA.
- 8 - Page open. This event is called each time a page is opened. Reserved for use by CitectSCADA.
- 9 - Page close. This event is called each time a page is closed. Reserved for use by CitectSCADA.
- 10 - Page always. This event is called while a page is active. Reserved for use by CitectSCADA.
- 11..17 - Undefined.
- 18 - Report start. The report server is about to start a new report. This event is called on the report server. The return value must be 0.
- 19 - Device history. A device history has just completed. The return value must be 0.
- 20 - Login. A user has just logged in.
- 21 - Logout. A user has just logged out.
- 22 - Trend needs repainting. This event is called each time CitectSCADA reanimates a real-time trend or scrolls an historical trend. You should use this event to add additional animation to a trend, because CitectSCADA deletes all existing animation when a trend is re-drawn. (For example, if you want to display extra markers, you must use this event.)
- 23 - Hardware error has been detected.
- 24 - Keyboard cursor moved. This event is called each time the keyboard command cursor moves. The cursor can be moved by the cursor keys, the mouse, or the Cicode function KeySetCursor(). Note that you can find where the keyboard command cursor is located by calling the function KeyGetCursor().
- 25 - Network shutdown. A Shutdown network command has been issued.
- 26 - Runtime system shutdown and restart. (Required because of configuration changes.)
- 27 - Event. An event has occurred.

28 - Accumulator. An accumulator has logged a value.

29 - Slider. A slider has been selected.

30 - Slider. A slider has moved.

31 - Slider. A slider has been released (that is stopped moving).

While responding to slider events 29, 30, and 31, you can set any variables but you cannot call functions that cause immediate changes to animations on the page (for example, DspText() and DspSym()). Types 29, 30, & 31 relate only to V3.xx and V4.xx animations, and will be superseded in future releases.

32 - Shutdown. CitectSCADA is being shutdown.

33 - Reserved for CitectSCADA internal use.

34 - 41 - CitectSCADA Confirmation Events. Reserved for CitectSCADA internal use. For the confirmation events, two sets of event type code are defined. The runtime calls the CitectSCADA event handler first, and conditionally proceed to the user's event handler depending on the return value of the CitectSCADA event handler.

34 - CitectSCADA Event: Child Window Close Confirmation.

35 - CitectSCADA Event: Main Window Close Confirmation.

36 - CitectSCADA Event: Maximize Window Confirmation.

37 - CitectSCADA Event: Minimize Window Confirmation.

38 - CitectSCADA Event: Restore Window Confirmation.

39 - CitectSCADA Event: Move Window Confirmation.

40 - CitectSCADA Event: Size Window Confirmation.

41 - CitectSCADA Event: Shutdown Confirmation Confirmation.

42 to 49 - User Confirmation Events. These functions are called when a specific event (mainly from Window title bar) occur and before the runtime performs the intended action. This gives a chance for the user to decide what to do with the event. If the return value is 0, the event will be passed on to the default handler so the intended action will be performed. If the return value is not 0, the event will be ignored and no further action will be taken.

42 - Child Window Close Confirmation, when the close button of the windows' title bar is clicked or an equivalent Windows' message is received.

43 - Main Window Close Confirmation, when close button of the windows' title bar is clicked which will cause the process to shutdown.

44 - Maximize Window Confirmation, when the maximize button of the windows' title bar is clicked or an equivalent Windows' message is received.

45 - Minimize Window Confirmation, when the minimize button of the windows' title bar is clicked or an equivalent Windows' message is received.

- 46 - Restore Window Confirmation, when the restore button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 47 - Move Window Confirmation, when the window is being dragged or an equivalent Windows' message is received.
- 48 - Size Window Confirmation, when the windows is being resized or an equivalent Windows' message is received.
- 49 - Shutdown Confirmation, when shutdown() function is called.
- 50 - 127 - Reserved for future CitectSCADA use.
- 128 - 256 - User-defined events. These events are for your own use.

Fn:

The function to call when the event occurs. This callback function needs to have no arguments, so you specify the function with no parentheses (). The callback function needs to return INT as its return data type. You cannot specify a CitectSCADA built-in function as a callback function.

Set *Fn* to 0 to disable the event.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GetEvent](#), [CallEvent](#), [ChainEvent](#)

Examples

Example 1 - Calls a function called KeyFn to determine if the ESC key has been pressed on a key press event.

```
OnEvent(1,KeyFn);
INT
FUNCTION KeyFn()
    INT Key;
    Key=KeyPeek(0);
    IF Key=27 THEN
        Prompt("ESC pressed");
        RETURN 1;
    ELSE
        RETURN 0;
    END
END
```

Example 2 - Calls a function called MouseFn to display the position of the mouse whenever it is moved.

```
OnEvent(0,MouseFn);
INT
FUNCTION MouseFn()
    INT X,Y;
    DspGetMouse(X,Y);
    RETURN 0;
END
```

Example 3 - Presents a user with a confirmation dialog box when the main window close button is pressed.

```
sFUNCTION XYZStartup()
    OnEvent(43, ConfirmShutdown);
END

INT FUNCTION ConfirmShutdown()
    TaskNew("_ShutdownDlg", "", 2+8);
    RETURN 1;
END

FUNCTION _ShutdownDlg()
    STRING sMsg = "Are you sure ?";
    INT nRC;

    nRC = Message("Close this window and shutdown", sMsg, 1+32);

    IF nRC = 0 THEN
        Shutdown("", "", 1, "", 0);
    END
END
```

See Also

[Event Functions](#)

SetEvent

Sets an event callback function by specifying a function handle. You can use this function with the GetEvent() function to restore an old event handler.

Syntax

SetEvent(*Type*, *hFn*)

Type:

The type of event:

0 - The mouse has moved. When the mouse moves the callback function is called. The return value must be 0.

1 - A key has been pressed. When the user presses a key, the callback function is called after CitectSCADA checks for hot keys. If the return value is 0, CitectSCADA checks for key sequences. If the return value is not 0, CitectSCADA assumes that you will process the key and does not check the key sequence. It is up to you to remove the key from the key command line.

If you are using a right mouse button click as an event, you should read about the ButtonOnlyLeftClick parameter.

2 - Error event. This event is called if an error is detected in Cicode, so you can write a single error function to check for your errors. If the return value is 0, CitectSCADA continues to process the error and generates a hardware error - it may then halt the Cicode task. If the return value is not 0, CitectSCADA assumes that you will process the error, and continues the Cicode without generating a hardware error.

3 - Page user communication error. A communication error has been detected in the data required for this page. If the return value is 0 (zero), CitectSCADA still animates the page. If the return value is not zero, it does not update the page.

4 - Page user open. A new page is being opened. This event allows you to define a single function that is called when all pages are opened. The return value must be 0.

5 - Page user close. The current page is being closed. This event allows you to define a single function that is called when all pages are closed. The return value must be 0.

6 - Page user always. The page is active. This event allows you to define a single function that is called when all pages are active. The return value must be 0.

7 - Page communication error. A communication error has been detected in the data required for this page. Reserved for use by CitectSCADA.

8 - Page open. This event is called each time a page is opened. Reserved for use by CitectSCADA.

9 - Page close. This event is called each time a page is closed. Reserved for use by CitectSCADA.

10 - Page always. This event is called while a page is active. Reserved for use by CitectSCADA.

11..17 - Undefined.

18 - Report start. The report server is about to start a new report. This event is called on the report server. The return value must be 0.

19 - Device history. A device history has just completed. The return value must be 0.

- 20 - Login. A user has just logged in.
- 21 - Logout. A user has just logged out.
- 22 - Trend needs repainting. This event is called each time CitectSCADA re-animates a real-time trend or scrolls an historical trend. You should use this event to add additional animation to a trend, because CitectSCADA deletes all existing animation when a trend is re-drawn. (For example, if you want to display extra markers, you must use this event.)
- 23 - Hardware error has been detected.
- 24 - Keyboard cursor moved. This event is called each time the keyboard command cursor moves. The cursor can be moved by the cursor keys, the mouse, or the Cicode function KeySetCursor(). Note that you can find where the keyboard command cursor is located by calling the function KeyGetCursor().
- 25 - Network shutdown. A Shutdown network command has been issued.
- 26 - Runtime system shutdown and restart. (Required because of configuration changes.)
- 27 - Event. An event has occurred.
- 28 - Accumulator. An accumulator has logged a value.
- 29 - Slider. A slider has been selected.
- 30 - Slider. A slider has moved.
- 31 - Slider. A slider has been released (that is stopped moving).
While responding to slider events 29, 30, and 31, you can set any variables but you cannot call functions that cause immediate changes to animations on the page (for example, DspText() and DspSym()). Types 29, 30, & 31 relate only to V3.xx and V4.xx animations, and will be superseded in future releases.
- 32 - Shutdown. CitectSCADA is being shutdown.
- 33 - Reserved for CitectSCADA internal use.
- 34 - 41 - CitectSCADA Confirmation Events. Reserved for CitectSCADA internal use. For the confirmation events, two sets of event type code are defined. The runtime calls the CitectSCADA event handler first, and conditionally proceed to the user's event handler depending on the return value of the CitectSCADA event handler.
- 34 - CitectSCADA Event: Child Window Close Confirmation.
- 35 - CitectSCADA Event: Main Window Close Confirmation.
- 36 - CitectSCADA Event: Maximize Window Confirmation.
- 37 - CitectSCADA Event: Minimize Window Confirmation.
- 38 - CitectSCADA Event: Restore Window Confirmation.
- 39 - CitectSCADA Event: Move Window Confirmation.

- 40 - CitectSCADA Event: Size Window Confirmation.
- 41 - CitectSCADA Event: Shutdown Confirmation Confirmation.
- 42 to 49 - User Confirmation Events. These functions are called when a specific event (mainly from Window title bar) occur and before the runtime performs the intended action. This gives a chance for the user to decide what to do with the event. If the return value is 0, the event will be passed on to the default handler so the intended action will be performed. If the return value is not 0, the event will be ignored and no further action will be taken.
- 42 - Child Window Close Confirmation, when the close button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 43 - Main Window Close Confirmation, when close button of the windows' title bar is clicked which will cause the process to shutdown.
- 44 - Maximize Window Confirmation, when the maximize button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 45 - Minimize Window Confirmation, when the minimize button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 46 - Restore Window Confirmation, when the restore button of the windows' title bar is clicked or an equivalent Windows' message is received.
- 47 - Move Window Confirmation, when the window is being dragged or an equivalent Windows' message is received.
- 48 - Size Window Confirmation, when the windows is being resized or an equivalent Windows' message is received.
- 49 - Shutdown Confirmation, when shutdown() function is called.
- 50 - 127 - Reserved for future CitectSCADA use.
- 128 - 256 - User-defined events. These events are for your own use.

hFn

The function handle, as returned from the GetEvent() function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GetEvent](#)

Example

See [GetEvent](#).

See Also

[Event Functions](#)

Chapter: 29 File Functions

The file functions provide access to standard ASCII files. You can open or create files and then read and write data in free format. Use these functions when you require more complex file operations than are possible with the device functions. For example, importing and exporting data to and from other programs (that support ASCII files).

You can build complex I/O functionality by combining these functions with the format functions.

File Functions

Following are functions relating to file operations:

FileClose	Closes a file.
FileCopy	Copies a file or group of files.
FileDelete	Deletes a file.
FileEOF	Checks for the end of a file.
FileExist	Checks if a file exists.
FileFind	Finds a file that matches a specified search criteria.
FileFindClose	Closes a find (started with FileFind) that did not run to completion.
FileGetTime	Gets the time on a file.
FileMakePath	Creates a file path string from individual components.
FileOpen	Opens or creates an ASCII file.
FilePrint	Prints a file on a device.
FileRead	Reads characters from a file.
FileReadBlock	Reads a block of characters from a file.

FileReadLn	Reads a line from a file.
FileRename	Renames a file.
FileRichTextPrint	Prints a rich text file.
FileSeek	Seeks a position in a file.
FileSetTime	Sets the time on a file.
FileSize	Gets the size of a file.
FileSplitPath	Splits a file path into individual string components.
FileWrite	Writes characters to a file.
FileWriteBlock	Writes a block of characters to a file.
FileWriteLn	Writes a line to a file.

See Also

[Functions Reference](#)

FileClose

Closes a file. All data written to the file is flushed to disk when the file is closed, and the file number becomes invalid.

Syntax

FileClose(*File*)

File:

The file number.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FileOpen](#)

Example

```
File=FileOpen("C:\Data\Report.Txt", "r");
..
! Do file operations.
..
! Close the file.
FileClose(File);
```

See Also

[File Functions](#)

FileCopy

Copies a file. You can use the DOS wild card characters (*) and (?) to copy groups of files. In asynchronous Mode, this function will return immediately and the copy will continue in the background. Unless you are accessing to the floppy drive, copying files does not interfere with the operation of other CitectSCADA tasks, because this function is time-sliced.

Syntax

FileCopy(*Source*, *Dest*, *Mode*)

Source:

The name of the source file to copy.

Dest:

The name of destination file to copy to. To copy a file to the printer, specify the name as "LPT1.DOS".

Mode:

The copy mode:

0 - Normal

1 - Copy only if the file time is different.

2 - Copy in asynchronous mode.

Multiple modes can be selected by adding them together (for example, set *Mode* to 3 to copy in asynchronous mode if the file time is different).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned. However, if you copy in asynchronous mode, the return value does not reflect whether the copy operation was successful or not, because the function returns before the actual copy is complete.

Related Functions

[FileDelete](#)

Example

```
! Copy Report.Txt to Report.Bak.  
FileCopy ("C:\Data\Report.Txt", "C:\Data\Report.Bak",0);  
/* Copy AlarmLog.Txt to AlarmLog.Bak only if the file time is  
different. Copy in the background. */  
FileCopy ("AlarmLog.Txt", "AlarmLog.Bak",1+2);
```

See Also

[File Functions](#)

FileDelete

Deletes a file.

Syntax

FileDelete(*Name*)

Name:

The name of the file to delete.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FileCopy](#)

Example

```
! Delete old report file.  
FileDelete("C:\Data\Report.Txt");
```

See Also

[File Functions](#)

FileEOF

Determines if the end of the file has been reached.

Syntax

FileEOF(*File*)

File:

The file number.

Return Value

1 if the end of file has been reached, otherwise 0 (zero).

Related Functions

[FileSeek](#)

Example

```
WHILE NOT FileEOF(File) DO  
    Str=FileReadLn(File);  
END
```

See Also

[File Functions](#)

FileExist

Checks if a file exists. If the return value is 1, the file exists.

Syntax

FileExist(*Name*)

Name:

The name of the file to check.

Return Value

TRUE (1) if the file exists, otherwise FALSE (0).

Related Functions

[FileOpen](#)

Example

```
! Check if the file exists
IF FileExist("C:\Data\Report.Txt") THEN
    ! The file exists
END
```

See Also

[File Functions](#)

FileFind

Finds a file that matches a specified search criteria. To find a list of files, you need to first call this function with the required path and mode (to find the first file), then call the function again with an empty path and a mode of 0 (to find the remaining files). After the last file is found, an empty string is returned.

If the search is for multiple files, FileFindClose needs to be called if the search does not run to completion (for example, you do not run until an empty string is returned).

Syntax

FileFind(*sPath*, *nMode*)

sPath:

The name of the file to check. To search for multiple files, the wildcards * and ? can be used to match multiple entries.

nMode:

The type of file to check:

0 - Normal files (includes files with read-only and archived attributes)

1 - Read-only files only

2 - Hidden files only

4 - System files only

16 - Subdirectories only

32 - Archived files only

128 - Files with no attributes only

These numbers can be added together to search for multiple types of files during one search.

Return Value

The full path and filename. If no files are found, an empty string is returned.

Related Functions

[FileOpen](#), [FileSplitPath](#), [FileMakePath](#)

Example

```

! Search for all dBase files in the run directory and make a backup
sPath = FileFind("[run]:\*.dbf", 0);
WHILE StrLength(sPath) > 0 DO
    FileSplitPath(sPath, sDrive, sDir, sFile, sExt);
    sBak = FileMakePath(sDrive, sDir, sFile, "BAK");
    FileCopy(sPath, sBak, 0);
    ! Find the next file
    sPath = FileFind("", 0);
END

```

See Also

[File Functions](#)

FileFindClose

Closes a find (started with `FileFind`) that did not run to completion.

Syntax

FileFindClose()

Return Value

0 if no error is detected, or a Cicode error code if an error occurred.

Related Functions

[FileFind](#)

Example

```

//Find the first dbf file starting with fred
sPath = FileFind("[run]:\fred*.dbf", 0);
IF (StrLength(sPath) > 0) THEN
    //Do work here
    FileFindClose();
END

```

See Also

[File Functions](#)

FileGetTime

Gets the time on a file.

Syntax

FileGetTime(*File*)

File:

The file number.

Return Value

The file time of the file (in the CitectSCADA time/date variable format).

Related Functions

[FileOpen](#), [FileClose](#), [FileSetTime](#)

Example

```
File = FileOpen("[data]:report.txt", "r");
! Get the time of the file
iTime = FileGetTime(File);
FileClose(File);
```

See Also

[File Functions](#)

FileMakePath

Creates a file path string from individual components.

Syntax

FileMakePath(*sDrive*, *sDir*, *sFile*, *sExt*)

sDrive:

The disk drive.

sDir:

The directory string.

sFile:

The file name (without the extension).

sExt:

The file extension.

Return Value

The full path as a string.

Related Functions

[FileSeek](#), [FileFind](#), [FileSplitPath](#)

Example

See [FileFind](#)

See Also

[File Functions](#)

FileOpen

Opens a file and returns a file number that can be used by other file functions. The maximum file size supported is 1 Megabyte for displaying text files.

You can also use this function to check if a file exists, by opening the file in read-only mode. A return value of -1 indicates that the file cannot be opened.

ErrSet(1) needs to be in the previous line of your code, else the execution stops and a hardware [error](#) is generated. If ErrSet(1) is used then it doesn't halt, and -1 is returned.

Syntax

FileOpen(*Name*, *Mode*)

Name:

The name of the file to open.

Mode:

The mode of the opened file:

"*a*" - Append mode. Allows you to append to the file without removing the end of file marker. The file cannot be read. If the file does not exist, it will be created.

"*a+*" - Append and read modes. Allows you to append to the file and read from it. The end of file marker will be removed before writing and restored when writing is complete. If the file does not exist, it will be created.

"*r*" - Read-only mode. Allows you to (only) read from the file. If the file does not exist or cannot be found, the function call will return the value -1.

"*r*+" - Read/write mode. Allows you to read from, and write to, the file. If the file already exists (before the function is called), its contents will be deleted. If the file does not exist or cannot be found, the function call will return the value -1.

"*w*" - Write mode. Opens an empty file for writing. If the file already exists (before the function is called), its contents will be deleted. If the file does not exist or cannot be found, the file will be created.

"*w*+" - Read/write mode. Opens an empty file for both reading and writing. If the file already exists (before the function is called), its contents will be deleted. If the file does not exist or cannot be found, the file will be created.

Return Value

The file number. If the file cannot be opened, -1 is returned and the code is halted.

Related Functions

[FileClose](#), [FileRead](#), [FileReadLn](#), [FileWrite](#), [FileWriteLn](#)

Example

```
! Open a file in read-only mode.  
ErrSet(1);  
File=FileOpen("C:\Data\Report.Txt", "r");  
ErrSet(0);
```

See Also

[File Functions](#)

FilePrint

Prints a file on a device.

Syntax

FilePrint(*Devicename*, *Filename*)

Devicename:

The name of the target device.

Filename:

The name of the file to print on the device.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FileOpen](#), [FileClose](#), [FileWrite](#), [FileWriteLn](#)

Example

```
! Print a data file on the system printer.
FilePrint("Printer_Device","Data.txt");
```

See Also

[File Functions](#)

FileRead

Reads a number of characters from a file. The string can contain less characters than requested if the end of file is reached. A maximum of 255 characters can be read in each call.

Syntax

FileRead(*File*, *Length*)

File:

The file number.

Length:

The number of characters to read.

Return Value

The text from the file (as a string).

Related Functions

[FileOpen](#), [FileClose](#), [FileReadLn](#)

Example

```
WHILE NOT FileEOF(File) DO
  Str=FileRead(File,20);
```

```
END
```

See Also

[File Functions](#)

FileReadBlock

Reads a number of characters from a file. The buffer can contain less characters than requested if the end of file is reached. A maximum of 255 characters can be read in each call. The data should be treated as a binary data and should not be passed to string functions. You may use StrGetChar() function to extract each character from the buffer, or pass the buffer to another function which will accept binary data.

Syntax

FileReadBlock(File, Buffer, Length)

File:

The file number.

Buffer:

The buffer to return the binary data. This may be a string or a string packed with binary data. The string terminator is ignored and the length argument specifies the number of characters to write.

Length:

The number of characters to read.

Return Value

The number of characters read from the file. When the end of the file is found 0 will be returned. If an error occurs -1 will be returned and IsError() will return the [error](#) code.

Related Functions

[FileOpen](#), [FileClose](#), [FileRead](#), [FileWriteBlock](#), [StrGetChar](#)

Example

```
// read binary file and copy to COM port
length = FileReadBlock(File, buf, 128);
WHILE length > 0 DO
    ComWrite(hPort, buf, length, 0);
    length = FileReadBlock(File, buf, 128);
END
```

See Also[File Functions](#)**FileReadLn**

Reads a line from a file. Up to 255 characters can be returned. The carriage return and newline characters are not returned. If the line is longer than 255 characters, the error overflow (code 275) is returned - you should call this function again to read the rest of the line.

Syntax**FileReadLn**(*File*)*File:*

The file number.

Return Value

The text, as a string.

Related Functions[FileOpen](#), [FileClose](#), [FileRead](#)**Example**

```
sLine = FileReadLn(hFile);
! do stuff with the string
WHILE IsError() = 275 DO
    ! read the rest of the line
    sLine = FileReadLn(hFile);
    ! do stuff with the rest of the line
END
```

See Also[File Functions](#)**FileRename**

Renames a file.

Syntax**FileRename**(*Oldname*, *Newname*)*Oldname*:

The original name of the file.

Newname:

The new name of the file.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FileCopy](#), [FileDelete](#)

Example

```
! Rename REPORT.TXT as REPORT.OLD.  
FileRename ("C:\Data\Report.Txt", "C:\Data\Report.Old");
```

See Also

[File Functions](#)

FileRichTextPrint

Prints the rich text file *sFilename* to the printer given by *sPortName*.

Syntax

FileRichTextPrint(*sFilename*, *sPortName*)

sFilename:

The filename of the rich text format file. The filename needs to be entered in quotation marks "".

Remember that the filename for a saved report comes from the File Name field in the Devices form. The location of the saved file needs to also be included as part of the filename. For example, if the filename in the Devices form listed [Data]\RepDev.rtf, then you would need to enter "[Data]\r-epdev.rtf" as your argument. Alternatively, you can manually enter the path, for example, "c:\My-Application\data\repdev.rtf".

If you are keeping a number of history files for the report, instead of using the extension rtf, you need to change it to reflect the number of the desired history file, for example, 001.

PortName:

The name of the printer port to which the rich text file will be printed. This name needs to be enclosed within quotation marks "". For example "LPT1", to print to the local printer, or "\\\Pserver\canon1" using UNC to print to a network printer.

Return Values

0 if successful, otherwise an [error](#) is returned.

Related Functions

[PageRichTextFile](#), [DspRichTextPrint](#)

Example

```
// This would print the file [Data]\richtext.rtf to LPT1. Remember
// that the [Data] path is specified in the Citect.ini file. The file
// richtext.rtf is the name of the output file for the report, as
// specified in the Devices form. //
iResult = FileRichTextPrint("[Data]\richtext.rtf","LPT1:");
// This would print the file f:\citect\data\richtext.rtf to LPT1.
// The file richtext.rtf is the name of the output file for the
// report, as specified in the Devices form. //
iResult =
FileRichTextPrint("f:\citect\data\richtext.rtf","LPT1:");
```

See Also

[File Functions](#)

FileSeek

Moves the file pointer to a specified position in a file.

Syntax

FileSeek(*File*, *Offset*)

File:

The file number.

Offset:

The offset in bytes, from 0 to (maximum file size -1). This value needs to be ≥ 0 .

Return Value

The new file position, or -1 if an [error](#) is detected.

Related Functions

[FileSize](#)

Example

```
! Seek to the start of the file.  
FileSeek(File,0);
```

See Also

[File Functions](#)

FileSetTime

Sets the time on a file.

Note: In order for this function to work, the file needs to first be opened in write or read/write mode.

Syntax

FileSetTime(*File*, *iTime*)

File:

The file number.

iTime:

The new file time, in the CitectSCADA time/date variable format.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FileOpen](#), [FileClose](#), [FileGetTime](#)

Example

```
File = FileOpen("[data]:report.txt", "r+");  
! set the file to the current time  
FileSetTime(File, TimeCurrent());  
FileClose(File);
```

See Also

[File Functions](#)

FileSize

Gets the size of a file.

Syntax

FileSize(*File*)

File:

The file number.

Return Value

The size of the file, in bytes.

Related Functions

[FileSeek](#)

Example

```
! Get the size of the file.  
Size=FileSize(File);
```

See Also

[File Functions](#)

FileSplitPath

Splits a file path into individual string components. You enter the full path string as *sPath*. The individual components of the path name are returned in the arguments *sDrive*, *sDir*, *sFile*, and *sExt*.

Syntax

FileSplitPath(*sPath*, *sDrive*, *sDir*, *sFile*, *sExt*)

sPath:

The full path string.

sDrive:

The disk drive.

sDir:

The directory string.

sFile:

The file name (without the extension).

sExt:

The file extension.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FileSeek](#), [FileFind](#), [FileMakePath](#)

Example

See [FileFind](#)

See Also

[File Functions](#)

FileWrite

Writes a string to a file. The string is written at the current file position.

Syntax

FileWrite(*File*, *String*)

File:

The file number.

String:

The string to write.

Return Value

The number of characters written.

Related Functions

[FileOpen](#), [FileClose](#), [FileWriteLn](#)

Example

```
! Write to the file.  
FileWrite(File,"Data");
```

See Also[File Functions](#)**FileWriteBlock**

Writes a string or buffer to a file. The data is written at the current file position. You may create the binary data by using the StrSetChar function or by reading the data from some other function. This function is similar to the FileWrite() function however you specify the length of data to write to the file. The FileWrite() function will send the data to the file until the sting terminator is found. FileWriteBlock() will ignore any string terminator and copy the length of bytes to the file. This allows this function to be used for binary data transfer.

Syntax**FileWriteBlock(*File*, *Buffer*, *Length*)***File:*

The file number.

Buffer:

The data to write to the file. This may be a string or a string packed with binary data. The string terminator is ignored and the length argument specifies the number of characters to write.

Length:

The number of characters to write. The maximum number of characters you may write in one call is 255. (If you use a string without a terminator in a function that expects a string, or in a Cicode expression, you could get invalid results.) To use the string to build up a buffer, you do not need the terminating 0 (zero).

Return Value

The number of characters written to the file. If an error is detected -1 will be returned and IsError() will return the [error](#) code.

Related Functions[FileOpen](#), [FileClose](#), [FileWrite](#), [FileReadBlock](#), [StrSetChar](#)**Example**

```
STRING buf;
FOR I = 0 TO 20 DO
    StrSetChar(buf, I, I); // put binary data into string
END
```

```
! Write binary data to the file.  
FileWrite(File, buf, 20);
```

See Also

[File Functions](#)

FileWriteLn

Writes a string to a file, followed by a newline character. The string is written at the current file position.

Syntax

FileWriteLn(*File*, *String*)

File:

The file number.

String:

The string to write.

Return Value

The number of characters written (including the carriage return and newline characters).

Related Functions

[FileOpen](#), [FileClose](#), [FileWrite](#)

Example

```
! Write a line to the file.  
FileWriteLn(File,"Line of file data");
```

See Also

[File Functions](#)

Chapter: 30 Form Functions

Form functions create and display data entry forms. Use them to display large amounts of data or request data from the operator; for example, to display, load, and/or edit a database of recipes.

Form Functions

Following are functions relating to forms:

<u>FormActive</u>	Checks if a form is currently active.
<u>FormAddList</u>	Adds a text string to a list box or combo box.
<u>FormButton</u>	Adds a button to a form.
<u>FormCheckBox</u>	Adds a check box to the current form.
<u>FormComboBox</u>	Adds a combo box to the current form.
<u>FormCurr</u>	Gets the current form and field handles.
<u>FormDestroy</u>	Removes a form from the screen.
<u>FormEdit</u>	Adds edit fields to a form.
<u>FormField</u>	Adds general fields to a form.
<u>FormGetCurrInst</u>	Gets data associated with a field.
<u>FormGetData</u>	Gets the data associated with a form.
<u>FormGetInst</u>	Gets data associated with a field on a form.
<u>FormGetText</u>	Gets field text on an active form.
<u>FormGoto</u>	Go to a specified form.
<u>FormGroupBox</u>	Adds a group box to the current form.

FormInput	Adds an input field to a form.
FormListAddText	Adds a new text entry to a combo box or a list box.
FormListBox	Adds a list box to the current form.
FormListDeleteText	Deletes existing text from combo box or list box.
FormListSelectText	Selects (highlights) a text entry in a combo box or a list box.
FormNew	Creates a new form.
FormNumPad	Provides a keypad form for the operator to add numeric values.
FormOpenFile	Displays a File Open dialog box.
FormPassword	Adds a password (no echo) input field.
Form-SecurePassword	Adds a secure password (no echo) input field.
FormPosition	Sets the position of a form on the screen, before it is displayed.
FormPrompt	Adds a prompt to a form.
FormRadioButton	Adds a radio button to the current form.
FormRead	Displays a form and reads user input.
FormSaveAsFile	Displays a File Save As dialog box.
FormSelectPrinter	Displays the Select Printer dialog box.
FormSetData	Sets data in a form.
FormSetInst	Associates data to a field on a form.
FormSetText	Sets field text on an active form.
FormWndHnd	Gets the window handle for the given form.

See Also

[Functions Reference](#)

FormActive

Checks if a form is currently active (displayed on the screen). This function is useful when forms are being displayed in asynchronous mode and another Cicode task is trying to access the form.

Syntax

FormActive(*hForm*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

Return Value

TRUE (1) if the form is active or FALSE (0) if it is not.

Related Functions

[FormDestroy](#), [FormNew](#)

Example

See [FormDestroy](#)

See Also

[Form Functions](#)

FormAddList

Adds a text string to a list box or combo box. You should call this function only after the FormNew() function, and immediately after either the FormComboBox() or the FormListBox(), and before the FormRead() function otherwise an error is returned. The text is added at the end of the list box or combo box.

To add text to a form that is already displayed, use the FormListAddText() function, and use the FormListSelectText() function to highlight text on the list.

Syntax

FormAddList(*sText*)

sText:

The text string to add to the list box or combo box.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormNew](#), [FormRead](#), [FormListBox](#), [FormComboBox](#), [FormListAddText](#), [FormListDeleteText](#), [FormListSelectText](#)

Example

See [FormComboBox](#) and [FormListBox](#)

See Also

[Form Functions](#)

FormButton

Adds a button to the current form. You can add buttons that run callback functions (specified in *Fn*) to perform any actions you need, as well as the standard buttons - an [OK] button to save the operator's entries and close the form, and a [Cancel] button to close the form but discard the changes.

You should call this function only after the FormNew() function and before the FormRead() function. The button is added to the form at the specified column and row position. The width of the button is automatically sized to suit the text.

Syntax

FormButton(*Col*, *Row*, *sText*, *Fn*, *Mode*)

Col:

The number of the column in which the button will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the button in column 8, enter 7.

Row:

The number of the row in which the button will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the button in row 6, enter 5.

sText:

The text to display on the button.

Fn:

The callback function to call when the button is selected. Set to 0 to call no function. Please be aware that the Fn parameter needs to be of type INT and the callback function cannot contain a blocking function.

Mode:

Button mode:

0 - **Normal** button. When this button is selected the callback function is called.

1 - **OK** button. When this button is selected, the form is closed, and all operator-entered data is copied to buffers (specified by the other form functions). FormRead() returns 0 (zero) to indicate a successful read. The callback function specified in Fn is called. Be aware that this mode destroys the form.

2 - **Cancel** button. When this button is selected, the form is closed and operator-entered data is discarded. FormRead() returns with an [error 299](#). The callback function specified in Fn is called. Be aware that this mode destroys the form.

Return Value

The field handle if the button is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#)

Example

```

! Create a form, add buttons and then display the form on the
current page
FUNCTION FnMenu()
    FormNew("MENU",20,6,1);
    FormButton(0 ,4 , " FIND ", FindMenu, 0);
    FormButton(10,4 , " TAG ", ShowTag, 0);
    FormButton(0 ,5 , " CANCEL ", KillForm, 0);
    FormButton(10,5 , " GOTO ", GotoPg, 0);
    FormRead(0);
END

```

See Also

[Form Functions](#)

FormCheckBox

Adds a check box to the current form. The check box is a form control that allows the operator to make individual selections. Each check box can be either checked (true) or cleared (false).

You should call this function only after the FormNew() function and before the FormRead() function. The check box is added to the form at the specified column and row position. The width of the button is automatically sized to suit the text.

Syntax

FormCheckBox(*Col*, *Row*, *sText*, *sBuf*)

Col:

The number of the column in which the check box will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the check box in column 8, enter 7.

Row:

The number of the row in which the check box will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the check box in row 6, enter 5.

sText:

The text associated with the check box.

sBuf:

The string buffer in which to put the state of the check box. You should initialize this buffer to the state of the check box. When the form returns, this buffer will contain either '1' or '0' if the box is checked.

Return Value

The field handle if the check box is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#)

Example

```
! Create a form, add check boxes, and display the form.  
! The operator may select none or all of the check boxes.  
FUNCTION FnMenu()  
    STRING sNuts, sCherrys, sChocolate;  
    sNuts      = "1";  
    sCherrys   = "0";  
    sChocolate = "1";  
    FormNew("IceCream",20,6,1);      ]  
    FormCheckBox(2,2,"Nuts",        sNuts);  
    FormCheckBox(2,3,"Cherrys",     sCherrys);  
    FormCheckBox(2,4,"Chocolate",   sChocolate);  
    FormRead(0);  
    If sNuts = "1" THEN  
        ! add the nuts  
    END  
    IF sCherrys = "1" THEN  
        ! add the cherrys  
    END  
    IF sChocolate = "1" THEN  
        ! add the chocolate  
    END  
END
```

See Also

[Form Functions](#)

FormComboBox

Adds a combo box to the current form. A combo box is a form control that allows the operator to type a selection or make a single selection from a text list.

You should call this function only after the FormNew() function and before the FormRead() function. The combo box is added to the form at the specified column and row position with the specified width and height. If more items are placed in the list than the list can display, a scroll bar displays (to scroll to the hidden items).

Use the FormAddList() function to add items for display in the list box. If the form is already displayed, you can use the FormListAddText() and FormListSelectText() functions to add (and highlight) text in the list box.

Syntax

FormComboBox(*Col*, *Row*, *Width*, *Height*, *sBuf* [, *Mode*])

Col:

The number of the column in which the combo box will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the combo box in column 8, enter 7.

Row:

The number of the row in which the combo box will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the combo box in row 6, enter 5.

Width:

The width of the list box, which should be wide enough to display your widest item. Items wider than the list box are clipped.

Height:

The height of the list box (the number of items that can be seen in the list box without scrolling).

sBuf:

The string buffer in which to store the selected item. The *sBuf* parameter can also hold the starting selection for the Combo box. For example if you set the *sBuf* string to "HELLO" before calling FormComboBox, HELLO will be displayed in the box upon opening the form.

Mode:

The mode in which to create the combo box:

0 - Sort the combo box elements alphabetically.

1 - Place elements in combo box in the order they were added.

Default mode is 0.

Return Value

The field handle if the combo box is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#), [FormAddList](#), [FormListAddText](#), [FormListSelectText](#), [FormListBox](#)

Example

```
! Create a form, add combo box and then display the form
! the operator may type in or select one of the items from the list
FUNCTION FnMenu()
    STRING    sBuf;
    FormNew("Select Item",20,6,1);
    FormComboBox(2 ,2, 15, 5, sBuf, 1);
    FormAddList("Item One");
    FormAddList("Item two");
    FormAddList("Item three");
    FormRead(0);
    ! sBuf should contain the selected item or entered text
END
```

See Also

[Form Functions](#)

FormCurr

Gets the form and field handles for the current form and field. You should call this function only from within a callback function. You can then use the same callback function for all forms and fields, regardless of how the boxes, buttons, etc. on the forms are used. You should use this function with the FormGetInst() function.

Syntax

FormCurr(*hForm*, *hField*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormGetInst](#)

Example

See [FormGetInst](#).

See Also

[Form Functions](#)

FormDestroy

Destroys a form, that is removes it from the screen. Use this function (from an event) to close a form.

Syntax

FormDestroy(*hForm*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormNew](#)

Example

```
/* Display message to the operator. If after 10 seconds the
operator has not selected OK, then destroy the form. */
hForm=FormNew("Hello",4,20,0);
FormPrompt(1,1,"Something bad has happened");
FormButton(5,2,"OK",0,1);
FormRead(1);
! Wait 10 seconds.
Sleep(10);
IF FormActive(hForm) THEN
```

```
! Destroy form.  
FormDestroy(hForm);  
END
```

See Also

[Form Functions](#)

FormEdit

Adds an edit field to the current form. You should call this function only after the Form-New() function and before the FormRead() function. A user input/edit box is added to the form at the specified column and row position. The operator can enter or edit the text in the edit box. This text is then passed to this function as *Text*.

Syntax

FormEdit(*Col*, *Row*, *Text*, *Width* [, *maxTextLength*])

Col:

The number of the column in which the edit field will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the edit field in column 8, enter 7.

Row:

The number of the row in which the edit field will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the edit field in row 6, enter 5.

Text:

The text in the edit field. Text initially contains the default text (if any) for the operator to edit. When the function closes, this argument is passed back with the operator's input.

Width:

The width of the edit field.

maxTextLength:

This optional parameter specifies the maximum length of input text. The default value is 0 meaning the string can have the maximum length allowed in the system (Cicode allows strings of 255 characters).

Return Value

The field handle if the string is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#)

Example

```
STRING Recipe;
FormNew("Recipe",5,30,0);
! Add edit field, default Recipe to "Jam".
Recipe="Jam";
FormEdit(1,2,Recipe,20);
! Read the form.
FormRead(0);
! Recipe will now contain the operator-entered data.
```

See Also

[Form Functions](#)

FormField

Adds a field control device (such as a button , check box, or edit field) to the current form. You should call this function only after the FormNew() function and before the FormRead() function. This function allows you to specify a control device with more detail than the other field functions.

Syntax

FormField(*Col*, *Row*, *Width*, *Height*, *Type*, *Buffer*, *Label*, *Fn* [, *maxTextLength*])

Col:

The number of the column in which the control will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the control in column 8, enter 7.

Row:

The number of the row in which the control will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the control in row 6, enter 5.

Width:

The width of the control device.

Height:

The height of the control device.

Type:

The type of control device:

0 - None

1 - Edit

2 - Edit Password

- 3 - Text
- 4 - Button
- 5 - OK button
- 6 - Cancel button
- 7 - Group box
- 8 - Radio button
- 9 - Check box

Buffer:

The output buffer for the field string. The default value of the control device is initialized to the value of the buffer. If you specify a Radio button or Check box, you should initialize the buffer to "0" or "1". The result of the field will also be set to "0" or "1".

Label:

The display label for a button, or the default label for an edit field

Fn:

The callback function to call when the button is selected. Set to 0 to call no function. Please be aware that the Fn parameter needs to be of type INT, and the callback function cannot contain a blocking function. For types other than 4,5, and 6, set this argument to 0.

maxTextLength:

This optional parameter specifies the maximum length of input text for edit fields (this parameter is ignored for other controls). The default value is 0 meaning the string can have the maximum length allowed in the system (Cicode allows strings of 255 characters).

Return Value

The field handle if the field is successfully added, otherwise it will return -1.

Related Functions

[FormNew](#)

Example

```
! Display a form with check boxes to start
!! specific motors.
FUNCTION SelectMotor()
    INT hform;
    STRING check1 = "0";
    STRING check2 = "0";
    STRING check3 = "0";
    hform = FormNew("Selection Menu", 26, 22, 6);
    FormField(16, 1, 12, 1, 9, check1, "Primary ", 0);
```

```

FormField(16, 2, 12, 1, 9, check2, "Secondary", 0);
FormField(16, 3, 12, 1, 9, check3, "backup    ", 0);
FormButton( 9, 20, " &Cancel ", 0, 2);
IF FormRead(0) = 0 THEN
    IF check1 = "1" THEN
        StartMotor(MOTOR_1);
    END
    IF check2 = "1" THEN
        StartMotor(MOTOR_2);
    END
    IF check3 = "1" THEN
        StartMotor(MOTOR_3);
    END
END

```

See Also[Form Functions](#)**FormGetCurrInst**

Extracts data associated with a field (set by the FormSetInst() function). You should call this function only from within a field callback function. This function is the same as calling the FormCurr() function and then the FormGetInst() function.

Syntax**FormGetCurrInst(*iData*, *sData*)***iData:*

Integer data.

sData:

String data.

Return Value0 (zero) if successful, otherwise an [error](#) is returned.**Related Functions**[FormCurr](#), [FormGetInst](#), [FormSetInst](#)**Example**

```

INT
FUNCTION GetNextRec()

```

```
INT hDev;
STRING Str;
FormGetCurrInst(hDev,Str);
DevNext(hDev);
RETURN 0;
END
```

See Also

[Form Functions](#)

FormGetData

Gets all data associated with a form and puts it into the output string buffers. Normally the field data is copied to the output string buffers only when the user selects the [OK] button. If you want to use the data while the form is displayed, call this function to get the data. You should call this function only while the form is displayed otherwise an error is returned, for example, from a field callback function.

Syntax

FormGetData(*hForm*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormCurr](#)

Example

```
! Field callback to save data.
FUNCTION Save()
    INT hForm,hField;
    FormCurr(hForm,hField);
    FormGetData(hForm);
    ! Access all data.
    ..
END
```

See Also

[Form Functions](#)

FormGetInst

Extracts the data associated with a field (set by the FormSetInst() function). You would normally use this function in a field callback function. It allows single callback functions to know that the form and field are associated.

Syntax

FormGetInst(*hForm*, *hField*, *iData*, *sData*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

iData:

Integer data.

sData:

String data.

Return Value

The data (as a string).

Related Functions

[FormSetInst](#), [FormCurr](#), [FormGetCurrInst](#)

Example

```

INT
FUNCTION GetNextRec()
    INT hDev,hForm,hField;
    STRING Str;
    ! Get field data, for example, the hDev value.
    ..
    FormCurr(hForm,hField);
    FormGetInst(hForm,hField,hDev,Str);
    DevNext(hDev);
    ! Display new record in form.
    ..
    RETURN 0;
END

```

See Also

[Form Functions](#)

FormGetText

Gets the current text from a form field. You should call this function only while the form is displayed; for example,, from a field callback function.

Syntax

FormGetText(*hForm*, *hField*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

Return Value

The field text (as a string).

Related Functions

[FormSetText](#)

Example

```
FUNCTION Search()
    INT hForm,hField;
    STRING Recipe;
    FormCurr (hForm,hField);
    Recipe=FormGetText (hForm,hField);
    ! Go and find recipe.
    ..
END
```

See Also

[Form Functions](#)

FormGoto

Goes to a specified form. The form is displayed on top of all windows and the keyboard focus is set to this form.

Syntax

FormGoto(*hForm*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormNew](#)

Example

```
FormGoto (hForm) ;
```

See Also

[Form Functions](#)

FormGroupBox

Adds a group box to the current form. A group box is a form control box drawn to the specified size. If the box contains radio buttons, they are grouped together. You should call this function only after the FormNew() function and before the FormRead() function.

The group box is added to the form at the specified column and row position with the specified width and height. Use the FormRadioButton() function to add the radio buttons to the box, and call this function between each group of radio buttons.

Syntax

FormGroupBox(*Col, Row, Width, Height [, Text]*)

Col:

The number of the column in which the group box will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the group box in column 8, enter 7.

Row:

The number of the row in which the group box will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the group box in row 6, enter 5.

Width:

The width of the group box, which should be wide enough to display your widest item.

Height:

The height of the group box.

Text:

The text to display as the group box label.

Return Value

The field handle if the group box is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#), [FormRadioButton](#)

Example

```
! Create a form, add to radio buttons groups and then display the
form
! The operator may select one of the radio buttons from each group
FUNCTION FnMenu()
    STRING sFast, sSlow, sMedium;
    STRING sNorth, sSouth, sEast, sWest;
    FormNew("Select Item",40,7,1);
    FormGroupBox(1,1,15,5,"Speed");
    FormRadioButton(2,2,"Fast", sFast);
    FormRadioButton(2,3,"Medium", sMedium);
    FormRadioButton(2,4,"Slow", sSlow);
    FormGroupBox(19,2,15,6,"Direction");
    FormRadioButton(20,2,"North", sNorth);
    FormRadioButton(20,3,"South", sSouth);
    FormRadioButton(20,4,"East", sEast);
    FormRadioButton(20,5,"West", sWest);
    FormRead(0);
END
```

See Also

[Form Functions](#)

FormInput

Adds a prompt and edit field to the current form. You should call this function only after the FormNew() function and before the FormRead() function. When FormRead() is called, the form will display with the prompt and edit box. The operator's input is passed back as a string (*Text*).

Syntax

FormInput(*Col,Row,Prompt,Text,Width [, maxTextLength]*)

Col:

The number of the column in which the prompt will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the prompt in column 8, enter 7.

Row:

The number of the row in which the prompt will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the prompt in row 6, enter 5.

Prompt:

The prompt string.

Text:

The output text string containing the operator's input.

Note: Only cicode variables can be used in output parameters, variable or local tags cannot be used and will result in a compiler error if attempted.

Width:

The width of the edit field.

maxTextLength:

This optional parameter specifies the maximum length of input text. The default value is 0 meaning the string can have the maximum length allowed in the system (Cicode allows strings of 255 characters).

Return Value

The field handle if it is added successfully, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#)

Example

```
FormInput(1,2,"Recipe",Recipe,20);
```

See Also

[Form Functions](#)

FormListAddText

Adds a new text entry to a combo box or a list box while the form is displayed. It only adds the text to the list - it does not select it. Use the FormListSelectText() function to select (highlight) an entry. Call this function only when the form is displayed, for example, from a field callback function.

Syntax

FormListAddText(*hForm*, *hField*, *Text*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

Text:

The output text string containing the operator's input.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormListSelectText](#), [FormListDeleteText](#), [FormSetText](#)

Example

```
/* create a form with a list */
hForm = FormNew("Ingredients", 40, 10, 1);
hField = FormListBox(2,2,20,5,sBuf);
FormAddList("Flour");
FormAddList("Water");
FormAddList("Salt");
FormAddList("Sugar");
/* Display the form */
FormRead(1);
..
/*Add Milk to list */
FormListAddText(hForm, hField, "Milk");
..
```

See Also

[Form Functions](#)

FormListBox

Adds a list box to the current form. The list box is a form control that allows the operator to select from a list of items. You should call this function only after the FormNew() function and before the FormRead() function.

The list box is added to the form at the specified column and row position with the specified width and height. If more items are placed in the list than the list can display, a scroll bar displays for scrolling to the hidden items.

Use the FormAddList() function to add items for display in the list box. If the form is already displayed, you can use the FormListAddText() and FormListSelectText() functions to add (and highlight) text in the list box.

Syntax

FormListBox(*Col*, *Row*, *Width*, *Height*, *sBuf* [, *Mode*])

Col:

The number of the column in which the list box will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the list box in column 8, enter 7.

Row:

The number of the row in which the list box will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the list box in row 6, enter 5.

Width:

The width of the list box, in characters. Width should be wide enough to display your widest item. Items wider than the list box are clipped.

Height:

The height of the list box, as the number of items that can be seen in the list box without scrolling.

sBuf:

The string buffer in which to store the selected item.

Mode:

The mode in which to create the list box:

0 - Sort the list box elements alphabetically.

1 - Place elements in list box in the order they were added.

Mode 0 is the default.

Return Value

The field handle if the list box is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#), [FormAddList](#), [FormListAddText](#), [FormListSelectText](#), [Form-ComboBox](#)

Example

```
! Create a form, add list box and then display the form.  
! The operator may select one of the items from the list.  
STRING sBuf;  
FUNCTION FnMenu()  
    FormNew("Select Item",20,6,1);  
    FormListBox(2,2,15,5,sBuf,1);  
    FormAddList("Item One");  
    FormAddList("Item two");  
    FormAddList("Item three");  
    FormButton(0,0,"OK",0,1);  
    FormButton(5,0,"CANCEL",0,2);  
    FormRead(0);  
    SELECTION= sBuf;  
END
```

See Also

[Form Functions](#)

FormListDeleteText

Deletes an existing text entry from a combo box or a list box while the form is displayed. It only deletes the text from the list - it does not change the selection. Call this function only when the form is displayed, for example, from a field callback function.

Syntax

FormListDeleteText(*hForm*, *hField*, *Text*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

Text:

The text to delete.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormListSelectText](#), [FormListAddText](#)

Example

```
/* create a form with a list */
hForm = FormNew("Ingredients", 40, 10, 1);
hField = FormListBox(2,2,20,5,sBuf);
FormAddList("Flour");
FormAddList("Water");
FormAddList("Salt");
FormAddList("Sugar");
/* Display the form */
FormRead(1);
..
/*Remove Sugar from the list */
FormListDeleteText(hForm, hField, "Sugar");
..
```

See Also

[Form Functions](#)

FormListSelectText

Selects (highlights) a text entry in a Combo box or a List box while the form is displayed. The text to be selected needs to exist in the list. (Use the FormListAddText() function to add a text entry to a list.) Call this function only when the form is displayed, for example, from a field callback function.

Syntax

FormListSelectText(*hForm*, *hField*, *Text*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

Text:

The text to be selected. If this text is not present in the list, then no item will be selected (and this text will not be added).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormListAddText](#), [FormSetText](#)

Example

```
/* Create a form with a list */
hForm = FormNew("Ingredients", 40, 10, 1);
hField = FormListBox(2,2,20,5,sBuf);
FormAddList("Flour");
FormAddList("Water");
FormAddList("Salt");
FormAddList("Sugar");
/* Display the form */
FormRead(1);
..
/*Select Flour */
FormListSelectText(hForm, hField, "Flour");
```

See Also

[Form Functions](#)

FormNew

Creates a new data entry form and defines its size and mode. After the form is created, you can add fields, and then display the form.

Before you can display a form on the screen, you need to call this function to set the size and mode of the form, and then call the various form field functions, FormInput(), FormButton(), FormEdit() etc to add user input fields to the form. To display the form on the screen (to allow the user to enter data) call the FormRead() function.

Syntax

FormNew(*Title*, *Width*, *Height*, *Mode*)

Title:

The title of the form.

Width:

The character width of the form (1 to 131).

Height:

The character height of the form (1 to 131).

Mode:

The mode of the form:

0 - Default font and text spacing

1 - Small font

2 - Fixed pitch font

4 - Static text compression where the vertical spacing is reduced. This can cause formatting errors if buttons are too close, because the vertical spacing will be less than the height of a button.

8 - Keep the form on top of the CitectSCADA window.

16 - The current window cannot be changed or closed until the form is finished or cancelled.

32 - Makes a form with no caption.

128 - The form will not close if the **ESC** or **ENTER** key is pressed, unless you specifically define at least one button on the form which acts as an **OK** or **Cancel** button. For a form with no buttons, the **ENTER** key normally closes the form; this mode disables that behavior.

256 – Makes a from with no system-menu (mostly appears as a single close button X) .

Multiple modes can be selected by adding them (for example, to use Modes 4 and 2, specify Mode 6).

Return Value

The form handle if the form is created successfully, otherwise -1 is returned. The form handle identifies the table where all data on the associated form is stored.

Related Functions

[FormDestroy](#), [FormInput](#), [FormButton](#), [FormEdit](#), [FormRead](#)

Example

```
FormNew("Recipe",30,5,0);
FormInput(1,1,"Recipe No",Recipe,20);
FormInput(1,2,"Amount",Amount,10);
FormRead(0);
```

See Also

[Form Functions](#)

FormNumPad

Provides a keypad form for the operator to add numeric values. You can customize the standard form as a mathematical keypad, with the +, -, and / operators and the decimal point. For a time keypad, use the AM, PM, and : (hour/minute divider) buttons. You can also include a password edit field.

Syntax

FormNumPad(*Title*, *Input*, *Mode*)

Title:

The title to display on the number pad form.

Input:

The existing or default value. This value is returned if the form is cancelled or accepted without changes.

Mode:

The buttons to include on the keypad form. The Mode can be a combination of the following:

- 0 - Standard keypad
- 1 - Password edit field
- 2 - not used
- 4 - With +/- button
- 8 - With / button
- 16 - With . button
- 32 - With : button
- 64 - With AM, PM buttons

Multiple modes can be selected by adding them. For example, to include +/- buttons and a . button, specify Mode 20 (16+4).

Return Value

The string value entered by the operator. The IsError() function returns 0 (zero). If the form was cancelled, the value of *Input* is returned, and the IsError() function returns [error](#) number 299.

Example

```
/* Set defaults first, then four keypad forms to adjust recipe. */
Qty_Flour=FormNumPad("Add Flour", Qty_Flour, 17);
Qty_Water=FormNumPad("Add Water", Qty_Water, 17);
Qty_Salt=FormNumPad("Add Salt", Qty_Salt, 17);
Qty_Sugar=FormNumPad("Add Sugar", Qty_Sugar, 17);
```

See Also

[Form Functions](#)

FormOpenFile

Displays a File Open dialog box.

Syntax

FormOpenFile(*sTitle*, *sFileName*, *sFilter*)

sFileName:

The name of the default file to display in the "File Name" field.

sTitle:

A title to display at the top of the form.

sFilter:

A file filter list to display in the "List Files of Type" field. The file filter list has the following format:

<File Type>|<Filter>|

where:

File Type is the text that displays in the drop-down box, for example All Files (*.*). *Filter* is the file type, for example *.CI

Return Value

The name and full path of the selected file (as a string) or an empty string ("") if the Cancel button is selected.

Related Functions

[FormSaveAsFile](#), [FormSelectPrinter](#)

Example

```
// Display the Open File dialog with the following filter list:  
// All Files (*.*)  
// Exe Files (*.EXE)  
// Cicode Files (*.CI)  
sFilename = FormOpenFile("Open", "*.CI", "All Files (*.*)|*.|Exe  
Files (*.EXE)|*.EXE|Cicode Files (*.CI)|*.CI|");
```

See Also

[Form Functions](#)

FormPassword

Adds both a password prompt and edit field to the current form. You should call this function only after the FormNew() function and before the FormRead() function. When FormRead() is called, the form will also display the password prompt and edit field.

The operator's input is not echoed in the field; a single asterisk (*) is displayed for each character.

Syntax

FormPassword(*Col*, *Row*, *Prompt*, *Password*, *Width*)

Col:

The number of the column in which the prompt will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the prompt in column 8, enter 7.

Row:

The number of the row in which the prompt will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the prompt in row 6, enter 5.

Prompt:

The prompt string.

Password:

The password entered by the operator.

Width:

The width of the edit field.

Return Value

The field handle if it is added successfully, otherwise -1 is returned.

Related Functions

[FormEdit](#)

Example

```
! Add Password input.  
FormPassword(1,2,"Enter Password",Password,10);
```

See Also

[Form Functions](#)

FormPosition

Sets the position of a form on the screen, before it is displayed. You should call this function only after the FormNew() function and before the FormRead() function.

Syntax

FormPosition(X, Y, Mode)

X, Y:

The x and y pixel coordinates of the form.

Mode:

Not used, set it to 0.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormNew](#), [FormRead](#)

Example

```
hForm = FormNew("title", 20, 5, 0);  
! display form at x=100, y=50  
FormPosition(100, 50, 0);
```

See Also

[Form Functions](#)

FormPrompt

Adds a prompt field to the current form. You should call this function only after the FormNew() function and before the FormRead() function.

Syntax

FormPrompt(*Col*, *Row*, *Prompt*)

Col:

The number of the column in which the prompt will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the prompt in column 8, enter 7.

Row:

The number of the row in which the prompt will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the prompt in row 6, enter 5.

Prompt:

The prompt string.

Return Value

The field handle if it is added successfully, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#)

Example

```
FormPrompt(1,2,"Enter Recipe");
```

See Also

[Form Functions](#)

FormRadioButton

Adds a radio button to the current form, allowing the operator to make a selection from a multiple choice list. You should call this function only after the FormNew() function and before the FormRead() function.

The radio button is added to the form at the specified column and row position. The width of the button will be sized to suit the text.

By default, all radio buttons are placed into the one group. If you require separate groups, use this function in conjunction with the FormGroupBox() function.

Syntax

FormRadioButton(*Col*, *Row*, *sText*, *sBuf*)

Col:

The number of the column in which the button will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the button in column 8, enter 7.

Row:

The number of the row in which the button will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the button in row 6, enter 5.

sText:

The text associated with the radio button.

sBuf:

The string buffer in which to put the state of the radio button. You should initialize this buffer to the state of the button. When the form returns, this buffer will contain either '1' or '0' if the radio button is checked.

Return Value

The field handle if the radio button is successfully added, otherwise -1 is returned.

Related Functions

[FormNew](#), [FormRead](#), [FormGroupBox](#), [FormCheckBox](#)

Example

```

! Create a form, add radio buttons and then display the form.
! The operator may only select one radio button , either Fast,
Medium or Slow
FUNCTION FnMenu()
    STRING sFast, sSlow, sMedium;
    sFast = "1";
    sMedium = "0";
    sSlow = "0";
    FormNew("Speed",20,6,1);
    FormRadioButton(2,2,"Fast", sFast);
    FormRadioButton(2,3,"Medium", sMedium);
    FormRadioButton(2,4,"Slow", sSlow);
    FormRead(0);
    If sFast = "1" THEN
        ! do fast stuff
    ELSE
        IF sMedium = "1" THEN
            ! do Medium stuff

```

```
ELSE
IF sSlow = "1" THEN
    ! do slow stuff
END
END
```

See Also

[Form Functions](#)

FormRead

Displays the current form (created with the FormNew() function), with all the fields that were added (with the form field functions).

You can display the form and wait for the user to finish entering data by setting the *Mode* to 0. This mode is the most commonly used, with [OK] and [Cancel] buttons to either save or discard operator entries and to close the form.

To display the form and return before the user has finished, use Mode 1. This mode is used to animate the data on the form or to perform more complex operations.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

FormRead(*Mode*)

Mode:

Mode of the form:

0 - Wait for the user.

1 - Do not wait for the user.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormNew](#)

Example

```
! Display the form and wait for the user.
FormRead(0);
! Display the form and do not wait for the user.
```

```

FormRead(1);
! While the form is displayed, update the time every second.
WHILE FormActive(hForm) DO
    FormSetText(hForm,hField,Time());
    Sleep(1);
END

```

See Also[Form Functions](#)**FormSaveAsFile**

Displays a File Save As dialog box.

Syntax

FormSaveAsFile(*sTitle*, *sFileName*, *sFilter* [, *sDefExt*])

sTitle:

A title to display at the top of the form.

sFileName:

The name of the default file to display in the "File Name" field.

sFilter:

A file filter list to display in the "List Files of Type" field. The file filter list has the following format:

<File Type>|<Filter>|

where:

File Type is the text that displays in the drop-down box, for example All Files (*.*). *Filter* is the file type, for example *.CI

sDefExt:

The file extension that will be used as a default when you use the FormSaveAsFile() function. If you do not specify a default extension, files will be saved without an extension.

Return Value

The name and full path of the selected file (as a string) or an empty string ("") if the Cancel button is selected.

Related Functions[FormOpenFile](#), [FormSelectPrinter](#)

Example

```
// Display the SaveAs dialog with the following filter list:  
// All Files (*.*)  
// Exe Files (*.EXE)  
// Cicode Files (*.CI)  
sFilename = FormSaveAsFile("Save As", "Alarms", "All Files  
(*.*)|*.Exe Files (*.EXE)|*.EXE|Cicode Files (*.CI)|*.CI|",  
"ci");
```

See Also

[Form Functions](#)

FormSecurePassword

Adds both a password prompt and edit field to the current form. You should call this function only after the FormNew() function and before the FormRead() function. When FormRead() is called, the form will also display the password prompt and edit field.

The operator's input is not echoed in the field; a single asterisk (*) is displayed for each character. The function does not return the password as a plain text, it returns an encrypted password string. The user can send this string to the UserLogin or UserVerify functions.

Syntax

FormSecurePassword(*Col*, *Row*, *Prompt*, *Password*, *Width*)

Col:

The number of the column in which the prompt will be placed. Enter a number from 0 (column 1) to the form width - 1. For example, to place the prompt in column 8, enter 7.

Row:

The number of the row in which the prompt will be placed. Enter a number from 0 (row 1) to the form height - 1. For example, to place the prompt in row 6, enter 5.

Prompt:

The prompt string.

Password:

The encrypted password entered by the operator.

Width:

The width of the edit field.

Return Value

The field handle if it is added successfully, otherwise -1 is returned.

Related Functions

[FormEdit](#), [UserLogin](#), [UserVerify](#)

Example

```
! Add Password input.  
FormSecurePassword(1,2,"Enter Password",Password,10);
```

See Also

[Form Functions](#)

FormSelectPrinter

Displays the Select Printer dialog box.

Syntax

FormSelectPrinter()

Return Value

The name of the selected printer (as a string) or an empty string ("") if the Cancel button is selected.

Related Functions

[FormOpenFile](#), [FormSaveAsFile](#)

Example

```
// Display the Select Printer dialog  
sPrinter = FormSelectPrinter();
```

See Also

[Form Functions](#)

FormSetData

Sets all the edit data from the string buffers into the form. You should call this function only while the form is active.

Syntax

FormSetData(*hForm*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormGetData](#)

Example

```
INT
FUNCTION MyNextRec()
    INT hForm,hField;
    FormCurr (hForm,hField);
    FormSetData (hForm);
    RETURN 0;
END
```

See Also

[Form Functions](#)

FormSetInst

Associates an integer and string value with each field on a form. This data could then be used by a callback function. You can use a single callback function for all fields, and use the data to perform different operations for each field.

Syntax

FormSetInst(*hForm*, *hField*, *iData*, *sData*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected.

iData:

Integer data.

sData:

String data.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormGetInst](#)

Example

```
! Open recipe database.
hDev=DevOpen("Recipe", 0);
hForm=FormNew("Recipe",20,5,0);
hField=FormButton(5,2,"Next",GetNextRec,0);
FormSetInst(hForm,hField,hDev,"");
/* The device handle hDev is put into the next button , so when the
button is selected it can get hDev and get the next record. */
```

See Also

[Form Functions](#)

FormSetText

Sets new field text on a field. This function allows you to change field text while the form is displayed. Call this function only when the form is displayed, for example, from a field callback function.

If you are using this function on a Combo box or a List box, it will select the text from the Combo box or List box list. If no text exists in the Combo box or List box list, the function will add it.

Syntax

FormSetText(*hForm*, *hField*, *Text*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where all data on the associated form is stored.

hField:

The field handle of the field currently selected. If the hField is a handle to the secure edit field created with FormSecurePassword, the text in the secure edit field will not be changed. However, when an empty string is passed to FormSetText(), the contents of the secure edit field will be cleared.

Text:

New field text.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FormCurr](#), [FormListSelectText](#), [FormListAddText](#)

Example

```
/* Create a form with a field */
hForm = FormNew("Ingredients", 40, 10, 1);
hField = FormPrompt(2,2,"Motor1:");
/* Display the form*/
FormRead(1);
..
/* Change the text in the field */
FormSetText(hForm, hField, "Pump1:");
..
```

See Also

[Form Functions](#)

FormWndHnd

Gets the window handle for the given form. The window handle may be used by 'C' programs and CitectSCADAWindow... functions. You should call this function only after the FormRead() function.

The window handle is not the same as the CitectSCADA window number and cannot be used with functions that expect the CitectSCADA window number (the Win... functions).

Syntax

FormWndHnd(*hForm*)

hForm:

The form handle, returned from the FormNew() function. The form handle identifies the table where data on the associated form is stored.

Return Value

The window handle if successful, otherwise a 0 is returned.

Related Functions

[FormNew](#), [FormRead](#), [WndFind](#)

Example

```
/* Create a form with a field */
hForm = FormNew("Ingredients", 40, 10, 1);
hField = FormPrompt(2,2,"Motor1:");
/* Display the form*/
FormRead(1);
/* Get the form's window number for future reference */
hWnd      = FormWndHnd(hForm);
```

See Also

[Form Functions](#)

Chapter: 31 Format Functions

Format functions convert data into formatted strings. You can convert many different items of data into single, formatted strings that can then be displayed, printed, or written to a file. The format functions also convert (formatted) data back into individual elements; for example,, strings that are read from files or other devices.

Format Functions

Following are functions used for formatting data:

FmtClose	Closes a format template.
FmtFieldHnd	Gets the handle of a field in a format template.
FmtGetField	Gets field data from a format template.
FmtGetFieldCount	Retrieves the number of fields in a format object.
FmtGetFieldHnd	Gets field data from a format template using a field handle.
FmtGetFieldName	Retrieves the name of a particular field in a format object.
FmtGetWidth	Retrieves the width of a particular field in a format object.
FmtOpen	Creates a new format template.
FmtSetField	Sets data in a field of a format template.
FmtSetFieldHnd	Sets data in a field of a format template using a field handle.
FmtToStr	Converts format template data to a string

See Also

[Functions Reference](#)

FmtClose

Closes a format template. After it is closed, the template cannot be used. Closing the template releases system memory.

Syntax

FmtClose(*hFmt*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FmtOpen](#)

Example

```
FmtClose(hFmt);
```

See Also

[Format Functions](#)

FmtFieldHnd

Gets the handle of a field in a format template. You can then use the field handle in the FmtGetFieldHnd() and FmtSetFieldHnd() functions. By using a handle, you only need to resolve the field name once and then call other functions as required (resulting in improved performance.)

Syntax

FmtFieldHnd(*hFmt*, *Name*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

Name:

The field name.

Return Value

The handle of the format template field, or -1 if the field cannot be found.

Related Functions

[FmtGetFieldHnd](#), [FmtSetFieldHnd](#)

Example

```
!Resolve names at startup.
hName=FmtFieldHnd(hFmt, "Name");
hDesc=FmtFieldHnd(hFmt, "Desc");
!Set field data.
FmtSetFieldHnd(hFmt, hName, "CV101");
```

See Also

[Format Functions](#)

FmtGetField

Gets field data from a format template. Use this function to extract data after a call to [StrToFmt\(\)](#).

Syntax

FmtGetField(*hFmt*, *Name*)

hFmt:

The format template handle, returned from the [FmtOpen\(\)](#) function. The handle identifies the table where all data on the associated format template is stored.

Name:

The field name.

Return Value

The data (as a string). If the field does not contain any data, an empty string will be returned.

Related Functions

[StrToFmt](#), [FmtSetField](#), [FmtToStr](#)

Example

```
StrToFmt (hFmt, "CV101 Raw Coal Conveyor");
Str=FmtGetField(hFmt, "Name");
! Str will contain "CV101".
```

See Also

[Format Functions](#)

FmtGetFieldCount

Retrieves the number of fields in a format object.

Syntax

FmtGetFieldCount(*hFmt*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where data on the associated format template is stored.

Return Value

Number of fields in the specified format.

Related Functions

[FmtGetField](#), [FmtOpen](#)

See Also

[Format Functions](#)

FmtGetFieldHnd

Gets field data from a format template. Use this function to extract data after a call to StrToFmt(). This function has the same effect as FmtGetField(), except that you use a field handle instead of the field name.

Syntax

FmtGetFieldHnd(*hFmt*, *hField*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

hField:

The field handle.

Return Value

The data (as a string). If the field does not contain any data, an empty string will be returned.

Related Functions

[StrToFmt](#), [FmtFieldHnd](#)

Example

```
StrToFmt(hFmt,"CV101 Raw Coal Conveyor");
hField=FmtFieldHnd(hFmt,"Name");
Str=FmtGetField(hFmt,hField);
! Str will contain "CV101".
```

See Also

[Format Functions](#)

FmtGetFieldName

Retrieves the name of a particular field in a format object.

Syntax

FmtGetFieldName(*hFmt*, *hField*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where data on the associated format template is stored.

hField:

The field handle.

Return Value

Name of requested field

Related Functions

[FmtGetField](#), [FmtOpen](#)

See Also

[Format Functions](#)

FmtGetWidth

Retrieves the width of a particular field in a format object.

Syntax

FmtGetWidth(hFmt, hField)

hFmt:

The handle to a format object. The handle identifies the table where data on the associated format template is stored.

hField:

The field handle.

Return Value

Width of the requested field.

Related Functions

[FmtGetField](#), [FmtGetFieldName](#), [FmtOpen](#)

See Also

[Format Functions](#)

FmtOpen

Creates a format template. After you create a template, you can use it for formatting data into strings or extracting data from a string. To format a template, use the same syntax as a device format, that is {<name>[,width[,justification]]}.

Syntax

FmtOpen(Name, Format, Mode)

Name:

The name of the format template or Alarm Category.

Format:

The format of the template, as {<name>[,width[,justification]]}. Not used for alarm format. See [Format Templates](#) for more information.

Mode:

The mode of the open:

0 - Open the existing format.

1 - Open a new format.

2 - Open Summary Format from Alarm Category specified by Name.

3 - Open Display Format from Alarm Category specified by Name.

Return Value

The format template handle, or -1 if the format cannot be created.

Related Functions

[FmtClose](#)

Examples

```

hFmt=FmtOpen("MyFormat","{Name}{Desc,20}",0);
FmtSetField(hFmt,"Name", "CV101");
FmtSetField(hFmt,"Desc","Raw Coal Conveyor");
Str =FmtToStr(hFmt);
! Str will contain "CV101 Raw Coal Conveyor".
FmtOpen("0", "", 2);
! Display Format from Alarm Category 0
FmtOpen("0", "", 3);
! Summary Format from Alarm Category 0.

```

See Also

[Format Functions](#)

FmtSetField

Sets data in a field of a format template. After you have set all the fields, you can build the formatted string with the FmtToStr() function.

Syntax

FmtSetField(*hFmt*, *Name*, *Data*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

Name:

The name of the format template.

Data:

Field data.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FmtGetField](#), [FmtToStr](#)

Example

```
hFmt=FmtOpen("MyFormat", "{Name}{Desc, 20}", 0);
FmtSetField(hFmt, "Name", "CV101");
FmtSetField(hFmt, "Desc", "Raw Coal Conveyor");
Str =FmtToStr(hFmt);
! Str will contain "CV101 Raw Coal Conveyor".
```

See Also

[Format Functions](#)

FmtSetFieldHnd

The fields, you can build the formatted string with the FmtToStr() function. This function has the same effect as FmtSetField() except that you use a field handle instead of the field name.

Syntax

FmtSetFieldHnd(*hFmt*, *hField*, *Data*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

hField:

The field handle.

Data:

Field data.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FmtFieldHnd](#), [FmtToStr](#), [FmtSetField](#)

Example

```
hField=FmtFieldHnd(hFmt, "Name");
FmtSetFieldHnd(hFmt, hField, "CV101");
```

See Also

[Format Functions](#)

FmtToStr

Builds a formatted string from the current field data (in a format template).

Syntax

FmtToStr(*hFmt*)

hFmt:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

Return Value

The formatted string as defined in the format description.

Related Functions

[StrToFmt](#)

Example

```
! Get the formatted string.
Str=FmtToStr(hFmt);
```

See Also

[Format Functions](#)

Chapter: 32 FTP Functions

FTP functions are used to manage your FTP communications and files (used when running your project over the Internet). These functions can only be used on the Internet Display Client.

FTP Functions

Following are functions relating to FTP:

FTPClose	Closes an FTP session.
FTPFileCopy	Copies a file from the FTP server to the Internet Display Client.
FTPFileFind	Finds a file on the FTP server that matches a specified search criteria.
FTPFile-FindClose	Closes a find (started with FTPFileFind) that did not run to completion.
FTPOpen	Connects to an FTP server

See Also

[Functions Reference](#)

FTPClose

Closes an FTP session. This function can only be used on the Internet Display Client.

Syntax

FTPClose(*hndFTP*)

hndFTP:

The handle of a valid FTP session, as returned by [FTPOpen\(\)](#).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FTPOpen](#)

Example

```
INT hFtp;
hFtp = FtpOpen("", "", "");
..
FtpClose(hFtp);
```

See Also

[FTP Functions](#)

FTPFileCopy

Copies a file from the FTP server to the Internet Display Client. Before calling this function, you need to call FtpOpen(). This function can only be used on the Internet Display Client.

Syntax

FTPFileCopy(*hndFTP*, *sSrcPath*, *sDestPath*)

hndFTP:

The handle of a valid FTP session, as returned by FTPOpen().

sSrcPath:

The file name and path of the file to be copied from the FTP Server to the Internet Display Client.
This can be any FTP server.

sDestPath:

The destination of the copied file (including the name of the file).

Return Values

0 (zero) if successful, otherwise an [error](#) is returned.

Note: The `[Internet]ZipFiles` parameter does not apply to files copied to the Internet Display Client using this function.

Related Functions

[FTPOpen](#), [FTPFileFind](#)

See Also

[FTP Functions](#)

FTPFileFind

Finds a file on the FTP server that matches a specified search criteria. Before you can call this function, you need to call FTPOpen(). This function can only be used on the Internet Display Client.

To find a list of files, you need to first call this function twice: once to find the first file, then again with an empty path to find the remaining files. After the last file is found, an empty string is returned.

If the search is for multiple files, FTPFileFindClose needs to be called if the search does not run to completion (for example, you do not run until an empty string is returned).

Syntax

FTPFileFind(*hndFTP*, *sPath*)

hndFTP:

The handle of a valid FTP session, as returned by FTPOpen().

sPath:

The path you want to search for the desired file. Do not use path substitution here. To search for multiple files, the wildcards * and ? may be used to match multiple entries.

nMode:

The type of file to check:

- 0 - Normal files (includes files with read-only and archived attributes)
- 1 - Read-only files only
- 2 - Hidden files only
- 4 - System files only
- 16 - Subdirectories only
- 32 - Archived files only
- 128 - Files with no attributes only

These numbers can be added together to search for multiple types of files during one search.

Return Value

The full path and filename. If no files are found, an empty string is returned.

Related Functions

[FTPFileCopy](#), [FTPOpen](#)

Example

```
INT hFtp;
STRING sFindPath;
STRING sPath;
sFindPath = "\User\Example\*.RDB";
hFtp = FtpOpen("", "", "");
sPath = FtpFileFind(hFtp, sFindPath);
WHILE StrLength(sPath) > +0 DO
    sPath = FtpFileFind(hFtp, "");
END
FtpClose(hFtp);
```

See Also

[FTP Functions](#)

FTPFileFindClose

Closes a find (started with `FTPFileFind`) that did not run to completion. This function can only be used on the Internet Display Client.

Syntax

FTPFileFindClose(*hndFTP*)

hndFTP

The handle of a valid FTP session, as returned by `FTPOpen()`.

Return Value

0 if no error is detected, or a Cicode error code if an error occurred.

Related Functions

[FTPFileFind](#)

Example

```
//Find the first DBF file starting with fred
sPath = FileFind("\User\Example\FRED*.DBF", 0);
IF (StrLength(sPath) > 0) THEN
    //Do work here
    FileFindClose();
```

END

See Also

[FTP Functions](#)

FTPOpen

Connects to an FTP server. This function can only be used on the Internet Display Client.

Syntax

FTPOpen([sIPAddress] [, sUsername] [, sPassword])

sIPAddress:

The TCP/IP address of the FTP server you wish to connect to (for example, 10.5.6.7 or plant.yourdomain.com). If you do not specify an IP address, the CitectSCADA FTP server running on the Internet Server you are connected to will be used.

sUsername:

The FTP login username. If you omit both the username and IP address, the CitectSCADA FTP password will be used. If you omit just the username, an anonymous logon will occur.

sPassword:

The FTP server password. If you wish to log on anonymously or you wish to log on to the CitectSCADA FTP server, do not specify a password, here.

Return Value

A handle to the FTP server otherwise -1 if an [error](#) occurs (for example, the server cannot be found).

Related Functions

[FTPclose](#)

See Also

[FTP Functions](#)

Chapter: 33 FuzzyTech Functions

The CitectSCADA FuzzyTech functions support fuzzy logic control and provide an interface to the FuzzyTech functions provided by INFORM Software Corporation. To use these functions you need to purchase the development environment from INFORM - the makers of FuzzyTech.

FuzzyTech Functions

Following are functions relating to fuzzy logic control:

FuzzyClose	Closes specified fuzzy instance.
FuzzyGetCodeValue	Gets a specified Code variable from the specified instance.
FuzzyGetShellValue	Gets a specified Shell variable from the specified instance.
FuzzyOpen	Creates a new fuzzy instance.
FuzzySetCodeValue	Sets a specified Code variable in the specified instance.
FuzzySetShellValue	Sets a specified Shell variable in the specified instance.
FuzzyTrace	Controls the tracing.

See Also

[Functions Reference](#)

FuzzyClose

Frees all memory and information for the specified instance. After the fuzzy instance is closed, the handle given in the *hFuzzy* parameter is no longer valid.

Syntax

FuzzyClose(*hFuzzy*)

hFuzzy:

The fuzzy instance handle (and integer greater than 0).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FuzzyOpen](#)

Example

See [FuzzyOpen](#).

See Also

[FuzzyTech Functions](#)

FuzzyGetCodeValue

Gets a value for the specified input of the specified instance.

Syntax

FuzzyGetCodeValue(*hFuzzy*, *iIOIndex*, *NoHitFlag*)

hFuzzy:

The fuzzy instance handle (and integer greater than 0).

iIOIndex:

Specifies the variable to receive the value. The I/O-Indices start with 0 and increment by 1 for each variable. To find the correct index for a specific variable, the variables need to be sorted in alpha-numerical order, first the inputs and then the outputs.

NoHitFlag:

Variable to receive the new value of the No-hit-flag. The No- hit-flag is TRUE if no rule was active for the variable specified by *iIOIndex*, otherwise it is FALSE. This needs to be a Cicode variable of INT type - it cannot be a constant or PLC variable tag.

Return Value

The code value if the function was successful, otherwise -1. Use `IsError()` to find the [error](#) number if the function does not succeed.

Related Functions

[FuzzyOpen](#), [FuzzySetCodeValue](#)

Example

See [FuzzyOpen](#)

See Also

[FuzzyTech Functions](#)

FuzzyGetShellValue

Gets a value for the specified input of the specified instance. The variables in the instance needs to have the data type REAL (floating point values).

Syntax

FuzzyGetShellValue(*hFuzzy*, *iIOIndex*, *NoHitFlag*)

hFuzzy:

The fuzzy instance handle (and integer greater than 0).

iIOIndex:

Specifies the variable to receive the value. The I/O-Indices start with 0 and increment by 1 for each variable. To find the correct index for a specific variable, the variables need to be sorted in alpha-numerical order, first the inputs and then the outputs.

NoHitFlag:

Variable to receive the new value of the No-hit-flag. The No- hit-flag is TRUE if no rule was active for the variable specified by iIOIndex, otherwise it is FALSE. This needs to be a Cicode variable of INT type - it cannot be a constant or PLC variable tag.

Return Value

The shell value if the function was successful. Use IsError() to find the [error](#) number if the function does not succeed.

Related Functions

[FuzzyOpen](#), [FuzzySetShellValue](#)

Example

See [FuzzyOpen](#)

See Also

[FuzzyTech Functions](#)

FuzzyOpen

This function loads a *.FTR file, allocates memory and creates a handle for this fuzzy instance. To use the FuzzyTech functions you need to be a registered user of one or more of the following fuzzyTech products: *fuzzyTECH* Online Edition, *fuzzyTECH* Precompiler Edition, or *fuzzyTECH* for Business PlusC. And you need to only use *fuzzyTECH* to generate the *.FTR file for FTRUN.

The application needs to call the FuzzyClose function to delete each fuzzy instance handle returned by the FuzzyOpen function.

Syntax

FuzzyOpen(*sFile*)

sFile:

Specifies the filename of the .FTR file to load.

Return Value

The handle to the fuzzy instance, or -1 if the function cannot complete the operation. Use IsError() to find the [error](#) number.

Related Functions

[FuzzyClose](#), [FuzzyGetShellValue](#), [FuzzySetShellValue](#), [FuzzyGetCodeValue](#), [FuzzySetCodeValue](#), [FuzzyTrace](#).

Example

```
INT hFuzzy;
INT NoHitFlag;
INT Status;
REAL MemOutput;
// open the Fuzzy Tech runtime instance
hFuzzy = FuzzyOpen
("C:\Program Files\Citect\CitectSCADA 7.10\bin\traffic.ftr");
Status = IsError();
IF hFuzzy <> -1 THEN
    MemOutput = PLCOutput;
    WHILE Status = 0 DO
        FuzzySetShellValue(hFuzzy, 0, 42.0);
        FuzzySetShellValue(hFuzzy, 1, 3.14150);
        MemOutput = FuzzyGetShellValue(hFuzzy, 2, NoHitFlag);
        Status = IsError();
        // Only write to PLC if output changes.
        // This reduces load on PLC communication.
        IF MemOutput <> PLCOutput THEN
            PLCOutput = MemOutput;
    END
    SleepMS(500);
```

```

    END
    FuzzyClose (hFuzzy) ;
END

```

See Also[FuzzyTech Functions](#)**FuzzySetCodeValue**

Sets a value for the specified input of the specified instance.

Syntax

FuzzySetCodeValue(*hFuzzy*, *iIOIndex*, *iCodeValue*)

hFuzzy:

The fuzzy instance handle (and integer greater than 0).

iIOIndex:

Specifies the variable to receive the value. The I/O-Indices start with 0 and increment by 1 for each variable. To find the correct index for a specific variable, the variables need to be sorted in alpha-numerical order, first the inputs and then the outputs.

iCodeValue:

The value to be copied to the variable specified by *iIOIndex*.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[FuzzyOpen](#), [FuzzyGetCodeValue](#)**Example**

See [FuzzyOpen](#).

See Also[FuzzyTech Functions](#)**FuzzySetShellValue**

Sets a value for the specified input of the specified instance.

Syntax

FuzzySetShellValue(*hFuzzy*, *iIOIndex*, *rShellValue*)

hFuzzy:

The fuzzy instance handle (and integer greater than 0).

iIOIndex:

Specifies the variable to receive the value. The I/O-Indices start with 0 and increment by 1 for each variable. To find the correct index for a specific variable, the variables need to be sorted in alpha-numerical order, first the inputs and then the outputs.

rShellValue:

The value to be copied to the variable specified by *iIOIndex*.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FuzzyOpen](#), [FuzzyGetShellValue](#)

Example

See [FuzzyOpen](#).

See Also

[FuzzyTech Functions](#)

FuzzyTrace

Controls the trace process (starting and stopping) of the specified instance.

Syntax

FuzzyTrace(*hFuzzy*, *TraceOn*)

hFuzzy:

The fuzzy instance handle (and integer greater than 0).

TraceOn:

Specifies whether to start or to stop a trace process for the Fuzzy instance specified by *hFuzzy*. If this parameter is TRUE (1), the trace process is started. If this parameter is FALSE (0), the trace process is stopped.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[FuzzyOpen](#)

Example

See [FuzzyOpen](#).

See Also

[FuzzyTech Functions](#)

Chapter: 34 Group Functions

Group functions manipulate groups of areas, alarm categories, and any other data that can be accessed as a group. Use these functions to create a group dynamically to use for various purposes; for example, to allow operators to change their areas, or to view alarms by category, and so on.

Group Functions

Following are functions relating to groups of objects:

GrpClose	Closes a group.
GrpDelete	Deletes items from a group.
GrpFirst	Gets the first item in a group.
GrpIn	Tests if an item is in a group.
GrpInsert	Inserts items into a group.
GrpMath	Performs mathematical operations on groups.
GrpName	Gets the name of a group from a group handle.
GrpNext	Gets the next item in a group.
GrpOpen	Opens a group.
GrpToStr	Converts a group into a string

See Also

[Functions Reference](#)

GrpClose

Closes a group. The group is destroyed and the group handle becomes invalid. You should close a group when it is not in use, to release the associated memory. Citect-SCADA closes all groups on shutdown.

Syntax

GrpClose(*hGrp*)

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#)

Example

```
hGrp=GrpOpen ("MyGrp", 1);
..
GrpClose (hGrp);
```

See Also

[Group Functions](#)

GrpDelete

Deletes a single element or all elements from a group. You can also delete another group from within the group.

Syntax

GrpDelete(*hGrp, Value*)

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Value:

The element to delete from the group, from 0 to 16375.

- Set *Value* to -1 to delete all elements from the group.
- Set *Value* to a group handle to delete another group from this group.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpInsert](#), [GrpOpen](#)

Example

```
! Delete 10 and 14 from a group.
GrpDelete(hGrp,10);
GrpDelete(hGrp,14);
```

See Also

[Group Functions](#)

GrpFirst

Gets the value of the first element in a group. The first element in the group is the element with the lowest value. You can follow this function with a GrpNext() call, to get the value of all the elements in a group.

Syntax

GrpFirst(*hGrp*)

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Return Value

The value of the first element in a group or -1 if the group is empty.

Related Functions

[GrpOpen](#), [GrpNext](#)

Example

```
Value=GrpFirst(hGrp);
IF Value<>-1 THEN
    Prompt("First value is "+Value:###);
END
```

See Also

[Group Functions](#)

GrpIn

Determines if an element is in a group.

Syntax

GrpIn(*hGrp*, *Value*)

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Value:

The element to locate, from 0 to 16375.

- Set *Value* to a group handle to check if another group exists in the group.

Return Value

1 if the element is in the group, otherwise 0 is returned.

Related Functions

[GrpOpen](#), [GrpInsert](#), [GrpDelete](#)

Example

```
IF GrpIn(hGrp, 10) THEN
    Prompt("Area 10 in this group");
END
```

See Also

[Group Functions](#)

GrpInsert

Adds an element (or another group) to a group.

Syntax

GrpInsert(*hGrp*, *Value*)

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Value:

The element to add to the group, from 0 to 16375.

- Set *Value* to -1 to add all elements (0 to 16375) to the group.
- Set *Value* to a group handle to insert another group into the group.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#), [GrpDelete](#), [GrpIn](#)

Example

```
! Add 10 and 14 to a group.
GrpInsert(hGrp,10);
GrpInsert(hGrp,14);
```

See Also

[Group Functions](#)

GrpMath

Performs mathematical operations on two groups, and stores the result in another group. You can add the two groups, subtract one from the other, or perform Boolean AND, NOT, and XOR operations on the two groups.

Syntax

GrpMath(*hResult*, *hOne*, *hTwo*, *Type*)

hResult:

The group number where the result is placed.

hOne:

Number of first group used in the mathematical operation.

hTwo:

Number of the second group used in the mathematical operation.

Type:

- Type of mathematical operation:
- 0 - Add groups one and two.
 - 1 - Subtract group two from group one.
 - 2 - AND groups one and two.
 - 3 - NOT groups one and two.
 - 4 - XOR groups one and two.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#)

Example

```
hOne=GrpOpen("Plantwide",0);
hTwo=GrpOpen("Section1",0);
hResult=GrpOpen("Result",0);
! Subtract Section1 from Plantwide and place in Result.
GrpMath(hResult,hOne,hTwo,1);
```

See Also

[Group Functions](#)

GrpName

Gets the name of a group from a group handle.

Syntax

GrpName(*hGrp*)

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Return Value

The name of the group (as a string).

Related Functions[GrpOpen](#)**Example**

```
! Get the current group name.
sName=GrpName(hGrp);
```

See Also[Group Functions](#)**GrpNext**

Gets the value of the next element in a group. You can get the value of all the elements in a group. Call the GrpFirst() function to get the value of the first element, and then call this function in a loop.

Syntax**GrpNext(*hGrp*, *Value*)***hGrp*:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Value:

The value returned from GrpFirst() or the latest GrpNext() call.

Return Value

The value of the next element in a group, or -1 if the end of the group has been found.

Related Functions[GrpFirst](#)**Example**

```
! Count all values in a group.
Count=0;
Value=GrpFirst(hGrp);
WHILE Value<>-1 DO
    Count=Count+1;
    Value=GrpNext(hGrp,Value);
```

```
END  
Prompt("Number of values in group is "+Count:###);
```

See Also

[Group Functions](#)

GrpOpen

Creates a group and returns a group handle, or gets the group handle of an existing group. After you open a group, you can use the group number in functions that use groups, for example, SetArea() and AlarmSetInfo(). You can open a group that is specified in the Groups database. You can also create groups at runtime.

When you open a group that is defined in the database, a copy of the group is made - the original group is not used. You can therefore change the values in the group without affecting other facilities that use this group.

Syntax

GrpOpen(*Name*, *Mode*)

Name

The name of the group to open.

Mode

The mode of the open:

0 - Open an existing group

1 - Create a new group

2 - Attempts to open an existing group. If the group does not exist, it will create it.

Return Value

The group handle , or -1 if the group cannot be created or opened. The group handle identifies the table where all data on the associated group is stored.

Related Functions

[GrpClose](#)

Example

```
! Open Plantwide group defined in the database.  
hGrp=GrpOpen("Plantwide",0);
```

```

! Set current user area to Plantwide.
SetArea(hGrp);
GrpClose(hGrp);
! Set area to 1...10, 20 and 25 by creating a new group.
hGrp=GrpOpen("MyGrp",1);
StrToGrp(hGrp,"1..10,20,25");
SetArea(hGrp);
GrpClose(hGrp);

```

See Also[Group Functions](#)**GrpToStr**

Converts a group into a string of values separated by ", " and " .. ". You can then display the group on the screen or in a report.

Syntax**GrpToStr(*hGrp*)***hGrp*:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Return Value

The group (as a string).

Related Functions[GrpOpen](#), [StrToGrp](#)**Example**

```

! Display current areas.
hGrp=GetArea();
Str=GrpToStr(hGrp);
DspStr(21,"WhiteFont",Str);

```

See Also[Group Functions](#)

Chapter: 35 I/O Device Functions

The I/O device functions allow you to read the values of variables in I/O devices such as PLCs, and to write data into these I/O device variables. These functions also allow you to control I/O devices and to display information about I/O devices.

I/O Device Functions

Following are functions relating to I/O Devices:

<u>DriverInfo</u>	Provides information about the driver for a particular I/O Device.
<u>IODeviceControl</u>	Provides control of individual I/O Devices.
<u>IODeviceInfo</u>	Gets information on an I/O Device.
<u>IODeviceStats</u>	Gets statistical information for I/O Devices.

See Also

[Functions Reference](#)

DriverInfo

Provides information about the driver for a specified I/O device. Select the device using the *IODevice* argument, and the information to be returned using the *Type* argument.

This function can only be used if the I/O Server is on the current machine. When the I/O Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

DriverInfo(*IODevice*, *Type* [, *ClusterName*] [, *ServerName*])

IODevice:

The name of the I/O device.

Type:

The type of information returned about the driver. Specify one of the following:

- 0 - Driver Name
- 1 - Driver Title
- 2 - Block constant
- 3 - Max Retrys
- 4 - Transmit Delay
- 5 - Receive Timeout
- 6 - Polltime
- 7 - Watchtime (milliseconds)

Note: The DISKDRV driver name is returned as "Disk" instead of "DISKDRV". If the Polltime is set as "Interrupt", the function returns "0".

ClusterName:

Specifies the name of the cluster in which the I/O Server resides. This is optional if you have one cluster or are resolving the I/O server via the current cluster context. The argument is enclosed in quotation marks "".

ServerName:

Specifies the name of the the I/O Server. This parameter is only required if you are running more than one I/O server process from the same cluster on the same computer and need to instruct the system which process to redirect to. The argument is enclosed in quotation marks "".

Return Value

The driver information as a string. In the case of an [error](#) the return value is an empty string.

Example

```
// Using the IODevice Number
sName = DriverInfo(20, 0); ! Get the name of the driver used with I/O device 20
sName = DriverInfo(2, 1); ! Get the title of the driver used with I/O device 2
// Using the IODevice Name
sName = DriverInfo("IODev", 3);
! Get the Max Retrys value of the driver used with IODev
sName = DriverInfo("IODev1", 5);
! Get the Receive Timeout value of the driver used with IODev1
```

See Also

[I/O Device Functions](#)

IODeviceControl

Provides control of individual I/O devices. You might need to call this function several times. If you use incompatible values for the various options of this function, you might get unpredictable results.

This function can only be used if the I/O Server is on the current machine. When the I/O Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

IODeviceControl(*IODevice*, *Type*, *Data* [, *ClusterName*] [, *ServerName*])

IODevice:

The number or name of the I/O device. If you call this function from an I/O server, you can use the I/O device name. If you call this function from a client, you may use either the I/O device number or name. To specify all I/O devices, use "*" (asterisk) or -1.

Type:

The type of control action:

0 - No longer supported.

1 - Enable/Disable the I/O device on the I/O server. If disabled, attempts to read and write from the I/O device are ignored. (If another I/O device is configured as a standby I/O server, CitectSCADA switches communications to that I/O device.) The I/O server does not attempt to communicate with the I/O device until it is re-enabled. When the I/O device is re-enabled, the I/O server attempts to re-establish communication immediately. Mode 1 can only be called by the I/O Server which is associated with this device.

2 - No longer supported. An invalid argument error is returned if this option is specified.

3 - No longer supported. An invalid argument error is returned if this option is specified.

4 - The data in the associated I/O device cache is flushed. This allows flushing of the data from the I/O device without waiting for the aging time. This is useful when you have long cache time and you want to force a read from the I/O device.

The Data value is ignored with this mode.

5 - (For scheduled and remote I/O devices). The I/O device is added to the bottom of the list of I/O devices to be contacted. I/O devices already in the list (already waiting to be contacted) are given priority over this I/O device.

6 - (For scheduled and remote I/O devices). The I/O device is added to the top of the list of I/O devices to be contacted; it is given high priority. If there are already I/O devices at the top of the list with high priority, then this I/O device will be added to the list after them (that is it will be contacted after them). For dial-up remote I/O devices, if the modem is already in use - connected to another I/O device - this I/O device will not be dialled until that connection has been terminated.

7 - (For scheduled and remote I/O devices). The I/O device is added to the top of the list of I/O devices to be contacted, and it is given top priority. For dial-up remote I/O devices, if the modem is currently connected to another I/O device, the connection will be cancelled, and the top priority I/O device will be dialled. It will also stay connected until manually disconnected with another call to `IODeviceControl()`.

Note: This mode will not attempt to disconnect any other persistent connections. Persistent connections can only be disconnected using mode 8.

8 - (For scheduled and remote I/O devices). Disconnect an I/O device. Current requests will be completed before the I/O device is disconnected.

9 - (For scheduled I/O devices). The communication schedule for the I/O device is disabled. This is to minimize the likelihood that the I/O device will be contacted when its scheduled dial-time occurs.

10 - (For scheduled I/O devices). Puts the I/O device into Write On Request mode. That is, as soon as a write request is made, the I/O device will be added to the list of I/O devices to be contacted. It is given priority over existing read requests, but not over existing write requests.

In this situation, there will be a delay while the I/O device is contacted. Do not mistake this pause for inactivity (for example, do not continually execute a command during this delay).

11 - Change the I/O device cache timeout. If the I/O Server is restarted, the cache timeout will return to its original value. (For scheduled I/O devices, this value can be checked using the Kernel Page Unit command. For all other I/O devices, this value is configured in the Cache Time field at the I/O Devices Properties form.)

12 - The time of day at which to add the I/O device to the list of I/O devices to be contacted. Set the time in Data in seconds from midnight (for example, specify 6 p.m. as $18 * 60 * 60$). Use Type 12 to specify a one-time-communication.

13 - The communication period (the time between successive communication attempts). The value you specify represents different periods, depending on what type of schedule you are using (that is daily, weekly, monthly, or yearly. This is set using Type 15.). You can choose to specify the communication period either in seconds from midnight, day of week (0 to 6,

Sunday = 0), month (1 to 12), or year. Enter the value in Data. For example, if your schedule is weekly, and you set Data = 3, you are specifying each Wednesday. If your schedule is monthly, Data = 3 indicates March. For daily communication, set the period in Data in seconds from midnight; for example, set Data to $6 * 60 * 60$ to contact the I/O device every 6 hours.

14 - The time at which the I/O Server will first attempt to communicate with the I/O device. Set the time in Data in seconds from midnight, for example, to synchronize at 10a.m., set Data to $10 * 60 * 60$.

15 - Type of schedule. Set *Data* to one of the following:

- 1 - Daily
- 2 - Weekly
- 3 - Monthly
- 4 - Yearly

16 - (For remote I/O devices) Read all tags from the I/O device. Data is unused - set it to 0 (zero).

18 - Set Control Inhibit (Control Mode) for all tags of the I/O device.

Data:

Data for the control operation*:

1:

- Disable the I/O device (Disable Write On Request mode for Type 10)
- Set Control Inhibit to ON (mode for type 18)

0:

- Enable the I/O device (Enable Write On Request mode for Type 10) or the I/O device name (for Type 2 or 3).
- Set Control Inhibit to OFF (mode for type 18)

* For Type 5-8, Data is ignored; enter 0 (zero).

ClusterName:

Specifies the name of the cluster in which the I/O Server resides. This is optional if you have one cluster or are resolving the I/O server via the current cluster context. The argument is enclosed in quotation marks "".

ServerName:

Specifies the name of the the I/O Server. This parameter is only required if you are running more than one I/O server process from the same cluster on the same computer and need to instruct the system which process to redirect to. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[IODeviceInfo](#), [IODeviceStats](#), [TagReadEx](#), [TagWrite](#)

Example

```
IODeviceControl(4, 1, 1); ! Disable I/O device 4
```

See Also

[I/O Device Functions](#)

IODeviceInfo

Gets information about a specified I/O device.

Apart from when *Type* is set to 3 or 17, this function can only be used if the I/O Server is on the current machine, otherwise the function will not succeed and will return empty string. When the I/O Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

If both the primary and standby I/O devices are on the same server and they have the same I/O device name, you can get information about them individually by specifying the following:

```
IODeviceInfo("PLC1,P",1); // for the Primary  
IODeviceInfo("PLC1,S",1); // for the Standby
```

where P represents the primary I/O device and S the standby I/O device.

If you have more than one standby device on the same server, there is currently no way of using this function for other than the first standby device.

Note: When the I/O server is not in the calling process, this function could become a blocking function if the information required by this function is on an I/O server (except types 3 and 17, which are normally non-blocking). If this is the case, this function cannot be called from a foreground task (such as a graphics page) or an expression. Otherwise the return value will be undefined and a Cicode hardware alarm raised.

Syntax

IODeviceInfo(*IODevice*, *Type* [, *ClusterName*] [, *ServerName*])

IODevice:

The I/O device number, or the I/O device name enclosed in double quotes.

Type:

The type of information:

0 - Name of I/O device

1 - Protocol of I/O device

2 - Protocol address

3 - Client I/O device state

- 1 = Running - Client is either talking to an online I/O device or talking to a scheduled device that is not currently connected but has a valid cache
- 2 = Standby - Client is talking to an online standby I/O device
- 4 = Starting - Client is talking to an I/O device that is attempting to come online
- 8 = Stopping - Client is talking to an I/O device that is in the process of stopping
- 16 = Offline - Client is pointing to an I/O device that is currently offline
- 32 = Disabled - Client is pointing to a device that is disabled
- 66 = Standby write - client is talking to an I/O device configured as a standby write device
 - 4 - Current generic error number (decimal)
 - 5 - Current driver error number (decimal)
 - 6 - Disabled flag
 - 7 - Statistics, minimum read time
 - 8 - Statistics, maximum read time
 - 9 - Statistics, average read time
 - 10 - I/O server I/O device state
- 1 = Running - I/O device for this I/O server is online or a scheduled device that is not currently connected but has a valid cache
- 2 = Standby - I/O device for this I/O server is online and a standby unit
- 4 = Starting - I/O device for this I/O server is attempting to come online. Starting may be combined with either Offline or Remote such as: 20 = Starting(4) + Offline(16) or 132 = Starting(4) + Remote(128).
- 8 = Stopping - I/O device for this I/O server is currently in the process of stopping
- 16 = Offline (only valid on an I/O server) - I/O device for this I/O server is currently offline
- 32 = Disabled - I/O device for this I/O server is disabled
- 66 = Standby write - I/O device for this I/O server is configured as a standby write device
- 128 = Remote (returned in combination with another value specified above - see Starting - I/O device for this I/O server is a scheduled device but not currently connected
 - 11 - Unit number
 - 12 - Configured I/O server name
 - 13 - Configured Port name
 - 14 - Configured startup mode
 - 15 - Configured comment

16 - The primary I/O server name the client uses to communicate to this device

17 - The I/O Server the client is using to communicate to this device. Will be Standby if the Primary is down.

18 - State of the remote unit:

- 0 = Remote unit is disconnected and OK
- 1 = Remote unit is connected and online
- 2 = Remote unit is in the dial queue
- 16 = Remote unit is disconnected and offline
- 32 = Remote unit is disconnected and disabled

This mode causes redirection to the I/O server if running in separate processes.

19 - Number of successful attempts to communicate with the scheduled I/O device.

20 - Number of unsuccessful attempts to communicate with the scheduled I/O device.

21 - Write mode: Write On Request, and normal (as per schedule defined in the Express Communications Wizard).

22 - Number of queued read requests for the scheduled I/O device. (This mode causes redirection to the I/O server if running in separate processes.)

23 - Number of queued write requests for the scheduled I/O device. (This mode causes redirection to the I/O server if running in separate processes.)

24 - The cache timeout (in milliseconds).

26 - The name of the line device (for example, modem) you are using to connect to the I/O device. (This mode causes redirection to the I/O server if running in separate processes.)

27 - The call_status of a currently connected remote I/O device.

DIALCALLSTATE_UNAVAIL	0
DIALCALLSTATE_IDLE	1
DIALCALLSTATE_OFFERING	2
DIALCALLSTATE_ACCEPTED	3
DIALCALLSTATE_DIALTONE	4
DIALCALLSTATE_DIALING	5
DIALCALLSTATE_RINGBACK	6

DIALCALLSTATE_BUSY	7
DIALCALLSTATE_SPECIALINFO	8
DIALCALLSTATE_CONNECTED	9
DIALCALLSTATE_PROCEEDING	10
DIALCALLSTATE_ONHOLD	11
DIALCALLSTATE_CONFERENCED	12
DIALCALLSTATE_ONHOLDPENDCONF	13
DIALCALLSTATE_ONHOLDPENDTRANSFER	14
DIALCALLSTATE_DISCONNECTED_NORMAL	16
DIALCALLSTATE_DISCONNECTED_LINELOST	17
DIALCALLSTATE_DISCONNECTED_UNKNOWN	18
DIALCALLSTATE_DISCONNECTED_REJECT	19
DIALCALLSTATE_DISCONNECTED_PICKUP	20
DIALCALLSTATE_DISCONNECTED_FORWARDED	21
DIALCALLSTATE_DISCONNECTED_BUSY	22
DIALCALLSTATE_DISCONNECTED_NOANSWER	23
DIALCALLSTATE_DISCONNECTED_BADADDRESS	24
DIALCALLSTATE_DISCONNECTED_UNREACHABLE	25
DIALCALLSTATE_DISCONNECTED_CONGESTION	26
DIALCALLSTATE_DISCONNECTED_INCOMPATIBLE	27
DIALCALLSTATE_DISCONNECTED_UNAVAIL	28
DIALCALLSTATE_DISCONNECTED_NODIALTONE	29

DIALCALLSTATE_DISCONNECTED_NUMBERCHANGED	30
DIALCALLSTATE_DISCONNECTED_OUTOFORDER	31
DIALCALLSTATE_DISCONNECTED_TEMPFAILURE	32
DIALCALLSTATE_DISCONNECTED_QOSUNAVAIL	33
DIALCALLSTATE_DISCONNECTED_BLOCKED	34
DIALCALLSTATE_DISCONNECTED_DONOTDISTURB	35
DIALCALLSTATE_DISCONNECTED_CANCELLED	36
DIALCALLSTATE_UNKNOWN	48

(This mode causes redirection to the I/O server if running in separate processes.)

28 - The call rate (in bits per second) which may be the DTE or DCE connection speed depending on the server modem settings. (This mode causes redirection to the I/O server if running in separate processes.)

30 - The last time an I/O device from the remote I/O device redundant group was connected (primary or any standbys).

31 -The state of the remote I/O device redundant group:

- 0 = not connected (none of the redundant I/O devices connected)
- 1 =connected (one of the redundant I/O devices is connected)

32 - The next time the specified I/O device is scheduled to connect (unless a higher priority I/O device comes online).

ClusterName:

Specifies the name of the cluster in which the I/O Server resides. This is optional if you have one cluster or are resolving the I/O server via the current cluster context. The argument is enclosed in quotation marks "".

ServerName:

Specifies the name of the I/O Server. This parameter is only required if you are running more than one I/O server process from the same cluster on the same computer and need to instruct the system which process to redirect to. The argument is enclosed in quotation marks "".

Return Value

The type of information (as a string).

Related Functions

[IODeviceControl](#), [IODeviceStats](#), [TagReadEx](#), [TagWrite](#)

Example

```
//Using the IODevice Number
sName = IODeviceInfo(20, 0); ! Get the name of I/O device 20
sName = IODeviceInfo(2, 1); ! Get the protocol of I/O device 2
//Using the IODevice Name
sName = IODeviceInfo("IODev",10); ! Get the I/O Server State
sName = IODeviceInfo("IODev1",3); ! Get the State of IODev1
```

See Also

[I/O Device Functions](#)

IODeviceStats

Gets statistical information for all I/O devices, and displays the information in a dialog box.

Note: In a multi-process environment this function needs to be called from the IOServer process or redirected there using [MsgRPC](#). If this isn't done, some of the information on the IODeviceStats form will not be displayed correctly.

Syntax

IODeviceStats()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[IODeviceInfo](#)

Example

```
IODeviceStats(); ! display all I/O device information
```

See Also

[I/O Device Functions](#)

Chapter: 36 Keyboard Functions

Keyboard functions control the processing of keyboard entries, including the movement of the keyboard cursor and manipulation of keyboard commands.

Keyboard Functions

Following are functions relating to the keyboard:

<u>Key- AllowCursor</u>	Allows the command cursor to move to any AN or only to ANs that have commands defined.
<u>KeyBs</u>	Deletes the last character from the key command line.
<u>KeyDown</u>	Moves the command cursor down.
<u>KeyGet</u>	Gets the raw key code from the key command line.
<u>Key- GetCursor</u>	Gets the AN where the cursor is positioned.
<u>KeyLeft</u>	Moves the command cursor left.
<u>KeyMove</u>	Moves the command cursor in the requested direction.
<u>KeyPeek</u>	Gets a key from the key command line without removing the key.
<u>KeyPut</u>	Puts a raw key code into the key command line.
<u>KeyPutStr</u>	Puts a string into the key command line.
<u>KeyReplay</u>	Replays the last key sequence.
<u>KeyReplayAll</u>	Replays and executes the last key sequence.
<u>KeyRight</u>	Moves the command cursor right.
<u>Key- SetCursor</u>	Moves the command cursor to a specified AN.

KeySetSeq	Adds a keyboard sequence at runtime.
KeyUp	Moves the command cursor up.
SendKeys	Sends a keystroke (or string of keystrokes) to a window.

See Also

[Functions Reference](#)

KeyAllowCursor

Allows (or disallows) the command cursor to move to the specified AN or to all ANs. The command cursor normally moves only to ANs that have commands defined.

Syntax

KeyAllowCursor(AN, State)

AN:

The AN where the command cursor can move. If 0, all ANs are implied.

State:

Allow state:

0 - Do not allow the cursor to move to this AN.

1 - Allow the cursor to move to this AN.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyGetCursor](#)

Example

```
KeyAllowCursor(20,1);
! Allows the command cursor to move to AN20.
KeyAllowCursor(0,1);
! Allows the command cursor to move to any AN.
```

See Also

[Keyboard Functions](#)

KeyBs

Removes the last key from the key command line. If the key command line is empty, this function will not perform any action.

You should call this function using a "Hot Key" command (as shown in the example below), otherwise the backspace character is placed into the key command line and the command does not execute. A "Hot Key" command is a command that executes as soon as it is placed into the key command line.

Syntax

KeyBs()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyGet](#)

Example

System Keyboard	
Key Sequence	*Bs
Command	KeyBs()
Comment	Define a backspace Hot Key

(*) represents a HotKey command)

```
/* If "START A B C" is in the key command line and "START" is
the Key Name for the "F1" key: */
KeyBs();
! Removes ASCII "C".
KeyBs();
! Removes ASCII "B".
KeyBs();
! Removes ASCII "A".
KeyBs();
! Removes Key_F1.
KeyBs();
! Performs no action.
```

See Also

[Keyboard Functions](#)

KeyDown

Moves the command cursor down the page to the closest AN. If an AN-Down cursor override is specified (in the Page Keyboard database) for the graphics page, the command cursor moves to that AN instead.

Syntax

KeyDown()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyUp](#), [KeyLeft](#), [KeyRight](#), [KeyMove](#)

Example

See [KeyDown](#).

See Also

[Keyboard Functions](#)

KeyGet

Gets the last key code from the key command line. The key is removed from the command line. Use this function to process the operator key commands directly. You should call this function from the keyboard event function.

Syntax

KeyGet()

Return Value

The last key code from the key command line. If the key command line is empty, 0 (zero) is returned.

Related Functions

[KeyPeek](#), [KeyPut](#)

Example

```
/* If "START A B C" is in the key command line and "START" is
the Key Name for the "F1" key: */
Variable=KeyGet();
! Sets Variable to 67 (ASCII "C").
Variable=KeyGet();
! Sets Variable to 66 (ASCII "B").
Variable=KeyGet();
! Sets Variable to 65 (ASCII "A").
Variable=KeyGet();
! Sets Variable to 170 (the ASCII value of the F1 key (Key_F1)).
Variable=KeyGet();
! Sets Variable to 0.
```

See Also

[Keyboard Functions](#)

KeyGetCursor

Gets the AN at the position of the command cursor.

If this function is called from within a larger piece of code, the cursor may have moved away from where it was originally positioned when the larger piece of code was started.

If you are using groups, and there are currently two command cursors, the AN for the innermost will be returned. For example, if there is a cursor for a group as well as a cursor for one of its objects, the AN for the object will be returned.

Syntax

KeyGetCursor()

Return Value

The AN at the position of the command cursor. If no cursor is visible, -1 is returned.

Related Functions

[KeySetCursor](#)

Example

```
! If the command cursor is on AN25:
AN=KeyGetCursor();
! Sets AN to 25.
```

See Also

[Keyboard Functions](#)

KeyLeft

Moves the command cursor left (across the page) to the closest AN. If an AN-Left cursor override is specified (in the Page Keyboard database) for the graphics page, the command cursor moves to that AN instead.

Syntax

`KeyLeft()`

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyRight](#), [KeyUp](#), [KeyDown](#), [KeyMove](#)

Example

See [KeyRight](#)

See Also

[Keyboard Functions](#)

KeyMove

Moves the command cursor in a specified direction to the closest AN. If an AN cursor override is specified, the command cursor moves to that AN directly. This function is equivalent to the KeyUp(), KeyDown(), KeyLeft(), and KeyRight() functions.

Syntax

`KeyMove(Direction)`

Direction:

Direction to move the cursor:

- 0 - Do not move
- 1 - Left
- 2 - Right
- 3 - Up
- 4 - Down

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyUp](#), [KeyDown](#), [KeyLeft](#), [KeyRight](#)

Example

```
KeyMove(1);  
! Moves the cursor left.
```

See Also

[Keyboard Functions](#)

KeyPeek

Gets the ascii key code from the key command line (at a specified offset), without removing the key from the key command line. An offset of 0 returns the key code from the last position in the key command line.

Syntax

KeyPeek(*Offset*)

Offset:

The offset from the end of the key command line

Return Value

The ASCII key code.

Related Functions

[KeyGet](#)

Example

```
! If "A B C" is in the key command line:  
Variable=KeyPeek(0);  
! Sets Variable to 67 (ASCII "C")  
Variable=KeyPeek(2);  
! Sets Variable to 65 (ASCII "A")
```

See Also

[Keyboard Functions](#)

KeyPut

Puts an ASCII key code or Keyboard key code into the last position of the key command line. If this key completes any command, that command will execute.

Syntax

KeyPut(*KeyCode*)

KeyCode:

The key code to put into the key command line.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyGet](#)

Example

```
KeyPut(Key_F1);
/* Puts "Key_F1" (the Key Code for the "F1" key) into the last
position of the key command line. If "START" is the Key Name for
the "F1" key, this would be equivalent to
KeyPutStr("START"). In either case, "START" will display on the
key command line. */
KeyPut(StrToChar("A"));
/* Puts the Key Code for the "A" key into the last position of the
key command line. */
```

See Also

[Keyboard Functions](#)

KeyPutStr

Puts a string at the end of the key command line. The string can contain either key names or data characters. If this string completes any command, that command will execute.

Syntax

KeyPutStr(*String*)

String:

The string to put into the key command line.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyPut](#)

Example

```
KeyPutStr("START ABC");
! Places "START ABC" at the end of the key command line.
KeyPutStr("TURN PUMP 1 ON");
! Places "TURN PUMP 1 ON" at the end of the key command line.
```

See Also

[Keyboard Functions](#)

KeyReplay

Replays the last key sequence (except for the last key, which would execute the command). This function is useful when a command is to be repeated. To call this function from the keyboard, use a Hot Key "*" (asterisk) command, otherwise the KeyReplay() function itself is replayed.

Syntax

KeyReplay(*sub*)

sub:

Number of characters to subtract before replay. Default value is 1.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyReplayAll](#)

Example

If the previous contents of the key command line were:

```
"START ABC ENTER"
KeyReplay();
! Replays "START ABC".
```

See Also

[Keyboard Functions](#)

KeyReplayAll

Replays the last key sequence and executes the command. To call this function from the keyboard, use a Hot Key " * " (asterisk) command, otherwise the KeyReplayAll() function itself is replayed.

Syntax

KeyReplayAll()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyReplay](#)

Example

If the previous contents of the key command line were:

```
"START ABC ENTER"
KeyReplayAll();
! Replays "START ABC ENTER" and executes the command.
```

See Also

[Keyboard Functions](#)

KeyRight

Moves the command cursor right (across the page) to the closest AN. If an AN-Right cursor override is specified (in the Page Keyboard database) for the graphics page, the command cursor moves to that AN instead.

Syntax

KeyRight()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[KeyUp](#), [KeyDown](#), [KeyLeft](#), [KeyMove](#)

Example

See [KeyLeft](#)

See Also

[Keyboard Functions](#)

KeySetCursor

Displays the command cursor at a specified AN. A keyboard command needs to exist, or you need to first call the KeyAllowCursor() function (at the AN) to allow the cursor to move to the AN. If the AN does not exist, or if a command does not exist at that AN, or if KeyAllowCursor() has not been called, the command cursor does not move.

Syntax

KeySetCursor(AN)

AN:

The AN where the command cursor will be displayed.

Return Value

If the AN does not exist, or if a command does not exist at that AN, or if KeyAllowCursor() has not been called, the return value is 1. Otherwise, the function will return 0.

Related Functions

[KeyGetCursor](#), [KeyAllowCursor](#)

Example

```
! Move the command cursor to AN20.  
KeySetCursor(20);
```

See Also

[Keyboard Functions](#)

KeySetSeq

Adds a keyboard sequence to the current page at runtime. The key sequence is only added to the current window. When the page is closed, the keyboard sequence is deleted.

Syntax

KeySetSeq(*sKeySeq*, *AN*, *Fn*)

sKeySeq:

The keyboard sequence.

AN:

The AN where the keyboard sequence will apply. If you set AN to 0 (zero), the keyboard sequence will apply to all ANs on the page.

Fn:

The function to call when the keyboard sequence matches. This function needs to be a callback function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspButton](#), [DspButtonFn](#)

Example

```
/* Set the key sequence and call the "Callback" function when the
sequence is found. */
KeySetSeq("F2 ### Enter", 0, CallBack);
! This function is called when the key sequence is found.
INT
FUNCTION CallBack()
    INT Value;
    ! Get user data.
    Value=Arg1;
    ..
    RETURN 0;
END
```

See Also

[Keyboard Functions](#)

KeyUp

Moves the command cursor up the page to the closest AN. If an AN-Up cursor override is specified (in the Page Keyboard database) for the graphics page, the command cursor moves to that AN.

Syntax

KeyUp()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

[KeyDown](#), [KeyLeft](#), [KeyRight](#), [KeyMove](#)

Example

See [KeyUp](#).

See Also

[Keyboard Functions](#)

SendKeys

Sends a keystroke (or string of keystrokes) to a window as if they were typed on the keyboard. The window receives input focus and is brought to the foreground.

Syntax

SendKeys(*sTitle*, *sKeys*)

sTitle:

The title (caption) of the destination window.

sKeys:

The key (or keys) to send to *sTitle*.

- To send a single keyboard character, use the character itself. For example, to send the letter a, set *sKeys* to a. To send more than one character, append each additional character to the string. For example, to send the letters a, b, and c, set *sKeys* to abc.
- The plus (+), caret (^), and percent sign (%) have special meanings. To send one of these special characters, enclose the character with braces. For example, to send the plus sign, use {+}. To send a { character or a } character, use {{}} and {{}}, respectively.

-
- To specify characters that are not displayed when you press a key (such as **Enter** or **Tab**) and other keys that represent actions rather than characters, use the codes shown below:

Key	Code
Backspace	{backspace} or {bs} or {bksp}
Break	{break}
Caps Lock	{capslock}
Clear	{clear}
Del	{delete} or {del}
End	{end}
Enter	{enter} or ~
Esc	{escape} or {esc}
Help	{help}
Home	{home}
Insert	{insert}
Num Lock	{numlock}
Page Down	{pgdn}
Page Up	{pgup}
Print Screen	{prtsc}
Scroll Lock	{scrolllock}
Tab	{tab}
Up Arrow	{up}
Down Arrow	{down}

Right Arrow	{right}
Left Arrow	{left}
F1	{f1}
F2	{f2}
F3	{f3}
F4	{f4}
F5	{f5}
F6	{f6}
F7	{f7}
F8	{f8}
F9	{f9}
F10	{f10}
F11	{f11}
F12	{f12}

- To specify keys combined with any combination of **Shift**, **Ctrl**, and **Alt**, precede the regular key code with one or more of these codes:

Key	Code
Shift	+
Ctrl	^
Alt	%

To specify that **Shift**, **Ctrl**, and/or **Alt** are held down while several keys are pressed, enclose the keys in parentheses. For example, to hold down the **Shift** key while sending E then C, use +(EC). To hold down **Shift** while sending E, followed by C without the **Shift** key, use +EC. To specify repeating keys, use the form {key number}. For example, {left 42} means send the left arrow key 42 times. Be aware that you need to leave a space between the key and number.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WndFind](#)

Example

```
SendKeys("Untitled - Notepad", "abc");
// Send the key sequence "abc" to the Notepad application
```

See Also

[Keyboard Functions](#)

Chapter: 37 Mail Functions

The mail facility enables you to send data (for example, a report) between CitectSCADA users (or any other computer). CitectSCADA can send mail automatically, triggered by an event such as a report or an alarm. It can also read mail, so any user on the system can send mail to CitectSCADA (for example, a batch recipe).

You can use the mail facility to send information from CitectSCADA to Managers, Supervisors or anyone on a LAN or WAN whether they are running CitectSCADA or not. You can use it to send mail directly to these people whenever an event occurs (for example, you can mail reports directly to the QA department when they are scheduled, or send mail to the maintenance department when equipment is due for service).

The mail system uses the MAPI standard interface, so you can use any mail system that supports this standard. The mail system allows data transfer across different computer platforms and to remote sites (using a data gateway), enabling you to send high-level data across a wide area network.

Mail Functions

Following are functions relating to sending or receiving mail:

MailError	Gets the last mail error code
MailLogoff	Logoff from the mail system
MailLogon	Logon to the mail system
MailRead	Reads a standard mail message
MailSend	Sends a standard mail message

See Also

[Functions Reference](#)

MailError

Gets the last mail error code. The error code is extracted from the MAPI mail system, and explains what caused the MAPI error.

Syntax

MailError()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned. Refer also to MAPI errors.

Related Functions

[MailLogon](#), [MailLogoff](#), [MailSend](#), [MailRead](#)

Example

```
! Logon to the mail system
IF MailLogon("RodgerG", "password", 0) THEN
    error = MailError();
    !do what is required
END
```

See Also

[Mail Functions](#)

MailLogoff

Logs off from the mail system. You should log off the mail system when all mail operations are complete. CitectSCADA automatically logs off the mail system on shutdown.

Syntax

MailLogoff()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MailLogon](#), [MailSend](#), [MailRead](#)

Example

```
! Send the report to Rodger
MailLogon("Andrew", "password", 0);
MailSend("Rodger Gaff", "Report", "This is the weekly report",
```

```
"[data]:weekly.txt", 0);
MailLogoff();
```

See Also

[Mail Functions](#)

MailLogon

Logs on to the mail system. You need to call this function before any other mail function.

The mail system uses the MAPI standard interface, so you can use any mail system that supports this standard.

You should log on to the mail system when CitectSCADA starts, and log off only at shutdown. (The logon procedure can take a few seconds to complete.) You can only log on as one user at a time for each computer, so you can only read mail for this user name.

Syntax

MailLogon(*sName*, *sPassword*, *iMode*)

sName:

The name of the mail user. This name is the user's mail box name (the unique shorthand name, not the full user's name).

sPassword:

The password of the mail user.

iMode:

The mode of the logon:

0 - Normal logon.

2 - Get unique logon, do not share existing mail client logon.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MailLogoff](#), [MailSend](#), [MailRead](#), [MailError](#)

Example

```
! Send the report to James
MailLogon("RodgerG", "password", 0);
```

```
MailSend("James Glover", "Report", "This is the weekly report",
"[data]:weekly.txt", 0);
MailLogoff();
```

See Also

[Mail Functions](#)

MailRead

Reads a standard mail message. The mail message can contain text, an attached file, or both.

Before you can use this function, you need to use the MailLogon() function to log on to the mail system. You can only read mail sent to the user name specified in the MailLogon() function.

Syntax

MailRead(*sName*, *sSubject*, *sNote*, *sFileName*, *iMode*)

sName:

The name of the mail user who sent the message.

sSubject:

The subject text of the mail message.

sNote:

The note section of the message. If the message is longer than 255 characters, CitectSCADA will save the message in a file and return the file name in *sNote*. Use the file functions to read the message. The name of the file will be in the form CTxxxx where x is a unique number. You need to delete the file after you have finished with the mail message.

sFileName:

The name of any attached file. If there is no attached file in the message, specify *sFileName* as an empty string "".

iMode:

The mode of the read:

0 - Read a message. If no message is available, wait for a message.

1 - Read a message. If no message is available, return with an error code.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MailLogon](#), [MailLogoff](#), [MailSend](#), [MailError](#)

Example

```

! Logon to the mail system
MailLogon("RodgerG", "password", 0);
! Read a message. Don't wait if no message
IF MailRead(sName, sSubject, sNote, sFileName, 1) = 0 THEN
    ! got message now do something with it
END
WHILE TRUE DO
    ! wait for a mail message
    MailRead(sName, sSubject, sNote, sFileName, 0);
END;
MailLogoff();

```

See Also

[Mail Functions](#)

MailSend

Sends a standard mail message. The mail message can contain text, an attached file, or both.

Before you can use this function, you need to use the MailLogon() function to log on to the mail system. You can only send mail from the user name specified in the MailLogon() function. You can send mail to any mail user or to another Citect client.

Syntax

MailSend(*sName*, *sSubject*, *sNote*, *sFileName*, *iMode*)

sName:

The name of the mail user who will receive the message. This name is the user's full name (not their mailbox name).

sSubject:

The subject text of the mail message (a short description of what the message is about).

sNote:

The note section of the message (the main section of the message text). You can enter up to 255 characters, or a file name for longer messages. If you enter a file name, set *iMode* to 1.

sFileName:

The name of any attached file. If there is no attached file in the message, set sFileName to an empty string "".

iMode:

The mode of the send:

0 - Normal mail message.

1 - The *sNote* argument is the name of a text file to send as the note.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MailLogon](#), [MailLogoff](#), [MailRead](#), [MailError](#)

Example

```
! Logon to the mail system
MailLogon("Wombat", "password", 0);
! send the report to Andrew
MailSend("Andrew Bennet", "Report", "Attached is the weekly report",
"[data]:weekly.txt", 0);
! send hello message to JR
MailSend("Jack Russell", "Hello", "You've only got yourself to blame!", "", 0);
! send a big note to Nigel
MailSend("Nigel Colless", "Big Message", "[data]:message.txt", "", 1);
MailLogoff();
```

See Also

[Mail Functions](#)

Chapter: 38 Math and Trigonometry Functions

The following functions allow you to perform mathematical calculations in your Cicode files.

Math/Trigonometry Functions

Following are mathematical or trigonometrical functions:

Abs	Gets the absolute value of a number.
ArcCos	Gets the arccosine of an angle.
ArcSin	Gets the arcsine of an angle.
ArcTan	Gets the arctangent of an angle.
Cos	Gets the cosine of an angle.
DegToRad	Converts an angle from degrees to radians.
Exp	Raises e to the power of a number.
Fact	Gets the factorial of a number.
HighByte	Gets the high-order byte of a two-byte integer.
HighWord	Gets the high-order word of a four-byte integer.
Ln	Gets the natural logarithm of a number.
Log	Gets the base 10 logarithm of a number.
LowByte	Gets the low-order byte of a two-byte integer.
LowWord	Gets the low-order word of a four-byte integer.
Max	Gets the higher of two numbers.

<u>Min</u>	Gets the lower of two numbers.
<u>Pi</u>	Gets the value of pi.
<u>Pow</u>	Raises a number to the power of another number.
<u>RadToDeg</u>	Converts an angle from radians to degrees.
<u>Rand</u>	Gets a random number.
<u>Round</u>	Rounds a number.
<u>Sign</u>	Gets the sign of a number.
<u>Sin</u>	Gets the sine of an angle.
<u>Sqrt</u>	Gets the square root of a number.
<u>Tan</u>	Gets the tangent of an angle.

See Also

[Functions Reference](#)

Abs

Calculates the absolute (positive) value of a number. The absolute value of a number is the number without its sign.

Syntax

Abs(*Number*)

Number:

Any number.

Return Value

The absolute (positive) value of *Number*.

Related Functions

[Sign](#)

Example

```
Variable=Abs(-67);  
! Sets Variable to 67.  
Variable=Abs(67);  
! Sets Variable to 67.
```

See Also

[Math/Trigonometry Functions](#)

ArcCos

Calculates the arccosine of an angle.

Syntax

ArcCos(*Number*)

Number

The cosine of the angle.

Return Value

The arccosine (the angle, in radians) of *Number*.

Related Functions

[Cos](#)

Example

```
Variable=ArcCos(0.4);  
! Sets Variable to 1.1592...
```

See Also

[Math/Trigonometry Functions](#)

ArcSin

Calculates the arcsine of an angle.

Syntax

ArcSin(*Number*)

Number:

The sine of the angle.

Return Value

The arcsine (the angle, in radians) of *Number*.

Related Functions

[Sin](#)

Example

```
Variable=ArcSin(1);  
! Sets Variable to 1.5707...
```

See Also

[Math/Trigonometry Functions](#)

ArcTan

Calculates the arctangent of an angle.

Syntax

ArcTan(*Number*)

Number:

The tangent of the angle.

Return Value

The arctangent (the angle, in radians) of *Number*.

Related Functions

[Tan](#)

Example

```
Variable=ArcTan(0.4);  
! Sets Variable to 0.3805...
```

See Also

[Math/Trigonometry Functions](#)

Cos

Calculates the trigonometric cosine of an angle.

Syntax

Cos(*Angle*)

Angle:

Any angle (in radians).

Return Value

The cosine of *Angle*.

Related Functions

[ArcCos](#)

Example

```
Variable=Cos(0.7854);  
! Sets Variable to 0.7071...
```

See Also

[Math/Trigonometry Functions](#)

DegToRad

Converts an angle from degrees to radians.

Syntax

DegToRad(*Angle*)

Angle:

Any angle (in degrees).

Return Value

The angle in radians.

Related Functions

[RadToDeg](#)

Example

```
Variable=DegToRad(180);  
! Sets Variable to 3.1415... (pi).
```

See Also

[Math/Trigonometry Functions](#)

Exp

Calculates the exponential of a number (natural logarithm base e).

Syntax

Exp(*Number*)

Number:

Any number.

Return Value

The exponential of *Number* (to the base e).

Related Functions

[Log](#)

Example

```
Variable=Exp(1);  
! Sets Variable to 2.7182...
```

See Also

[Math/Trigonometry Functions](#)

Fact

Calculates the factorial of a number.

Syntax

Fact(*Number*)

Number:

Any number.

Return Value

The factorial of *Number*.

Example

```
Variable=Fact(6);  
! Sets Variable to 720 (that is 720=1x2x3x4x5x6).
```

See Also

[Math/Trigonometry Functions](#)

HighByte

Gets the high-order byte of a two-byte integer.

Syntax

HighByte(*TwoByteInteger*)

TwoByteInteger:

A two-byte integer.

Return Value

The high-order byte (that is $|X| - 1$)

Related Functions

[LowByte](#), [HighWord](#), [LowWord](#)

Example

```
Variable=HighByte(0x5678);  
! Sets Variable to 0x56.
```

See Also

[Math/Trigonometry Functions](#)

HighWord

Gets the high-order word of a four-byte integer.

Syntax

HighWord(*FourByteInteger*)

FourByteInteger:

A four-byte integer.

Return Value

The high-order word (that is | X | X | - | - |)

Related Functions

[LowWord](#), [HighByte](#), [LowByte](#)

Example

```
Variable=HighWord(0x12345678);  
! Sets Variable to 0x1234.
```

See Also

[Math/Trigonometry Functions](#)

Ln

Calculates the natural (base e) logarithm of a number.

Syntax

Ln(*Number*)

Number:

Any number.

Return Value

The natural (base e) logarithm of *Number*.

Related Functions

[Log](#)

Example

```
Variable=Ln(2);  
! Sets Variable to 0.6931...
```

See Also

[Math/Trigonometry Functions](#)

Log

Calculates the base 10 logarithm of a number.

Syntax

Log(*Number*)

Number:

Any number.

Return Value

The base 10 logarithm of *Number*.

Related Functions

[Ln](#)

Example

```
Variable=Log(100);  
! Sets Variable to 2 (that is 100=10 to the power of 2).
```

See Also

[Math/Trigonometry Functions](#)

LowByte

Gets the low-order byte of a two-byte integer.

Syntax

LowByte(*TwoByteInteger*)

TwoByteInteger:

A two-byte integer.

Return Value

The low-order byte (that is $| - | X |$)

Related Functions

[HighByte](#), [LowWord](#), [HighWord](#)

Example

```
Variable=LowByte(0x5678);  
! Sets Variable to 0x78.
```

See Also

[Math/Trigonometry Functions](#)

LowWord

Gets the low-order word of a four-byte integer.

Syntax

LowWord(*FourByteInteger*)

FourByteInteger:

A four-byte integer.

Return Value

The low-order word (that is $| - | - | X | X |$)

Related Functions

[HighByte](#), [LowByte](#), [HighWord](#)

Example

```
Variable=LowWord(0x12345678);  
! Sets Variable to 0x5678
```

See Also

[Math/Trigonometry Functions](#)

Max

Gets the higher of two numbers.

Syntax

Max(*Number1*, *Number2*)

Number1:

The first number.

Number2:

The second number.

Return Value

The higher of numbers *Number1* and *Number2*.

Related Functions

[Min](#)

Example

```
Variable=Max(24,12);  
! Sets Variable to 24.
```

See Also

[Math/Trigonometry Functions](#)

Min

Returns the lower of two numbers.

Syntax

Min(*Number1*, *Number2*)

Number1:

The first number.

Number2:

The second number.

Return Value

The lower of numbers *Number1* and *Number2*.

Related Functions

[Max](#)

Example

```
Variable=Min(24,12);  
! Sets Variable to 12.
```

See Also

[Math/Trigonometry Functions](#)

Pi

Gets the value of pi (the ratio of the circumference of a circle to its diameter).

Syntax

Pi()

Return Value

The value of pi.

Example

```
Variable=Pi();  
! Sets Variable to 3.1415...
```

See Also

[Math/Trigonometry Functions](#)

Pow

Calculates x to the power of y.

Syntax

Pow(X, Y)

X:

The base number.

Y:

The exponent.

Return Value

X to the power of Y.

Related Functions

[Exp](#)

Example

```
Variable=Pow(5,3);  
! Sets Variable to 125.
```

See Also

[Math/Trigonometry Functions](#)

RadToDeg

Converts an angle from radians to degrees.

Syntax

RadToDeg(*Angle*)

Angle:

Any angle (in degrees).

Return Value

The angle in degrees.

Related Functions

[DegToRad](#)

Example

```
Variable=RadToDeg(Pi());  
! Sets Variable to 180.
```

See Also

[Math/Trigonometry Functions](#)

Rand

Generates a random number between 0 and a specified maximum number less one.

The **Rand** function is zero-based, so the resultant number generated will range from zero to one less than the number provided in the **Maximum** argument.

Syntax

Rand(*Maximum*)

Maximum:

The maximum number. This number needs to be between 2 and 32767 (inclusive).

Return Value

A random number of integer type.

Example

```
Variable=Rand(101);
! Sets Variable to a random number from 0 to 100.
// To create a random number between 0 and 1 with 2 decimal places,
divide the above variable by 100, as shown here: //
Variable = Variable/100;
```

See Also

[Math/Trigonometry Functions](#)

Round

Rounds a number to a specified number of decimal places.

Syntax

Round(*Number, Places*)

Number:

The floating-point number to round.

Places:

The number of decimal places.

Return Value

The number rounded to *Places* decimal places.

Example

```
Variable=Round(0.7843,2);
! Sets Variable to 0.78 (result is rounded to 2 decimal places).
Variable=Round(123.45,-1);
! Sets Variable to 120.0 (rounded to -1 decimal place).
```

See Also

[Math/Trigonometry Functions](#)

Sign

Gets the sign of a number.

Syntax

Sign(*Number*)

Number:

Any number.

Return Value

The sign of *Number*.

Related Functions

[Abs](#)

Example

```
Variable=Sign(100);
! Sets Variable to 1.
Variable=Sign(-300);
! Sets Variable to -1.
Variable=Sign(0);
! Sets Variable to 0.
```

See Also

[Math/Trigonometry Functions](#)

Sin

Calculates the trigonometric sine of an angle.

Syntax

Sin(*Angle*)

Angle:

Any angle (in radians).

Return Value

The sine of *Angle*.

Related Functions

[ArcSin](#)

Example

```
Variable=Sin(0.7854);  
! Sets Variable to 0.7071...
```

See Also

[Math/Trigonometry Functions](#)

Sqrt

Gets the square root of a number.

Syntax

Sqrt(*Number*)

Number:

Any positive number.

Return Value

The square root of *Number*.

Related Functions

[Pow](#)

Example

```
Variable=Sqrt(4);  
! Sets Variable to 2.
```

See Also

[Math/Trigonometry Functions](#)

Tan

Calculates the trigonometric tangent of an angle.

Syntax

Tan(*Angle*)

Angle:

Any angle (in degrees).

Return Value

The tan of *Angle*.

Related Functions

[ArcTan](#)

Example

```
Variable=Tan(1);  
! Sets Variable to 1.5574
```

See Also

[Math/Trigonometry Functions](#)

Chapter: 39 Menu Functions

Menu functions can be used to access the contents defined in the Menu Configuration database at runtime.

Base Menu Tree

After your project is compiled, menu configurations defined in your project and the included project are merged into a single menu structure. The menu tree is configured according to the different pages specified for your menu items. The menu tree is sorted according to the order field defined in the menu configuration records. At runtime, the structure of the menu tree is as follows:

```
root node
|   L__<page node for generic pages>
|       |   L__<Level 1 items>...
|       |       L__<Level 2 items>...
|       |           L__<Level 3 items>...
|       |               L__<Level 4 items>...
|
L__<page node for specific page>
|   L__<Level 1 items>...
|       L__<Level 2 items>...
|           L__<Level 3 items>...
|               L__<Level 4 items>...
|
L__<page node for specific page>
```

The above menu tree represents a static view of the overall menu structure defined in your project. You can access a particular branch of the menu tree by the name of the page that the menu configuration belongs to.

To access the menu tree that is available for all pages (that is, generic pages), call Cicode function [MenuGetGenericNode](#) to get its node handle. You can browse its descendant child nodes using Cicode functions: [MenuGetFirstChild](#), [MenuGetNextChild](#) and [MenuGetPrevChild](#) etc.

To access the menu tree that is specific for a particular page, call Cicode function [MenuGetPageNode](#) to get its node handle. Similarly, you can use the menu browse functions to walk through the tree.

Dynamic Menu Tree

When a new page is displayed, an instance of the menu tree that merges the menu items for the generic pages and the menu items specified for the page will be automatically created. Under the merged tree, the level that represents different pages will no longer exist. The structure of the menu tree looks as follows:

```
root node
|____<Level 1 item>...
|      |____<Level 2 items>...
|          |____<Level 3 items>...
|              |____<Level 4 items>...
|____<Level 1 items>...
```

To access to the instance of the menu tree for a particular window, call Cicode function [MenuGetWindowNode](#) to get the handle of the root node. Similarly, you can use the same set of menu browse functions as above to walk through the tree.

See Also

[Menu Functions](#)

[Configuring Page Menus](#)

Menu Functions

The following functions allow you to access the contents of the menu configuration database in a tree-like format, and change the contents of the tree. Be reminded that changes made to the menu tree will not be persisted back to the menu configuration database.

MenuGetChild	Returns the handle to the child node with the specified name.
MenuGetFirstChild	Returns the handle to the first child of a menu node.
MenuGetGenericNode	Returns the handle to the base node of the menu tree for the generic pages.
MenuGetNextChild	Returns the next node that shares the same parent.
MenuGetPageNode	Returns the handle to the base node of the menu tree of a specified page.
MenuGetParent	Returns the parent node of the menu item.
MenuGetPrevChild	Returns the previous node that shares the same parent.
MenuGetWindowNode	Returns the handle to the root node for a given window.

MenuNodeAddChild	Dynamically adds a new item to the menu at runtime.
MenuNodeGetProperty	Return the item value of the specified menu node.
MenuNodeHasCommand	Checks whether the menu node has a valid cicode command associated with it.
MenuNodeIsDisabled	Checks whether the menu node is disabled by evaluating its DisabledWhen cicode expression.
MenuNodeIsHidden	Checks whether the menu node is hidden by evaluating its HiddenWhen cicode expression.
MenuNodeRemove	Remove the menu node from the menu tree.
MenuNodeRunCommand	Run the associated command for a menu node.
Menu- NodeSetDisabledWhen	Set the DisabledWhen expression for a newly added node.
Menu- NodeSetHiddenWhen	Set the HiddenWhen expression for a newly added node.
MenuNodeSetProperty	Set the item value of the specified menu node.
MenuReload	Reload base Menu Configuration from the compiled database.

See Also

[Functions Reference](#)

MenuGetChild

Returns the handle to the child node with the specified name.

Syntax

MenuGetChild(*hParent*, *sName*)

hParent:

Handle to the parent node in the menu tree.

sName:

The name of the child Menu node requested.

Return Value

The handle of the child node with the requested name, or -1 if unsuccessful.

Related Functions

[MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetFirstChild

Returns the handle to the first child of a menu node.

Syntax

MenuGetFirstChild(*hNode*)

hNode:

The handle to the parent node in the menu tree.

Return Value

The handle to the first child node of a menu node, or -1 if unsuccessful.

Related Functions

[MenuGetChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetGenericNode

Returns the handle to the base node of the menu tree for the generic pages. Its child nodes represent the menu items that do not have a page specified in the menu configuration database.

Syntax

MenuGetGenericNode([*bCreate*])

bCreate:

Determines if the node should be created if it does not exist. Defaults to 0, do not create.

Return Value

The handle to the base node of the menu tree, or -1 if it cannot find the node.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetNextChild

Returns the next node that shares the same parent.

Syntax

MenuGetNextChild(*hChild*)

hChild:

Handle to the current node in the menu tree

Return Value

The handle to next node that shares the same parent, or -1 if unsuccessful.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetPageNode

Returns the handle to the base node of the menu tree of a specified page. Its child nodes represent the menu items that have the particular page specified in the menu configuration database.

Syntax

MenuGetPageNode(*sPage* [, *bCreate*])

sPage:

The name of the page to return the base menu tree handle for.

bCreate:

Determines if the node should be created if it does not exist. Defaults to 0, do not create.

Return Value

The handle to the base node of the menu tree, or -1 if no handle returned to the base node of menu.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetParent

Returns the parent node of the menu item.

Syntax

MenuGetParent(*hNode*)

hNode:

Handle to the current node in the menu tree.

Return Value

The handle to parent menu node of the given menu item, or -1 if unsuccessful.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetPrevChild

Returns the previous node that shares the same parent.

Syntax

MenuGetPrevChild(*hChild*)

hChild:

Handle to the current node in the menu tree.

Return Value

The handle to previous node that shares the same parent, or -1 if unsuccessful.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuGetWindowNode

Returns the handle to the root node for a given window. This menu node is dynamically created for each page instance. Its child nodes represent the menu items for the generic pages merged with the ones specific to this page.

Syntax

MenuGetWindowNode(*hWin*)

hWin:

The window number of the desired window. You can call WinNumber() to get the window number of the page that calls this Cicode function.

Return Value

The handle to the root node of a given window, or -1 if unsuccessful.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuNodeAddChild

Dynamically add a new item to the menu at runtime. Be reminded that the changes are for the current session only and will not be persisted to the _Pagemen.RDB file.

Be reminded that changes made to the menu tree will not be persisted back to the menu configuration database.

Syntax

MenuNodeAddChild(*hParent*, *sName*, *sCommandName* [, *sCommandArgs*] [, *sSymbol*] [, *iOrder*])

hParent:

Handle of the parent node to add the new menu item under.

sName:

The string label of the new menu item.

sCommandName:

Specifies the name of the Cicode function to run.

sCommandArgs:

Specifies the parameters of the Cicode function to run.

sSymbol:

The symbol to be associated with the menu item.

iOrder:

The relative position in the menu for new item.

Return Value

The handle of the new node, or -1 if unsuccessful.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions Introduction](#)

[Menu Functions](#)

MenuNodeGetProperty

Return the item value of the specified menu node.

Syntax

MenuNodeGetProperty(*hNode*, *iField*)

hNode:

Handle to the current node in the menu tree.

iField:

Field for which you want the value:

0 - Name of Menu Item.

1 - Icon symbol to be associated with the menu item

2 - Privilege level required to run the command, otherwise the menu item is disabled.

3 - Area level required to run the command, otherwise the menu item is disabled.

4 - Disabled Style. Allows different display style for a disabled menu item.

5 - Checked setting. Whether the menu item will display a checkbox next to the label.

6 - Width. Specifies the menu item width in pixels.

7 - Comment

Return Value

An value for the specified Menu node field.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuNodeHasCommand

Checks whether the menu node has a valid cicode command associated with it.

Syntax

MenuNodeHasCommand(*hNode*)

hNode:

Handle of node to check.

Return Value

1 if the menu node has a valid cicode command, 0 if the menu node has no cicode command.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuNodeIsDisabled

Checks whether the menu node is disabled by evaluating its DisabledWhen cicode expression.

Syntax

MenuNodeIsDisabled(*hNode*)

hNode:

Handle of node to check.

Return Value

1 if menu node DisabledWhen expression evaluates to true, 0 if menu node DisabledWhen expression evaluates to false.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

Example

```
INT hNode = MenuNodeFirstChild(hParent);
IF (MenuNodeIsDisabled(hNode)) THEN
    ! set the menu item graphic state to disabled
END
```

See Also

[Menu Functions](#)

MenuNodeIsHidden

Checks whether the menu node is hidden by evaluating its HiddenWhen cicode expression.

Syntax

MenuNodeIsHidden(*hNode*)

hNode:

Handle of node to check.

Return Value

1 if menu node HiddenWhen expression evaluates to true, 0 if menu node HiddenWhen expression evaluates to false.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

Example

```

INT hNode = MenuNodeFirstChild(hParent);
IF (MenuNodeIsHidden(hNode)) THEN
    ! set the menu item graphic state to hidden
END

```

See Also

[Menu Functions](#)

MenuNodeRemove

Remove the menu node from the menu tree.

Be reminded that changes made to the menu tree will not be persisted back to the menu configuration database.

Syntax

MenuNodeRemove(*hNode*)

hNode:

Handle of node to remove.

Return Value

Zero (0) if node successfully removed. -1 if *hNode* is an invalid menu handle.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions Introduction](#)

[Menu Functions](#)

MenuNodeRunCommand

Run the associated command for a menu node.

Syntax

MenuNodeRunCommand(*hNode*)

hNode:

Handle of node to run command.

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if *hNode* does not refer to a valid node.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

See Also

[Menu Functions](#)

MenuNodeSetDisabledWhen

Set the DisabledWhen expression for a newly added node. Be aware this function only works for menu nodes added with [MenuNodeAddChild\(\)](#). The DisabledWhen expression may only be set once for a node.

Be reminded that changes made to the menu tree will not be persisted back to the menu configuration database.

Syntax

MenuNodeSetDisabledWhen(*hNode*, *sDisabledWhenName* [, *sDisabledWhenArgs*] [, *iDisabledStyle*])

hNode:

Handle of node to run command

sDisabledWhenName:

Cicode function for DisabledWhen expression. The function needs to return an INT.

sDisabledWhenArgs:

Cicode parameters for DisabledWhen expression. Only supports static arguments.

iDisabledStyle:

Disabled Style. Allows different display styles for a disabled menu item.

Return Value

CT_ERROR_NO_ERROR (0) on success, CT_ERROR_BAD_HANDLE (269) if hNode does not refer to a valid node, CT_ERROR_INVALID_ARG (274) if DisabledWhen Cicode has already been set or is not a valid expression.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

Example

```
INT hNode = MenuNodeAddChild((hParent, "LogIn", "LogIn");
INT Error = MenuNodeSetDisabledWhen(hNode, "UserInfo", "0", 1);
```

See Also

[Menu Functions Introduction](#)

[Menu Functions](#)

MenuNodeSetHiddenWhen

Set the HiddenWhen expression for a newly added node. Be aware this function only works for menu nodes added with MenuNodeAddChild(). The HiddenWhen expression may only be set once for a node.

Be reminded that changes made to the menu tree will not be persisted back to the menu configuration database.

Syntax

MenuNodeSetHiddenWhen(*hNode*, *sHiddenWhenName* [, *sHiddenWhenArgs*])

hNode:

Handle of node to run command.

sHiddenWhenName:

Cicode function for HiddenWhen expression. The function needs to return an INT.

sHiddenWhenArgs:

Cicode parameters for HiddenWhen expression. Only supports static arguments.

Return Value

CT_ERROR_NO_ERROR (0) on success, CT_ERROR_BAD_HANDLE (269) if hNode does not refer to a valid node, or CT_ERROR_INVALID_ARG (274) if HiddenWhen Cicode has already been set or is not a valid expression.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetProperty](#), [MenuReload](#)

Example

```
INT hNode = MenuNodeAddChild(hParent, "LogIn", "LogIn");
INT Error = MenuNodeSetHiddenWhen(hNode, "UserInfo", "0");
```

See Also

[Menu Functions Introduction](#)

[Menu Functions](#)

MenuNodeSetProperty

Set the item value of the specified menu node.

Be reminded that changes made to the menu tree will not be persisted back to the menu configuration database.

Syntax

MenuNodeSetProperty(*hNode*, *iField*, *sValue*)

hNode:

Handle to the current node in the menu tree.

iField:

Field for which you want to set the value:

0 - Name of Menu Item.

1 - Icon symbol to be associated with the menu item

- 2 - Privilege level required to run the command, otherwise the menu item is disabled.
- 3 - Area level required to run the command, otherwise the menu item is disabled.
- 4 - Disabled Style. Allows different display style for a disabled menu item.
- 5 - Checked setting. Whether the menu item will display a checkbox next to the label.
- 6 - Width. Specifies the menu item width in pixels.
- 7 - Comment

sValue:

The item value to set for the Menu node.

Return Value

Zero (0) if successful. -1 if *hNode* or *iField* is invalid

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodesHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuReload](#)

See Also

[Menu Functions Introduction](#)

[Menu Functions](#)

MenuReload

Reload base Menu Configuration from the compiled database.

Be reminded that the menu configuration loaded on the currently displayed page will not change until the page is refreshed. By default, menu configuration is automatically reloaded whenever a page is displayed.

Syntax

MenuReload()

Return Value

None.

Related Functions

[MenuGetChild](#), [MenuGetFirstChild](#), [MenuGetGenericNode](#), [MenuGetNextChild](#), [MenuGetPageNode](#), [MenuGetParent](#), [MenuGetPrevChild](#), [MenuGetWindowNode](#), [MenuNodeAddChild](#), [MenuNodeGetProperty](#), [MenuNodeHasCommand](#), [MenuNodeIsDisabled](#), [MenuNodeIsHidden](#), [MenuNodeRemove](#), [MenuNodeRunCommand](#), [MenuNodeSetDisabledWhen](#), [MenuNodeSetHiddenWhen](#), [MenuNodeSetProperty](#)

See Also

[Menu Functions](#)

Chapter: 40 Miscellaneous Functions

This section describes general Cicode functions.

Miscellaneous Functions

Following are miscellaneous functions.

AreaCheck	Determines whether the current user has access to a specified area.
Assert	Verifies a particular condition is true, or halts the task.
Beep	Beeps the speaker or sound card in the computer.
CitectInfo	Gets information about a CitectSCADA variable.
CodeTrace	Traces Cicode into the Kernel and the SYSLOG.DAT file.
DebugBreak	Causes a breakpoint error to start the Cicode Debugger.
DebugMsg	In-line debug messages of user Cicode.
DebugMsgSet	Enables/disables the DebugMsg function.
DelayShutdown	Causes CitectSCADA to shut down after a specified period
DisplayRuntimeManager	Starts and displays CitectSCADA Runtime Manager.
DumpKernel	Dumps Kernel data to the Kernel.dat file.
EngToGeneric	Converts a variable into generic scale format.
Exec	Executes an application or PIF file.
GetArea	Gets the current viewable areas.
GetEnv	Gets an environment variable.
GetLogging	Gets the current value for one or more logging parameters.

InfoForm	Displays graphics object help information for the AN closest to the cursor.
InfoFormAn	Displays graphics object help information for an AN.
Input	Gets text input from the operator.
IntToReal	Converts an integer variable into a real (floating point) number.
KerCmd	Executes a command in a kernel window.
KernelQueueLength	Obtains the number of rows in a queue.
KernelTableInfo	Provides a consistent method of accessing items within a Kernel Table.
KernelTableItemCount	Obtains the number of rows in a Kernel Table.
LanguageFileTranslate	Translates an ASCII file into the local language.
Message	Displays a message box on the screen.
ParameterGet	Gets the value of a system parameter.
ParameterPut	Updates a system parameter.
ProcessIsClient	Determines if the currently executing process contains a Client component.
ProcessIsServer	Determines if the currently executing process contains a particular server component.
ProcessRestart	Restarts the current process in which Cicode is running.
ProductInfo	Returns information about the CitectSCADA product.
ProjectInfo	Returns information about a particular project, which is identified by a project enumerated number.
ProjectRestartGet	Gets the path to the project to be run the next time CitectSCADA is restarted.
ProjectRestartSet	Sets the path to the project to be run the next time CitectSCADA is restarted.

ProjectSet	Sets the name or path of the current project.
Prompt	Displays a message in the prompt line.
Pulse	Pulses (jogs) a variable tag every two seconds.
ServiceGetList	Gets information about services running in the component calling this function.
SetArea	Sets the current viewable areas.
SetLanguage	Sets the current language for runtime text, and the character set.
SetLogging	Adjusts logging parameters while online.
Shutdown	Ends CitectSCADA's operation.
ShutdownForm	Displays a form that allows an operator to shut down the CitectSCADA system.
ShutdownMode	Gets the mode of the shutdown/restart.
SwitchConfig	Switches focus to the CitectSCADA configuration application.
TestRandomWave	Generates a random wave.
TestSawWave	Generates a saw wave.
TestSinWave	Generates a sine wave.
TestSquareWave	Generates a square wave.
TestTriangWave	Generates a triangular wave.
Toggle	Toggles a digital tag on or off.
TraceMsg	Displays a message in the Kernel and Debugger debug windows.
Version	Gets the version number of the CitectSCADA software.

See Also

[Functions Reference](#)

AreaCheck

Determines whether the current user has access to a specified area.

Syntax

AreaCheck(*Area*)

Area:

The area number (0 - 255)

Return Value

TRUE (1) if the user has access to the *Area* or FALSE (0) if not.

Related Functions

[GetArea](#), [GetPriv](#)

Example

```
IsArea = AreaCheck(5);
```

See Also

[Miscellaneous Functions](#)

Assert

Verifies that the specified expression is TRUE. If then expression is FALSE, the current task will be halted. This is useful to help prevent the execution of code you do not want to run in the event an error has been detected.

This function can be used in a debug mode, where the FALSE assertion will be logged to the Kernel and SysLog.DAT, with the time, date, Cicode file name, and line number.

Additionally the operator will be prompted with a dialog. The debug mode can be set by using the [Code]DebugMessage parameter or DebugMsgSet() function.

Note: If you have the "Citect will start debugger on hardware errors" option set in the Cicode Editor, the Debugger will start with the position before the Halt() instruction. You need to 'step over' this command if you want to continue debugging the function that called the Assert().

Syntax

Assert(*bCondition*)

bCondition:

The boolean expression. This expression needs to evaluate to TRUE (1) or FALSE (0).

Return Value

None. However, if the assertion tests as FALSE, error 347 is generated.

Related Functions

[Halt](#), [DebugMsg](#), [DebugMsgSet](#), [CodeTrace](#), [TraceMsg](#), [ErrLog](#)

Example

```
INT
FUNCTION FileDisplayEx(STRING sFileName);
    INT hFile;
    hFile = FileOpen(sFileName, "r");
    Assert(hFile <> -1);
    ...
    FileClose(hFile);
    RETURN 0;
END
```

See Also

[Miscellaneous Functions](#)

Beep

Beeps the internal speaker or sound card (installed in the computer). If you use the internal speaker on your computer, the function does not return until the sound has completed. If you use a sound card, the function returns immediately and the sound plays in the background.

Use the Windows Control Panel to set up waveforms.

Syntax

Beep(*nSound*)

nSound:

The type of sound:

-1 - Standard beep

0 - Default beep waveform

1 - Critical stop waveform

2 - Question waveform

3 - Exclamation waveform

4 - Asterisk waveform

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspPlaySound](#)

Example

```
/* Beeps the speaker with the default waveform. */
Beep(0);
```

See Also

[Miscellaneous Functions](#)

CitectInfo

Gets information about a CitectSCADA variable. This function returns internal statistics and other information about the CitectSCADA runtime system.

Note: This function is a non-blocking function and can only access data contained within the calling process; consequently it cannot return data contained in a different server process. This function is not redirected automatically by CitectSCADA runtime. If you want to make a call from one process to return data in another, use MsgRPC() to make a remote procedure call on the other process. Alternatively, include the server process that has the information that CitectInfo requires in the calling process.

Syntax

CitectInfo(*sGroup*, *sName*, *sType*)

sGroup:

The name of the group to which the variable belongs. Valid group names are: "General", "Port", "IODevice", "Network", "Stats", "Memory", or "Disk".

sName:

The name of the variable. This name depends on *sGroup*:

- "*General*" - the name is ignored.
- "*Port*" - the name of the I/O port configured in the database (with the Ports database). The port information is only valid for an I/O server. If the port name is "total", the total statistics for the I/O server are returned.

- "*IODevice*" - the name of the I/O device configured in the I/O devices database.

- "*Network*" - the name is ignored.

- "*Stats*" - The name of the statistics buffer or Statistical Information Record (SIR):

"Alarm Proc" - Alarm Processing (includes Digital, Analog, Advanced and High Resolution alarms).

"Citect n" - The CitectSCADA window where n is the window number (returned from the WinNumber() function)

"Code n" - The user Cicode task (thread) where n is the task handle (returned from the TaskHnd() function)

"Reset" - Reset the CitectSCADA statistics.

"Elapsed Time MS" - The elapsed time since statistics have been reset. Returns -1 if more than 20 days has elapsed.

- "*Memory*" - the measurement used

0 = bytes

KB = kilobytes

MB = megabytes

GB = gigabytes

- "*Disk*" - The disk drive to access:

0 = The current drive

1 = A:

2 = B:

3 = C: and so on.

sType:

The type of information to get, depending on sGroup:

"General" - General statistics:

0 - CPU usage

1 - CitectSCADA Kernel cycles per second

2 - CitectSCADA Kernel tasks per second

3 - CitectSCADA Kernel boot time

4 - CitectSCADA Kernel running time (in seconds)

5 - CitectSCADA startup time

6 - CitectSCADA running time in seconds

7 - Not supported in v7.10 or later

8 - Total read requests

9 - Total read requests per second

- 10 - Total write requests
 - 11 - Total write requests per second
 - 12 - Total Physical read requests
 - 13 - Total Physical read requests per second
 - 14 - Total Physical write requests
 - 15 - Total Physical write requests per second
 - 16 - Total Blocked read requests
 - 17 - Total Blocked write requests
 - 18 - Total Digital read requests
 - 19 - Total Register read requests
 - 20 - Total Digital read requests per second
 - 21 - Total Register read requests per second
 - 22 - Total Cache reads count
 - 23 - Total Cache reads %
 - 24 - Overall Average response time (ms)
 - 25 - Overall Minimum response time (ms)
 - 26 - Overall Maximum response time (ms)
 - 27 - Request sample for response times
 - 28 - Static point count is no longer supported. Calling the function with parameter 28 returns a value of 0 and a hardware alarm is raised.
 - 29 - Dynamic point count currently in use
 - 30 - Number of pending read requests from the device
 - 31 - Number of pending write requests to the device
 - 32 - Determines if CitectSCADA Kernel window is open
 - 33 - Percentage of the CPU used by the current CitectSCADA process
 - 34 - Total CPU time spent by the current CitectSCADA process in milliseconds
 - 35 - Total number of handles opened by the current CitectSCADA process
 - 36 - Total number of threads owned by the current CitectSCADA process
- "Port" - Port information for the I/O Server:**
- 0 - Read requests
 - 1 - Write requests
 - 2 - Physical read requests
 - 3 - Physical write requests
 - 4 - Cached read requests
 - 5 - Cached write requests
 - 6 - Blocked read requests
 - 7 - Blocked write requests

8 - Read requests per second
9 - Write requests per second
10 - Error count
11 - Read bytes counter
12 - Channel usage %
13 - Read bytes per second
14 - Statistics, minimum read time
15 - Statistics, maximum read time
16 - Statistics, average read time
17 - Statistics, time of samples
18 - Statistics, number of sample
100 - 119 - Driver specific counter values. CitectSCADA drivers can maintain up to 20 unique counters that can be accessed via this function. They are zero based, indexed from 100 to 119. If a value is not defined or maintained by the driver, 0 is returned for the value of the counter.

"IODevice" - I/O device information for the I/O device:

0 - Client side status:

- 1 = Running - Client is either talking to an online IO device or talking to a scheduled device that is not currently connected but has a valid cache
- 2 = Standby - Client is talking to an online standby IO device
- 4 = Starting - Client is talking to an IO device that is attempting to come online
- 8 = Stopping - Client is talking to an IO device that is in the process of stopping
- 16 = Offline - Client is pointing to an IO device that is currently offline
- 32 = Disabled - Client is pointing to a device that is disabled
- 66 = Standby write - Client is talking to an I/O device configured as a standby write device

1 - I/O Server status:

- 1 = Running - I/O Server is either talking to an online IO device or talking to a scheduled device that is not currently connected but has a valid cache
- 2 = Standby - I/O Server is talking to an online standby IO device
- 4 = Starting - I/O Server is talking to an IO device that is attempting to come online
- 8 = Stopping - I/O Server is talking to an IO device that is in the process of stopping
- 16 = Offline - I/O Server is pointing to an IO device that is currently off-line
- 32 = Disabled - I/O Server is pointing to a device that is disabled

- 66 = Standby write - I/O Server is talking to an I/O device configured as a standby write device
- 2 - If this I/O device is a standby device
3 - Last generic error
4 - Last driver error
5 - Error count
6 - Initialization count
7 - Statistics, minimum read time
8 - Statistics, maximum read time
9 - Statistics, average read time
10 - Statistics, number of samples

"Network" - Network statistical information:

- 0 - Read Network Control Blocks (NCBs)
1 - Maximum pending read NCBs
2 - Minimum pending read NCBs
3 - Current pending read NCBs
4 - Number of short read NCBs
5 - Write NCBs
6 - Maximum pending write NCBs
7 - Minimum pending write NCBs
8 - Current pending write NCBs
9 - Number of short write NCBs
10 - Total NCBs
11 - Maximum pending total NCBs
12 - Minimum pending total NCBs
13 - Current pending total NCBs
14 - Number of short total NCBs
15 - Minimum send response time in milliseconds
16 - Maximum send response time in milliseconds
17 - Average send response time in milliseconds
18 - Send packet count

"Stats" - Statistical information:

- 0 - Minimum time between code executions (cycles)
1 - Maximum time between code executions (cycles)
2 - Average time between code executions (cycles)
3 - Total cycle time in milliseconds
4 - Minimum time to execute the code in milliseconds

5 - Maximum time to execute the code in milliseconds

6 - Average time to execute the code in milliseconds

7 - Total execute time in milliseconds

"Memory" - Memory information:

0 - Free virtual memory

1 - Free windows system resources as %

2 - Free Physical Memory

3 - Memory Paging File Size

4 - Total Physical Memory

5 - Total % of Physical Memory Used (Win 2000 or later) and % of the last 1000 pages in memory that are in use (Win NT or earlier).

6 - Total working set size counter for current CitectSCADA process

7 - Private bytes counter of current CitectSCADA process

"Disk" - Disk information:

0 - Free disk space in bytes

1 - Total disk space in bytes

2 - Free disk space in kilobytes

3 - Total disk space in kilobytes

Return Value

The type of information (as an integer).

Related Functions

[IODeviceInfo](#), [WinNumber](#), [TaskHnd](#)

Example

```

! Get free memory
FreeMemory = CitectInfo("Memory", "", 0);
! Get free disk space on C:
FreeDisk = CitectInfo("Disk", 3, 0);
! Get max cycle time for digital alarms
MaxCycleTime = CitectInfo("Stats","Digital Alm","1");

```

See Also

[Miscellaneous Functions](#)

CodeTrace

Traces Cicode into the Kernel and the SYSLOG.DAT file. Use this function for finding bugs in your Cicode. It will trace the functions called, the arguments to those functions, and their return values. It will also trace any errors, writes to the I/O devices, and task state changes.

Note: Use this function only during system development; it can cause excessive loading on the CPU.

WARNING

UNINTENDED EQUIPMENT OPERATION

Only use the CodeTrace() function during system development and testing. This function is not intended for use in an operational system.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax

CodeTrace(*hTask*, *nMode*)

hTask:

The Cicode task handle as returned from TaskHnd() function or any of the following special values:

- 0 - Foreground Cicode.
- 2 - The next created task.
- 3 - New created tasks.
- 4 - All tasks.

nMode:

The mode of the trace:

- 0 - Tracing off
- 1 - Trace user Cicode functions calls
- 2 - Trace built-in function calls
- 4 - Trace errors
- 8 - Trace writes to the I/O devices
- 16 - Trace task state changes
- 1 - All modes (except 0)

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Assert](#), [TaskHnd](#), [DebugMsg](#), [DebugMsgSet](#), [TraceMsg](#), [ErrLog](#)

Example

```
// Start tracing errors
CodeTrace(TaskHnd(), 4);
...
// Stop tracing
CodeTrace(TaskHnd(), 0);
// trace functions in new task
CodeTrace(-2, 1 + 2);
TaskNew("MyFunc", "data", 0);
```

See Also

[Miscellaneous Functions](#)

DebugBreak

Causes a breakpoint exception error to occur (error number 342). This allows programmers to trap invalid states in their Cicode. If the Cicode Editor is not running, and the **Citect will start debugger on hardware errors** option is set (**Debug menu - Options**), the Debugger will be started. When the debugger starts, the correct Cicode file, function, and line will be displayed.

Syntax

DebugBreak()

Return Value

None.

Related Functions

[DspKernel](#), [KerCmd](#), [DumpKernel](#), [TraceMsg](#)

Example

```
!Check to see that rSpan is greater than zero else cause a break.
```

```
If rSpan equals 0 it would cause a Divide by Zero hardware error
anyway.
IF rSpan > 0 THEN
    rCalcRate = iAmount/rSpan;
ELSE
    DebugBreak();
END
```

See Also

[Miscellaneous Functions](#)

DebugMsg

Provides in-line debug messages of user Cicode, to the Kernel, Debugger Debug window, and the SysLog.DAT file. This function can be enabled or disabled with the [Code]DebugMessage parameter or DebugMsgSet() function at runtime.

Syntax

DebugMsg(*sMessage*)

sMessage:

The debugging message to log. Be sure to enclose this message in double quotes ("").

Return Value

None.

Related Functions

[Assert](#), [DebugMsgSet](#), [CodeTrace](#), [TraceMsg](#), [ErrLog](#)

Example

```
INT
FUNCTION
FileDisplayEx(STRING sFileName);
    INT hFile;
    hFile = FileOpen(sFileName, "r");
    DebugMsg("When opening file " + sFileName + ", the handle was:
" + IntToStr(hFile));
    ...
    FileClose(hFile);
    RETURN 0;
END
```

See Also

[Miscellaneous Functions](#)

DebugMsgSet

Enables/disables the DebugMsg() logging functionality. It also controls whether logging is enabled for the Assert() function. This function also sets the [Code]DebugMessage parameter appropriately.

Syntax

DebugMsgSet(*nMode*)

nMode:

The logging mode:

- 0 - Disable logging.
- 1 - Enable logging.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Assert](#), [DebugMsg](#)

Example

Buttons	
Text	Debug Enable
Command	DebugMsgSet(1)
Comment	Enable debug logging

See Also

[Miscellaneous Functions](#)

DelayShutdown

Terminates CitectSCADA's operation after the specified delay period (in milliseconds). This function is suitable to be called by the CTAPI. The delay period enables the user to close the connection between the CTAPI and third-party applications before Citect-SCADA shuts down.

Syntax

DelayShutdown(*Delay*)

Delay:

The period (in milliseconds) after which CitectSCADA will shut down.

Return Value

No return value.

Related Functions

[CtOpen](#), [CtClose](#), [CtCicode](#)

Example

```
DelayShutdown(10 000)
!Terminates CitectSCADA's operation after 10 seconds
```

See Also

[Miscellaneous Functions](#)

DisplayRuntimeManager

This function will start the CitectSCADA Runtime Manager if it is not already running , otherwise it will just bring the CitectSCADA Runtime Manager to the foreground.

Syntax

DisplayRuntimeManager()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

DumpKernel

Dumps Kernel data to the KERNEL.DAT file.

Syntax

DumpKernel(*iMode*, *sName*)

iMode:

The Kernel data to dump:

0x0001	Dump general statistics.
0x0002	Dump the task.
0x0004	Dump the I/O device.
0x0008	Dump the driver.
0x0010	Dump netstat. (This mode is deprecated and no longer active in version 7.10 or later.)
0x0020	Dump the table.
0x0040	Dump the queue.
0x4000	Dump in verbose mode.
0x8000	Dump kernel data in non-verbose mode. To dump data in verbose mode, set <i>iMode</i> to 0xc0000 (0x8000 + 0x4000).

You can select any one of the above modes or may add them together to get more than one type of information. For example, to dump netstat and I/O device data in verbose mode, set *iMode* to 0x4014 (0x0004 + 0x0010 + 0x4000). Using 0x4000 on its own will dump no data, it needs to be combined with another mode.

sName:

The queue or table name(empty for all queues or tables). Only valid if *iMode* is 0x0020 and 0x0040.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspKernel](#), [KerCmd](#), [TraceMsg](#)

Example

```
DumpKernel(0x8000, "");
```

```
!Dump the Kernel data
```

See Also

[Miscellaneous Functions](#)

EngToGeneric

Gets a variable in the CitectSCADA generic scale format. CitectSCADA uses this scale to display trends. It calls this function automatically for trends defined in the project. If you want to display a trend using Cicode you need to call this function.

Syntax

EngToGeneric(*Value*, *EngLow*, *EngHigh*)

Value:

The value to convert to the CitectSCADA generic scale format.

EngLow:

The engineering units zero scale.

EngHigh:

The engineering units full scale.

Return Value

The variable (in the range 0 - 32000).

Related Functions

[DspBar](#), [DspTrend](#)

Example

```
/* Using trend definition 5 at AN20, display the value of Tag1 on
Pen1 of the trend. Tag1 has an engineering scale of 0 to 100. */
DspBar(20,5,EngToGeneric(Tag1,0,100));
```

See Also

[Miscellaneous Functions](#)

Exec

Executes an application or PIF file. The application or command starts up and continues to run in parallel with CitectSCADA.

This function can return while the application is still starting up, so you should use the Sleep() function to allow the application enough time to start.

Syntax

Exec(Command [, Mode])

Command:

The operating system command to execute.

Mode:

The mode of the window:

1 - Normal

3 - Maximized

6 - Minimized

If you do not enter a mode, the default mode is 1.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Sleep](#)

Example

```
Exec("c:\winnt\system32\mspaint.exe");
! Starts up the Paint application with a normal window.
Exec("cmd /c mkdir c:\test");
! Uses the DOS shell to create a new directory
```

you need to quote paths and applications passed to the exec function (using ^" to embed quotes) so that the function can be executed correctly when long file names are used. For example:

```
Exec(PathToStr("^^" + "[BIN]:\Ctcicode") + "^^" + 
PathToStr("[RUN]:\cicode.ci") + "^^")
```

creates a command line that ends up as:

```
C:\Program Files\CitectSCADA\CitectSCADA 7.10\Bin\CtCicode C:\Program Files\C-
itectSCADA\CitectSCADA 7.10\Project\Cicode.ci
```

See Also

[Miscellaneous Functions](#)

GetArea

Gets the current logged-in areas.

Syntax

GetArea()

Return Value

The login area groups as an integer that represents a group handle. If this group is modified, the actual login areas do not change.

Related Functions

[SetArea](#)

Example

```
! If the current areas are 1, 5 and 20:  
hGrp=GetArea();  
Str=GrpToStr(hGrp);  
! sets Str to "1,5,20".
```

See Also

[Miscellaneous Functions](#)

GetEnv

Gets a DOS environment variable.

Syntax

GetEnv(*sName*)

sName:

The name of the environment variable.

Return Value

The DOS environment variable (as a string).

Example

```
/* Get the current DOS path. */
```

```
sPath = GetEnv("Path");
```

See Also

[Miscellaneous Functions](#)

GetLogging

Gets the current value for logging parameters.

The parameters you can query include the following:

- [Debug]DriverTrace
- [Debug]SysLogSize
- [Debug]Priority
- [Debug]CategoryFilterMode
- [Debug]CategoryFilter
- [Debug]SeverityFilterMode
- [Debug]SeverityFilter
- [Debug]LogShutdown
- [Debug]DebugAllTrans
- [Debug]EnableLogging
- [IOServer]RedundancyDebug
- [General]Verbose
- [General]VerboseToSysLog
- [CtAPI]Debug

Syntax

GetLogging(*Section*, *Name*)

Section:

The INI section name.

Name:

The system parameter name.

Return Value

If the function succeeds, the value is returned as a string. An empty string is returned if an error occurs. To get extended error information, call [IsError\(\)](#).

Related Functions

[SetLogging](#)

See Also

[Miscellaneous Functions](#)

InfoForm

Displays graphics object information for the object under the mouse pointer. If there is no object directly under the mouse pointer, it displays information for the nearest object. Each tag associated with the object is displayed, along with its raw and engineering values and a button that displays the details from the Variable Tags form. The function also displays the Cicode expression, with any result that the expression has generated.

This function only supports the display of simple Cicode expressions.

Syntax

InfoForm(*Mode*)

Mode:

For security purposes, the Write button on the information form displayed by this function can be disabled.

0 - The **Write** button on the displayed information form will be available and will function normally.

1 - The **Write** button will not be shown.

If you omit the mode, the default mode is 0.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[InfoFormAn](#)

Example

System Keyboard	
Key Sequence	AnHelp
Command	InfoForm();
Comment	Display graphics object help for the AN closest to the cursor

See Also[Miscellaneous Functions](#)**InfoFormAn**

Displays graphics object information for a specified AN. This function displays each tag associated with the object, along with its raw and engineering values and a button that displays the details from the Variable Tags form. The function also displays the Cicode expression, with any result that the expression has generated.

Syntax**InfoFormAn(AN [, Mode])***AN:*

The AN of the graphics object for which information is displayed.

Mode:

For security purposes, the Write button on the information form displayed by this function can be disabled.

0 - The **Write** button on the displayed information form will be available and will function normally.

1 - The **Write** button will not be shown.

If you omit the mode, the default mode is *0*.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[InfoForm](#)**Example**

System Keyboard	
Key Sequence	# ## AnHelp
Command	InfoFormAn(Arg1);
Comment	Display graphics object help for a specified AN

See Also

[Miscellaneous Functions](#)

Input

Displays a dialog box in which an operator can input a single value. The dialog box has a title, a prompt, and a single edit field. For multiple inputs, use the Form functions.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

Input(*Title*, *Prompt*, *Default*)

Title:

The title of the input box.

Prompt:

The prompt text.

Default:

The default text that the operator can edit or replace.

Return Value

The edit field entry (as a string). If the user presses the **Cancel** button , an empty string is returned and the IsError() function returns the error code 299.

Related Functions

[Message](#), [FormNew](#)

Example

```
/* Shut down CitectSCADA if the user inputs "Yes". */
STRING sStr;
sStr=Input("Shutdown","Do you wish to shutdown?","Yes");
IF sStr="Yes" THEN
    Shutdown();
END
```

See Also

[Miscellaneous Functions](#)

IntToReal

Converts an integer into a real (floating point) number.

Syntax

IntToReal(*Number*)

Number:

The integer to convert.

Return Value

The real number.

Related Functions

[RealToStr](#), [StrToInt](#)

Example

```
! Sets Variable to 45.0
Variable=IntToReal(45);
! Sets Variable to -45.0
Variable=IntToReal(-45);
```

See Also

[Miscellaneous Functions](#)

KerCmd

Executes a command in a Kernel window.

Syntax

KerCmd(*Window*, *sCommand*)

Window:

The name of the Kernel window.

sCommand:

The command to execute in the Kernel window.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspKernel](#), [TraceMsg](#), [DumpKernel](#)

See Also

[Miscellaneous Functions](#)

KernelQueueLength

Obtains the number of records in a queue.

Syntax

KernelQueueLength(*sName*)

sName:

The name of the queue, which can be enumerated by the page queue kernel command.

Return Value

This function returns the Length value of the page queue command.

Related Functions

None

See Also

[Miscellaneous Functions](#)

KernelTableInfo

Provides a consistent method of accessing items within Kernel Table.

Syntax

KernelTableInfo(*sTable*, *sRecord*, *sField*)

sTable:

The name of the table. The following tables are supported:

sTable	sRecord	sField	Notes
Bufferpool	The value of the 'Name' field	Mode, Size, Total, Free, InUse, Max, Peak, Short, Gets, TotErr, Grow, Shrink	These names are unique when each Bufferpool is created

sTable	sRecord	sField	Notes
CiCode	The value of the 'Name' field or the 'HND' field. If the user specifies a number as the sRecord we will assume it is a handle, otherwise we will look it up as a string name	Name, HND, State, CPU_Time, Poll_Time, Slice	The CiCode function TaskNew exposes the handle. That is why record access by handle is acceptable for this table. The name field may or may not be unique (if you run the same CiCode task twice with different parameters). Accessing by 'Name' will not succeed if there are duplicates
Task	The value of the 'Name' field	Mode, Hnd, State, Prty, Cpu, Min, Max, Avg, Count	None
Tran	The value of the TRAN 'Name' field	Name, Node, Type, Mode, Hnd, Cnt, Send, Rec, Wait, Stack, Service, State, Login, ReconnectCount, UpTime, TotalTxRxCount	The counters ReconnectCount, UpTime and TotalTxRxCount can be viewed in verbose mode on Table Tran page of the Kernel
Trendqueues	The value of the 'DriverName' field	FlushQueueLength, FlushQueuesMax , TotalFlushes, FileWrites	None

sRecord:

The key to a column in the table depending on the *sTable* parameter.

sField:

The key to a column in the table depending on the *sTable* parameter.

Return Value

Returns the content of a field of the given table in the format of a Cicode STRING.

Related Functions

[KernelTableItemCount](#), [DumpKernel](#)

See Also

[Miscellaneous Functions](#)

KernelTableItemCount

Obtains the number of rows in a Kernel Table.

Syntax

KernelTableItemCount(*sName*)

sName:

The name of the table that can be enumerated by the page table kernel command.

Return Value

Returns the number of active records in the table (not the length value displayed by the page table kernel command).

The following tables row counts are not returned by this command (it returns 0)

- IODevices.Tags
- IODevices.Subs
- IODevices.Blocks
- CSAToPSI.Subs"

[KernelTableInfo](#), [DumpKernel](#)

See Also

[Miscellaneous Functions](#)

LanguageFileTranslate

Translates an ASCII file into a local language. Use this function to translate RTF reports for viewing on client screens.

The local language used by this function is specified by the [Language]LocalLanguage parameter. You need to set this parameter accordingly.

Syntax

LanguageFileTranslate(*sSourceFile*, *sDestFile*)

sSourceFile:

The name of the ASCII file containing multi-language text, which will be copied and translated. You should specify the full path or use path substitution. The path and name should be contained within quotation marks.

sDestFile:

The name of the destination file which will be created/ replaced with the local/translated version of the source file. You should specify the full path or use path substitution. The path and name should be contained within quotation marks.

Return Value

1 if successful, otherwise 0.

Related Functions

[SetLanguage](#), [StrToLocalText](#)

Example

```
/* Translates the text in Shift.RTF and saves it in LShift.RTF. */
LanguageFileTranslate("c:\MyApplication\data\Shift.RTF","c:\MyApplication\data\
LShift.RTF");
/* Translates the text in [Data]:Shift.RTF and saves it in
[LocalData]:LShift.RTF.  */
LanguageFileTranslate("[Data]:Shift.RTF","[LocalData]:LShift.RTF"
);
```

See Also

[Miscellaneous Functions](#)

Message

Displays a message box on the screen and waits for the user to select the **OK** or **Cancel** button.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

Message(*Title*, *Prompt*, *Mode*)

Title:

The title of the message box.

Prompt:

The prompt displayed in the message box.

Mode:

The mode of the message box:

0 - **OK** button

1 - **OK** and **Cancel** button

16 - Stop Icon

32 - Question Icon

48 - Exclamation Icon

64 - Information Icon

Select more than one mode by adding the modes. For example, set Mode to 33 to display the **OK** and **Cancel** buttons and the Question icon. You can only display one icon for the message box.

Return Value

0 (zero) if successful, otherwise an error is returned. If the user presses the Cancel button the function returns an [error](#) code of 299.

Related Functions

[Input](#)

Example

```
/* Display an error message in a message box. */
IF Total<>100 THEN
    Message("Error","Total not 100%",48);
END
```

See Also

[Miscellaneous Functions](#)

ParameterGet

Gets the value of a system parameter. The system parameter can exist in the CITECT.INI file and/or in the Parameters database.

If the system parameter does not exist in the CITECT.INI file or the database, the default value is returned. If the system parameter exists in both CITECT.INI and the database, the value of the system parameter is taken from CITECT.INI.

Syntax

ParameterGet(*Section*, *Name*, *Default*)

Section:

The section name.

Name:

The system parameter name.

Default:

The default value of the parameter.

Note: If in your Cicode you perform a ParameterGet("alarm", "alarmsave", 1000) for say the [Alarm]SavePeriod parameter, and NO entry exists in Citect.ini or the parameters records, the returned value will be 1000 however CitectSCADA will internally be using the default value of 600.

Return Value

The parameter (as a string).

Related Functions

[ParameterPut](#), [WndGetFileProfile](#), [WndPutFileProfile](#)

Example

```
Variable=ParameterGet("Page", "Windows", "4");
```

See Also

[Miscellaneous Functions](#)

ParameterPut

Updates a system parameter in the CITECT.INI file. If the system parameter does not exist, it is added to the CITECT.INI file.

Syntax

ParameterPut(*Section*, *Name*, *Value*)

Section:

The section name.

Name:

The system parameter name.

Value:

The value to put in the system parameter.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ParameterGet](#), [WndGetFileProfile](#), [WndPutFileProfile](#)

Example

```
/* Changes the [Page] Windows system parameter in CITECT.INI to "4". */
ParameterPut("Page", "Windows", "4");
```

See Also

[Miscellaneous Functions](#)

ProcessIsClient

Determines if the currently executing process contains a Client component.

Syntax

ProcessIsClient()

Return Value

TRUE (1) if the process contains a Client component, otherwise FALSE (0).

Related Functions

[ProcessIsServer](#), [ServerInfo](#), [ServerInfoEx](#), [ServiceGetList](#)

Example

To execute local variable simulation code on the client only:

```
IF (ProcessIsClient()) THEN
    SimulateLocalVariables();
END
```

See Also

[Miscellaneous Functions](#)

ProcessIsServer

Determines if the currently executing process contains a particular server component.

Syntax

ProcessIsServer(*sServerType* [, *sClusterName*] [, *sServerName*])

sServerType:

Case insensitive string specifying the type of server to check for. Supported values are "IOServer", "Trend", "Alarm" and "Report".

sClusterName:

Optional case insensitive string specifying the cluster name to combine with the server type specified in the first argument and the server name (if specified).

sServerName:

Optional case insensitive string specifying a server name to combine with the server type specified in the first argument and the cluster name (if specified).

Return Value

TRUE (1) if the process contains the specified component, otherwise FALSE (0).

Related Functions

[ProcessIsClient](#), [ServerInfo](#), [ServerInfoEx](#), [ServiceGetList](#)

Example

To execute disk PLC tag simulation code only on the I/O server in Cluster1 with the name IOServer3:

```
IF (ProcessIsServer("IOServer", "Cluster1", "IOServer3")) THEN
    SimulateDiskTags();
END
```

See Also

[Miscellaneous Functions](#)

ProcessRestart

Restarts the current process in which Cicode is running.

Syntax

INT error = ProcessRestart()

Return Value

0 (zero) if successful, otherwise the following error is returned:

256 - General software error

See Also

[Cicode and General errors](#)

Related Functions

[ServerRestart](#)

Example

```
ProcessRestart()
```

See Also

[Miscellaneous Functions](#)

ProductInfo

Returns information about the CitectSCADA product.

Syntax

ProductInfo(*iType*)

iType:

Type of information required:

- 0 - product name, default
- 1 - product company
- 2 - product major version
- 3 - product minor version
- 4 - product version string

Return Value

The product information. An empty string will be returned if the type is invalid.

Related Functions

[ProjectInfo](#)

See Also

[Miscellaneous Functions](#)

ProjectInfo

Returns information about a particular project, which is identified by a project enumerated number.

Syntax

ProjectInfo(*iProject*, *iType*)

iProject:

Project number. 0 refers to the main project.

iType:

Type of information to return:

0 - Project name

1 - Project description

2 - Project major version

3 - Project minor version

4 - Project date

5 - Project time

Return Value

The specified project information. An empty string will be returned if the project number or type is invalid

Related Functions

[ProductInfo](#)

See Also

[Miscellaneous Functions](#)

ProjectRestartGet

Gets the path to the project to be run the next time CitectSCADA is restarted. (you need to have a project already set using either ProjectSet or ProjectRestartSet. Use this function with the Shutdown() function to shut down and then restart the project that is currently running.

Syntax

ProjectRestartGet()

Return Value

The path to the project to be run the next time CitectSCADA is restarted.

Related Functions

[Shutdown](#), [ShutdownMode](#), [ProjectSet](#), [ProjectRestartSet](#)

Example

See [Shutdown](#).

See Also

[Miscellaneous Functions](#)

ProjectRestartSet

Sets the path to the project to be run the next time CitectSCADA is restarted.

Syntax

ProjectRestartSet(*sPath*)

sPath:

The path to the project. You need to use the full path, for example to specify the path to the project "Demo", use: "C:\CITECT\USER\DEMO".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Shutdown](#), [ShutdownMode](#), [ProjectSet](#), [ProjectRestartGet](#)

See Also

[Miscellaneous Functions](#)

ProjectSet

Sets either the name or the path of the project to be run next time CitectSCADA is restarted. The project path is written to the [CtEdit]Run parameter.

Syntax

ProjectSet(*sProject*)

sProject:

The name of the project (for example "DEMO"), or the path to the project. If you specify the path to the project, you need to use the full path. If you do not specify a project, the current project will be used.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Shutdown](#), [ShutdownMode](#), [ProjectRestartGet](#)

Example

```
/* Set the next project to "Demo". */
ProjectSet("Demo");
/* Set the next project to "MyPath". */
ProjectSet("I:\CITECT\PROJECT1\MYPATH");
```

See Also

[Miscellaneous Functions](#)

Prompt

Displays a message in the prompt line (AN=2) on the operator's computer.

Syntax

Prompt(*String*)

String:

The message to be displayed.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Message](#), [DspError](#)

Example

```
/* Display "This is a prompt!" at the prompt AN. */
Prompt("This is a prompt!");
```

See Also

[Miscellaneous Functions](#)

Pulse

Pulses (jogs) a variable tag on, then off. The variable tag is switched ON (1) and two seconds later it is switched OFF (0). The exact period of the pulse is determined by the communication channel to the I/O device. If the communication channel is busy, the pulse time may be longer than two seconds. The code in the I/O device should not be dependant on a pulse time of exactly 2 seconds. Use the pulse as a trigger only.

Syntax

Pulse(*sTag*)

sTag:

The digital tag to pulse.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Toggle](#)

Example

Buttons	
Text	Jog 145
Command	Pulse(M145)
Comment	Pulse the variable tag M145 every two seconds

See Also

[Miscellaneous Functions](#)

ServiceGetList

Determines the service type(s), cluster name(s), and service name(s) of all services currently running in the component that called this function. It also determines if the client component is running.

If you call this function from a component that is running purely as a Control Client or View-only Client, the function will return "Client".

If you call this function from a single-process component that includes:

- I/O server called IOServ1 on ClustA
- Trend server called Trend1 on ClustB
- Alarm server called Alarm1 on ClustA
- Report server called Report1 on ClustB
- Client

The function will return a string similar to:

```
"IOServer.Clu-
stA.IOServ1,Trend.ClustB.Trend1,Alarm.ClustA.Alarm1,Report.ClustB.Report1,Client"
```

The order of components in the string is not fixed so the exact string may vary from the above. You should parse the returned string or alternatively use [ProcessIsClient](#) or [ProcessIsServer](#) to find the information you need.

Syntax

ServiceGetList()

Return Value

String in the form:

serviceType1.clusterName1.serviceName1,serviceType2.clusterName2.serviceName2, ... ,Client

Related Functions

[ProcessIsClient](#), [ProcessIsServer](#), [ServerInfo](#), [ServerInfoEx](#)

See Also

[Miscellaneous Functions](#)

SetArea

Sets the current viewable areas. You can pass a single area number, or a group of areas to set multiple areas. You can only set areas that are flagged for the current user.

Syntax

SetArea(*Area*)

Area:

The area to set (1 to 255).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#)

Example

```
/* Set current viewable area to Area 1. */
SetArea(1);
```

See Also

[Miscellaneous Functions](#)

SetLanguage

Sets the language database from which the local translations of native strings in the project will be drawn, and specifies the character set to be used. Native strings are those that are preceded by a @, and enclosed in brackets (for example, @**(Motor Overload)**).

This function will dynamically change the language of display items such as alarm descriptions, button text, keyboard/alarm logs, graphic text, Cicode strings etc. The language will only be changed on the client that calls the function. This means that you can display different languages at different clients, even though they are running the same project.

If the local language character set differs from the default character set of the Windows installation, the runtime text may be garbled. You can set the local language and character set by using this function, or through the [Language]LocalLanguage and [Language]CharSet Parameters.

Syntax

SetLanguage(*sLanguage*, *n CharSet*)

sLanguage:

The name of the language database from which the local translations of native strings in the project will be drawn. The .dbf extension is optional.

n CharSet:

The character set to use when displaying the localized text in runtime:

0	ANSI
1	Default

128	Japanese - Shiftjis
129	Korean - Hangul
130	Korean - Johab
134	Chinese - simplified
136	Chinese - traditional
161	Greek
162	Turkish
163	Vietnamese
177	Hebrew
178	Arabic
186	Baltic
204	Russian
222	Thai

Return Value

0 (zero) if successful, otherwise 262 (the file could not be opened).

Related Functions

[LanguageFileTranslate](#), [StrToLocalText](#)

Example

```
SetLanguage("French",1);
! Changes the current language to French, using the Windows
default character set.
```

See Also

[Miscellaneous Functions](#)

SetLogging

Adjusts logging parameters while online. The parameters you can modify include the following:

- [Debug]DriverTrace
- [Debug]SysLogSize
- [Debug]Priority
- [Debug]CategoryFilterMode
- [Debug]CategoryFilter
- [Debug]SeverityFilterMode
- [Debug]SeverityFilter
- [Debug]LogShutdown
- [Debug]DebugAllTrans
- [Debug]EnableLogging
- [IOServer]RedundancyDebug
- [General]Verbose
- [General]VerboseToSysLog
- [CtAPI]Debug

Syntax

SetLogging(*Section*, *Name*, *Value*, *Persist*)

Section:

The INI section name.

Name:

The system parameter name.

Value:

The value you would like to set the parameter to.

Persist:

Makes the value persistent across restarts by writing the value to the INI file.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GetLogging](#)

See Also

[Miscellaneous Functions](#)

Shutdown

Terminates CitectSCADA's operation. Use this function to shut down the CitectSCADA system, otherwise buffered data could be lost.

Note: With one exception, the Shutdown command will succeed only if there is an Alarm Server in your system. The exception to this is if you specify an empty string for the *sDest* parameter (shutdown this computer only). In this case the shutdown will succeed even if there is no Alarm Server.

The shutdown can affect only the computer that calls it, or all or part of a CitectSCADA network. If you are shutting down a network, specify the computers (Control Clients and servers) to be shut down in *sDest*, and the extent of the shutdown in *Mode*.

Note: If [Shutdown]NetworkIgnore parameter is set to 0 (zero) and a client receives a shutdown request message from a server. Phase 2 clients only receive a shutdown request when the first phase 1 client has reconnected to the server.

You can allow selected computers to override the shutdown with the [Shutdown]NetworkIgnore parameter. (You might set this parameter for key servers, for example, I/O servers.)

Use the ShutdownForm() function to prompt the user for verification before shutting CitectSCADA down.

Note: If the [Shutdown]NetworkStart parameter is set to 0 (zero), the Shutdown() function will ignore the *sDest* argument. This will result in the shutting down and restarting of the machine the function is run on regardless of the machine specified.

Syntax

Shutdown([sDest] [, sProject] [, Mode] [, ClusterName] [, CallEvent])

sDest:

The destination computer(s) to be shut down, as a string:

"" (blank string) - This computer only - Default value

["Computer_Name"] - The specified CitectSCADA client (defined in the computer's CITECT.INI file)

["Server_Name"] - The specified CitectSCADA server

"All Clients" - All CitectSCADA clients on the network

"All Servers" - All CitectSCADA servers on the network

"Everybody" - All CitectSCADA computers on the network

sProject:

The full path of the project to run on restart as a string. The path is written to the configuration files and is used when the system restarts. The default value is "", which means that no changes are made to the configuration and the current project is restarted.

Mode:

The type of shutdown:

- 1 - Shutdown CitectSCADA only - Default value
- 2 - Shutdown and restart CitectSCADA (without logging off Windows)
- 3 - Shutdown and restart CitectSCADA and log off Windows (needs to set up an auto login to the Operating System and CitectSCADA needs to be configured to run on start up or log in)
- 4 - Shutdown CitectSCADA and re-boot the computer
- 5 - Shutdown CitectSCADA only
- 6 - Shutdown and restart CitectSCADA clients, but not this computer
- 7 - Shutdown CitectSCADA and shutdown the computer. If the computer supports power off feature the power will be turned off

ClusterName:

The name of the cluster that the machine(s) named in Dest belongs to. This is not required if:

- the function is called with *Dest* empty (the default); OR
- the client is connected to only one cluster containing an Alarm Server.

CallEvent:

Flag for initiating a user-specified shutdown event prior to shutting down. Refer to [OnEvent\(\)](#) type code for the value of shutdown event.

- 0 - no event
- 1 - raise event. The user defined shutdown event handler will be called before the shutdown occurs - Default value

Note: If the event handler is non-interactive with an instant return value, it can be called directly.

Note: If the event handler is interactive or with a long delay in processing the event, it needs to be called indirectly using the NewTask("EventHandler") function, and the actual handler, EventHandler(), needs to call Shutdown() with the CallEvent flag set to 0 from the handler if it decides the shutdown is permitted.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[ProjectRestartGet](#), [ProjectRestartSet](#), [ProjectSet](#), [ShutdownMode](#), [ShutdownForm](#), [OnEvent](#)

Example

```
/* Shut down CitectSCADA on this computer. */
Shutdown();
/* Shut down and restart CitectSCADA clients, but not this computer. */
Shutdown("All Clients", ProjectRestartGet(), 6, "ClusterXYZ");
```

See Also

[Miscellaneous Functions](#)

ShutdownForm

Displays a dialog box to verify that the user really wants to shut down the CitectSCADA system. If the user selects [Yes], CitectSCADA is shut down.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

ShutdownForm()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Shutdown](#)

Example**System Keyboard**

Key Sequence	Shutdown
--------------	----------

Command	ShutdownForm();
Comment	Display the shutdown form
Buttons	
Text	Shutdown
Command	ShutdownForm();
Comment	Display the shutdown form

See Also

[Miscellaneous Functions](#)

ShutdownMode

Gets the mode of the last Shutdown function call. The mode is useful to identify the type of Shutdown that was performed.

Syntax

ShutdownMode()

Return Value

The shutdown mode set when shutdown was called.

Related Functions

[Shutdown](#)

Example

```
nMode = ShutdownMode()
If nMode = 4 Then
    Prompt ("Citect closed down and computer was rebooted")
END
```

See Also

[Miscellaneous Functions](#)

SwitchConfig

Switches focus to a specific CitectSCADA configuration application. If the specified application is not running, it will be started.

Syntax

SwitchConfig(*nApp*)

nApp:

The configuration application:

- 1 - Citect Graphics Builder (CTDRAW32.EXE)
- 2 - Citect Project Editor (CTEDIT32.EXE)
- 3 - Citect Explorer (CTEXPLOR.EXE)
- 4 - Citect Cicode Editor (CTCICODE.EXE)

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Example

```
! Switch to the Graphics Builder.
SwitchConfig(1);
```

See Also

[Miscellaneous Functions](#)

TestRandomWave

Generates a random wave. The height of the wave is restricted by minimum and maximum values. You can offset the starting point of the wave, for example, to display multiple waves at the same AN. You can use this function to generate test input.

Syntax

TestRandomWave([*Period*] [, *Low*] [, *High*] [, *Offset*])

Period:

The number of seconds required to generate one cycle of the wave. If no period is entered, the period has a default of 60.

Low:

The lowest value of the wave. If no low value is entered, this value has a default of 0.

High:

The highest value of the wave. If no high value is entered, this value has a default of 100.

Offset:

The offset in seconds, to generate the wave. If no offset is entered, the offset has a default of 0.

Return Value

The value of the random wave.

Related Functions

[TestSawWave](#), [TestSinWave](#), [TestSquareWave](#), [TestTriangWave](#)

Example

```
/* Specify a random wave of 60 seconds duration, with a minimum
   value of 0 units and a maximum value of 100 units, with no offset.
 */
TestRandomWave(60,0,100,0);
```

See Also

[Miscellaneous Functions](#)

TestSawWave

Generates a saw wave. The height of the wave is restricted by minimum and maximum values. You can offset the starting point of the wave, for example, to display multiple waves at the same AN. You can use this function to generate test input.

Syntax

TestSawWave([Period] [, Low] [, High] [, Offset])

Period:

The number of seconds required to generate one cycle of the wave. If no period is entered, the period has a default of 60.

Low:

The lowest value of the wave. If no low value is entered, this value has a default of 0.

High:

The highest value of the wave. If no high value is entered, this value has a default of 100.

Offset:

The offset in seconds, to generate the wave. If no offset is entered, the offset has a default of 0.

Return Value

The value of the saw wave.

Related Functions

[TestRandomWave](#), [TestSinWave](#), [TestSquareWave](#), [TestTriangWave](#)

Example

```
/* Specifies a saw wave of 60 seconds duration, with a minimum
   value of 0 units and a maximum value of 100 units, with no offset.
 */
TestSawWave();
```

See Also

[Miscellaneous Functions](#)

TestSinWave

Generates a sine wave. The height of the wave is restricted by minimum and maximum values. You can offset the starting point of the wave, for example, to display multiple waves at the same AN. You can use this function to generate test input.

Syntax

TestSinWave([Period] [, Low] [, High] [, Offset])

Period:

The number of seconds required to generate one cycle of the wave. If no period is entered, the period has a default of 60.

Low:

The lowest value of the wave. If no low value is entered, this value has a default of 0.

High:

The highest value of the wave. If no high value is entered, this value has a default of 100.

Offset:

The offset in seconds, to generate the wave. If no offset is entered, the offset has a default of 0.

Return Value

The value of the sine wave.

Related Functions

[TestRandomWave](#), [TestSawWave](#), [TestSquareWave](#), [TestTriangWave](#)

Example

```
/* Specifies a sine wave of 30 seconds duration, with a minimum
   value of 0 units and a maximum value of 100 units, with no offset.
 */
TestSinWave(30);
```

See Also

[Miscellaneous Functions](#)

TestSquareWave

Generates a square wave. The height of the wave is restricted by minimum and maximum values. You can offset the starting point of the wave, for example, to display multiple waves at the same AN. You can use this function to generate test input.

Syntax

TestSquareWave([Period] [, Low] [, High] [, Offset])

Period:

The number of seconds required to generate one cycle of the wave. If no period is entered, the period has a default of 60.

Low:

The lowest value of the wave. If no low value is entered, this value has a default of 0.

High:

The highest value of the wave. If no high value is entered, this value has a default of 100.

Offset:

The offset in seconds, to generate the wave. If no offset is entered, the offset has a default of 0.

Return Value

The value of the square wave.

Related Functions

[TestRandomWave](#), [TestSawWave](#), [TestSinWave](#), [TestTriangWave](#)

Example

```
/* Specifies a square wave of 30 seconds duration, with a minimum
value of -1000 units and a maximum value of 1000 units, with an
offset of 10 seconds. */
TestSquareWave(30,-1000,1000,10);
```

See Also[Miscellaneous Functions](#)**TestTriangWave**

Generates a triangular wave. The height of the wave is restricted by minimum and maximum values. You can offset the starting point of the wave, for example, to display multiple waves at the same AN. You can use this function to generate test input.

Syntax**TestTriangWave([Period] [, Low] [, High] [, Offset])***Period:*

The number of seconds required to generate one cycle of the wave. If no period is entered, the period has a default of 60.

Low:

The lowest value of the wave. If no low value is entered, this value has a default of 0.

High:

The highest value of the wave. If no high value is entered, this value has a default of 100.

Offset:

The offset in seconds, to generate the wave. If no offset is entered, the offset has a default of 0.

Return Value

The value of the triangular wave.

Related Functions[TestRandomWave](#), [TestSawWave](#), [TestSinWave](#), [TestSquareWave](#)**Example**

```
/* Specifies a triangular wave of 60 seconds duration, with a
```

```
minimum value of 0 units and a maximum value of 100 units, with no
offset. */
TestTriangWave();
```

See Also

[Miscellaneous Functions](#)

Toggle

Toggles a digital tag on or off. If the variable tag is ON (1), this function will turn it OFF. If the variable tag is OFF (0), this function will turn it ON.

Syntax

Toggle(*sTag*)

sTag:

The digital tag to toggle.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Pulse](#)

Example

Buttons

Text	Motor 145
Command	Toggle(M145)
Comment	Toggle the variable tag M145 (Motor 145) on or off

To toggle the Paging Alarm property:

```
Toggle(MyCluster.Alarm_1.Paging);
```

See Also

[Miscellaneous Functions](#)

TraceMsg

Displays a message in the Kernel and Debugger debug windows.

Syntax

TraceMsg(*String*)

String:

The message to display.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspKernel](#), [KerCmd](#), [DumpKernel](#), [DebugBreak](#), [Assert](#), [DebugMsg](#), [DebugMsgSet](#), [CodeTrace](#), [ErrLog](#)

Example

```
TraceMsg("Error Found");
// Displays "Error Found" in the debug window
```

See Also

[Miscellaneous Functions](#)

Version

Gets the version number of the CitectSCADA software in use.

Syntax

Version(*Type*)

Type:

The type of version:

- 0 - Major version number
- 1 - Minor version number
- 2 - Revision number
- 3 - Version text

Return Value

The version number as a string.

Example

```
! If the CitectSCADA version number is 1.2:  
Version(0);  
! Returns 1.  
Version(1);  
! Returns 2.  
Version(3);  
! Returns "1.2".
```

See Also

[Miscellaneous Functions](#)

Chapter: 41 Page Functions

Page functions display graphics pages, files, and the standard alarm, trend, and menu pages.

Page Functions

Following are functions relating to graphics pages:

<u>PageAlarm</u>	Displays a category of active alarms on the predefined alarms page.
<u>PageBack</u>	Displays the previously displayed page in the Window.
<u>PageDisabled</u>	Displays a category of disabled alarms on the predefined alarms page.
<u>PageDisplay</u>	Displays a graphics page.
<u>PageFile</u>	Displays a file on the predefined file to screen page.
<u>PageFileInfo</u>	Returns the width or height of an unopened page in pixels.
<u>PageForward</u>	PageForward() restores the previously displayed page in the window following a PageBack command.
<u>PageGetInt</u>	Gets a local page-based integer.
<u>PageGetStr</u>	Gets a local page-based string.
<u>PageGoto</u>	Displays a graphics page without pushing the last page onto the PageLast stack.
<u>PageHardware</u>	Displays the active hardware alarms on the predefined alarms page.
<u>PageHistoryDspMenu</u>	Displays a pop-up menu which lists the page history of current window.
<u>PageHistoryEmpty</u>	Used to determine if the page history of the current window is empty.

<u>PageHome</u>	Displays the predefined home page in the window.
<u>PageInfo</u>	Gets page information.
<u>PageLast</u>	Displays the last ten graphics pages.
<u>PageMenu</u>	Displays a menu page with page selection buttons.
<u>PageNext</u>	Displays the next graphics page.
<u>PagePeekCurrent</u>	Return the index in the page stack for the current page..
<u>PagePeekLast</u>	Gets any page on the PageLast stack.
<u>PagePopLast</u>	Gets the last page on the PageLast stack.
<u>PagePopUp</u>	Displays the pop up window at the mouse position.
<u>PagePrev</u>	Displays the previous graphics page.
<u>PageProcessAnalyst</u>	Displays a Process Analyst page (in the same window) pre-loaded with the pre-defined Process Analyst View (PAV) file.
<u>Page-ProcessAnalystPens</u>	Display a page and add the specified pens to the first pane of the specified PA object on the page.
<u>PagePushLast</u>	Puts a page on the end of the PageLast stack.
<u>PageRecall</u>	Displays the page at a specified depth in the stack of previously displayed pages.
<u>PageRichTextFile</u>	Used as a page entry function - creates a rich text object, and loads a rich text file into that object.
<u>PageSelect</u>	Displays a dialog box with a list of graphics pages defined in the project.
<u>PageSetInt</u>	Stores a local page-based integer.
<u>PageSetStr</u>	Stores a local page-based string.
<u>PageSummary</u>	Displays a category of alarm summary entries on the predefined alarm page.
<u>PageTask</u>	Used for running preliminary Cicode before displaying a page in

a window.

PageTransformCoords	Converts Page coordinates to absolute screen coordinates.
-------------------------------------	---

PageTrend	Displays a standard trend page in a single cluster system.
---------------------------	--

PageTrendEx	Displays a standard trend page in a multi-cluster system.
-----------------------------	---

See Also

[Functions Reference](#)

PageAlarm

Displays a category of active alarms on the alarm page.

To use this function, you need to have a page in your project that was created using the Alarm template. By default, the name of the page is expected to be "Alarm". However, you can use an alarm page with a different name by adjusting the setting for the INI parameter [Page]AlarmPage.

Notes

- The operation of this function has changed. In Version 2.xx this function displayed the built-in alarm page from the Include project.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageAlarm([Category])

Category:

The alarm category to display. Set to 0 (the default) to display all alarm categories.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageDisabled](#), [PageHardware](#)

Example

System Keyboard	
Key Sequence	AlarmPage
Command	PageAlarm(0)
Comment	Display active alarms on the alarm page
System Keyboard	
Key Sequence	Alarm # ## Enter
Command	PageAlarm(Arg1)
Comment	Display a specified category of active alarms on the alarm page

See Also

[Page Functions](#)

PageBack

Displays the previously displayed page in the Window.

Syntax

PageBack([*iCount*])

iCount:

Optional parameter to specify the number of pages to move back. Default value is 1.

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if current window handle does not correspond to a valid window. CT_ERROR_INVALID_ARG (274) if count is outside of allowable bounds.

Related Functions

[PageForward](#), [PageHistoryDspMenu](#), [PageHistoryEmpty](#)

See Also

[Page Functions](#)

PageDisabled

Displays a category of disabled alarms on the disabled alarms page.

To use this function, you need to have a page in your project that was created using the Disabled template. By default, the name of the page is expected to be "Disabled". However, you can use a page with a different name by adjusting the setting for the INI parameter [Page]DisabledPage.

Notes

- The operation of this function has changed. In Version 2.xx this function displayed the built-in disabled alarm page from the Include project.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageDisabled([Category])

Category:

The alarm category to display. Set to 0 (the default) to display all alarm categories.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageAlarm](#)

Example

System Keyboard	
Key Sequence	DisabledPage
Command	PageDisabled(0)
Comment	Display disabled alarms on the alarm page
System Keyboard	
Key Sequence	Disabled # ## Enter

Command	PageDisabled(Arg1)
Comment	Display a specified category of disabled alarms on the alarm page

See Also[Page Functions](#)

PageDisplay

Displays a graphics page in the active window. The page needs to be in one of the operator's current areas. The page to be displayed is identified by its Page Name or the Page Number.

When this function is executed, it tests whether or not the identified page is based on a CSV_Include project template. If it is, the function uses TaskNew to run a CSV version of the PageDisplay function, "CSV_MM_PageDisplay". This confirms if multi-monitor support is required. As TaskNew is used to execute this function, no return value becomes available to PageDisplay. Under these circumstances, PageDisplay will return zero (0).

If this function is called to change the page in a pop-up window of a CSV_Include project, the page displayed by the base window will change. To change this default behaviour, set the [\[Page\]NewPageInBase](#) parameter to 0. Then a pop-up window can be changed with this function in single monitor mode.

You can specify if the page operates within the context of a particular cluster in a multiple cluster project. When the page launches during runtime, the ClusterName argument is used to resolve any tags that have a cluster omitted.

CitectSCADA stores the current page (onto a stack) before it displays the required page. You can call PageLast() to re-display the pages on the stack.

You cannot call this function from the Exit command field (see Page Properties) or a Cicode Object.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageDisplay(*Page*,[*ClusterName*])

Page:

The name or page number of the page to display (in quotation marks ""). Can be prefixed by the name of a host cluster, that is "ClusterName.Page". This will take precedence over the use of the ClusterName parameter if the two differ.

ClusterName:

The name of the cluster that will accommodate the page at runtime (in quotation marks ""). The specified cluster is used to resolve any tags that have a cluster omitted. If the Page parameter is prefixed with the name of a cluster, this parameter will not be used. This parameter is optional, however if you omit a cluster context in the Page properties, then any tags which omit an explicit Cluster. Tag-Name will be ambiguous and become unresolved if you have multiple clusters defined in the project.

Return Value

0 (zero) if the page is successfully displayed, otherwise an [error](#) is returned.

If the page is based on a CSV_Include project template, the function will return 0 (zero) as a CSV version of the function is executed via TaskNew.

Related Functions

[PageGoto](#), [PageLast](#)

Example

Buttons	
Text	Mimic Page
Command	PageDisplay("Mimic")
Comment	Display the "Mimic" page
System Keyboard	
Key Sequence	Page ##### Enter
Command	PageDisplay(Arg1)
Comment	Display a specified page

```
PageDisplay("MIMIC1");
! Displays page "MIMIC1".
PageDisplay("MIMIC2");
/* Displays page "MIMIC2" and places page "MIMIC1" onto the
```

```
PageLast stack. */
PageDisplay("10");
/* Displays page "10" and places page "MIMIC2" onto the PageLast
stack. */
PageLast();
/* Displays the last page on the stack, that is page "MIMIC2" and
removes it from the stack. */
```

Note: Before CitectSCADA version 5.0, page records could be edited in the Project Editor. One of the fields available for configuration was "Page Number". The value entered for a page could then be used in runtime with the Page Cicode functions such as `PageDisplay()`, `PageGoto()`, and `PageInfo(1)`.

For example, `PageDisplay("1")` can be used to display the page that has "1" (without the quotes) set in the **Page Number** field. `PageInfo(1)` returns the Page Number of the current page.

From version 5.0 on, this feature is only backwards-supported. The "Alias" field in the project Pages.DBF file still contains the Page Numbers from upgraded projects; however, the Pages database records are no longer available for direct editing in CitectSCADA.

See Also

[Page Functions](#)

[Cluster Context](#)

[Improved Client Side Online Changes](#)

PageFile

Displays a file on the page. After the file is displayed, you can scroll up and down through the file. To use this function, you need to use the Graphics Builder to create a page called "File" (using the file template).

Notes

- The operation of this function has changed. In Version 2.xx this function displayed the built-in file page from the Include project.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageFile(*sName*)

sName:

The name of the file to display.

Return Value

0 (zero) if the file is successfully displayed, otherwise an [error](#) is returned.

Related Functions

[DspFile](#)

Example

System Keyboard	
Key Sequence	File ##### Enter
Command	PageFile(Arg1)
Comment	Display the specified file on the page

See Also

[Page Functions](#)

PageFileInfo

Returns the width or height of an unopened page.

Syntax

PageFileInfo(*sPageName*, *nMode*)

sPageName:

The name of the page you would like to retrieve size information for.

nMode:

Retrieves either the width or the height of the specified page in pixels.

0 - returns the page width

1 - returns the page height

Return Value

The height or width of the specified page in pixels, depending on the value set for *nMode*.

See Also

[Page Functions](#)

PageForward

If PageBack() is called, PageForward() will restore the previously displayed page in the window. PageForward requires PageBack to have been called and no other page display functions to have been called in between. Calling the display functions PageDisplay or PageGoto will remove the forward pages from the display stack.

Syntax

PageForward([*iCount*])

iCount:

Optional parameter to specify the number of pages to move forward. Default value is 1.

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if current window handle does not correspond to a valid window. CT_ERROR_INVALID_ARG (274) if count is outside of allowable bounds.

Related Functions

[PageBack](#), [PageHistoryDspMenu](#), [PageHistoryEmpty](#)

See Also

[Page Functions](#)

PageGetInt

Returns the integer value associated with a variable name on a particular page. If two or more windows are displayed, each window has a unique copy of the variable. You can use these variables in Cicode to keep track of variables unique to each window.

Notes

- You can find out the current setting for [Page]ScanTime parameter, by calling this function as follows: PageGetInt(-2).
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageGetInt(*sLabel* [, *iWinNum*])

sLabel:

String name of the variable to return

iWinNum:

Window number of the page. Default is current window.

Return Value

Integer stored in variable sLabel.

Related Functions

[PageSetInt](#), [PageGetStr](#), [PageSetStr](#)

See Also

[Page Functions](#)

PageGetStr

Gets the string associated with a variable name on a particular page. Page-based variables are local to each display page. If two or more windows are displayed, each window has a unique copy of the variable. You can use these variables in Cicode to keep track of variables unique to each window.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageGetStr(*sLabel* [, *iWinNum*])

sLabel:

String name of the variable to return

iWinNum:

Window number of the page. Default is current window.

Return Value

Value stored in variable sLabel.

Related Functions

[PageSetInt](#), [PageGetInt](#), [PageSetStr](#)

See Also

[Page Functions](#)

PageGoto

Displays a graphics page in the active window. The page needs to be in one of the operator's current areas. You can specify either the Page Name or the Page Number of the graphics page.

You can also specify if the page operates within the context of a particular cluster in a multiple cluster project. When the page launches during runtime, the ClusterName argument is used to resolve any tags that have the cluster name omitted.

This function is similar to PageDisplay(), however PageGoto() does not put the current page onto the PageLast stack.

You cannot call this function from the Exit command field (see Page Properties) or a Cicode Object.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageGoto(*Page*,*ClusterName*)

Page:

The name or page number of the page to display (in quotation marks ""). Can be prefixed by the name of a host cluster, that is "ClusterName.Page". This will take precedence over the use of the ClusterName parameter if the two differ.

ClusterName:

The name of the cluster that will accommodate the page at runtime (in quotation marks ""). The specified cluster is used to resolve any tags that have the cluster name omitted. If the Page parameter is prefixed with the name of a cluster, this parameter will not be used.

Return Value

0 (zero) if the page is successfully displayed, otherwise an [error](#) is returned.

Related Functions

[PageDisplay](#)

Example

```
PageDisplay("MIMIC1");
```

```

! Displays page "MIMIC1".
PageDisplay("MIMIC2");
/* Displays page "MIMIC2" and places page "MIMIC1" onto the
PageLast stack. */
PageGoto("10");
/* Displays page "10". Page "MIMIC2" is not placed onto the
PageLast stack. */

```

Note: Before CitectSCADA version 5.0, page records could be edited in the Project Editor. One of the fields available for configuration was "Page Number". The value entered for a page could then be used in runtime with the Page Cicode functions such as `PageDisplay()`, `PageGoto()`, and `PageInfo(1)`.

For example, `PageDisplay("1")` can be used to display the page that has "1" (without the quotes) set in the **Page Number** field. `PageInfo(1)` returns the Page Number of the current page.

From version 5.0 on, this feature is only backwards-supported. The "Alias" field in the project Pages.DBF file still contains the Page Numbers from upgraded projects; however, the Pages database records are no longer available for direct editing in CitectSCADA.

See Also

[Page Functions](#)

[Improved Client Side Online Changes](#)

PageHardware

Displays the active hardware alarms on the hardware alarms page.

To use this function, you need to have a page in your project that was created using the Hardware template. By default, the name of the page is expected to be "Hardware". However, you can use a page with a different name by adjusting the setting for the INI parameter [Page]HardwarePage.

Notes

- The operation of this function has changed. In Version 2.xx this function displayed the built-in hardware alarm page from the Include project.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

`PageHardware()`

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageAlarm](#)

Example

System Keyboard	
Key Sequence	Hardware
Command	PageHardware()
Comment	Display active hardware alarms on the hardware alarms page

See Also

[Page Functions](#)

PageHistoryDspMenu

Displays a pop-up menu which lists the page history of current window. The user can select any page in the history to recall the page. When full page history is specified, the currently displayed page will also be listed and marked in the menu.

Syntax

PageHistoryDspMenu([*iType*])

iType:

The type of page history to be listed:

0 - full history (default)

1 - back history

2 - forward history

Return Value

Zero (0) if the function is executed successfully. Otherwise an error is returned.

Related Functions

[PageBack](#), [PageForward](#), [PageHistoryEmpty](#)

See Also[Page Functions](#)

PageHistoryEmpty

Used to determine if the page history of the current window is empty. The currently displayed page is not counted as history.

Syntax

PageHistoryEmpty([*iType*])

iType:

The type of page history to be checked:

- 0 - full history (default)
- 1 - back history
- 2 - forward history

Return Value

1 if page history of specified type is empty, or 0 if it is not empty.

Related Functions

[PageBack](#), [PageForward](#), [PageHistoryDspMenu](#)

See Also[Page Functions](#)

PageHome

Displays the predefined home page in the window.

Syntax

PageHome([*sCluster*])

sCluster:

Optional parameter to the Cluster to associate the page being opened with. Default value "".

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if current window handle does not correspond to a valid window. CT_ERROR_INVALID_ARG (274) if INI parameter [Page]HomePage is not set.

See Also

[Page Functions](#)

PagelInfo

Gets information about the current page.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageInfo(*Type*)

Type:

The type of page information required:

0 - Page Name

1 - Page Number

2 - Page Title

3 - Display filename

4 - Symbol filename

5 - Next Page Name

6 - Previous Page Name

7 - Previous display count, incremented at each page scan. The page scan rate defaults to the value of the Citect.ini parameter [Page]ScanTime, and can be overridden per page by changing the scan time setting in the General tab of the page properties in Graphics Builder.

8 - Parent window number. Returns -1 if there is no parent

9 - First child window number. Returns -1 if there are no children

10 - Next child in child link. Returns -1 for the end of the list

11 - Window mode (set by the WinNewAt() function)

12 - Width of window

13 - Height of window

14 - X position of window

15 - Y position of window

16 - Dynamic window horizontal scale

17 - Dynamic window vertical scale

Types 16 and 17 return a real number between 0 and 1 (as a STRING) and will be identical, as the dynamic scaling does not allow a change in the aspect ratio.

18 - Flashing color state. Type 18 returns one of the following:

- "0" - the palette does not flash
- "1" - the palette is primary now
- "2" - the palette is secondary now

19 - In animation cycle. Returns a 1 (true) or 0 (false)

20 - In communications cycle. Returns a 1 (true) or 0 (false)

21 - Width of background page

22 - Height of background page

23 - The number of animation points on a page

24 - The value of the highest animation point on the page

25 - Indicates when the page's "On Page Shown" event has been triggered.
Returns 1 if triggered, 0 if it has not.

26 - The cluster that has been specified to host the page. Returns the cluster name, or an empty string if no cluster has been specified.

27 - Indicates whether the Cicode library used by the page is different from the currently loaded library. Returns 1 if different, 0 if the versions are the same.

28 – Return X Coordinate of Client rectangle origin.

29 – Return Y Coordinate of Client rectangle origin.

Return Value

The information (as a string).

Related Functions

[PageDisplay](#)

Example

```
! If currently on page "MIMIC1";
Variable=PageInfo(0);
! Sets Variable to "MIMIC1".
```

Note: Before CitectSCADA version 5.0, page records could be edited in the Project Editor. One of the fields available for configuration was "Page Number". The value entered for a page could then be used in runtime with the Page Cicode functions such as `PageDisplay()`, `PageGoto()`, and `PageInfo(1)`.

For example, `PageDisplay("1")` can be used to display the page that has "1" (without the quotes) set in the **PageNumber** field. `PageInfo(1)` returns the Page Number of the current page.

From version 5.0 on, this feature is only backwards-supported. The "Alias" field in the project Pages.DBF file still contains the Page Numbers from upgraded projects; however, the Pages database records are no longer available for direct editing in CitectSCADA.

See Also

[Page Functions](#)

PageLast

Displays the graphics page that was last displayed. With this function, you can successively recall the last ten pages that were displayed.

Graphics pages displayed using this command cannot be subsequently recalled.

You cannot call this function from the Exit command field (see Page Properties) or a Cicode Object.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

`PageLast()`

Return Value

0 (zero) if the page is successfully displayed, otherwise an [error](#) is returned.

Related Functions

[PagePeekLast](#), [PagePopLast](#), [PagePushLast](#)

Example

Buttons

Text	Last Page
Command	PageLast()
Comment	Display the graphics page that was last displayed

```

PageDisplay("MIMIC1");
! Displays page "MIMIC1".
PageDisplay("MIMIC2");
/* Displays page "MIMIC2" and places page "MIMIC1" onto the
PageLast stack. */
PageDisplay("10");
! Displays page "10" and places page "MIMIC2" onto the PageLast
stack.
PageLast();
/* Displays the last page on the stack, that is page "MIMIC2" and
removes it from the stack. */
PageLast();
/* Displays the last page on the stack, that is page "MIMIC1" and
removes it from the stack. */
PageLast();
/* Returns an "Out of range" error code as there are no more pages
on the stack.*/

```

See Also[Page Functions](#)**PageMenu**

Displays a menu page with page selection buttons. A page goto button is displayed for each of the first 40 pages defined in the project.

Syntax[PageMenu\(\)](#)**Return Value**

0 (zero) if the page is successfully displayed, otherwise an [error](#) is returned.

Related Functions[PageGoto](#), [PageLast](#), [PagePrev](#), [PageSelect](#)

Example

Buttons	
Text	Menu
Command	PageMenu()
Comment	Display the menu page
System Keyboard	
Key Sequence	Menu
Command	PageMenu()
Comment	Display the menu page

See Also

[Page Functions](#)

PageNext

Displays the next page as specified in the project.

You cannot call this function from the Exit command field (see Page Properties) or a Cicode Object.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

`PageNext()`

Return Value

0 (zero) if the page is successfully displayed, otherwise an [error](#) is returned.

Related Functions

[PagePrev](#)

Example**Buttons**

Text	Page Next
Command	PageNext()
Comment	Display the next page in the browse sequence

System Keyboard

Key Sequence	PageNext
Command	PageNext()
Comment	Display the next page in the browse sequence

```
/* If graphics page 1 is currently displayed, and the graphics
page 1 has Next Page Name=10. */
PageNext();
! Displays graphics page 10.
```

See Also[Page Functions](#)**PagePeekCurrent**

Return the index in the page stack for the current page.

Syntax**PagePeekCurrent()****Return Value**

Index in the page stack for the current page. -1 indicates that current window handle does not correspond to a valid window.

Related Functions[PagePeekLast](#), [PagePopLast](#), [PagePopUp](#), [PagePushLast](#), [PageRecall](#)**See Also**[Page Functions](#)

PagePeekLast

Gets information about a Page at an offset in the PageLast stack (without removing the page from the stack).

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PagePeekLast(*iOffset* [, *iType*])

iOffset:

The offset from the end of the PageLast stack. Offset 0 is the last page on the stack, Offset 1 is the second last page on the stack, etc

iType:

An enumeration representing the type of information required. The default value is 0.

0 - Page Name

1 - Configured Page Title

2 - Active Page Title

Return Value

String value of the requested information, or empty string if no valid result for given arguments.

Related Functions

[PagePeekCurrent](#), [PagePopLast](#), [PagePopUp](#), [PagePushLast](#)

See Also

[Page Functions](#)

PagePopLast

Gets the Page Name of the last item on the PageLast stack and removes the page from the stack.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax**PagePopLast()****Return Value**

The page name or an empty string if there is no last page.

Related Functions

[PageLast](#), [PagePeekCurrent](#), [PagePeekLast](#), [PagePopUp](#), [PagePushLast](#)

Example

```
PageDisplay("MIMIC1");
! Displays page "MIMIC1".
PageDisplay("MIMIC2");
/* Displays page "MIMIC2" and places page "MIMIC1" onto the
PageLast stack. */
PageDisplay("10");
! Displays page "10" and places page "MIMIC2" onto the PageLast
stack.
Variable=PagePopLast();
/* Sets Variable to "MIMIC2" and removes "MIMIC2" from the
PageLast stack. */
PageLast();
! Displays page "MIMIC1".
```

See Also

[Page Functions](#)

PagePopUp

Display pop up window at the mouse position. If the mouse position is not known then the pop up will display in the centre of the screen. The window is displayed with no resize and will be closed if the page is changed.

Syntax**PagePopUp(*sPage*)***sPage:*

The name of the page (drawn with the Graphics Builder).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Note: Before CitectSCADA version 5.0, page records could be edited in the Project Editor. One of the fields available for configuration was "Page Number". The value entered for a page could then be used in runtime with the Page Cicode functions such as `PageDisplay()`, `PageGoto()`, and `PageInfo(1)`.

For example, `PageDisplay("1")` can be used to display the page that has "1" (without the quotes) set in the **Page Number** field. `PageInfo(1)` returns the Page Number of the current page.

From version 5.0 on, this feature is only backwards-supported. The "Alias" field in the project Pages.DBF file still contains the Page Numbers from upgraded projects; however, the Pages database records are no longer available for direct editing in CitectSCADA.

Related Functions

[PageLast](#), [PagePeekCurrent](#), [PagePeekLast](#), [PagePopLast](#), [PagePushLast](#)

See Also

[Page Functions](#)

PagePrev

Displays the previous page as specified in the project.

You cannot call this function from the Exit command field (see Page Properties) or a Cicode Object.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

`PagePrev()`

Return Value

0 (zero) if the page is successfully displayed, otherwise an [error](#) is returned.

Related Functions

[PageNext](#)

Example

Buttons

Text	Page Previous
Command	PagePrev()
Comment	Display the previous page in the browse sequence

System Keyboard

Key Sequence	PagePrev
Command	PagePrev()
Comment	Display the previous page in the browse sequence

```
/* If graphics page 10 is currently displayed, and graphics page
10 has Prev Page Name=1. */
PagePrev();
! Displays graphics page 1.
```

See Also[Page Functions](#)**PageProcessAnalyst**

Displays a Process Analyst page (in the same window) preloaded with the pre-defined Process Analyst View (PAV) file.

Syntax

PageProcessAnalyst(*sPage*, *sPAVFile1* [, *iFileLocation1* [, *iButtonMask1* [, *sObjName1* [, *sPAVFile2* [, *iFileLocation2* [, *iButtonMask2* [, *sObjName2*]]]]]])

sPage:

The name of the page that contains Process Analyst object(s). For example, pages based on the Process Analyst templates found in the Tab_Style_Include project.

sPAVFile1:

Name of the 1st PAV file

iFileLocation1:

PAV file location code for the 1st PAV file, see PA doc LoadFromFile() for details.

iButtonMask1:

Bit mask for removing command buttons from the 1st PA, bit flags as shown below:

- 1 - Load View
- 2 - Save View
- 4 - Print
- 8 - Copy to Clipboard
- 16 - Copy to File
- 32 - Add Pens
- 64 - Remove Pens
- 128 - Show Properties
- 256 - Help

sObjName1:

Name of the PA object on the given Page where the 1st PAV file will be loaded. If this parameter is not specified or empty string, it is defaulted to the object name used in the tab style templates, that is "_templatePA1".

sPAVFile2:

Name of the 2nd PAV file

iFileLocation2:

PAV file location code for the 2nd PAV file

iButtonMask2:

Bit mask for removing command buttons from the 2nd PA, refer *iButtonMask1* for details

sObjName2:

Name of the PA object on the given Page where the 2nd PAV file will be loaded. If this parameter is not specified or empty string, it is defaulted to the object name used in the tab style templates, that is "_templatePA2".

Return Value

Zero (0) if the page is successfully displayed. Otherwise an error is returned.

Related Functions

[PageProcessAnalystPens](#), [ProcessAnalystLoadFile](#), [ProcessAnalystPopUp](#), [ProcessAnalystSelect](#), [ProcessAnalystSetPen](#), [ProcessAnalystWin](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Page Functions](#)

PageProcessAnalystPens

Display a page and add the specified pens to the first pane of the specified PA object on the page. If a PAV file is also specified, it will be loaded first, and the pens in the first pane will be removed before the specified pens are created on the PA.

Syntax

PageProcessAnalystPens(*sPage*, *sTag1* [, *sTag2..sTag8* [, *iButtonMask* [, *sObjName* [, *iPane* [, *sPAVFile* [, *iFileLocation*]]]]])

sPage:

The name of the page that displays the PA.

sTag1..sTag8:

Up to 8 Trend tags can be added to the PA.

iButtonMask:

Mask to remove button(s) from the main tool bar of PA. The following values can be combined to remove multiple buttons:

- 1 - Load View
- 2 - Save View
- 4 - Print
- 8 - Copy to Clipboard
- 16 - Copy to File
- 32 - Add Pens
- 64 - Remove Pens
- 128 - Show Properties
- 256 - Help

sObjName

The name of the PA object. If not specified it is defaulted to "_templatePA1" which is the name used by the built-in templates.

iPane

The pane in PA where the trend or variable tags are added. If this is not specified or less than 1, it is defaulted to 1 (the 1st pane). If the specified pane does not exist in the PA object, a new pane will be created.

sPAVFile:

Optional Process Analyst View file to be loaded, default = "".

iFileLocation:

Optional location of the PAV file. The allowed values are:

- 0 - Analyst Views subfolder in project folder (default)
- 1 - Folders specified in properties primary/standby server path
- 2 - My documents folder

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageProcessAnalyst](#), [ProcessAnalystLoadFile](#), [ProcessAnalystPopUp](#), [ProcessAnalystSelect](#), [ProcessAnalystSetPen](#), [ProcessAnalystWin](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Page Functions](#)

PagePushLast

Places a page at the end of the PageLast stack.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PagePushLast(*Page*)

Page:

The Page Name or Page Number (of the page) to place at the end of the PageLast stack.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageLast](#), [PagePeekCurrent](#), [PagePeekLast](#), [PagePopLast](#), [PagePopUp](#)

Example

```
PageDisplay("MIMIC1");
! Displays page "MIMIC1".
PageDisplay("MIMIC2");
/* Displays page "MIMIC2" and places page "MIMIC1" onto the
```

```

PageLast stack. */
PageDisplay("10");
! Displays page "10" and places page "MIMIC2" onto the PageLast
stack.
PagePushLast("TREND1");
! Places page "TREND1" onto the PageLast stack.
PageLast();
/* Displays the last page on the stack, that is page "TREND1" and
removes it from the stack. */
PageLast();
/* Displays the last page on the stack, that is page "MIMIC2" and
removes it from the stack. */

```

See Also[Page Functions](#)**PageRecall**

Displays the page at a specified depth in the stack of previously displayed pages.

Syntax**PageRecall(*iIndex*)***iIndex*:

The index into the Page History Stack of the Page to be displayed. To get the index for the currently displayed page, call PagePeekCurrent(). Then increment it to access pages in the forward history, or decrement it to access pages in the backward history. Be reminded that you cannot recall the page that is currently displayed.

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if current window handle does not correspond to a valid window. CT_ERROR_INVALID_ARG (274) if index is outside of allowable bounds.

Related Function[PagePeekCurrent](#)**See Also**[Page Functions](#)**PageRichTextFile**

This function creates a rich edit object, and loads a copy of the rich text file *Filename* into that object. The rich text object will be rectangular in shape, with dimensions determined by *nHeight*, and *nWidth*. If you do not specify *nHeight* and *nWidth*, *AN* will define the position of one corner, and (*AN* + 1) the position of the diagonally opposite corner. This function would often be used as a page entry function.

Syntax

PageRichTextFile(AN, Filename, nMode [, nHeight] [, nWidth])

AN:

The animation point at which to display the rich text object.

Filename:

The name of the file to be copied and loaded into the rich text object. The filename needs to be entered in quotation marks "".

If you are loading a copy of an RTF report, the report needs to have already been run and saved to a file.

Remember that the filename for the saved report comes from the File Name field in the Devices form. The location of the saved file needs to also be included as part of the filename. For example, if the filename in the Devices form listed [Data];RepDev.rtf, then you would need to enter "[Data]\r-epdev.rtf" as your argument. Alternatively, you can manually enter the path, for example, "c:\My-Application\data\repdev.rtf".

If you are keeping a number of history files for the report, instead of using the rtf extension, you need to change it to reflect the number of the desired history file, for example, 001.

nMode:

The display mode for the rich text object. The mode can be any combination of:

0 - Disabled - should be used if the rich text object is to be used for display purposes only.

1 - Enabled - allows you to select and copy the contents of the RTF object (for instance an RTF report), but you will not be able to make changes.

2 - Read/Write - allows you to edit the contents of the RTF object. Remember, however, that the object needs to be enabled before it can be edited. If it has already been enabled, you can just enter Mode 2 as your argument. If it is not already enabled, you will need to enable it. By combining Mode 1 and Mode 2 in your argument (3), you can enable the object, and make it read/write at the same time.

Because the content of the rich text object is just a copy of the original file, changes will not affect the actual file, until saved using the DspRichTextSave function.

nHeight:

The height of the rich text object in pixels. The height is established by measuring down from the animation point.

If you do not enter a height and width, the size of the object will be determined by the position of *AN* and *AN+1*.

nWidth:

The width of the rich text object in pixels. The width is established by measuring across to the right of the animation point.

If you do not enter a height and width, the size of the object will be determined by the position of *AN* and *AN+1*.

Return Value

None

Related Functions

[DspRichText](#), [DspRichTextLoad](#), [DspRichTextEdit](#), [DspRichTextEnable](#), [DspRichTextGetInfo](#), [DspRichTextPgScroll](#), [DspRichTextPrint](#), [DspRichTextSave](#), [DspRichTextScroll](#), [FileRichTextPrint](#)

Example

```
PageRichTextFile(108,"f:\citect\data\richtext.rtf",0);
// This function would produce a rich text object at animation
point 108. Into this object a copy of f:\citect\data\richtext.rtf
would then be loaded. Remember, richtext.rtf is the name of the
output file for the report, as specified in the Devices form.
Because 0 was specified as the nMode for this example, the
contents of this object will be display only. //
PageRichTextFile(53,"[Data]\richtext.rtf",1);
//This function would produce a rich text object at animation
point 53. Into this object a copy of [Data]\richtext.rtf would
then be loaded. It will be possible to select and copy the
contents of the object, but not make changes. //
```

See Also

[Page Functions](#)

PageSelect

Displays a dialog box with a list of graphics pages defined in the project. AN operator can select a page name for display.

Syntax

PageSelect()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageGoto](#), [PageLast](#), [PagePrev](#), [PageMenu](#)

Example

Buttons

Text	Page Select
Command	PageSelect()
Comment	Display the page selection dialog box

```
PageSelect();  
! Displays the page selection dialog box.
```

See Also

[Page Functions](#)

PageSetInt

Associates an integer variable with a particular page. Page-based variables are stored in an array, local to each display page. This function allows you to save integer variables in temporary storage.

Notes

- You can dynamically change the setting for [Page]ScanTime parameter, by calling this function as follows: PageSetInt(-2, <scantime>).
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageSetInt(*sLabel*, *sVar* [, *iWinNum*])

sLabel:

String name of the variable which will contain *sValue*.

sVar:

The integer to store.

iWinNum:

Window number of the page. Default is current window.

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if WinNum handle does not correspond to a valid window. CT_ERROR_INVALID_ARG (274) if Label or Var is not a valid variable.

Related Functions

[PageGetInt](#), [PageGetStr](#), [PageSetStr](#)

See Also

[Page Functions](#)

PageSetStr

Stores a local page-based string and associates the string with the page. Page-based variables are stored in an array, local to each display page. This function allows you to save string variables in temporary storage.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageSetStr(*sLabel*, *sVar* [, *iWinNum*])

sLabel:

String name of the variable which will contain *sValue*.

sVar:

The string to store. The string length is 128 characters.

iWinNum:

Window number of the page. Default is current window.

Return Value

CT_ERROR_NO_ERROR (0) on success. CT_ERROR_BAD_HANDLE (269) if WinNum handle does not correspond to a valid window. CT_ERROR_INVALID_ARG (274) if Label or Var is not a valid variable.

Related Functions

[PageGetInt](#), [PageGetStr](#), [PageSetInt](#)

See Also

[Page Functions](#)

PageSummary

Displays a category of alarm summary entries on the alarms summary page.

To use this function, you need to have a page in your project that was created using the Summary template. By default, the name of the page is expected to be "Summary". However, you can use an alarm page with a different name by adjusting the setting for the INI parameter [Page]SummaryPage.

Notes

- The operation of this function has changed. In Version 2.xx this function displayed the built-in summary alarm page from the Include project.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageSummary(*Category*)

Category:

The category number for the alarms you want to summarise.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageAlarm](#)

Example

System Keyboard	
Key Sequence	SummaryPage
Command	PageSummary(0)
Comment	Display alarm summary entries on the alarm page
System Keyboard	
Key Sequence	Summary ### Enter
Command	PageSummary(Arg1)
Comment	Display a specified category of alarm summary entries on the alarm page

See Also

[Page Functions](#)

PageTask

PageTask() is used for running preliminary Cicode before displaying a page in a window. It makes it possible for the same Cicode to be run if the page is re-entered by navigating forward or back. PageTask() is similar to TaskNew().

PageTask() returns a handle to a code task the first time it is run. The custom Cicode of the sFunctionName parameter needs to call PageDisplay() in order to display the page. When the page changes, the function and its parameters will be pushed onto the Page History stack. The Cicode fnTask will be called again when the page is navigated to using the PageForward or PageBackward functions.

Syntax

PageTask(*iWinNum*, *sFunctionName*, *sFunctionArg*)

iWinNum:

The Window number of the window in which to display the page.

sFunctionName:

String representing the Cicode function to run each time the page is navigated to using the forward and backward navigation functions.

sFunctionArg:

String representing the parameters to use with function fnTask.

Return Value

A handle to a code task the first time it is run. BAD_HANDLE (-1) if the function did not complete.

Related Functions

[PageBack](#), [PageForward](#)

Example

```
FUNCTION MyTrendDisplay(INT Area)
    ConfigurePens(area)
    PageDisplay("MyTrendPage")
END
```

See Also

[Page Functions](#)

PageTransformCoords

Converts Page coordinates to absolute screen coordinates.

Syntax

PageTransformCoords(*hPage*, *iPageX*, *iPageY*, *iDisplayX*, *iDisplayY*, *iType*)

hPage:

Page handle of the relevant Window.

PageX:

X coordinate of page coordinate.

iPageY:

Y coordinate of page coordinate.

iDisplayX:

Output parameter – Transformed X coordinate.

iDisplayY:

Output parameter – Transformed Y coordinate.

iType:

The value of output coordinate:

- 0 - Absolute screen coordinates
- 1 - Coordinates relative to Window origin
- 2 - Coordinates relative to Client rectangle origin

Return Value

0 – Success

269 – ERROR_BAD_HANDLE – hPage is not a valid Page handle

274 – ERROR_INVALID_ARG – either DisplayX, DisplayY or Type is invalid.

Related Functions

[PageInfo](#), [DspGetMouse](#), [DspAnGetPo](#), [DspGetAnExtent](#)

See Also

[Page Functions](#)

PageTrend

Displays a trend page with the specified trend pens. Use this function to display trends in a single cluster system with a single trend page. You need to create the trend page with the Graphics Builder and set the pen names to blank. Then display that page by calling this function and passing the required trend tags. Call this function from a menu of trend pages.

Note: Because you cannot mix templates in a project, PageTrend will not work if you have included the CSV_Include project in your project. For example, if your project includes the CSV_Include project, PageTrend("pagename", "trendtag") only works on trend pages based on XP-style templates (that is, "Trend"). When using PageTrend to go to a page based on a standard template (for example, "SingleTrend"), the page displays, but no trend tag is added. This also applies for the CSV_Trend_Page function.

This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised. For a multi-cluster system use [PageTrendEx](#).

Syntax

PageTrend(*sPage*, *sTag1* [*, sTag2..sTag8*])

sPage:

Name of the trend page (drawn with the Graphics Builder).

sTag1:

The first trend tag to display on the page.

sTag2..sTag8:

Optionally trend tags 2 to 8 to display on the page.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Note: Before CitectSCADA version 5.0, page records could be edited in the Project Editor. One of the fields available for configuration was "Page Number". The value entered for a page could then be used in runtime with the Page Cicode functions such as `PageDisplay()`, `PageGoto()`, and `PageInfo(1)`.

For example, `PageDisplay("1")` can be used to display the page that has "1" (without the quotes) set in the **Page Number** field. `PageInfo(1)` returns the Page Number of the current page.

From version 5.0 on, this feature is only backwards-supported. The "Alias" field in the project Pages.DBF file still contains the Page Numbers from upgraded projects; however, the Pages database records are no longer available for direct editing in CitectSCADA.

Related Functions

[TrnNew](#), [TrnSelect](#), [TrendWin](#), [TrendPopUp](#), [PageTrendEx](#)

Example

Buttons

Text	Process Trend
Command	<code>PageTrend("MyTrend", "PV1", "PV2", "PV3")</code>
Comment	Display the trend page with three trend pens

```
PageTrend("MyTrend", "PV1", "PV2", "PV3")
/* Display three trend tags on a single trend page. */
```

See Also

[Page Functions](#)

PageTrendEx

Displays a trend page of a specified cluster in a multi-cluster system with the specified trend pens. Use this function to display trends in a multi-cluster system with a single trend page. You need to create the trend page with the Graphics Builder and set the pen names to blank. Then display that page by calling this function and passing the required trend tags. Call this function from a menu of trend pages. This function can also be used in a single cluster system, the *sCluster* argument is optional in such a case.

Note: Because you cannot mix templates in a project, PageTrendEx will not work if you have included the CSV_Include project in your project. For example, if your project includes the CSV_Include project, PageTrend("pagename", "trendtag") only works on trend pages based on XP-style templates (that is, "Trend"). When using PageTrend to go to a page based on a standard template (for example, "SingleTrend"), the page displays, but no trend tag is added. This also applies for the CSV_Trend_Page function.

This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

PageTrendEx(*sPage*, *sCluster*, *sTag1* [*sTag2..sTag8*])

sPage:

Name of the trend page (drawn with the Graphics Builder).

sCluster:

Name of the cluster in which the trend page is located.

sTag1:

The first trend tag to display on the page.

sTag2..sTag8:

Optionally trend tags 2 to 8 to display on the page.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Note: Before CitectSCADA version 5.0, page records could be edited in the Project Editor. One of the fields available for configuration was "Page Number". The value entered for a page could then be used in runtime with the Page Cicode functions such as `PageDisplay()`, `PageGoto()`, and `PageInfo(1)`.

For example, `PageDisplay("1")` can be used to display the page that has "1" (without the quotes) set in the **Page Number** field. `PageInfo(1)` returns the Page Number of the current page.

From version 5.0 on, this feature is only backwards-supported. The "Alias" field in the project Pages.DBF file still contains the Page Numbers from upgraded projects; however, the Pages database records are no longer available for direct editing in CitectSCADA.

Related Functions

[TrnNew](#), [TrnSelect](#), [TrendWin](#), [TrendPopUp](#), [PageTrend](#)

Example

Buttons

Text	Process Trend
Com- mand	<code>PageTrendEx("MyTrend", "MyCluster", "PV1", "PV2", "PV3")</code>
Com- ment	Display the trend page on the specified cluster with three trend pens

```
PageTrendEx("MyTrend", "MyCluster", "PV1", "PV2", "PV3")
/* Display three trend tags on a single trend page on the
specified cluster. */
```

See Also

[Page Functions](#)

Chapter: 42 Plot Functions

With the plot functions, you can plot system data on the screen or on your system printer(s).

Plot Functions

Following are functions relating to plotting data:

PlotClose	Displays and/or prints the plot, then closes the plot.
PlotDraw	Draws a point, line, box, or circle on a plot.
PlotFile	This function is now obsolete.
Plot-GetMarker	Gets the marker number of a symbol that is registered as a marker.
PlotGrid	Draws gridlines to be used for plotted lines.
PlotInfo	Gets information about a plot.
PlotLine	Plots a line through a set of data points.
PlotMarker	Draws markers on a plotted line or at a specified point.
PlotOpen	Opens a new plot, sets its output device, and returns a plot handle for use by the other plot functions.
Plot-ScaleMarker	Draws scale lines (with markers) beside the grid on your plot (if there is one).
Plot-SetMarker	Sets (registers) a symbol as a marker.
PlotText	Draws text on a plot.
PlotXYLine	Draws an XY line through a set of data points.

See Also

[Functions Reference](#)

PlotClose

Displays the plot on screen or sends it to the printer (depending on the output device you specified in the PlotOpen() function), then closes the plot. Once the plot is closed, it cannot be used.

Syntax

PlotClose(*hPlot*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where all data on the plot is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotDraw](#), [PlotGetMarker](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotOpen](#), [PlotScaleMarker](#), [PlotSetMarker](#), [PlotText](#), [PlotXYLine](#), [TrnPlot](#)

Example

See [PlotOpen](#)

See Also

[Plot Functions](#)

PlotDraw

Constructs drawings on your plot. Use the coordinates ($X1, Y1$) and ($X2, Y2$) to define a point, line, rectangle, square, circle, or ellipse. You can specify the style, color, and width of the pen, and a fill color for a box or circular shape.

you need to call the PlotOpen() function first, to get the handle for the plot (*hPlot*) and to specify the output device.

Syntax

PlotDraw(*hPlot*, *Type*, *PenStyle*, *PenCol*, *PenWidth*, *nFill*, *X1*, *Y1*, *X2*, *Y2*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where data on the plot is stored.

Type:

The type of drawing:

- 1 - Rectangle or square
- 2 - Circle or ellipse
- 3 - Line
- 4 - Point

PenStyle:

The style of the pen used to draw:

- 0 - Solid
- 1 - Dash (- - - -)
- 2 - Dot (.....)
- 3 - Dash and dot (- . - . - . -)
- 4 - Dash, dot, dot (- . . - . - . -)
- 5 - Hollow

PenCol:

The color of the pen (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

PenWidth:

Pen width in pixels. If the width is thicker than one pixel, you need to use a solid pen (PenStyle = 0). Maximum width is 32.

nFill:

The fill color of the rectangle, square, circle, or ellipse (flashing color is not supported). Select a color from the list of predefined color names and codes or create an RGB-based color using the function [MakeCitectColor](#). For a point or line, nFill is ignored.

X1, Y1:

X and y coordinates (in pixels) of the upper-left corner of the drawing (the origin).

X2, Y2:

X and y coordinates (in pixels) of the lower-right corner of the drawing.

For a point, (X1,Y1) and (X2,Y2) are assumed to be the same, so (X2,Y2) is ignored. To draw a circle or ellipse, enter the coordinates for a square or rectangle; the circle or ellipse is automatically drawn within the box.

If the plot is for display on the screen, coordinates are relative to the AN specified in the PlotOpen() function. If the output device is a printer, coordinates are relative to the point (0,0).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotClose](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotOpen](#), [PlotScaleMarker](#), [PlotText](#), [PlotXYLine](#), [TrnPlot](#)

Example

See [PlotOpen](#).

See Also

[Plot Functions](#)

PlotGetMarker

Gets the marker number of a symbol. The symbol needs to be a symbol and registered with the PlotSetMarker() function.

Syntax

PlotGetMarker(*sSymbolName*)

sSymbolName:

The library name and symbol name ("Library.Symbol") of the symbol that is registered as a marker.

Return Value

The marker number if successful, otherwise -1 is returned.

Related Functions

[PlotMarker](#), [PlotScaleMarker](#), [PlotSetMarker](#)

Example

```
/*Assume that the symbol was registered by PlotSetMarker function */
PlotSetMarker(20,"Global.Hourglass");
/*Later on, this symbol can be used as shown below*/
hPlot = PlotOpen(36,"Display",1);
..
/* Display red hourglass as marker at point (100,200) on AN36. */
```

```

MarkerNo = PlotGetMarker("Global.Hourglass");
PlotMarker(hPlot,MarkerNo,red,1,1,100,200);
..
PlotClose(hPlot);

```

See Also

[Plot Functions](#)

PlotGrid

Defines a frame and draws horizontal and vertical grid lines within this frame. These grid lines can then be used by the PlotLine(), PlotXYLine(), and PlotScaleMarker() functions. You need to define the frame for a plot before you can plot points with the PlotLine() and PlotXYLine() functions. *nSamples* specifies the maximum number of samples that can be plotted for a single line. If you set *FrameWidth* to 0 (zero), the frame will be defined but not displayed (however, the plot will still be displayed).

You can specify the number of grid lines and their color, as well as the background color which will fill the frame. If *nHorGrid* and *nVerGrid* are set to 0 (zero), then the grid lines will not be drawn.

you need to call the PlotOpen() function, first, to get the handle for the plot (*hPlot*), and to specify the output device. Then call this function to set up the frame and grid. You can then call the PlotScaleMarker() function to draw scale lines beside the frame, and call the PlotLine() or PlotXYLine() to plot a set of data points.

Syntax

PlotGrid(*hPlot*, *nSamples*, *X1*, *Y1*, *X2*, *Y2*, *nHorGrid*, *HorGridCol*, *nVerGrid*, *VerGridCol*, *FrameWidth*, *FrameCol*, *nFill*, *nMode*)

hPlot:

Plot handle, returned from the PlotOpen() function. The plot handle identifies the table where data on the plot is stored.

nSamples:

The maximum number of samples that can be plotted for a single line in this grid (valid values between 2 and 16000 inclusive). For example, if you set *nSamples* to 10, then plot 2 lines in this grid (using the PlotLine() function), each line will be plotted with a maximum of 10 samples. For this example, a line can possess less than 10 samples, but if it has more, it will be shortened to 10 samples.

X1, *Y1*:

The x and y coordinates of the upper-left corner of the frame containing the grid lines.

X2, *Y2*:

The x and y coordinates of the lower-right corner of the frame containing the grid lines.

If the plot is for display on the screen, you should set $(X1, Y1)$ to $(0, 0)$. The origin of the frame is then positioned at the AN specified in the PlotOpen() function.

If the output device is a printer, both $(X1, Y1)$ and $(X2, Y2)$ are relative to the point $(0, 0)$.

nHorGrid:

The number of rows (formed by the horizontal grid lines) to draw within the frame. If there is no need of grid lines, set nHorGrid to 0 (zero) and HorGridCol to 0. *nHorGrid* cannot exceed the pixel width of the plot.

HorGridCol:

The color of the horizontal grid lines (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [Make-CitectColor](#).

nVerGrid:

The number of columns (formed by the vertical grid lines) to draw within the frame. If there is no need of grid lines, set nVerGrid to 0 (zero) and VerGridCol to 0. *nVerGrid* cannot exceed the pixel height of the plot.

VerGridCol:

The color of the vertical grid lines (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [Make-CitectColor](#).

FrameWidth:

The width (also called pen width) of the frame enclosing the grid, in pixels. To define the frame without drawing its boundaries, set FrameWidth to 0 (zero) and FrameCol to 0. The maximum is 32.

FrameCol:

The color of the frame enclosing the grid (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [Make-CitectColor](#).

nFill:

The background color for the frame (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [Make-CitectColor](#).

nMode:

The mode of the plot. For future use only - set it to 0 (zero).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotClose](#), [PlotDraw](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotOpen](#), [PlotScaleMarker](#), [PlotText](#), [PlotXYLine](#), [TrnPlot](#)

Example

See [PlotOpen](#)

See Also

[Plot Functions](#)

PlotInfo

Gets information about the plot. You can call this function to determine the number of pixels per page or inch, the resolution of a plot, and the size and spacing of characters for a specified text font. You can also check whether a printer can print rotated text. (See [PlotText\(\)](#).)

you need to first call the [PlotOpen\(\)](#) function to get the handle for the plot (*hPlot*) and specify the output device.

Syntax

PlotInfo(*hPlot*, *Type* [, *sInput*])

hPlot:

The plot handle, returned from the [PlotOpen\(\)](#) function. The plot handle identifies the table where all data on the plot is stored.

Type:

The type of plot information to get:

- 0 - Horizontal pixels on a printout page
- 1 - Vertical pixels on a printout page
- 2 - Horizontal pixels per inch
- 3 - Vertical pixels per inch
- 4 - Horizontal resolution
- 5 - Vertical resolution
- 6 - Height of the font used
- 7 - External leading of the font used

- 8 - Character width of the font used
- 9 - Rotatable text is allowed or not
- 10 - Indicates whether or not a font is supported
- 11 - Horizontal size of a page in millimeters
- 12 - Vertical size of a page in millimeters

Input:

The font handle (hFont), returned from the DspFont() function. Useful only for Type 6, 7, 8, or 10.

Return Value

The attributes of the plot as a string.

Related Functions

[PlotClose](#), [PlotDraw](#), [PlotGrid](#), [PlotLine](#), [PlotMarker](#), [PlotOpen](#), [PlotScaleMarker](#), [PlotText](#), [PlotXYLine](#), [TrnPlot](#)

Example

```
hPlot = PlotOpen(36,"Display",1);
:
/* Print text in upward direction but first check if printer
supports text rotation. Set default text orientation to left to
right (just in case). */
Orient = 0;
IF PlotInfo(hPlot,9) THEN
    Orient = 1;
END
PlotText(hPlot,hFont,Orient,100,100,"scale");
..
/* Print text "Citect Graph" centred horizontally at top of page.*/
PageWidth = PlotInfo(hPlot,0);           ! Get width of page
hFont = DspFont("Courier",14,black,-1);
TextWidth = PlotInfo(hPlot,8,hFont);      ! Get width of each character
TextPosn = (PageWidth - TextWidth * 12) / 2   ! Get start of 1st character
PlotText(hPlot,hFont,0,TextPosn,0,"Citect Graph");
..
PlotClose(hPlot);
```

See Also

[Plot Functions](#)

PlotLine

Draws a line (in the CitectSCADA plot system) for a set of data points. You specify the data points in the table *pTable*, and plot these points between the *LoScale* and *HiScale* values. The line is drawn inside the frame defined by the PlotGrid() function.

For each line on a plot, you can specify a different pen style, color, and width, and a different marker style and color. You can draw lines either from left to right or from right to left.

you need to first call the PlotOpen() function to get the handle for the plot (*hPlot*) and specify the output device. You should then use the PlotGrid() function to set up the frame and grid, before you call this function to plot the line.

Syntax

PlotLine(*hPlot*, *PenStyle*, *PenCol*, *PenWidth*, *MarkerStyle*, *MarkerCol*, *nMarker*, *Length*, *pTable*, *LoScale*, *HiScale*, *Mode*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where all data on the plot is stored.

PenStyle:

The style of the pen used to draw:

- 0 - Solid
- 1 - Dash (- - - -)
- 2 - Dot (.....)
- 3 - Dash and dot (- . - . - . -)
- 4 - Dash, dot, dot (- . . - . . - . -)
- 5 - Hollow

PenCol:

The color of the pen (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

PenWidth:

The width of the pen, in pixels. If the width is thicker than one pixel, you need to use a solid pen (*PenStyle* = 0). The maximum width is 32.

MarkerStyle:

The style of the markers:

- 0 - No markers
- 1 - Triangle
- 2 - Square
- 3 - Circle
- 4 - Diamond
- 5 - Filled triangle

6 - Filled square

7 - Filled circle

8 - Filled diamond

20 - 32000 - User-defined markers. You can register any symbol as a marker with the PlotSetMarker() function. Call the PlotGetMarker() function if the number of markers you have previously registered are unknown.

MarkerCol:

The color of the markers (flashing color is not supported). Select a color from the list of predefined color names and codes or create an RGB-based color using the function [MakeCitectColor](#).

nMarker:

The number of samples between markers.

Length:

The length of the array, that is, the number of points in the table pTable for PlotLine(), or in tables xTable and yTable for PlotXYLine().

- For every line you draw with the PlotLine() and PlotXYLine() functions within a plot, you need to add the Length arguments for each call, and pass the total to the PlotGrid() function (in the nSamples argument).

pTable:

The points to be plotted (as an array of floating-point values).

LoScale:

The lowest value that will be displayed on the plot (that is the value assigned to the origin of your grid). The LoScale and HiScale values determine the scale of your grid. This scale is used to plot values. for example, If LoScale = 0 (zero) and HiScale = 100, a value of 50 will be plotted half way up the Y-axis of your grid. LoScale needs to be in the same units as the values in pTable.

HiScale:

The highest value that will be displayed on the plot. The LoScale and HiScale values determine the scale of your grid. This scale is used to plot values. for example, If LoScale = 0 (zero) and HiScale = 100, a value of 50 will be plotted half way up the Y- axis of your grid. HiScale needs to be in the same units as the values in pTable.

Mode:

The origin of your grid, and the direction of the plotted line:

- 1 - Origin is bottom-left, x is left to right, y is upwards
- 2 - Origin is bottom-right, x is right to left, y is upwards
- 4 - Origin is top-left, x is left to right, y is downwards
- 8 - Origin is top-right, x is right to left, y is downwards

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotClose](#), [PlotDraw](#), [PlotGetMarker](#), [PlotGrid](#), [PlotInfo](#), [PlotMarker](#), [PlotOpen](#), [PlotScaleMarker](#), [PlotSetMarker](#), [PlotText](#), [PlotXYLine](#), [TrnPlot](#)

Example

See [PlotOpen](#)

See Also

[Plot Functions](#)

PlotMarker

Draws markers on a plotted line or at a specified point. You can plot any one of the standard markers, or use a symbol of your choice. (you need to first register your symbol as a marker, by using the PlotSetMarker() function.)

To draw a single marker at a specified point, set *X* and *Y* to the coordinates of the point, and set *Length* to 1.

You can draw markers on a plotted line when you draw the line, that is within the PlotLine() or PlotXYLine() function. You would use the PlotMarker() function only if you need to draw a second set of markers on the same line. Call PlotMarker() immediately after the line is drawn. Set *X* and *Y* to -1 and *Length* to the number of data points (specified in the *Length* argument of the PlotLine() or PlotXYLine() function).

you need to first call the PlotOpen() function to get the handle for the plot (*hPlot*) and specify the output device.

Syntax

PlotMarker(*hPlot*, *MarkerStyle*, *MarkerCol*, *nMarker*, *Length*, *X*, *Y*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where data on the plot is stored.

MarkerStyle:

The style of the markers:

0 - No markers

1 - Triangle

2 - Square
3 - Circle
4 - Diamond
5 - Filled triangle
6 - Filled square
7 - Filled circle
8 - Filled diamond
20 - 32000: User-defined markers. You can register any symbol as a marker with the PlotSetMarker() function. Call the PlotGetMarker() function if the number of markers you have previously registered are unknown.

MarkerCol:

The color of the marker (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

nMarker:

The number of samples between markers.

Length:

The length of the array (the number of line points in the table pTable) plotted in the PlotLine() or PlotXYLine() function. To draw only one marker at a specified point, set Length to 1.

X, Y:

The x and y coordinates, in pixels, of the point where the marker is to be drawn. If the plot is for display on the screen, the coordinates are relative to the AN specified in the PlotOpen() function. If the output device is a printer, the coordinates are relative to the point (0,0).

To draw the markers on a plotted line, set both X and Y to -1, and set Length to the same value as the Length passed in the PlotLine() or PlotXYLine() function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotClose](#), [PlotDraw](#), [PlotGetMarker](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotOpen](#), [PlotScaleMarker](#), [PlotSetMarker](#), [PlotText](#), [PlotXYLine](#), [TrmPlot](#)

Example

```
hPlot=PlotOpen(36,"Display",1);
..
/* Draw a filled red square marker at the point (X=100,Y=200). */
```

```

PlotMarker(hPlot,6,red,1,1,100,200);
..
/* Draw 10 black triangles and 5 green cylinders along a plot
line. */
PlotLine(hPlot,0,black,3,5,black,10,100,Buf2[0],0,100,2);
PlotSetMarker(20,"Global.Cylinder");
PlotMarker(hPlot,20,green,5,100,-1,-1)
..
PlotClose(hPlot);

```

See Also[Plot Functions](#)**PlotOpen**

Opens a new plot, sets its output device, and returns its plot handle. You can send the plot to any one of your system printers, or display it on screen at the specified AN. you need to call this function before you can call the other plot functions.

Syntax**PlotOpen(*AN, sOutput, Mode*)***AN:*

The animation point (AN) where the plot will display. Set the AN to 0 (zero) when *sOutput* is a printer.

Do not use an animation point number at which a graphic object exists as this will prevent the PlotOpen() function from succeeding.

sOutput:

The output device where the plot is sent, for example:

- "Display" - Display on screen. The plot is recorded in a metafile and displayed (at the specified AN) when the plot system is closed.
- "LPT1:" - Send to printer LPT1.
- "LPT2:" - Send to printer LPT2.
- "\\\ABC\Printers\Color1" - Send to UNC port (and so on for any output device)

Mode:

When a plot is removed or updated, the portion of the background screen beneath it is blanked out. The mode determines how the background screen is restored. The mode of the plot system:

1 - Normal mode

2 - Use for compatibility with the old graph functions

- 17 - Soft (valid for normal mode). The background screen (a rectangular region beneath the plot) is restored with the original image. Any objects that are within the rectangular region are destroyed when the background is restored.
- 33 - Hard (valid for normal mode). The background screen (a rectangular region beneath the plot) is painted with the color at the AN.
- 65 - Persistent (valid for normal mode). The plot is not erased. As the plot is updated, it is re-displayed on top. This mode provides fast updates. Transparent color is supported in this mode.
- 129 - Opaque animation (valid for normal mode). The plot is not erased. As the plot is updated, it is re-displayed on top. This mode provides the fastest updates. Transparent color is not supported in this mode.
- 257 - Overlapped animation (valid for normal mode). The background screen (the rectangular region beneath the plot) is completely repainted.

Return Value

The plot handle if the plot is opened successfully, otherwise -1 is returned. The plot handle identifies the table where all data on the associated plot is stored.

Related Functions

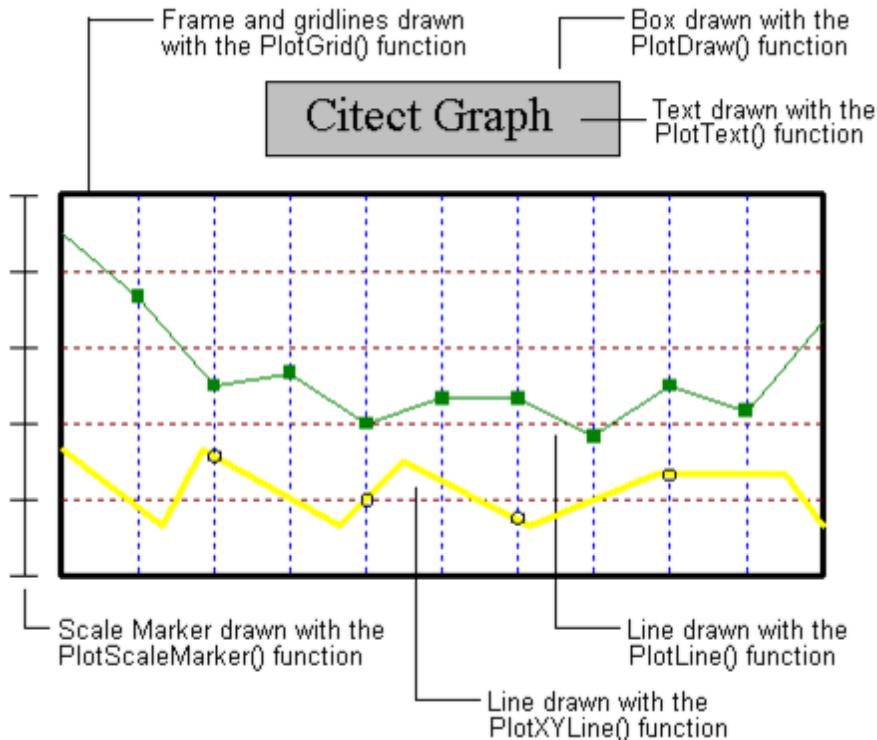
[PlotClose](#), [PlotDraw](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotScaleMarker](#), [PlotText](#), [PlotXYLine](#), [TrnPlot](#)

Example

```
hPlot=PlotOpen(0,"LPT2:",1);
IF hPlot <> -1 THEN
    /* Set up a black frame with red & blue grid lines. */
    PlotGrid(hPlot,18,450,800,1850,1600,5,red,10,blue,4,black,white,0);
    /* Draw a scale line to the left of the frame. */
    PlotScaleMarker(hPlot,400,1600,6,1,black,0);
    /* Plot a simple line in green for a table of 10 values. */
    PlotLine(hPlot,0,green,3,6,green,2,10,Buf1,0,100,1);
    /* Plot a line in yellow (with black markers) for tables of 8 X and Y values. */

    PlotXYLine(hPlot,0,yellow,4,3,black,2,8,Buf2,0,150,Buf3,0,100,1);
    /* Draw a title box above the plot frame, with the heading "Citect Graph". */
    PlotDraw(hPlot,1,0,black,1,grey,900,250,1400,400);
    hFont = DspFont("Times",-60,black,grey);
    PlotText(hPlot,hFont,0,950,350,"Citect Graph");
    PlotClose(hPlot);
END
```

The above example prints the following (on the printer):



```

PlotOpen(0,"LPT1:",1)      // opens a new plot to be sent to printer
PlotOpen(20,"DISPLAY",17)   // normal plot with soft animation
PlotOpen(20,"DISPLAY",257)  // normal plot with overlap animation
PlotOpen(20,"DISPLAY",1)    // normal plot with overlap animation
                           // (for default animation mode is overlap animation)
PlotOpen(20,"DISPLAY",16)   // INVALID
                           // (does not specify whether it is normal or Version 2.xx mode).
PlotOpen(20,"DISPLAY",2)    // INVALID for Version 2.xx graph system
                           // (does not support display as output).

```

See Also

[Plot Functions](#)

PlotScaleMarker

Draws scale lines beside the grid on your plot (if there is one) and places markers on them. The height of the scale line is automatically set to the height of the frame set in the PlotGrid() function.

You need to first call the PlotOpen() function to get the handle for the plot (hPlot) and specify the output device. You should then use the PlotGrid() function to set up the frame and grid, before you call this function to draw the scale lines.

Syntax

PlotScaleMarker(*hPlot*, *X*, *Y*, *nMarker*, *PenWidth*, *PenCol*, *Mode*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where data on the plot is stored.

X, *Y*:

The x and y coordinates of the point where the scale line starts. The end coordinates of the scale line are automatically defined by the size of the frame (set in the PlotGrid() function).

If the plot is for display on the screen, coordinates are relative to the AN specified in the PlotOpen() function. If the output device is a printer, coordinates are relative to the point (0,0).

nMarker:

The number of markers on the scale line.

PenWidth:

The width of the scale line, in pixels.

PenCol:

The color of the pen (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

Mode:

The mode of the markers:

- 0 - Both sides of the scale line
- 1 - Left of the scale line
- 2 - Right of the scale line

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotClose](#), [PlotDraw](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotOpen](#), [PlotText](#), [PlotXY-Line](#), [TrnPlot](#)

Example

See [PlotOpen](#)

See Also[Plot Functions](#)**PlotSetMarker**

Registers a symbol as a marker. You can then draw the new marker at points and on plotted lines, by specifying the *MarkerNo* of the symbol as the *MarkerStyle* in the *PlotMarker()* function. Call the *PlotGetMarker()* function if you do not know the number of a marker.

Syntax**PlotSetMarker(*MarkerNo*, *sSymbolName*)***MarkerNo*:

The number of the marker, to be used as the *MarkerStyle* in the *PlotMarker()* function. Your marker numbers need to be greater than or equal to 20 (to a maximum of 32000).

sSymbolName:

The name and path of the symbol to be defined as a marker.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[PlotMarker](#), [PlotScaleMarker](#), [PlotGetMarker](#)**Example**

```

hPlot=PlotOpen(30,"Display",1);
..
/* Display red hourglass as marker at point (100,200). */
PlotSetMarker(20,"Global.Hourglass");
PlotMarker(hPlot,20,red,1,1,100,200);
..
PlotClose(hPlot);

```

See Also[Plot Functions](#)**PlotText**

Prints text on a plot. You can specify the font, position, and orientation of the text. If you specify an orientation other than 'left-to-right', you need to check that the font (and the printer) supports the orientation.

you need to first call the PlotOpen() function to get the handle for the plot (*hPlot*) and specify the output device. You also need to call the DspFont() function to get a handle for the font (*hFont*).

Syntax

PlotText(*hPlot*, *hFont*, *Orientation*, *X*, *Y*, *sText*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where all data on the plot is stored.

hFont:

The font handle, returned from the DspFont() function. The font handle identifies the table where details of that font are stored.

Orientation:

The orientation of the text:

0 - Left-to-right

1 - Upwards

2 - Right-to-left

3 - Downwards

You should check that the font supports rotation (where Orientation = 1, 2, or 3). Most true type and vector fonts support rotation. If the PlotInfo(*hPlot*, 9) function returns false, you need to specify an Orientation of 0 (zero).

X, *Y*:

The x and y coordinates (in pixels) of the start of the text. If the plot is for display on the screen, the coordinates are relative to the AN specified in the PlotOpen() function. If the output device is a printer, the coordinates are relative to the point (0,0).

sText:

The text string to be plotted.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[DspFont](#), [PlotClose](#), [PlotDraw](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotScaleMarker](#), [PlotXYLine](#), [TrnPlot](#)

Example

See [PlotOpen](#).

See Also

[Plot Functions](#)

PlotXYLine

Plots values from two different tables. Values from one table are considered X coordinates, and values from the other are considered Y coordinates. Points are plotted between the low and high scale values specified for x and y. The line is plotted inside the frame defined by the PlotGrid() function.

For each line, you can specify a different pen style, color, and width, and a different marker style and color. You can draw lines either from left to right or from right to left. You need to first call the PlotOpen() function to get the handle for the plot (*hPlot*) and specify the output device. You should then use the PlotGrid() function to set up the frame and grid, before you call this function to plot the line.

Syntax

PlotXYLine(*hPlot*, *PenStyle*, *PenCol*, *PenWidth*, *MarkerStyle*, *MarkerCol*, *nMarker*, *Length*, *xTable*, *LoXScale*, *HiXScale*, *yTable*, *LoYScale*, *HiYScale*, *Mode*)

hPlot:

The plot handle, returned from the PlotOpen() function. The plot handle identifies the table where all data on the plot is stored.

PenStyle:

The style of the pen used to draw:

0 - Solid

1 - Dash (- - - -)

2 - Dot (.....)

3 - Dash and dot (- . - . - . -)

4 - Dash, dot, dot (- . . - . - . -)

5 - Hollow

PenCol:

The color of the pen (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

PenWidth:

The width of the pen, in pixels. If the width is thicker than one pixel, you need to use a solid pen (PenStyle = 0). The maximum width is 32.

MarkerStyle:

The style of the markers:

0 - No markers

1 - Triangle

2 - Square

3 - Circle

4 - Diamond

5 - Filled triangle

6 - Filled square

7 - Filled circle

8 - Filled diamond

20 - 32000 - User-defined markers. You can register any symbol as a marker with the PlotSetMarker() function. Call the PlotGetMarker() function to recall the number of a marker you have previously registered.

MarkerCol:

The color of the markers (flashing color is not supported). Select a color from the list of Predefined Color Names and Codes or create an RGB-based color using the function [MakeCitectColor](#).

nMarker:

The number of samples between markers.

Length:

The length of the array, that is the number of points in the table pTable for PlotLine(), or in tables xTable and yTable for PlotXYLine().

For every line you draw with the PlotLine() and PlotXYLine() functions within a plot, you need to add the Length arguments for each call, and pass the total to the PlotGrid() function (in the nSamples argument).

xTable:

The x coordinates for the points in the line, as an array of floating point values.

LoXScale:

The lowest X-axis value that will be displayed on the plot (that is the X-coordinate of the origin of your grid). The LoXScale and HiXScale values determine the scale of your grid. This scale is used to plot values. for example, If LoXScale = 0 (zero) and HiXScale = 100, a value of 50 will be plotted half way along the X-axis of your grid.

LoXScale needs to be in the same units as the values in xTable.

HiXScale:

The highest X-axis value that will be displayed on the plot. The LoXScale and HiXScale values determine the scale of your grid. This scale is used to plot values. for example, If LoXScale = 0 (zero) and HiXScale = 100, a value of 50 will be plotted half way along the X-axis of your grid.

HiXScale needs to be in the same units as the values in xTable.

yTable:

The y coordinates for the points in the line, as an array of floating point values.

LoYScale:

The lowest Y-axis value that will be displayed on the plot (that is the Y-coordinate of the origin of your grid). The LoYScale and HiYScale values determine the scale of your grid. This scale is used to plot values. for example, If LoYScale = 0 (zero) and HiYScale = 100, a value of 50 will be plotted half way up the Y-axis of your grid.

LoYScale needs to be in the same units as the values in xTable.

HiYScale:

The highest Y-axis value that will be displayed on the plot. The LoYScale and HiYScale values determine the scale of your grid. This scale is used to plot values. for example, If LoYScale = 0 (zero) and HiYScale = 100, a value of 50 will be plotted half way up the Y-axis of your grid.

HiYScale needs to be in the same units as the values in xTable.

Mode:

The origin of your grid, and the direction of the plotted line:

- 1 - Origin is bottom-left, x is left to right, y is upwards
- 2 - Origin is bottom-right, x is right to left, y is upwards
- 4 - Origin is top-left, x is left to right, y is downwards
- 8 - Origin is top-right, x is right to left, y is downwards

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PlotClose](#), [PlotDraw](#), [PlotGrid](#), [PlotInfo](#), [PlotLine](#), [PlotMarker](#), [PlotScaleMarker](#), [PlotText](#),
[TrnPlot](#)

Example

See [PlotOpen](#)

See Also

[Plot Functions](#)

Chapter: 43 Process Analyst Functions

With the process analyst functions, you can load PAV files and manipulate pens in the Process Analyst display.

Process Analyst Functions

Following are functions relating to Process Analyst:

ProcessAnalystLoadFile	Loads the specified PAV file to a Process Analyst object.
ProcessAnalystPopUp	Displays a Process Analyst page (in a new page child window) at the current mouse position.
ProcessAnalystSelect	Allows a set of pens to be selected before displaying the PA page.
ProcessAnalystSetPen	Allows a new pen to be added to a PA display.
ProcessAnalystWin	Displays a Process Analyst page (in a new window) preloaded with the pre-defined Process Analyst View (PAV) file.

See Also

[Functions Reference](#)

ProcessAnalystLoadFile

Loads the specified PAV file to a Process Analyst object, which is identified by parameter ObjName.

Syntax

ProcessAnalystLoadFile(*sPAVFile*, *iFileLocation*, *iButtonMask*, *sObjName* **)**

sPAVFile:

Name of the PAV file

iFileLocation:

PAV file location code for the PAV file. Indicates which known location to load the file from.

Member Name	Description	Value
FileLocation_Local	Refers to the project folder	0
FileLocation_Server	Refers to the both the primary/standby server paths	1
FileLocation_User	Refers to the My Documents folder	2

iButtonMask:

Bit mask for removing command buttons from the PA, bit flags as shown below:

- 1 - Load View
- 2 - Save View
- 4 - Print
- 8 - Copy to Clipboard
- 16 - Copy to File
- 32 - Add Pens
- 64 - Remove Pens
- 128 - Show Properties
- 256 - Help

sObjName:

Name of the PA object on the given Page where the PAV file will be loaded.

Return Value

Zero (0) if the function is successfully run. Otherwise an error is returned.

Related Functions

[PageProcessAnalyst](#), [PageProcessAnalystPens](#), [ProcessAnalystPopUp](#), [ProcessAnalystSelect](#), [ProcessAnalystSetPen](#), [ProcessAnalystWin](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Process Analyst Functions](#)

ProcessAnalystPopup

Displays a Process Analyst page (in a new page child window) at the current mouse position preloaded with the pre-defined Process Analyst View (PAV) file.

Syntax

ProcessAnalystPopup(*sPage* [, *sPAVFile* [, *iFileLocation* [, *iButtonMask* [, *sObjName* [, *iMode*]]]])

sPage:

The name of the page that contains Process Analyst object(s). For example, pages based on the Process Analyst templates found in the Tab_Style_Include project.

sPAVFile:

Name of the PAV file

iFileLocation:

PAV file location code for the PAV file, see PA doc LoadFromFile() for details.

iButtonMask:

Bit mask for removing command buttons from the PA, bit flags as shown below:

- 1 - Load View
- 2 - Save View
- 4 - Print
- 8 - Copy to Clipboard
- 16 - Copy to File
- 32 - Add Pens
- 64 - Remove Pens
- 128 - Show Properties
- 256 - Help

sObjName:

Name of the PA object on the given Page where the PAV file will be loaded. If this parameter is not specified or empty string, it is defaulted to the object name used in the tab style templates, that is "_templatePA1".

iMode:

The mode of the window (see [WinNewAt\(\)](#) for details).

Return Value

Window number if the window is successfully displayed. Otherwise -1 is returned.

Related Functions

[PageProcessAnalyst](#), [PageProcessAnalystPens](#), [ProcessAnalystLoadFile](#), [ProcessAnalystSelect](#), [ProcessAnalystSetPen](#), [ProcessAnalystWin](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Process Analyst Functions](#)

ProcessAnalystSelect

Works like the existing Cicode Function TrnSelect(). It allows a set of pens to be selected before displaying the PA page. When PageProcessAnalystPens() is called after ProcessAnalystSelect(), the pens specified by both functions will be available in the final PA display. You can also repeat the call sequence of ProcessAnalystSelect() and ProcessAnalystSetPen() multiple times to set up multiple PA objects for the same page before displaying the page.

Syntax

ProcessAnalystSelect((*iWindow*, *sPage* [, *sObjName* [, *sClusterName* [, *iButtonMask* [, *sPAV-File* [, *iFileLocation*]]]])))

iWindow:

The window number (returned from the [WinNumber\(\)](#) function):

-3 - for the current window

-2 - For the next window displayed

sPage:

The name of the page that displays the PA.

sObjName:

The name of the PA object. If this is not specified, it is defaulted to "_TemplatePA1" which is the name used by the built-in templates.

sClusterName:

The name of the cluster that is associated with any trend tag for this PA. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

iButtonMask:

Bit mask for removing command buttons from the PA, bit flags as shown below:

- 1 - Load View
- 2 - Save View

- 4 - Print
- 8 - Copy to Clipboard
- 16 - Copy to File
- 32 - Add Pens
- 64 - Remove Pens
- 128 - Show Properties
- 256 - Help

sPAVFile:

Name of the PAV file

iFileLocation:

PAV file location code for the PAV file, see PA doc LoadFromFile() for details.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageProcessAnalyst](#), [PageProcessAnalystPens](#), [ProcessAnalystLoadFile](#), [ProcessAnalystPopup](#), [ProcessAnalystSetPen](#), [ProcessAnalystWin](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Process Analyst Functions](#)

ProcessAnalystSetPen

Works like the existing function TrnSetPen(). Allows a new pen to be added to a PA display. The pane defaults to the first pane of the PA if it is not specified.

Syntax

ProcessAnalystSetPen((*iPen*, *sTag* [, *sObjName* [, *iPane*]]))

iPen:

Pen number. The allowed values are:

<0 - new pen

0 - the currently selected pen

existing pen number - change existing pen

>existing pen number - new pen

Up to 8 pens can be added to the PA using the Cicode function if ObjName is set to "-2".

Be reminded that unlike trend objects, the pen numbers in Process Analyst are not fixed. They are dynamically reassigned when pens are added or deleted. When setting pens to the Process Analyst on the current display, pens are numbered within the scope of the pane they are in. On the other hand, when setting pens for the next display, pens are numbered in a flat scope regardless of pane number specified.

sTag:

The trend tag name to be assigned to the pen.

sObjName:

The name of the PA object. If this is set to "-2", the pen is set to the next displayed PA page set up by ProcessAnalystSelect(). If the specified ObjName is valid, the changes will be applied to the currently displayed PA. Otherwise, the function will try to set the pen to the specified object on the currently displayed page. If this parameter is not specified or is an empty string, it will default to the object name used in the tab style templates, that is "_templatePA1".

iPane:

Optional number of the pane where the trend or variable tags are added. Please see the same parameter for function PageProcessAnalystPens() for details. Defaulted to 0, that is, the first pane.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageProcessAnalyst](#), [PageProcessAnalystPens](#), [ProcessAnalystLoadFile](#), [ProcessAnalystPopup](#), [ProcessAnalystSelect](#), [ProcessAnalystWin](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Process Analyst Functions](#)

ProcessAnalystWin

Displays a Process Analyst page (in a new window) preloaded with the pre-defined Process Analyst View (PAV) file.

Syntax

ProcessAnalystWin(*sPage*, *iX*, *iY*, *iMode* [, *sPAVFile* [, *iFileLocation* [, *iButtonMask* [, *sObjName*]]]])

sPage:

The name of the page that contains Process Analyst object(s). For example, pages based on the Process Analyst templates found in the Tab_Style_Include project.

iX:

The X pixel coordinate

iY:

The Y pixel coordinate

iMode:

The mode of the window (see [WinNewAt\(\)](#) for details).

sPAVFile:

Name of the PAV file

iFileLocation:

PAV file location code for the PAV file, see PA doc LoadFromFile() for details.

iButtonMask:

Bit mask for removing command buttons from the PA, bit flags as shown below:

- 1 - Load View
- 2 - Save View
- 4 - Print
- 8 - Copy to Clipboard
- 16 - Copy to File
- 32 - Add Pens
- 64 - Remove Pens
- 128 - Show Properties
- 256 - Help

sObjName:

Name of the PA object on the given Page where the PAV file will be loaded. If this parameter is not specified or empty string, it is defaulted to the object name used in the tab style templates, that is "_templatePA1".

Return Value

Window number if the window is successfully displayed. Otherwise -1 is returned.

Related Functions

[PageProcessAnalyst](#), [PageProcessAnalystPens](#), [ProcessAnalystLoadFile](#), [ProcessAnalystPopup](#), [ProcessAnalystSelect](#), [ProcessAnalystSetPen](#), [TrnSetPen](#), [WinNewAt](#)

See Also

[Process Analyst Functions](#)

Chapter: 44 Quality Functions

The Quality functions allow you to programmatically read and write the quality value for tag elements and access the quality information associated with a tag value.

No function taking either Timestamp or Quality as an argument can be called from the Cicode Kernel Window or through a CtCicode CtAPI function.

Quality Functions

The following functions are used to interface with the [QUALITY](#) data type.

QualityCreate	Creates a quality value based on the quality fields provided.
QualityGetPart	Extracts a requested part of the Quality value from the QUALITY variable.
QualityIsBad	Returns a value indicating whether the quality is bad.
QualityIsGood	Returns a value indicating whether the quality is good.
QualityIsUncertain	Returns a value indicating whether the quality is uncertain.
QualitySetPart	Sets a Quality part's value to the QUALITY variable.
QualityIsOverride	Returns a value indicating whether the tag is in Override Mode.
QualityIsControlInhibit	Returns a value indicating whether the tag is in Control inhibit mode.
QualityToStr	Returns a textual representation of the CitectSCADA quality.
VariableQuality	Extracts the quality from a given variable.

See Also

[Functions Reference](#)

QualityCreate

Creates a quality value based on the quality fields provided. When the value of a particular quality field is out of range, the value of its corresponding part in the returned quality remains at 0, and hardware error is generated.

Syntax

QualityCreate(INT generalQuality [, INT qualitySubstatus [, INT qualityLimit [, INT extendedSubstatus [, INT bOverride [, bControlInhibit [, INT datasourceErrorCode]]]]]))
generalQuality:

Specifies the general quality.

qualitySubstatus:

Specifies the quality substatus.

qualityLimit:

The Quality Limit.

extendedSubstatus:

Specifies the extended quality substatus.

bOverride:

Specifies the Tag Status Override Flag.

bControlInhibit:

Specifies the Tag Status Control Inhibit Flag.

datasourceErrorCode:

Specifies the data source error code.

For further information on the quality arguments listed above, refer to the Tag Extensions documentation in the main help.

Return Value

The Quality value of the element.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsContolInhibit](#), [QualityIsGood](#), [QualityIsOverride](#), [QualityIsUncertain](#), [QualitySetPart](#), [QualityToStr](#),

Example

```
QUALITY q1;
q1 = QualityCreate(QUAL_BAD, QUAL_BAD_NON_SPECIFIC, QUAL_LIMITED_HIGH, QUAL_EXT_NOT_
REPLICATED);
```

See Also

[Quality Functions](#)

QualityGetPart

Extracts a requested part of the Quality value from the QUALITY variable.

Syntax

QualityGetPart(QUALITY quality, INT part)

quality:

Specifies the quality variable.

part:

The part to extract:

0 – The General Quality value

1 – Quality Substatus value

2 - The Quality Limit value

3 - The Extended Quality Substatus value

4 – The Tag Status Override flag

5 – The Tag Status Control Inhibit flag

6 - The DataSource error code

7 – The OPC Quality (General + Substatus + Limit)

Return Value

The value of the requested Quality part, or -1 if error.

Related Functions

[QualityIsBad](#), [QualityIsControlInhibit](#), [QualityIsGood](#), [QualityIsOverride](#), [QualityIsUncertain](#), [QualitySetPart](#), [QualityToStr](#), [QualityCreate](#)

Example

```
LONG qualityGeneral;
qualityGeneral = QualityGetPart(Tag1.Field.Q, 0);
```

See Also

[Quality Functions](#)

QualityIsControlInhibit

Returns a value indicating whether the tag is in Control Inhibit Mode.

Syntax

QualityIsControlInhibit(QUALITY quality)

quality:

Specifies the QUALITY variable.

Return Value

0: the tag is not in Control inhibit Mode.

1: the tag is in Control inhibit Mode.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsGood](#), [QualityIsOverride](#), [QualityIsUncertain](#),
[QualitySetPart](#), [QualityToStr](#), [QualityCreate](#)

Example

```
INT controlInhibitEnabled;  
controlInhibitEnabled = QualityIsControlInhibit(Tag1.Field.Q);
```

See Also

[Quality Functions](#)

QualityIsBad

This function will return a value indicating whether the general part of quality is bad.

Syntax

QualityIsBad(QUALITY quality)

quality:

Specifies the QUALITY variable.

Return Value

0: the quality is not bad.

1: the quality is bad.

Related Functions

[QualityGetPart](#), [QualityIsContolInhibit](#), [QualityIsGood](#), [QualityIsOverride](#), [QualityIsUncertain](#), [QualitySetPart](#), [QualityToStr](#), [QualityCreate](#)

Example

```
INT bad;
bad = QualityIsBad(Tag1.Field.Q);
```

See Also

[Quality Functions](#)

QualityIsGood

This function will return a value indicating whether the general part of quality is good.

Syntax

QualityIsGood(QUALITY quality)

quality:

Specifies the QUALITY variable.

Return Value

0: the quality is not good.

1: the quality is good.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsContolInhibit](#), [QualityIsOverride](#), [QualityIsUncertain](#), [QualitySetPart](#), [QualityToStr](#), [QualityCreate](#)

Example

```
INT good;
good = QualityIsGood(Tag1.Field.Q);
```

See Also

[Quality Functions](#)

QualityIsOverride

Returns a value indicating whether the tag is in Override Mode.

Syntax

QualityIsOverride(QUALITY quality)

quality:

Specifies the QUALITY variable.

Return Value

0: the tag is not in Override Mode.

1: the tag is in Override Mode.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsContolInhibit](#), [QualityIsGood](#), [QualityIsUncertain](#),
[QualitySetPart](#), [QualityToStr](#), [QualityCreate](#)

Example

```
INT overrideEnabled;  
overrideEnabled = QualityIsOverride(Tag1.Q);
```

See Also

[Quality Functions](#)

QualityIsUncertain

This function will return a value indicating whether the general part of quality is uncertain.

Syntax

QualityIsUncertain(QUALITY quality)

quality:

Specifies the QUALITY variable.

Return Value

0: the quality is not uncertain.

1: the quality is uncertain.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsControlInhibit](#), [QualityIsGood](#), [QualityIsOverride](#),
[QualitySetPart](#), [QualityToStr](#), [QualityCreate](#)

Example

```
INT uncertain;
uncertain = QualityIsUncertain(Tag1.Field.Q);
```

See Also

[Quality Functions](#)

QualitySetPart

Sets a Quality part's value to the QUALITY variable.

Syntax

QualitySetPart(QUALITY quality, INT part, LONG value)

quality:

Specifies the quality variable.

part:

The part to extract:

0 – The General Quality value

1 – Quality Substatus value

2 - The Quality Limit value

3 - The Extended Quality Substatus value

4 – The Tag Status Override flag

5 – The Tag Status Control Inhibit flag

6 - The DataSource error code

7 – The OPC Quality (General + Substatus + Limit)

value:

The new value for the given part.

Return Value

The modified Quality value, or the original value if the given part is not applicable.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsControlInhibit](#), [QualityIsGood](#), [QualityIsOverride](#),
[QualityIsUncertain](#), [QualityToStr](#), [QualityCreate](#)

Example

```
QUALITY quality;
LONG qualityGeneral;
// insert code here
quality = QualitySetPart(quality, 0, qualityGeneral);
```

See Also

[Quality Functions](#)

QualityToStr

Returns a textual representation of the quality.

Syntax

QualityToStr(QUALITY quality, INT part, INT localized)

quality:

Specifies the QUALITY variable.

part:

Specifies the part of quality to obtain the textual representation.

-2: Short representation in the format <General Quality> [-<Quality Substatus>]

-1: Full representation in the format <General Quality> [Override] [Control Inhibit] -<Quality Substatus>

0: <General Quality>

1: <Quality Substatus>

2: <Quality Limit>

3: <Extended Quality Substatus>

4: <Quality Override>

5: <Control Inhibit>

localized:

The flag indicating if the returned text should be in native language or in Runtime localized language.

Return Value

A textual representation of the quality, or an empty string if the part given is not applicable.

Related Functions

[QualityGetPart](#), [QualityIsBad](#), [QualityIsContolInhibit](#), [QualityIsGood](#), [QualityIsOverride](#),
[QualityIsUncertain](#), [QualitySetPart](#), [QualityCreate](#)

Example

```

QUALITY q;;
STRING str;

q = QualityCreate(QUAL_GOOD, 0, QUAL_LIMITED_NOT_LIMITED,
QUAL_EXT_NON_SPECIFIC, 1, 1, 0);

str = QualityToStr(q, -1, 0);

// The result is: Good [Override] [Control Inhibit]

q = QualityCreate(QUAL_GOOD, QUAL_GOOD_LOCAL_OVERRIDE,
QUAL_LIMITED_NOT_LIMITED, QUAL_EXT_NON_SPECIFIC);

str = QualityToStr(q, 1, 0);

// The result is: Overriden

```

```
q = QualityCreate(QUAL_GOOD, 0, QUAL_LIMITED_QL_LOW_LIMITED,
```

```
QUAL_EXT_NON_SPECIFIC);
```

```
str = QualityToStr(q, 2, 0);
```

```
// The result is: Below Low
```

```
SetLanguage("FRENCH");
```

```
q = QualityCreate(QUAL_GOOD, 0, QUAL_LIMITED_NOT_LIMITED,
```

```
QUAL_EXT_NON_SPECIFIC, 1, 1, 0);
```

```
str = QualityToStr(q, -1, 1);
```

```
// The result is: Bon [Supplémenté] [Contrôle Inhibé]
```

```
// Entries for 'Bon', 'Supplémenté' and 'Contrôle Inhibé'
```

```
// needs to have been provided in FRENCH.dbf
```

```
SetLanguage("ENGLISH");
```

```
q = QualityCreate(QUAL_BAD, QUAL_BAD_CONFIGURATION_ERROR,
```

```
QUAL_LIMITED_NOT_LIMITED, QUAL_EXT_NON_SPECIFIC);  
  
str = QualityToStr(q, -1, 0);  
  
// The result is: Bad - Configuration Error  
  
  
SetLanguage("FRENCH");  
  
  
q = QualityCreate(QUAL_BAD, QUAL_BAD_CONFIGURATION_ERROR,  
QUAL_LIMITED_NOT_LIMITED, QUAL_EXT_NON_SPECIFIC);  
  
str = QualityToStr(q, -1, 1);  
  
// The result is: Mauvais - Erreur de Configuration  
  
  
SetLanguage("ENGLISH");  
  
  
q = QualityCreate(QUAL_UNCR, QUAL_UNCR_NON_SPECIFIC,  
QUAL_LIMITED_NOT_LIMITED, QUAL_EXT_TAG_OUT_OF_RANGE);  
  
str = QualityToStr(q, 0, 0);  
  
// The result is: Uncertain  
  
  

```

```
q = QualityCreate(QUAL_UNCR, QUAL_UNCR_NON_SPECIFIC,
```

```
QUAL_LIMITED_NOT_LIMITED, QUAL_EXT_TAG_OUT_OF_RANGE);
```

```
str = QualityToStr(q, 3, 0);
```

```
// The result is: Tag Address Out Of Range
```

```
q = QualityCreate(QUAL_UNCR, QUAL_UNCR_SUBNORMAL,
```

```
QUAL_LIMITED_NOT_LIMITED, QUAL_EXT_NON_SPECIFIC);
```

```
str = QualityToStr(q, 1, 0);
```

```
// The result is: Subnormal
```

See Also

[Quality Functions](#)

VariableQuality

Extracts the quality from a given variable.

Note: This function is designed to be used within Cicode; using it on graphical pages may result in displaying an error message instead of an expected quality message when either its argument has not good quality or an execution error is set.

Syntax

VariableQuality(*Variable*)

Variable:

The variable from which the quality will be extracted.

Return Value

The QUALITY of the given variable. If Variable is NULL, it returns quality uncertain (0x40).

Timestamps of uninitialized stack variables, uninitialized code variables and constants are equal to 0 - invalid timestamp, while their qualities are GOOD

Related Functions

[QualityCreate](#), [QualityGetPart](#), [QualityIsGood](#), [QualityIsUncertain](#), [QualitySetPart](#), [QualityIsOverride](#), [QualityIsControlInhibit](#), [QualityToStr](#)

Example

```

INT codeVariable = 1;

INT
FUNCTION
MyFunction(REAL arg1)

STRING str = "My string";

QUALITY q;

q = VariableQuality(codeVariable);           //code variable

q = VariableQuality(arg1);                  //function argument

q = VariableQuality(str);                  //stack variable

q = VariableQuality(Tag1);                //any tag/local variable

RETURN 1;

```

END

See Also

[Quality Functions](#)

Chapter: 45 Report Functions

With the report functions, you can run reports on the report server, change their scheduling, or get their status.

Report Functions

Following are functions relating to Reports:

RepGetCluster	Retrieves the name of the cluster the report is running on.
RepGetControl	Gets report control information.
Report	Runs a report.
RepSetControl	Sets report control information.

See Also

[Functions Reference](#)

RepGetCluster

This function retrieves the name of the cluster a report is running on. This function should only be called from a report file.

Syntax

`RepGetCluster()`

Return Value

The name of the cluster the report is running on.

See Also

[Report Functions](#)

RepGetControl

Gets report control information on a report. This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

RepGetControl(ReportName, Type [, ClusterName])

ReportName:

The name of the report (can be prefixed by the name of the cluster that is ClusterName.ReportName).

Type:

The type of report control information to get (send back in the return value):

0 - State of the report - returns one of:

- 0 -Idle
- 1 - Waiting for PLC data for trigger
- 2 - Waiting for PLC data
- 3 - Running

1 - Time of day that the report is due to run next.

2 - The report period, in seconds, or week day, month or year, for example, if the report is weekly, this is the day of the week, 0 (Sunday) to 6 (Saturday).

3 - Synchronisation time of day of the report, for example, 10:00:00 (In seconds from midnight).

4 - Type of report schedule - returns one of:

- 0 - Event triggered
- 1 - Daily
- 2 - Weekly
- 3 - Monthly
- 4 - Yearly

5 - Report state - returns one of:

- 0 - Enabled
- 1 - Disabled

ClusterName:

Name of the cluster in which the report resides. This is optional if you have one cluster or are resolving the report server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The control information, as an integer.

Related Functions

[RepSetControl](#), [Report](#)

Example

```
Next=RepGetControl("SHIFT",1,"ClusterXYZ");
! Sets Next to the time that the report is due to run.
! Display a message at the prompt AN (AN2) if
! the report is running.
IF RepGetControl("SHIFT",0,"ClusterXYZ")=3 THEN
    Prompt("Shift report is running");
END
```

See Also

[Report Functions](#)

Report

Runs a report on the Report Server. This function only schedules the report for execution. The running of the report is controlled entirely by the Report Server.

This function will start the specified report on the Reports Server to which the Citect-SCADA computer is communicating. If you are using the Reports Servers in Primary/Standby mode, the report can run on the Standby Server. If you call this function on the Standby Server then the report will definitely run on the Standby Server, even if the Primary Server is active.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

Report(*ReportName* [, *ClusterName*])

ReportName:

The name of the report to run (can be prefixed by the name of the cluster that is *ClusterName.ReportName*).

ClusterName:

Name of the cluster in which the report resides. This is optional if you have one cluster or are resolving the report server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[RepSetControl](#), [RepGetControl](#)

Example

Buttons	
Text	Shift Report
Command	Report("Shift", "ClusterXYZ")
Comment	Runs the Shift Report
System Keyboard	
Key Sequence	Report ##### Enter
Command	Report(Arg1)
Comment	Runs a specified Report

```
Report("SHIFT","ClusterXYZ");
! Runs the report named "SHIFT".
Report("DAY","ClusterXYZ");
! Runs the report named "DAY".
/* The "SHIFT" and "DAY" reports are started. The order in which
the reports are run cannot be determined. If you want the "DAY"
report to run after the "SHIFT" report, call Report("DAY") at the
end of the "SHIFT" report. */
```

See Also[Report Functions](#)**RepSetControl**

Sets report control information to temporarily override the normal settings for a specified report. You can change the report schedule for a periodic report, and run one-time or event-triggered reports. These new settings are set on both the primary and standby report servers, but are not saved to the database. When you restart your system, Citect-SCADA uses the existing settings, defined in the Reports database.

You might need to call this function several times. For example, to change an event-triggered report to run at 6 hourly intervals, you need to change the schedule (Type 4), synchronization time (Type 3), and period (Type 2). If you use incompatible values for these options, you can get unpredictable results. To change more than one option, disable the report, set the options, and then re-enable the report.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

RepSetControl(ReportName, Type, Data [, ClusterName])

ReportName:

The name of the report (can be prefixed by the name of the cluster that is ClusterName.ReportName).

Type:

The type of report control information to set:

1 - The time of day at which to run the next report in Cicode (date/time) variable type. Subsequent reports are run at the times calculated from the period (Type 2) and synchronisation time (Type 3). Use Type 1 to specify a one-time report. Set the time in Data in seconds from midnight (for example, specify 6 p.m. as TimeMidNight() + (18 * 60 * 60)).

2 - The report period. Set the new period in Data according to the report schedule (Type 4), in seconds from midnight, day of week (0 to 6, Sunday = 0), month (1 to 12), or year.

For a daily report schedule, set the report frequency in Data in seconds from midnight; for example, set Data to 6 * 60 * 60 for a 6 hourly shift report. If the report is weekly, set Data to the day of the week, for example, when Data = 2, the day is Tuesday.

3 - Synchronisation time of day of the report. Set the time in Data in seconds from midnight, for example, to synchronize at 10a.m., set Data to 10 * 60 * 60.

4 - Type of report schedule. Set Data to one of the following:

- 0 - Event triggered
- 1 - Daily
- 2 - Weekly
- 3 - Monthly
- 4 - Yearly

5 - Report state. Set Data to either:

- 0 - Enabled
- 1 - Disabled

Data

The new data value, dependent on the Type.

ClusterName:

Name of the cluster in which the report resides. This is optional if you have one cluster or are resolving the report server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[RepGetControl](#), [Report](#)

Examples

Run the "Shift" report in 1 minute.

```
RepSetControl("Shift",1,TimeCurrent()+60,"ClusterXYZ");
```

Change weekly report to 8 hour shift starting at 7 am

```
RepSetControl("Weekly", 5, 1,"ClusterXYZ");           ! disable report
RepSetControl("Weekly", 4, 1,"ClusterXYZ");           ! change mode to daily
RepSetControl("Weekly", 3, 7 * 60 * 60,"ClusterXYZ"); ! sync at 7:00:00 am
RepSetControl("Weekly", 2, 8 * 60 * 60,"ClusterXYZ"); ! run every 8 hours
RepSetControl("Weekly", 5, 0,"ClusterXYZ");           ! enable report
```

Change yearly report to run on March 10 at 7 am

```
RepSetControl("Yearly", 5, 1,"ClusterXYZ");           ! disable report
RepSetControl("Yearly", 4, 4,"ClusterXYZ");           ! change mode to yearly
RepSetControl("Yearly", 3, 7 * 60 * 60,"ClusterXYZ"); ! sync at 7:00:00 am
RepSetControl("Yearly", 2, 31 + 28 + 10,"ClusterXYZ"); ! run on March 10th
RepSetControl("Yearly", 5, 0,"ClusterXYZ");           ! enable report
```

See Also

[Report Functions](#)

Chapter: 46 Security Functions

The security functions allow you to control logins, logouts, and general security, and to add, delete, and modify user records during run time. By giving selected users access to these functions, you can provide them with supervisory control of the system.

Security Functions

Following are functions relating to Security:

<u>FullName</u>	Gets the full name of the current operator.
<u>GetPriv</u>	Checks the privilege and area of the current operator.
<u>Login</u>	Logs an operator into the CitectSCADA system. Not available when logged in as Windows user.
<u>LoginForm</u>	Displays a form that allows an operator to log in to the CitectSCADA system.
<u>Logout</u>	Logs an operator out of the CitectSCADA system.
<u>LogoutIdle</u>	Sets an idle time logout for the current operator.
<u>MultiSignatureForm</u>	Displays a form that allows up to 4 users to have their credentials verified in order to approve an operation.
<u>MultiSignatureTagWrite</u>	Displays a form that allows up to 4 users to have their credentials verified in order to approve a write of a specific value to a specific tag.
<u>Name</u>	Gets the user name of the current operator.
<u>UserCreate</u>	Creates a new user record during run time. Not available when logged in as Windows user.
<u>UserCreateForm</u>	Displays a form to create a record for a new user. Not available when logged in as Windows user.
<u>UserDelete</u>	Deletes a new user record during run time. Not available when logged in as Windows user.

<u>UserEditForm</u>	Displays a form for a selected user to create or delete user records during run time. Not available when logged in as Windows user.
<u>UserInfo</u>	Gets information about the operator who is currently logged-in to the system.
<u>UserLogin</u>	Logs an operator into the CitectSCADA system using a secure password string.
<u>UserPassword</u>	Changes a user's password during run time. Not available when logged in as Windows user.
<u>User-PasswordExpiryDays</u>	Returns the number of days left before the user's password is due to expire. Not available when logged in as Windows user.
<u>UserPasswordForm</u>	Displays a form for the operator to change their own password during run time. Not available when logged in as Windows user.
<u>UserSetStr</u>	Sets the value of the given field for the given user record in the project configuration (users.dbf) on the local machine.
<u>UserUpdateRecord</u>	Triggers a recompile of the local project configuration, then notifies the running system that user configuration has been modified and needs to be reloaded.
<u>UserVerify</u>	Uses the authentication functionality in the user login system.
<u>VerifyPrivilegeForm</u>	Displays a form that allows a single user to enter their credentials.
<u>Ver-ifyPrivilegeTagWrite</u>	Displays a form that allows any single user to enter their credentials in order to approve a write of a specific value to a specific tag.
<u>WhoAmI</u>	Displays the name of the operator who is currently logged-in to the system.

See Also

[Functions Reference](#)

FullName

Gets the full name of the user who is currently logged on to the system. The user can be a Citect or a Windows user. For a Citect user the full name is the one defined in the users form. For a Windows user the full name is in the format of <Domain-Name>\<UserName>. When there is no one logged in or the logged in user is a "system user" this function returns an empty string.

Syntax

FullName()

If the user is logged on as a Domain user the name should be the Windows domain name and user account name in the format of <DomainName\UserName>.

If the user is logged on as a local user the name should be the local machine name and user account name in the format of <MachineName\UserName>.

Return Value

The user name (as a string).

Related Functions

[Name](#), [UserInfo](#)

Example

```
/* Display the full name of the current user at AN20. */
DspText(20, 0, FullName());
```

See Also

[Security Functions](#)

GetPriv

Checks if the current user has a privilege for a specified area. With this function, you can write your own Cicode functions to control user access to the system.

Syntax

GetPriv(*Priv*, *Area*)

Priv

The privilege level (1..8).

Area

The area of privilege (0..255).

Return Value

Returns 1 if the user has the specified privilege in the area, or 0 (zero) if the user does not have the privilege.

Related Functions

[SetArea](#)

Example

```
/* User needs to have privilege 2, or cannot do operation. */
IF GetPriv(2, 0) THEN
    ! Do operation here
ELSE
    Prompt("No privilege for command");
END
```

See Also

[Security Functions](#)

Login

Not available when logged in as Windows user.

Logs a user into the CitectSCADA system, using CitectSCADA security and gives users access to the areas and privileges assigned to them in the Users database. Only one user can be logged into a computer at any one time. If a user is already logged in when a second user logs in, the first user is automatically logged out.

When using Windows security use [UserLogin](#) to limit Windows credentials being exposed as plain text.

At startup, or when the user logs out, a default user is active, with access to area 0 (zero) and privilege 0 (zero) only. Use the LoginForm() function to display a form for logging in to the system.

Syntax

Login(*sUserName*, *sPassword*, *bSync*)

sUserName:

The user's name, as defined in the Users database.

sPassword:

The user's password, as defined in the Users database.

bSync:

Specifies whether the function operates in blocking or non-blocking mode. . If set to 1 blocks caller until login is complete. If set to 0 (default) does not block caller.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[LoginForm](#), [Logout](#), [LogoutIdle](#), [Message](#), [Input](#), [UserLogin](#)

Example

```
/* Log in a user whose user name is "FRED" and whose password is
 "ABC". */
Login("FRED", "ABC");
```

See Also

[Security Functions](#)

LoginForm

Displays a form in which a user can log in to the CitectSCADA system by entering their name and password. If the login is correct, the user is logged into the CitectSCADA system with the area(s) and privilege(s) assigned to them in the Users database.

From version 7.10 this form can be pre-filled by the caller. Both Citect and Windows users are supported.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

LoginForm(*sUserName*, *sPassword*)

sUserName:

The user's name, as defined in the Users database.

sPassword:

The user's password, as defined in the Users database.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Login](#), [UserLogin](#)

Example

System Keyboard	
Key Sequence	Login
LoginForm	Display the Login form
Comment	Allow user login
Buttons	
Text	Operator Login
LoginForm	Display the Login form
Comment	Allow user login

See Also

[Security Functions](#)

Logout

Logs the current user out of the CitectSCADA system. CitectSCADA continues to run, but with access to area 0 (zero) and privilege 0 (zero) only. If the current page requires access for a specified area (as defined by the page's area property), the system returns to the home page as specified by the parameter[Page]HomePage, and if unsuccessful that returns to the startup page. When multiple pages are currently displayed, this occurs for each open window.

Calling this function to logout the logged on user will cause an automatically logged in user to be logged back on. If there is no user logged in, calling this function will return an error. When the logged on user is an automatically logged in user, calling this function will return an error.

Syntax

`Logout()`

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Login](#), [LoginForm](#), [LogoutIdle](#), [UserInfo](#), [Message](#), [Input](#)

Example

```
/* Log the current user out of the system. */
Logout();
```

See Also

[Security Functions](#)

LogoutIdle

Sets an idle time for logging out the current user. If the current user does not execute a command within the specified idle time, a prompt is displayed. If the prompt is ignored, then the user is logged out. For every user to have the same idle time, you would call this function at startup. Otherwise, you can call the function from the Users database to specify an idle time for each user. This function will not log out an automatically logged on user.

Until reset LogoutIdle remains active. To reset call LogoutIdle (-1) from the Exit command of the Users database record.

Syntax

LogoutIdle(*Idle*)

Idle:

The number of minutes the user needs to be idle before logout will occur. Set Idle to -1 to disable the current logout timeout.

Return Value

No return value.

Related Functions

[Logout](#), [Login](#), [LoginForm](#)

Example

Users	
User Name	Operator1
LoginForm	LogoutIdle(5)
Comment	Logs the user out after five minutes

See Also

[Security Functions](#)

MultiSignatureForm

Displays a form that allows up to 4 users to have their credentials verified in order to approve an operation. The usernames can be Citect or Windows users.

Syntax

MultiSignatureForm(*sOperationDescription*, *sLogDevice*, *sUser1*, *sUser2*, *sUser3*, *sUser4*)

sOperationDescription:

A description of the operation that requires approval. This string will be displayed on the signature form and logged to the log device if the operation is approved.

sLogDevice:

The name of a log device if logging is required, otherwise pass an empty string.

sUser1..4:

Each *sUser* argument needs to be either a Citect user name, a Windows user name (including domain\ prefix) or a blank string. Even though the *sUser* arguments are numbered 1 through 4, this only controls the order in which users are displayed on the multi-signature form. You can pass empty strings for any of these arguments, but at least one user needs to be specified.

Return Value

TRUE (1) if the operation approved (that is all users' credentials were verified and the operator clicked the "Approve" button, otherwise FALSE (0).

Related Functions

[FormSecurePassword](#)

[MultiSignatureTagWrite](#)

[VerifyPrivilegeForm](#)[VerifyPrivilegeTagWrite](#)**Example**

```
// This example sets the page integer to indicate the approval status, but
// it can be used to perform any logic necessary to trigger the operation
// that was approved.

PageSetInt(1, MultiSignatureForm("Shut down plant", "ApprovalLog",
"shiftsupervisor", "DOMAIN\mike.manager", "", ""));
```

See Also[Security Functions](#)[Form Functions](#)**MultiSignatureTagWrite**

Displays a form that allows up to 4 users to have their credentials verified in order to approve a write of a specific value to a specific tag. If all users are verified successfully, the write to the tag is performed by this function before it returns. The usernames can be Citect or Windows users.

Syntax

MultiSignatureTagWrite(*sTagName*, *sValueToWrite*, *sLogDevice*, *sUser1*, *sUser2*, *sUser3*, *sUser4*)

sTagName:

The name of the tag to which a write needs to be approved.

sValuetowrite:

The value to write to the tag if approval succeeds.

sLogDevice:

The name of a log device if logging is required, otherwise pass an empty string.

sUser:

Each *sUser* argument needs to be either a Citect user name, a Windows user name (including domain\ prefix) or an empty string. Even though the *sUser* arguments are numbered 1 through 4, this only controls the order in which users are displayed on the multi-signature form. You can pass empty strings for any of these arguments, but at least one user needs to be specified.

Return Value

TRUE (1) if the operation was approved (that is all users' credentials were verified and the operator clicked the "Approve" button, otherwise FALSE (0).

Related Functions

[FormSecurePassword](#)
[MultiSignatureForm](#)
[VerifyPrivilegeForm](#)
[VerifyPrivilegeTagWrite](#)

Example

```
// This example generates a form to request two users to approve the tag write
operation.
// When approved, the PLC_VAR1 tag is written with the value 123 and a page string
// is set to indicate the approval status.
IF
    (MultiSignatureTagWrite("PLC_VAR1", "123", "", "John Smith", "Angela Huth", "", ""))
THEN
    PageSetStr(1, "TagWrite Successful");
ELSE
    PageSetStr(1, "TagWrite Not Successful");
END
```

See Also

[Security Functions](#)
[Form Functions](#)

Name

Gets the name of the operator who is currently logged on to the display system. The user can be a Citect or a windows user. If this function is called on a server, it returns the name of the local operator. If there is no one logged on, or the logged on user is a "system user" this function returns an empty string.

Syntax

Name()

Return Value

The name of the user as a string. If the user is logged on as a Windows user the name will be the Windows user account name.

Related Functions

[FullName](#), [Login](#), [LoginForm](#)

Example

```
/* Display the user name of the current user at AN20. */
DspText(20,0,Name());
```

See Also

[Security Functions](#)

UserCreate

Not available for a Windows user.

Creates a record for a new user. A new user of the specified type is created. The name of the user needs to be unique.

This function is not supported on the Internet Display Client. If this function is called on the Internet Display Client then it will return an error.

Syntax

UserCreate(*sName*, *sFullName*, *sPassword*, *sType* [, *sAccess*] [, *sPrivGlobal*] [, *sPriv1..sPriv8*])

sName:

The name of the user.

sFullName:

The full name of the user.

sPassword:

The password of the user.

The *sPassword* argument is optional. If not passed, this argument defaults to an empty string which is subsequently ignored. It is included for the purposes of handling duplicate user names and separate password identification compatibility.

sType:

The generic type of user. The type needs to be defined in the Users database (with the Users form).

sAccess:

User's viewable areas.

sPrivGlobal:

User's global privilege.

sPriv1-8:

User's privilege for areas 1 - 8.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserDelete](#), [UserEditForm](#), [UserPassword](#), [UserPasswordForm](#), [UserCreateForm](#)

Example

```
/* Create a new user */
UserCreate("Fred", "Fred Jones", "secret", "Operator");
```

See Also

[Security Functions](#)

UsercreateForm

Not available for a Windows user.

Displays a form to create a record for a new user. A new user of the specified type is created. The name of the user needs to be unique.

Syntax

UsercreateForm()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserDelete](#), [UserEditForm](#), [UserPassword](#), [UserPasswordForm](#), [UserCreate](#)

Example

```
UsercreateForm()
```

See Also

[Security Functions](#)

UserDelete

Not available for a Windows user.

Deletes the record for a user. Changes are written to both the Users database and the runtime database in memory.

Syntax

UserDelete(*sName*)

sName:

The name of the user.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserCreate](#), [UserEditForm](#)

Example

```
/* Delete Fred from the database */
UserDelete("Fred");
```

See Also

[Security Functions](#)

UserEditForm

Not available for a Windows user.

Displays a form to allow the user to create or delete any user record in the database. This function should have restricted access. Changes are written to both the Users database and the runtime database in memory.

Syntax

UserEditForm()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserCreate](#), [UserDelete](#)

Example

```
/* Display a form for the user to create or delete user records. */
UserEditForm();
```

See Also

[Security Functions](#)

UserInfo

Gets information about the operator who is currently logged-in to the system.

Syntax

UserInfo(*Type*)

Type:

The type of user information:

0 - Flag to indicate whether any user other than a view-only user is logged in

1 - The login name of the user

2 - The full name of the user

3 - The time the user logged in

4 - The time the user entered the last command

5 - The number of commands entered by the user

6 - The type of login:

- "0" indicates that the current user is a view-only user.

- "1" indicates there is CitectSCADA or Windows non-default user explicitly logged in.

- "2" indicates the logged on user is a CitectSCADA default user (control client auto login CitectSCADA user).

- "3" indicates the logged on user is a Windows default user (control client auto login windows user).

Return Value

The information (as a string). If an [error](#) is detected, an empty string is returned.

Related Functions

[Login](#)

Example

```
/* Check if a user has logged on. If so, get the user's full name
and the number of commands they have performed. */
String sName;
String sCount;
IF UserInfo(0) = "1" THEN
    sName = UserInfo(2);
    sCount = UserInfo(5);
END
```

See Also

[Security Functions](#)

UserLogin

Logs a user into the CitectSCADA system, using either Windows security or Citect-SCADA security and gives users access to the areas and privileges assigned to them in the Users database. Only one user can be logged into a computer at any one time. If a user is already logged in when a second user logs in, the first user is replaced by the newly logged on user. When a newly logged in user does not have access to view the current page (as defined by the page's area), the system returns to the home page as specified by the parameter[Page]HomePage, and if unsuccessful that returns to the startup page. When multiple pages are currently displayed, this occurs for each open window.

To call this function at user login, the password argument passed needs to be in secure string format.

At startup, or when the user logs out, a default user is active, with access to area 0 (zero) and privilege 0 (zero) only. Use the LoginForm() function to display a form for logging in to the system.

Syntax

UserLogin(*sUserName*, *sPassword*)

sUserName:

The user's name as defined in the Users database, or the Windows User account name, in plain text.

sPassword:

The user's password, as defined in the Users database or Windows account formatted as a secure string.

To improve the user credentials protection provides a system built-in user login function that takes the user name and secure password as the arguments. This reduces the chance that the user's password can be exposed in plain text from the runtime system

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[LoginForm](#), [Logout](#), [LogoutIdle](#), [Message](#), [Input](#)

Example

```
/*
 ** FUNCTION NAME: LoginForm
 **
 ** This function displays the login form, get the user name and
 ** password then tries to log the user in. If the login does not succeed it
 ** will retry until login is ok or user presses the cancel button.
 **
 */

INT
FUNCTION
LoginForm(STRING sName="", STRING sPassword="")
    INT bDone;
    INT nStatus;
    INT hForm;

    bDone = FALSE;
    WHILE bDone = FALSE DO
        FormNew("@(Login Form)", 35, 5, 5);
        FormPrompt(1, 0, "@(Name)");
        FormInput(16, 0, "", sName, 16);
        FormPrompt(1, 2, "@(Password)");
        FormSecurePassword(18, 2, "", sPassword, 16);
        FormButton( 6, 4, " " + "@(OK)" + " ", 0, 1);
        FormButton(20, 4, "@(Cancel)", 0, 2);

        IF FormRead(0) = 0 THEN

            hForm = FormNew("@(User Login)", 36, 1, 8 + 16 + 128 +
256);
            FormPrompt(1, 0, "@(Authentication in progress ...)");
            FormRead(1);
            SleepMs(200);

            IF UserLogin(sName, sPassword) = 0 THEN
                bDone = TRUE
                nStatus = 0;
            ELSE
                sPassword = "";
            END

            IF FormActive(hForm) THEN
                FormDestroy(hForm);
            END
        END
    END

```

```

        END
    ELSE
        bDone = TRUE;
        nStatus = 298;
    END
END
RETURN nStatus;
END

```

See Also

[Security Functions](#)

UserPassword

Not available for a Windows user.

Changes the password for the user. Changes are written to both the Users database and the runtime database in memory.

Syntax

UserPassword(*sName* [, *sPassword*] [, *sOldPassword*])

sName:

The name of the user.

sPassword:

The password of the user.

The *sPassword* argument is optional. If not passed, this argument defaults to an empty string which is subsequently ignored. It is included for the purposes of handling duplicate user names and separate password identification compatibility.

sOldPassword:

The password assigned to the user before the UserPassword() function is run.

The *sOldPassword* argument is optional. If passed, CitectSCADA will only permit the password change (and consequent re-setting of the expiry period) when the old password is correctly entered. If the *sOldPassword* parameter is not passed, the password change will proceed without restriction.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserPasswordForm](#), [UserCreate](#), [UserEditForm](#)

Example

```
/* Change Fred's password */
UserPassword("Fred", "secret");
```

See Also

[Security Functions](#)

UserPasswordExpiryDays

Not available for a Windows user.

Returns the number of days left before the user's password is due to expire.

To use this function, you can build a form page by using cicode that takes the user name and password as inputs and output the number of days that return by UserPasswordExpiryDays().

Syntax

UserPasswordExpiryDays(*sUserName* [, *sPassword*])

sUserName:

The name of the user.

sPassword:

The password of the user.

The *sPassword* argument is optional. If not passed, this argument defaults to an empty string which is subsequently ignored. It is included for the purposes of handling duplicate user names and separate password identification compatibility.

Return Value

The return value contains either the number of days before password expiry, or one of two exception conditions:

- 0 to 365 - number of days
- -1 - no expiry
- -2 - user not found or password wrong

Related Functions

[UserPassword](#)

See Also

[Security Functions](#)

UserPasswordForm

Not available for a Windows user.

Display a form to allow users to change their own passwords. Changes are written to both the Users database and the runtime database in memory.

Syntax

UserPasswordForm()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserPassword](#), [UserCreate](#), [UserEditForm](#)

Example

```
/* Allow users to change their own passwords */
UserPasswordForm();
```

See Also

[Security Functions](#)

UserSetStr

Sets the value of the given field for the given user record in the project configuration (users.dbf) on the local machine.

After this function has been called, use the function [UserUpdateRecord](#) to update the user record on the running system.

Syntax

UserSetStr(*sName*, *sField*, *sData*)

sName:

The name of the user who's configuration record we wish to modify.

sField:

The name of the field in users.dbf to modify.

sData:

The new value of the field.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserDelete](#), [UserEditForm](#), [UserPassword](#), [UserPasswordForm](#), [UserCreateForm](#)

Example

```
UserSetStr("Fred", "Comment", "Fred is an engineer");
UserUpdateRecord();
```

See Also

[Security Functions](#)

UserUpdateRecord

Triggers a recompile of the local project configuration, then notifies the running system that user configuration has been modified and needs to be reloaded.

Use this function in conjunction with [UserSetStr](#) to modify user configuration online.

In order to perform user configuration changes online in a system with multiple computers running SCADA nodes using these functions, you will need to use the RUN and COPY parameters to check the updates are distributed throughout the system, and that the functions are called from the computer which uses the COPY path as its RUN path.

Syntax

UserUpdateRecord()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[UserDelete](#), [UserEditForm](#), [UserPassword](#), [UserPasswordForm](#), [UserCreateForm](#)

Example

```
UserSetStr("Fred", "Comment", "Fred is an engineer");
UserUpdateRecord();
```

See Also[Security Functions](#)**UserVerify**

Verifies a given user by authenticating the user's credential, verifies the user privileges and areas against those specified in the functions parameters. Successful verification however does not log the user into the CitectSCADA runtime system.

Syntax

UserVerify(*sName*, *sPassword* [, *sAccess*] [, *sPrivGlobal*] [, *sPriv1..sPriv8*])

sName:

The name of the user.

sPassword:

The password of the user. The *sPassword* argument needs to be passed as a secure string.

sAccess:

Specifies the required user's viewable areas.

sPrivGlobal:

Specifies the required user's global privilege.

sPriv1-8:

Specifies the required areas for privileges 1 - 8. That is, *sPriv1* contains the areas (1,2,3,4,...,255) where the user has Privilege 1.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

The successful verification has to meet the following conditions:

- The selected user credentials can be authenticated
- The required user privileges are included in the authenticated user's total privileges.

Related Functions

[UserDelete](#), [UserEditForm](#), [UserPassword](#), [UserPasswordForm](#), [UsercreateForm](#)

Example

```
INT FUNCTION UserVerifyTest()
```

```

STRING sName;
STRING sPassword;
INT bDone;
INT nStatus;
bDone = FALSE;
WHILE bDone = FALSE DO
    FormNew("@(Login Form)", 30, 5, 5);
    FormInput(1, 0, "@(Name)" + " ", sName, 16);
    FormSecurePassword(1, 2, "@(Password)" + " ", sPassword, 16);
    FormButton(1, 4, " " + "@(OK)" + " ", 0, 1);
    FormButton(17, 4, "@(Cancel)", 0, 2);
    IF FormRead(0) = 0 THEN
        IF UserVerify(sName, sSecurePassword) = 0 THEN
            bDone = TRUE;
            nStatus = 0;
            Message("Info", "Verification successful", 0)
        ELSE
            sPassword = "";
            Message("Info", "Verification not successful", 0)
        END
    ELSE
        bDone = TRUE;
        nStatus = 298;
    END
END
RETURN nStatus;
END

```

See Also

[Security Functions](#)

VerifyPrivilegeForm

Displays a form that allows a single user to enter their credentials. These credentials are checked against a specified set to ensure the user has the required privileges before allowing the operation to proceed.

The user can be a Citect or Windows user.

Syntax

VerifyPrivilegeForm(*sOperationDescription*, *sLogDevice*, *sAccess*, *sGlobalPriv*, *sPriv1*, *sPriv2*,
sPriv3, *sPriv4*, *sPriv5*, *sPriv6*, *sPriv7*, *sPriv8*)

sOperationDescription:

A description of the operation that requires approval.

sLogDevice:

The name of a log device if logging is required, otherwise pass an empty string.

sAccess:

The required user viewable areas, or pass an empty string for none.

sGlobalPriv:

The required global privilege, otherwise pass an empty string.

sPriv1..8:

Specifies the required areas for privileges 1 - 8. That is, *sPriv1* contains the areas (1,2,3,4,...,255) where the user has Privilege 1. Each argument needs to be either specified or an empty string for none.

Return Value

The name of the user that met the required privileges, otherwise ""

Related Functions

[FormSecurePassword](#)

[MultiSignatureForm](#)

[MultiSignatureTagWrite](#)

[VerifyPrivilegeTagWrite](#)

Example

```
// This example generates a form to request a user to approve an operation.
// This user needs the global privilege level of 8.
// When approved, the operation is completed and a page string
// is set to indicate the approval status.
IF (VerifyPrivilegeForm("Shut Down Plant", "ApprovalLog",
    "PlantWide", "8", "", "", "", "", "", "", "")<>"") THEN
    // Do operation
    PageSetStr(1, "Operation approved");
ELSE
    PageSetStr(1, "Operation not approved");
END
```

See Also

[Security Functions](#)

[Form Functions](#)

VerifyPrivilegeTagWrite

Displays a form that allows any single user to enter their credentials in order to approve a write of a specific value to a specific tag. These credentials are checked against a specified set to ensure the user has the required privileges before allowing the operation to proceed.

The usernames can be Citect or Windows users.

Syntax

VerifyPrivilegeTagWrite(*sTagName*, *sValueToWrite*, *sLogDevice*, *sAccess*, *sGlobalPriv*,
sPriv1, *sPriv2*, *sPriv3*, *sPriv4*, *sPriv5*, *sPriv6*, *sPriv7*, *sPriv8*)

sTagName:

The name of the tag to which a write needs to be approved.

sValueToWrite:

The value to write to the tag if approval succeeds.

sLogDevice:

The name of a log device if logging is required, otherwise pass an empty string.

sAccess:

The required user viewable areas, or pass an empty string for none.

sGlobalPriv:

The required global privilege, otherwise pass an empty string.

sPriv1..8:

Specifies the required areas for privileges 1 - 8. That is, *sPriv1* contains the areas (1,2,3,4,...,255) where the user has Privilege 1. Each argument needs to be either specified or an empty string for none.

Return Value

Name of user that met the required privileges (and therefore the value was written to the specified tag), otherwise ""

Related Functions

[FormSecurePassword](#)

[MultiSignatureForm](#)

[MultiSignatureTagWrite](#)

[VerifyPrivilegeForm](#)

Example

```
// This example generates a form to request a user to approve the tag write operation.  
// This user needs privilege levels of 6 and 3.  
// When approved, the PLC_VAR1 tag is written with the value 123 and a page string
```

```

// is set to indicate the approval status.
IF (VerifyPrivilegeTagWrite("PLC_VAR1", "123", "ApprovalLog",
    "PlantWide", "", "6", "3", "", "", "", "", "")> "") THEN
    PageSetStr(1, "TagWrite Successful");
ELSE
    PageSetStr(1, "TagWrite Not Successful");
END

```

See Also[Security Functions](#)[Form Functions](#)**WhoAmI**

Displays the user name and full name of the user currently logged-in to the system. When the current logged on user is Windows user this function returns the user's full name in the format of <DomainName>\<UserName>. The names are displayed at the prompt AN.

Syntax**WhoAmI()****Return Value**

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[Name](#), [FullName](#), [UserInfo](#)**Example**

```

/* Display the user's full name and user name at the prompt AN. */
WhoAmI();

```

See Also[Security Functions](#)

Chapter: 47 Server Functions

Server functions control and monitor Trend, Alarm, Report and I/O Servers.

Note: The functions ServerGetProperty and ServerReload can only be called for Alarm, Report and Trend Servers

Server Functions

Following are functions relating to servers.

ServerBrowseClose	The ServerBrowseClose function terminates an active data browse session and cleans up resources associated with the session.
ServerBrowseFirst	The ServerBrowseFirst function places the data browse cursor at the first record.
ServerBrowseGetField	The ServerBrowseGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.
ServerBrowseNext	The ServerBrowseNext function moves the data browse cursor forward one record.
ServerBrowseNumRecords	The ServerBrowseNumRecords function returns the number of records that match the filter criteria.
ServerBrowseOpen	The ServerBrowseOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.
ServerBrowsePrev	The ServerBrowsePrev function moves the data browse cursor back one record.
ServerGetProperty	Returns information about a specified server and can be called from any client.
ServerInfo	Gets client and server information.
ServerInfoEx	Gets client and server information from a specified process in

	a multiprocessor environment.
ServerIsOnline	The ServerIsOnline function checks if the given server can be contacted by the client for giving the online/offline status of the server.
ServerReload	Reloads the server specified by cluster and server name.
ServerRestart	Restart any specific alarm, report, trend or I/O server from any Cicode node in system, without affecting other server processes running on the same machine.
ServerRPC	Calls a remote procedure on the Citect server specified by the <i>ServerName</i> argument.

See Also

[Functions Reference](#)

ServerBrowseClose

The ServerBrowseClose function terminates an active data browse session and cleans up resources associated with the session.

Syntax

ServerBrowseClose(*iSession*)

iSession:

Handle to a browse session previously opened by a [ServerBrowseOpen](#) call.

Return Value

0 (zero) if the server browse session exists, otherwise an error is returned.

Related Functions

[ServerBrowseFirst](#), [ServerBrowseGetField](#), [ServerBrowseNext](#), [ServerBrowseNumRecords](#), [ServerBrowseOpen](#), [ServerBrowsePrev](#)

See Also

[Server Functions](#)

ServerBrowseOpen

The ServerBrowseOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.

Syntax

ServerBrowseOpen(*sFilter* , *sFields* , *sClusters*)

sFilter:

A filter expression specifying the records to return during the browse. An empty string indicates that all records will be returned.

sFields:

Specifies via a comma-delimited string the columns to be returned during the browse. An empty string indicates that the server will return available columns. Supported fields are:

NAME, TYPE, COMMENT, CLUSTER, MODE, NETADDR, PORT,
LEGACYPORT.

See [Browse Function Field Reference](#) for information about fields.

sClusters:

An optional parameter that specifies via a comma delimited string the subset of clusters to browse. An empty string indicates that connected clusters will be browsed.

Return Value

An integer handle to the browse session. Returns -1 on error.

The returned entries will be ordered alphabetically by name.

Related Functions

[ServerBrowseClose](#), [ServerBrowseFirst](#), [ServerBrowseGetField](#), [ServerBrowseNext](#), [ServerBrowseNumRecords](#), [ServerBrowsePrev](#)

See Also

[Server Functions](#)

ServerBrowseFirst

The ServerBrowseFirst function places the data browse cursor at the first record.

Syntax

ServerBrowseFirst(*iSession*)

iSession:

Handle to a browse session previously opened by a [ServerBrowseOpen](#) call.

Return Value

0 (zero) if the server browse session exists, otherwise an error is returned.

Related Functions

[ServerBrowseClose](#), [ServerBrowseGetField](#), [ServerBrowseNext](#),
[ServerBrowseNumRecords](#), [ServerBrowseOpen](#), [ServerBrowsePrev](#)

See Also

[Server Functions](#)

ServerBrowseNext

The ServerBrowseNext function moves the data browse cursor forward one record. If you call this function after you have reached the end of the records, error 412 is returned (Databrowse session EOF).

Syntax

ServerBrowseNext(*iSession*)

iSession:

Handle to a browse session previously opened by a [ServerBrowseOpen](#) call.

Return Value

0 (zero) if the server browse session exists, otherwise an error is returned.

Related Functions

[ServerBrowseClose](#), [ServerBrowseFirst](#), [ServerBrowseGetField](#),
[ServerBrowseNumRecords](#), [ServerBrowseOpen](#), [ServerBrowsePrev](#)

See Also

[Server Functions](#)

ServerBrowsePrev

The ServerBrowsePrev function moves the data browse cursor back one record. If you call this function after you have reached the beginning of the records, error 412 is returned (Databrowse session EOF).

Syntax

ServerBrowsePrev(*iSession*)

iSession:

Handle to a browse session previously opened by a [ServerBrowseOpen](#) call.

Return Value

0 (zero) if the server browse session exists, otherwise an error is returned.

Related Functions

[ServerBrowseClose](#), [ServerBrowseFirst](#), [ServerBrowseGetField](#), [ServerBrowseNext](#), [ServerBrowseNumRecords](#), [ServerBrowseOpen](#)

See Also

[Server Functions](#)

ServerBrowseGetField

The ServerBrowseGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.

Syntax

ServerBrowseGetField(*iSession*, *sFieldName*)

iSession:

Handle to a browse session previously opened by a [ServerBrowseOpen](#) call.

sFieldName:

The name of the field that references the value to be returned. Supported fields are:

NAME, TYPE, COMMENT, CLUSTER, MODE, NETADDR, PORT,
LEGACYPORT.

See [Browse Function Field Reference](#) for information about fields.

Return Value

The value of the specified field as a string. An empty string may or may not be an indication that an error has been detected. The last error should be checked in this instance to determine if an error has actually occurred.

Related Functions

[ServerBrowseClose](#), [ServerBrowseFirst](#), [ServerBrowseNext](#), [ServerBrowseNumRecords](#),
[ServerBrowseOpen](#), [ServerBrowsePrev](#)

See Also

[Server Functions](#)

ServerBrowseNumRecords

The ServerBrowseNumRecords function returns the number of records that match the filter criteria.

Syntax

ServerBrowseNumRecords(*iSession*)

iSession:

Handle to a browse session previously opened by a [ServerBrowseOpen](#) call.

Return Value

The number of records that matched the filter criteria. A value of 0 denotes that no records matched. A value of -1 denotes that the browse session is unable to provide a fixed number. This could be the case if the data being browsed changed during the browse session.

Related Functions

[ServerBrowseClose](#), [ServerBrowseFirst](#), [ServerBrowseGetField](#), [ServerBrowseNext](#), [ServerBrowseOpen](#), [ServerBrowsePrev](#)

See Also

[Server Functions](#)

ServerGetProperty

This function returns information about a specified server and can be called from any client.

Note: This function can only be called for Alarm, Report and Trend Servers

Syntax

ServerGetProperty(*sServer*, *sProperty* [, *sCluster*]).

sServer:

The name of the server to be queried in quotations marks "". Can be prefixed by the name of the host cluster, that is "ClusterName.ServerName".

sProperty:

The name of the requested property. Can be one of the following:

"RDBMemTime" - Returns the date and time of currently loaded RDB (in-memory)

"RDBDiskTime" - Returns the date and time of RDB on disk (compiled)

"LibRDBMemTime" - Date and time of currently loaded cicode library (_library.RDB)

"LibRDBDiskTime" - Date and time of cicode library on disk (_library.RDB)

"LastReloadError" - Error Code from the latest reload

"ReloadStatus" - Returns 1 if the server is reloading, 0 if not

"ReloadProgress" - Reload Progress (in percent) Range: 0 – 100

"SyncStatus" - Returns the startup synchronization status: 0 - Sync complete, 1 - Sync pending, 2 - Sync in progress.

"SyncProgress" - Returns the percentage synchronization completion for trends:
Range: 0-100

Note: The "SyncStatus" and "SyncProgress" properties are only supported for Trend and Alarm servers.

sCluster:

The cluster of the server to be queried in quotation marks "". This parameter is optional. However, if the Server Name is not local or not specified in ClusterName.ServerName format an error is returned.

Return Value

The value of the server property requested.

Related Functions

[ServerInfo](#), [ServerInfoEx](#), [ServerIsOnline](#), [ServerReload](#), [ServerRestart](#)

Example

```
ServerGetProperty("AlarmServer1", "ReloadStatus", "Cluster12");
```

See Also

[Server Functions](#)

ServerInfo

Gets status information on clients and servers.

Note: This function is a non-blocking function and can only access data contained within the calling process; consequently it cannot return data contained in a different server process. This function is not redirected automatically by CitectSCADA runtime. If you want to make a call from one process to return data in another, use MsgRPC() to make a remote procedure call on the other process. Alternatively, use the [ServerInfoEx](#) function that allows you to specify the name of the component from which you want to retrieve data.

Syntax

ServerInfo(*sName*, *nType* [, *ClusterName*])

sName:

The name of the client or server, either "Client", "Server", "Alarm", "Trend", or "Report".

- You can also pass a number instead of the name (but it still needs to be enclosed in quotes). The number represents the target client. For example, if there are 12 clients, passing "3" will get information on the 3rd client.
- If this server is an Alarm, Trend, Report, and I/O server then each client will be attached 4 times. So 12 clients would mean there are 3 CitectSCADA computers using this server - one of which is itself.

nType:

The type of information required (depends on the Name you specify):

"Alarm", "Trend", or "Report" name:

0 - Active flag (returns 1 if this is the active server, 0 if an inactive server).

1 - Number of clients attached to this server.

2 - If this client is attached to the primary or standby server for the specified server name. If Name is "Alarm" and if this client is attached to the primary alarm server, the return value is 0. If this client is attached to the standby, the return value is 1.

3 - The status of the client connection to the specified server name. If Name is "Report" and the client is talking to a report server (either primary or standby), the return value is 1. If not, the return value is 0.

"Client" name:

0 - The computer name, as specified by [LAN]Node.

1 - Not supported.

2 - Not supported.

3 - Not supported.

For modes 1,2 and 3, use [ServerInfoEx](#) instead.

"Server" name:

0 - Not supported.

1 - The number of clients attached to this server. This is the total number of Alarm, Trend Report, and I/O server clients.

"<number>":

0 - The name of the server this client is talking to. For example, "Alarm", "Trend", "Report", or "IOServer".

1 - The login name of the client. This may be an empty string if the client has not logged in.

2 - The CitectSCADA computer name of the client computer.

3 - The time the client logged in.

4 - The number of messages received from this client.

5 - The number of messages sent to this client.

6 - If this client has a licence (1) from this server or not (0).

7 - The type of the licence; full licence (0), View-only Client (1), or Control Client (2).

8 - If the client is remote (1) or local (0).

ClusterName:

The name of the cluster that the server belongs to. This is only relevant if:

- The Name is "alarm", "report", or "trend"; AND
- The type of information required, *nType*, is 2 or 3.

Return Value

Status information specified by *nType*.

Related Functions

[ServerGetProperty](#), [ServerInfoEx](#), [ServerIsOnline](#), [ServerReload](#), [ServerRestart](#)

Example

```
sSrvInfo=ServerInfo("Report",0);
IF sSrvInfo THEN
    ! This is a primary report server.
ELSE
    ! This is a stand-by report server.
END
/* Get and store the names of clients attached to this server */
iCount = 0;
iClients = ServerInfo("Server", 1);
WHILE iCount < iClients DO
    sName[iCount] = ServerInfo(IntToStr(iCount), 2);
    iCount = iCount + 1;
END
```

See Also

[Server Functions](#)

ServerInfoEx

Gets status information on clients and servers from a specified component in a multiprocess runtime environment.

When this function is called, the system redirects the call to the process that contains the specified component. If the specified component is in the calling process, the call is not redirected. If the specified component is not one of the servers listed in the *sComponent* argument description (see below), or if the system cannot find the component from the connected local processes, a hardware alarm is raised.

Syntax

ServerInfoEx(sName, nType, sComponent [, ClusterName] [, ServerName])

sName:

The name of the client or server, either "Client", "Server", "Alarm", "Trend", or "Report".

- You can also pass a number instead of the name (but it still needs to be enclosed in quotes). The number represents the target client. For example, if there are 12 clients, passing "3" will get information on the 4th client.

- If this server is an Alarm, Trend, Report, and I/O server then each client will be attached 4 times. So 12 clients would mean there are 3 CitectSCADA computers using this server - one of which is itself.

nType:

The type of information required (depends on the Name you specify): "Alarm", "Trend", or "Report" name:

0 - Active flag (returns 1 if this is the active server, 0 if an inactive server).

1 - Number of clients attached to this server.

2 - If this client is attached to the primary or standby server for the specified server name. If Name is "Alarm" and if this client is attached to the primary alarm server, the return value is 0. If this client is attached to the standby, the return value is 1.

3 - The status of the client connection to the specified server name. If Name is "Report" and the client is talking to a report server (either primary or standby), the return value is 1. If not, the return value is 0.

Note: If the sComponent is "IOServer" and the sName is "Client", Types 1 and 2 return an empty string, as the concept of primary and standby servers does not apply to IOServers.

"Client" name:

0 - The computer name, as specified by [LAN]Node.

1 - The primary server name, as specified in the server configuration forms in Project Editor.

2 - The secondary server name, as specified , as specified in the server configuration forms in Project Editor.

3 - The name of the INI file being used, for example, Citect.INI.

"Server" name:

0 - The server name, as specified in the server configuration forms in Project Editor.

1 - The number of clients attached to this server. This is the total number of Alarm, Trend Report, and I/O server clients.

"<number>":

0 - The name of the server this client is talking to. For example, "Alarm", "Trend", "Report", or "IOServer".

1 - The login name of the client. This may be an empty string if the client has not logged in.

2 - The CitectSCADA computer name of the client computer.

3 - The time the client logged in.

- 4 - The number of messages received from this client.
- 5 - The number of messages sent to this client.
- 6 - If this client has a licence (1) from this server or not (0).
- 7 - The type of the licence; full licence (0), View-only Client (1), or Control Client (2).
- 8 - If the client is remote (1) or local (0).

sComponent:

Specifies the component name from which to retrieve the status information:

- " " - An empty string causes the function to run on the calling process.
- "Alarm" - Redirects the function to the alarm server process.
- "Trend" - Redirects the function to the trend server process.
- "Report" - Redirects the function to the report server process.
- "IOServer" - Redirects the function to the I/O server process.

Note: If *sName* is "Client", the function will NOT be redirected to the component specified by *sComponent*. Instead, the function gets information of type *nType* from the current client connection to the component specified by *sComponent*.

ClusterName:

The name of the cluster that the server belongs to. This is only relevant if:

- The Name is "alarm", "report", or "trend"; AND
- The type of information required, *nType*, is 2 or 3.

ServerName:

Specifies the name of the the I/O Server. This parameter is only required if you are running more than one I/O server process from the same cluster on the same computer and need to instruct the system which process to redirect to. The argument is enclosed in quotation marks "".

Return Value

Status information specified by *nType*.

Related Functions

[ServerGetProperty](#), [ServerInfo](#), [ServerIsOnline](#), [ServerReload](#), [ServerRestart](#)

Example

This example gets the server information from the report process.

```

sSrvInfo=ServerInfoEx("Report",0, "Report");
IF sSrvInfo THEN
    ! This is a primary report server.
ELSE
    ! This is a stand-by report server.
END
/* Get and store the names of clients attached to the report server */
iCount = 1;
iClients = ServerInfoEx("Server", 1, "Report");
WHILE iCount <= iClients DO
    sName[iCount] = ServerInfoEx(IntToStr(iCount), 2, "Report");
    iCount = iCount + 1;
END

```

See Also[Server Functions](#)**ServerIsOnline**

The **ServerIsOnline** function checks if the given server can be contacted by the client for giving the online/offline status of the server.

Syntax**ServerIsOnline(*sServerName*[, *sClusters*])***sServerName*:

The name of the server to be queried in quotations marks "".

sCluster:

The cluster of the server to be queried in quotation marks "". An empty string indicates that the default cluster will be used.

Return Value

An integer value of online/offline status. Returns 1 for online, 0 for offline.

Related Functions[ServerGetProperty](#), [ServerInfo](#), [ServerInfoEx](#), [ServerReload](#), [ServerRestart](#)**Example**

```
ServerIsOnline("AlarmServer1", "Cluster12");
```

See Also[Server Functions](#)

ServerReload

This function can only be called for Alarm, Report and Trend Servers and reloads the server specified by cluster and server name. If the server is not found an error is returned. ServerReload can be used in blocking or non-blocking modes using the bSync parameter. When used in non-blocking mode, the server status can be returned using the [ServerGetProperty](#) function.

For this function to be successful you need to set the [LAN]AllowRemoteReload parameter in the Citect.ini file to "1".

It is recommended that the ServerGetProperty cicode function be used with the *LibRDBMemTime* and *LibRDBDiskTime* properties to check if there is a change to the Cicode library before attempting a reload. Following a reload please check the corresponding server's syslog.dat file for any reload messages. The cicode changes will not be reloaded, therefore a restart may be more appropriate.

WARNING

UNINTENDED EQUIPMENT OPERATION

Restart the server process if a "Cicode library timestamp differs" error is detected. A library mismatch is indicated on the server in either the hardware alarm or the server's syslog.dat file.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Note: A message in the Syslog.dat file and hardware alarm of "Cicode library timestamp differs" (error code 454) will be raised if the Cicode library used by one or more server runtime databases is different from the one in memory. The timestamps will be different if the project has been fully recompiled (with or without Cicode modification), or if the project has been incrementally recompiled after any Cicode has been modified.

Syntax

ServerReload(*sServer* [*sCluster*] [*bSync*])

sServer:

The name of the server to be reloaded in quotations marks "". Can be prefixed by the name of the host cluster, that is "*ClusterName.ServerName*".

sCluster:

The cluster of the server to be reloaded in quotation marks "". This parameter is optional. However, if the server name is not local or not specified in ClusterName.ServerName format, an error is returned.

bSync:

Specifies whether the function operates in blocking or non-blocking mode. If bSync is set to 1, the function will not return until the server reload is complete. The reload is complete when all the records of all rdb files have been processed and updated. Blocking mode cannot be used from a foreground task (for example on graphic pages). When bSync is set to 0, the function operates in non-blocking mode. You can get the latest status of the reload using the [ServerGetProperty](#) function. Default value is 0.

Return Value

0 (zero) if the function was successful. Returns an error if unsuccessful. Outline of errors:

418 - Server is not found or if the client is not allowed to visit the cluster described in "sCluster".

281 - Server is sitting on a remote machine to the client and the connection towards the server is not available.

519 - Server connection was available but interrupted.

451 - Server is busy with previous load request.

Related Functions

[ServerGetProperty](#), [ServerInfo](#), [ServerInfoEx](#), [ServerIsOnline](#), [ServerRestart](#)

Example

```
ServerReload("AlarmServer1", "Cluster1", 0);
```

See Also

[Server Functions, Server Side Online Changes](#)

ServerRestart

Restart any specific alarm, report, trend or I/O server from any Cicode node in system, without affecting other server processes running on the same machine.

For this function to be successful you need to set the [\[Shutdown\]NetworkStart](#) parameter in the Citect.ini file to "1".

Syntax

INT error = **ServerRestart** (STRING sServerName, STRING sCluster = "")

sServerName:

The name of the server to restart

sCluster:

The cluster the server belongs to. This parameter is optional. If sCluster is not specified the current system cluster is used.

Return Value

0 (zero) if successful, otherwise one of the following error codes is returned:

256 - General software error

292 - Invalid function

403 - Cluster not found

418 - No server of type on cluster

512 - Time out error

513 - Access denied error

See Also

[Cicode and General errors.](#)

Related Functions

[ServerGetProperty](#), [ServerInfo](#), [ServerInfoEx](#), [ServerIsOnline](#), [ServerReload](#)

Example

```
ServerRestart("AlarmServer1", "Cluster1");
```

See Also

[Server Functions](#)

Chapter: 48 Statistical Process Control Functions

SPC (Statistical Process Control) functions allow you to alter the properties and parameters that affect the SPC calculations. By using the SPC functions you can also gain direct access to the SPC data and alarms.

SPC Functions

Following are functions relating to Statistical Process Control:

<u>SPCAlarms</u>	Returns the status of the specified SPC alarm.
<u>SPCClientInfo</u>	Returns SPC data for the given SPC tag.
<u>SPCGe-tHistogramTable</u>	Returns an array containing the frequency of particular ranges for the given SPC tag.
<u>SPCGet-SubgroupTable</u>	Returns an array containing the specified subgroup's elements with the mean, range and standard deviation.
<u>SPCPlot</u>	Generates a single page print showing three separate trends of the SPC Mean, Range, and Standard Deviation.
<u>SPCProcessXRSGet</u>	Gets the process mean, range and standard deviation overrides.
<u>SPCProcessXRSSet</u>	Sets the process mean, range and standard deviation overrides.
<u>SPCSelLimit</u>	Sets the upper or lower control limits of X-bar, range, or standard deviation charts.
<u>SPCSpecLimitGet</u>	Gets the upper and lower specification limits for the specified tag.
<u>SPCSpecLimitSet</u>	Sets the upper and lower specification limits for the specified tag.
<u>SPCSub-groupSizeGet</u>	Gets the size of a subgroup for the specified SPC tag.
<u>SPCSub-groupSizeSet</u>	Sets the subgroup size for the specified SPC tag.

See Also

[Functions Reference](#)

SPCalarms

Returns the status of the specified SPC alarm. This function is used to configure SPC alarms, by defining alarms with this trigger in Advanced Alarms.

This function can only be used if the Alarm Server is on the current machine. When the Alarm Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

SPCalarms(*sSPCTag*, *AlarmType*)

sSPCTag:

The SPC Tag name as defined in SPC Tags.

AlarmType:

The description of the alarm type. The following types are valid:

XFreak	XGradualUp	XStratification
XOutsideCL	XGradualDown	XMixture
XAboveUCL	XUpTrend	ROutsideCL
XBelowLCL	XDownTrend	RAboveUCL
XOutsideWL	XErratic	RBelowLCL

Return Value

Alarm status, ON (1) or OFF (0).

Related Functions

[AlarmAck](#)

Example

Advanced Alarms

Alarm Tag Feed_SPC_XBLCL

Alarm Desc Process mean below LCL

Expression	SPCALarms("Feed_SPC", XBelowLCL)
Comment	Trigger an alarm when XBelowLCL condition becomes true.
Advanced Alarms	
Alarm Tag	Temp_SPC_GRADUP
Alarm Desc	Mean is drifting up
Expression	SPCALarms("Temp_SPC", XGradualUp)
Comment	Trigger an alarm if mean drifts up.

See Also

[SPC Functions](#)

SPCClientInfo

Returns SPC data for the given SPC tag. The information retrieved through this function is from the cache maintained by the client. This function will give a faster response than the related functions which access the SPC (trend) server.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCClientInfo(*sSPCTag*, *iType*)

sSPCTag:

The SPC Tag name as defined in SPC Tags.

iType:

The information to be returned:

- 1 - Subgroup Size
- 2 - No. of Subgroups
- 3 - Process Mean (x double bar)
- 4 - Process Range
- 5 - Process Standard Deviation
- 6 - Lower Specification Limit (LSL)
- 7 - Upper Specification Limit (USL)
- 8 - Cp - Process Capability Actual
- 9 - Cpk - Process Capability Potential

10 - Process Skewness

11 - Process kurtosis

Return Value

The requested data specified by iType. It is of type REAL.

Related Functions

[SPCSpecLimitGet](#), [SPCProcessXRSGet](#), [SPCSubgroupSizeGet](#)

Example

```
/* This function will check the capability of a particular SPC tag.*/
REAL
FUNCTION
CheckCapability(STRING sTAG)
    REAL rReturn;
    rReturn = SPCClientInfo(sTag, 8);
    !rReturn holds the inherent capability value
    IF rReturn > 1.0 THEN
        Message(sTag + "Assessment","The process is Capable.",64);
    ELSE
        Message(sTag + "Assessment","The process is not Capable.",64);
    END
    Return rReturn;
END
```

See Also

[SPC Functions](#)

SPCGetHistogramTable

Returns an array containing the frequencies of particular ranges for the given SPC tag. The histogram structure is implied in the order of the table as follows - the first array element is the data less than -3 sigma. The second value is the data between -3 sigma and -3 sigma plus the bar width etc. The last value is the data greater than +3 sigma.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCGetHistogramTable(*sSPCTag*, *iNoBars*, *TableVariable*)

sSPCTag:

The SPC Tag name as defined in SPC Tags.

iNoBars:

The number of bars in the table. The valid range is restricted to values from 7 to 100. This also indicates the size of the array to be returned.

TableVariable:

The Cicode array that will store the histogram data. This variable needs to be defined as a global array of type REAL. The number of elements in the array needs to be equal to (or greater than) iNoBars.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. The histogram table is written to *TableVariable*.

Related Functions

[TableMath](#)

Example

```
/* This function will get the maximum frequency present in the
histogram of a particular SPC tag.*/
INT iFrequency[7];
! This variable needs to be global to the file so is declared outside
of the function
INT
FUNCTION
GetMaxFreq(STRING sTAG)
    INT      iError;
    INT      iMax = -1;
    iError = SPCGetHistogramTable(sTag, 7, iFrequency);
    !The elements of iFrequency now hold the histogram table frequencies.
    IF iError = 0 THEN
        ! Get maximum
        iMax = TableMath(iFrequency,7,1,0);
    END
    Return iMax;
END
```

See Also

[SPC Functions](#)

SPCGetSubgroupTable

Returns an array containing the specified subgroup's elements with the mean, range and standard deviation. The data will be in the following order:

Element0, Element1, ... , Element(**n**-1), Mean, Range, StdDev

where **n** is the subgroup size.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCGetSubgroupTable(*sSPCTag*, *iSubgroup*, *TableVariable*)

sSPCTag:

The SPC Tag name as defined in SPC Tags.

iSubgroup:

The number of the subgroup being displayed whose data is to be retrieved. Zero ('0') represents the latest subgroup.

TableVariable:

The first element of the Cicode array that will store the sample data. This variable needs to be defined as a global array of type REAL. The number of elements in the array needs to be equal to (or greater than) the subgroup size + 3.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. The subgroup's data is written to TableVariable.

Related Functions

[TableMath](#)

Example

```
/* This function will get the minimum value present in the sample
data of a particular SPC tag.*/
REAL rSubgroup[8];           ! 5 samples + mean + range + stddev.
! This variable needs to be global to the file, so is declared outside
of the function
REAL
FUNCTION
GetMinSample(STRING sTAG)
    INT      iError;
    REAL      iMin = 0;
    iError = SPCGetSubgroupTable(sTag, 7, rSubgroup);
    !The elements of rSubgroup now hold the group samples, mean, range and stddev.
    IF iError = 0 THEN
        ! Get minimum. Be aware that the range of data is 5
        iMin = TableMath(rSubgroup,5,0,0);
    END
    Return iMin;
END
```

See Also

[SPC Functions](#)

SPCPlot

This function is designed to work only on an SPCXRSChart page. It prints a single page showing three separate trends of the SPC Mean, Range, and Standard Deviation. The Mean needs to be at *AN*, the Range at *AN + 1*, and the Standard Deviation at *AN + 2*. You can specify a title and a comment for the plot, and whether it is printed in color or in black and white.

Syntax

SPCPlot(*sPort*, *AN* [, *sTitle*] [, *sComment*] [, *iMode*])

sPort:

The name of the printer port to which the plot will be printed. This name needs to be enclosed within quotation marks. For example LPT1:, to print to the local printer, or \\Pserver\canon1 using UNC to print to a network printer.

AN:

The animation point at which the Mean chart is currently situated. The Range and Standard Deviation charts need to be on the next two consecutive animation numbers. For example, if the Mean chart is at animation point 40, the Range chart needs to be at animation point 41, and the Standard Deviation chart needs to be at animation point 42.

sTitle:

The title of the trend plot.

sComment:

The comment that is to display beneath the title of the trend plot. You do not have to enter a comment.

iMode:

The color mode of the printer.

0 - Black and White (default)

1 - Color

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnPlot](#), [TrnComparePlot](#), [TrnPrint](#), [PlotOpen](#)

Example

```
/* This function will print the Mean trend (currently displayed at
animation point 40), the Range trend (currently at animation point
41), and the Standard Deviation trend (currently at animation
point 42). The result is a one page, black and white combination
of all three trends, printed to LPT1. */
SPCPlot("LPT1:",40, "CitectSCADA SPC Chart","Gradually
increasing trend",0);
```

See Also

[SPC Functions](#)

SPCProcessXRSGet

Gets the process mean, range, and standard deviation overrides for the specified SPC tag. The values that are returned are the values that are currently being used by the SPC (trend) server, not necessarily the values specified in the SPC Tag definition.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCProcessXRSGet(*sSPCTag*, *XVariable*, *RVariable*, *SVariable* [, *ClusterName*])

sSPCTag:

The SPC Tag name as defined in SPC Tags.

XVariable:

The Cicode variable that stores the process mean (X double bar). This variable needs to be defined as a global of type REAL. A constant is not allowed.

RVariable:

The Cicode variable that stores the range (R). This variable needs to be defined as a global of type REAL. A constant is not allowed.

SVariable:

The Cicode variable that stores the standard deviation (S). This variable needs to be defined as a global of type REAL. A constant is not allowed.

ClusterName:

Specifies the name of the cluster of the SPC tag.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. The process mean is written to XVariable, the process range to RVariable, and the standard deviation to SVariable.

Related Functions

[SPCClientInfo](#), [SPCProcessXRSSet](#)

Example

```
/* This function will set a new override value for Mean, without
   overwriting the values already in place for Standard Deviation and
   Range */
REAL rOldMean;
REAL rRange;
REAL rStdDev;
! These variables need to be global to the file, so are declared
outside of the function
INT
FUNCTION
Tank1SPCNewMean(REAL rNewMean)
    INT iError;
    iError = SPCProcessXRSGet("TANK_1_TEMP", rOldMean, rRange, rStdDev);
    ! If no error, rOldMean, rRange and rStdDev now hold the current values of XRS.
    IF iError = 0 THEN
        iError = SPCProcessXRSSet("TANK_1_TEMP", rNewMean, rRange, rStdDev);
    END
    Return iError;
END
```

See Also

[SPC Functions](#)

SPCProcessXRSSet

Sets the process mean, range and standard deviation overrides for the specified SPC tag. The values entered here will override CitectSCADA's automatic calculations, and the overrides specified in the SPC Tags definition.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCProcessXRSSet(sSPCTag, rMean, rRange, rStdDev [, ClusterName])

sSPCTag:

The SPC Tag name as defined in SPC Tags.

rMean:

The new value of process mean (x double bar) to set.

rRange:

The new value of process range to set.

rStdDev:

The new value of process standard deviation to set.

ClusterName:

Specifies the name of the cluster of the SPC tag.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned.

Related Functions

[SPCProcessXRSGet](#)

Example

See [SPCProcessXRSGet](#)

See Also

[SPC Functions](#)

SPCSetLimit

Sets the upper or lower control limits of X-bar, range, or standard deviation charts. Using this function will only set the controller limits on the Client display which will not affect the SPC Alarms. To set the server control limits, use the SPCProcessXRSSet function.

Syntax

SPCSetLimit(AN, Type, Value, Setting)

AN:

The AN where the SPC chart is located.

Type:

The SPC type:

-
- 1 - X-bar upper control limit
 - 2 - X-bar lower control limit
 - 3 - Range upper control limit
 - 4 - Range lower control limit
 - 5 - Standard deviation upper control limit
 - 6 - Standard deviation lower control limit
 - 7 - X-bar centre line
 - 8 - Range centre line
 - 9 - Standard deviation centre line

Value:

The value for the control limit.

Setting:

Automatic calculation or manual setting of control limits:

- 0 - Automatic
- 1 - Manual

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Example

```
SPCSetLimit(40,1,250,1);
! Sets X-bar upper control limit to 250 at AN40.
```

See Also

[SPC Functions](#)

SPCSpecLimitGet

Gets the process Upper and Lower Specification Limits (USL and LSL) for the specified SPC tag. The values that are returned are the values that are currently being used by the SPC (trend) server, not necessarily the values specified in the SPC Tag definition.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

SPCSpecLimitGet(*sSPCTag*, *LSLVariable*, *USLVariable* [, *ClusterName*])

sSPCTag:

The SPC Tag name as defined in SPC Tags.

LSLVariable:

The Cicode variable that stores the Lower Specification Limit (LSL). This variable needs to be defined as a global of type REAL. Do not specify a constant in this field.

USLVariable:

The Cicode variable that stores the Upper Specification Limit (USL). This variable needs to be defined as a global of type REAL. Do not specify a constant in this field.

ClusterName:

Specifies the name of the cluster of the SPC tag.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. The LSL is written to LSLVariable, while the USL is written to USLVariable.

Related Functions

[SPCClientInfo](#), [SPCSpecLimitSet](#)

Example

```
/* This function will increase the current USL and LSL of the
specified Tag by 10 percent.*/
REAL rLSL;
REAL rUSL;
! These variables need to be global to the file, so are declared
outside of the function
INT
FUNCTION
ExpSLbyPercent(STRING sTAG)
    REAL      rIncPercent = 1.1;
    REAL      rDecPercent = 0.9;
    INT       iError;
    iError = SPCSpecLimitGet(sTag, rLSL, rUSL);
    ! If no error, rLSL and rUSL now hold the current values of
    ! LSL and USL for sTAG
    rLSL = rLSL * rDecPercent;
    rUSL = rUSL * rIncPercent;
    IF iError = 0 THEN
        iError = SPCSpecLimitSet(sTAG, rLSL, rUSL);
    END
    Return iError;
END
! The function would be called as follows;
```

Page Button

Button Text	Expand Temperature Limits
Expression	ExpSLby10Percent("TANK_1_TEMP");

See Also[SPC Functions](#)**SPCSpecLimitSet**

Sets the process Upper and Lower Specification Limits (USL and LSL) for the specified SPC tag. The values entered here will override those specified in the SPC Tags definition.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

SPCSpecLimitSet(*sSPCTag*, *rLSL*, *rUSL* [, *ClusterName*])

sSPCTag:

The SPC Tag name as defined in SPC Tags.

rLSL:

The new value of Lower Specification Limit (LSL) to set.

rUSL:

The new value of Upper Specification Limit (USL) to set.

ClusterName:

Specifies the name of the cluster of the SPC tag.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned.

Related Functions[SPCSpecLimitGet](#)**Example**

See [SPCSpecLimitGet](#)

See Also

[SPC Functions](#)

SPCSubgroupSizeGet

Gets the subgroup size for the specified SPC tag. The value that is returned is the value that is currently being used by the SPC (trend) server, not necessarily the value specified in the SPC Tag definition.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCSubgroupSizeGet(*sSPCTag*, *SizeVariable* [, *ClusterName*])

sSPCTag:

The SPC Tag name as defined in SPC Tags.

SizeVariable:

The Cicode variable that stores the subgroup size. This variable needs to be defined as a global of type INT. A constant is not allowed.

ClusterName:

Specifies the name of the cluster of the SPC tag.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. The subgroup size is written to *SizeVariable*.

Related Functions

[SPCCClientInfo](#), [SPCSubgroupSizeSet](#)

Example

See [SPCSubgroupSizeSet](#).

See Also

[SPC Functions](#)

SPCSubgroupSizeSet

Sets a new subgroup size for the specified SPC tag. The new subgroup size becomes the new size as long as the SPC (trend) server is running. The subgroup size is updated first in the SPC server, which then informs the clients to update. This will force re-calculation of SPC values (UCL and LCL) across the span of any displayed charts.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

This function can only be called while the SPC tag is being displayed on an SPC page.

Syntax

SPCSubgroupSizeSet(*sSPCTag*, *iSize* [, *ClusterName*])

sSPCTag:

The SPC Tag name as defined in SPC Tags.

iSize:

The new size of the subgroup to set.

ClusterName:

Specifies the name of the cluster of the SPC tag.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned.

Related Functions

[SPCSubgroupSizeGet](#)

Example

```
/* This function increments the subgroup size for FEED_RATE_1 by
the specified amount. */
INT iSize;
! This variable needs to be global to the file, so is declared outside
of the function
INT
FUNCTION
IncSubgroupSize(INT iIncrement)
    INT iError;
    iError = SPCSubgroupSizeGet("FEED_RATE_1", iSize);
    ! If no error, iSize now contains the current subgroup size of FEED_RATE_1
    iSize = iSize + iIncrement;
    IF iError = 0 and (iSize > 1) THEN
        iError = SPCSubgroupSizeSet("FEED_RATE_1", iSize );
    END
```

```
    Return iError;
END
```

See Also

[SPC Functions](#)

Chapter: 49 SQL Functions

SQL functions allow you to define, manipulate, and control data in SQL databases and other relational databases. By using the SQL functions (or the device functions with an SQL device), you can directly access data on database servers, mini-computers, and mainframe computers.

SQL Functions

Following are functions related to SQL operations:

<u>Execute- eDTSPkg</u>	Loads and executes a Data Transformation Services package which initiates data transfer between OLE DB data sources.
<u>SQLAppend</u>	Appends a text string to the SQL buffer.
<u>SQLBe- ginTran</u>	Starts a database transaction.
<u>SQLCommit</u>	Commits a transaction to the database.
<u>SQLConnect</u>	Makes a connection to a database system for execution of SQL statements.
<u>SQLDis- connect</u>	Closes a database connection.
<u>SQLEnd</u>	Terminates an SQL query.
<u>SQLErrMsg</u>	Returns an error message from the SQL system.
<u>SQLExec</u>	Executes an SQL query on a database.
<u>SQLFieldInfo</u>	Gets information about the fields or columns selected in an SQL query.
<u>SQLGetField</u>	Gets field or column data from a database record.
<u>SQLInfo</u>	Gets information about a database connection.
<u>SQLNext</u>	Gets the next database record from a SQL query.

SQLNoFields	Gets the number of fields or columns that were returned by the last SQL statement.
SQLNumChange	Gets the number of records that were modified in the last insert, update, or delete SQL statement.
SQLRollBack	Rolls back (or cancels) the last database transaction.
SQLSet	Sets a statement string in the SQL buffer.
SQLTraceOff	Turns off the debug trace.
SQLTraceOn	Turns on the debug trace.

See Also[Functions Reference](#)

ExecuteDTSPkg

Loads and executes a DTS (Data Transformation Services) package which initiates data transfer and transformations between OLE DB data sources.

A DTS package is created using the DTS utility provided in Microsoft SQL Server 7.0. It can be saved in a COM structured file, a Microsoft Repository, or in an SQL Server Database.

All except the first of this function's parameters are optional, and their use will depend on your needs.

Syntax

ExecuteDTSPkg(*sFileOrSQLSvrName* [, *sPkgName*] [, *sParam1*, ... , *sParam5*] [, *sPkgPwd*] [, *sPkgVer*] [, *sLogFile*] [, *sSQLSvrUsr*] [, *sSQLSvrPwd*])

sFileOrSQLSvrName:

The path and name of the file containing the package (for file-based packages), or the SQL Server name (for SQL Server stored packages).

sPkgName:

The package name.

- For file-based packages where only one package is stored in a file, you can ignore this parameter, as the package name defaults to the name of the file.
- If the package has been named differently to the file, or a file contains more than one package, you need to specify the package name. You need to also specify the package name for SQL Server stored packages.

sParam1, ,sParam5:

Five optional variables which may be used as global variables within the DTS package. The variables need to be named Param1, Param2, Param3, Param4, and Param5.

sPkgPwd:

The package password.

The creator of the DTS package may have implemented a password so that unauthorized users cannot access it. In this case, you need to specify the package password. If no password has been implemented, you can omit this parameter.

sPkgVer:

The package version. If you don't specify a version, the most recent version is used.

sLogFile:

An optional path and name for a log file. The log file can track activity such as:

- File DTS package detected
- SQL DTS package detected
- Package initialized successfully
- Package executed sucessfully
- Package execution was not successful

sSQLSvrUsr:

The user name providing access to the SQL Server where the DTS package is stored. A user's account on the SQL Server consists of this user name and, in most cases, a password.

This parameter also determines which method is used to load the package.

If *sSQLSvrUsr* is specified, the package is assumed to be an SQL Server stored package. In this case, the package is loaded using the *LoadFromSQLServer()* method. Otherwise, the package is file-based and *LoadFromStorageFile()* is called.

sSQLSvrPwd:

The password providing access to the SQL Server, if the user's account on the server requires a password.

Return Value

0 (zero) if the package was executed successfully, otherwise a DTS error number is returned.

Example

```
/* File-based package with one package per file, where the package
```

```
name is the same as the file name.*/
iResult = ExecuteDTSPkg("c:\dtspackages\package.dts");
/*SQL Server stored package with additional parameters */
iResult = ExecuteDTSPkg("Server1", "TestPackage", "Param1", "Param2", "Param3",
"Param4", "Param5", "Fred", "1", "c:\packages\PkgLog.txt", "jsmith", "secret");
```

See Also

[SQL Functions](#)

SQLAppend

Appends a statement string to the SQL buffer. Cicode cannot send an SQL statement that is longer than 255 characters. If you have an SQL statement that is longer than the 255 character limit, you can split the statement into smaller strings, and use this function to append the statements in the SQL buffer.

Syntax

SQLAppend(*hSQL*, *String*)

hSQL:

The handle to the SQL connection, returned from the [SQLConnect\(\)](#) function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

String:

The statement string to append to the SQL buffer.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLSet](#), [SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#)

Example

See [SQLSet](#)

See Also

[SQL Functions](#)

SQLBeginTran

Starts a database transaction. When you make a transaction, your changes are not written to the database until you call the SQLCommit() function. Alternatively, you can use the SQLRollBack function() to discard all changes made during the transaction.

After you begin a transaction, you need to call either SQLCommit() to save the changes or SQLRollBack() to discard the changes. You need to use one of these functions to complete the transaction and release all database locks. Unless you complete the transaction, you cannot successfully disconnect the SQL connection.

A single database connection can only handle one transaction at a time. After you call SQLBeginTran(), you need to complete that transaction before you can call SQLBeginTran() again.

If you disconnect from a database while a transaction is active (not completed), CitectSCADA automatically "rolls back" the transaction any changes you made to the database in that transaction are discarded.

You do not need to begin a transaction to modify a database. Any changes you make to a database before you call the SQLBeginTran() are automatically committed, and no database locks are held.

The SQLBeginTran() function is not supported by all databases. If the function is not performing as you expect, check that both your database and ODBC driver support transactions. Refer to the documentation for your database for more information on transactions.

Syntax

SQLBeginTran(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

```
/* Increase each employee's salary and superannuation by a
specified amount. If any errors occur, the changes are aborted */
INT
FUNCTION
PayIncrease(STRING sIncrease)
    INT hSQL;
    INT Count1;
    INT Count2;
    hSQL = SQLConnect("DRV=QEDBF");
    SQLBeginTran(hSQL);
    SQLExec(hSQL, "UPDATE C:\DATA\EMPLOYEE SET Salary = Salary + " +sIncrease);
    Count1 = SQLNumChange(hSQL);
    SQLExec(hSQL, "UPDATE C:\DATA\EMPLOYEE SET Super = Super + " +sIncrease);
    Count2 = SQLNumChange(hSQL);
    IF Count1 = Count2 THEN
        SQLCommit(hSQL);
    ELSE
        SQLRollBack(hSQL);
    END
    SQLEnd(hSQL);
    SQLDisconnect(hSQL);
END
```

See Also

[SQL Functions](#)

SQLCommit

Commits (to the database) all changes made within a transaction. If you call the SQLBeginTrans() function to begin a transaction, you need to call the SQLCommit() function to save the changes you make to the database during that transaction (with the Insert, Delete, and Update SQL commands).

The SQLCommit() and SQLRollBack() functions both complete a transaction and release all database locks. But while the SQLCommit() function saves all changes made during the transaction, the SQLRollBack() function discards these changes. Unless you call the SQLCommit() function before you disconnect the database, CitectSCADA automatically rolls back the transaction any changes you made to the database in that transaction are discarded.

The SQLCommit() function could affect different databases in different ways. If the function is not performing as you expect, check that your database is ODBC compatible. Refer to the documentation for your database for information on committing transactions.

Syntax

SQLCommit(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the SQLErrMsg() function).

Related Functions

[SQLBeginTran](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLBeginTran](#)

See Also

[SQL Functions](#)

SQLConnect

Makes a connection to a database system, and returns a handle to the connection for use by the other SQL functions. Through this connection, you can execute SQL statements in the specified database. You need to call this function before any other SQL function.

You only require one connection for each database system to be accessed (for example, Oracle, dBASE, Excel, etc.).

It is strongly recommended not to use an SQL database for storage of real-time data (such as alarms), because SQL databases do not provide real-time performance when accessing database data. Only use an SQL database where data transfer is not essential (for example, recipes or reports). If you try to use SQL to store real time data, Citect-SCADA's performance could be greatly decreased.

Syntax

SQLConnect(*sConnect*)

sConnect:

The connection string, in the format:

<attribute>=<value>[;<attribute>=<value>. . .]

The following attributes can be used in a connection string:

DSN	Data Source Name. The name of the data source defined with the ODBC utility in the Windows Control Panel. You need to use the DSN attribute, unless you are using CitectSCADA v2.01 or earlier.
DLG	Dialog box. Set DLG to 1 to display a dialog box that allows the user to input their user ID, password, and connection string. DLG is an optional attribute.
UID	User name or Authorization/Login ID. Check the documentation for your ODBC driver and database to see if you need to use the UID attribute.
PWD	Password. Check the documentation for your ODBC driver and database to see if you need to use the PWD attribute.
MOD- IFY SQL	The ability of CitectSCADA to understand and accept native SQL depends on the database driver being used. Set MODIFYSQL to 1 (the default) for an ODBC-compliant SQL. Set MODIFYSQL to 0 to use the native SQL syntax of the database system, as well as for any CitectSCADA databases you created with versions 2.01 or earlier, that employ database-specific SQL statements. The Q+E ODBC database drivers are backward compatible with those supplied with earlier versions of CitectSCADA.
REREAD AFTER UPDATE	Set to 1 to reread records from the database after updating them. Use this attribute to get the correct value of automatically updated columns, such as time and date stamps.
REREAD AFTER INSERT	Set to 1 to reread records from the database after inserting into it. Use this attribute to get the correct value of automatically-updated columns, such as time and date stamps.
DRV	Use the DRV attribute for compatibility with CitectSCADA v2.01 and earlier. Use the DRV instead of the data source name (DSN) in the connection string. It is strongly recommended not to use DRV in new CitectSCADA applications.

CitectSCADA recognizes the above attributes for all the database systems in the table below, but not all these attributes are provided for all databases. The asterisks (*) beside each database indicate the attributes you need to use to connect to that database. The acceptable values for each attribute also vary according to the database system, so select from the list to see the attributes and values:

DATABASE SYSTEM	DSN	UID	PWD	DRV
-----------------	-----	-----	-----	-----

Btrieve Files	*			QEBTR
dBASE Files	*			QEDBF
EXCEL Files	*			QEXLS
IBM DB2	X	X	X	QEDEB2
Informix				
INGRES	*	*		QEING
Netware SQL	*			QEXQL
Oracle	*	*	*	QEORA
OS/2 and DB2/2	*	*	*	QEDEE
Paradox	*	*		QEPDX
SQLBase/ (Gupta)	*	*	*	QEGUP
SQL Server	*	*	*	QELESS
Text Files	*	*		QETXT
XDB Databases	*	*	*	QEXDB

DRV:

DRV names are included only for maintaining CitectSCADA applications built using v2.01 or earlier. For these early version, use DRV instead of the data source name (DSN).

X:

No longer supported directly. See information on the OS/2 and DB2/2 database drivers and the "Q+E Database Drivers Reference Manual".

Return Value

The SQL connection handle if the connection is successful, otherwise -1 is returned. (For details of the 307 error code, call the SQLErrMsg() function). The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

```
/* Make a connection to an SQL server and select the name field
from each record in the employee database. */
FUNCTION
ListNames()
    INT hSQL;
    STRING sName;
    INT Status;
    hSQL = SQLConnect ("DSN=MyDatabase;UID=billw;SRVR=CI1");
    IF hSQL <> -1 THEN
        Status = SQLExec (hSQL, "SELECT NAME FROM EMPLOYEE");
        IF Status = 0 THEN
            WHILE SQLNext (hSQL) = 0 DO
                sName = SQLGetField (hSQL, "NAME");
                ..
            END
            SQLEnd (hSQL);
        ELSE
            Message ("Error", SQLErrMsg (), 48);
        END
        SQLDisconnect (hSQL);
    ELSE
        Message ("Error", SQLErrMsg (), 48);
    END
END
```

See Also

[SQL Functions](#)

SQLDisconnect

Closes a database connection. You should close all connections to databases before you shut down CitectSCADA, to release system resources.

For each active transaction (that is, for each SQLBeginTran() call), you should complete the transaction before you disconnect from the database call SQLCommit() to save your changes, or SQLRollBack() function to discard changes. If you call SQLDisconnect() while a transaction is still active, CitectSCADA automatically "rolls back" the transaction any changes you made to the database in that transaction are discarded.

CitectSCADA also automatically ends any queries that are active when the database is disconnected. If you have called SQLExec() during a database connection, you need to call SQLEnd() before you disconnect from the database or the disconnection attempt might not succeed.

Syntax

SQLDisconnect(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

You should not call SQLErrMsg() if SQLDisconnect() returns zero (that is, if the disconnection is successful). SQLErrMsg() would provide information about a connection that does not exist; the information could be meaningless.

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLConnect](#)

See Also

[SQL Functions](#)

SQLEnd

Ends the execution of an SQL query (from the latest SQLExec() call). If you have called the SQLExec() function from within a database connection, you should call SQLEnd() before you disconnect from that database. When the SQLEnd() function ends the execution of the current SQL query, it frees the memory that was allocated for that query.

Only one query can be active at a time, so you do not need to end one query before you execute another query each time you call SQLEexec(), the previous query (through a previous SQLEexec() call) is automatically ended. Similarly, CitectSCADA automatically ends the latest query when it disconnects the database, even if you have not called SQLEnd(). However, the SQLEnd() function aids efficiency SQLEnd() releases the memory that was allocated when the latest query was executed.

Syntax

SQLEnd(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the SQLErrMsg() function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLErrMsg](#), [SQLEexec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLConnect](#)

See Also

[SQL Functions](#)

SQLErrMsg

Returns an error message from the SQL system. If a 307 error code occurs when one of the SQL functions is called, an SQL error message is generated. Call this function to get that error message.

Syntax

SQLErrMsg()

Return Value

The [error](#) message (as a string).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLConnect](#)

See Also

[SQL Functions](#)

SQLExec

Executes an SQL query on a database. With this function, you can execute any SQL query or command supported by the SQL database. Only "CHAR" type fields are supported in database tables.

Keywords such as "DATE", "TIME", and "DESC" cannot be used as field names by some database systems. To use fields with these names, you need to append underscores to the names (for example, "TIME_ ", "DATE_ ", "DESC_ ").

The SQLNext() function needs to be called after the SQLExec() function before you can access data in the first record.

Only one query can be active at a time, so there is no need to end one query before you execute another query; each time you call SQLExec(), the previous query (through a previous SQLExec() call) is automatically ended. Similarly, CitectSCADA automatically ends the latest query when it disconnects the database, even if you have not called SQLEnd(). However, the SQLEnd() function aids efficiency; SQLEnd() releases the memory that was allocated when the latest query was executed.

Syntax

SQLExec(*hSQL*, *sSelect*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

sSelect:

The SQL query to be sent to the SQL database.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the SQLErrMsg() function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

These examples assume that the following tables are setup in a SQL server (with the name configured in Windows Control Panel) and opened with the SQLConnect() function:

PEOPLE

SURNAME	FIRSTNAME	OCCUPATION	DEPARTMENT
MARTIAN	MARVIN	ENGINEER	MANAGEMENT
CASE	CARRIE	SUPPORT	CITECT
LIGHT	LARRY	PROGRAMMER	CITECT
BOLT	BETTY	ENGINEER	SYSTEMS

PHONE

SURNAME	NUMBER
MARTIAN	5551000
CASE	5551010
BOLT	5551020
LIGHT	5551030

Each SQL string (sSQL) should be encased within the SQLExec function, for example:

```
SQLExec(hSQL, sSQL);
```

To add a record to a table:

```
sSQL = "INSERT INTO PEOPLE (SURNAME, FIRSTNAME, OCCUPATION,
DEPARTMENT)           VALUES ('ALLEN', 'MATTHEW', 'PROGRAMMER', 'CITECT')";
```

This SQL command changes the PEOPLE table to:

PEOPLE

SURNAME	FIRSTNAME	OCCUPATION	DEPARTMENT
MARTIAN	MARVIN	ENGINEER	MANAGEMENT
CASE	CARRIE	SUPPORT	CITECT
LIGHT	LARRY	PROGRAMMER	CITECT
BOLT	BETTY	ENGINEER	SYSTEMS
ALLEN	MATTHEW	PROGRAMMER	CITECT

To remove records from a table:

```
sSQL = "DELETE FROM (PEOPLE, PHONE) WHERE SURNAME='MARTIAN'";
SQLBeginTran(hSQL);
SQLExec(hSQL,sSQL);
IF (Message("Alert", "Do you really want to DELETE MARTIAN", 33) = 0) THEN
    SQLCommit(hSQL);
ELSE
    SQLRollback(hSQL);
END
```

Assuming that OK was clicked on the Message Box, the tables change to:

PEOPLE

SURNAME	FIRSTNAME	OCCUPATION	DEPARTMENT
CASE	CARRIE	SUPPORT	CITECT
LIGHT	LARRY	PROGRAMMER	CITECT
BOLT	BETTY	ENGINEER	SYSTEMS

PHONE

SURNAME	NUMBER
CASE	5551010
BOLT	5551020
LIGHT	5551030

To change a record:

```
sSQL = "UPDATE PEOPLE SET OCCUPATION='SUPPORT' WHERE
FIRSTNAME='LARRY'";
This SQL command changes the PEOPLE table to:
```

PEOPLE

SURNAME	FIRSTNAME	OCCUPATION	DEPARTMENT
MARTIAN	MARVIN	ENGINEER	MANAGEMENT
CASE	CARRIE	SUPPORT	CITECT
LIGHT	LARRY	SUPPORT	CITECT
BOLT	BETTY	ENGINEER	SYSTEMS

To select a group of records from a table:

```
sSQL = "SELECT SURNAME FROM PEOPLE WHERE OCCUPATION='ENGINEER'";
```

This SQL command will return the following table back to CitectSCADA. The table can then be accessed by the SQLNext() function and the SQLGetField() functions.

CITECT TABLE for hSQL

SURNAME
MARTIAN
BOLT

You can also select data using a much more complete SQL string, for example:

```
sSQL = "SELECT (SURNAME, OCCUPATION, NUMBER) FROM (PEOPLE, PHONE)
```

```
WHERE DEPARTMENT='CITECT' AND PEOPLE.SURNAME = PHONE.SURNAME';
```

This SQL command retrieves the following table:

SURNAME	OCCUPATION	NUMBER
CASE	SUPPORT	5551010
LIGHT	PROGRAMMER	5551030

To extract information from a table:

```
STRING sInfo[3][10]
int i = 0;
WHILE ((SQLNext(hSQL) = 0) and (i < 10)) DO
    sInfo[0][i] = SQLGetField(hSQL, "SURNAME");
    sInfo[1][i] = SQLGetField(hSQL, "OCCUPATION");
    sInfo[2][i] = SQLGetField(hSQL, "NUMBER");
END
```

This code example leaves the information in the *sInfo* two dimensional array as follows:

sInfo

0	1	2
0	CASE	SUPPORT
1	LIGHT	PROGRAMMER
2		
3		
4		
...		

See Also

[SQL Functions](#)

[SQLFieldInfo](#)

Gets information about the fields or columns selected by a SQL query. The function returns the name and width of the specified field. If you call the function within a loop, you can return the names and sizes of all the fields in the database.

Keywords such as "DATE", "TIME", and "DESC" cannot be used as field names by some database systems. To use fields with these names, you need to append underscores to the names (for example, "TIME_", "DATE_", "DESC_").

Syntax

SQLFieldInfo(*hSQL*, *hField*, *sName*, *Width*)

hSQL:

The handle to the SQL connection, returned from the `SQLConnect()` function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

hField:

The field (or column) handle, indicating the position of the field in the database.

sName:

A string variable in which the function stores the field name.

Width:

An integer variable in which the function stores the field width.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

```
! Lists all fields in the Employee database
FUNCTION
ListFields()
    INT hSQL;
    STRING sField;
    INT Count;
    INT Width;
    INT Index;
```

```

SQLTraceOn ("C:\DATA\TRACE.LOG");
hSQL = SQLConnect ("DRV=QEDBF");
SQLExec (hSQL, "SELECT * FROM C:\DATA\EMPLOYEE");
Count = SQLNoFields (hSQL);
Index = 0;
WHILE Index < COUNT DO
    SQLFieldInfo (hSQL, Index, sField, Width);
    ..
END
SQLEnd (hSQL);
SQLDisconnect (hSQL);
SQLTraceOff ();
END

```

See Also[SQL Functions](#)

SQLGetField

Gets field or column data from a database record. To get the database record, use the SQLExec() and SQLNext() functions.

Keywords such as "DATE", "TIME", and "DESC" cannot be used as field names by some database systems. To use fields with these names, you need to append underscores to the names (for example, "TIME_", "DATE_", "DESC_").

Syntax**SQLGetField**(*hSQL*, *sField*)*hSQL*:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

sField:

The name of the field or column.

Return Value

The field or column data (as a string). A null string is returned if the field or column does not contain data.

The maximum length of the return data is 255 characters. If the returned data is longer than this, the function will return error 306.

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#),
[SQLExec](#), [SQLFieldInfo](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#),
[SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLConnect](#)

See Also

[SQL Functions](#)

SQLInfo

Gets information about a database connection.

Syntax

SQLInfo(*hSQL*, *Type*)

hSQL:

The handle to the SQL connection, returned from the [SQLConnect\(\)](#) function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Type:

The type of information to get:

0 - The connection string

1 - The current SQL statement

2 - The current database filename (only works with SQL device)

3 - The SQL format handle

4 - The current Q+E library SQL handle. This handle can be used with functions in the Q+E library which can be called in Cicode with the DLL functions.

Return Value

The information (as a string).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#),
[SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRoll-](#)
[Back](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

```
SQLInfo(1, 2);
```

See Also

[SQL Functions](#)

SQLNext

Gets the next database record from an SQL query. Use the SQLExec() function to select a number of records or rows from the SQL database, and then use the SQLNext() function to step through each record separately.

Syntax

SQLNext(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLConnect](#)

See Also

[SQL Functions](#)

SQLNoFields

Gets the number of fields or columns that were returned by the last SQL statement.

Syntax

SQLNoFields(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

The number of fields. A value of 0 is returned if no fields were returned or if an [error](#) has been detected. (For details of an error, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLFieldInfo](#)

See Also

[SQL Functions](#)

SQLNumChange

Gets the number of records that were modified in the last SQL Insert, Update, or Delete statement.

Syntax

SQLNumChange(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

The number of records that were modified. A value of 0 is returned if no fields were returned or if an [error](#) has occurred. (For details of an error, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLRollBack](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLBeginTran](#)

See Also

[SQL Functions](#)

SQLRollBack

Rolls back (discards) all changes made to the database within the current transaction. If you call the SQLBeginTrans() function to begin a transaction, you are not committed to changes to the database made by the Insert, Delete, and Update commands until you call the SQLCommit() function. You can discard these changes by calling the SQLRollBack() function.

You can only call the SQLRollBack() function if you have called SQLBeginTran() to begin a transaction. You do not need to begin a transaction to modify a database, but any changes you make to a database outside of a transaction are automatically committed.

The SQLRollBack() function could affect different databases in different ways. If the function is not performing as you expect, check that your database is ODBC-compatible. Refer to the documentation for your database for more information on rolling back transactions.

Syntax

SQLRollBack(*hSQL*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLTraceOff](#), [SQLTraceOn](#)

Example

See [SQLBeginTran](#)

See Also

[SQL Functions](#)

SQLSet

Sets a statement string in the SQL buffer. Cicode cannot send an SQL statement that is longer than 255 characters. If you have an SQL statement that is longer than the 255 character limit, you can split the statement into smaller strings, and use this function and the SQLAppend() function to append the statements in the SQL buffer.

Syntax

SQLSet(*hSQL*, *String*)

hSQL:

The handle to the SQL connection, returned from the SQLConnect() function. The SQL connection handle identifies the table where details of the associated SQL connection are stored.

String:

The statement string to set in the SQL buffer. The string needs to contain the first part of an SQL statement.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the SQLErrMsg() function).

Related Functions

[SQLAppend](#), [SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#)

Example

```
hSQL = SQLConnect ("DRV=QEDBF") ;
SQLBeginTran(hSQL) ;
SQLSet(hSQL, "SELECT *")
SQLAppend(hSQL, " FROM EMP");
SQLAppend(hSQL, " ORDER BY last_name");
SQLExec(hSQL, "");
```

See Also

[SQL Functions](#)

SQLTraceOff

Turns off the debug trace. Use this function to stop tracing function calls that are made to the database.

Syntax

SQLTraceOff()

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#), [SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#), [SQLRollBack](#), [SQLTraceOn](#)

Example

See [SQLFieldInfo](#)

See Also

[SQL Functions](#)

SQLTraceOn

Turns on a debug trace. Use this function to begin tracing function calls that are made to the database. The information is written to a log file.

Syntax

SQLTraceOn(*sFile*)

sFile:

The output file name for the debug trace.

Return Value

0 (zero) if successful, otherwise an [error](#) number is returned. (For details of the 307 error code, call the [SQLErrMsg](#) function).

Related Functions

[SQLBeginTran](#), [SQLCommit](#), [SQLConnect](#), [SQLDisconnect](#), [SQLEnd](#), [SQLErrMsg](#),
[SQLExec](#), [SQLFieldInfo](#), [SQLGetField](#), [SQLInfo](#), [SQLNext](#), [SQLNoFields](#), [SQLNumChange](#),
[SQLRollBack](#), [SQLTraceOff](#)

Example

See [SQLFieldInfo](#)

See Also

[SQL Functions](#)

Chapter: 50 String Functions

String functions allow you to manipulate strings in various ways. You can extract characters or substrings from a string, convert strings into other data types, format strings, search for strings, and perform other operations.

String Functions

Following are functions relating to Strings:

<u>CharToStr</u>	Converts an ASCII code into a string.
<u>HexToStr</u>	Converts a number into a hexadecimal string.
<u>IntToStr</u>	Converts an integer variable into a string.
<u>PathToStr</u>	Converts a CitectSCADA path into a string.
<u>RealToStr</u>	Converts a floating-point variable into a string.
<u>StrCalcWidth</u>	Retrieves the pixel width of a string using a particular font.
<u>StrClean</u>	Removes control characters from a string.
<u>StrFill</u>	Fills a string with characters.
<u>StrFormat</u>	Formats a variable into a string.
<u>StrGetChar</u>	Gets a single character from a string or buffer.
<u>StrLeft</u>	Gets the left-hand characters from a string.
<u>StrLength</u>	Gets the length of a string.
<u>StrLower</u>	Converts a string to lower-case.
<u>StrMid</u>	Gets characters from the middle of a string.
<u>StrPad</u>	Pads a string to the required length.

StrRight	Gets the right-hand characters from a string.
StrSearch	Searches for a string within a string.
StrSetChar	Sets a single character into string or buffer.
StrToChar	Converts a string to an ASCII code.
StrToDate	Converts a string into a date variable.
StrToFmt	Converts a string into format fields.
StrToGrp	Converts a string into a group.
StrToHex	Converts a hexadecimal string into an integer.
StrToInt	Converts a string into an integer variable.
StrToLines	Converts a string into lines of limited length.
StrTo-LocalText	Converts a string from Native language to Local language.
StrToPeriod	Converts a string into a (time) period.
StrToReal	Converts a string into a floating-point variable.
StrToTime	Converts a string into a time variable.
StrToValue	Converts a string into a floating-point variable.
StrTrim	Trims spaces from a string.
StrTruncFont	Returns the truncated string using a particular font (specified by name) or the specified number of characters.
StrTrunc-FontHnd	Returns the truncated string using a particular font (specified by font number) or the specified number of characters.
StrUpper	Converts a string to upper-case.
StrWord	Gets a word from a string.

See Also

[Functions Reference](#)

CharToStr

Converts an ASCII code into a string.

Syntax

CharToStr(ASCIICode)

ASCIICode:

The ASCII code to convert.

Return Value

A string containing the converted ASCII code.

Related Functions

[StrSetChar](#)

Example

```
str = CharToStr(65);  
! Sets str to "A".
```

See Also

[String Functions](#)

HexToStr

Converts a number into a hexadecimal string. The string is the width specified (padded with zeros).

Syntax

HexToStr(Number, Width)

Number:

The number to convert.

Width:

The width of the string.

Return Value

A string containing the converted number.

Related Functions

[StrToHex](#), [IntToStr](#)

Example

```
Variable=HexToStr(123, 4);  
! Sets Variable to "007b".  
Variable = HexToStr(0x12ABFE, 8);  
! Sets Variable to "0012abfe"
```

See Also

[String Functions](#)

IntToStr

Converts a number into a string.

Syntax

IntToStr(*Number*)

Number:

The number to convert.

Return Value

A string containing the converted number.

Related Functions

[StrFormat](#)

Example

```
Variable=IntToStr(5);  
! Sets Variable to "5".
```

See Also

[String Functions](#)

PathToStr

Converts a CitectSCADA path into a string. The path string can contain one of the standard CitectSCADA path operators, BIN, DATA, RUN, or a user-configured path. If the string does not contain a CitectSCADA path, it is unchanged.

Syntax

PathToStr(*sPath*)

sPath:

The CitectSCADA path to convert.

Return Value

A string containing the converted path.

Example

```
Variable=PathToStr("[data]:test.txt");
! Sets Variable to "c:\MyApplication\data\test.txt".
! assuming that DATA=C:\MyApplication\DATA
```

See Also

[String Functions](#)

RealToStr

Converts a floating-point number into a string.

Syntax

RealToStr(*Number, Width, Places*)

Number:

The floating-point number to convert.

Width:

The width of the string.

Places:

The number of decimal places contained in the string.

Return Value

The floating-point number (as a string).

Related Functions

[StrToReal](#)

Example

```
Variable=RealToStr(12.345,10,1);
! Sets Variable to "      12.3" (10 characters long).
```

See Also

[String Functions](#)

StrCalcWidth

Retrieves the pixel width of a string using a particular font.

Syntax

StrCalcWidth(*sText*, *iFont*)

sText:

The text to determine the pixel width of

iFont:

The font number used to calculate the pixel width of the text. (To use the default font, set to -1).

Return Value

The pixel width of a string using the particular font.

Related Functions

[StrTruncFont](#), [StrTruncFontHnd](#), [DspFont](#), [DspFontHnd](#)

See Also

[String Functions](#)

StrClean

Removes control characters from a string. Any character that is not a displayable ASCII character is removed from the string.

Syntax

StrClean(*String*)

String:

The source string.

Return Value

The string with all control characters removed.

Related Functions

[StrTrim](#)

Example

```
Variable=StrClean("*****Text*****");
/* Sets Variable to "Text" (the "*" character in this example
represents an unprintable character). */
```

See Also

[String Functions](#)

StrFill

Fills a string with a number of occurrences of another string.

Syntax

StrFill(*String*, *Length*)

String:

The string to be repeated.

Length:

The length of the string.

Return Value

The filled string.

Related Functions

[StrPad](#)

Example

```
Variable=StrFill("abc",10);
! Sets Variable to "abcabcabca".
```

```
Variable=StrFill("x",10);  
! Sets Variable to "xxxxxxxxxx".
```

See Also

[String Functions](#)

StrFormat

Converts a variable into a formatted string. This function is the equivalent of the Cicode " ##### " operator.

Syntax

StrFormat(*Variable*, *Width*, *DecPlaces*, *EngUnits*)

Variable:

The variable to format into a string.

Width:

The width of the variable after it has been converted to string format.

DecPlaces:

The number of decimal places in the converted string.

EngUnits:

The engineering units of the variable.

Return Value

The variable (as a formatted string).

Related Functions

[StrToReal](#), [StrToInt](#), [RealToStr](#), [IntToStr](#)

Example

```
Variable=StrFormat(10.345,5,2,"%");  
! Sets Variable to "10.35%".
```

See Also

[String Functions](#)

StrGetChar

Gets a single character from a string or buffer. Use this function to read a string, character by character.

Syntax

StrGetChar(*String*, *iOffset*)

String:

The source string.

iOffset:

The offset in the string, commencing at 0.

Return Value

The character at the offset in the string.

Related Functions

[StrSetChar](#)

Example

```
FOR i = 0 To length DO
    char = StrGetChar(str, i);
    ! Get char from string
END
```

See Also

[String Functions](#)

StrLeft

Gets the left-most characters from a string.

Syntax

StrLeft(*String*, *N*)

String:

The source string.

N:

The number of characters to get from the source string.

Return Value

A string containing the left-most *N* characters of *String*.

Related Functions

[StrRight](#), [StrMid](#), [StrLength](#)

Example

```
Variable=StrLeft ("ABCDEF", 2);  
! Sets Variable to "AB".
```

See Also

[String Functions](#)

StrLength

Gets the length of a string.

Syntax

StrLength(*String*)

String:

The source string.

Return Value

The length of the string (as an integer).

Related Functions

[StrRight](#), [StrMid](#), [StrLeft](#)

Example

```
Variable=StrLength ("ABCDEF");  
! Sets Variable to 6.
```

See Also

[String Functions](#)

StrLower

Converts a string to lowercase.

Syntax

StrLower(*String*)

String:

The source string.

Return Value

The string (as lowercase).

Related Functions

[StrUpper](#)

Example

```
Variable=StrLower("ABCDEF");
! Sets Variable to "abcdef".
Variable=StrLower("AbCdEf");
! Sets Variable to "abcdef".
```

See Also

[String Functions](#)

StrMid

Gets characters from the middle of a string.

Syntax

StrMid(*String, Offset, Characters*)

String:

The source string.

Offset:

The offset in the string, commencing at 0.

Characters:

The number of characters to get, commencing at the offset.

Return Value

A string containing the number of characters from the offset.

Related Functions

[StrLeft](#), [StrLength](#), [StrRight](#)

Example

```
Variable=StrMid("ABCDEF",1,3);
! Sets Variable to "BCD".
Variable=StrMid("ABCDEF",4,1);
! Sets Variable to "E".
```

See Also

[String Functions](#)

StrPad

Pads a string with a number of occurrences of another string. Padding can be added to the left or to the right of a string. If the string is already longer than the required string length, the string is truncated.

Syntax

StrPad(*String*, *PadString*, *Length*)

String:

The source string.

PadString:

The padding string.

Length:

The length of the string. If a positive length is specified, padding will be added to the right of the string. If a negative length is specified, padding will be added to the left of the string.

Return Value

A padded string.

Related Functions

[StrFill](#)

Example

```
Variable=StrPad("Test", " ",10);
! Sets Variable to "Test      ".
Variable=StrPad("Test","abc",-14);
! Sets Variable to "abcabcabcaTest".
```

See Also[String Functions](#)**StrRight**

Gets the rightmost characters from a string.

Syntax**StrRight(*String*, *N*)**

String:

The source string.

N:

The number of characters to get from the source string.

Return Value

A string containing the rightmost *N* characters of *String*.

Related Functions[StrLeft](#), [StrMid](#), [StrLength](#)**Example**

```
Variable=StrRight("ABCDEF",2);
! Sets Variable to "EF".
```

See Also[String Functions](#)**StrSearch**

Searches for a string within a string, commencing at a specified offset. The result of the search is the index in the source string, where the first character of the sub-string is found. Index 0 is the first character in the string, index 1 is the second, and so on.

Syntax

StrSearch(*Offset*, *String*, *Substring*)

Offset:

The offset in the string, commencing at 0.

String:

The source string.

Substring:

The substring to search for.

Return Value

The index in the search string, or -1 if the sub-string does not exist in the string.

Related Functions

[StrLength](#)

Example

```
Variable=StrSearch(1,"ABCDEF","CD");
! Sets Variable to 2.
Variable=StrSearch(4,"ABCDEF","CD");
! Sets Variable to -1.
Variable=StrSearch(5,"ABCDEF","F");
! Sets Variable to 5.
```

See Also

[String Functions](#)

StrSetChar

Sets a single character into a string or buffer. Use this function to build up a string, character by character, and terminate the string with the end-of-string character 0 (zero). (If you use a string without a terminator in a function that expects a string, or in a Cicode expression, you could get invalid results.) To use the string to build up a buffer, you do not need the terminating 0 (zero).

Syntax

StrSetChar(*sText*, *iOffset*, *Char*)

sText:

The destination string.

iOffset:

The offset in the string, commencing at 0.

Char:

The ASCII character to set into the string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[StrGetChar](#)

Example

```

! Set chars in buffer, Buf is NOT a valid string
! and cannot be used where a normal string would be used.
FOR i = 0 To length DO
    StrSetChar(Buf, i, 30 + i);
END
StrSetChar(sStr, 0, 13);           ! put CR into string
StrSetChar(sStr, 1, 0);           ! terminate so may be used as a normal string

```

See Also

[String Functions](#)

StrToChar

Gets the ASCII code of the first character in a string.

Syntax

StrToChar(*String*)

String:

The source string.

Return Value

The ASCII code of the first character in *String*.

Related Functions

[StrGetChar](#)

Example

```
Variable=StrToChar("ABC");
! Sets Variable to 65 (ASCII "A").
```

See Also

[String Functions](#)

StrToDate

Converts a "date" string into a time/date variable. This variable is the same as returned from the TimeCurrent() function. To set the order of the day, month, and year, and the delimiter, use the Windows Control panel.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with the following Cicode:

```
IF StrToDate(Arg1)>0 THEN
    ...
ELSE
    ...
END
```

Syntax

StrToDate(*String*)

String:

The source string.

Return Value

A time/date variable, or 274 if the time/date is out of range.

Related Functions

[StrToPeriod](#), [StrToTime](#)

Example

```
! Australian format (dd/mm/yy) is set in the Windows Control panel.
DateVariable=DateAdd(StrToDate("3/11/95"),86400);
NewDate= TimeToStr(DateVariable, 2);
! Adds 24 hours to 3/11/95 and sets NewDate to "4/11/95".
```

See Also[String Functions](#)**StrToFmt**

Converts a string into field data for a format template. This function is useful for splitting a string into separate strings. After the string is converted, you can call the FmtGetField() function to extract the individual data from the template fields.

Syntax**StrToFmt(*hFmt*, *String*)***hFmt*:

The format template handle, returned from the FmtOpen() function. The handle identifies the table where all data on the associated format template is stored.

String:

The source string.

Return Value

The string (formatted as template field data).

Related Functions[FmtOpen](#), [FmtGetField](#)**Example**

```
StrToFmt (hFmt,"CV101 Raw Coal Conveyor");
Name=FmtGetField(hFmt,"Name");
! Sets Name to "CV101".
```

See Also[String Functions](#)**StrToGrp**

Converts a string into a group and places it into a group number. Any existing values in the group are cleared before the new values are inserted. The group string is a series of numbers separated by ", " to list individual values or " .. " to specify a range of values.

Syntax**StrToGrp(*hGrp*, *Str*)**

hGrp:

The group handle, returned from the GrpOpen() function. The group handle identifies the table where all data on the associated group is stored.

Str:

The string to convert.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[GrpOpen](#)

Example

```
hGrp=GrpOpen ("MyGrp",1);
! Set group to 1 ... 10 and 20, 30 and 40.
StrToGrp (hGrp,"1..10,20,30,40");
```

See Also

[String Functions](#)

StrToHex

Converts a hexadecimal string into an integer. This function will search the string for the first non-blank character, and then start converting until it finds the end of the string or a non-hexadecimal numeric character. If the first non-blank character is not one of the following hexadecimal characters, the return value is 0 (zero):

- (0-9, a-f, A-F);
- A space;
- A "+" (plus) or a "-" (minus) sign.

Syntax

StrToHex(*String*)

String:

The string to convert.

Return Value

An integer (converted from *String*).

Related Functions[StrToReal](#), [StrToValue](#), [HexToStr](#)**Example**

```
Variable=StrToHex("123");
! Sets Variable to hex 123, decimal 291
Variable=StrToHex("F2");
! Sets Variable to hex F2, decimal 242
Variable=StrToHex("G45");
! Sets Variable to 0.
Variable=StrToHex("-FG");
! Sets Variable to hex, -F decimal -15.
```

See Also[String Functions](#)**StrToInt**

Converts a string into an integer. This function will search the string for the first non-blank character, and then start converting until it finds the end of the string or a non-numeric character. If the first non-blank character is not a numeric character (0-9), a space, a "+" or a "-" sign, the return value is 0 (zero).

Syntax**StrToInt(*String*)***String*:

The string to convert.

Return Value

An integer (converted from *String*).

Related Functions[StrToReal](#), [StrToValue](#)**Example**

```
Variable=StrToInt("45");
! Sets Variable to 45.
Variable=StrToInt("45.23");
! Sets Variable to 45.
```

```
Variable=StrToInt("A45");
! Sets Variable to 0.
```

See Also

[String Functions](#)

StrToLines

Converts a string into separate lines that contain no more than the number of characters specified in the MaxChars argument.

The function splits the string by inserting newline characters into the text string, thus dividing it into separate lines. The string will be split at a whitespace character if possible, and that whitespace will be replaced by the newline character. If no whitespace characters are available then the insertion will be made at the maximum number of characters from the previous line break.

Syntax

StrToLines(*String*,*MaxChars*, *nLines*)

String:

The string to convert.

MaxChars:

The maximum number of characters permitted in each new line produced by the StrToLines() function.

nLines:

The number of lines produced by the StrToLines() function from the input string.

Return Value

An integer (*nLines*) containing the number of lines produced by the StrToLines() function from the input string.

Example

```
BrokenString=StrToLines("Was that a real Stegosaur?", 5, nLines);
!The function returns the value 6 in nLines, and Broken String now contains:
Was
that
a
real
Stego
saur?
```

```

BrokenString=StrToLines("It breaks the string by inserting newline characters into
the text.", 16, nLines);
!The function returns the value 6 in nLines, and Broken String now contains:
It breaks the
string by
inserting
newline
characters into
the text.

```

See Also[String Functions](#)**StrToLocalText**

Converts a native string into the local version of that string. (The string needs to be contained within quotation marks, as shown in the example below.) The local version is taken from the current language database(as specified using the [Language]LocalLanguage parameter).

StrToLocalText(*sText*)

sText:

The string for which you would like the local translation returned. This string needs to be enclosed in quotation marks. For example:

"@ (Motor Overload) "

Return Value

The local version of the text if it was found, otherwise the native text or "#MESS" is returned, depending on the setting of the [Language]DisplayError parameter.

Related Functions[LanguageFileTranslate](#), [SetLanguage](#)**Example**

```

StrToLocalText("@ (Motor Overload) ");
! Returns the Local translation of Motor Overload.

```

See Also[String Functions](#)**StrToPeriod**

Converts a string into a time period. You would normally use this function to convert operator input, for example, to set a trend period.

A valid period string is in the format HH:MM:SS, MM:SS or SS, where HH is the hours, MM is the minutes and SS is the seconds. The colon character (':') represents the time delimiter for these fields, which will be the current system time delimiter as set in the Windows Control Panel.

If minutes are specified, seconds need to be in the range 0-59. If hours are specified, minutes need to be in the range 0-59.

Syntax

StrToPeriod(*String*)

String:

The string to convert.

Return Value

A period (converted from *String*), or -1 if no conversion can be performed.

Related Functions

[StrToTime](#), [StrToDate](#)

Example

```
Variable=StrToPeriod("200");
! Sets Variable to 200 (seconds).
Variable=StrToPeriod("200:40");
! Sets Variable to 12040 (12000 + 40 seconds).
Variable=StrToPeriod("48:00:40");
! Sets Variable to 172840 (172800 + 40 seconds).
```

See Also

[String Functions](#)

StrToReal

Converts a string into a floating-point number. This function will search the string for the first non-blank character, and then start converting until it finds the end of the string or a non-numeric character. If the first non-blank character is not a numeric character (0-9), a space, a decimal point, a "+" or a "-" sign, the return value is 0 (zero).

Syntax

StrToReal(*String*)

String:

The string to convert.

Return Value

A floating-point number (converted from *String*).

Related Functions

[StrToInt](#), [StrToValue](#)

Example

```
Variable=StrToReal("45");
! Sets Variable to 45.
Variable=StrToReal("45.23");
! Sets Variable to 45.23
Variable=StrToReal("A45");
! Sets Variable to 0.
```

See Also

[String Functions](#)

StrToTime

Converts a "time" string into a time/date variable. The value returned is the number of seconds from midnight. You can add this value to the date to get the current time value. To set the time delimiter, use the Windows Control Panel.

A valid time string is in the format HH:MM:SS or HH:MM:SS tt, where HH is the hour in the range 0-23, MM is the minute in the range 0-59, SS is the second in the range 0-59 and tt is the time extension; for example,, am or pm. The colon character ':' represents the time delimiter for these fields, which will be the current system time delimiter as set in the Windows control panel.

Times may also be passed in the for HH or HH:MM. In other words, you may omit the right-hand fields if they are 0.

Syntax

StrToTime(*String*)

String:

The string to convert.

Return Value

A time/date variable, or -1 if no conversion can be performed.

Related Functions

[Time](#), [Date](#)

Example

```
Variable=StrToTime("11:43:00");
! Sets Variable to (11*3600+43*60+0) seconds.
Variable=StrToTime("9:02");
! Sets Variable to (9*3600+2*60) seconds.
Variable=StrToTime("2");
! Sets Variable to (2*3600) seconds.
```

See Also

[String Functions](#)

StrToValue

Converts a string into a floating-point number. This function is similar to the StrToReal() function except that the function halts if it is passed an empty or invalid string. The function will search the string for the first non-blank character, and then start converting until it finds the end of the string or a non-numeric character. If the first non-blank character is not a numeric character (0-9), a space, a decimal point, a "+" or a "-" sign, the function will halt.

Use this function to check keyboard input from the operator by setting control points (for example, it minimizes the likelihood of a setpoint being set to 0 if the operator presses ENTER or enters invalid data by mistake).

Syntax

StrToValue(*String*)

String:

The string to convert.

Return Value

A floating-point number (converted from *String*).

Related Functions[StrToReal](#), [StrToInt](#)**Example****System Keyboard**

Key Sequence	F3 ##### Enter
Command	SP123 = StrToValue(Arg1);
Comment	Set setpoint 123 to value unless value is invalid

Note: The Cicode is halted. Any other Cicode after the StrToValue() function will not execute.

See Also[String Functions](#)**StrTrim**

Removes leading and trailing spaces from a string. Internal spaces are not removed from the string.

Syntax**StrTrim**(*String*)*String*:

The source string.

Return Value

String with leading and trailing spaces removed.

Related Functions[StrPad](#), [StrFill](#)**Example**

Variable=StrTrim("Test String");

```
! Sets Variable to "Test String".
```

See Also

[String Functions](#)

StrTruncFont

Returns the truncated string using a particular font (specified by name) or the specified number of characters.

Syntax

StrTruncFont(*sText*, *sFont* [, *iLength*] [, *iLengthMode*])

sText:

The text to truncate

sFont:

The name of the font that is used to display the text. The Font Name needs to be defined in the Fonts database. If the font is not found, the default font is used.

iLength:

Length of the Text to display, either in characters or pixels depending on *iLengthMode* (default -1, no truncation)

iLengthMode:

The length mode of the text string:

- 0 - Length as pixels truncated (default)
- 1 - Length as pixels truncated with ellipsis
- 2 - Length interpreted as characters.

Return Value

A truncated string or the original one.

Related Functions

[StrCalcWidth](#), [StrTruncFontHnd](#)

See Also

[String Functions](#)

StrTruncFontHnd

Returns the truncated string using a particular font (specified by font number) or the specified number of characters.

Syntax

StrTruncFontHnd(*sText*, *hFont* [, *iLength*] [, *iLengthMode*])

sText:

The text to truncate

hFont:

The font handle used to calculate the pixel width of the text. (To use the default font, set to -1).

iLength:

Length of the Text to display, either in characters or pixels depending on *iLengthMode* (default -1, no truncation)

iLengthMode:

The length mode of the text string:

- 0 - Length as pixels truncated (default)
- 1 - Length as pixels truncated with ellipsis
- 2 - Length interpreted as characters.

Return Value

A truncated string or the original one.

Related Functions

[StrCalcWidth](#), [StrTruncFont](#), [DspFont](#), [DspFontHnd](#)

See Also

[String Functions](#)

StrUpper

Converts a string to uppercase.

Syntax

StrUpper(*String*)

String:

The source string.

Return Value

The string (as uppercase).

Related Functions

[StrLower](#)

Example

```
Variable=StrUpper("abcdef");
! Sets Variable to "ABCDEF".
Variable=StrUpper("AbCdEf");
! Sets Variable to "ABCDEF".
```

See Also

[String Functions](#)

StrWord

Gets the first word from a string. The word is removed from the string to allow the function to be repeated. Word separators can be a space, newline, carriage return, or tab character.

Syntax

StrWord(*String*)

String:

The source string.

Return Value

The first word from *String* (as a string).

Related Functions

[StrSearch](#)

Example

```
Str="THIS IS A STRING";
Variable=StrWord(Str);
! Sets Variable to "THIS".
Variable=StrWord(Str);
```

```
! Sets Variable to "IS".
Variable=StrWord(Str);
! Sets Variable to "A".
Variable=StrWord(Str);
! Sets Variable to "STRING".
```

See Also

[String Functions](#)

Chapter: 51 Super Genie Functions

The Super Genie functions allow you to use SuperGenies in your runtime system.

Super Genie Functions

Following are functions relating to Super Genies:

<u>Ass</u>	Associates a variable tag with a Super Genie.
<u>AssChain</u>	Chains the associations from the current Super Genie to a new Super Genie.
<u>AssMetadata</u>	Performs Super Genie associations using the "Name" and "Value" fields.
<u>AssMetadataPage</u>	Uses the metadata information from the current animation point for the page associations for a new Super Genie page, and displays the new Super Genie in the current page.
<u>AssMetadataPopup</u>	Uses the metadata information from the current animation point for the associations for a new Super Genie page, and displays the new Super Genie in a new pop up window.
<u>AssMetadataWin</u>	Uses the metadata information from the current animation point for the associations for a new Super Genie page, and displays the new Super Genie in a new window.
<u>AssChainPage</u>	Chains the associations from the current Super Genie to a new Super Genie, and displays the new Super Genie (in the current window).
<u>AssChainPopUp</u>	Chains the associations from the current Super Genie to a new Super Genie, and displays the new Super Genie in a new popup window.
<u>AssChainWin</u>	Chains the associations from the current Super Genie to a new Super Genie, and displays the new Super Genie in a new window. The new window will be of the same type as the current window.
<u>AssChainWinFree</u>	Stores the tag associations on an existing Super Genie, closes it, then assigns the tags to a new window.
<u>AssGetProperty</u>	Gets association information about the current Super Genie from the datasource

<u>AssGetScale</u>	Gets scale information about the associations of the current Super Genie from the datasource
<u>AssInfo</u>	Gets association information about the current Super Genie (that is information about a variable tag that has been substituted into the Super Genie).
<u>AssInfoEx</u>	Gets association information about the current Super Genie (that is information about a variable tag that has been substituted into the Super Genie). Replacement for AssInfo supporting online changes.
<u>AssPage</u>	Associates up to eight variable tags with a Super Genie and displays the Super Genie in the current window.
<u>AssPopUp</u>	Associates up to eight variable tags with a Super Genie and displays the Super Genie in a popup window.
<u>AssScaleStr</u>	Gets scale information about the associations of the current Super Genie (that is scale information about a variable tag that has been substituted into the Super Genie).
<u>AssTag</u>	Associates a variable tag with the current Super Genie. The association will be created for the current Super Genie only, and will only come into effect after you re-display the Super Genie.
<u>AssTitle</u>	Sets the runtime window title to the tag name of the first variable substituted into the Super Genie.
<u>AssVarTags</u>	Associates up to eight variable tags with a Super Genie. This association is only made for the next Super Genie you display (either in the current window or in a new window). You can use this function repeatedly to associate more than 8 variable tags to a Super Genie.
<u>AssWin</u>	Associates up to eight variable tags with a Super Genie, and displays the Super Genie in a new window.

See Also

[Functions Reference](#)

Ass

Associates a variable tag with a Super Genie. This association is only made for the next Super Genie you display (either in the current window or in a new window). You cannot create an association for a Super Genie that is already displayed. You need to call this function once for every Super Genie substitution string in the Super Genie, otherwise the variable (substitution string) will remain uninitialized and it will be displayed as #ASS.

This function provides the lowest level of support for associating Super Genie variables with physical tags. The higher level functions (listed below) are simpler to use.

Syntax

Ass(*hWin*, *nArg*, *sTag*, *nMode* [, *ClusterName*])

hWin:

The association will be created for the next Super Genie to display in the window specified here; enter the window number or:

- -3 - for the current window when the page is changed. The page can be changed by using the Page Cicode functions like [PageDisplay](#), [PageGoto](#), etc.
- -2 - for the next new window or page displayed.

nArg:

The argument number or name (substitution string number or name) of the Super Genie string to be replaced by *sTag*. For example, to replace ?INT 3? with *sTag*, set *nArg* to 3, or ?Level? set *nArg* to Level.

sTag:

The variable tag that will replace the Super Genie substitution string. The tag needs to be the same data type as that specified by the Super Genie substitution string. For example, only a digital tag could replace the substitution string ?DIGITAL 4?. Any other type of tag may raise a hardware alarm or display #err. If the substitution string does not specify a type (for example, ?5?), you can use any type except STRING.

The name of the tag can be prefixed by the name of the cluster for example, "*ClusterName.Tag*".

nMode:

The mode of the association. Set to 0.

ClusterName:

Specifies the name of the cluster in which the Variable Tag resides. This is optional if you have one cluster or are resolving the tag via the page's current cluster context. The argument is enclosed in quotation marks "".

Resolution of the tag's cluster context occurs when the page is displayed. It is resolved to the page's cluster context, not the context in force when this function is called.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
//Using a string identifier for the substitution parameter  
Ass(-2,"Level", "MILK_LEVEL",0);
```

```
// Associate variable tag PV123 with the next Genie to display in  
// the current window  
Ass(-3, 5, "PV123", 0);
```

See Also

[Super Genie Functions](#)

AssChain

Chains the associations from the current Super Genie to a new Super Genie. Use this function to display a new Super Genie when you already have one displayed. The new Super Genie will inherit the associations of the first Super Genie.

This function provides the lowest level of support for chaining associations from one Super Genie to another. You should call the higher level functions [AssChainPage\(\)](#), [AssChainWin\(\)](#), and [AssChainPopUp\(\)](#) - these functions are simpler to use.

Syntax

AssChain(*hDest*, *hSource*, *nMode*)

hDest:

The next Super Genie to display in the window specified here will inherit the associations of the current Super Genie - enter the window number, or:

- -3: for the current window when the page is changed. The page can be changed by using the Page Cicode functions like [PageDisplay](#), [PageGoto](#), etc.
- -2: for the next new window or page displayed.

hSource:

The number of the window containing the source Super Genie (that is the Super Genie from which the associations will be inherited).

nMode:

The mode of the association. Set to 0.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Copy associations from the current Super Genie to !NewGenie
AssChain(WinNumber(), WinNumber(), 0);
PageDisplay("!NewGenie");
```

See Also

[Super Genie Functions](#)

AssMetadata

This non-blocking function performs Super Genie associations using the "Name" and "Value" fields defined on the Object Properties - Metadata tab, and matches it to the 'Name' field in the page associations table. While performing associations any additional metadata entries are ignored.

Syntax

AssMetadata(*hWin* [, *nAn*])

hWin:

The associations will be created for the next Super Genie to display in the window specified. Enter the window number or

- -3: for the current window when the page is changed. The page can be changed by using the Page Cicode functions like [PageDisplay](#), [PageGoto](#), etc.
- -2: for the next new window or page displayed.

nAN:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. This parameter is optional with -2 being the default value. When -2 is specified, it is the same as using the function KeyGetCursor() which returns the animation number of the current active command cursor. Refer to [KeyGetCursor\(\)](#) for usage and behavior.

WARNING

UNINTENDED EQUIPMENT OPERATION

If called after other cicode functions in a command expression field, retrieve the animation number first, then pass it through the *nAN* parameter.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Example

```
/* Example of calling AssMetadata after other cicode functions */
An = KeyGetCursor();
SomeVal = TagRead("SomeTag"); // do additional work
AssMetadata(-2, An);
PageOpen("!TestSG");
```

Related Functions

[Ass](#), [AssChain](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

See Also

[Super Genie Functions](#)

AssMetadataPage

Uses the metadata information from the current object for the page associations for a new Super Genie page, and displays the new Super Genie (in the current page).

Syntax

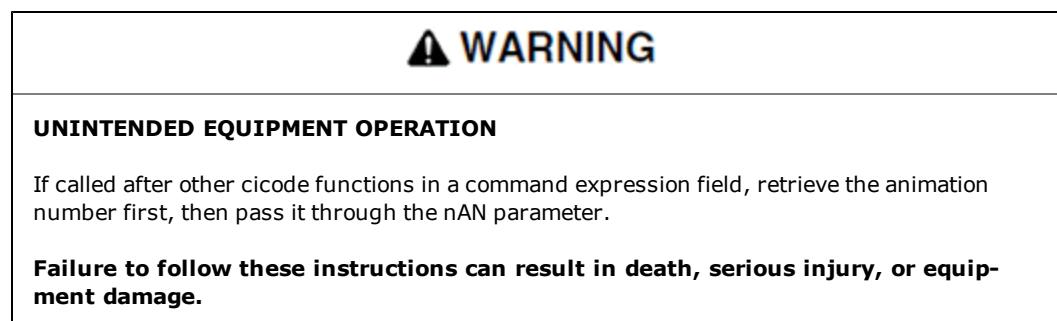
AssMetadataPage(*sPage*, *nAN*)

sPage:

The name of the Super Genie page to open.

nAN:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. This parameter is optional with -2 being the default value. When -2 is specified, it is the same as using the function KeyGetCursor() which returns the animation number of the current active command cursor. Refer to [KeyGetCursor\(\)](#) for usage and behavior.



Return Value

0 (zero) if successful, error code if unsuccessful.

Example

```
/* Example of calling AssMetadataPage after other cicode functions */
An = KeyGetCursor();
SomeVal = TagRead("SomeTag"); // do additional work
AssMetadataPage("!TestSG", An);
```

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

See Also

[Super Genie Functions](#)

AssMetadataPopUp

Uses the metadata information from the current animation point for the associations for a new Super Genie page, and displays the new Super Genie in a pop up window.

Syntax

AssMetadataPopUp(sPage)

sPage

The name of the Super Genie page to open.

nAN:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. This parameter is optional with -2 being the default value. When -2 is specified, it is the same as using the function KeyGetCursor() which returns the animation number of the current active command cursor. Refer to [KeyGetCursor\(\)](#) for usage and behavior.

WARNING

UNINTENDED EQUIPMENT OPERATION

If called after other cicode functions in a command expression field, retrieve the animation number first, then pass it through the nAN parameter.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Example

```
/* Example of calling AssMetadataPopup after other cicode functions */
An = KeyGetCursor();
SomeVal = TagRead("SomeTag"); // do additional work
AssMetadataPopup("!TestSG", An);
```

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

See Also

[Super Genie Functions](#)

AssMetadataWin

Uses the metadata information from the current animation-point for the associations for a new Super Genie page, and displays the new Super Genie in a new window.

Syntax

AssMetadataWin(*sPage*, INT *x*, INT *y*, INT *mode*)

sPage:

The name of the Super Genie page to open.

X:

The x pixel coordinate of the top left corner of the window. Default value is 0.

Y:

The y pixel coordinate of the top left corner of the window. Default value is 0.

Mode:

The mode of the window:

0 - Normal page (default value).

1 - Page child window. The window is closed when a new page is displayed, for example, when the PageDisplay() or PageGoto() function is called. The parent is the current active window.

2 - Window child window. The window is closed automatically when the parent window is freed with the WinFree() function. The parent is the current active window.

4 - No re-size. The window is displayed with thin borders and no maximize/minimize icons. The window cannot be re-sized.

8 - No icons. The window is displayed with thin borders and no maximize/minimize or system menu icons. The window cannot be re-sized.

16 - No caption. The window is displayed with thin borders, no caption, and no maximize/minimize or system menu icons. The window cannot be re-sized.

32 - Echo enabled. When enabled, keyboard echo, prompts, and error messages are displayed on the parent window. This mode should only be used with child windows (for example, Mode 1 and 2).

64 - Always on top.

128 - Open a unique window. This mode stops this window from being opened more than once.

256 - Display the entire window. This mode commands that no parts of the window will appear off the screen

512 - Open a unique Super Genie. This mode stops a Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

- 1024 - Disables dynamic resizing of the new window, overriding the setting of the [Page]DynamicSizing parameter.
- 4096 - Allows the window to be resized without maintaining the current aspect ratio. The aspect ratio defines the relationship between the width and the height of the window, which means this setting allows you to stretch or compress the window to any proportions. This option overrides the setting of the [Page]MaintainAspectRatio parameter.
- 8192 - Text on a page will be resized in proportion with the maximum scale change for a resized window. For example, consider a page that is resized to three times the original width, and half the original height. If this mode is set, the font size of the text on the page will be tripled (in proportion with the maximum scale). This option overrides the setting of the [Page] ScaleTextToMax parameter.
- 16384 - Hide the horizontal scroll bar.
- 32768 - Hide the vertical scroll bar.
- 65536 - Disable horizontal scrolling.
- 131072 - Disable vertical scrolling.

You can select multiple modes by adding modes together (for example, set Mode to 9 to open a page child window without maximize, minimize, or system menu icons).

nAN:

An animation number that uniquely identifies an object. This object contains the list of metadata definitions that will be used to perform the association operations. This parameter is optional with -2 being the default value. When -2 is specified, it is the same as using the function KeyGetCursor() which returns the animation number of the current active command cursor. Refer to [KeyGetCursor\(\)](#) for usage and behavior.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

If called after other cicode functions in a command expression field, retrieve the animation number first, then pass it through the nAN parameter.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Return Value

0 (zero) if successful, error code if unsuccessful.

Example

```
/* Example of calling AssMetadataWin after other cicode functions */
```

```
An = KeyGetCursor();
SomeVal = TagRead("SomeTag"); // do additional work
AssMetadataWin("!TestSG", 50, 50, 1, An);
```

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

See Also

[Super Genie Functions](#)

AssChainPage

Chains the associations from the current Super Genie to a new Super Genie, and displays the new Super Genie (in the current window). Use this function to display a new Super Genie when you already have one displayed. The new Super Genie will inherit the associations of the first Super Genie.

Syntax

AssChainPage(*sPage*)

sPage:

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Display new Super Genie in current window, using current associations
AssChainPage("!NewGenie");
```

See Also

[Super Genie Functions](#)

AssChainPopUp

Chains the associations from the current Super Genie to a new Super Genie, and displays the new Super Genie in a new popup window. Use this function to display a new Super Genie in a new popup window when a Super Genie is already displayed. The new Super Genie will inherit the associations of the first.

Note: This function helps to prevent the Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

Syntax

AssChainPopUp(*sPage*)

sPage

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Display new Super Genie in new popup using current associations  
AssChainPopUp ("!NewGenie");
```

See Also

[Super Genie Functions](#)

AssChainWin

Chains the associations from the current Super Genie to a new Super Genie, and displays the new Super Genie in a new window. The new window will be of the same type as the current window. Use this function to display a new Super Genie in a new window when a Super Genie is already displayed. The new Super Genie will inherit the associations of the first.

Syntax

AssChainWin(*sPage*, *X*, *Y*, *Mode*)

sPage:

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

- *X* - The x pixel coordinate of the top left corner of the window.
- *Y* - The y pixel coordinate of the top left corner of the window.

Mode:

The mode of the window:

0 - Normal page.

1 - Page child window. The window is closed when a new page is displayed, for example, when the PageDisplay() or PageGoto() function is called.
The parent is the current active window.

2 - Window child window. The window is closed automatically when the parent window is freed with the WinFree() function. The parent is the current active window.

4 - No re-size. The window is displayed with thin borders and no maximize/minimize icons. The window cannot be re-sized.

8 - No icons. The window is displayed with thin borders and no maximize/minimize or system menu icons. The window cannot be re-sized.

16 - No caption. The window is displayed with thin borders, no caption, and no maximize/minimize or system menu icons. The window cannot be re-sized.

32 - Echo enabled. When enabled, keyboard echo, prompts, and error messages are displayed on the parent window. This mode should only be used with child windows (for example, Mode 1 and 2).

64 - Always on top.

128 - Open a unique window. This mode helps to prevent this window from being opened more than once.

256 - Display the entire window. This mode helps to ensure that no parts of the window will appear off the screen

512 - Open a unique Super Genie. This mode helps to prevent a Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

1024 - Disables dynamic resizing of the new window, overriding the setting of the [Page]DynamicSizing parameter.

You can select multiple modes by adding modes together (for example, set *Mode* to 9 to open a page child window without maximize, minimize, or system menu icons).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Displays a new super genie in a new window using the current associations
AssChainWin("!NewGenie", 100, 200, 1 + 8);
```

See Also

[Super Genie Functions](#)

AssChainWinFree

Stores the tag associations on an existing Super Genie, closes it, then assigns the tags to a new window. This allows a Super Genie popup window to call another popup window, and close the parent popup.

This function is effectively the same as the AssChainWin() function, but frees the current Super Genie.

Syntax

AssChainWinFree(*sPage*, *X*, *Y*, *Mode*)

sPage:

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

X - the x pixel coordinate of the top-left corner of the window.

Y - the y pixel coordinate of the top-left corner of the window.

Mode:

The mode of the window:

0 - Normal page.

- 1 - Page child window. The window is closed when a new page is displayed, for example, when the PageDisplay() or PageGoto() function is called. The parent is the current active window.
- 2 - Window child window. The window is closed automatically when the parent window is freed with the WinFree() function. The parent is the current active window.
- 4 - No re-size. The window is displayed with thin borders and no maximize/minimize icons. The window cannot be re-sized.
- 8 - No icons. The window is displayed with thin borders and no maximize/minimize or system menu icons. The window cannot be re-sized.
- 16 - No caption. The window is displayed with thin borders, no caption, and no maximize/minimize or system menu icons. The window cannot be re-sized.
- 32 - Echo enabled. When enabled, keyboard echo, prompts, and error messages are displayed on the parent window. This mode should only be used with child windows (for example, Mode 1 and 2).
- 64 - Always on top.
- 128 - Open a unique window. This mode helps to prevent this window from being opened more than once.
- 256 - Display the entire window. This mode helps to ensure that no parts of the window will appear off the screen
- 512 - Open a unique Super Genie. This mode helps to prevent a Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.
- 1024 - Disables dynamic resizing of the new window, overriding the setting of the [Page]DynamicSizing parameter.

You can select multiple modes by adding modes together (for example, set *Mode* to 9 to open a page child window without maximize, minimize, or system menu icons).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Close the current genie window and display a new genie using the current associations  
AssChainWinFree("!GeniePopup", 200, 300, 1 + 8);
```

See Also

[Super Genie Functions](#)

AssGetProperty

This function gets association information about the current Super Genie from the data-source (that is information about a variable tag that has been substituted into the Super Genie). You can only call this function on a Super Genie after the associations are completed.

Use this function to display association information as part of the Super Genie. For example, if you have a Super Genie that is a loop controller, you could display the name of the loop at the top of the loop controller box. Each time the Super Genie is used with different associations (specifically a different tag name association) the correct loop name will be displayed.

If a constant value is associated, then only the constant value can be retrieved through the *TagName* property. The remaining properties are not valid.

This function replaces [AssInfo](#).

Syntax

AssGetProperty(*iArg*, *sProperty* [, *iCached*])

iArg:

The argument number or name of the association from which to get information.

sProperty:

The property to read. Property names are case sensitive. Supported properties are:

Address - The configured address of the associated tag (as specified in the variable tags form)

ArraySize - Array size of the associated tag. Returns 1 for non-array types

AssFullName - Full name of the association tag in the form *cluster.tagname* even if the tag is not resolved

ClusterName - Name of the cluster the associated tag resides on.

DataBitWidth - Number of bits used to store the value

Description - Description of the associated tag

EngUnitsHigh - Maximum scaled value

EngUnitsLow - Minimum scaled value

Format - Format bit string. The format information is stored in the integer as follows:

- Bits 0-7 - format width
- Bits 8-15 - number of decimal places
- Bits 16 - zero-padded
- Bit 17 - left-justified
- Bit 18 - display engineering units
- Bit 20 - exponential (scientific) notation

ErrorValUsed - Returns 1 if the defined error value was used for the SuperGenie association. This means that tag name is invalid/unresolved or the substitutions are not complete. This is only relevant for named SuperGenies. 0 is returned if the association string provided a value, or a default value was not defined.

FormatDecPlaces - Number of decimal places for default format

FormatWidth - Number of characters used in default format

FullName - Full name of the association tag in the form *cluster.tagname*. If the association tag is not resolved, returns an empty string.

Literal - Returns 1 if the substitution is a literal value, returns 0 if the substitution is a tag name.

RangeHigh - Maximum unscaled value

RangeLow - Minimum unscaled value

TagName - Name of the tag for the specified association

Type - Raw type of associated tag

Units - Engineering Units for example, %, mm, Volts

iCached:

Optional flag to attempt to retrieve the cached value for the property rather than the current value. This makes the function non-blocking. If the property has not yet been cached, an error is set.

0 - Do not force cached read. Cicode is blocking

1 - Force cached read. Cicode is non-blocking

Default value is 1 (true)

Return Value

String representation of the property of the association. On detection of an error, an empty string and an [error](#) are set.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#), [TagGetProperty](#), [TagGetScale](#), [TagScaleStr](#), [TagInfo](#)

Example

```
//Using a string identifier for the substitution parameter  
AssGetProperty("MILK_LEVEL", "TagName", 0);
```

```
// Get the engineering full scale value for the 2nd  
// argument of the association of the current Super Genie  
EngFullScale = AssGetProperty(2, "EngUnitsHigh", 0);  
// Get the cached engineering units for the 3rd argument  
// of the association of the current Super Genie  
MeasureUnits = AssGetProperty(3, "Units", 1);
```

See Also

[Super Genie Functions](#)

AssGetScale

Gets scale information about the associations of the current Super Genie from the data-source (that is scale information about a variable tag that has been substituted into the Super Genie). You can only call this function on a Super Genie after the associations are completed.

Use this function to display association scale information as part of the Super Genie. For example, if you have a bar graph illustrating output, you could indicate zero, 50%, and full scale output on the vertical axis of the graph. Each time the Super Genie is used with different associations the correct scale values will be displayed.

The value is returned as a formatted string using the association format specification and (optionally) the engineering units.

This function replaces [AssScaleStr](#).

Syntax

AssGetScale(*iArg*, *iPercent*, *iEngUnits* [, *iCached*])

iArg:

The argument number or name of the association from which to get information.

iPercent:

The percentage of full scale of the returned value.

iEngUnits:

Flag to determine if the value is returned with engineering units:

- 0 - Return the value without engineering units
- 1 - Return the value with engineering units

iCached:

Optional flag to attempt to retrieve the cached value for the property rather than the current value. This makes the function non-blocking. If the property has not yet been cached, an error is set.

- 0 - Do not force cached read. Cicode is blocking
- 1 - Force cached read. Cicode is non-blocking

Default value is 1 (true).

Return Value

The scale of the association (as a string).

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#), [TagGetProperty](#), [TagGetScale](#), [TagScaleStr](#), [TagInfo](#)

Example

```
//Using a string identifier for the substitution parameter
AssGetScale("MILK_LEVEL", 50, 1, 0);
```

```
// Display the zero, 50% and full scale of the 2nd argument
// of the association of the current Super Genie
DspText(31,0,AssGetScale(2, 0, 1));
DspText(32,0,AssGetScale(2, 50, 1));
DspText(33,0,AssGetScale(2, 100, 1));
```

See Also

[Super Genie Functions](#)

AssInfo

Gets association information about the current Super Genie (that is information about a variable tag that has been substituted into the Super Genie). You can only call this function on a Super Genie after the associations are completed.

Use this function to display association information as part of the Super Genie. For example, if you have a Super Genie that is a loop controller, you could display the name of the loop at the top of the loop controller box. Each time the Super Genie is used with different associations (specifically a different tag name association) the correct loop name will be displayed.

Note: This function is being deprecated and is replaced by the [AssGetProperty](#) function. If the Tag properties are updated, AssInfo does not get the updated values whereas AssGetProperty does. In addition, the function [AssInfoEx](#) has been introduced to make it easier to make legacy Cicode compatible with online changes. In a large number of cases AssInfo can be replaced with AssInfoEx using Find and Replace (see Using Find and Replace in a project). Please be aware that if you are replacing an instance of AssInfo with AssInfoEx in a loop, you may want to make AssInfoEx blocking using the iCached argument to verify you are using the correct value for the Tag in your logic.

Syntax

AssInfo(*nArg*, *nType*)

nArg:

When you associate variable tags with super Genies, the Super Genie substitution strings are replaced by variable tags. The *nArg* argument allows you to get information about one of those variable tags. What you need to know is which substitution string it replaced when the association was performed.

Enter the argument number or name (substitution string number or name) of the relevant substitution string. For example, if you want information about the variable that replaced substitution string

?INT 3?

set *nArg* to 3.

Or

?Level?

set *nArg* to Level

nType:

The type of information to get:

0 - The Tag name of the association. If the association tag is not resolved, returns an empty string.

1 - Engineering units

2 - Raw zero scale

3 - Raw full scale

4 - Engineering zero scale

5 - Engineering full scale

6 - Width of the format

7 - Number of decimal places of format

8 - The Tag format as a long integer. The format information is stored in the integer as follows:

- Bits 0-7 - format width
- Bits 8-15 - number of decimal places
- Bits 16 - zero-padded
- Bit 17 - left-justified
- Bit 18 - display engineering units
- Bit 20 - exponential (scientific) notation

9 - Logical Unit Number - I/O device number (for internal use)

10 - Raw Type - Protocol's raw data type number for this tag. Type numbers are:

- 0 - Digital
- 1 - Integer
- 2 - Real
- 3 - BCD
- 4 - Long
- 5 - Long BCD
- 6 - Long Real
- 7 - String
- 8 - Byte
- 9 - Void
- 10 - Unsigned integer

11 - Bit Width - Tag's size in bits. For example, an INT is 16 bits

12 - Unit Type - Protocol's unit type number for this tag

13 - Unit Address - Tag's address after the protocol DBF's template is applied.

14 - Unit Count - Array size. For example, if the tag's address is I1[50], the unit count is 50.

15 - Record Number - Tag's record number in variable.DBF - 1. That is, the first tag has a record number of 0.

16 - Comment - As defined in the variable tags list.

- 17 - ClusterName of the tag. If the association tag is not resolved, returns an empty string.
- 18 - Full name (*cluster.tagname*) of the associated tag. If the association tag is not resolved, returns an empty string.
- 19 - Full name (*cluster.tagname*) of the associated tag even if the association tag is not resolved.

Return Value

The value of the information as a string.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#), [TagGetProperty](#), [TagGetScale](#), [TagScaleStr](#), [TagInfo](#), [TagInfoEx](#)

Example

```
//Using a string identifier for the substitution parameter  
AssInfo("MILK_LEVEL", 1); (to get Engineering Units of the association )
```

```
sTag = AssInfo(1, 0); // Get the name of association 1  
sEngLow = AssInfo(1, 4); // get the low engineering scale of association 1
```

See Also

[Super Genie Functions](#)

AssInfoEx

Gets association information about the current Super Genie (that is information about a variable tag that has been substituted into the Super Genie). You can only call this function on a Super Genie after the associations are completed.

Use this function to display association information as part of the Super Genie. For example, if you have a Super Genie that is a loop controller, you could display the name of the loop at the top of the loop controller box. Each time the Super Genie is used with different associations (specifically a different tag name association) the correct loop name will be displayed.

Note: When replacing an instance of AssInfo with AssInfoEx in a loop, you may want to make AssInfoEx blocking using the iCached argument to verify you are using the correct value for the Tag in your logic.

Syntax

AssInfoEx(nArg, nType [, iCached])

nArg:

When you associate variable tags with super Genies, the Super Genie substitution strings are replaced by variable tags. The nArg argument allows you to get information about one of those variable tags. What you need to know is which substitution string it replaced when the association was performed.

Enter the argument number (substitution string number) of the relevant substitution string. For example, if you want information about the variable that replaced substitution string

?INT 3?

set nArg to 3.

nType:

The type of information to get:

0 - The Tag name of the association. If the association tag is not resolved, returns an empty string.

1 - Engineering units

2 - Raw zero scale

3 - Raw full scale

4 - Engineering zero scale

5 - Engineering full scale

6 - Width of the format

7 - Number of decimal places of format

8-16 - Not supported

17 - ClusterName of the tag. If the association tag is not resolved, returns an empty string.

18 - Full name (*cluster.tagname*) of the associated tag. If the association tag is not resolved, returns an empty string.

19 - Full name (*cluster.tagname*) of the associated tag even if the association tag is not resolved.

iCached:

Optional flag to attempt to retrieve the cached value for the property rather than the current value. This makes the function non-blocking. If the property has not yet been cached, an error is set.

0 - Do not force cached read. Cicode is blocking

1 - Force cached read. Cicode is non-blocking

Default value is 1 (true).

Return Value

The value of the information as a string. If an error is detected an empty string is returned. The error code can be obtained by calling the [IsError](#) Cicode function.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#), [TagGetProperty](#), [TagGetScale](#), [TagScaleStr](#), [TagInfo](#), [TagInfoEx](#)

Example

```
//Using a string identifier for the substitution parameter  
  
AssInfoEx("MILK_LEVEL", 1, 0); // to get Engineering units, no cache read  
  
  
sTag = AssInfoEx(1, 0); // Get the name of association 1 in non-blocking mode  
sEngLow = AssInfoEx(1, 4, 0); // get the low engineering scale of association 1,  
block until data is available
```

See Also

[Super Genie Functions](#)

AssPage

Associates up to eight variable tags with a Super Genie and displays the Super Genie in the current window. The first variable tag (*sTag1*) replaces Super Genie substitution string 1. The second variable tag (*sTag2*) replaces substitution string 2, and so on.

This function has the same effect as calling Ass() or AssTag() eight times, and then calling the PageDisplay() function. The AssPage() function provides a quick way of associating eight Super Genie variables and displaying the Super Genie - at the same time.

If you want to associate more than eight tags with the Super Genie, it is strongly recommended you call the AssVarTags(), AssTag(), or Ass() function to create the associations before you call this function.

Syntax

AssPage(*sPage*, *sTag1*, [*sTag2..8*])

sPage:

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

sTag1..sTag8:

The first eight physical tags to be associated with the Super Genie. For any given Super Genie, the variable tags will replace the Super Genie substitution strings as follows:

Variable tag...	replaces substitution string...
sTag1	1
sTag2	2
sTag3	3
sTag4	4
sTag5	5
sTag6	6
sTag7	7
sTag8	8

Because there is a strict correlation between the variable tag numbers and the substitution string numbers, you need to know how your Super Genie substitutions are numbered. For example, if your Super Genie has three unique substitution strings, numbered 1, 3, & 4, you need to enter a blank ("") for sTag2.

The variable tags that you specify here need to be the same data type as that specified by the relevant Super Genie substitution strings. For example, only a digital tag could replace the substitution string ?DIGITAL 4?. If the substitution string does not specify a type (for example, ?5?), you can use any type except STRING.

The name of the tag can be prefixed by the name of the cluster for example, "ClusterName.Tag".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Associate 3 tags with the Super Genie then display the Super Genie
AssPage("!MyGenie", "PV123", "OP123", "SP123");
```

See Also

[Super Genie Functions](#)

AssPopUp

Associates up to eight variable tags with a Super Genie and displays the Super Genie in a popup window. The first variable tag (*sTag1*) replaces Super Genie substitution string 1. The second variable tag (*sTag2*) replaces substitution string 2, and so on.

This function has the same effect as calling the Ass() function or the AssTag() function eight times, and then calling the WinNewAt() function to create a window at the position of the mouse. The AssPopUp() function is a quick way of associating eight Super Genie variables and displaying the Super Genie in a new window at the same time.

If you want to associate more than eight tags with the Super Genie, you need to call the AssVarTags(), AssTag(), or Ass() function to create the associations before you call this function.

Note: This function helps to prevent the Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

Syntax

AssPopUp(*sPage*, *sTag1*, [*sTag2..8*])

sPage:

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

sTag1..sTag8:

The first 8 physical tags to be associated with the Super Genie. For any given Super Genie, the variable tags will replace the Super Genie substitution strings as follows:

Variable tag...	replaces substitution string...
sTag1	1
sTag2	2
sTag3	3
sTag4	4
sTag5	5
sTag6	6
sTag7	7
sTag8	8

Because there is a strict correlation between the variable tag numbers and the substitution string numbers, you need to know how your Super Genie substitutions are numbered. For example, if your Super Genie has three unique substitution strings, numbered 1, 3, & 4, you have to enter a blank ("") for sTag2.

The variable tags that you specify here needs to be the same data type as that specified by the relevant Super Genie substitution strings. For example, only a digital tag could replace the substitution string ?DIGITAL 4?. If the substitution string does not specify a type (for example, ?5?), you can use any type except STRING.

The name of the tag can be prefixed by the name of the cluster for example, "ClusterName.Tag".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Associate 3 tags with the Super Genie then display it  
AssPopUp("!MyGenie", "PV123", "OP123", "SP123");
```

See Also

[Super Genie Functions](#)

AssScaleStr

Gets scale information about the associations of the current Super Genie (that is scale information about a variable tag that has been substituted into the Super Genie). You can only call this function on a Super Genie after the associations are completed.

Use this function to display association scale information as part of the Super Genie. For example, if you have a bar graph illustrating output, you could indicate zero, 50%, and full scale output on the vertical axis of the graph. Each time the Super Genie is used with different associations the correct scale values will be displayed.

The value is returned as a formatted string using the association format specification and (optionally) the engineering units.

Note: This function is being deprecated and is replaced by the [AssGetScale](#) function. If the Tag properties are updated AssScaleStr does not get the updated values whereas AssGetScale does.

Syntax

AssScaleStr(*nArg*, *Percent*, *EngUnits*)

nArg:

When you associate variable tags with super Genies, the Super Genie substitution strings are replaced by variable tags. The *nArg* argument allows you to get scale information about a particular variable tag. You need to know which substitution string the tag replaced when the association was performed.

Enter the argument number or name(substitution string number or name) of the relevant substitution string. For example, if you want scale information about the variable that replaced substitution string:

?INT 3?

set *nArg* to 3.

or

?Level?

set nArg to Level

Percent:

The percentage of full scale of the returned value.

EngUnits:

Determines if the value is returned with engineering units:

- 0 - Do not return the value with engineering units
- 1 - Return the value with engineering units

Return Value

The scale of the association (as a string).

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssTag](#), [AssTitle](#), [AssVarTags](#), [AssWin](#), [TagGetProperty](#), [TagGetScale](#), [TagScaleStr](#), [TagInfo](#)

Example

```
//Using a string identifier for the substitution parameter
AssScaleStr("MILK_LEVEL", 50, 1);
```

```
// Display the zero, 50% and full scale of the variable that was substituted for
// Super Genie arg no. 3
DspText(31,0,AssScaleStr(3, 0, 1));
DspText(32,0,AssScaleStr(3, 50, 1));
DspText(33,0,AssScaleStr(3, 100, 1));
```

See Also

[Super Genie Functions](#)

AssTag

Associates a variable tag with the a Super Genie. The association will only be created for the next Super Genie you display in the current window, and will only come into effect after you re-display the Super Genie. You need to call this function once for every substitution string in the current Super Genie, or the super-genie variable (substitution

string) will remain uninitialized and it will display as #ASS. You cannot use this function to create associations for variables that will display in new windows.

Syntax

AssTag(*nArg*, *sTag* [, *ClusterName*])

nArg:

The argument number (substitution string number) of the Super Genie string to be replaced by *sTag*. For example, to replace ?INT 3? with *sTag*, set *nArg* to 3.

sTag:

The variable tag that will replace the Super Genie substitution string. The tag needs to be the same data type as that specified by the Super Genie substitution string. For example, only a digital tag could replace the substitution string ?DIGITAL 4?. If the substitution string does not specify a type (for example, ?5?), you can use any type except STRING.

The name of the tag can be prefixed by the name of the cluster for example, "*ClusterName.Tag*".

ClusterName:

Specifies the name of the cluster in which the Variable Tag resides. This is optional if you have one cluster or are resolving the tag via the current cluster context. The argument is enclosed in quotation marks "".

Resolution of the tag's cluster context occurs when the page is displayed. It is resolved to the page's cluster context, not the context in force when this function is called.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTitle](#), [AssVarTags](#), [AssWin](#)

Example

```
// Associate variable tag PV123 and PV124 with !MyGenie
AssTag(1, "PV123");
AssTag(2, "PV124");
// Re-display the current Super Genie
PageDisplay("!MyGenie");
```

See Also

[Super Genie Functions](#)

AssTitle

Sets the runtime window title to the tag name of the first variable substituted into the Super Genie.

Note: This function does not support named associations.

See [Page Properties - General](#) for information regarding using named associations in the Window Title field.

Syntax

AssTitle([Mask] [, Prefix] [, Suffix])

Mask:

The number of characters to mask (hide) from the right of the title string (optional).

Prefix:

A string to add to the beginning of the title string (optional).

Suffix:

A string to add to the end of the title string (optional).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssVarTags](#), [AssWin](#)

See Also

[Super Genie Functions](#)

AssVarTags

Associates up to eight variable tags with a Super Genie. This association is only made for the next Super Genie you display (either in the current window or in a new window). This function has an offset that allows you to specify which substitution string the first variable tag will replace. This means that if you have a Super Genie with more than 8 substitution strings, you can use this function repeatedly (while increasing the offset), until you have associated the necessary variable tags.

This function has the same effect as calling the Ass() function or the AssTag() function eight times. The AssVarTags() function is a quick way of associating up to eight Super Genie variables at the same time.

Note: This function does not support named associations. To use named associations refer to [AssMetadata](#).

Syntax

AssVarTags(*hWin*, *nOffset*, *sTag1*, [*sTag2..8*])

hWin:

The association will be created for the next Super Genie to display in the window specified here - enter the window number or:

- 3 - for the current window when the page is changed. The page can be changed by using the Page Cicode functions like [PageDisplay](#), [PageGoto](#), etc.
- 2 - for the next new window or page displayed.

nOffset:

By default, the first variable tag (*sTag1*) will replace substitution string 1, and *sTag2* will replace substitution string 2, and so on. Enter an offset to change this so that *sTag1* replaces a substitution string other than the first. For example, an offset of 8 means that *sTag1* replaces string 9 instead of the default string 1 (8+1=9), and *sTag2* replaces string 10 instead of string 2 (8+2=10) etc. This means that you can use this function repeatedly to associate more than eight variables.

sTag1..8:

The physical variable tags (up to eight) to be associated with the Super Genie. For any given Super Genie, the variable tags will replace the Super Genie substitution strings as follows:

If <i>nOffset</i> is 0...	<i>sTag1</i> will replace the substitution string 1,
	<i>sTag2</i> will replace the substitution string 2, etc.
If <i>nOffset</i> is 8...	<i>sTag1</i> will replace the substitution string 9,
	<i>sTag2</i> will replace the substitution string 10, etc.

Because there is a strict correlation between the variable tag numbers and the substitution string numbers, you need to know how your Super Genie substitutions are numbered. For example, if your Super Genie has three unique substitution strings, numbered 1, 3, & 4, you need to enter a blank ("") for sTag2.

The name of the tag can be prefixed by the name of the cluster for example, "ClusterName.Tag".

Return Value

No value is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssWin](#)

Example

```
// Associate 12 variables to the Super Genie
AssVarTags(WinNumber(), 0, "PV123", "SP123", "OP123", "PV124", "SP124", "OP124",
            "PV125", "SP125");
AssVarTags(WinNumber(), 8, "OP125", "PV126", "SP126", "OP126");
PageDisplay("!MyGenie"); // Display the Super Genie
```

See Also

[Super Genie Functions](#)

AssWin

Associates up to eight variable tags with a Super Genie, and displays the Super Genie in a new window. This function has the same effect as calling the Ass() or AssTag() function eight times, and then calling the WinNewAt() function. The AssWin() function is a quick way of associating eight Super Genie variables and creating a new window - at the same time.

If you want to associate more than eight tags with the Super Genie you need to call the AssVarTags(), AssTag(), or Ass() function to create the associations before you call this function.

Syntax

AssWin(*sPage*, *X*, *Y*, *Mode*, *sTag1*, [*sTag2..8*])

sPage:

The page name of the Super Genie. If you prefixed your Super Genie page name with an exclamation mark (!), remember to include it here.

X - The x pixel coordinate of the top left corner of the window.

Y - The y pixel coordinate of the top left corner of the window.

Mode:

The mode of the window:

0 - Normal page.

1 - Page child window. The window is closed when a new page is displayed, for example, when the PageDisplay() or PageGoto() function is called. The parent is the current active window.

2 - Window child window. The window is closed automatically when the parent window is freed with the WinFree() function. The parent is the current active window.

4 - No re-size. The window is displayed with thin borders and no maximize/minimize icons. The window cannot be re-sized.

8 - No icons. The window is displayed with thin borders and no maximize/minimize or system menu icons. The window cannot be re-sized.

16 - No caption. The window is displayed with thin borders, no caption, and no maximize/minimize or system menu icons. The window cannot be re-sized.

32 - Echo enabled. When enabled, keyboard echo, prompts, and error messages are displayed on the parent window. This mode should only be used with child windows (for example, Mode 1 and 2).

64 - Always on top.

128 - Open a unique window. This mode stops this window from being opened more than once.

256 - Display the entire window. This mode commands that no parts of the window will appear off the screen

512 - Open a unique Super Genie. This mode stops a Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

1024 - Disables dynamic resizing of the new window, overriding the setting of the [Page]DynamicSizing parameter.

You can select multiple modes by adding modes together (for example, set Mode to 9 to open a page child window without maximize, minimize, or system menu icons).

sTag1..8:

The first eight physical tags to be associated with the Super Genie. For any given Super Genie, the variable tags will replace the Super Genie substitution strings as follows:

Variable tag...	replaces substitution string...
sTag1	1
sTag2	2
sTag3	3
sTag4	4
sTag5	5
sTag6	6
sTag7	7
sTag8	8

Because there is a strict correlation between the variable tag numbers and the substitution string numbers, you need to know how your Super Genie substitutions are numbered. For example, if your Super Genie has three unique substitution strings, numbered 1, 3, & 4, you need to enter a blank ("") for sTag2.

The variable tags that you specify here need to be the same data type as that specified by the relevant Super Genie substitution strings. For example, only a digital tag could replace the substitution string ?DIGITAL 4?. If the substitution string does not specify a type (for example, ?5?), you can use any type except STRING.

The name of the tag can be prefixed by the name of the cluster for example, "ClusterName.Tag".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Ass](#), [AssChain](#), [AssMetadata](#), [AssMetadataPage](#), [AssMetadataPopup](#), [AssMetadataWin](#), [AssChainPage](#), [AssChainPopUp](#), [AssChainWin](#), [AssChainWinFree](#), [AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssInfoEx](#), [AssPage](#), [AssPopUp](#), [AssScaleStr](#), [AssTag](#), [AssTitle](#), [AssVarTags](#)

Example

```
// Associate 3 tags with the Super Genie
// then display the new window at (100,200) in mode 9
```

```
AssWin("!MyGenie", 100, 200, 1 + 8, "PV123", "OP123", "SP123");
```

See Also

[Super Genie Functions](#)

Chapter: 52 Table (Array) Functions

Table functions perform mathematical functions on entire tables (or arrays), such as the calculation of minimum, maximum, average, and standard deviation values.

Note: This function only supports arrays declared in Cicode and not variable tag arrays.

Table (Array) Functions

Following are functions relating to Tables:

Table-Lookup	Gets a value from a table.
Table-Math	Performs mathematical operations on a table, for example, average, maximum, minimum, etc.
Table-Shift	Shifts a table, left or right.

Note: This function only supports arrays declared in Cicode and not variable tag arrays.

See Also

[Functions Reference](#)

TableLookup

Searches for a value in a table, and returns the position (offset) of the value in the table. Be aware that the first item in a table is offset 0 (zero), the next item is offset 1, etc.

Note: This function only supports arrays declared in Cicode and not variable tag arrays.

Syntax

TableLookup(*Table*, *Size*, *Value*)

Table:

The table to search. The table needs to be an array of real numbers.

Size:

The maximum number of items in the table.

Value:

The value to locate.

Return Value

The offset to the table value, or -1 if the value does not exist.

Related Functions

[TableMath](#)

Example

```
REAL Levels[5]=10,15,50,100,200;
Variable=TableLookup(Levels,5,50);
! Sets Variable to 2.
Variable=TableLookup(Levels,5,45);
! Sets Variable to -1.
```

See Also

[Table \(Array\) Functions](#)

TableMath

Performs mathematical operations on a table of real (floating-point) numbers. This function supports minimum, maximum, average, standard deviation, and total operations on a table of values. Use this function for operating on tables returned from the trend system with the TrnGetTable() function. You can set the *Mode* to either accept or ignore invalid or gated data returned from TrnGetTable().

Notes:

This function cannot check the length of any arrays passed to it. If the array is shorter than the size argument, unpredictable results can occur, such as data in memory being overwritten, or a general protection fault.

This function only supports arrays declared in Cicode and not variable tag arrays.

WARNING

UNINTENDED EQUIPMENT OPERATION

Always confirm that arrays are of appropriate length before passing them to the TableMath function.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax

TableMath(*Table*, *Size*, *Command* [, *Mode*])

Table:

Any table of floating-point numbers.

Size:

The maximum number of items in the table.

Command:

The mathematical operation to perform on the table:

0 - Minimum

1 - Maximum

2 - Average

3 - Standard deviation

4 - Total

Mode:

The mode of the operation:

0 - Operate on all data - default

1 - Ignore invalid or gated data returned from the TrnGetTable() function

Return Value

Returns the value related to the requested mathematical operation performed on the table (Minimum, Maximum, Average, Standard deviation or Total).\\

Related Functions

[TableLookup](#) [TrnGetTable](#)

Example

```
REAL Array[5]=10,15,50,100,200;
REAL Min,Avg;
! Get the minimum value.
Min=TableMath(Array, 5, 0, 0);      ! Sets Min to 10.
! Get the average value.
Avg=TableMath(Array, 5, 2, 0);      ! Sets Avg to 75.
```

See Also

[Table \(Array\) Functions](#)

TableShift

Shifts table items in a table by a number of positions. You can shift the table left or right. Items shifted off the end of the table are lost. Items within a table that are not replaced by other items (that have moved) are set to 0.

Note: This function only supports arrays declared in Cicode and not variable tag arrays.

Syntax

TableShift(*Table*, *Size*, *Count*)

Table:

The table to shift, an array of real numbers.

Size:

The maximum number of items in the table.

Count:

The number of positions to shift the table items. A negative Count moves items to the right and a positive Count moves items to the left.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TableMath](#), [TableLookup](#)

Example

```
REAL Levels[5]=10,15,50,100,200;
TableShift(Levels,5,2);
/* Shifts the table items by 2 positions to the left, that is
Levels[0]=50
Levels[1]=100
Levels[2]=200
Levels[3]=0
Levels[4]=0 */
```

See Also

[Table \(Array\) Functions](#)

Chapter: 53 Tag Functions

The Tag functions allow you to read the values of variables in I/O devices such as PLCs, and to write data into these I/O device variables. These functions also allow you to control I/O devices and to display information about I/O devices.

Tag Functions

Following are functions relating to Tags:

SubscriptionAddCallback	Adds a callback function to a tag subscription.
SubscriptionGetAttribute	Reads an attribute value of a tag subscription.
SubscriptionGetInfo	Reads the specified text information about a subscribed tag.
SubscriptionGetQuality	Reads quality of a subscribed tag.
SubscriptionGetTag	Reads a value, quality and timestamps of a subscribed tag.
SubscriptionGetTimestamp	Reads the specified timestamp of a subscribed tag
SubscriptionGetValue	Reads a value of a subscribed tag.
Sub- scriptionRemoveCallback	Removes a callback function from a tag subscription
TagDebug	Displays a dialog which allows you to select any configured tag to read or change (write) its value.
TagEventFormat	Returns a handle to the format of the data used by the TagEventQueue().
TagEventQueue	Opens the tag update event queue.
TagGetProperty	Gets a property for a variable tag from the datasource.
TagGetScale	Gets the value of a tag at a specified scale from the datasource
TagInfo	Gets information about a variable tag.
TagInfoEx	Gets information about a variable tag.
TagRamp	Increments a tag by a percentage amount.
TagRDBReload	Reloads the variable tag database so when TagInfo is called it picks up online changes to the tag database.

TagRead	Reads a variable from an I/O Device.
TagReadEx	Reads the value, quality and timestamp of a particular tag element.
TagScaleStr	Gets the value of a tag at a specified scale.
TagSetOverrideBad	Sets a quality Override element for a specified tag to Bad Non Specific.
TagSetOverrideGood	Sets a quality Override element for a specified tag to Good Non Specific.
TagSetOverrideUncertain	Sets a quality Override element for a specified tag to Uncertain Non Specific.
TagSetOverrideQuality	Sets a quality Override element for a specified tag.
TagSubscribe	Subscribes a tag for periodic monitoring and event handling.
TagUnsubscribe	Unsubscribes a tag for periodic monitoring and event handling.
TagWrite	Writes to an I/O Device variable
TagWriteEventQue	Opens the tag write event queue.

See Also[Functions Reference](#)**SubscriptionAddCallback**

Adds a function callback to a tag subscription. When the value change for a subscribed tag is detected, a callback function can be called. This implements change based Cicode and avoids continuously polling a tag value to monitor changes.

Multiple callbacks are possible to the same subscription.

To remove a callback from a subscription use the [SubscriptionRemoveCallback](#) function.

Syntax

SubscriptionAddCallback(*iHandle*, *sCallback*)

iHandle

Integer handle of the subscription to add a callback to.

sCallback

String stating the name of a function to call when the value is updated. The function should have the structure:

```
FUNCTION evtHandler(INT subsHandle)
    ..
```

End

Where *subsHandle* is the subscription that raised the event.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TagSubscribe](#), [TagUnsubscribe](#), [SubscriptionGetAttribute](#), [SubscriptionRemoveCallback](#)

See Also

[Tag Functions](#)

SubscriptionGetAttribute

Reads the specified attribute value of a subscribed tag. Similar to [TagRead](#).

Note: This function has been superseded and may be deprecated in a future release. Please use the following functions: [SubscriptionGetInfo](#), [SubscriptionGetTimestamp](#), [SubscriptionGetQuality](#), [SubscriptionGetValue](#) or [SubscriptionGetTag](#) as substitutions in new projects.

Syntax

SubscriptionGetAttribute(*iHandle*, *sAttribute* [, *iOffset*])

iHandle

Integer handle of the subscription to read from.

sAttribute

Attribute of the tag to read. Supported Attributes are:

- **ClusterName** - Returns the resultant cluster context of the subscription. For example, for the tag subscribed as "cluster1.tagname", the return value is "cluster1".

There are several possible outcomes where no cluster is specified in the subscription:

- If the tag is a local tag, an empty string will be returned.
- If the tag is a variable tag and the system only contains one cluster, this cluster will be returned.

- If the tag is a variable tag and there is a default cluster being run in the Cicode, the default cluster will be returned.
- If none of these options are true, an empty string will be returned.
- **FullName** - Return the full subscription name. For example, for the tag subscribed as "cluster1.tagname", return value is "cluster1.tagname". If the tag was subscribed without a cluster the return value will be the tagname.
- **TagName** - Return the tagname for the subscription. For example, for the tag subscribed as "cluster1.tagname", return value is "tagname".
- **Value** - The current value of the tag.
- **ValueQuality** - An indication of the current quality of the tag as an integer number.

Note: The return value is not compatible with the QUALITY data type or the quality Cicode functions. Use SubscriptionGetQuality.

- **ValueTimestamp** - The time when the tag last changed. It is returned as an integer value compatible with a time/data variable.

Note: The return value is not compatible with the TIMESTAMP data type. Use SubscriptionGetTimestamp.

- **ValueTimestampMS** - The millisecond part of the time when the tag last changed.

iOffset

Optional integer expressing the zero based index of an array attribute. This is only valid for the Value attribute. Default value is 0.

Return Value

String representation of the cached value for a subscribed tag. On error, an empty string and an [error](#) is set.

Related Functions

[SubscriptionGetInfo](#), [SubscriptionGetTimestamp](#), [SubscriptionGetQuality](#), [SubscriptionGetValue](#), [SubscriptionGetTag](#)

See Also[I/O Device Functions](#)

SubscriptionGetInfo

Reads the specified text information about a subscribed tag.

Syntax

SubscriptionGetInfo(*iHandle*, *sAttribute*)

iHandle

Integer handle of the subscription to read from.

sAttribute

Attribute of the tag to read. Supported Attributes are:

- **ClusterName** - Return the cluster context of the subscription. For example, for the tag subscribed as "cluster1.tagname", return value is "cluster1". If the tag was subscribed without a cluster the return value will be an empty string.
- **FullName** - Return the full subscription name. For example, for the tag subscribed as "cluster1.tagname.field", return value is "cluster1.tagname.field". If the tag was subscribed without a cluster, the return value will be the tag name and/or element name. If the tag was subscribed without an element, the return value will be the tag name and/or cluster name.
- **TagName** - Return the tagname for the subscription. For example, for the tag subscribed as "cluster1.tagname", return value is "tagname".
- **ElementName** - Retrieve the element name of the subscription.

Return Value

String representation of the requested information for a subscribed tag. On error, returns an empty string and an error is set.

Related Functions

[SubscriptionGetTimestamp](#), [SubscriptionGetQuality](#), [SubscriptionGetValue](#), [SubscriptionGetTag](#)

Example

```
STRING TagName = SubscriptionGetInfo(hSub, "TagName");
```

See Also

[Tag Functions](#)

SubscriptionGetQuality

Reads quality of a subscribed tag.

Syntax

SubscriptionGetQuality(*iHandle*)

iHandle

Integer handle of the subscription to read from.

Return Value

The quality for a subscribed tag. On error, QUAL_BAD.

Related Functions

[SubscriptionGetInfo](#), [SubscriptionGetTimestamp](#), [SubscriptionGetValue](#), [SubscriptionGetTag](#)

Example

```
QUALITY theQuality = SubscriptionGetQuality(hSub);
```

See Also

[Tag Functions](#)

SubscriptionGetTag

Reads a value, quality and timestamps of a subscribed tag.

Syntax

SubscriptionGetTag(*iHandle*, *sOffset*)

iHandle

Integer handle of the subscription to read from.

sOffset

Optional integer expressing the zero based index of an array attribute. This is only valid for the Value attribute. Default value is 0.

Return Value

Returns a value, quality and timestamps of a subscribed tag. The type of the returned value depends on a type of the subscribed tag. The quality and timestamps of the subscribed tag are read and passed with the returned value.

Using `SubscriptionGetValue` gives similar results as using direct reference to a tag without item ex. `tag1`, `tag1.Field`.

On error, returns either 0 for numerical data types or an empty string for strings.

Related Functions

[SubscriptionGetTimestamp](#), [SubscriptionGetQuality](#), [SubscriptionGetInfo](#), [SubscriptionGetValue](#)

Example

```
INT Value = SubscriptionGetTag(hSub);
```

See Also

[Tag Functions](#)

SubscriptionGetTimestamp

Reads the specified timestamp of a subscribed tag.

Syntax

SubscriptionGetTimestamp(*iHandle*, *sAttribute*)

iHandle

Integer handle of the subscription to read from.

sAttribute

Attribute of the tag to read. Supported Attributes are:

- **Timestamp** - The timestamp when the tag last changed. It is default value used when this argument is not specified.
- **QualityTimestamp** - The timestamp when quality of the tag last changed.
- **ValueTimestamp** - The timestamp when value of the tag last changed.

Return Value

The requested timestamp for a subscribed tag. On error, 0 (INVALID TIMESTAMP).

Related Functions

[SubscriptionGetInfo](#), [SubscriptionGetQuality](#), [SubscriptionGetValue](#), [SubscriptionGetTag](#)

Example

```
TIMESTAMP theTime = SubscriptionGetTimestamp(hSub, "ValueTimestamp");
```

```
TIMESTAMP theTime = SubscriptionGetTimestamp(hSub);
```

See Also

[Tag Functions](#)

SubscriptionGetValue

Reads a value of a subscribed tag.

Syntax

SubscriptionGetValue(*iHandle*, *sOffset*)

iHandle

Integer handle of the subscription to read from.

sOffset

Optional integer expressing the zero based index of an array attribute. Default value is 0.

Return Value

Returns a value of a subscribed tag. The type of the returned variable depends on a type of the subscribed tag. The quality and timestamps of the subscribed tag are not read i.e. quality of the returned value can be consider as GOOD and its timestamps as 0 (INVALID TIMESTAMP).

Using SubscriptionGetValue gives similar results as using direct reference to tag's v item ex. tag1.v, tag1.Field.v.

On error, returns either 0 for numerical data types or an empty string for strings.

Related Functions

[SubscriptionGetTimestamp](#), [SubscriptionGetQuality](#), [SubscriptionGetInfo](#), [SubscriptionGetTag](#)

Example

```
INT Value = SubscriptionGetValue(hSub);
```

See Also

[Tag Functions](#)

SubscriptionRemoveCallback

Removes a function callback from a tag subscription. The subscription handle and callback function needs to match those used when adding the callback.

Syntax

SubscriptionRemoveCallback(*iHandle*, *sCallback*)

iHandle

Integer handle of the subscription of the callback.

sCallback

String stating the name of the callback function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TagSubscribe](#), [TagUnsubscribe](#), [SubscriptionAddCallback](#), [SubscriptionGetAttribute](#)

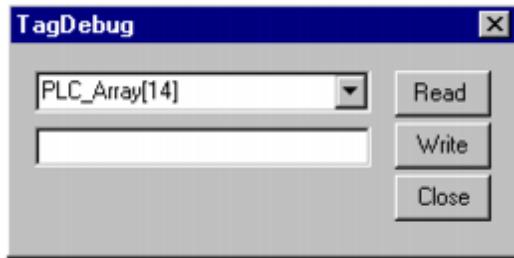
See Also

[Tag Functions](#)

TagDebug

Displays a dialog which allows you to select from a list of the configured variable tags in your system. Once you have selected a tag, you can either press the **Read** button to get the tag's current value; or change the value by entering a new one, and pressing the **Write** button. This function should only be used for debugging or commissioning.

To read or change (write) the value of an element in an array variable, type it into the dialog's variable tag field as follows:



Syntax

TagDebug()

Return Value

The name of the tag entered in the dialog.

Related Functions

[TagInfo](#), [TagRead](#), [TagWrite](#)

Example

```
TagDebug(); /* Display debug form to allow user to debug */
```

See Also

[Tag Functions](#)

TagEventFormat

Returns a handle to the format of the data used by the TagEventQueue().

Syntax

TagEventFormat()

Parameters - None

Return Value

The format handle.

Related Functions

[QuePeek](#), [QueRead](#), [TagEventQueue](#)

Example

Not applicable.

See Also

[Tag Functions](#)

TagEventQueue

Opens the tag update event queue. The I/O server writes events into this queue as they are processed. These events include tag updates from drivers that support time-stamped data.

To read events from this queue, use the [QueRead\(\)](#) or [QuePeek\(\)](#) functions. The data put into the queue contains the following fields:

- Driver (the driver used to communicate with the I/O device)
- Port (the name of the port to which the I/O device is connected)
- Unit (the name of the I/O device)
- Tag (the variable tag name)
- Seconds (a UTC timestamp representing the number of seconds since 1970)
- Milliseconds (number of milliseconds since the last second)
- Value (the tag value)
- Quality (OPC component only equivalent to QualityGetPart, mode 7)

To use this function, you need to enable the tag update event queue with the [IOServer]EnableEventQueue parameter. This parameter will tell the I/O Server to start placing events into the queue. The function TagEventFormat() returns a handle to the format of the data placed into the string field.

Enabling this formatting feature can increase CPU loading and reduce performance of the I/O Server as every tag update event is formatted and placed in the queue. You should reconsider using this feature if a decrease in performance is noticeable.

The maximum length of each queue is controlled by the [Code]Queue parameter. You may need to adjust this parameter so as not to miss alarm events. When the queue is full, the I/O Server will discard events.

Syntax

TagEventQueue()

Parameters - None

Return Value

The queue handle of the Tag Update Event queue.

Related Functions

[QuePeek](#), [QueRead](#), [TagEventFormat](#)

Example

```
FUNCTION
```

```
    ReadEvents ()
```

```
        INT status;
```

```
        INT queue;
```

```
        INT format;
```

```
        INT eventId;
```

```
        STRING event;
```

```
        INT sec;
```

```
        INT ms;
```

```
        TIMESTAMP time;
```

```
queue = TagEventQueue ();
```

```
format = TagEventFormat ();
```

```
IF (queue <> -1 AND format <> -1) THEN  
  
    WHILE (true) DO  
  
        status = QueRead(queue, eventId, event, 1);  
  
        IF status = 0 THEN  
  
            ErrLog("eventId: " + IntToStr(eventId));  
  
            StrToFmt(format, event);  
  
            ErrLog("    driver: " + FmtGetField(format, "Driver"));  
  
            ErrLog("    port: " + FmtGetField(format, "Port"));  
  
            ErrLog("    unit: " + FmtGetField(format, "Unit"));  
  
            ErrLog("    tag: " + FmtGetField(format, "Tag"));  
  
            sec = FmtGetField(format, "Seconds");  
  
            ms = FmtGetField(format, "Milliseconds");  
  
            time = TimeIntToTimestamp(sec, ms, 1);  
  
            ErrLog("    time: " + TimestampToStr(time,14));  
  
            ErrLog("    value: " + FmtGetField(format, "Value"));  
  
            ErrLog("    quality: " + FmtGetField(format, "Quality"));  
  
        END  
  
    END
```

END

END

See Also

[Tag Functions](#)

TagGetProperty

This function reads a property of a variable tag from the datasource. This function replaces [TagInfo](#).

Syntax

TagGetProperty(*sName*, *sProperty* [*iCached*] [*ClusterName*])

sName:

The name of the tag from which to get information. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

sProperty:

The property to read. Property names are case sensitive. Supported properties are:

Address - Returns the configured address of the tag (as specified in the variable tags form).

ArraySize - Array size of the associated tag. Returns 1 for non-array types.

DataBitWidth - Number of bits used to store the value

Description - Tag description

EngUnitsHigh - Maximum scaled value

EngUnitsLow - Minimum scaled value

Format - Format bit string. The format information is stored in the integer as follows:

- Bits 0-7 - format width
- Bits 8-15 - number of decimal places
- Bits 16 - zero-padded
- Bit 17 - left-justified
- Bit 18 - display engineering units
- Bit 20 - exponential (scientific) notation

FormatDecPlaces - Number of decimal places for default format

FormatWidth - Number of characters used in default format

RangeHigh - Maximum unscaled value

RangeLow - Minimum unscaled value

Type - Type of tag. Allowed values are:

- 0 - Digital
- 1 - Byte
- 2 - Integer16
- 3 - UInteger16
- 4 - Long
- 5 - Real
- 6 - String
- 7 - ULong
- 8 - Undefined

Units - Engineering Units for example, %, mm, Volts

iCached:

Optional flag to attempt to retrieve the cached value for the property rather than the current value. This makes the function non-blocking. If the property has not yet been cached, an error is set.

0 - Do not force cached read. Cicode is blocking

1 - Force cached read. Cicode is non-blocking

Default value is 1 (true).

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

String representation of the property of the tag. On error, an empty string and an [error](#) is set.

Related Functions

[AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssScaleStr](#), [TagGetProperty](#), [TagScaleStr](#), [TagInfo](#)

Example

```
// Get the engineering full scale value for the variable "PV131"
EngFullScale = TagGetProperty("PV131", "EngUnitsHigh", 0);
// Get the cached array size for the array variable "PLC_Array"
ArrayLength = TagGetProperty("PLC_Array", "ArraySize", 1);
```

See Also

[Tag Functions](#)

TagGetProperty

Gets the value of a tag at a specified scale from the datasource. The value is returned as a formatted string using the tags format specification and (optionally) the engineering units. Use this function to write generic Cicode that will work with any type of tag. This function replaces [TagScaleStr](#).

Syntax

TagGetScale(*sName*, *iPercent*, *iEngUnits* [, *iCached*] [, *ClusterName*])

sName:

The name of the tag. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

iPercent:

The percentage of full scale of the returned value.

iEngUnits:

Flag to determine if the value is returned with engineering units:

0 - Do not return the value with engineering units

1 - Return the value with engineering units

iCached:

Optional flag to attempt to retrieve the cached value for the property rather than the current value. This makes the function non-blocking. If the property has not yet been cached, an error is set.

0 - Do not force cached read. Cicode is blocking

1 - Force cached read. Cicode is non-blocking

Default value is 1 (true).

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

The scale of the tag (as a string).

Related Functions

[AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssScaleStr](#), [TagGetProperty](#), [TagScaleStr](#), [TagInfo](#)

Example

```
// Display the zero, 50% and full scale of the tag CV_123_PV
```

```
DspText(31,0,TagGetScale("CV_123_PV", 0, 1));
DspText(32,0,TagGetScale("CV_123_PV", 50, 1));
DspText(33,0,TagGetScale("CV_123_PV", 100, 1));
```

See Also[Tag Functions](#)**TagInfo**

Gets information about a variable tag. This function allows you to develop generic Cicode and Super Genies.

Syntax**TagInfo(sName, nType [, ClusterName])***sName:*

The name of the tag from which to get information. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

To get information on a particular element in an array, enter the array name here, followed by the number of the element as follows:

"PLC_Array[9]"

The above example tells the function to get information on the tenth element in PLC_Array (remember, the address of the first element in an array is 0 (zero)).

nType:

The type of information to get:

0 - The Tag name from the variables table. This is the same as sName argument. (Returned to be compatible with the AssInfo() function).

1 - Engineering units

2 - Raw zero scale

3 - Raw full scale

4 - Engineering zero scale

5 - Engineering full scale

6 - Width of the format

7 - Number of decimal places of format

8 - The Tag format as a long integer. The format information is stored in the integer as follows:

- Bits 0-7 - format width
- Bits 8-15 - number of decimal places
- Bits 16 - zero-padded

- Bit 17- left-justified
- Bit 18 - display engineering units
- Bit 20 - exponential (scientific) notation

9 - Logical Unit Number - I/O device number (for internal use)

10 - Raw Type - Protocol's raw data type number for this tag. Type numbers are:

- 0 - Digital
- 1 - Integer
- 2 - Real
- 3 - BCD
- 4 - Long
- 5 - Long BCD
- 6 - Long Real
- 7 - String
- 8 - Byte
- 9 - Void
- 10 - Unsigned integer

11 - Bit Width - Tag's size in bits. For example, an INT is 16 bits

12 - Unit Type - Protocol's unit type number for this tag

13 - Unit Address - Tag's address after the protocol DBF's template is applied.

14 - Unit Count - Array size. For example, if the tag's address is I1[50], the unit count is 50.

15 - Record Number - Tag's record number in variable.DBF - 1. That is, the first tag has a record number of 0.

16 - Comment - As defined in the variable tags list.

17 - ClusterName of the tag. If the tag is not resolved, returns an empty string.

18 - Full name (*cluster.tagname*) of the tag. If the tag is not resolved, returns an empty string.

19 - Reserved for internal operation.

20 - Configured Address of the tag. If the tag is not resolved, returns an empty string.

21 - Network Number - I/O device number (as defined by the Number field of the I/O Devices dialog).

If the tag is a local variable, modes 9, 11, 12, 13, 14, 15 and 17 will return an empty string, while mode 18 will return only the tag name (without the cluster specified).

ClusterName

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

The value of the information as a string.

Related Functions

[AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssScaleStr](#), [TagGetProperty](#), [TagGetScale](#), [TagInfoEx](#), [TagScaleStr](#)

Example

```
/* Get the engineering full scale value for the variable "PV131" */
EngFullScale = TagInfo("PV131", 5);
/* Get the engineering zero scale value for the array variable "PLC_Array" */
EngZeroScale = TagInfo("PLC_Array", 4);
```

See Also

[Tag Functions](#)

TagInfoEx

This function replaces TagInfo and supports online changes. It is recommended therefore that instances of TagInfo in legacy code are migrated to either TagInfoEx or TagGetProperty. New Cicode should use TagGetProperty.

Gets information about a variable tag. This function allows you to develop generic Cicode and Super Genies. Execution can be blocking or non-blocking depending on the iCached argument.

Note: When replacing an instance of TagInfo with TagInfoEx in a loop, you may want to make TagInfoEx blocking using the iCached argument so that you are using the correct value for the Tag in your logic. You should also be aware that TagInfo has different return values if you are using mode 10 for *nType*.

Syntax

TagInfoEx(*sName*, *nType* [, *ClusterName*] [, *iCached*])

sName:

The name of the tag from which to get information. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

To get information on a particular element in an array, enter the array name here, followed by the number of the element as follows:

"PLC_Array[9]"

The above example tells the function to get information on the tenth element in PLC_Array (remember, the address of the first element in an array is 0 (zero)).

nType:

The type of information to get:

0 - The Tag name from the variables table. This is the same as sName argument. (Returned to be compatible with the AssInfo() function).

1 - Engineering units

2 - Raw zero scale

3 - Raw full scale

4 - Engineering zero scale

5 - Engineering full scale

6 - Width of the format

7 - Number of decimal places of format

8 - The Tag format as a long integer. The format information is stored in the integer as follows:

- Bits 0-7 - format width
- Bits 8-15 - number of decimal places
- Bits 16 - zero-padded
- Bit 17 - left-justified
- Bit 18 - display engineering units
- Bit 20 - exponential (scientific) notation

9 - Logical Unit Number - I/O device number (for internal use)

10 - Raw Type - Protocol's raw data type number for this tag. Type numbers are:

- 0 - Digital
- 1 - Byte
- 2 - Integer16
- 3 - UInteger16
- 4 - Long
- 5 - Real
- 6 - String
- 7 - ULONG
- 8 - Undefined

11 - Bit Width - Tag's size in bits. For example, an INT is 16 bits

12-15 - Not supported

16 - Comment - As defined in the variable tags list.

17 - ClusterName of the tag.

18 - Full name (*cluster.tagname*) of the tag.

19 - Reserved for internal operation.

- 20 - Configured Address of the tag. If the tag is not resolved, returns an empty string.
- 21 - Network Number - I/O device number (as defined by the Number field of the I/O Devices dialog).

If the tag is a local variable, modes 9, 11 and 17 will return an empty string, while mode 18 will return only the tag name (without the cluster specified).

ClusterName

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

iCached:

Optional flag to attempt to retrieve the cached value for the property rather than the current value. This makes the function non-blocking. If the property has not yet been cached, an error is set.

- 0 - Do not force cached read. Cicode is blocking
 1 - Force cached read. Cicode is non-blocking

Default value is 1 (true).

Return Value

The value of the information as a string. In case of error an empty string is returned. The error code can be obtained by calling the [IsError](#) Cicode function.

Related Functions

[AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssScaleStr](#), [TagGetProperty](#), [TagGetScale](#), [TagInfo](#), [TagScaleStr](#)

Example

```
/* Get the engineering full scale value for the variable "PV131".
Obtain the value from Cluster1 in blocking mode */
EngFullScale = TagInfoEx("PV131", 5, "Cluster1", 0);
/* Get the engineering zero scale value for the array variable
"PLC_Array" in non-blocking mode*/
EngZeroScale = TagInfoEx("PLC_Array", 4);
```

See Also

[Tag Functions](#)

TagRamp

This function will increment a Tag by the amount defined by iPercentInc. It is often used for incrementing a tag while a button is held down.

Syntax

TagRamp(*sTag*, *iPercentInc*)

sTag:

The variable tag name (or alarm property name), as a string. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

To read a particular element in an array, you can enter the array name here, followed by an index to the element as follows:

"PLC_Array[9] "

The above example tells the function to read the 10th element in the array variable PLC_Array (remember, the address of the first element in an array is 0 (zero)).

If you enter an array offset using the *nOffset* argument, it will be added to the index value specified here. For example, TagRead("PLC_Array[9]", 4) will read the 14th element in PLC_Array (because [9] means the 10th element, and an offset of 4 means 4 elements after the 10th = element 14).

iPercentInc:

The percentage by which you want to increase the value of the variable. A negative number will decrease the variable. The increment is calculated as a percentage of the full range.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TagInfo](#), [TagRead](#), [TagWrite](#)

Example

Buttons	
Text	Ramp Up
Repeat Command	TagRamp("PLC_VAR_1",2);
Comment	Continual increment by 2%

See Also

[Tag Functions](#)

[TagRDBReload](#)

Works in conjunction with the TagInfo function. Reloads the variable tag database so when TagInfo is called it picks up all online changes to the tag database.

Syntax

TagRDBReload()

Return Value

Returns 1 if the tag database was successfully reloaded, and 0 if the tag database fails to load.

Note: This function will fail and return 0 if the parameter [\[General\]TagDB](#) as been set to 0.

Related Functions

[TagInfo](#)

See Also

[Tag Functions](#)

TagRead

Reads a variable from the I/O device. The variable tag needs to be defined in the Variable Tags database. Because the variable tag is specified as a string (not as a tag), you can ignore the data type of the variable.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

TagRead should only be used when the variable tag name is a calculation such as sAlarmExt+.Paging". For simple assignment of variables use the assignment operator. For example, MyString = MyCluster.MyAlarm.MyProperty.

If you try to read many variables at the same time, the TagRead() function may be slow. The offset index for array variables is checked against the size of the array.

Note: TagRead can only return the values of elements for those tags having "Good" quality. If the quality of a tag is not "Good", TagRead returns an empty string. In order to obtain values of tags having not "Good" quality one can use their 'v' item. For example, TagRead("MyBadQualityTag.v"). However, the value returned by a TagRead call on a tag with not "Good" quality may be obsolete (due to TagRead asynchronous nature). Use the [IsError](#) Cicode function to control the returned error when using this function.

Syntax

TagRead(STRING sTag [, INT nOffset [, STRING ClusterName]])

sTag:

The string can refer to either the alarm name and the alarm property name or the tag name with optional elements and items as it is defined by Tag Extensions [Cluster.]Tag[-.Element][.Item][[N]...]. (Refer to [Tag Extensions](#) for more information). Currently only v item is supported by TagRead, references to other items will give an error, for eg. TagRead("PLC_Tag.q"). If the element name is not specified, it will be resolved at runtime as an unqualified tag reference. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

To read a particular element in an array, you can enter the array name here, followed by an index to the element as follows:

"PLC_Array[9] "

The above example tells the function to read the 10th element in the array variable PLC_Array (remember, the address of the first element in an array is 0 (zero)).

If you enter an array offset using the *nOffset* argument, it will be added to the index value specified here. For example, TagRead("PLC_Array[9]", 4) will read the 14th element in PLC_Array (because [9] means the 10th element, and an offset of 4 means 4 elements after the 10th = element 14). If you want to read the value of <Valid> element in the above example,

TagRead("PLC_Array.Valid.V[9]", 4)

nOffset:

The offset for an array variable. This argument is optional - if not specified, it has a default value of 0.

If you enter an array index as part of the *sTag* argument, it will be added to this offset value. For example, TagRead("PLC_Array[9]", 4) will read the 14th element in PLC_Array (because [9] means the 10th element, and an offset of 4 means 4 elements after the 10th = element 14).

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

The function can return:

- The tag element value (depending on the *sTag* parameter, of the I/O device variable) when the tag has "Good" quality.
- The tag element value (depending on the *sTag* parameter, of the I/O device variable) and the error 257 (can be checked by IsError() built-in function) when the value of the tag is out of the predefined range.
- An empty string and an error when the tag has not "Good" quality".

Related Functions

[TagReadEx](#), [TagWrite](#), [IODeviceControl](#), [IODeviceInfo](#)

Example

```
STRING sStringTagValue;  
  
STRING sStringTagValueField;  
  
INT nIntTagValue;  
  
REAL fRealTagValue;  
  
// Tag1 has a STRING data type.  
  
sStringTagValue = TagRead("Tag1");  
  
sStringTagValueField = TagRead("Tag1.Field");  
  
// Tag2 has an INTEGER data type.  
  
nIntTagValue = TagRead("Tag2");  
  
// Tag3 has a REAL data type.  
  
fRealTagValue = TagRead("Tag3");
```

See Also

[Tag Reference and TagReadEx\(\) behavior in Cicode Expressions](#)

[Tag Functions](#)

TagReadEx

Reads the value, quality or timestamp of a particular tag from the I/O device. The variable tag needs to be defined in the Variable Tags database. Because the variable tag is specified as a string (not as a tag), you can ignore the data type of the variable.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

TagReadEx should only be used when the variable tag name is a calculation such as sAlarmExt+.Paging". For simple assignment of variables use the assignment operator. For example, MyString = MyCluster.MyAlarm.MyProperty.

If you try to read many variables at the same time, the TagReadEx() function may be slow. The offset index for array variables is checked against the size of the array.

Note: TagReadEx can only return the values of elements for those tags having "Good" quality. If the quality of a tag is not "Good", TagReadEx may return an obsolete value (due to TagRead asynchronous nature). Control the returned error/quality while using this function.

Syntax

TagReadEx(*STRING sTag* [*INT nOffset* [*INT ClusterName*]])

sTag:

The string can refer to either the alarm name and the alarm property name or the tag name with optional elements and items as it is defined by Tag Extensions [Cluster.]Tag[-.Element][.Item][[N]...]. (Refer to [Tag Extensions](#) for more information). If the element name is not specified, it will be resolved at runtime as an unqualified tag reference. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

To read a particular element in an array, you can enter the array name here, followed by an index to the element as follows:

"PLC_Array[9]"

The above example tells the function to read the 10th element in the array variable PLC_Array (remember, the address of the first element in an array is 0 (zero)).

If you enter an array offset using the *nOffset* argument, it will be added to the index value specified here. For example, TagReadEx("PLC_Array[9]", 4) will read the 14th element in PLC_Array (because [9] means the 10th element, and an offset of 4 means 4 elements after the 10th = element 14). If you want to read the value of <Valid> element in the above example,

TagReadEx ("PLC_Array.Valid.V[9]", 4)

nOffset:

The offset for an array variable. This argument is optional - if not specified, it has a default value of 0.

If you enter an array index as part of the *sTag* argument, it will be added to this offset value. For example, TagReadEx("PLC_Array[9]", 4) will read the 14th element in PLC_Array (because [9] means the 10th element, and an offset of 4 means 4 elements after the 10th = element 14).

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

The function can return:

- The tag element value (depending on the *sTag* parameter, of the I/O device variable) when the tag has "Good" quality.
- The tag element value (depending on the *sTag* parameter, of the I/O device variable) and the error 257 (can be checked by IsError() built-in function) when the value of the tag is out of the predefined range.
- Some value and an error when the tag has not "Good" quality".

Related Functions

[TagRead](#), [TagWrite](#), [IODeviceControl](#), [IODeviceInfo](#)

Example

```
STRING sStringTagValue;  
  
STRING sStringTagValueField;  
  
INT nIntTagValue;  
  
REAL fRealTagValue;  
  
// Tag1 has a STRING data type.  
  
sStringTagValue = TagReadEx("Tag1");  
  
sStringTagValueField = TagReadEx("Tag1.Field");
```

```
// Tag2 has an INTEGER data type.
```

```
nIntTagValue = TagReadEx("Tag2");
```

```
// Tag3 has a REAL data type.
```

```
fRealTagValue = TagReadEx("Tag3");
```

```
TIMESTAMP t1 = TagReadEx("Tag1.T");
```

```
TIMESTAMP t2 = TagReadEx("Tag1.VT");
```

```
TIMESTAMP t3 = TagReadEx("Tag1.Qt");
```

```
QUALITY q1 = TagReadEx("Tag1.Q");
```

See Also

[Tag Reference /TagReadEx\(\) behavior in Cicode Expressions](#)

[Tag Functions](#)

TagScaleStr

Gets the value of a tag at a specified scale. The value is returned as a formatted string using the tags format specification and (optionally) the engineering units. Use this function to write generic Cicode that will work with any type of tag.

Note: This function is being deprecated and is replaced by the [TagGetScale](#) function. If the Tag properties are updated TagScaleStr does not get the updated values whereas TagGetScale does.

Syntax

TagScaleStr(*sTag*, *Percent* [, *EngUnits*])

sTag:

The name of the tag.

Percent:

The percentage of full scale of the returned value.

EngUnits:

Optional flag to determine if the value is returned with engineering units:

- 0 - Do not return the value with engineering units
- 1 - Return the value with engineering units

Return Value

The scale of the tag (as a string).

Related Functions

[AssGetProperty](#), [AssGetScale](#), [AssInfo](#), [AssScaleStr](#), [TagGetProperty](#), [TagGetScale](#), [TagInfo](#)

Example

```
// Display the zero, 50% and full scale of the tag CV_123_PV
DspText(31,0,TagScaleStr("CV_123_PV", 0, 1));
DspText(32,0,TagScaleStr("CV_123_PV", 50, 1));
DspText(33,0,TagScaleStr("CV_123_PV", 100, 1));
```

See Also

[Tag Functions](#)

TagSetOverrideBad

Sets a quality Override element for a specified tag to Bad Non Specific.

Syntax

TagSetOverrideBad(*STRING sTag [,INT bSynch [, STRING ClusterName]]*)

sTag:

The variable tag name as a string. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

bSynch:

An optional boolean argument that specifies whether the command is synchronous (blocking) or asynchronous (non-blocking). If it is specified as synchronous (blocking) the function will wait until the write has completed and returned from the server before further code execution. This parameter is "False", or asynchronous, by default. If you specify this parameter the other parameters need to be explicitly specified.

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[TagSetOverrideQuality](#), [TagSetOverrideGood](#), [TagSetOverrideUncertain](#), [QualityIsOverride](#)

Example

```
TagSetOverrideBad("Tag1");
```

Setting the value of the OverrideMode element to anything other than 3 will overwrite the quality that has been applied to the Override element using this function. See the example below and [OverrideMode](#) for more details.

```
//The OverrideMode is switched off
```

```
Tag1.OverrideMode = 0;
```

```
//The quality of Override element is set to Bad
```

```
TagSetOverrideUncertain("Tag1");
```

```
//The OverrideMode is set to 1 and
```

```
//the Field element is copied to the Override element.
```

```
//The quality of the Override element is overwritten.
```

```
Tag1.OverrideMode = 1;
```

```
//The quality is set again to Bad
```

```
TagSetOverrideBad("Tag1");
```

See Also

[Tag Functions](#)

TagSetOverrideGood

Sets a quality Override element for a specified tag to Good Non Specific.

Syntax

TagSetOverrideGood (*STRING sTag [,INT bSynch [, STRING ClusterName]]*)

sTag:

The variable tag name as a string. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

bSynch:

An optional boolean argument that specifies whether the command is synchronous (blocking) or asynchronous (non-blocking). If it is specified as synchronous (blocking) the function will wait until the write has completed and returned from the server before further code execution. This parameter is "False", or asynchronous, by default. If you specify this parameter the other parameters need to be explicitly specified.

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[TagSetOverrideQuality](#), [TagSetOverrideUncertain](#), [TagSetOverrideBad](#), [QualityIsOverride](#)

Example

```
TagSetOverrideGood("Tag1");
```

Setting the value of the OverrideMode element to anything other than 3 will overwrite the quality that has been applied to the Override element using this function. See the example below and [OverrideMode](#) for more details.

```
//The OverrideMode is switched off  
  
Tag1.OverrideMode = 0;  
  
//The quality of Override element is set to Good  
  
TagSetOverrideGood("Tag1");  
  
//The OverrideMode is set to 1 and  
  
//the Field element is copied to the Override element.  
  
//The quality of the Override element is overwritten.  
  
Tag1.OverrideMode = 1;  
  
//The quality is set again to Good  
  
TagSetOverrideGood("Tag1");
```

See Also

[Tag Functions](#)

TagSetOverrideQuality

Sets a quality of Override element for a specified tag.

Syntax

TagSetOverrideQuality(*STRING sTag*, *QUALITY qualityNew* [*INT bSynch* [*STRING ClusterName*]])

sTag

The variable tag name as a string. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

qualityNew

The new quality for the tag's Override element.

bSynch

An optional boolean argument that specifies whether the command is synchronous (blocking) or asynchronous (non-blocking). If it is specified as synchronous (blocking) the function will wait until the write has completed and returned from the server before further code execution. This parameter is "False", or asynchronous, by default. If you specify this parameter the other parameters need to be explicitly specified.

ClusterName

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[TagSetOverrideGood](#), [TagSetOverrideUncertain](#), [TagSetOverrideBad](#), [QualityIsOverride](#)

Example

```
QUALITY q1 = QualityCreate(QUAL_UNCR);
```

```
TagSetOverrideQuality("Tag1", q1);
```

Setting the value of the OverrideMode element to anything other than 3 will overwrite the quality that has been applied to the Override element using this function. See the example below and [OverrideMode](#) for more details.

```
//The OverrideMode is switched off
```

```
Tag1.OverrideMode = 0;
```

```
//The quality of Override element is set to Uncertain
```

```
QUALITY q1 = QualityCreate(QUAL_UNCR);

TagSetOverrideQuality("Tag1", q1);

//The OverrideMode is set to 1 and

//the Field element is copied to the Override element.

//The quality of the Override element is overwritten.

Tag1.OverrideMode = 1;

//The quality is set again to Uncertain

QUALITY q1 = QualityCreate(QUAL_UNCR);

TagSetOverrideQuality("Tag1", q1);
```

See Also

[Tag Functions](#)

TagSetOverrideUncertain

Sets a quality Override element for a specified tag to Uncertain Non Specific.

Syntax

TagSetOverrideUncertain(*STRING sTag [,INT bSynch [, STRING ClusterName]]*)

sTag:

The variable tag name as a string. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

bSynch:

An optional boolean argument that specifies whether the command is synchronous (blocking) or asynchronous (non-blocking). If it is specified as synchronous (blocking) the function will wait until the write has completed and returned from the server before further code execution. This parameter is "False", or asynchronous, by default. If you specify this parameter the other parameters need to be explicitly specified.

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[TagSetOverrideQuality](#), [TagSetOverrideGood](#), [TagSetOverrideBad](#), [QualityIsOverride](#)

Example

```
TagSetOverrideUncertain("Tag1");
```

Setting the value of the OverrideMode element to anything other than 3 will overwrite the quality that has been applied to the Override element using this function. See the example below and [OverrideMode](#) for more details.

```
//The OverrideMode is switched off
```

```
Tag1.OverrideMode = 0;
```

```
//The quality of Override element is set to Uncertain
```

```
TagSetOverrideUncertain("Tag1");
```

```
//The OverrideMode is set to 1 and
```

```
//the Field element is copied to the Override element.
```

```
//The quality of the Override element is overwritten.
```

```
Tag1.OverrideMode = 1;
```

```
//The quality is set again to Uncertain
```

```
TagSetOverrideUncertain("Tag1");
```

See Also

[Tag Functions](#)

TagSubscribe

Subscribes a tag so that Cicode functions can be called when a tag's value changes. The subscription checks each poll period whether the tag has changed value and if it has, the specified callback function is called. This avoids continuously polling a tag value to monitor changes. To add a function callback to the subscription, use the optional parameter in this command or the [SubscriptionAddCallback](#) function.

Multiple subscriptions are possible to the same tag. Each new subscription returns a new subscription handle. Multiple callbacks are possible to the same subscription.

To unsubscribe a tag use the [TagUnsubscribe](#) function.

Syntax

TagSubscribe(*sTagName* [, *iPollTime*] [, *sScaleMode*] [, *dDeadband*] [, *sCallback*] [, *bLightweight*])

sTagName

String representing the tag or tag element to subscribe to in the form of "<cluster_name>.<tag_name>.<element name>". If the element name is not specified, it will be resolved at runtime as an unqualified tag reference.

iPollTime

Optional integer representing the Datasource Poll time in milliseconds (default 250).

sScaleMode

Optional string stating the mode to subscribe. Supported modes are: Raw, Eng, Percent. Default is "Eng".

dDeadband

Optional real value specifying the percentage of the variable tag's engineering range that a tag needs to change by for an update to be sent through the system. Default value is -1.0, indicating the deadband specified by the tag definition is to be used.

sCallback

Optional string stating the name of a function to call when the value is updated. If an empty string is specified, no handler is registered. Default value is "" (empty string).

The function should have the structure:

```
FUNCTION evtHandler(INT subsHandle)
    ...
END
```

Where *subsHandle* is the subscription that raised the event.

bLightweight

This optional boolean argument indicates whether or not subscription updates use a "lightweight" version of the tag value that does not include a quality timestamp or a value timestamp.

If not used, this option is set to 1 which means lightweight tag values will be used by default. For a client to retrieve quality and value timestamps for a tag, you should explicitly specify that a full tag value is required by setting this option to 0.

Return Value

Integer representing the subscription handle that can be used to read values, hook to events or unsubscribe. If unsuccessful, -1 is returned and an [error](#) is set. Even though a subscription handle is returned immediately, it can't be used to get attributes until the subscription has been confirmed as this is an asynchronous Cicode function call. The typical Cicode error is 423 when a subscription handled is used too soon. We recommend the use of a callback function or the direct use of the tag extension, e.g <tag>.VT

Related Functions

[TagUnsubscribe](#), [SubscriptionAddCallback](#), [SubscriptionGetAttribute](#), [SubscriptionRemoveCallback](#)

Example

The following example subscribes the tag "Conveyor1" in order to manually poll for attributes of the tag.

```
...
// Get the last changed value, quality and timestamp for the tag every 1s
...
// LOOP START

SleepMs(1000);

ErrSet(1);
convValue = SubscriptionGetAttribute(subsHandle, "Value");

IF IsError() = 0

    convQual = SubscriptionGetAttribute(subsHandle, "ValueQuality");

    convTime = SubscriptionGetAttribute(subsHandle, "ValueTimestamp");
```

```
// Format and use data here

END

// LOOP END
...
// Unsubscribe the tag TagUnsubscribe(subsHandle);
```

The following example subscribes the "conveyor1" tag as a percentage and polls it every 100ms to check for changes. When the value changes the functions OnValueChanged and ValChanged2 are called. This is the recommended way to do polling of special variables:

```
...
INT subsHandle = TagSubscribe("Conveyor1", 100, "Percent",
"OnValueChanged");
...
// Later on if no callback was registered initially, a new one can be added..
SubscriptionAddCallBack(subsHandle, "ValChanged2");
...
Function OnValueChanged(INT handle)
STRING sTag;

sTag = SubscriptionGetAttribute(handle, "FullName");           // If the name is needed
subsVal = SubscriptionGetAttribute(handle, "Value");
subsQual = SubscriptionGetAttribute(handle, "ValueQuality");
...
END
...
Function ValChanged2(INT handle)
STRING sTag;

sTag = SubscriptionGetAttribute(handle, "FullName");           // If the name is needed
subsVal = SubscriptionGetAttribute(handle, "Value");
subsTime = SubscriptionGetAttribute(handle, "ValueTimestamp");
...
END
...
// Remove all callbacks and unsubscribe
TagUnsubscribe(subsHandle);
```

See Also

[Tag Functions](#)

TagUnsubscribe

Unsubscribes the tag subscription specified by the integer subscription handle that was returned from the [TagSubscribe](#) function. This function also removes any callbacks that are associated with the subscription.

Syntax

TagUnsubscribe(*iHandle*)

iHandle

Integer handle of the subscription to unsubscribe.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TagSubscribe](#), [SubscriptionAddCallback](#), [SubscriptionGetAttribute](#), [SubscriptionRemoveCallback](#)

See Also

[Tag Functions](#)

TagWrite

Writes to an I/O device variable by specifying the variable tag name or the variable tag name and the name of the requested element having read/write access. The variable tag needs to be defined in the Variable Tags database.

Note: For this function to be successful a user needs to be logged in.

This function completes asynchronously to the caller. An error occurs if the tag does not exist or if a write request could not be sent. This function does not test whether the write succeeded. In cases where the write does not succeed, TagWrite does not return a driver error code. For important write operations, perform a [TagReadEx](#) to confirm that the write took place.

TagWrite should only be used when the variable tag name is a calculation such as sAlarmExt+.Paging". For assignment of variables use the assignment operator. For example, MyCluster.MyAlarm.MyProperty = MyString.

Note: When using this function and parameter [Code]ScaleCheck is set to 1, the attempt to write an out-of-range value to a device will not occur. No hardware alarm will be generated. This function checks a value before writing it to a PLC.

Syntax

TagWrite(*STRING sTag*, *STRING sValue* [, *INT nOffset*] [, *INT bSynch*] [, *STRING ClusterName*])

sTag:

The string can refer to either: the variable tag name, the alarm name and the alarm property name, the tag name and the tag element name (Refer to [Tag Extensions](#) for more information). If the element name is not specified, the writing will be performed to the Field VQT element. The name of the tag can be prefixed by the name of the cluster that is "ClusterName.Tag".

sValue:

The value to be written to the I/O device variable. The value is specified as a string, however if an integer or real is used the compiler will convert it to a string. The function converts the string into the correct format, and then writes it to the variable.

To write to a particular element in an array, you can enter the array name here, followed by an index to the element as follows:

"PLC_Array[9]"

The above example tells the function to write to the 10th element in the array variable PLC_Array (remember, the address of the first element in an array is 0 (zero)).

If you enter an array offset using the *nOffset* argument, it will be added to the index value. See example below.

nOffset:

Optional offset for an array variable. Default is 0.

Note: If you enter an array index as part of the *sValue* argument, it will be added to this offset value. For example, TagWrite("PLC_Array[9]", 24, 4) will set the 14th element in PLC_Array to 24 (because [9] means the 10th element, and an offset of 4 means 4 elements after the 10th = element 14).

bSynch:

An optional boolean argument that specifies whether the command is synchronous (blocking) or asynchronous (non-blocking). If it is specified as synchronous (blocking) the function will wait until the write has completed and returned from the server before further code execution. This parameter is "False", or asynchronous, by default. If you specify this parameter the rest of the parameters need to be explicitly specified, including *nOffset* which should be set as 0 if the tag is not an array tag.

ClusterName:

Specifies the name of the cluster in which the Tag resides. The argument is enclosed in quotation marks.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TagRead](#), [TagReadEx](#), [IODeviceControl](#), [IODeviceInfo](#)

Example

```
TagWrite("PLC_VAR1", 123);
TagWrite("PLC_VAR1", 123, 0, TRUE); ! Write to PLC variable
! and block until write is successful.
TagWrite("PLC_VAR_STR", "string data to write");
TagWrite("PLC_ARRAY", 42, 3); ! Write to element 4 in array
TagWrite("PLC_Array[9]", 2); ! Write to element 12 in array
TagWrite ("Tag1", "123");
TagWrite("Tag1.Field", "123");
```

See Also

[Tag Functions](#)

TagWriteEventQue

Opens the tag write event queue. The TagWriteEventQue is a queue of data containing details of tag value changes initiated by the process. To read events from the queue, use the QueRead() or QuePeek() functions. The queue contains timestamp, tagname and value data for each change event.

This queue is enabled by the corresponding INI parameter [General]TagWriteEventQue. Writes are logged to the queue for all tags whose IODevices have their Log Write parameter enabled.

Syntax

TagWriteEventQue()

Return Value

The handle of the tag write event queue, or -1 if the queue cannot be opened.

Related Functions

[AlarmEventQue](#), [QueRead](#), [QuePeek](#)

Example

To enable tag logging:

1. On the client, set the INI param [General]TagWriteEventQue = 1.
2. Enable the IODevices that you want to be logged by setting their Log Write parameters to TRUE. See I/O Devices Properties.

To read the queue you need to create a Cicode function. For example:

```
FUNCTION
checkWrite()
    STRING sTagNameAndValue = "";
    INT nDateTime = 0;
    INT hQue = TagWriteEventQue();
    IF hQue = -1 THEN
        RETURN;
    END
    WHILE 1 DO
        QueRead(hQue, nDateTime, sTagNameAndValue, 1);
        Message("Value written", sTagNameAndValue, 64);
    END
END
```

Where:

- *nDateTime* is the timestamp of the tag write.
- *sTagNameAndValue* is the tagname and value written to the queue.

When the function is run, successful writes to tags on the IODevice will show as a message "Value written <tagname> <value>".

Note: The TagWriteEventQue is enabled on each process and will only log the data changes initiated by this process. By combining this data with the current user and machine name CitectSCADA can generate a user activity log with respect to setting data within the control system. This functionality can also be combined with CitectSCADA Reports to augment the detail of the historian data.

See Also

[Tag Functions](#)

Chapter: 54 Task Functions

Task functions support advanced multi-tasking operations in Cicode, handling queues, semaphores, messages, and other process functions. The task functions control the transfer of data between different Cicode tasks and across the network to different computers (by remote procedure calls).

Task Functions

Following are functions relating to Tasks:

CodeSetMode	Sets the execution mode of a Cicode task.
Enter-CriticalSection	Requests permission for the current thread to have access to a critical shared resource (critical section). If the critical section is already being accessed, the thread will be granted access when it is free.
Halt	Halts the current Cicode task.
Leave-CriticalSection	Relinquishes the current thread's ownership of a critical shared resource (critical section).
MsgBrdcst	Broadcasts a message.
MsgClose	Closes a message.
MsgGetCurr	Gets the handle of the message that called the current report or remote procedure.
MsgOpen	Opens a message session with a CitectSCADA server or client.
MsgRead	Reads a message from a session.
MsgRPC	Calls a remote procedure on another CitectSCADA computer.
MsgState	Verifies the status of a message session.
MsgWrite	Writes a message to a session.

QueClose	Closes a queue.
QueLength	Gets the current length of a queue.
QueOpen	Creates or opens a queue.
QuePeek	Searches a queue for a queue element.
QueRead	Reads elements from a queue.
QueWrite	Writes elements to a queue.
ReRead	ReRead is deprecated in this version.
SemClose	Closes a semaphore.
SemOpen	Creates or opens a semaphore.
SemSignal	Signals a semaphore.
SemWait	Waits on a semaphore.
ServerRPC	Calls a remote procedure on a Citect server.
Sleep	Suspends the current Cicode task for a specified time.
SleepMS	Suspends the current Cicode task for a specified time (in milliseconds).
TaskCall	Calls a Cicode function by specifying the function name and providing an arguments string.
TaskCluster	Gets the name of the cluster context in which the current task is executing.
TaskGetSignal	Retrieves a value that indicates the stop signal for a specific task.
TaskHnd	Gets the handle of a particular task.
TaskKill	Kills a running task.
TaskNew	Creates a new task.
TaskNewEx	Creates a new task with a subscription rate.
TaskResume	Resumes a task.

TaskSetSignal	Ends a task by manually triggering its stop signal.
-------------------------------	---

TaskSuspend	Suspends a task.
-----------------------------	------------------

See Also

[Functions Reference](#)

CodeSetMode

Sets various execution modes for Cicode tasks in the current thread. Using this function, you can specify whether to:

- Write to a local image of an I/O device.
- Check if a variable is within range before writing it to the I/O device.
- Echo error messages to the operator.
- Check the case of string data in Cicode.

Syntax

CodeSetMode(*Type*, *Value*)

Type:

Type of mode:

0 - Write to a local image of an I/O device. If you set Value to 1, this mode is enabled, and Cicode writes its local memory image of the I/O device whenever you write to the I/O device. (Cicode assumes that most writes to the I/O device will be done immediately).

This local image might produce problems, or you might want to verify that the data was actually written to the I/O device. If you set Value to 0 (zero), this check is disabled, and Cicode does not write to the local memory image.

1 - Check if a variable is within range before writing it to the I/O device. If you set Value to 1, this mode is enabled. When a variable tag is modified, Cicode checks the new value of the variable against the Scales specified in the Variable Tags database. If the value of the variable is out of scale, Cicode generates a hardware error, and does not write to the I/O device.

If you set Value to 0 (zero), this check is disabled. Cicode writes the variable to the I/O device without checking if its value is within range.

2 - Echo error messages to the operator. If you set Value to 1, this mode is enabled. When a simple user error occurs (for example, if the PageDisplay() function is passed a bad page name), Cicode displays an error message at the Error AN, and returns an error code from the function.

If you set Value to 0 (zero), the echo is disabled. Cicode does not display the error message, and the output of the DspError() function is stopped.

3 - Ignore the case of string data in the current thread of Cicode. If you set Value to 1, this mode is enabled, and CitectSCADA will ignore case in string data. For example, CitectSCADA will equate "Hello" to "HELLO".

If you set Value to 0 (zero), CitectSCADA will be case sensitive to string data. Case sensitivity is used when a string comparison operation is performed. For example:

```
IF sStr1 = sStr2 THEN
```

(You can also make CitectSCADA case sensitive to strings in all of your Cicode, using the [Code]IgnoreCase parameter.)

Value:

The value of the mode:

0 - Disable

1 - Enable

Return Value

-1 if there is an [error](#), otherwise the last value of the mode.

Example

```
! disable local image write  
CodeSetMode(0, 0);
```

See Also

[Task Functions](#)

EnterCriticalSection

Requests permission for the current thread to have access to a critical section (shared critical resource). If the critical section is already being accessed by another thread (using the EnterCriticalSection() function), the current thread will be granted access when the other thread relinquishes ownership using the LeaveCriticalSection() function.

Once a thread has ownership of a critical section, it can access the same section repeatedly (using the EnterCriticalSection() function each time). Remember, however, that LeaveCriticalSection() needs to be called once for each EnterCriticalSection() used.

Note: This function is process-based, not computer-based, and so is only effective for threads within the same process. Any threads attempting to gain access to this critical section will be blocked until the thread relinquishes ownership. This function will have no effect on threads running within other processes.

⚠ WARNING

UNINTENDED EQUIPMENT OPERATION

Do not attempt to access a critical section that is already in use by another thread. This will cause the thread to be blocked until the critical section is relinquished.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Syntax

EnterCriticalSection(*sName*)

sName:

The name of the critical section. The name needs to be entered in quotation marks.

Return Value

This function does not return a value.

Related Functions

[LeaveCriticalSection](#)

Example

```
/* Request access to critical section, execute code and relinquish
ownership of critical section. */
FUNCTION
MyCriticalSection()
    EnterCriticalSection("MyCritical");
    // critical code is placed here
    LeaveCriticalSection("MyCritical");
END
```

See Also

[Task Functions](#)

Halt

Stops the execution of the current Cicode task and returns to CitectSCADA. This function does not affect any other Cicode tasks that are running.

Use this function to stop execution in nested function calls. When Halt() is called, Cicode returns to CitectSCADA and does not execute any return function calls.

Syntax

Halt()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[Assert](#), [TaskKill](#)

Example

```
INT
FUNCTION
MyFunc(INT Arg)
    IF Arg<0 THEN
        Prompt("Invalid Arg");
        Halt();
    END
    ...
END
```

See Also

[Task Functions](#)

LeaveCriticalSection

Relinquishes the current thread's ownership of a critical section (shared critical resource). Once ownership is relinquished, access to the critical section is available to the next thread that requests it (using the EnterCriticalSection() function). If a thread has been waiting for access, it will be granted at this point.

LeaveCriticalSection() needs to be called once for each EnterCriticalSection() used.

Note: This function is process-based, not computer-based, and so will only prevent access to a critical section within a single process. This function only works between Cicode tasks within the same process.

Syntax

LeaveCriticalSection(*sName*)

sName:

The name of the critical section. The name needs to be entered in quotation marks.

Return Value

This function does not return a value.

Related Functions

[EnterCriticalSection](#)

Example

```
/* Request access to critical section, execute code and relinquish
ownership of critical section. */
FUNCTION
MyCriticalSection()
    EnterCriticalSection("MyCritical");
    // critical code is placed here
    LeaveCriticalSection("MyCritical");
END
```

See Also

[Task Functions](#)

MsgBrdcst

Broadcasts a message to all the clients of a server. You should call this function only on a CitectSCADA server. The message is only received by clients that have a current message session (opened with the MsgOpen() function).

Syntax

MsgBrdcst(*Name*, *Type*, *Str*)

Name:

The name of the CitectSCADA server.

Type:

The message number.

Str:

The message text.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MsgOpen](#), [MsgClose](#), [MsgRead](#), [MsgWrite](#), [MsgRPC](#)

Example

```
! Send a message to all alarm clients.  
MsgBrdcst("Alarm", 0, "Alarm Occurred");
```

See Also

[Task Functions](#)

MsgClose

Closes a message. After the message is closed, the message post function (the callback function specified in the `MsgOpen()` function) is not called if a message is received. When the server side is closed, all clients are closed. When the client side is closed, only the specified client is closed.

Syntax

MsgClose(*Name*, *hMsg*)

Name:

The name of the CitectSCADA server.

hMsg:

The message handle, returned from the `MsgOpen()` function. The message handle identifies the table where all data on the associated message is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MsgOpen](#), [MsgRead](#), [MsgWrite](#), [MsgRPC](#)

Example

```
MsgClose("Alarm", hMsg);
```

See Also[Task Functions](#)**MsgGetCurr**

Gets the handle of the client message that called the report or remote procedure that is currently running. You can call this function only in a report or a remote procedure call.

If the report was called by a client, this function returns that client message handle. The report can then send a message back to the client. If a function was called remotely by `MsgRPC()`, this function returns the message handle for the remote client.

Syntax[MsgGetCurr\(\)](#)**Return Value**

The handle for the client message. The message handle identifies the table where all data on the associated message is stored. The function returns -1 if no client called the report or function.

Related Functions[MsgOpen](#), [MsgRPC](#)**Example**

```
! Send message back to the client.
hMsg=MsgGetCurr();
IF hMsg<>-1 THEN
    MsgRPC(hMsg,"Prompt","^"Hello Client from Report Server^"",1);
END
```

See Also[Task Functions](#)**MsgOpen**

Opens a message session with a CitectSCADA server. You can specify a message post function - a callback function that is automatically called when a message arrives. In this function you can call MsgRead() to get the message, and perform other tasks common to your message sessions. You can then call MsgWrite() to send a message back to the caller, MsgRPC() to call a procedure on the caller, and so on.

Note: For this function to be successful a user needs to be logged in.

Syntax

MsgOpen(*Name*, *Mode*, *Fn* [, *ClusterName*])

You should use MsgState() to check the return value of MsgOpen(). The message post function is set effectively only if MsgState() returns 1, which means the message session is online.

You can open a client-server message session or a session between redundant servers. This function does not create extra network sessions; it uses CitectSCADA's existing sessions, so you create sessions to the Alarm Server, Report Server, Trend Server, or a named I/O Server.

Name:

The name of the server to open, either:

- For Mode 0, 1, or 3: "Alarm", "Report", "Trend", or the name of an I/O Server.
- For Mode 2: The default computer name, as set in the [Lan]Node parameter.

Mode:

The mode of the message session to open:

0 - Open the client side.

1 - Open the server side.

2 - Open a session from a server to the default computer name. Set Name to the computer name of the computer, as defined by the [LAN]Node parameter.

3 - Open a message session between redundant servers. (Clients cannot tell which server they are connected to or if something has changed on the server. Changes should be performed on the redundant server as well.)

4 - Open a message session from the calling process to the client process. The Name and Fn are ignored in this mode. The message session opened in this mode does not need to call MsgClose.

Fn:

The message post function, that is a callback function for the message event. Set Fn to 0 if no event callback function is required.

ClusterName:

The name of the cluster the server being communicated with belongs to, this is used when mode is 0, 1 or 3. This is not required if the client is connected to only one cluster containing a server of the type set in the name parameter.

Return Value

The message handle, or -1 if the session cannot be opened. The message handle identifies the table where data on the associated message is stored.

Related Functions

[MsgClose](#), [MsgRead](#), [MsgWrite](#), [MsgRPC](#)

Example

```

INT hClient = -1;
// Open message session on the client, connecting using the
//existing message session to the current Alarm server
FUNCTION
MsgClientOpen()
    INT nState;
    hClient = MsgOpen("Alarm", 0, MsgPostClient);
    IF hClient <> -1 THEN
        nState = MsgState(hClient);
        SELECT CASE nState
        CASE 1
            Prompt("Message session is online!");
            //Send a message to the server
            MsgWrite(hClient, 1, "MyMessage");
        CASE -1
            Prompt("Message session handle is invalid!");
        CASE 0
            Prompt("Message session is offline!");
        CASE 2
            Prompt("Message session is connecting!");
        CASE 3
            Prompt("Message session is disconnecting!");
        CASE ELSE
            Prompt("Message session has unknown status!");
        END SELECT
    ELSE
        Prompt("Message session could not be opened!");
    END
END
// This function is called when the client receives a message from
//the server
INT

```

```
FUNCTION
MsgPostClient()
    INT Type;
    STRING Str;
    MsgRead(Type,Str);
    Prompt("Client received message: Type = "+Type:###+" Str = "+Str);
    RETURN 0;
END
```

See Also

[Task Functions](#)

MsgRead

Reads a message from a message session. You can call this function only in a message post function (the callback function specified in the `MsgOpen()` function), to read the current message.

The *Type* and *Str* variables of this function return the message number and the text of the message. The return value of this function is the message handle (allowing a response to be sent back if required).

you need to open the message session using the `MsgOpen()` function, to enable the callback function.

Syntax

MsgRead(*Type*, *Str*)

Type:

The message number.

Str:

The message text.

Return Value

The message handle of the message being read.

Related Functions

[MsgOpen](#), [MsgClose](#), [MsgWrite](#), [MsgRPC](#)

Example

```
/* This function will read a message from the session and if
```

```
Type=1, will display the string as a prompt. If Type=2 then the
speaker beeps and an acknowledgment is sent back to the caller. */
INT
FUNCTION
MsgPostClient()
    INT Type;
    STRING Str;
    INT hMsg;
    hMsg=MsgRead(Type,Str);
    IF Type=1 THEN
        Prompt("Message"+Str);
    ELSE
        IF Type=2 THEN
            Beep();
            MsgWrite(hMsg,2,"DONE");
        END
    END
END
```

See Also

[Task Functions](#)

MsgRPC

Calls a remote procedure on another CitectSCADA computer. You can call any of the built-in Cicode functions remotely, or your own functions. You pass the *Name* of the function as a string, not as the function tag, and pass all the arguments for that function in *Arg*.

You can call the function in synchronous or asynchronous Mode. In synchronous mode, *MsgRPC()* does not return until the remote function is called and the result is returned. In asynchronous mode, *MsgRPC()* returns before the function is called, and the result cannot be returned.

Syntax

MsgRPC(*hMsg*, *Name*, *Arg*, *Mode*)

hMsg:

The message handle, returned from the *MsgOpen()* function. The message handle identifies the table where all data on the associated message is stored.

Name:

The name of the function to call remotely, as a string.

If this function returns an error, you should confirm that the name you have used is not a label instead of the actual function name. Some functions are aliased using a label, for example, the function *_AlarmGetFieldRec* is defined in the labels database as "AlarmGetFieldRec". In this case, only "*_AlarmGetFieldRec*" should be passed to *MsgRPC*.

Arg:

The arguments to pass to the function, separated by commas (,). Enclose string arguments in quotes "" and use the string escape character (^) around strings enclosed within a string. If you do not enclose the string in quotes, then the string is only the first tag found.

Mode:

The mode of the call:

0 - Blocking mode - synchronous.

1 - Non-blocking mode - asynchronous.

Return Value

The result of the remote function call (as a string). If the function is called in asynchronous mode the result of the remote function cannot be returned, so an empty string is returned.

Related Functions

[MsgOpen](#), [MsgClose](#), [MsgRead](#), [MsgWrite](#)

Example

```
! Call remote procedure, call MyRPC() on server. Wait for result
Str=MsgRPC(hMsg,"MyRPC","Data",0);
! Call remote procedure, pass two strings. Don't wait for call to complete.
! be careful of your string delimiters as shown.
MsgRPC(hMsg,"MyStrFn","^"First string^",^"Second string^"",1);
! Call remote procedure, pass Cicode string. Don't wait for call to complete.
STRING sMessage = "this is a message";
MsgRPC(hMsg,"MyStrFn","^" + sMessage + "^",1);
! These functions could be used to acknowledge an alarm by record
from any CitectSCADA Client on the network.
! The AlmAck() function is initialized by the Control Client
(Don't forget that servers are also Control Clients.)
! The Alarm tag is passed into the function as a string and a
message is sent to the Alarms Server to initialize
! the AlmAckMsg() function.
FUNCTION
AlmAck(String AlmTag)
    INT hAlarm1;
    hAlarm1 = MsgOpen("Alarm", 0, 0);
    MsgRPC(hAlarm1,"AlmAckMsg",AlmTag,1);
    MsgClose("Alarm", hAlarm1);
END
! The AlmAckMsg() function is executed on the Alarms Server that
the client is connected to. This could be
! either the primary or standby Alarms Server. The function
performs the alarm acknowledge.
FUNCTION
```

```

AlmAckMsg(String AlmTag)
    AlarmAckRec(AlarmFirstTagRec(AlmTag,"","",""));
END

```

See Also[Task Functions](#)**MsgState**

Verifies the status of a message session. Use `MsgState()` to check the return value of `MsgOpen()`. A message post function is set effectively only if `MsgState()` returns 1, which means the message session is online.

Syntax**MsgState(*hMsg*)***hMsg*:

The message handle, returned from the `MsgOpen()` function. The message handle identifies the table where all data on the associated message is stored.

Return Value

This function has the following possible return values:

- -1 if the message session handle is invalid
- 0 if the message session is offline
- 1 if the message session is online
- 2 if the message session is connecting
- 3 if the message session is disconnecting.

Related Functions[MsgOpen](#)**Example**

```

INT hClient = -1;

// Open message session on the client, connecting using the
existing
// message session to the current Alarm server
FUNCTION
MsgClientOpen()
    INT nState;

```

```

hClient = MsgOpen("Alarm", 0, MsgPostClient);
IF hClient <> -1 THEN
    nState = MsgState(hClient);
    SELECT CASE nState
    CASE 1
        Prompt("Message session is online!");
        //Send a message to the server
        MsgWrite(hClient, 1, "MyMessage");
    CASE -1
        Prompt("Message session handle is invalid!");
    CASE 0
        Prompt("Message session is offline!");
    CASE 2
        Prompt("Message session is connecting!");
    CASE 3
        Prompt("Message session is disconnecting!");
    CASE ELSE
        Prompt("Message session has unknown status!");
    END SELECT
ELSE
    Prompt("Message session could not be opened!");
END
END

```

See Also[Task Functions](#)**MsgWrite**

Writes a message to a message session. The message is sent to the remote computer's callback function and can be read by calling `MsgRead()`. If the remote computer has not opened the session, this message is disregarded.

This function returns immediately after passing the message to CitectSCADA. CitectSCADA sends the message over the LAN in the background.

you need to first open the message session using the `MsgOpen()` function, to obtain the message handle.

Syntax**MsgWrite**(*hMsg*, *Type*, *Str*)*hMsg*:

The message handle, returned from the `MsgOpen()` function. The message handle identifies the table where all data on the associated message is stored.

Type:

The integer message data, that is the message number.

Str:

The message text.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[MsgRead](#), [MsgOpen](#)

Example

```
MsgWrite (hMsg, 10, "MyMsg");
```

See Also

[Task Functions](#)

QueClose

Closes a queue opened with the QueOpen() function. All data is flushed from the queue.

If a Cicode task is waiting on the QueRead() function, it returns with a "queue empty" status. You should close all queues when they are no longer required, because they consume memory. At shutdown, CitectSCADA closes all open queues.

Syntax

QueClose(*hQue*)

hQue:

The queue handle, returned from the QueOpen() function. The queue handle identifies the table where all data on the associated queue is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[QueLength](#), [QueOpen](#), [QueRead](#), [QueWrite](#), [QuePeek](#)

Example

```
hQue=QueOpen ("MyQue", 1);
...
```

```
QueClose (hQue) ;
```

See Also

[Task Functions](#)

QueLength

Gets the current length of the queue.

Syntax

QueLength(*hQue*)

hQue:

The queue handle, returned from the QueOpen() function. The queue handle identifies the table where all data on the associated queue is stored.

Return Value

The current length of the queue. If the queue is closed then 0 is returned.

Related Functions

[QueClose](#), [QueOpen](#), [QueRead](#), [QueWrite](#), [QuePeek](#)

Example

```
Length=QueLength (hQue) ;
```

See Also

[Task Functions](#)

QueOpen

Open a queue for reading and writing data elements. Use this function to create a new queue or open an existing queue. Use queues for sending data from one task to another or for other buffering operations.

Syntax

QueOpen(*Name*, *Mode*)

Name:

The name of the queue.

Mode:

The mode of the queue open:

0 - Open existing queue.

1 - Create new queue.

2 - Attempts to open an existing queue. If the queue does not exist, it will create it.

Return Value

The queue handle, or -1 if the queue cannot be opened. The queue handle identifies the table where all data on the associated queue is stored.

Related Functions

[QueClose](#), [QueLength](#), [QueRead](#), [QueWrite](#), [QuePeek](#)

Example

```
! Create a queue.
hQue=QueOpen("MyQue",1);
! Write data into the queue.
QueWrite(hQue,1,"Quetext");
QueWrite(hQue,1,"Moretext");
! Read back data from the queue.
QueRead(hQue,Type,Str,0);
```

See Also

[Task Functions](#)

QuePeek

Searches a queue for a queue element. You can search for the element by specifying a string, an integer, or both. You can remove the element from the queue by adding 8 to the *Mode*.

Note: This function may modify the arguments *Type* and *Str* depending on the Mode. Therefore, these arguments need to be variables. You should consider that they may be impacted by the setting for Mode when calling the function.

Syntax

QuePeek(*hQue*, *Type*, *Str*, *Mode*)

hQue:

The queue handle, returned from the QueOpen() function. The queue handle identifies the table where all data on the associated queue is stored.

Type:

The number to search for (if using the search mode for a matching number). If you are using a matching string mode, the number found is returned in Type.

Str:

The string to search for (if using the search mode for a matching string). If you are using a matching number mode, the string found is returned in Str.

Mode:

The mode of the search:

- 1 - Search for a matching string.
- 2 - Search for a matching number.
- 4 - Search for a matching string and use a case-sensitive search.
- 8 - If the element is found, remove it from the queue.
- 16 Search the queue, in order, for the element at the offset specified by Type.

Use mode 16 when you know the location of the element you want. For example if you set Type = 0, QuePeek will return the first element in the queue, type = 2, will return the 3rd element in the queue, etc. If you specify an offset which is greater than the length of the queue, the "queue empty" error (296) is returned.

You can extend the search by adding modes. For example, set Mode to 3 to search for a matching string and matching number, or set Mode to 11 to also remove the string and number from the queue.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[QueClose](#), [QueLength](#), [QueOpen](#), [QueRead](#), [QueWrite](#)

Example

```
STRING Str;
INT Type;
! search for 'mystring' in queue, don't remove if found
Str = "mystring";
status=QuePeek(hQue,Type,Str,1);
IF Status = 0 THEN
    ! Now use found Type
    ...

```

END

See Also[Task Functions](#)**QueRead**

Reads data from a queue, starting from the head of the queue. Data is returned in the same order as it was written onto the queue and is removed from the queue when read. If the *Mode* is 0 (non-blocking) and the queue is empty, the function returns with an error. If the *Mode* is 1 (blocking) the function does not return until another Cicode task writes data onto the queue.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

QueRead(*hQue*, *Type*, *Str*, *Mode*)

hQue:

The queue handle, returned from the QueOpen() function. The queue handle identifies the table where all data on the associated queue is stored.

Type:

The integer variable to read from the queue (written to the queue as Type by the QueWrite() function).

Str:

The string variable to read from the queue (written to the queue as Str by the QueWrite() function).

Mode:

The mode of the read:

0 - Non-blocking.

1 - Wait for element.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[QueClose](#), [QueLength](#), [QueOpen](#), [QueWrite](#), [QuePeek](#)

Example

```
Status=QueRead(hQue,Type,Str,0);
IF Status = 0 THEN
    ! Now use Type and Str.
    ...
END
```

See Also

[Task Functions](#)

QueWrite

Writes an integer and string onto the end of a queue. The integer and string have no meaning to the queue system, they are just passed from QueWrite() to QueRead(). Queue data is written to the end of the queue. When the data is later read from the queue, it is returned on a first-in-first-out basis.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

QueWrite(*hQue*, *Type*, *Str*)

hQue:

The queue handle, returned from the QueOpen() function. The queue handle identifies the table where all data on the associated queue is stored.

Type:

The integer to put into the queue.

Str:

The string to put into the queue.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[QueClose](#), [QueLength](#), [QueOpen](#), [QueRead](#), [QuePeek](#)

Example

```
QueWrite(hQue,2,"Hello there");
QueWrite(hQue,4,"Help");
```

See Also[Task Functions](#)**ReRead**

ReRead is deprecated in this version of CitectSCADA.

Tags are now subscribed at the start of a function and updated tag values are sent to the subscribing function. Tag subscriptions are made at the update rate of:

- the graphics page if called from a page
- the default subscription rate as determined by [Code]TimeData if called from Cicode (default 250ms)
- the update rate requested of a task created using [TaskNewEx](#)
- the update rate requested of a subscription created using [TagSubscribe](#)

You will want to verify that the subscription update rate matches the requirements of your system.

After removing ReRead from looping code you may need to extend the period of the [Sleep](#) function. This is to replace the pause ReRead created while it read all the tag values.

Syntax**ReRead(*Mode*)***Mode:*

The mode of the read:

0 - Read only if data is stale.

1 - Read anyway.

Return Value

No value (void).

See Also[Task Functions](#)**SemClose**

Closes a semaphore opened with SemOpen(). You should close all semaphores when they are no longer required, because they consume memory. If any Cicode tasks are waiting on this semaphore, the tasks are released with an error.

Note: This function is process-based, not computer-based, and so will only prevent access to an important section within a single process. This function only works between Cicode tasks within the same process.

Syntax

SemClose(*hSem*)

hSem:

The semaphore handle, returned from the SemOpen() function. The semaphore handle identifies the table where all data on the associated semaphore is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[SemOpen](#), [SemSignal](#), [SemWait](#)

Example

```
SemClose (hSem) ;
```

See Also

[Task Functions](#)

SemOpen

Opens a semaphore for access control. When the semaphore is opened, it is initially signalled. Use a semaphore for controlling access to a restricted device, for example, to stop another Cicode task accessing a device while it is in use. You might require semaphores for some Cicode operations, because they can access a device that is critical. (Cicode is a multi-tasking system.)

Note: This function is process-based, not computer-based, and so will only prevent access to a critical section within a single process. This function only works between Cicode tasks within the same process.

Syntax

SemOpen(*Name*, *Mode*)

Name:

The name of the semaphore.

Mode:

The mode of the open:

0 - Open existing semaphore.

1 - Create new semaphore.

2 - Attempts to open an existing semaphore. If the semaphore does not exist, it will create it.

Return Value

The semaphore handle, or -1 if the semaphore was not opened successfully. The semaphore handle identifies the table where all data on the associated semaphore is stored.

Related Functions

[SemClose](#), [SemSignal](#), [SemWait](#)

Example

```
hSem=SemOpen ("MySem", 1);
```

See Also

[Task Functions](#)

SemSignal

Signals a semaphore. If several Cicode tasks are waiting on this semaphore, the first task is released. This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Note: This function is process-based, not computer-based, and so will only prevent access to a critical section within a single process. This function only works between Cicode tasks within the same process.

Syntax

SemSignal(*hSem*)

hSem:

The semaphore handle, returned from the SemOpen() function. The semaphore handle identifies the table where all data on the associated semaphore is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[SemClose](#), [SemOpen](#), [SemWait](#)

Example

```
SemSignal (hSem) ;
```

See Also

[Task Functions](#)

SemWait

Waits on a semaphore to be signalled. This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Note: This function is process-based, not computer-based, and so will only prevent access to a critical section within a single process. This function only works between Cicode tasks within the same process.

Syntax

SemWait(*hSem*, *Timeout*)

hSem:

The semaphore handle, returned from the SemOpen() function. The semaphore handle identifies the table where all data on the associated semaphore is stored.

Timeout:

Semaphore time-out time:

-1 - Wait until semaphore is clear (regardless of how long).

0 - Do not wait - return immediately. (This timeout can be used to check the state.)

> 0 - The number of seconds to wait if semaphore is not signalling, then return.

Return Value

0 (zero) if the semaphore has been gained, otherwise an [error](#) is returned.

Related Functions

[SemClose](#), [SemOpen](#), [SemSignal](#)

Example

```
Status=SemWait(hSem,10);
IF Status=0 THEN
    ...
ELSE
    Prompt("Could not get semaphore");
END
```

See Also

[Task Functions](#)

ServerRPC

Calls a remote procedure on the Citect server specified by the *ServerName* argument. You can call any of the built-in Cicode functions remotely, or your own functions. You pass the Name of the function as a string and pass the arguments for that function in Arg.

You can call the function in synchronous or asynchronous Mode. In synchronous mode, ServerRPC() does not return until the function call has completed on the server and the result is returned. In asynchronous mode, ServerRPC() returns before the function is called, an empty string is returned as the result cannot be returned.

Syntax

ServerRPC(*sServerName*, *sName*, *sArg*, *iMode* [, *sClusterName*])

sServerName:

Citect server name where the Cicode function needs to be executed. You can optionally specify this name in <ClusterName>.<ServerName> syntax.

sName:

The name of the Cicode function to call remotely as string.

sArg:

The arguments to pass to the function, separated by commas (,). Enclose string arguments in quotes "" and use the string escape character (^) around strings enclosed within a string. If you forget to enclose the string in quotes, then the string is only the first tag found.

iMode:

The mode of the call:

0 - Blocking mode - synchronous

1 - Non-blocking mode - asynchronous

sClusterName:

The name of the cluster that the server resides in. This argument is optional, as in several situations it may not be required. In single cluster systems, it is not required, or if the current Cicode task already has the correct cluster context for the server you may omit this argument.

Return Value

The result of the remote function call (as a string). If the function is called in asynchronous mode the result of the remote function cannot be returned, so an empty string is returned. If the function cannot work due to an error, empty string is also returned and the error can be obtained by calling the IsError function.

Related Functions

[TaskNew](#), [TaskNewEx](#)

See Also

[Task Functions](#)

Sleep

Suspends the current Cicode task for a specified number of seconds. After the time delay, the Cicode task wakes and continues execution. If the sleep time is 0, the Cicode task is pre-empted for 1 time slice only.

This function does not affect any other Cicode tasks - only the task calling Sleep() is suspended. If you have Cicode that runs continuously in a loop, you should call the Sleep() function somewhere within the loop, to pause the loop and allow other tasks to run.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

Sleep(*Seconds*)

Seconds:

The number of seconds. Set to 0 to pre-empt the task for one time-slice.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TaskNew](#), [ReRead](#), [SleepMS](#)

Example**Buttons**

Text	Step
Command	PLCBit=1;Sleep(2);PLCBit=0;
Comment	Switch Bit ON and then OFF 2 seconds later

```

! Display "Hello" 10 times at 60 second intervals.
WHILE I < 10 DO
    Sleep(60);
    Prompt("Hello");
    I = I + 1;
END
! Sleep a while in polling loops
WHILE < waiting for event or time> DO
    ! do what ever here
    ...
    Sleep(10);      ! sleep a while to give other tasks a go.
    ! the longer the sleep the better for other tasks.
END

```

See Also

[Task Functions](#)

SleepMS

Suspends the current Cicode task for a specified number of milliseconds. After the time delay, the Cicode task wakes and continues execution. This function is similar to the Sleep function but with greater resolution.

Although a value of 0 milliseconds is accepted, it is not recommended. Try to use at least a value of 1.

This function does not affect any other Cicode tasks; only the task calling SleepMS() is suspended. If you have Cicode that runs continuously in a loop, you should call the SleepMS() or Sleep() function somewhere within the loop, to pause the loop and allow other tasks to run.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

SleepMS(*Milliseconds*)

Milliseconds:

The number of milliseconds (1000 milliseconds per second). Set to 0 to pre-empt the task for one time-slice. Be careful not to use a value that is too small. Setting the value to 0 would generally have no desirable effect.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TaskNew](#), [ReRead](#), [Sleep](#)

Example

Buttons	
Text	Step
Command	PLCBit=1;SleepMS(500);PLCBit=0;
Comment	Switch Bit ON and then OFF 500 milliseconds later.

```
! Increment a memory variable by ten, 120 times over one minute
(twice a second).
I = 0;
WHILE I < 180 DO
    SleepMS(500);
    iRamp = iRamp + 10;
    I = I + 1;
END
! sleep a while in polling loops
WHILE < waiting for event or time> DO
    ! do what ever here
```

```

    ...
SleepMS(200);    ! sleep a while to give other tasks a go.
! the longer the sleep the better for other tasks.
END

```

See Also

[Task Functions](#)

TaskCall

Calls a Cicode function by specifying the function name and providing an arguments string.

The function will be executed in a new Cicode task with the same cluster context as the current task. The current task will be blocked until the new task completes and a value can be returned.

This function cannot be called from page foreground animation code. If this is attempted, a hardware alarm will be raised and IsError() will return 282 (Foreground Cicode cannot block).

TaskCall allows the function to be called and the arguments to be provided to be specified dynamically by the cicode logic. This may be useful in some cases where the function needed is not known until runtime.

CitectSCADA requests the required I/O device data and waits for the data to be returned before starting the function.

Syntax

TaskCall(*sName*, *sArgs*)

sName:

The name of the function to call, as a string.

sArgs:

The arguments to pass to the function, separated by commas (,). Enclose string arguments in quotes "" and use the string escape character (^) around strings enclosed within a string.

Return Value

The result of the function call (as a string). If a void function is called, an empty string is returned. To see if an error occurred (such as an invalid function name or invalid arguments) call IsError(..).

Related Functions

[TaskNew](#), [TaskNewEx](#)

Example

```
STRING result;
result = TaskCall("StrFill", "^"abc^",10");
// result will be set to "abcabcabca"
```

See Also

[Task Functions](#)

TaskCluster

Gets the name of the cluster context in which the current task is executing.

Syntax

TaskCluster()

Return Value

The cluster name of the current context or an empty string if the task is executing without a cluster context.

Related Functions

[ClusterActivate](#), [ClusterDeactivate](#), [ClusterFirst](#), [ClusterGetName](#), [ClusterIsActive](#), [ClusterNext](#), [ClusterServerTypes](#), [ClusterSetName](#), [ClusterStatus](#), [ClusterSwapActive](#), [TaskNew](#), [TaskNewEx](#)

Example

```
! Get the cluster context of the current task
sCluster = TaskCluster();
```

See Also

[Task Functions](#)

[Cluster Functions](#)

About cluster context

TaskGetSignal

Retrieves a value that indicates the signal that is currently set for a specific task. This function can be used to check the value of the current signal before using TaskSetSignal to apply a new signal.

Syntax

TaskGetSignal(*Hnd*)

Hnd:

The task's handle. To retrieve this use the function TaskHnd().

Return Value

The value of the current signal. (0 (zero) represents normal operation, 1 indicates the task is stopped).

Related Functions

[TaskSetSignal](#), [TaskHnd](#), [TaskKill](#), [TaskNew](#), [TaskResume](#)

See Also

[Task Functions](#)

TaskHnd

Gets the task handle of a specific task. You can then use the task handle with other task functions to control the task. If you do not specify a thread name, it will default to that of the current task.

Syntax

TaskHnd([*sName*])

sName:

The thread name of the task. The thread name is the name of the function that was passed to the TaskNew() function. For example, if. . .

```
TaskNew("MyTask", "", 0);
```

then:

```
hTask=TaskHnd ("MyTask");
```

will return the handle of this task.

If you do not specify a thread name, it will default to that of the current task.

Return Value

The task handle, identifying the table where all data on the task is stored.

Related Functions

[TaskKill](#), [TaskNew](#), [TaskResume](#), [TaskSuspend](#)

Example

```
! Get the task handle of the current task and then kill it.  
hTask=TaskHnd();  
TaskKill(hTask);  
! Get the task handle of MyTask and then kill it.  
hTask=TaskHnd("MyTask");  
TaskKill(hTask);
```

See Also

[Task Functions](#)

TaskKill

Kills a task. The Cicode task will be stopped and will not run again.

Syntax

TaskKill(*hTask*)

hTask:

The task handle, returned from the TaskNew() or TaskHnd() function. The task handle identifies the table where all data on the associated task is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Note: TaskKill is an abrupt way to stop a Cicode task. It can cause system errors and may have unintended consequences. Whenever possible, use TaskGetSignal and TaskSetSignal to stop Cicode tasks. Use TaskKill as a last resort and after observing the following:

WARNING

UNINTENDED EQUIPMENT OPERATION

- Do not use TaskKill to stop Cicode tasks until you have attempted the alternative methods stated above.
- Place the processes and devices controlled by CitectSCADA into a state preventing unintended operation before using TaskKill to stop a Cicode task.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

Related Functions

[TaskGetSignal](#), [TaskSetSignal](#), [TaskHnd](#), [TaskNew](#), [TaskResume](#), [TaskSuspend](#)

Example

```

! Create a task, run it for 10 seconds and then kill it.
hTask=TaskNew("MyFunc","",0);
Sleep(10);
TaskKill(hTask);
FUNCTION
MyFunc()
    INT Count;
    WHILE 1 DO
        Prompt("Hello "+Count:###);
        Count=Count+1;
    END
END

```

See Also

[Task Functions](#)

TaskNew

Creates a new Cicode task and returns the task handle. You pass the *Name* of the function (that creates the task) as a string, not as the function tag, and pass the arguments for that function in *Arg*. After the task is created, it runs in parallel to the calling task. The new task will run forever unless it returns from the function or is killed by another task.

By default, CitectSCADA requests necessary I/O device data and waits for the data to be returned before starting the task - the task is provided with the correct data, but there will be a delay in starting the task. If you add 16 to the Mode, CitectSCADA starts the task immediately, without waiting for any data from the I/O devices - any I/O device variable that you use will either contain 0 (zero) or the last value read.

WARNING

UNINTENDED EQUIPMENT OPERATION

When mode 16 is used, ensure either appropriate delays are applied before processing references to tags or check qualities of tags. Otherwise, the execution system may process invalid data and returns incorrect results.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

You can specify that if the task is already running, the function will exit without launching a new task and an [error](#) will display. This is useful when you want only a single instance of the function running at any point in time.

It is also possible to run the task within the context of a particular cluster in a multi cluster system by setting the ClusterName parameter. If a cluster is not specified, the task will use the cluster context of the caller, be it a page or Cicode task. Please be aware that the cluster context cannot be changed once the code is running.

Any animation output for the new task is displayed in the window where it was created. If you want to send output to other windows, use the WinSelect() function.

Syntax

TaskNew(*sName*, *sArg*, *Mode* [, *ClusterName*])

sName:

The name of the function to create the task, as a string.

sArg:

The set of arguments to be passed to the function. Individual arguments need to be separated by commas (,). Enclose string arguments in quotes ("") and use the string escape character (^) around strings enclosed within a string. If the string in quotes is not enclosed, then the string is only the first tag found. The entire set of arguments need to be enclosed in quotes ("").

Mode:

The mode of the task:

0 - Task runs forever.

1 - Task runs until the current page is changed.

2 - Task runs until the current window is freed.

4 - This mode is deprecated and not active. Currently, by default, task requests I/O device data before starting.

8 - If the task already exists, the function will exit without launching the new task.

16- Task doesn't wait for necessary I/O device data and starts immediately.

You can select any one of modes 0, 1 or 2 and may add mode 4 and/or mode 8 and/or mode 16. For example, set Mode to 6 to request I/O device data before starting the task, and to run the task until the current window is freed.

ClusterName:

The name of the cluster context to be applied to the new Cicode task. This is optional if you have one cluster or are resolving the task via the current cluster context. The argument is enclosed in quotation marks "". You may pass "-" as the ClusterName argument to run the requested Cicode task without a cluster context.

Return Value

The task handle, or -1 if the task cannot be successfully created. The task handle identifies the table where data on the associated task is stored.

Related Functions

[TaskCall](#), [TaskHnd](#), [TaskKill](#), [TaskNewEx](#), [TaskResume](#), [TaskSuspend](#), [ReRead](#), [WinSelect](#)

Example

```

! Create a task that displays a message for 10 seconds.
hTask=TaskNew("MyFunc","Data",0);
! Continue to run while task runs.
FUNCTION
MyFunc(STRING Msg)
    FOR I=0 TO 10 DO
        Prompt(Msg);
        Sleep(1);
    END
END
...
! Call a function which expects more complex argument
hTask=TaskNew("ArgFunc","^"string one^","^"string two^",1,2",0);
hTask=TaskNew("ArgFunc","^""+sOne+"^",^""+sTwo+"^","+iOne:##+","+
iTwo:##,0);
FUNCTION
ArgFunc(STRING sOne, STRING sTwo, INT iOne, INT iTwo)
    ...
END

```

See Also

[Task Functions](#)

TaskNewEx

Creates a new Cicode task with an individual subscription rate and returns the task handle. You pass the *Name* of the function (that creates the task) as a string, not as the function tag, and pass the arguments for that function in *Arg*. After the task is created, it runs in parallel to the calling task. The new task will run forever with tags updated at the specified time interval unless it returns from the function or is killed by another task.

By default, CitectSCADA requests necessary I/O device data and waits for the data to be returned before starting the task - the task is provided with the correct data, but there will be a delay in starting the task. If you add 16 to the Mode, CitectSCADA starts the task immediately, without waiting for any data from the I/O devices - any I/O device variable that you use will either contain 0 (zero) or the last value read. Use Mode 16 when the task has to start immediately.

WARNING

UNINTENDED EQUIPMENT OPERATION

When mode 16 is used, ensure either appropriate delays are applied before processing references to tags or check qualities of tags. Otherwise, the execution system may process invalid data and return incorrect results.

Failure to follow these instructions can result in death, serious injury, or equipment damage.

You can specify that if the task is already running, the function will exit without launching the new task and an [error](#) will display. This is useful when you want only a single instance of the function running at any point in time.

It is also possible to run the task within the context of a particular cluster in a multi cluster system by setting the ClusterName parameter. If a cluster is not specified, the task will use the cluster context of the caller, be it a page or Cicode task. Please be aware that the cluster context cannot be changed once the code is running.

Any animation output for the new task is displayed in the window where it was created. If you want to send output to other windows, use the WinSelect() function.

Syntax

TaskNewEx(sName, sArg, Mode, SubscriptionRate [, ClusterName])

sName:

The name of the function to create the task, as a string.

sArg:

The set of arguments to be passed to the function. Individual arguments need to be separated by commas (,). Enclose string arguments in quotes "" and use the string escape character (^) around strings enclosed within a string. If you do not enclose the string in quotes, then the string is only the first tag found. The entire set of arguments need to be enclosed in quotes ("").

Mode:

The mode of the task:

0 - Task runs forever.

1 - Task runs until the current page is changed.

2 - Task runs until the current window is freed.

4 -This mode is deprecated and not active. Currently, by default, task requests all I/O device data before starting.

8 - If the task already exists, the function will exit without launching the new task.

16- Task doesn't wait for necessary I/O device data and starts immediately.

You can select any one of modes 0, 1 or 2 and may add mode 4 and/or mode 8, and/or mode 16. For example, set Mode to 6 to request I/O device data before starting the task, and to run the task until the current window is freed.

SubscriptionRate

The subscription rate for the task, between 0 and 60000 milliseconds, which determines the frequency at which the I/ O Server reads I/O device data in order to provide tags with up-to-date Cicode variables. A value of -1 may be passed which will use the current task's subscription rate or the default value as set by the existing parameter [CODE]TimeData which defaults to 250.

ClusterName:

The name of the cluster context to be applied to the new Cicode task. This is optional if you have one cluster or are resolving the task via the current cluster context. The argument is enclosed in quotation marks "". You may pass "-" as the ClusterName argument to run the requested Cicode task without a cluster context.

Return Value

The task handle, or -1 if the task cannot be successfully created. The task handle identifies the table where data on the associated task is stored.

Related Functions

[TaskCall](#), [TaskHnd](#), [TaskKill](#), [TaskNew](#), [TaskResume](#), [TaskSuspend](#), [ReRead](#), [WinSelect](#)

Example

```
! Create a task that calls a function every half second.
```

```
hTask=TaskNewEx("MyFunc","Data",0,500);
```

See Also

[Task Functions](#)

TaskResume

Resumes a task that was suspended by the TaskSuspend() function. After a task is resumed, it runs on the next time-slice.

Syntax

TaskResume(*hTask*)

hTask:

The task handle, returned from the TaskNew() or TaskHnd() function. The task handle identifies the table where all data on the associated task is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TaskHnd](#), [TaskKill](#), [TaskSuspend](#), [TaskNew](#)

Example

```
TaskResume (hTask);
```

See Also

[Task Functions](#)

TaskSetSignal

Manually applies a signal to a specified task.

Syntax

TaskSetSignal(*Hnd*, *nSignal*)

Hnd:

The task's handle. To retrieve this use the function TaskHnd().

nSignal:

Allows you to signal a specified task. Set to 0 (zero) for normal operation, 1 to stop the task or any other number that represents a defined signal.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TaskGetSignal](#), [TaskHnd](#), [TaskKill](#), [TaskSuspend](#), [TaskNew](#), [TaskResume](#)

See Also

[Task Functions](#)

TaskSuspend

Suspends a task. The task will stop running and will start again only when TaskResume() is called.

Syntax

TaskSuspend(*hTask*)

hTask:

The task handle, returned from the TaskNew() or TaskHnd() function. The task handle identifies the table where all data on the associated task is stored.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TaskHnd](#), [TaskKill](#), [TaskNew](#), [TaskResume](#)

Example

```
TaskSuspend (hTask) ;  
...  
TaskResume (hTask) ;
```

See Also

[Task Functions](#)

Chapter: 55 Time and Date Functions

Time/date functions manipulate time and date variables. CitectSCADA stores time/date-related variables as a single integer. This integer represents the number of seconds since 01/01/1970. It is in GMT, but it has an offset that updates it to local time (determined by the timezone the application is in). The Time/date functions convert this integer into time and date variables.

Note: The Time/date functions can only be used with dates between 1980 and 2035.

Time/Date Functions

Following are functions relating to Time and Date:

Date	Gets the current system date in string format.
DateAdd	Adds time to a date.
DateDay	Gets the day from a date.
DateInfo	Returns the date format currently in effect on the CitectSCADA Server.
DateMonth	Gets the month from a date.
DateSub	Subtracts two dates.
DateWeekDay	Gets the day of week from a date.
DateYear	Gets the year from a date.
OLE- DateToTime	Converts an OLE DATE value to a CitectSCADA time/date value.
SysTime	Marks the start of an event.
SysTimeDelta	Calculates the time-span of an event.
Time	Gets the current system time in string format.

TimeCurrent	Gets the current time/date value.
TimeHour	Gets hours from a time.
TimeInfo	Returns the time format currently in effect on the CitectSCADA Server.
TimeMidNight	Converts a time variable into the time at midnight.
TimeMin	Gets minutes from a time.
TimeSec	Gets seconds from a time.
TimeSet	Sets the new system time.
TimeToStr	Converts a time/date variable into a string.
TimeUT-COffset	Determines the local time bias from UTC at a specified time.

See Also

[Functions Reference](#)

Date

Gets the current date in string format.

Note: Time/Date functions can only be used with dates between 1980 and 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

Date([Format])

Format:

The format required:

2 - Short date format, dd/mm/yy

3 - Long date format, day month year

9 - Extended date format, dd/mm/yyyy

If omitted, the default Format is 2. These formats follow the Regional Settings found in the Windows Control Panel.

Return Value

The current date (in string format).

Related Functions

[Time](#), [TimeToStr](#), [TimeCurrent](#)

Example

```
/* If the current system date is 3rd November 1991 and the Windows
date format is dd/mm/yy; */
str = Date();
! Sets str to "3/11/91".
str = Date(2);
! Sets str to "3/11/91".
str = Date(3);
! Sets str to "3rd November 1991".
```

See Also

[Time/Date Functions](#)

DateAdd

Adds time (in seconds) to a time/date value. The return value is in time/date variable format. Use this function for time and date calculations.

Syntax

DateAdd(*Time*, *AddTime*)

Time:

The time/date to which the AddTime will be added.

AddTime:

The time to add, in seconds.

Return Value

The date as a time/date variable.

Related Functions

[TimeToStr](#), [DateSub](#)

Example

```
DateVariable=DateAdd(StrToDate("3/11/91"),86400);
! Adds 24 hours to 3/11/91.
NewDate=TimeToStr(DateVariable);
! Sets NewDate to 4/11/91.
```

See Also

[Time/Date Functions](#)

DateDay

Gets the day of the month from a time/date variable.

Time/date functions can only be used with dates between 1980 and 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
    ...
ELSE
    ...
END
```

Syntax

DateDay(*Time*)

Time:

The time/date variable.

Return Value

The day of the month as an integer.

Related Functions

[Date](#)

Example

```
! If the current system date is 3rd November 1991;
Variable=DateDay(TimeCurrent());
```

```
! Sets Variable to 3.
```

See Also

[Time/Date Functions](#)

DateInfo

Returns the date format currently used on the CitectSCADA Server.

Syntax

DateInfo(*nInfo*, *nExtra*)

nInfo:

Determines the contents of the string returned by the DateInfo() function. Valid values and resulting strings are:

1 - The current date order:

- "0" - MMDDYY
- "1" - DDMMYY
- "2" - YYMMDD.

2 - The current date delimiter.

3 - The current short date format.

4 - The current long date format.

5 - The current extended date format.

6 - The short weekday string. The particular weekday returned is determined by the *nExtra* argument.

7 - The long weekday string. The particular weekday returned is determined by the *nExtra* argument.

8 - The short month string. The particular month returned is determined by the *nExtra* argument.

9 - The long month string. The particular month returned is determined by the *nExtra* argument.

nExtra:

When an *nInfo* argument of 6 or 7 is specified, the *nExtra* argument determines which weekday (1-7) is returned by the DateInfo() function.

When an *nInfo* argument of 8 or 9 is specified, the *nExtra* argument determines which month (1-12) is returned by the DateInfo() function.

The *nExtra* argument is ignored if any other *nInfo* value is passed to the function.

Return Value

A string containing one of the following:

- Current date order ("0" for MMDDYY, "1" for DDMMYY, "2" for YYMMDD);
- Current date delimiter;
- Current short date format;
- Current long date format;
- Current extended date format;
- Short weekday string;
- Long weekday string;
- Short month string;
- Long month string;

depending on the *nInfo* and *nExtra* arguments passed to the function.

Related Functions

[TimeInfo](#)

Example

```
! If the current system date is the fourth of December 2002;
TwelfthMonth=DateInfo(9,12);
! Sets TwelfthMonth to "December".
```

See Also

[Time/Date Functions](#)

DateMonth

Gets the month from a time/date variable.

Note: Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

DateMonth(*Time*)

Time:

The time/date variable.

Return Value

The month of the year as an integer.

Related Functions

[Date](#)

Example

```
! If the current system date is 3rd November 1991;
Variable=DateMonth(TimeCurrent());
! Sets Variable to 11.
```

See Also

[Time/Date Functions](#)

DateSub

Subtracts time (in seconds) from a time/date value. The return value is in time/date variable format. Use this function for time and date calculations.

Syntax

DateSub(*Time, SubTime*)

Time:

The time/date from which the SubTime will be subtracted.

SubTime:

The time to subtract, in seconds.

Return Value

The time difference (in seconds) as an integer.

Related Functions

[Date](#), [DateAdd](#)

Example

```
Variable=DateSub(StrToDate("05/11/91"),StrToDate("03/11/91"));
! Sets Variable to number of seconds between 2 date/times.
Str=TimeToStr(Variable,5);
! Sets Str to "48:00:00".
```

See Also

[Time/Date Functions](#)

DateWeekDay

Gets the day of the week from a time/date variable.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

DateWeekDay(*Time*)

Time:

The time/date variable.

Return Value

An integer representing the day of the week as follows:

- 1 - Sunday
- 2 - Monday
- 3 - Tuesday
- 4 - Wednesday
- 5 - Thursday
- 6 - Friday
- 7 - Saturday

Related Functions[Date](#), [TimeCurrent](#)**Example**

```
! If the current system date is Sunday, 3rd November 1991;
Variable=DateWeekDay(TimeCurrent());
! Sets Variable to 1.
```

See Also[Time/Date Functions](#)**DateYear**

Gets the year from a time/date variable.

Time/date functions can only be used with dates between 1980 and 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax**DateYear**(*Time* [, *Mode*])*Time*:

The time/date variable.

Mode:

The format required:

0 - Short year, yy. If you use this mode during the year 2000, 0 (zero) will be returned.

1 - Long year, yyyy

If omitted, the default Mode is 0.

Return Value

The year as an integer.

Note: In the year 2000, this function will return 0 (zero) if mode 0 (zero) is used.

Related Functions

[Date](#)

Example

```
! If the current system date is 3rd November 1991;
Variable=DateYear(TimeCurrent(),0);
! Sets Variable to 91.
! If the current system date is 18th October 2000;
Variable=DateYear(TimeCurrent(),0);
! Sets Variable to 0.
Variable=DateYear(TimeCurrent(),1);
! Sets Variable to 1991.
```

See Also

[Time/Date Functions](#)

OLEDATOTIME

Converts an OLE DATE value (stored in a REAL) to a CitectSCADA time/date value.

Note: An OLE DATE representing a local time in the daylight savings(DST) to standard time(Std) transition period will be converted to the DST value internally. For example, if the DST transition is 30/3/2003 2:00:00 Std, the local time will behave in the following manner: 2:00:00 DST > 2:59:59 DST > 2:00:00 Std. Because of this, a value representing the period between 2:00:00 and 2:59:59 on that date will be interpreted as 2:00:00 DST, not Std.

Syntax

OLEDATOTIME(OLEDATe, Local)

OLEDATe:

The OLE DATE value to convert (stored as a REAL).

Local:

0 - OleDate represents a UTC time.

1 - OleDate represents a Local time.

Return Value

Returns a CitectSCADA time/date value.

Related Functions

[TimeCurrent](#), [TimeToOLEDate](#)

Example

```
Real = TimeToOLEDate(TimeCurrent(), 1);
! Sets Real to the local date/time value
TimeVariable = OLEDateToTime(Real, 1);
! Sets TimeVariable to the value of Real when interpreted as Local
time.
```

See Also

[Time/Date Functions](#)

SysTime

Gets the CitectSCADA internal system millisecond counter. The counter is not based on time, but counts from 0 up to the maximum integer value and then back to 0.

You can use this function to time events down to the millisecond level, either by subtracting the current SysTime from the SysTime at the start of the event, or by using the SysTimeDelta() function (which will give the same result).

The SysTime() function does not return the time of day. Use the Time() or TimeCurrent() function to obtain the time of day.

Time/date functions can only be used with dates between 1980 and 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

SysTime()

Return Value

The CitectSCADA internal system millisecond counter (as an integer).

Related Functions

[SysTimeDelta](#), [Time](#), [TimeCurrent](#)

Example

```
Start=SysTime();
! Gets the current time.
...
Delay=SysTime()-Start;
! Sets Delay to the time difference, in milliseconds.
```

See Also

[Time/Date Functions](#)

SysTimeDelta

Calculates the time difference between a start time and the current time, and updates the start time to the current time. You can time continuous events in a single operation. See the SysTime() function for information on its use.

Time/date functions can only be used with dates between 1980 and 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

SysTimeDelta(*Start*)

Start:

The start time returned from the SysTime() function.

Return Value

The time difference from a start time and the current time.

Related Functions

[SysTime](#)

Example

```

Start=SysTime();
! Gets the current time.
...
Delay1=SystimeDelta(Start);
! Sets Delay1 to the time difference from Start.
...
Delay2=SystimeDelta(Start);
! Sets Delay2 to the time difference from the last SystimeDelta()
call.

```

See Also

[Time/Date Functions](#)

Time

Gets the current time in string format.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```

IF StrToDate(Arg1)>0 THEN
    ...
ELSE
    ...
END

```

Syntax

Time([Format])

Format:

The format of the time:

0 - Short time format, hh:mm AM/PM

1 - Long time format., hh:mm:ss AM/PM

If omitted, the default *Format* is 0.

Return Value

The current time (as a string).

Related Functions

[Date](#), [TimeToStr](#)

Example

```
! If the current time is 10:45:30;
Variable=Time();
! Sets Variable to "10:45".
Variable=Time(0);
! Sets Variable to "10:45 AM".
Variable=Time(1);
! Sets Variable to "10:45:30 AM".
```

See Also

[Time/Date Functions](#)

TimeCurrent

Gets the current system time/date in time/date variable format. Please be aware that CitectSCADA stores time as the number of seconds since 01/01/1970. You can convert this value into usable date and time variables by using the various Date and Time functions.

Time/date functions can only be used with dates between 1980 and 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

TimeCurrent()

Return Value

A time/date variable.

Related Functions

[StrToDate](#), [StrToTime](#)

Example

```
! If the current system time is 11:43:10 a.m. ;
TimeVariable=TimeToStr(TimeCurrent(),0);
! Sets TimeVariable to "11:43".
```

See Also

[Time/Date Functions](#)

TimeHour

Gets the hour value from a time/date variable.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

TimeHour(*Time*)

Time:

The time/date variable.

Return Value

The hour (as an integer).

Related Functions

[TimeCurrent](#)

Example

```
! If the current system time is 11:43:10 a.m.;
HoursVariable=TimeHour(TimeCurrent());
! Sets HoursVariable to 11.
```

See Also

[Time/Date Functions](#)

TimelInfo

Returns the time format currently used on the CitectSCADA Server.

Syntax

TimeInfo(*nInfo*)

nInfo:

Determines the contents of the string returned by the TimeInfo() function. Valid values and resulting strings are:

1- The current time hour format:

- "0" - 12 hour
- "1" - 24 hour

2- The current time delimiter.

3- The current morning time extension.

4- The current evening time extension.

Return Value

Depending on the *nInfo* argument passed to the function, a string containing:

- Current time hour format ("0" for 12 hour, "1" for 24 hour)
- Current time delimiter
- Current morning time extension
- Current evening time extension

Related Functions

[DateInfo](#)

Example

```
! If the current system time is 15:43:10.;  
ClockType=TimeInfo(1);  
! Sets ClockType to "1".
```

See Also

[Time/Date Functions](#)

TimeMidNight

Returns the number of seconds between midnight on January 1, 1970, and the midnight immediately prior to the specified time/date. This function is useful for performing calculations with the time and date.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

TimeMidNight(*Time*)

Time:

The time/date variable.

Return Value

A time/date variable.

Related Functions

[TimeCurrent](#)

Example

```
timeNow = TimeCurrent();
! get the time variable at 7am today
time7am = TimeMidNight(timeNow) + 7*60*60;
IF timeNow > time7am AND timeNow < time7am + 10 THEN
  Beep();
  Prompt("Wake Up!");
END
```

See Also

[Time/Date Functions](#)

TimeMin

Gets the minutes value from a time/date variable.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
  ...
ELSE
  ...
END
```

Syntax

TimeMin(*Time*)

Time:

The time/date variable.

Return Value

The minute (as an integer).

Related Functions

[TimeCurrent](#)

Example

```
! If the current system time is 11:43:10 a.m.  
MinutesVariable=TimeMin(TimeCurrent());  
! Sets MinutesVariable to 43.
```

See Also

[Time/Date Functions](#)

TimeSec

Gets the seconds value from a time/date variable.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN  
    ...  
ELSE  
    ...  
END
```

Syntax

TimeSec(*Time*)

Time:

The time/date variable.

Return Value

The second (as an integer).

Example

```
! If the current system time is 11:43:10 a.m.;
SecondsVariable=TimeSec(TimeCurrent());
! Sets SecondsVariable to 10.
```

See Also

[Time/Date Functions](#)

TimeSet

Sets the new system time. You can set the time only on the computer which this function is called.

Time/date functions can only be used with dates from 1980 to 2035.

If you call TimeSet without the required privileges to change the system time you will receive a hardware alarm indicating this.

Note: that when using Vista, UAC needs to be disabled in order for the time to be set.

Syntax

TimeSet(*Time*)

Time:

The time/date variable to which the new time is set. Sets the time on this computer only.

Return Value

The error status of the set

Related Functions

[DateInfo](#)

Example

```
! set the time to 11:43 on June 23 1993
time = StrToTime("11:43:00") + StrToDate("23/6/93");
```

```
TimeSet(time); .
```

See Also

[Time/Date Functions](#)

TimeToOLEDate

Converts a CitectSCADA time/date value to an OLE DATE value (this should be stored in a REAL).

Syntax

TimeToOLEDate(*Time*, *Local*)

Time:

Time/date variable.

Local:

0 - The return value is output as UTC time.

1 - The return value is output as Local time.

Return Value

Returns an OLE date value.

Related Functions

[TimeCurrent](#), [OLEDateToTime](#)

Example

```
Real = TimeToOLEDate(TimeCurrent(), 1);
! Sets Real to the local date/time value
```

See Also

[Time/Date Functions](#)

TimeToStr

Converts a time/date variable into a string. Use this function for calculating time differences or run times, and so on. Set *Format* to 6 to convert time periods that are in milliseconds, such as the times that are returned from the SysTime() and SysTimeDelta() functions.

Note: Once a date/time is retrieved as UTC, the string cannot be used by the Cicode functions StrToDate and StrToTime to synthesize a date/time value as these functions support local time only.

Time/date functions can only be used with dates from 1980 to 2035. You should check that the date you are using is valid with Cicode similar to the following:

```
IF StrToDate(Arg1)>0 THEN
    ...
ELSE
    ...
END
```

Syntax

TimeToStr(*Time*, *Format* [, *UTC*])

Time:

The time/date variable.

Format:

Format of the string:

- 0 - Short time format, hh:mm AM/PM.
- 1 - Long time format, hh:mm:ss AM/PM.
- 2 - Short date format, dd/mm/yy.
- 3 - Long date format, day month year.
- 4 - Time and date, weekday month day year hh:mm:ss AM/PM.
- 5 - Long time period, hh:mm:ss. Time needs to be in seconds.
- 6 - Millisecond time period, hh:mm:ss.xxx ("xxx" represents milliseconds).
Time needs to be in milliseconds.
- 7 - Short time period, hh:mm. Time needs to be in seconds.
- 8 - Long time period, "xxxxx Days hh Hours mm min ss sec where xxxx = number of days since 1/1/1970". Time needs to be in seconds.
- 9 - Extended date format, dd/mm/yyyy.

UTC:

Universal Time Co-ordinate (optional)

- 0 - Display the string as a local date/time (default).
- 1 - Display the string as a UTC date/time (valid for formats 0-4 and 9).

Return Value

A string containing the converted time/date or period variable, or an empty string if invalid.

Related Functions

[Time](#), [TimeCurrent](#), [Date](#)

Example

```
! If the current system time is 11:50:00 a.m.  
String=TimeToStr(TimeCurrent(),0);  
! Sets String to "11:50 AM".  
String=TimeToStr(125 + TimeCurrent(),5);  
! Sets String to "11:52:05" (the current time + 2 minutes and 5  
seconds).
```

See Also

[Time/Date Functions](#)

TimeUTCOffset

Determines the local time bias from UTC that was in force at a specified time. For example, US Pacific Standard Time is -8 hrs from UTC, so -28800 would be returned (-8 hours x 60 minutes x 60 seconds). However, if the specified time occurred during daylight saving, the returned value would be -7 hours (or -25200 seconds).

Syntax

TimeUTCOffset(*Time*)

Time:

The time/date variable.

Return Value

The local time bias in seconds.

Related Functions

[TimeCurrent](#)

See Also

[Time/Date Functions](#)

Chapter: 56 Timestamp Functions

The Timestamp functions enable you to programmatically read and write the Timestamp values of tag elements and access the timestamp information associated with a tag value.

No function taking either Timestamp or Quality as an argument can be called from the Cicode Kernel Window or through a CtCicode CtAPI function.

Timestamp Functions

The following functions are used to interface with the [TIMESTAMP](#) data type.

TimestampAdd	Adds time (in part of) to a TIMESTAMP variable.
TimestampCreate	Returns a timestamp variable created from the parts.
TimestampToStr	Converts a TIMESTAMP variable into a string.
TimestampDifference	Returns a difference between two TIMESTAMP variables as a number of milliseconds.
TimestampCurrent	Returns the current system date and time as a TIMESTAMP variable.
TimestampFormat	Format a TIMESTAMP variable into a string.
TimestampGetPart	Returns one part (year, month, day, etc) of the timestamp variable.
TimeIntToTimestamp	Converts a time INTEGER which is represented as a number of seconds since 01/01/1970 to a TIMESTAMP
TimestampSub	Subtracts time (in part of) from a TIMESTAMP variable.
TimestampToInt	Converts a TIMESTAMP variable into a time INTEGER which is represented as a number of seconds since 01/01/1970.
VariableTimestamp	Extracts the timestamp from a given variable.

See Also

[Functions Reference](#)

TimeIntToTimestamp

Converts a time INTEGER which is represented as a number of seconds since 01/01/1970 to a TIMESTAMP

Syntax

TimeIntToTimestamp(INT TimeInt [, INT Millisecond [, INT UTC]])

TimeInt:

The number of seconds since 01/01/1970.

Millisecond:

The number of milliseconds since last second (optional).

UTC:

Universal Time Co-ordinate (optional):

0 – The given time INTEGER is a local date/time.

1 – The given time INTEGER is a UTC date/time (default).

Return Value

A TIMESTAMP variable or INVALID_TIMESTAMP if invalid.

Related Functions

[TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampAdd](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToInt](#)

Example

```
INT TimeInt = TimeCurrent();
```

```
TIMESTAMP t1 = TimeIntToTimestamp(t1);
```

```
STRING sTimestamp = TimestampToStr(t1, 0, 0);
```

```
// sTimestamp equals current time in the short time format
```

```
// i.e. 'HH:MM AM/PM'
```

See Also

[Timestamp Functions](#)

TimestampAdd

Adds an offset to a TIMESTAMP variable.

Syntax

TimestampAdd(TIMESTAMP *Timestamp*, INT *Offset* [, INT *Part*])

Timestamp:

The timestamp to which Offset will be added

Offset:

The offset to add, expressed in units of the part parameter

Part:

Indicates which part to add:

0 – Offset is in years.

1 – Offset is in months.

2 – Offset is in days.

3 - Offset is in hours.

4 - Offset is in minutes.

5 - Offset is in seconds (default)

6 - Offset is in milliseconds

Return Value

The TIMESTAMP variable, or INVALID_TIMESTAMP if invalid.

Related Functions

[TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimeIntToTimestamp](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToInt](#)

Example

```
TIMESTAMP t1 = TimestampAdd(Tag1.T, 100); // 100 seconds
```

```
TIMESTAMP t2 = TimestampAdd(Tag1.T, 1, 0); // 1 year
```

See Also

[Timestamp Functions](#)

TimestampCurrent

Return the current system date and time as a TIMESTAMP variable.

Syntax

TimestampCurrent()

Return Value

A TIMESTAMP variable containing the current system date and time.

Related Functions

[TimestampAdd](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToTimeInt](#), [TimeIntToTimestamp](#)

Example

```
TIMESTAMP t1 = TimestampCurrent();
```

See Also

[Timestamp Functions](#)

TimestampCreate

Returns a timestamp variable created from the parts.

Syntax

TimestampCreate(INT Year, INT Month, INT Day, INT Hour, INT Minute, INT Second, INT Millisecond [, INT bUtc])

Timestamp:

The timestamp from which the part will be extracted.

Year:

The year part.

Month:

The month part.

Day:

The day part.

Hour:

The hour part.

Minute:

The minute part.

Second:

The second part.

Millisecond:

The millisecond part.

UTC:

Universal Time Co-ordinate (optional):

0 – The given time INTEGER is a local date/time.

1 – The given time INTEGER is a UTC date/time (default).

Return Value

The composed TIMESTAMP variable, or INVALID_TIMESTAMP if invalid

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToInt](#), [TimeIntToTimestamp](#),

Example

```
TIMESTAMP timestamp = TimestampCreate(2009, 6, 29, 11, 2, 10, 468);
```

```
STRING sTimestamp = TimestampFormat(t1, "dd/MM/yyyy hh:mm:ss.fff");
```

```
// sTimestamp equals '29/06/2009 11:02:10.468'
```

See Also

[Timestamp Functions](#)

TimestampDifference

Returns the difference between two TIMESTAMP variables as a number of milliseconds.

Syntax

TimestampDifference(TIMESTAMP Timestamp1, TIMESTAMP Timestamp2 [, INT Part [, INT bCumulative]])

Timestamp 1:

The TIMESTAMP variable 1.

Timestamp 2:

The TIMESTAMP variable 2.

Part:

The type of time units being used for the result:

0 – Result is in years.

1 – Result is in months.

2 – Result is in days.

3 - Result is in hours

4 - Result is in minutes.

5 - Result is in seconds (default).

6 - Result is in milliseconds.

bCumulative:

Defines how to pass results which values are greater than their time units (see example below).

0 – Non-cumulative

1 – Cumulative mode (default).

Return Value

The time period between Timestamp1 and Timestamp2. The value is equal or greater than zero. If error, returns 0 with an error code.

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampSub](#), [TimestampToStr](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToInt](#), [TimeIntToTimestamp](#)

Example

```
TIMESTAMP t1 = TimestampCreate(2008, 11, 28, 09, 01, 30);
```

```
TIMESTAMP t2 = TimestampCreate(2008, 11, 28, 09, 00, 00);
```

```
INT nTimespanCumulative = TimestampDifference(t1, t2, 5, 0);
```

```
// nTimespanCumulative is equal 90 (seconds)
```

```
INT nTimespanNonCumulative = TimestampDifference(t1, t2, 5, 1);
```

```
// nTimespanNonCumulative is equal 30 (seconds)
```

See Also

[Timestamp Functions](#)

TimestampFormat

Format a TIMESTAMP variable into a string.

Syntax

TimestampFormat(TIMESTAMP *Timestamp*, STRING *Format* [, INT *UTC*])

Timestamp:

The timestamp variable.

Format:

The format of the string is the same as .NET Framework DateTime format. Specifically be reminded that the format is case sensitive. For example 'MM' is the zero padded month number, whereas 'mm' is the zero padded current minute within the hour. Therefore no Year, Day or Seconds will be displayed if they are specified in uppercase as: YYYY, DD, SS. The correct display will only occur when they are specified in lowercase as: yyyy, dd, ss.

For more details regarding this format refer to "Custom Date and Time Format Strings" in the Microsoft .NET Framework Developer's Guide from the MSDN Library.

UTC:

Universal Time Co-ordinate (optional):

0 - Returns the time as a local date/time (default).

1 - Returns the time as a UTC date/time.

Return Value

A string containing the converted time/date, or an empty string if invalid.

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampCreate](#), [TimestampGetPart](#), [TimestampToInt](#), [TimeIntToTimestamp](#)

Example

```
TIMESTAMP t1 = TimestampCreate(2009,07,11,09,27,34,123);
```

```
STRING sTimestamp = TimestampFormat(t1, "dd/MM/yyyy hh:mm:ss.fff");
```

```
// sTimestamp equals "11/07/2009 09:27:34.123"
```

See Also

[Timestamp Functions](#)

TimestampGetPart

Returns one part (year, month, day, etc) of the timestamp variable.

Syntax

TimestampGetPart(TIMESTAMP *Timestamp*, INT *Part* [, INT *bUtc*])

Timestamp:

The timestamp from which the part will be extracted.

Part:

Indicates which part to extract:

- 0 – The year part
- 1 – The month part.
- 2 – The day part.
- 3 – The hour part.
- 4 – The minute part.
- 5 – The second part.
- 6 – The millisecond part.
- 7 - The number of milliseconds since midnight last occurred.

UTC:

Universal Time Co-ordinate (optional):

- 0 – The given time INTEGER is a local date/time.(default).
- 1 – The given time INTEGER is a UTC date/time.

Return Value

The required part of the TIMESTAMP variable.

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampToInt](#), [TimeIntToTimestamp](#)

Example

```
TIMESTAMP t1;
// insert code here
INT year = TimestampGetPart(t1, 0);
INT second = TimestampGetPart(t1, 5);
```

See Also

[Timestamp Functions](#)

TimestampSub

Subtracts an offset from a TIMESTAMP variable.

Syntax

TimestampSub(TIMESTAMP Timestamp, INT Offset [, INT Part])

Timestamp:

The timestamp to which Offset will be subtracted

Offset:

The offset to subtract, expressed in units of the part parameter

Part:

Indicates which part to subtract

0 – Offset is in years.

1 – Offset is in months.

2 – Offset is in days.

3 - Offset is in hours.

4 - Offset is in minutes.

5 - Offset is in seconds (default)

6 - Offset is in milliseconds

Return Value

The TIMESTAMP variable, or INVALID_TIMESTAMP if invalid.

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampToStr](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToTimeInt](#), [TimeIntToTimestamp](#)

Example

```
TIMESTAMP t1 = TimestampSub(Tag1.T, 1, 0); // 1 year;
```

See Also

[Timestamp Functions](#)

TimestampToStr

Converts a TIMESTAMP variable into a string.

Syntax

TimestampToStr(Timestamp, INT Format [, INT UTC])

Timestamp:

Specifies the TIMESTAMP variable.

Format:

The format number determines which of date/time patterns are used for formatting returned string. Date/time patterns are defined in regional settings on a particular computer and can vary depend on national or individual preferences. The possible format numbers together with examples based on en-US regional settings are listed below:

- 0 – Short time format, hh:mm.
- 1 – Long time format, hh:mm:ss.
- 2 – Short date format, dd/MM/yyyy.
- 3 – Long date format, dddd, dd MMMM yyyy.
- 4 – Short date & short time format, dd/MM/yyyy hh:mm.
- 5 – Short date & long time format, dd/MM/yyyy hh:mm:ss.
- 6 – Long date & short time format, dddd, dd MMMM yyyy hh:mm.
- 7 – Long date & long time format, dddd, dd MMMM yyyy hh:mm:ss.
- 8 – Month day format, dd MMMM.
- 9 – Year month format, MMMM yyyy.
- 10 – RFC1123 format, ddd, dd MMM yyyy hh:mm:ss GMT
- 11 – Sortable date time format, yyyy-MM-ddThh:mm:ss
- 12 – Short universal format, yyyy-MM-dd hh:mm:ssZ
- 13 – Long universal format, dddd, dd MMMM yyyy hh:mm:ss
- 14 – Long Time & millisecond, hh:mm:ss.fff

UTC (optional - short for Universal Time Coordinate):

- 0 - Display the string as a local date/time (default).
- 1 - Display the string as a UTC date/time.

Return Value

A string containing the converted time/date or period variable, or an empty string if invalid.

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToInt](#), [TimeIntToTimestamp](#)

Example

```
TIMESTAMP t1 = TimestampCreate(2009,07,11,09,27,34,123);
STRING sTimestamp = TimestampToStr(t1, 0, 0);
// sTimestamp equals '9:27 AM'
```

See Also

[Timestamp Functions](#)

TimestampToInt

Converts a TIMESTAMP variable into a time INTEGER which is represented as a number of seconds since 01/01/1970.

Syntax

TimestampToInt(TIMESTAMP *Timestamp* [, INT *UTC*])

Timestamp:

The timestamp variable.

UTC:

Universal Time Co-ordinate (optional):

0 – Returns the time as a local date/time.

1 – Returns the time as a UTC date/time (default)

Return Value

Time as a number of seconds since 01/01/1970 in UTC or local time depending on the last input parameter,-1 if invalid.

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampCreate](#), [TimestampFormat](#), [TimestampGetPart](#), [TimeToInt](#), [ToTimestamp](#)

Example

```
TIMESTAMP t1 = TimestampCreate(2009,07,11,09,27,34,123);
```

```
INT TimeInt = TimestampToInt(t1);
```

```
STRING sTimestamp = TimeToStr(t1, 0, 0);
```

```
// sTimestamp equals '9:27 AM'
```

See Also

[Timestamp Functions](#)

VariableTimestamp

Extracts the timestamp from a given variable.

Note: This function is designed to be used within Cicode; using it on graphical pages may result in displaying an error message instead of an expected timestamp message when either its argument has not good quality or an execution error is set.

Syntax

VariableTimestamp(Variable, INT Type)

Variable:

The variable from which the timestamp will be extracted.

Type:

The type of timestamp:

0 – The element's date/time (default)

1 – The element's quality date/time

2 – The element's value date/time

Return Value

A TIMESTAMP of the given variable depending on the type. If Variable is NULL, returns INVALID_TIMESTAMP.

Timestamps of uninitialized stack variables, uninitialized code variables and constants are equal to 0 - invalid timestamp, while their qualities are GOOD

Related Functions

[TimestampAdd](#), [TimestampCurrent](#), [TimestampDifference](#), [TimestampSub](#), [TimestampToStr](#), [TimestampFormat](#), [TimestampGetPart](#), [TimestampToInt](#), [TimeIntToTimestamp](#),

Example

```
INT codeVariable = 1;
```

```
INT
```

```
FUNCTION
```

```
MyFunction(REAL arg1)
```

```
STRING str = "My string";
```

```
TIMESTAMP ts;
```

```
ts = VariableTimestamp(codeVariable, 0); //code variable
```

```
ts = VariableTimestamp(arg1, 0); //function argument
```

```
ts = VariableTimestamp(str, 0); //stack variable
```

```
ts = VariableTimestamp(Tag1, 0); //any tag/local variable
```

```
    RETURN 1;
```

```
END
```

See Also

[Timestamp Functions](#)

Chapter: 57 Trend Functions

You can control a trend's operation by using the trend functions. CitectSCADA has standard trend pages, so you would not normally use these low-level functions unless you want to define your own trend displays. You can control the movement of the trend cursor, trend scaling, and the definition of trend attributes (such as the trend starting time and sampling period). You can also create, and subsequently delete trends.

Trend Functions

Following are functions relating to Trends:

TrnAddHistory	Restores an old history file to the trend system.
TrnBrowseClose	Closes a trend browse session.
TrnBrowseFirst	Gets the oldest trend entry.
TrnBrowseGetField	Gets the field indicated by the cursor position in the browse session.
TrnBrowseNext	Gets the next trend entry in the browse session.
TrnBrowse- seNumRecords	Returns the number of records in the current browse session.
TrnBrowseOpen	Opens a trend browse session.
TrnBrowsePrev	Gets the previous trend entry in the browse session.
TrnClientInfo	Gets information about the trend that is being displayed at the AN point.
TrnComparePlot	Prints two trends (one overlaid on the other), each with up to four trend tags.
TrnDelete	Deletes a trend created by the TrnNew() function.
TrnDelHistory	Deletes an old history file from the trend system.

TrnEcho	Enables and disables the echo on the trend display.
TrnEventGetTable	Stores event trend data and the associated time stamp in an event table and time table, for a specified trend tag.
TrnEventGetTableMS	Stores event trend data and time data (including milliseconds) for a specified trend tag.
TrnEventSetTable	Sets trend data from a table, for a specified trend tag.
TrnEventSetTableMS	Sets event trend data and time data (including milliseconds) for a specified trend tag.
TrnExportClip	Exports trend data to the clipboard.
TrnExportCSV	Exports trend data to a file in CSV (comma separated values) format.
TrnExportDBF	Exports trend data to a file in dBASE III format.
TrnExportDDE	Exports trend data to applications via DDE.
TrnFlush	Flushes the trend to disk.
TrnGetBufEvent	Gets the event number of a trend at an offset for a pen.
TrnGetBufTime	Gets the trend time at an offset for a pen.
TrnGetBufValue	Gets the trend value at an offset for a pen.
TrnGetCluster	Gets the name of the cluster the trend graph is associated with.
TrnGetCursorEvent	Gets the event number of a trend at the trend cursor.
TrnGetCursorMSTime	Gets the time (in milliseconds from the previous midnight) at a trend cursor for a specified pen.
TrnGetCursorPos	Gets the position of the trend cursor.
TrnGetCursorTime	Gets the time/date at the trend cursor.
TrnGetCursorValue	Gets the current trend cursor value of a pen.
TrnGetCursorValueStr	Gets the current trend cursor value of a pen as a formatted string.

TrnGetDefScale	Gets the default engineering zero and full scales of a trend tag.
TrnGetDisplayMode	Gets the display mode of a trend.
TrnGetEvent	Gets the event number of a trend at a percentage along the trend.
TrnGetFormat	Gets the format of a pen.
TrnGetGatedValue	Returns the internally stored value for <GATED>.
TrnGetInvalidValue	Returns the internally stored value for <TRN_NO_VALUES>.
TrnGetMode	Gets the display mode of trends (historical or real-time).
TrnGetMSTime	Gets the time (in milliseconds from the previous midnight) of the trend (plotted by a specified pen) at a percentage along the trend, using the time and date of the latest sample displayed.
TrnGetPen	Gets the trend tag of a pen.
TrnGetPenComment	Gets the comment of a trend pen.
TrnGetPenFocus	Gets the number of the pen currently in focus.
TrnGetPenNo	Gets the pen number of a pen name.
TrnGetPeriod	Gets the display period of a trend.
TrnGetScale	Gets the scale of a pen.
TrnGetScaleStr	Gets the scale of a pen as a formatted string.
TrnGetSpan	Gets the span time of a trend.
TrnGetTable	Stores trend data in an array.
TrnGetTime	Gets the time/date of a pen.
TrnGetUnits	Gets the data units of a trend pen.
TrnInfo	Gets the configured values of a trend tag.
TrnIsValidValue	Determines whether a logged trend value is <VALID>, <GATED>, or invalid <TRN_NO_VALUES>.

TrnNew	Creates a new trend at run time.
TrnPlot	Prints a plot of one or more trend tags.
TrnPrint	Prints a trend that is displayed on the screen.
TrnSam-plesConfigured	Gets the number of samples configured for the currently displayed trend.
TrnScroll	Scrolls a trend pen.
TrnSelect	Sets up a page for a trend.
TrnSetCursor	Moves the trend cursor a specified number of samples.
TrnSetCursorPos	Moves the trend cursor to the given x-axis position.
TrnSetDisplayMode	Specifies how trend samples will be displayed on the screen.
TrnSetEvent	Sets the start event of a trend pen.
TrnSetPen	Sets a trend pen to a new trend tag.
TrnSetPenFocus	Sets the pen focus.
TrnSetPeriod	Sets the display period (time base) of a trend.
TrnSetScale	Re-scales a pen.
TrnSetSpan	Sets the span time of a trend.
TrnSetTable	Sets trend data from an array.
TrnSetTime	Sets the starting time/date of a pen.

The following trend functions are used on standard trend templates. Use these functions only if you create your own trend templates
 (These functions are written in Cicode and can be found in the include project.)

TrendDsp-CursorComment	Displays the Trend Comment for the currently selected pen.
TrendDspCursorScale	Displays a scale value for the current pen.

TrendDspCursorTag	Displays the tag name of the current pen.
TrendDspCursorTime	Displays the cursor time of the current pen.
TrendDspCursorValue	Displays the cursor value of the current pen.
TrendGetAn	Gets the AN number of the trend under the mouse position.
TrendPopUp	Displays a pop-up trend with the specified trend pens.
TrendRun	Initializes the cursor and rubber-band features on a trend page.
TrendSetDate	Sets the starting date for the pens on a trend.
TrendSetScale	Sets the scale of one or more pens on a trend.
TrendSetSpan	Sets the span time of a trend.
TrendSetTime	Sets the starting time for the pens on a trend.
TrendSetTimebase	Sets a new sampling period for a trend.
TrendWin	Displays a trend page (in a new window) with the specified trend pens.
TrendZoom	Zooms a trend in either one or both axes.

See Also

[Functions Reference](#)

TrendDspCursorScale

Displays a scale value for the current pen in the current pen font.

Syntax

TrendDspCursorScale(AN, Percent)

AN:

The AN of the trend.

Percent:

The percentage of full scale to display for the current pen, as an integer.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorComment](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendDspCursorTag

Displays the trend tag name of the current pen in the pen font.

Syntax

TrendDspCursorTag(AN [, Mode])

AN:

The AN of the trend.

Mode:

An optional argument used to specify whether the trend tag name is displayed with a cluster prefix.

Set:

- 0 display tag without cluster prefix (default)
- 1 display tag with cluster prefix.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorComment](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendDspCursorTime

Displays the cursor time of the current pen in the current pen font.

Syntax

TrendDspCursorTime(AN, Format)

AN:

The AN number of the trend.

Format:

Format of the string:

- 0 - Short time format, hh:mm AM/PM.
- 1 - Long time format, hh:mm:ss AM/PM.
- 2 - Short date format, dd/mm/yy.
- 3 - Long date format, day month year.
- 4 - Time and date, weekday month day year hh:mm:ss AM/PM.
- 5 - Long time period, hh:mm:ss. Time needs to be in seconds.
- 6 - Millisecond time period, hh:mm:ss:xxx ("xxx" represents milliseconds).
Time needs to be in milliseconds.
- 7 - Short time period, hh:mm. Time needs to be in seconds.
- 8 - Long time period, days:hh:mm:sec. Time needs to be in seconds.
- 9 - Extended date format, dd/mm/yyyy.

Note: If *Format* is set to 1, 2, or 3, the mode will be ignored when the sampling period of the trend changes to less than 1 second. Instead, the time is displayed as hh:min:ss.xxx, that is, displayed as mode 6.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorComment](#), [TrendDspCursorTag](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See the built-in trend templates.

See Also

[Trend Functions](#)

TrendDspCursorValue

Display the cursor value of the current pen in the current pen font.

Syntax

TrendDspCursorValue(AN)

AN:

The AN of the trend.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorComment](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendGetAn

Gets the AN number of the trend beneath the current mouse position.

Syntax

TrendGetAn()

Return Value

The AN of the trend, or 0 (zero) if the mouse is not positioned over a valid trend.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendPopUp

Displays a pop-up trend with the specified trend pens. You need to create the trend page with the graphic builder and set the pen names to blank.

Syntax

TrendPopUp(*sPage*, *sTag1* [*sTag2..sTag8*])

sPage:

The name of the trend page (drawn with the Graphics Builder).

sTag1:

The First trend tag to display on the page.

sTag2..sTag8:

The trend tags to display on the page.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageTrend](#), [TrendWin](#), [WinNewAt](#)

Example

Buttons	
Text	Popup Trend
Command	TrendPopUp("MyPop", "PV1", "PV2", "PV3")
Comment	Display a popup trend with three trend pens

See Also

[Trend Functions](#)

TrendRun

Initializes the cursor and rubber-band features on a trend page. This function is included as a Cicode Object on all new trend pages. Only use this function when configuring a trend template that requires this functionality.

Syntax

TrendRun(*iPageType*)

iPageType:

The type of the page:

0 - Normal trend page template

1 - Compare trend page template

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendSetDate](#), [TrendsetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendSetDate

Sets the end date for all pens on a trend. Samples taken after this date will not be displayed. You can enter the date in the *Value* argument, or leave the *Value* blank - a form is then displayed in run time for the operator to enter an end date.

Syntax

TrendSetDate(AN, Value)

AN:

The AN of the trend.

Value:

The new date, as a string. Samples taken after date will not be displayed. This argument is optional.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendSetScale

Sets the scale of the current pen or of all pens on a trend. You can enter a scale in the *Value* argument, or leave the *Value* blank. A form is then displayed in run time for the operator to enter a value for the scale.

Syntax

TrendSetScale(AN, Percent [, Value])

AN:

The AN of the trend.

Percent:

The scale to be set:

0 - Zero scale

100 - Full scale

Value:

An optional value for the scale, as a string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendSetSpan

Sets the span time of the trend. The span time is the time period covered in the trend window. You can enter a span time in the *Value* argument, or leave the *Value* blank - a form is then displayed in run time for the operator to enter a value for the span time.

Syntax

TrendSetSpan(AN [, Value])

AN:

The AN of the trend.

Value:

An optional value for the span time, as a string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetTime](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendSetTime

Sets the end time for all the pens on a trend. Samples taken after this time will not be displayed. You can enter an end time in the *Value* argument, or leave the *Value* blank - a form is then displayed in run time for the operator to enter a value for the end time.

Syntax

TrendSetTime(AN [, Value])

AN:

The AN of the trend.

Value:

An optional value for the end time, as a string. Samples taken after this time will not be displayed.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTimebase](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendSetTimebase

Sets a new sampling period for a trend. You can enter a sampling period in the *Value* argument, or leave the *Value* blank. A form is then displayed in run time for the operator to enter a value for the sampling period.

Syntax

TrendSetTimebase(AN [, Value])

AN:

The AN of the trend.

Value:

An optional value for the sampling period, as a string.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendZoom](#)

Example

See built-in trend templates.

See Also

[Trend Functions](#)

TrendWin

Displays a trend page (in a new window) with the specified trend pens. You need to create the trend page with the graphic builder and set the pen names to blank. You then display that page by calling this function and pass the required trend tags. The function will create a new window with the specified window mode.

Syntax

TrendWin(sPage, X, Y, Mode, sTag1 [, sTag2..sTag8])

sPage:

The name of the trend page (drawn with the Graphics Builder).

X:

The x pixel coordinate of the top left corner of the window.

Y:

The y pixel coordinate of the top left corner of the window.

Mode:

The mode of the window:

0 - Normal page.

1 - Page child window. The window is closed when a new page is displayed, for example, when the PageDisplay() or PageGoto() function is called. The parent is the current active window.

2 - Window child window. The window is closed automatically when the parent window is freed with the WinFree() function. The parent is the current active window.

4 - No re-size. The window is displayed with thin borders and no maximize/minimize icons. The window cannot be re-sized.

8 - No icons. The window is displayed with thin borders and no maximize/minimize or system menu icons. The window cannot be re-sized.

16 - No caption. The window is displayed with thin borders, no caption, and no maximize/minimize or system menu icons. The window cannot be re-sized.

32 - Echo enabled. When enabled, keyboard echo, prompts, and error messages are displayed on the parent window. This mode should only be used with child windows (for example, Mode 1 and 2).

64 - Always on top.

128 - Open a unique window. This mode helps prevent this window from being opened more than once.

256 - Display the entire window. This mode commands that no parts of the window will appear off the screen

512 - Open a unique Super Genie. This mode helps prevent a Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

1024 - Disables dynamic resizing of new window, overriding the setting of the [Page] DynamicSizing parameter.

You can select multiple modes by adding modes together (for example, set Mode to 9 to open a page child window without maximize, minimize, or system menu icons).

sTag1:

The first trend tag to display on the page.

sTag2..sTag8:

The trend tags to display on the page.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[PageTrend](#), [TrendPopUp](#), [WinNew](#)

Example

Buttons	
Text	Trend Window
Command	TrendWin("MyTrend", 0, 0, 4, "PV1", "PV2", "PV3")
Comment	Display a trend page in a new window with no maximize and minimize icons

See Also

[Trend Functions](#)

TrendZoom

"Zooms" a specified trend in either one or both axes. Set the zoom values (*TimeZoom* and/or *ScaleZoom*) to greater than one to "zoom in" or to less than one to "zoom out".

If you specify a destination AN, you can zoom one trend (at *SourceAn*) onto another (at *DestAn*), in the same way as on the standard zoom trend page.

Syntax

TrendZoom(*SourceAn*, *TimeZoom*, *ScaleZoom* [, *DestAn*])

SourceAn:

The AN on which the source trend is located.

TimeZoom:

The scale by which the Time axis will be changed (as a real number).

ScaleZoom:

The scale by which the Scale axis will be changed (as a real number).

DestAn:

The AN on which the destination or target trend is located. If you do not enter a DestAn, it is set to the same AN as SourceAn.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrendDspCursorScale](#), [TrendDspCursorTag](#), [TrendDspCursorTime](#), [TrendDspCursorValue](#), [TrendGetAn](#), [TrendRun](#), [TrendSetDate](#), [TrendSetScale](#), [TrendSetSpan](#), [TrendSetTime](#), [TrendSetTimebase](#)

Example

```
TrendZoom(30, 2.0, 2.0);
/* Zoom in by a factor of 2 on both the time and scale axes. */
TrendZoom(30, 0.5, 0.5);
/* Zoom out by a factor of 2 on both the time and scale axes. */
```

See Also

[Trend Functions](#)

TrnAddHistory

Adds an old (backed up) trend history file to the trend system so that its data can be used. When you back-up a trend file, change its extension so that it indicates the age of the file's trend data (for example, the month and year).

An archived trend file does not need to reside in the same directory as existing active trends. CitectSCADA retrieves the trend name from the header of the specified file and adds its data to the trend history. Please be aware that only a reference to the archived file, and not the file itself, is kept in the trend history. Therefore, care needs to be taken if using this function to access archived files residing on removable storage media. When you remove the media, the archived data is no longer available for display.

This function can only be used if the Trend Server is on the current machine. When the Trend Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

TrnAddHistory(*FileName* [, *ClusterName*])

FileName:

The file name and directory path of an old history file. The old file does not need to reside in the same directory as existing active trends to be restored.

ClusterName:

Specifies the name of the cluster in which the Trend Server resides. This is optional if you have one cluster or are resolving the trend server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnDelHistory](#), [MsgRPC](#)

Example

```
TrnAddHistory("C:\CITECT\DATA\TR1.MAY91");
! Adds the file TR1.MAY91 to the trend system.
```

See Also

[Trend Functions](#)

TrnBrowseClose

The TrnBrowseClose function terminates an active data browse session and cleans up all resources associated with the session.

This function is a non-blocking function. It does not block the calling Cicode task.

TrnBrowseClose(*iSession*)

iSession

The handle to a browse session previously returned by a TrnBrowseOpen call.

Return Value

0 (zero) if the trend browse session exists, otherwise an [error](#) is returned.

Related Functions

[TrnBrowseFirst](#), [TrnBrowseGetField](#), [TrnBrowseNext](#), [TrnBrowseNumRecords](#), [TrnBrowseOpen](#), [TrnBrowsePrev](#)

See Also

[Trend Functions](#)

TrnBrowseFirst

The TrnBrowseFirst function places the data browse cursor at the first record.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnBrowseFirst(*iSession*)

iSession

The handle to a browse session previously returned by a TrnBrowseOpen call.

Return Value

0 (zero) if the trend browse session exists, otherwise an [error](#) is returned.

Related Functions

[TrnBrowseClose](#), [TrnBrowseGetField](#), [TrnBrowseNext](#), [TrnBrowseNumRecords](#), [TrnBrowseOpen](#), [TrnBrowsePrev](#)

See Also

[Trend Functions](#)

TrnBrowseGetField

The TrnBrowseGetField function retrieves the value of the specified field from the record the data browse cursor is currently referencing.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

TrnBrowseGetField(*iSession*, *sFieldName*)

iSession

The handle to a browse session previously returned by a TrnBrowseOpen call.

sFieldName

The name of the field that references the value to be returned. Supported fields are:

ACQERROR, AREA, EXPRESSION, FILENAME, FILES, LSL, PRIV, RANGE,
SDEVIATION, SPCFLAG, STORMETHOD, SUBGRPSIZE, TIME,
TRIGGER, USL, XDOUBLEBAR.

See [Browse Function Field Reference](#) for information about fields.

Return Value

The value of the specified field as a string. An empty string may or may not be an indication that an error has been detected. The last error should be checked in this instance to determine if an error has actually occurred.

Related Functions

[TrnBrowseClose](#), [TrnBrowseFirst](#), [TrnBrowseNext](#), [TrnBrowseNumRecords](#), [TrnBrowseOpen](#), [TrnBrowsePrev](#)

Example

```
STRING fieldValue = "";
STRING fieldName = "TYPE";
INT errorCode = 0;
...
fieldValue = TrnBrowseGetField(iSession, sFieldName);
IF fieldValue <> "" THEN
    // Successful case
ELSE
    // Function did not succeed
END
...
```

See Also

[Trend Functions](#)

TrnBrowseNext

The TrnBrowseNext function moves the data browse cursor forward one record. If you call this function after you have reached the end of the records, error 412 is returned (Databrowse session EOF).

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnBrowseNext(*iSession*)

iSession

The handle to a browse session previously returned by a TrnBrowseOpen call.

Return Value

0 (zero) if the trend browse session exists, otherwise an [error](#) is returned.

Related Functions

[TrnBrowseClose](#), [TrnBrowseFirst](#), [TrnBrowseGetField](#), [TrnBrowseNumRecords](#), [TrnBrowseOpen](#), [TrnBrowsePrev](#)

See Also

[Trend Functions](#)

TrnBrowseNumRecords

The TrnBrowseNumRecords function returns the number of records that match the filter criteria.

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

TrnBrowseNumRecords(*iSession*)

iSession

The handle to a browse session previously returned by a TrnBrowseOpen call.

Return Value

The number of records that have matched the filter criteria. A value of 0 denotes that no records have matched. A value of -1 denotes that the browse session is unable to provide a fixed number. This may be the case if the data being browsed changed during the browse session.

Related Functions

[TrnBrowseClose](#), [TrnBrowseFirst](#), [TrnBrowseGetField](#), [TrnBrowseNext](#), [TrnBrowseOpen](#), [TrnBrowsePrev](#)

Example

```
INT numRecords = 0;
...
numRecords = TrnBrowseNumRecords(iSession);
IF numRecords <> 0 THEN
    // Have records
ELSE
    // No records
END
...
```

See Also

[Trend Functions](#)

TrnBrowseOpen

The TrnBrowseOpen function initiates a new browse session and returns a handle to the new session that can be used in subsequent data browse function calls.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnBrowseOpen(*sFilter*, *sFields* [, *sClusters*])

sFilter

A filter expression specifying the records to return during the browse. An empty string indicates that all records will be returned. Where a fieldname is not specified in the filter, it is assumed to be tagname. For example, the filter "AAA" is equivalent to "name=AAA".

sFields

Specifies via a comma delimited string the columns to be returned during the browse. An empty string indicates that the server will return all available columns. Supported fields are:

ACQERROR, AREA, EXPRESSION, FILENAME, FILES, FORMAT, LSL,
PERIOD, PRIV, RANGE, SDEVIATION, SPCFLAG, STORMETHOD, SUB-
GRPSIZE, TAGGENLINK, TIME, TRIGGER, USL, XDOUBLEBAR.

See [Browse Function Field Reference](#) for information about fields.

sClusters

An optional parameter that specifies via a comma delimited string the subset of the clusters to browse. An empty string indicates that the connected clusters will be browsed.

Return Value

Returns an integer handle to the browse session. Returns -1 on error.

The returned entries will be ordered alphabetically by name.

Related Functions

[TrnBrowseClose](#), [TrnBrowseFirst](#), [TrnBrowseGetField](#), [TrnBrowseNext](#), [TrnBrowseNumRecords](#), [TrnBrowsePrev](#)

Example

```

INT iSession;
...
iSession = TrnBrowseOpen("NAME=ABC*", "NAME,TYPE",
"ClusterA,ClusterB");
IF iSession <> -1 THEN
    // Successful case
ELSE
    // Function did not succeed
END
...

```

See Also

[Trend Functions](#)

TrnBrowsePrev

The TrnBrowsePrev function moves the data browse cursor back one record. If you call this function after you have reached the beginning of the records, error 412 is returned (Databrowse session EOF).

This function is a non-blocking function. It does not block the calling Cicode task.

Syntax

TrnBrowsePrev(*iSession*)

iSession

The handle to a browse session previously returned by a TrnBrowseOpen call.

Return Value

0 (zero) if the trend browse session exists, otherwise an [error](#) is returned.

Related Functions

[TrnBrowseClose](#), [TrnBrowseFirst](#), [TrnBrowseGetField](#), [TrnBrowseNext](#), [TrnBrowseNumRecords](#), [TrnBrowseOpen](#)

See Also

[Trend Functions](#)

TrnClientInfo

Gets information about the trend that is being displayed at the AN point.

Syntax

TrnClientInfo(AN, Pen, Type, Data, Error)

hAN:

The AN point number at which the trend is displayed.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Type:

The type of information you would like returned.

1 - The number of samples configured for the trend. If you select Type 1, the *Data* argument is ignored.

Data:

For future enhancements; is currently ignored.

Error:

An output argument. If the function is successful, the error is set to 0 (zero). If unsuccessful, an error value is set, and a hardware alarm is triggered.

Return Value

The requested information (as a string) if successful, otherwise a hardware alarm is generated.

Related Functions

[TrnInfo](#)

Example

```
INT nError;
INT nSamples;
INT nTime;
!Gets the number of samples configured for the current pen of the
trend displayed at AN 30.
nSamples = TrnClientInfo(30, 0, 1, "", nError);
IF nError = 0 THEN
    nTime = nSamples * 50;
ELSE
    nTime = 0;
```

```
END
```

See Also

[Trend Functions](#)

TrnComparePlot

Prints two trends, one overlaid on the other. Each trend can have up to four tags configured on it. The significance of this type of plot is that the two trends show different times and display periods. It is possible to compare a trend tag or tags over different time slots. Each trend line is drawn with a different pen style and marker as appropriate. The trend plot includes a comment and a legend, and you can specify the vertical high and low scales.

For more advanced trend plotting, you can use the low-level plot functions.

Syntax

TrnComparePlot(*sPort*, *sTitle*, *sComment*, *AN*, *iMode*, *nSamples*, *iTime1*, *rPeriod1*, *iTime2*, *rPeriod2*, *Tag1.....Tag8*, *rLoScale1*, *rHiScale1*,...,*rLoScale8*, *rHiScale8*)

sPort:

The name of the printer port to which the plot will be printed. This name needs to be enclosed within quotation marks. For example LPT1:, to print to the local printer, or \\Pserver\canon1 using UNC to print to a network printer.

sTitle:

The title of the trend plot.

sComment:

The comment that is to display beneath the title of the trend plot. You do not have to enter a comment.

AN:

Sets the display mode. A value of 0 causes the default display mode to be used. Otherwise, the display mode of the specified AN is set.

iMode:

The color mode of the printer.

0 - black and white (default)

1 - Color

nSamples:

The number of data points on the plot.

iTime1:

The end point in time (the most recent point) for the first trend.

rPeriod1:

The period (in seconds) of the first trend. This can differ from the actual trend period. If you do not enter a period, it defaults to the sample period of Tag1.

iTime2:

The end point in time (the most recent point) for the second trend.

rPeriod2:

The period (in seconds) of the second trend. This can differ from the actual trend period. If you do not enter a period, it defaults to the sample period of Tag5.

Tag1 . . . Tag8:

The trend tags for the plot. Tags 1 to 4 needs to be for the first trend, and tags 5 to 8 needs to be for the second.

rLoScale1, HiScale1,...LoScale8, HiScale8

The minimum and maximum on the vertical scale for the trend line of each of the tags (Tag1 . . . Tag8)

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnPlot](#), [TrnPrint](#), [PlotOpen](#) [SPCPlot](#)

Example

```
/* Prints two black and white trends (one overlaid on the other)
to LPT1, comparing the trend lines of one trend tag (Feed_Flow) at
different times. The first trend line has a starting time of
12 noon, on 11/12/96, and the second has a starting time of 9am, on
11/10/96. Both contain 200 sample points, and have a period of 2
seconds. Both trend lines will be on a vertical scale of 10-100.
*/
INT Time;
INT RefTime;
Time = StrToDate("11/12/96") + StrToTime("12:00:00");
RefTime = StrToDate("11/10/96") + StrToTime("09:00:00");
TrnComparePlot("LPT1:", "Citect Flow Comparison Plot", "Comparison
with Reference", 0,
```

```
0,200,Time,2,RefTime,2,"Feed_Flow","","","","","Feed_Flow","","","","",
10,100,0,0,0,0,0,0,10,100);
```

See Also[Trend Functions](#)**TrnDelete**

Deletes a trend that is displayed on a current page. This trend may have been created by the TrnNew() function or by a trend object.

Syntax**TrnDelete(AN)***AN:*

The AN where the trend is located.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[TrnNew](#)**Example**

```
TrnDelete(20);
! Deletes the trend at AN20.
```

See Also[Trend Functions](#)**TrnDelHistory**

Removes a trend history file that has been added to the trend system by the TrnAddHistory() function. This function does not delete the file completely, it only disconnects it from the historical trend system.

This function can only be used if the Trend Server is on the current machine. When the Trend Server is not in the calling process, this function will become blocking and cannot be called from a foreground task. In this case, the return value will be undefined and a Cicode hardware alarm will be raised.

Syntax

TrnDelHistory(FileName [, ClusterName])

FileName:

The trend history file to disconnect from the historical trend system.

ClusterName:

Specifies the name of the cluster in which the Trend Server resides. This is optional if you have one cluster or are resolving the trend server via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnAddHistory](#), [MsgRPC](#)

Example

```
TrnDelHistory("C:\CITECT\DATA\TR1_91.MAY");
! Disconnects the file from the trend system.
```

See Also

[Trend Functions](#)

TrnEcho

Enables and disables the echo of the trend display. Use this function when you need to make many changes to a trend display, so that the display updates only once. You should first disable the trend echo, do all the trend changes, and then enable the echo to show the changes.

Syntax

TrnEcho(AN, nMode)

AN:

The animation number of the trend.

nMode:

The mode of the echo:

0 - Disable echo of the trend display.

1 - Enable echo of the trend display, to show changes.

Return Value

The current echo mode, otherwise 0 (zero) is returned, and an error code is set. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnSetScale](#), [TrnSetPeriod](#)

Example

```

! Disable echo of trend display at AN40
TrnEcho(40,0);
! Change the scales on pens 1 and 2
TrnSetScale(40,1,0,-1000);
TrnSetScale(40,1,100,-1000);
TrnSetScale(40,2,0,-1000);
TrnSetScale(40,2,100,-1000);
! Enable echo to show changes to the display
TrnEcho(40,1);

```

See Also

[Trend Functions](#)

TrnEventGetTable

Stores event trend data in an event table and the associated time stamp in a time table, for a specified trend tag. Data is stored at the specified *Period*, working backwards from the starting point specified by *EventNo*. If *Period* differs from the trend period configured in the Trend Tags database, the values to be stored are calculated from the trend data. Values are either averaged or interpolated.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnEventGetTable(*Tag*, *EventNo*, *Period*, *Length*, *Table*, *TimeTable*, *Mode* [, *ClusterName*])

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is ClusterName.Tag).

EventNo:

The starting event number for entries in the trend table.

Period:

The time difference between tabulated trend values (in seconds). For example, if you set this period to 30 seconds, CitectSCADA will get the last trend value (sampled at the end of the trend section), then get the trend value that was sampled 30 seconds before that, and so on until it reaches the start time of the trend section.

If this period is different to the trend's sampling period, the trend values will be averaged or interpolated. Set to 0 (zero) to default to the actual trend period.

Length:

The number of trend values to store in the trend table, from 1 to the maximum number of items in the table.

Table:

The Cicode array in which the trend data is stored. You can enter the name of an array here (see the example).

TimeTable:

The table of integer values in which the time stamp is stored.

Mode:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you want from the list below, adding their associated numbers together. The resulting integer is the `DisplayMode` parameter for that trend.

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Ordering trend sample options:

- 0 - Order returned trend samples from oldest to newest.
- 2 - Order returned trend samples from newest to oldest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) x 4096.

The options listed in each group are mutually exclusive. The default value for each Display Mode is 258 (0 + 2 + 256).

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The actual number of samples read. The return value is 0 if an error is detected. You can call the IsError() function to get the actual [error](#) code. If EventNo is 0 (zero) then the EventNo will be set to the current event number.

Related Functions

[TrnEventSetTable](#), [TrnGetEvent](#), [TrnGetDisplayMode](#)

See Also

[Trend Functions](#)

TrnEventGetTableMS

Stores event trend data and time data (including milliseconds) for a specified trend tag. The event trend data is stored in an event table, and the time stamp in time tables. Data is stored at the specified *Period*, working backwards from the starting point specified by *EventNo*. If *Period* differs from the trend period configured in the Trend Tags database, the values to be stored are calculated from the trend data. Values are either averaged or interpolated.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnEventGetTableMS(*Tag*, *EventNo*, *Period*, *Length*, *Table*, *TimeTable*, *Mode*, *MSTimeTable*, *[ClusterName]*)

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is ClusterName.Tag).

EventNo:

The starting event number for entries in the trend table.

Period:

The time difference between tabulated trend values (in seconds). For example, if you set this period to 30 seconds, CitectSCADA will get the last trend value (sampled at the end of the trend section), then get the trend value that was sampled 30 seconds before that, and so on until it reaches the start time of the trend section.

If this period is different to the trend's sampling period, the trend values will be averaged or interpolated. Set to 0 (zero) to default to the actual trend period.

Length:

The number of trend values to store in the trend table, from 1 to the maximum number of items in the table.

Table:

The Cicode array in which the trend data is stored. You can enter the name of an array here (see the example).

TimeTable:

The table of integer values in which the time stamp is stored.

Mode:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you wish to use from those listed below, adding their associated numbers together. The resulting integer is the *DisplayMode* parameter for that trend.

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Ordering trend sample options:

- 0 - Order returned trend samples from oldest to newest.
- 2 - Order returned trend samples from newest to oldest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) x 4096.

Note: Options listed in each group are mutually exclusive. The default value for each Display Mode is 258 (0 + 2 + 256).

MSTimeTable:

The table of integer values in which the millisecond component of the time stamp is stored.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The actual number of samples read. The return value is 0 if an error is detected. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnEventSetTableMS](#) [TrnGetEvent](#), [TrnEventGetTable](#)

See Also

[Trend Functions](#)

TrnEventSetTable

Adds new event to a trend, or overwrites existing points with new values.

To add new events, set 'EventNo' to zero. The events are inserted at a point determined by the time stamp associated with each event. If the timestamp of a new event is identical to that of an existing event, the new event will overwrite the old one.

To overwrite specific existing events, set 'EventNum' to the *last* event number of the block of events to be overwritten, and set the times of the new events to zero.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnEventSetTable(*Tag*, *EventNo*, 0, *Length*, *Table*, *TimeTable* [, *ClusterName*])

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is ClusterName.Tag).

EventNo:

Event Number:

- When adding new events to a trend, set *EventNo* to 0.
- When overwriting existing values, set *EventNo* to the last event number to be overwritten. For example, if overwriting events 100 to 110, set *EventNo* to 110.

Length:

The number of trend values in the trend table.

Table:

The table of floating-point values in which the trend data is stored. You can enter the name of an array here (see the example).

TimeTable:

The table of integer values in which the time stamp is stored. If you would like events to stay in correct time-stamp order, sort the values in this table in ascending order. When *EventNo* is non-zero the values in this table may be zero. This will result in the values of the requested events being overwritten but the time-stamps staying the same.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The actual number of samples written. The return value is 0 if an error is detected. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnEventGetTable](#)

Example

```
REAL TrendTable1[100];
INT TimeTable[100];
/* Defines an array of a maximum of 100 entries. Assume that
TrendTable1 has been storing data from a source. */
TrnEventSetTable("OP1",nEventNo, 1,10,TrendTable1[0],
TimeTable[0], "ClusterXYZ");
/* A set of 10 trend data values are set for the OP1 trend tag. */
```

See Also

[Trend Functions](#)

TrnEventSetTableMS

Sets event trend data and time data (including milliseconds) for a specified trend tag.

The event trend data is set in an event table, and the time stamp in time tables. Data is set at the period specified, working backwards from the starting point specified by *EventNo*.

If the period of setting data differs from the trend period (defined in the Trend Tags database), the values to be set are calculated from the trend data, either averaged or interpolated. The user needs to have the correct privilege (as specified in the Trend Tags form), otherwise the data is not written.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnEventSetTableMS(*Tag*, *EventNo*, *Period*, *Length*, *Table*, *TimeTable*, *MSTimeTable* [, *ClusterName*])

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is *ClusterName.Tag*).

EventNo:

Event Number:

- When adding new events to a trend, set *EventNo* to 0.
- When overwriting existing values, set *EventNo* to the last event number to be overwritten. For example, if overwriting events 100 to 110, set *EventNo* to 110.

Period:

This will be the interval (in seconds) between trend values when they are set (that is it will be the perceived sampling period for the trend). This period can differ from the actual trend period. Set to 0 (zero) to default to the actual trend period.

Length:

Number of trend values in the trend table.

Table:

Table of floating-point values in which the trend data is stored. You can enter the name of an array here (see example).

TimeTable:

Table of integer values in which the time stamp is stored. If you would like events to stay in correct time-stamp order, sort the values in this table in ascending order. When EventNo is non-zero the values in this table may be zero. This will result in the values of the requested events being overwritten but the timestamps staying the same.

MSTimeTable:

The table of integer values in which the millisecond component of the time stamp is stored.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The actual number of samples written. The return value is 0 if an error is detected. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnEventGetTable](#)

Example

```
// Arrays for trend data
REAL garSingleValue[1];
INT ganSingleTime[1];
INT ganSingleMS[1];
// Push the data in the trend system
INT
FUNCTION LogTrend(STRING sTagName, REAL rValue, INT nDateTime, INT
nMS)
    INT nSamplesWritten;
    garSingleValue[0] = rValue;
    ganSingleTime[0] = nDateTime;
    ganSingleMS[0] = nMS;
    nSamplesWritten      = TrnEventSetTableMS
(sTagName, 0, 0, 1, garSingleValue[0], ganSingleTime[0], ganSingleMS[0], "Clus-
terXYZ");
    RETURN nSamplesWritten;
END
```

See Also

[Trend Functions](#)

[TrnExportClip](#)

Exports trend data to the Windows Clipboard. The data is set at the specified *Time* and *Period*, and listed from earliest to latest. Any gated or invalid data is written as 0.0.

Data is stored as a grid, with each row time-stamped. The first column/field is the date, followed by the time, followed by the tags 1 to 8.

You can use the *ClipMode* argument to make the output more useful. For example, to paste the data into Excel, use *ClipMode* 2 for CSV format. If you use *ClipMode* 1 or 3, the default paste menu option causes data to be pasted into the user's spreadsheet as text. If you use *ClipMode* 3, use the Paste Special option to paste the required format. Please be aware that not all packages support multiple clipboard formats in this way.

Syntax

TrnExportClip(*Time*, *Period*, *Length*, *Mode*, *ClipMode*, *sTag1* ... *sTag8*, *iDisplayMode1* ... *iDisplayMode 8*)

Time:

The starting time for the data being exported.

Period:

The period (in seconds) of the entries being exported. (This period can differ from the actual trend period.)

Length:

The length of the data table, that is The number of rows of samples to be exported. for example if you put the length as 12, and you declare two tags to be exported, you get a grid with 12 rows of samples. Each row has values for each of the two tags making a total of 24 samples.

Mode:

The format mode to be used:

Periodic trends

- 1 - Export the Date and Time, followed by the tags.
- 2 - Export the Time only, followed by the tags.
- 4 - Ignore any invalid or gated values. (Only supported for periodic trends.)
- 8 - The time returned will have millisecond accuracy.

Event trends

- 1 - Export the Time, Date and Event Number, followed by the tags.
- 2 - Export the Time and Event Number, followed by the tags.
- 8 - The time returned will have millisecond accuracy.

ClipMode:

The format for the data being exported.

1 - Text

2 - CSV

You can add these modes for a combination of formats.

sTag1 ... sTag8:

The trend tag names for the data being exported.

iDisplayMode1 ... iDisplayMode8:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you wish to use from those listed below, adding their associated numbers together. The resulting integer is the DisplayMode parameter for that trend.

By default, this argument is set to 3 (see the details for options 1 and 2 below).

Invalid/Gated trend options:

0 - Convert invalid/gated trend samples to zero.

1 - Leave invalid/gated trend samples as they are.

Invalid and gated samples that are not converted to zero will appear in the destination file as the string "na" (for invalid) or "gated".

Ordering trend sample options:

0 - Order returned trend samples from newest to oldest.

2 - Order returned trend samples from oldest to newest.

Condense method options:

0 - Set the condense method to use the mean of the samples.

4 - Set the condense method to use the minimum of the samples.

8 - Set the condense method to use the maximum of the samples.

12 - Set the condense method to use the newest of the samples.

Stretch method options:

0 - Set the stretch method to step.

128 - Set the stretch method to use a ratio.

256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) x 4096.

Display as Periodic options:

0 - Display according to trend type.

1048576 - Display as periodic regardless of trend type.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Note: If using this function to export trends by using event numbers, you need to specify a valid event number in the Time argument, rather than a time.

Related Functions

[ClipSetMode](#), [TrnExportCSV](#)

Example

```
TrnExportClip(TimeCurrent(), 2, 60 * 60/2, 2, 3, "Feed",
"Weight");
/* Export the last hour of data from the trend tags Feed and Weight
to the clipboard in both Text and CSV formats. Be aware that the 60 *
60/2 is a decomposed way of writing 1800, which is the number of 2
second samples in 1 hour. */
```

See Also

[Trend Functions](#)

TrnExportCSV

Exports trend data to a file in CSV (Comma Separated Variable) format. The data is set at the specified *Time* and *Period*, and listed from earliest to latest. Any gated or invalid data is written as 0.0.

Data is stored as a grid, with each row time-stamped. The first column/field is the date, followed by the time, followed by the tags 1 to 8.

You can view the CSV file with a text editor, and import the file directly into other packages such as Excel for data analysis and presentation.

If you're using this function to export trends by using event numbers, you need to specify a valid event number in the Time argument, rather than a time.

Syntax

TrnExportCSV(*Filenname*, *Time*, *Period*, *Length*, *Mode*, *sTag1* ... *sTag8*, *iDisplayMode1* ... *iDisplayMode 8*)

Filenname:

The name of the destination path and file.

Time:

The starting time for the data being exported.

Period:

The period (in seconds) of the entries being exported. (This period can differ from the actual trend period.)

Length:

The length of the data table, that is, The number of rows of samples to be exported. for example if you put the length as 12, and you declare two tags to be exported, you get a grid with 12 rows of samples. Each row has values for each of the two tags making a total of 24 samples.

Mode:

The format mode to be used:

Periodic trends

- 1 - Export the Date and Time, followed by the tags.
- 2 - Export the Time only, followed by the tags.
- 4 - Ignore any invalid or gated values. (This mode is only supported for periodic trends.)
- 8 - The time returned will have millisecond accuracy.

Event trends

- 1 - Export the Time, Date and Event Number, followed by the tags.
- 2 - Export the Time and Event Number, followed by the tags.
- 8 - The time returned will have millisecond accuracy.

sTag1 ... sTag8:

The trend tag names for the data being exported.

iDisplayMode1 ... iDisplayMode8:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you wish to use from those listed below, adding their associated numbers together. The resulting integer is the DisplayMode parameter for that trend. By default, this argument is set to 3 (see the details for options 1 and 2 below).

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Invalid and gated samples that are not converted to zero will appear in the destination file as the string "na" (for invalid) or "gated".

Ordering trend sample options:

- 0 - Order returned trend samples from newest to oldest.
- 2 - Order returned trend samples from oldest to newest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.
- 12 - Set the condense method to use the newest of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

- n - the number of missed samples that the user wants to gap fill) x 4096.

Options listed in each group are mutually exclusive.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnExportDBF](#), [TrnPrint](#)

Example

```
TrnExportCSV("c:\TrnData.CSV", TimeCurrent(), 2, 60 * 60/2, 2,
"Feed", "Weight");

/* Export the last hour of data from the trend tags Feed and
Weight.
The 60 * 60/2 is a decomposed way of writing 1800, which is the
number of 2 second samples in 1 hour. */
```

See Also

[Trend Functions](#)

TrnExportDBF

Exports trend data to a file in dBASE III format. The data is set at the specified *Time* and *Period*, and listed from earliest to latest. Any gated or invalid data is written as 0.0.

Data is stored as a grid, with each row time-stamped. The first column/field is the date, followed by the time, followed by the tags 1 to 8.

You can import the DBF file directly into other packages such as Excel, for data analysis and presentation.

Syntax

TrnExportDBF(Filename, Time, Period, Length, Mode, sTag1 ... sTag8, iDisplayMode1 ... iDisplayMode 8)

*Filenam*e:

The name of the destination path and file.

Time:

The starting time for the data being exported.

Period:

The period (in seconds) of the entries being exported. (This period can differ from the actual trend period.)

Length:

The length of the data table, that is The number of rows of samples to be exported. for example if you put the length as 12, and you declare two tags to be exported, you get a grid with 12 rows of samples. Each row has values for each of the two tags making a total of 24 samples.

Mode:

The format mode to be used:

Periodic trends

- 1 - Export the Date and Time, followed by the tags.
- 2 - Export the Time only, followed by the tags.
- 4 - Ignore any invalid or gated values. (This mode is only supported for periodic trends.)
- 8 - The time returned will have millisecond accuracy.

Event trends

- 1 - Export the Time, Date and Event Number, followed by the tags.
- 2 - Export the Time and Event Number, followed by the tags.
- 8 - The time returned will have millisecond accuracy.

sTag1 ... sTag8:

The trend tag names for the data being exported. Tag names longer than 10 characters will be truncated, as the standard DBF field format is 10 characters.

iDisplayMode1 ... iDisplayMode8:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you wish to use from those listed below, adding their associated numbers together. The resulting integer is the DisplayMode parameter for that trend. By default, this argument is set to 3 (see the details for options 1 and 2 below).

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Invalid and gated samples that are not converted to zero will appear in the destination file as the string "na" (for invalid) or "gated".

Ordering trend sample options:

- 0 - Order returned trend samples from newest to oldest.
- 2 - Order returned trend samples from oldest to newest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.
- 12 - Set the condense method to use the newest of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) x 4096.

Options listed in each group are mutually exclusive.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TmExportCSV](#), [TrnPrint](#)

Example

```
TrnExportDBF("c:\TrnData.DBF", TimeCurrent(), 2, 60 * 60/2, 2,  
"Feed", "Weight");  
/* Export the last hour of data from the trend tags Feed and  
Weight. Be aware that the 60 * 60/2 is a decomposed way of writing  
1800, which is the number of 2 second samples in 1 hour. */
```

See Also

[Trend Functions](#)

TrnExportDDE

Exports trend data via DDE. The data is set at the specified Time and Period, and listed from earliest to latest. Any gated or invalid data is written as 0.0. Data is stored as a grid, with each row time-stamped. The first column/field is the date, followed by the time, followed by the tags 1 to 8.

You can use the DDEMMode argument to make the output more useful. For example; to paste the data into Excel, use DDEMMode 2 for CSV format. If you use DDEMMode 1, data will be put into the user's spreadsheet as text.

Note: If you're using this function to export trends by using event numbers, you need to specify a valid event number in the Time argument, rather than a time.

Syntax

TrnExportDDE(*sApplication*, *sDocument*, *sTopic*, *Time*, *Period*, *Length*, *Mode*, *DDEMMode*,
sTag1 ... *sTag8*, *iDisplayMode1* ... *iDisplayMode 8*)

sApplication:

The application name to export the data.

sDocument:

The document in the application to export the data.

sTopic:

The topic in the application to export the data. Be aware you may have to use a special topic format to make the data export correctly. See your application documentation for details; For example with Excel you need to specify the matrix of rows and columns as "R1C1:R8C50" depending on the size of the data.

Filename:

The name of the destination path and file.

Time:

The starting time for the data being exported.

Period:

The period (in seconds) of the entries being exported. (This period can differ from the actual trend period.)

Length:

The length of the data table, that is The number of rows of samples to be exported. for example if you put the length as 12, and you declare two tags to be exported, you get a grid with 12 rows of samples. Each row has values for each of the two tags making a total of 24 samples.

Mode:

The format mode to be used:

Periodic trends

- 1 - Export the Date and Time, followed by the tags.
- 2 - Export the Time only, followed by the tags.
- 4 - Ignore any invalid or gated values. (This mode is only supported for periodic trends.)
- 8 - The time returned will have millisecond accuracy.

Event trends

- 1 - Export the Time, Date and Event Number, followed by the tags.
- 2 - Export the Time and Event Number, followed by the tags.
- 8 - The time returned will have millisecond accuracy.

DDEMMode:

The format for the data being exported. CSV format allows the application to separate the data into each individual element, however not every application will support this mode. See your applications documentation for details.

- 1 - Text (default)
- 2 - CSV

sTag1 ... sTag8:

The trend tag names for the data being exported. Tag names longer than 10 characters will be truncated, as the standard DBF field format is 10 characters.

iDisplayMode1 ... iDisplayMode8:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you wish to use from those listed below, adding their associated numbers together. The resulting integer is the DisplayMode parameter for that trend.

By default, this argument is set to 3 (see the details for options 1 and 2 below).

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Invalid and gated samples that are not converted to zero will appear in the destination file as the string "na" (for invalid) or "gated".

Ordering trend sample options:

- 0 - Order returned trend samples from newest to oldest.
- 2 - Order returned trend samples from oldest to newest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.
- 12 - Set the condense method to use the newest of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) x 4096.

Options listed in each group are mutually exclusive.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnExportCSV](#), [TrnExportClip](#), [TrnExportDBF](#)

Example

```
TrnExportDDE("Excel", "data.xls", "R1C1:R61C4", TimeCurrent(), 1,
```

```
60, 2, 2, "Feed", "Weight");
/* Export the last 60 seconds of data from the trend tags Feed and
Weight into Excel at R1C1:R61C4 in CSV formats */
```

See Also[Trend Functions](#)**TrnFlush**

Writes acquired trend data to disk without waiting for the trend buffer to be filled. Citect-SCADA normally buffers the trend data in memory and only writes to disk when required, to give optimum performance. Because this function reduces the performance of the Trends Server, use it only when necessary.

Syntax**TrnFlush**(*Name* [, *ClusterName*])*Name*:

The name of the logging tag (can be prefixed by the name of the cluster that is *ClusterName.Tag*). Set to "*" to flush all trend data.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[TrnSetTable](#)**Example**

```
TrnFlush("Trend1", "ClusterXYZ");
! Forces the Trend1 data to be written to disk.
```

See Also[Trend Functions](#)**TrnGetBufEvent**

Gets the event number of a trend at an offset for a specified pen. This function only operates on an event-based trend.

Syntax

TrnGetBufEvent(AN, Pen, Offset)

AN:

The AN where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Offset:

The trend buffer offset, in samples. The number of samples can range from 0 to the maximum number of samples that can display on the trend - 1.

Return Value

The event number. If *Offset* is not within boundaries, 0 (zero) is returned. If *AN* or *Pen* is invalid, 0 (zero) is returned and an [error](#) code is set.

Related Functions

[TrnGetEvent](#), [TrnSetEvent](#), [TrnGetCursorEvent](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetBufEvent(20,0,10));
/* Displays the trend event at offset 10 for the pen currently in
focus. The event will display at AN31. */
```

See Also

[Trend Functions](#)

TrnGetBufTime

Gets the time and date of a trend at an offset for a specified pen. The *Offset* should be a value between 0 (zero) and the number of samples displayed on the trend.

Syntax

TrnGetBufTime(AN, Pen, Offset)

AN:

The AN where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Offset:

The trend buffer offset, in samples. The number of samples can range from 0 to the maximum number of samples that can display on the trend - 1.

Return Value

A time/date variable. If *Offset* is not within boundaries, 0 (zero) is returned. If *AN* or *Pen* is invalid, 0(zero) is returned and an [error](#) code is set.

Related Functions

[TrnGetCursorTime](#)

Example

```

! For the trend at AN20
INT time;
time = TrnGetBufTime(20,0,10);
IF time <> 0 THEN
    DspText(31,0,TimeToStr(time,2));
END
/* Displays the trend date at offset 10 for the pen currently in
focus. The time will display at AN31. */

```

See Also

[Trend Functions](#)

TrnGetBufValue

Gets the value of a trend at an offset for a specified pen. The offset should be a value between -1 and the number of samples displayed on the trend.

Syntax

TrnGetBufValue(AN, Pen, Offset)

AN:

The AN where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Offset:

The trend buffer offset, in samples. The number of samples can range from -1 to the maximum number of samples that can display on the trend minus 1.

-1 means get the last valid value in the display (provided it is less than 1.5 sample periods old).

If there is no invalid or gated sample within the last 1.5 sample periods, it is assumed that a sample has been missed and an invalid trend value is returned. See the [TrnIsValidValue](#) function.

Return Value

The trend value. If the actual value is gated or invalid, the standard invalid or gated values are returned (no [error](#) is set). You can check this return value using TrnIsValidValue().

Related Functions

[TrnGetCursorValue](#), [TrnIsValidValue](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetBufValue(20,0,10):###.#);
/* Displays the trend value at offset 10 for the pen currently in
focus. */
```

See Also

[Trend Functions](#)

TrnGetCluster

Gets the cluster name of a trend graph on a page.

Syntax

TrnGetCluster(AN)

AN:

The AN of the chosen trend graph.

Return Value

The name of the cluster the trend graph is associated with.

See Also

[Trend Functions](#)

TrnGetCursorEvent

Gets the event number of a trend, at the trend cursor position for a specified pen. This function only operates on an event-based trend.

Syntax

TrnGetCursorEvent(AN, Pen)

AN:

The AN where the trend is located.

Pen:

The trend pen number:

- 0 - The pen currently in focus
- 1...8 - Pen1..Pen8

Return Value

The event number, or 0 (zero) if the trend cursor is disabled.

Related Functions

[TrnSetEvent](#), [TrnGetCluster](#), [TrnGetEvent](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetCursorEvent(20,0));
/* Displays the trend event at the cursor for the pen currently in
```

```
focus. The event will display at AN31. */
```

See Also

[Trend Functions](#)

TrnGetCursorMSTime

Gets the time (in milliseconds from the previous midnight) at a trend cursor for a specified pen.

Syntax

TrnGetCursorMSTime(AN, Pen)

AN:

The AN where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Return Value

The number of milliseconds since the previous midnight. If the trend cursor is disabled, 0 (zero) is returned. If *AN* or *Pen* is invalid, 0 (zero) is returned and an [error](#) code is set.

Related Functions

[TrnGetCursorTime](#)

Example

```
! For the trend at AN20
STRING timeStr;
STRING msecStr;
timeStr = TimeToString(TrnGetCursorTime(20,1),2) + " ";
msecStr = TimeToString(TrnGetCursorMSTime(20,1),6);
DspText(31,0,timeStr + msecStr);
! Returns "23/02/01 10:53:22.717"
```

See Also

[Trend Functions](#)

TrnGetCursorPos

Gets the offset of a trend cursor from its origin, in samples.

Syntax

TrnGetCursorPos(AN)

AN:

The AN where the trend is located.

Return Value

The offset of a trend cursor from its origin, in samples, or -1 if the trend cursor is disabled.

Related Functions

[TrnSetCursorPos](#)

Example

```

! For the trend at AN20
! If the trend cursor is disabled
Offset=TrnGetCursorPos(20);
! Sets Offset to -1.
! If the trend cursor is 50 samples from the origin
Offset=TrnGetCursorPos(20);
! Sets Offset to 50.

```

See Also

[Trend Functions](#)

TrnGetCursorTime

Gets the time and date at a trend cursor for a specified pen.

Syntax

TrnGetCursorTime(AN, Pen)

AN:

The AN where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Return Value

A time/date variable. If the trend cursor is disabled, 0 (zero) is returned. If *AN* or *Pen* is invalid, 0 (zero) is returned and an [error](#) code is set.

Related Functions

[TrnGetBufTime](#)

Example

```
! For the trend at AN20
INT time;
time = TrnGetCursorTime(20,1);
DspText(31,0,TimeToStr(time,2));
! Displays the trend cursor date for Pen1.
DspText(32,0,TimeToStr(time,1));
! Displays the trend cursor time for Pen1.
```

See Also

[Trend Functions](#)

TrnGetCursorValue

Gets the value at a trend cursor for a specified pen.

Syntax

TrnGetCursorValue(*AN*, *Pen*)

AN:

The *AN* where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Return Value

The trend value. If the actual value is gated or invalid, the standard invalid or gated values are returned (no [error](#) is set). You can check this return value using *TrnIs-ValidValue()*.

Related Functions[TrnGetBufValue](#)**Example**

```
! For the trend at AN20
DspText(31,0,TrnGetCursorValue(20,0));
! Displays the value at the trend cursor for the focus pen.
```

See Also[Trend Functions](#)**TrnGetCursorValueStr**

Gets the value at a trend cursor for a specified pen. The value is returned as a formatted string using the pen's format specification and (optionally) the engineering units.

Syntax**TrnGetCursorValueStr(AN, Pen, EngUnits)***AN:*

The AN where the trend is located.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

EngUnits:

Engineering units mode:

0 - Do not include the engineering units at the end of the formatted string.

1 - Include the engineering units at the end of the formatted string.

Return Value

The trend value (as a string). If trend data is invalid, or an argument passed to the function is invalid "<na>" is returned. If the actual value is gated (not triggered) "<gated>" is returned. If the trend cursor is disabled, an empty string is returned.

Related Functions[TrnGetCursorValue](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetCursorValueStr(20,0,1));
/* Displays the value at the trend cursor for the focus pen. The
value will display as a formatted string (including the
engineering units).*/
```

See Also

[Trend Functions](#)

TrnGetDefScale

Gets the default engineering zero and full scales of a trend tag.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

TrnGetDefScale(*Tag*, *LoScale*, *HiScale* [, *ClusterName*])

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is "ClusterName.Tag").

LoScale:

The engineering zero scale.

HiScale:

The engineering full scale.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetScale](#), [TrnInfo](#)

Example

```
REAL LoScale;
REAL HiScale;
TrnGetDefScale("PV1",LoScale,HiScale,"ClusterXYZ");
/* Returns engineering zero and full scales of the trend tag
"PV1". */
```

See Also[Trend Functions](#)**TrnGetDisplayMode**

Returns the display mode of the selected trend pen. The display mode is set using TrnSetDisplayMode.

Syntax**TrnGetDisplayMode(AN, PenNumber)***AN:*

The AN of the chosen trend.

PenNumber:

The trend pen number:

0 - The pen currently in focus.

1...8 - Pen1. . .Pen8.

Return Value

An integer representing the trend's display mode:

Invalid/Gated trend options:

0 - Convert invalid/gated trend samples to zero.

1 - Leave invalid/gated trend samples as they are.

Ordering trend sample options:

0 - Order returned trend samples from oldest to newest.

2 - Order returned trend samples from newest to oldest.

Condense method options:

0 - Set the condense method to use the mean of the samples.

4 - Set the condense method to use the minimum of the samples.

8 - Set the condense method to use the maximum of the samples.

12 - Set the condense method to use the newest of the samples.

Stretch method options:

0 - Set the stretch method to step.

128 - Set the stretch method to use a ratio.

256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) $\times 4096$.

Display as Periodic options:

0 - Display according to trend type.

1048576 - display as periodic regardless of trend type.

Related Functions

[TrnSetDisplayMode](#)

Example

```
int DisplayMode = TrnGetDisplayMode (10, 7)
/* Returns The Display Mode of pen 7 for the trend at AN 10.*/
```

See Also

[Trend Functions](#)

TrnGetEvent

Gets the event number of the trend at a percentage along the trend, using the current event as the base point. This function only operates on an event-based trend. The first recorded event (the start event) would be event number 1 and the highest number would be the latest event. The event number is stored in a LONG and would eventually wrap around if you have enough events.

Syntax

TrnGetEvent(AN, Pen, Percent)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Percent:

The percentage of the trend from the starting event, from 0 (the start event) to 100 (the end event).

Return Value

The event number.

Related Functions

[TrnSetEvent](#), [TrnGetCluster](#), [TrnGetCursorEvent](#)

Example

```
/* Display the start event for the current pen of the trend at
AN20. */
DspText(31,0,TrnGetEvent(20,0,0));
```

See Also

[Trend Functions](#)

TrnGetFormat

Gets the format of a trend tag being plotted by a specified pen.

Syntax

TrnGetFormat(AN, Pen, Width, DecPlaces)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Width:

The width of the format.

DecPlaces:

The number of decimal places in the format.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetScale](#), [TrnGetUnits](#)

Example

```
/* If the trend tag being plotted by Pen1 of the trend at AN20 has
   a format of "###.#" */
TrnGetFormat(20,1,Width,DecPlaces);
! Sets Width to 5 and DecPlaces to 1.
```

See Also

[Trend Functions](#)

TrnGetGatedValue

Returns the internally stored value for <GATED>. If the internally stored value changes in the future, you will not need to modify your Cicode, as this function will return the correct value.

Syntax

TrnGetGatedValue()

Return Value

The internally stored value for <GATED>.

Related Functions

[TrnGetInvalidValue](#), [TrnIsValidValue](#)

Example

```
REAL MyTrendValue;
IF MyTrendValue = TrnGetGatedValue() THEN
    Prompt ("This value is <GATED>")
ELSE
    IF MyTrendValue = TrnGetInvalidValue() THEN
        Prompt("This value is <TRN_NO_VALUES>")
    ELSE
        Prompt("Trend value is = " + RealToStr(MyTrendValue, 10, 1));
```

```
    END
END
```

See Also[Trend Functions](#)**TrnGetInvalidValue**

Returns the internally stored value for <INVALID>. If the internally stored value changes in the future, you will not need to modify your Cicode, as this function will return the correct value.

Syntax**TrnGetInvalidValue()****Return Value**

The internally stored value for <INVALID>.

Related Functions[TrnGetGatedValue](#), [TrnIsValidValue](#)**Example**

```
REAL newArray[100];
REAL oldArray[90];
INT trigger;
INT
FUNCTION
DoubleArray()
    INT i;
    FOR i = 0 TO 99 DO
        IF TrnIsValidValue(oldArray[i]) = 1 OR trigger = 0 THEN
            newArray[i] = TrnGetGatedValue();
        ELSE
            IF i >= 90 OR TrnIsValidValue(oldArray[i]) = 2 THEN
                newArray[i] = TrnGetInvalidValue();
            ELSE
                newArray[i] = oldArray[i] * 2;
            END
        END
    END
    RETURN i;
END
```

See Also[Trend Functions](#)

TrnGetMode

Gets the mode (real-time or historical trending) of the trend pen.

Syntax

TrnGetMode(AN, Pen)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Return Value

The current mode, 0 for real-time or 1 for historical.

Related Functions

[TrnScroll](#)

Example

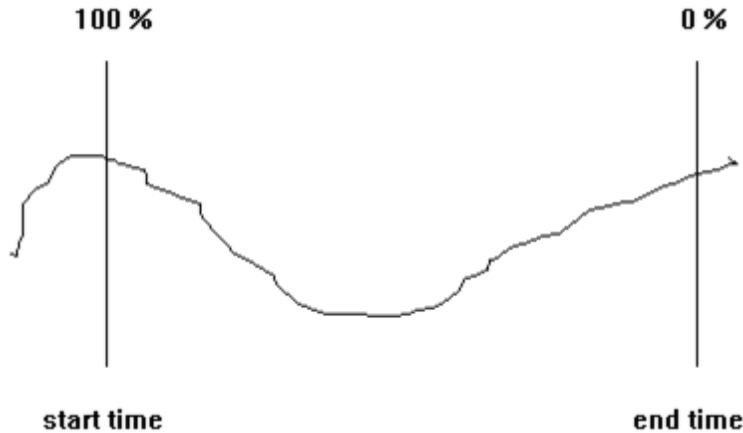
```
! For the trend at AN20
INT Mode;
Mode=TrnGetMode(20,0);
! Gets the current mode of the pen in focus.
IF Mode=0 THEN
    DspText(31,0,"Real Time Trending");
ELSE
    DspText(31,0,"Historical Trending");
END
```

See Also

[Trend Functions](#)

TrnGetMSTime

Gets the time (in milliseconds from the previous midnight) of the trend (plotted by a specified pen) at a percentage along the trend, using the time and date of the right-most sample displayed. The time associated with the right-most sample displayed is known as the end time. The start time is the time of the left-most sample displayed. Percent 0 (zero) will correspond to the end time, and Percent 100 will correspond to the start time



Syntax

TrnGetMSTime(AN, Pen, Percent)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Percent:

The percentage of the trend from the time and date of the right-most sample displayed (end time), from 0 to 100.

Return Value

The number of milliseconds since the previous midnight. Zero (0) is returned if an [error](#) is detected.

Related Functions

[TrnGetTime](#)

Example

```
! For Pen 1 at AN20
STRING timeStr;
STRING msecStr;
timeStr = TimeToString(TrnGetTime(20,1,100),2) + " ";
```

```
msecStr = TimeToString(TrnGetMSTime(20,1,100),6);
DspText(31,0,timeStr + msecStr);
```

returns

```
"23/02/01 10:53:22.717"
```

See Also

[Trend Functions](#)

TrnGetPen

Gets the trend tag being plotted by a specified pen.

Syntax

TrnGetPen(AN, Pen [, Mode])

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Mode:

An optional argument used to specify whether the trend tag name is returned with a cluster prefix.

Set:

- 0 return tag without cluster prefix (default)
- 1 return tag with cluster prefix.

Return Value

The trend tag (as a string) being plotted by *Pen*. If *AN* or *Pen* is invalid, an empty string is returned, and an error code is set. You can call the `IsError()` function to get the actual [error](#) code.

Related Functions

[TrnSetPen](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetPen(20,0,1));
! Displays the trend tag with cluster prefix of the focus pen.
```

See Also

[Trend Functions](#)

TrnGetPenComment

Retrieves the comment of a pen.

Syntax

TrnGetPenComment(AN, Pen)

AN

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Return Value

The comment of the pen as a string.

Related Functions

[TrnGetPen](#)

Example

```
! For the trend at AN18
DspText(31,0,TrnGetPen(18,0));
! Displays the trend comment of the focus pen.
```

See Also

[Trend Functions](#)

TrnGetPenFocus

Gets the number of the pen currently in focus.

Syntax

TrnGetPenFocus(AN)

AN:

The AN of the chosen trend.

Return Value

The pen currently in focus (between 1 and 8). If *AN* is invalid, -1 is returned and an [error](#) code is set.

Related Functions

[TrnSetPenFocus](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetPenFocus(20));
! Displays the pen currently in focus.
```

See Also

[Trend Functions](#)

TrnGetPenNo

Gets the pen number of a pen name. The pens on a trend are either defined in the Page Trends database or set by the TrnSetPen() function.

Syntax

TrnGetPenNo(AN, Tag)

AN:

The AN of the chosen trend.

Tag:

The trend tag.

Return Value

The pen number, or 0 (zero) if an [error](#) is detected.

Related Functions

[TrnSetPen](#)

Example

```
/* Assume that 8 trend fonts, Pen1TrendFont ... Pen8TrendFont are
defined in the Fonts database. The following code will display the
trend tag using the matching font for that pen. */
! For the trend at AN20
STRING sFont;
INT iPen;
iPen = TrnGetPenNo(20,"PV1");
IF 0 < iPen AND iPen < 9 THEN
    sFont = "Pen" + IntToStr(iPen) + "TrendFont";
    DspStr(31,sFont,"PV1");
END
```

See Also

[Trend Functions](#)

TrnGetPeriod

Gets the current display period of a trend. (To obtain the sampling period, use the TrnInfo function.)

Syntax

TrnGetPeriod(AN)

AN:

The AN of the chosen trend.

Return Value

The current display period of a trend (in seconds), or 0 (zero) if an [error](#) is detected.

Related Functions

[TrnSetPeriod](#), [TrnInfo](#)

Example

```
/* For the trend at AN20, get and display the current display
period. */
! If the period is 10 seconds
INT Period;
```

```
STRING Str;
Period=TrnGetPeriod(20);
Str=TimeToStr(Period,5);
DspStr(31,"",Str);
```

See Also

[Trend Functions](#)

TrnGetScale

Gets the display scale of the trend tag being plotted by a specified pen.

Syntax

TrnGetScale(*AN*, *Pen*, *Percent*)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Percent:

The percentage of the full scale, from 0 to 100.

Return Value

The scale of the trend tag being plotted by *Pen*. If *AN* or *Pen* is invalid, 0 (zero) is returned and an [error](#) code is set.

Related Functions

[TrnSetScale](#), [TrnGetDefScale](#)

Example

```
! For the trend at AN20
DspText(31,0,TrnGetScale(20,0,0));
! Displays the zero scale of the focus pen.
DspText(32,0,TrnGetScale(20,0,50));
! Displays the 50% scale of the focus pen.
DspText(33,0,TrnGetScale(20,0,100));
! Displays the full scale of the focus pen.
```

See Also[Trend Functions](#)**TrnGetScaleStr**

Gets the scale of the trend tag being plotted by a specified pen. The value is returned as a formatted string using the pen's format specification and (optionally) the engineering units.

Syntax**TrnGetScaleStr(AN, Pen, Percent, EngUnits)***AN:*

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Percent:

The percentage of the full scale, from 0 to 100.

EngUnits:

Engineering units mode:

0 - Do not include the engineering units at the end of the formatted string.

1 - Include the engineering units at the end of the formatted string.

Return Value

The scale of the trend tag being plotted by *Pen* (as a string). If *AN* or *Pen* is invalid, <na> is returned and an [error](#) code is set.

Related Functions[TrnGetScale](#)**Example**

```
! For the trend at AN20
```

```
DspText(31,0,TrnGetScaleStr(20,0,0,1));
/* Displays the zero scale of the focus pen. The scale displays as
a formatted string (including the engineering units). */
DspText(32,0,TrnGetScaleStr(20,2,50,1));
/* Displays the 50% scale of Pen2. The scale displays as a
formatted string (including the engineering units). */
DspText(33,0,TrnGetScaleStr(20,0,100,0));
/* Displays the full scale of the trend tag being plotted by the
focus pen. The scale displays as a formatted string (excluding the
engineering units). */
```

See Also

[Trend Functions](#)

TrnGetSpan

Gets the span time of a trend (if the span was set by the TrnSetSpan() function). The span time is the total time displayed in the trend window.

Note: If you call the TrnSetPeriod() function after the TrnSetSpan() function, the span is automatically set to 0 (zero).

Syntax

TrnGetSpan(AN)

AN:

The AN of the chosen trend.

Return Value

The span time, in seconds. 0(zero) is returned if the AN is invalid or if the span was not set by the TrnSetSpan() function.

Related Functions

[TrnSetSpan](#), [TrnGetPeriod](#), [TrnSetPeriod](#)

Example

```
// Use a keyboard command or button to set a span of 2 hours.
TrnSetSpan(40,StrToTime("2:00:00");
// Then use TrnGetSpan function to display the span
Time = TrnGetSpan(40)
DspText(31,0,TimeToStr(Time,5));
```

See Also[Trend Functions](#)**TrnGetTable**

This function allows you to tabulate values from a specific section of trend. The values in the table (possibly an array variable) are arranged by time.

If the period (*Period*) is different to the trend's sampling period (configured in the Trend Tags database), the returned values are determined by *DisplayMode*.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

TrnGetTable(*Tag*, *Time*, *Period*, *Length*, *Table*, *DisplayMode*, *Milliseconds* [, *ClusterName*])

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is ClusterName.Tag).

Time:

The end time and date (long integer) of the desired trend section. Once you have entered the end time and date (Time), period (Period), and number of trend tag values collected (Length), the start time and date will be calculated automatically. For example, if Time = StrToDate("18/12/96") + StrToTime("09:00"), Period = 30, and Length = 60, the start time would be 08:30. In other words, the trend values for the period between 8.30am and 9am (on December 18, 1996) would be tabulated.

If this argument is set to 0 (zero), the time used will be the current time.

Period:

The time difference between tabulated trend values (in seconds). For example, if you set this period to 30 seconds, CitectSCADA will get the last trend value (sampled at the end of the trend section), then get the trend value that was sampled 30 seconds before that, and so on until it reaches the start time of the trend section.

If this period is different to the trend's sampling period, the trend values will be averaged or interpolated. Set to 0 (zero) to default to the actual trend period.

Length:

The number of trend values to store in the trend table, from 1 to the maximum number of items in the table. This argument has a max of 4000 (in v6). Specifying a length of greater than 4000 results in a return value of 0 and IsError()=274 (INVALID_ARGUMENT). This limit can be configured using the citect.ini parameter [Trend]MaxRequestLength.

Table:

The Cicode array in which the trend data is stored. You can enter the name of an array here (see the example).

DisplayMode:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer that you should enter for a particular trend, select the options you want to use from those listed below, adding their associated numbers together. The resulting integer is the DisplayMode parameter for that trend.

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Ordering trend sample options:

- 0 - Order returned trend samples from oldest to newest.
- 2 - Order returned trend samples from newest to oldest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.
- 12 - Set the condense method to use the newest of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

- n - the number of missed samples that the user wants to gap fill) $\times 4096$.

Options listed in each group are mutually exclusive. The default value for each Display Mode is 258 ($0 + 2 + 256$).

Milliseconds:

This argument allows you to set your sample request time with millisecond precision. After defining the time and date in seconds with the Time argument, you can then use this argument to define the milliseconds component of the time.

For example, if you wanted to request data from the 18/12/96, at 9am, 13 seconds, and 250 milliseconds you could set the Time and Milliseconds arguments as follows:

```
Time = StrToDate("18/12/96") + StrToTime("09:00:13")
Milliseconds = 250
```

If you don't enter a Milliseconds value, it defaults to 0 (zero). There is no range constraint, but as there are only 1000 milliseconds in a second, you should keep your entry between 0 (zero) and 999.

ClusterName:

Name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The actual number of samples read. 0(zero) is returned if an error occurs. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnSetTable](#), [TrnGetDisplayMode](#)

Example

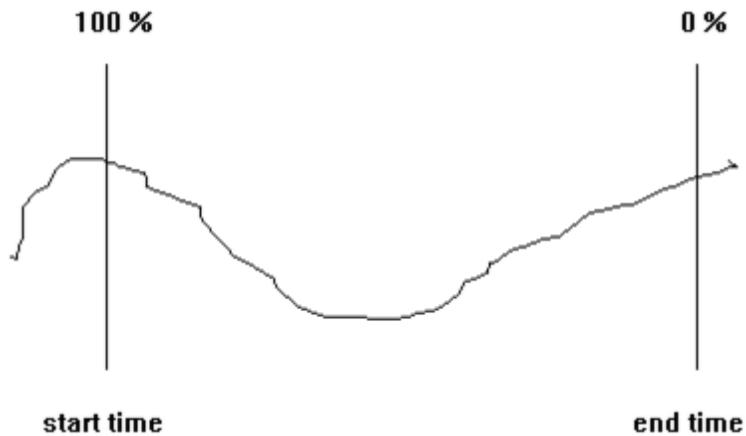
```
REAL TrendTable1[100];
/* Defines an array of a maximum of 100 entries in which the trend
data is stored. */
TrnGetTable("OP1",StrToDate("18/12/91")
+StrToTime("09:00"),2,10,TrendTable1[0],0, "ClusterXYZ");
/* Stores the values of trend tag "OP1" in Table TrendTable1. Data
is stored at the following times:
18/12/91 09:00:00 TrendTable1[0]
08:59:58 TrendTable1[1]
08:59:56 TrendTable1[2]
...
18/12/91 08:59:42 TrendTable1[9] */
Average=TableMath(TrendTable1[0],100,2);
/* Gets the average of the trend data. */
```

See Also

[Trend Functions](#)

TrnGetTime

Gets the time and date of the trend (plotted by a specified pen) at a percentage along the trend, using the time and date of the right-most sample displayed. The time associated with the rightmost sample displayed is known as the end time. The start time is the time of the left-most sample displayed. Percent 0 (zero) will correspond to the end time, and Percent 100 will correspond to the start time.



Syntax

TrnGetTime(AN, Pen, Percent)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Percent:

The percentage of the trend from the time and date of the right-most sample displayed (end time), from 0 to 100.

Return Value

A time/date variable. 0 (zero) is returned if an [error](#) is detected.

Related Functions

[TrnSetTime](#)

Example

```
! For the trend at AN20
DspText(31,0,TimeToStr(TrnGetTime(20,0,0),2));
! Displays the trend current date for the focus pen.
DspText(32,0,TimeToStr(TrnGetTime(20,0,0),1));
! Displays the trend current time for the focus pen.
```

```
DspText(33,0,TimeToStr(TrnGetTime(20,0,50),1));
! Displays the time 50% along the trend for the focus pen.
```

See Also[Trend Functions](#)**TrnGetUnits**

Gets the data units for the trend tag plotted by a specified *Pen*.

Syntax**TrnGetUnits(*AN*, *Pen*)***AN*:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1..8 - Pen1. . .Pen8

Return Value

The data units for the trend tag plotted by *Pen*, otherwise an empty string is returned, and an error code is set. You can call the IsError() function to get the actual [error](#) code.

Related Functions[TrnGetFormat](#), [TrnGetScale](#)**Example**

```
! For the trend at AN20
DspText(31,0,TrnGetUnits(20,0));
! Displays the data units for the focus pen.
```

See Also[Trend Functions](#)**TrnInfo**

Gets the configured values of a trend tag.

This function is a blocking function. It blocks the calling Cicode task until the operation is complete.

Syntax

TrnInfo(*Tag*, *Type* [, *ClusterName*])

Tag:

The name of the trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is *ClusterName.Tag*).

Type:

The type of information required:

1 - Trend Type

2 - Sample Period (to obtain the Display Period, use the TrnGetPeriod function)

3 - Trend File Name (without file extension)

4 - Area

5 - Privilege

6 - Current Event Number. Valid only for event type trends

7 - Engineering Units

8 - The storage method used for the tag. A returned value of 2 represents two byte storage (scaled), 8 represents eight byte storage (floating point).

9 - The file period of the tag in seconds. If the file period is set to monthly or yearly, a file period cannot be calculated as months and years vary in length. Therefore, a file period of 0 will be returned for trends with such file periods.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The value (as a string), otherwise an empty string is returned, and an error code is set. You can call the IsError() function to get the actual [error](#) code.

Example

```
! Get the file name of trend tag LT131
sFileName = TrnInfo("LT131", 3, "ClusterXYZ");
```

See Also[Trend Functions](#)**TrnIsValidValue**

Determines whether a logged trend value is:

- <VALID> - an actual trend value;
- <GATED> - if a periodic trend has a trigger condition, and that condition is FALSE, a standard substitute (or GATED) value is logged instead of the actual value; or
- <INVALID> - for some reason, no value was logged.

Syntax**TrnIsValidValue**(*TrendValue*)

TrendValue:

A trend value (of type REAL).

Return Value

0 for <VALID>

1 for <GATED>

2 for <INVALID>

Related Functions[TmGetGatedValue](#), [TmGetInvalidValue](#)**Example**

```

INT
FUNCTION
DoubleArray()
    INT i;
    FOR i = 0 TO 99 DO
        IF TrnIsValidValue(oldArray[i]) = 1 OR trigger = 0 THEN
            newArray[i] = TrnGetGatedValue();
            Prompt ("This value is <GATED>");
        ELSE
            IF i >= 90 OR TrnIsValidValue(oldArray[i]) = 2 THEN
                newArray[i] = TrnGetInvalidValue();
            ELSE
                newArray[i] = oldArray[i] * 2;
                Prompt ("This value is <TRN_NO_VALUES>");
```

```
END  
RETURN i;  
END
```

See Also

[Trend Functions](#)

TrnNew

Creates a new trend at run time. This function performs the same operation as an entry in the Page Trends database. After the trend is created by the TrnNew() function, all the other trend functions can access and control the trend.

Syntax

TrnNew(AN, Trend [, Tag1 ... Tag8] [, ClusterName])

AN:

The AN where the bottom right-hand corner of the trend is located.

Trend:

The trend definition number (as a STRING).

Tag1 . . . Tag8:

The trend tags. (These tags cannot be prefixed with cluster name, cluster should be specified with the ClusterName argument).

ClusterName:

The name of the cluster in which all the trend tags reside. This is optional if you have one cluster or are resolving the trends via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnDelete](#)

Example

```
TrnNew(20,"trn002","PV1","OP1", "ClusterXYZ");  
/* Creates a new trend at AN20 using trend definition 2, plotting
```

```
"PV1" on Pen1 and "OP1" on Pen2. */
```

See Also

[Trend Functions](#)

TrnPlot

Prints the trend line of one or more trend tags. Each trend line is drawn with a different pen style and marker as appropriate. The trend plot includes a comment and a legend, and you can specify the vertical high and low scales. The Mode defines the color mode of the printer. The default mode is black and white.

For more advanced trend plotting, you can use the low-level plot functions.

Syntax

TrnPlot(*sPort*, *nSamples*, *iTime*, *rPeriod*, *sTitle*, *AN*, *Tag1.....Tag8*, *iMode*, *sComment*, *rLoScale1*, *rHiScale1*,*rLoScale8*, *rHiScale8*)

sPort:

The name of the printer port to which the plot will be printed. This name needs to be enclosed within quotation marks. For example LPT1:, to print to the local printer, or \\Pserver\canon1 using UNC to print to a network printer.

nSamples:

The number of data points on the plot.

iTime:

For periodic trend or event trends displayed as periodic:

The end point in time (the latest point) for the trend plot.

rPeriod:

The event sample number (e.g. using TrnGetEvent)

The period (in seconds) of the trend plot. This can differ from the actual trend period.

If you omit the period, it defaults to the sample period of Tag1.

sTitle:

The title of the trend plot.

Tag1..Tag8:

The trend tags.

AN:

The AN of the chosen trend. If you enter 0 (zero), the display mode will default to 258. (This is the display mode that is passed into TrnGetTable() when it is called internally by TrnPlot().) If you call TrnPlot() from a report, you need to enter 0 (zero) here.

iMode:

The color mode of the printer.

0 - Black and White

1 - Color

sComment:

The comment that is to display beneath the title of the trend plot. You may pass an empty string if no comment is required.

rLoScale1, HiScale1, ..., LoScale8, HiScale8:

The minimum and maximum on the vertical scale for the trend line of each of the tags (Tag1... Tag8).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnComparePlot](#), [TrnPrint](#), [PlotOpen](#), [SPCPlot](#)

Example

```
/* Prints a black and white plot to LPT1, containing the trend
lines of two variable tags (PV1 & PV2). The trend lines have a
starting time of 9am, on 11/10/96, 200 sample points, and a period
of 2 seconds. The trend line of PV1 will be on a vertical scale of
0-200, and PV2 will be on a vertical scale of 0-400. */
INT time;
Time = StrToDate("11/10/96") + StrToTime("09:00:00");
TrnPlot("LPT1:",200,Time,2,"Citect Trend
Plot","PV1","PV2","","","","","","",0,"Process variable operation
at shutdown",0,200,0,400);
```

See Also

[Trend Functions](#)

[TrnPrint](#)

Prints the trend that is displayed on the screen (at AN) using the current display mode for each trend. You can specify the trend title, the target printer, whether to print in color or black and white, and whether to display the Plot Setup form when the function is called.

Syntax

TrnPrint(sPort, sTitle, AN, iModeColor, iDisplayForm)

sPort:

The name of the printer port to which the plot will be printed. This name needs to be enclosed within quotation marks "". For example "LPT1:", to print to the local printer, or "\\Pserver\canon1" using UNC to print to a network printer.

It is not necessary to enter a printer port. The first time the printer port is omitted, you will be prompted to select one at the Printer Setup form. The selection you make will then be used as the default.

sTitle:

The title to print at the top of the trend plot. If you omit the title in sTitle, the page title will be used.

AN:

The AN where the trend plot is located.

iModeColor:

The color mode of the printer.

-1 - Color to be decided (Default). CitectSCADA refers to the [GENERAL]PrinterColorMode parameter to determine print color. If there is no setting for this parameter, it will default to black and white.

0 - Black and White

1 - Color

DisplayForm:

Defines whether or not the Plot Setup form will display when the function is called. This form allows you to enter the color mode of the printer, and define the printer setup etc. (See Printing Trend Data for more information on this form.)

-1 - CitectSCADA refers to the [GENERAL]DisablePlotSetupForm parameter to determine if the form will display.

0 - Do not display form

1 - Display form

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnPlot](#), [TrnComparePlot](#), [WinPrint](#), [SPCPlot](#)

Example

```
TrnPrint("LPT1:","Test Print",40,0,0);
/* Prints the trend plot displayed at AN40, without prompting for
setup details.*/
```

See Also

[Trend Functions](#)

TrnSamplesConfigured

Gets the number of samples configured for the currently displayed trend.

Syntax

TrnSamplesConfigured(AN)

AN:

The AN where the trend is located.

Return Value

The number of samples configured for the trend, or 0 (zero) if an error is detected. You can call the IsError() function to get the actual [error](#) code.

Example

```
/* For the trend at AN20, get and display the number of samples */
INT nSamples;
nSamples=TrnSamplesConfigured(20);
DspStr(31,"",IntToStr(nSamples));
```

See Also

[Trend Functions](#)

TrnScroll

Scrolls the trend pen by a specified percentage (of span), or number of samples.

Syntax

TrnScroll(AN, Pen, nScroll [, nMode])

AN:

The AN where the trend is located. Set to -1 for all trends on the current page.

Pen:

The trend pen number. Set to -1 for all pens.

nScroll:

The amount by which the trend will be scrolled. Use nMode to specify whether the trend will be scrolled by percentage or by number of samples.

Because the resolution of Client requests is 1 second, requests of millisecond accuracy are rounded to 1 second. For example, if requested to scroll 2 samples of 400 milliseconds (a total of 0.8 seconds), the trend will actually scroll 1 second.

nMode

The type of scrolling to be performed.

1 - The trend will be scrolled by a percentage of span. Default.

2 - The trend will be scrolled by a number of samples. This mode is not available if the user puts the trend into the 'trend span' mode by setting the span. In this case no scrolling would take place; the user needs to use nMode 1.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnSetTime](#)

Example

```

! Scroll all pens (of the trend at an20) 100% forwards.
TrnScroll(20,-1,100); or TrnScroll(20,-1,100,1);
! Scrolls all pens (of all trends on the current trend page) 300% backwards.
TrnScroll(-1, -1, -300); or TrnScroll(20,-1,-300,1);
! Scrolls all pens (of all trends on the current trend page) 3 samples forwards.
TrnScroll(20,-1,3,2);
! Scrolls all pens (of all trends on the current trend page) 1 sample backwards.
TrnScroll(20,-1,-1,2);

```

See Also

[Trend Functions](#)

[TrnSelect](#)

Sets up a page for a trend. This function allows you to set up a trend before the trend page is displayed. You can therefore use a single trend page to display any trend in the project by selecting the trend first, and then displaying the trend page. The PageTrend() function uses this function to display the standard trend pages.

Call this function and a set of TrnSetPen() functions before you display a trend page. When the trend page is displayed, all pens set by the TrnSetPen() functions are displayed. You can use the TrnSelect() function to configure different set of pens to be displayed on one generic trend page. The pen settings in the Page Trend database are overridden.

Note: Trend functions used after the TrnSelect() function needs to use the special value -2 as their AN. (See the example below).

Syntax

TrnSelect(*Window*, *Page*, *AN* [, *ClusterName*])

Window:

The window number (returned from the WinNumber function).

-3 - for the current window.

-2 - for the next window displayed.

Page:

The name of the page that displays the trend.

AN:

The AN where the trend displays, or -3 for the first trend on the page.

ClusterName:

The name of the cluster that is associated with any trend tag for this trend graph. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnSetPen](#), [PageTrend](#), [WinNumber](#)

Example

```
TrnSelect(WinNumber(), "TrendPage", 40, "ClusterXYZ");
TrnSetPen(-2,1,"PV1");
TrnSetPen(-2,2,"PV2");
TrnSetPen(-2,3,"PV3");
TrnSetPen(-2,4,"PV4");
PageDisplay("TrendPage");
```

See Also

[Trend Functions](#)

TrnSetCursor

Moves the trend cursor by a specified number of samples. If the trend cursor is disabled, this function enables it. If the cursor is enabled and the number of samples is 0 (zero), the cursor is disabled. If the cursor is moved off the current trend frame, the trend scrolls.

Syntax

TrnSetCursor(AN, Samples)

AN:

The AN where the trend is located. Set to -1 for all trends on the current page.

Samples:

The number of samples to move the cursor.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetCursorTime](#), [TrnGetCursorValue](#), [TrnGetCursorValueStr](#), [TrnSetCursorPos](#)

Example

```
! For the trend at AN20
TrnSetCursor(20,1);
! Moves the trend cursor forwards 1 sample.
TrnSetCursor(-1,-40);
! Moves the trend cursor (of all trends on the current trend page)
backwards 40 samples.
```

See Also

[Trend Functions](#)

TrnSetCursorPos

Moves the trend cursor to a specified x-axis point, offset from the trend cursor origin. If the trend cursor is disabled, this function enables it. If the position is outside of the trend frame, it sets the trend cursor to half of the frame.

Syntax

TrnSetCursorPos(*AN*, *Position*)

AN:

The AN where the trend is located. Set to -1 for all trends on the current page.

Position:

The x-axis point at which to position the trend cursor, offset from the trend cursor origin.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetCursorPos](#), [TrnSetCursor](#)

Example

```
! For the trend at AN20, if the trend frame is 400 points
TrnSetCursorPos(20,0);
! Moves the trend cursor to its origin.
TrnSetCursorPos(20,200);
! Moves the trend cursor to half of its frame size (200 points).
```

See Also

[Trend Functions](#)

TrnSetDisplayMode

Specifies how raw trend samples are displayed on the screen.

Syntax

TrnSetDisplayMode(*AN*, *PenNumber*, *DisplayMode*)

AN:

The animation number of the chosen trend.

PenNumber:

The pen number of the chosen trend. Specify:

- 0 - The current pen
- 1-8 - Pens 1 through 8
- 1 - All pens

DisplayMode:

The Display Mode parameters allow you to enter a single integer to specify the display options for a trend (for a maximum of eight trends).

To calculate the integer you should enter, select the options you want to use from the list below, adding their associated numbers together. The resulting integer is the DisplayMode parameter for that trend.

Note: Options listed in each group are mutually exclusive. The default value for each Display Mode is 258 (0 + 2 + 256).

Invalid/Gated trend options:

- 0 - Convert invalid/gated trend samples to zero.
- 1 - Leave invalid/gated trend samples as they are.

Ordering trend sample options:

- 0 - Order returned trend samples from oldest to newest.
- 2 - Order returned trend samples from newest to oldest.

Condense method options:

- 0 - Set the condense method to use the mean of the samples.
- 4 - Set the condense method to use the minimum of the samples.
- 8 - Set the condense method to use the maximum of the samples.
- 12 - Set the condense method to use the newest of the samples.

Stretch method options:

- 0 - Set the stretch method to step.
- 128 - Set the stretch method to use a ratio.
- 256 - Set the stretch method to use raw samples.

Gap Fill Constant option:

n - the number of missed samples that the user wants to gap fill) x 4096.

Display as Periodic options:

- 0 - Display according to trend type.

1048576 - display as periodic regardless of trend type.

Since the Display as Periodic options are read-only, they cannot be set using TrnSetDisplayMode. They can be retrieved using TrnGetDisplayMode() and also used with the TrnExport group of functions.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetDisplayMode](#), [TrnGetTable](#)

See Also

[Trend Functions](#)

TrnSetEvent

Sets the start event of a trend pen. This function only operates on an event-based trend.

Syntax

TrnSetEvent(*AN*, *Pen*, *Event*)

AN:

The AN of the chosen trend.

Pen:

The trend pen number:

0 - The pen currently in focus

1...8 - Pen1..Pen8

Event:

The number of the start event.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetEvent](#), [TrnGetCluster](#), [TrnGetCursorEvent](#)

Example

```

! Sets pen1 to event number 123456
TrnSetEvent(20,1,123456);
! Scrolls pen1 back by 100 events
TrnSetEvent(20,1,TrnGetBufEvent(20,1,0)-100);

```

See Also

[Trend Functions](#)

TrnSetPen

Sets the trend tag of a trend pen. The trend pen changes to the specified tag and the trend is refreshed. The trend pen needs to be in the operator's area to be displayed. If outside of the operator's area, data is not displayed. You cannot mix periodic trends and event trends in the same trend window.

This function may sometimes return before the pen is actually set when called on a PC which is not the trend server. This may create difficulties for following functions such as TrnsetScale. A wrapper function can be used to confirm the pen is set before returning. See example 2 below.

Syntax

TrnSetPen(AN, Pen, Tag)

AN:

The AN where the trend is located.

- 1 - All trends on the current trend page.
- 2 - The function being called is using the special AN setup by the TrnSelect() function.

Pen:

The pen for which the trend tag will be changed.

- 2 - The first available pen (This value is automatically changed to 0 for SPC trends because they have only one pen per trend.)
- 1 - All pens on the trend. (Not allowed for SPC trends.)
- 0 - The pen currently in focus.
- 1...8 - Pen1....Pen8

Tag:

The trend tag. If Tag = ! the pen is deleted.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned. Be aware that if a mixture of periodic and event trends is detected, the return value is 0 (zero), but the hardware alarm #329 is set.

Related Functions

[TrnGetPen](#), [TrnSelect](#)

Example 1

```
! For the trend at AN20
TrnSetPen(20,1,"PV1");
! Sets the trend tag of Pen1 to "PV1".
```

Example 2

```
INT
FUNCTION
BlockedTrnSetPen(INT hAN, INT nPen, STRING sTrend)
    INT timeout = 5000
    INT sleepTime = 10
    INT error = -1
    INT elapsed = 0
    INT currentTime
    STRING sPenName
    error = TrnSetPen(hAN, nPen, sTrend)
    IF error = 0 THEN
        error = -1
        currentTime = SysTime()
        WHILE error <> 0 AND elapsed < timeout DO
            sPenName = TrnGetPen(hAN, nPen)
            IF sPenName = sTrend THEN
                error = 0
            ELSE
                SleepMS(sleepTime)
                elapsed = elapsed + SysTimeDelta(currentTime)
            END
        END
    END
    RETURN error
END
```

See Also

[Trend Functions](#)

[TrnSetPenFocus](#)

Sets the focus to a specified pen. After the focus is set, the focus pen is used with other trend functions.

Syntax

TrnSetPenFocus(AN, Pen)

AN:

The AN of the chosen trend.

Pen:

The trend pen:

- 4 - Make the next pen the focus pen; without skipping blank pens.
- 3 - Make the previous pen the focus pen; without skipping blank pens.
- 2 - Make the next pen the focus pen; skip blank pens.
- 1 - Make the previous pen the focus pen; skip blank pens.
- 0 - Keep the current focus.
- 1...8 - Change Pen1..8 to be the focus pen.

Return Value

The old pen focus number, or -1 if an error is detected. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnGetPenFocus](#)

Example

System Keyboard	
Key Sequence	NextPen
Command	TrnSetPenFocus(20, -2)
Comment	For the trend at AN20, make the next pen the focus pen

See Also

[Trend Functions](#)

[TrnSetPeriod](#)

Sets the display period (time base) of a trend. When the period is changed, CitectSCADA reads the historical data to reconstruct the trend data, and refreshes the trend. Every pen has the same display period.

This function clears the span set by the TrnSetSpan() function.

Syntax

TrnSetPeriod(AN, Period)

AN:

The AN where the trend is located. Set to -1 for every trend on the current page.

Period:

The new sampling period (in seconds) of the trend. To set the display period to the sampling period, set this argument to 0 (zero).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetPeriod](#), [TrnEcho](#), [TrendSetTimebase](#), [TrendSetSpan](#)

Example

System Keyboard	
Key Sequence	# # Enter
Command	TrnSetPeriod(20, Arg1)
Comment	Set a new sampling period for the trend at AN20

See Also

[Trend Functions](#)

TrnSetScale

Sets a new scale for a trend pen. In the automatic scaling mode, the zero and full scales are automatically generated.

Syntax

TrnSetScale(AN, Pen, Percent, Scale)

AN:

The AN where the trend is located. Set to -1 for all trends on the current page.

Pen:

The trend pen number:

-1 - All pens

0 - The pen currently in focus

1...8 - Pen1...Pen8

Percent:

The scale mode:

-2 - Set both zero and full scales to the default scales.

-1 - Place the trend into automatic scale mode.

0 - Set the zero scale.

100 - Set the full scale.

Scale:

The new value of the scale. Scale is ignored if Percent is -2.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetScale](#), [TrnEcho](#), [TrnSetScale](#)

Example

```
! For the trend at AN20
TrnSetScale(20,-1,100,5000.0);
! Sets the full scale of all pens to 5000.0
```

See Also

[Trend Functions](#)

TrnSetSpan

Sets the span time of a trend. The span time is the total time displayed in the trend window. You can set the period to contain fractions of a second. For example, if you set a trend with 240 samples to a span of 10 minutes, then each sample would be 2.5 seconds. Choose a span long enough to provide a sufficient sample rate to capture accurate real time data.

Syntax

TrnSetSpan(AN, Span)

AN:

The AN of the chosen trend.

Span:

The span time (in seconds).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnSetPeriod](#), [TrnGetSpan](#), [TrnSetSpan](#)

Example

```
// Set a span of 2 hours.  
TrnSetSpan(40,StrToTime("2:00:00"));  
// Then use TrnGetSpan function to display the span  
Time = TrnGetSpan(40);  
DspText(31,0,TimeToStr(Time,5));
```

See Also

[Trend Functions](#)

TrnSetTable

Writes trend tag data from a table to the trend logging system (starting at the top of the table, and continuing to the bottom). Each value is written with a time and date, as specified by *Period*. If *Period* differs from the trend sampling period (defined in the Trend Tags database), the trend's sample values will be calculated (averaged or interpolated) from the tabulated trend data.

The user needs to have the correct privilege (as specified in the database), otherwise the data is not written.

This function is a blocking function. It will block the calling Cicode task until the operation is complete.

Syntax

TrnSetTable(*Tag*, *Time*, *Period*, *Length*, *Table*, *Milliseconds* [, *ClusterName*])

Tag:

The trend tag enclosed in quotation marks "" (can be prefixed by the name of the cluster that is ClusterName.Tag).

Time:

The time and date (long integer) to be associated with the first value in the table when it is set. Once you have entered the end time and date (Time), set period (Period), and number of trend tag values to be set (Length), the start time and date will be calculated automatically. For example, if Time = StrToDate("18/12/96") + StrToTime("09:00"), Period = 30, and Length = 60, the start time would be 08:30. In other words, the first value from the table would be set with time 9am, and the last would be set with time 8.30am (on December 18, 1996).

If this argument is set to 0 (zero), the time used will be the current time.

Period:

This will be the interval (in seconds) between trend values when they are set (that is it will be the perceived sampling period for the trend). This period can differ from the actual trend period. Set to 0 (zero) to default to the actual trend period.

Length:

The number of trend values in the trend table.

Table:

The table of floating-point values in which the trend data is stored. You can enter the name of an array here (see the example).

Milliseconds:

This argument allows you to set the time of the first sample in the table with millisecond precision. After defining the time and date in seconds with the Time argument, you can then use this argument to define the milliseconds component of the time.

For example, if you wanted to set data from the 18/12/96, at 9am, 13 seconds, and 250 milliseconds you could set the Time and Milliseconds arguments as follows:

```
Time = StrToDate("18/12/96") + StrToTime("09:00:13")
Milliseconds = 250
```

If you don't enter a milliseconds value, it defaults to 0 (zero). There is no range constraint, but as there are only 1000 milliseconds in a second, you should keep your entry between 0 (zero) and 999.

ClusterName:

The name of the cluster in which the trend tag resides. This is optional if you have one cluster or are resolving the trend via the current cluster context. The argument is enclosed in quotation marks "".

Return Value

The actual number of samples written. The return value is 0 if an error is detected. You can call the IsError() function to get the actual [error](#) code.

Related Functions

[TrnGetTable](#)

Example

```
REAL TrendTable1[100];
/* Defines an array of a maximum of 100 entries. Assume that
TrendTable1 has been storing data from a source. */
TrnSetTable("OP1",StrToDate("18/12/91")
+StrToTime("09:00"),2,10,TrendTable1,0,"ClusterXYZ");
/* A set of 10 trend data values are set for the OP1 trend tag. */
```

See Also

[Trend Functions](#)

TrnSetTime

Sets the end time and date of a trend pen. If you set a time less than the current time, the trend display is set to historical mode and samples taken after this time and date will not be displayed. If you set the time to the current time, for example by using the [Time-Current](#) or [TrendZoom](#) cicode functions, the trend is displayed in real-time mode and samples after this date and time will display.

Syntax

TrnSetTime(AN, Pen, Time)

AN:

The AN where the trend is located, or:

- 1 - All trends on the current page
- 0 - The trend where the cursor is positioned

Pen:

The trend pen number:

- 1 - All pens

0 - The pen currently in focus

1...8 - Pen1. . .Pen8

Time:

The end time and date of the trend. Samples taken after this time and date will not be displayed. Set to 0 (zero) to set the trend to the current time (real-time mode).

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[TrnGetTime](#), [TrnSetTime](#)

Example

```
TrnSetTime(20,1,TimeCurrent()-60*30);
/* Sets Pen1 to 30 minutes before the current time (30 minutes ago). */
TrnSetTime(20,1,0);
/* Sets the trend to real-time mode. */
```

See Also

[Trend Functions](#)

Chapter: 58 Window Functions

Window functions control the display of windows. You can open, move, size, activate, and de-activate windows. You can also specify titles for your windows.

Window Functions

Following are functions relating to Windows:

<u>GetWinTitle</u>	Returns the name of the active window as a string.
<u>HtmlHelp</u>	Invokes the Microsoft HTML Help application
<u>Mul-tiMonitorStart</u>	Displays a CitectSCADA window on each of the configured monitors when a display client starts up.
<u>WinCopy</u>	Copies the active window to the Windows clipboard.
<u>WinFile</u>	Writes the active window to a file.
<u>WinFree</u>	Removes a display window.
<u>WinGetFocus</u>	Gets the number of the CitectSCADA window that has the keyboard focus.
<u>Win-GetWndHnd</u>	Gets the window handle for the current window.
<u>WinGoto</u>	Changes the active window.
<u>WinMode</u>	Sets the display mode of the active window.
<u>WinMove</u>	Moves the active window.
<u>WinNew</u>	Opens a display window.
<u>WinNewAt</u>	Opens a display window at specified coordinates.
<u>WinNext</u>	Makes the next window active.

WinNumber	Gets the window number of the active CitectSCADA window.
WinPos	Positions a window on the screen.
WinPrev	Makes the previous window active.
WinPrint	Prints the active window.
WinPrintFile	Prints a file to the printer.
WinSelect	Selects a window for Cicode output.
WinSetName	Associates a name with a particular window by its window number.
WinSize	Sizes a window.
WinStyle	Switches on and off scrolling and scroll bar features for existing windows.
WinTitle	Sets the title of the active window.
WndFind	Gets the Windows number of any window in any application.
WndGet-FileProfile	Gets a profile string from any .INI file.
WndGetProfile	This function is obsolete from this version of the product.
WndHelp	Invokes the Windows Help application.
WndInfo	Gets the Windows system metrics information.
WndMon-itorInfo	Returns information about a particular monitor.
WndPut-FileProfile	Puts a profile string into any .INI file.
WndPutProfile	This function is obsolete from this version of the product.
WndShow	Sets the display mode of any window of any application.
WndViewer	Invokes the Windows Multimedia application.

See Also[Functions Reference](#)**GetWinTitle**

Returns the name of the active window as a string.

Note: This function is unavailable in the Web Client.

Syntax[GetWinTitle\(\)](#)**Return Value**

The title of the active window as a string if successful; otherwise, an [error](#) is returned.

Related Functions[WinTitle](#)**See Also**[Window Functions](#)**HtmlHelp**

Invokes the Microsoft HTML Help application (hh.exe) to display a specific topic from a specific HTML help file (.chm).

Syntax[HtmlHelp\(*sHelpFile*, *nCommand*, *sData*\)](#)*sHelpFile*:

The help file to display. For example: "C:\Program Files\Citect\CitectSCADA 7.10\bin\-\CitectSCADA.chm"

nCommand:

The type of help:

0 - Display a topic identified by an internal file name in the *sData* field. This is the name of the file within the .chm file. For example: "CitectSCADA_Help_Overview.html"

1 - Display a topic identified by the Mapped Topic ID in the *sData* field. For example: "1000"

2 - Terminate the help application

3 - Display the index

sData:

Optional data, depending on the value of nCommand. See above.

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[WndHelp](#)

Example

The following example displays the overview page of the CitectSCADA help:

```
HtmlHelp("C:\Program  
Files\Citect\CitectSCADA 7.10\bin\CitectSCADA.chm", 0,  
"CitectSCADA_Help_Overview.html");
```

See Also

[Window Functions](#)

MultiMonitorStart

Displays a CitectSCADA window on each of the configured monitors when a display client starts up. It sets up the windows according to the Multi-Monitor Parameters [MultiMonitors]Monitors and [MultiMonitors]StartPage#.

Syntax

MultiMonitorStart()

Return Value

None.

See Also

[Window Functions](#)

WinCopy

Copies the graphics image of the active window to the Windows Clipboard. You can paste this Clipboard image into other applications.

Notes:

- This function might not work as expected if called directly from the Kernel; instead, this function should be called from a graphics page.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinCopy([xScale] [, yScale] [, bSwapBlackWhite] [, sMap])

xScale:

The x scaling factor for the item being copied. This argument is optional, as a default setting of 1 is used to maintain the current horizontal scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the width of the image.

yScale:

The y scaling factor for the item being copied. This argument is optional, as a default setting of 1 is used to maintain the current vertical scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the height of the image.

bSwapBlackWhite:

Swaps the colors black and white for the purpose of printing pages with a lot of black content. Use the default value of 1 to swap black and white (optional).

sMap:

The file name or path of a text based map file used to specify colors to swap when printing. By default CitectSCADA will look in the bin directory for the map file. The format for the map file is:

```
RRR GGG BBB RRR GGG BBB [HHH] [SSS] [LLL] //
RRR GGG BBB RRR GGG BBB [HHH] [SSS] [LLL] //
```

Where:

RRR is a decimal red intensity value between 000 and 255.

GGG is a decimal green intensity value between 000 to 255.

BBB is a decimal blue intensity value between 000 to 255.

HHH, *SSS* and *LLL* are optional tolerance values to enable swapping a range of colors. Default values are shown below.

HHH is a decimal value representing the Hue tolerance. Valid range is 0 to 360. Default = 0.

SSS is a decimal value representing the Saturation tolerance. Valid range is 0 to 255. Default = 2 * Hue tolerance.

LLL is a decimal value representing the Luminance tolerance. Valid range is 0 to 255. Default = 2 * Hue tolerance.

Leading zeros are not required, however they aid readability.

The first three RGB values specify the FROM color, and the second three RGB values specify the TO color. The tolerance values are applied to the FROM color when replacing individual pixels in the image.

Comments may be placed at the end of each line, or on individual lines, and needs to be proceeded with // or !.

Example map file to swap the CSV_Include project:

```
043 043 255 155 205 255 025 000 025
//Change dark blue to light blue using Hue and Luminance tolerance of 25. (Converts
the CSV title bar)
000 000 000 255 255 255 //Swap black to white with no tolerance
```

Note:If swap color ranges overlap, the behavior is undefined (i.e. a color that falls in both ranges may end up as either color)

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinPrint](#)

Example

```
WinCopy();
! Copies the active window to the Windows Clipboard.
WinCopy(0.5,0.5);
! Copies the active window to the Windows Clipboard at half the
current size.
```

See Also

[Window Functions](#)

WinFile

Writes the graphics image of the active window to a file. The file is saved in the Citect-SCADA compressed .bmp format.

Notes:

- This function might not work as expected if called directly from the Kernel; instead, this function should be called from a graphics page.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinFile(sFile [, xScale] [, yScale] [, bSwapBlackWhite] [, sMap])

sFile:

The name of the file to be created.

xScale:

The x scaling factor for the item being printed. This argument is optional, as a default setting of 1 is used to maintain the horizontal scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the width of the image .

yScale:

The y scaling factor for the item being printed. This argument is optional, as a default setting of 1 is used to maintain the current vertical scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the height of the image.

bSwapBlackWhite:

Swaps the colors black and white for the purpose of printing pages with a lot of black content. Use the default value of 1 to swap black and white (optional).

sMap:

The file name or path of a text based map file used to specify colors to swap when printing. By default CitectSCADA will look in the bin directory for the map file. The format for the map file is:

```
RRR GGG BBB RRR GGG BBB [HHH] [SSS] [LLL] //
RRR GGG BBB RRR GGG BBB [HHH] [SSS] [LLL] //
```

Where:

RRR is a decimal red intensity value between 000 and 255.

GGG is a decimal green intensity value between 000 to 255.

BBB is a decimal blue intensity value between 000 to 255.

HHH, *SSS* and *LLL* are optional tolerance values to enable swapping a range of colors. Default values are shown below.

HHH is a decimal value representing the Hue tolerance. Valid range is 0 to 360. Default = 0.

SSS is a decimal value representing the Saturation tolerance. Valid range is 0 to 255. Default = 2 * Hue tolerance.

LLL is a decimal value representing the Luminance tolerance. Valid range is 0 to 255. Default = 2 * Hue tolerance.

Leading zeros are not required, however they aid readability.

The first three RGB values specify the FROM color, and the second three RGB values specify the TO color. The tolerance values are applied to the FROM color when replacing individual pixels in the image.

Comments may be placed at the end of each line, or on individual lines, and needs to be proceeded with // or !.

Example map file to swap the CSV_Include project:

```
043 043 255 155 205 255 025 000 025
//Change dark blue to light blue using Hue and Luminance tolerance of 25. (Converts
the CSV title bar)
000 000 000 255 255 255 //Swap black to white with no tolerance
```

Note:If swap color ranges overlap, the behavior is undefined (i.e. a color that falls in both ranges may end up as either color)

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinPrint](#)

Example

```
WinFile("DUMP");
/* Writes the active window to a file named DUMP in the current
directory. */
```

See Also

[Window Functions](#)

WinFree

Removes the active display window. Be aware that the last window (and any child windows owned by the last window) cannot be removed. You cannot call this function as an exit command (see Page Properties) or from a Cicode Object.

Notes

- This function cannot be used in the CitectSCADA Web Client.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinFree()

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinNew](#), [WinNewAt](#)

Example

```
WinFree();  
! Removes the active display window.
```

See Also

[Window Functions](#)

WinGetFocus

Gets the number of the CitectSCADA window that has the keyboard focus.

Syntax

WinGetFocus()

Return Value

The window number of the CitectSCADA window that has the keyboard focus. Be aware that this is not the same as the window handle, returned from the WndFind() function.

Related Functions

[WndFind](#)

Example

```
nCitectWin=WinGetFocus();  
! Gets the number of the CitectSCADA window that has  
the keyboard focus
```

See Also

[Window Functions](#)

WinGetWndHnd

Gets the window handle for the current window. The window handle may be used by 'C' programs and CitectSCADA Wnd... functions. You may pass the windows handle to a 'C' program by using the DLL functions.

Syntax

WinGetWndHnd()

Return Value

The window handle if successful, otherwise 0 (zero) is returned. Be aware that this is not the same as a CitectSCADA window number returned from the WinNumber() function.

Related Functions

[DLLCall](#), [WinNew](#), [WndFind](#), [WndShow](#)

Example

```
INT hWnd;  
hWnd = WinGetWndHnd();  
WinShow(hWnd, 6); //iconize the window
```

See Also

[Window Functions](#)

WinGoto

Changes the active window. The specified window is placed in front of all other windows and all keyboard commands will apply to this window. You cannot call this function as an exit command (see Page Properties) or from a Cicode Object.

Note: This function is not supported in the server process in a multiprocessor

environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinGoto(*Window*)

Window:

The window number (returned from the WinNumber() function). Be aware that this is not the same as the window handle, returned from the WndFind() function.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinNew](#)

Example

```
! If two windows are displayed;
WinGoto(1);
! Changes the active window to Window 1.
WinGoto(0);
! Changes the active window to Window 0.
```

See Also

[Window Functions](#)

WinMode

Sets the display mode of the active CitectSCADA window.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinMode(*Mode*)

Mode:

The mode:

- 0 - Hide the window.
- 2 - Activate the window in an iconized state.
- 3 - Activate the window in a maximized state.
- 4 - Display the window in its previous state without activating it.
- 5 - Activate the window in its current state.
- 6 - Iconize the window.
- 7 - Display the window in an iconized state without activating it.
- 8 - Display the window in its current state without activating it.
- 9 - Activate the window in its previous state.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinNew](#)

Example

```
! Iconize the active CitectSCADA window.  
WinMode(7);
```

See Also

[Window Functions](#)

WinMove

Moves the active window to a new location and sizes the window in a single operation. This is the same as calling the WinPos() and the WinSize() functions. You use [PageInfo](#) to get the current window position.

Notes

- This function cannot be used in the CitectSCADA Web Client.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinMove(X, Y, Width, Height)

X, Y:

The new x and y pixel coordinates of the top-left corner of the active window.

Width:

The width of the window, in pixels.

Height:

The height of the window, in pixels.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinSize](#), [WinPos](#), [PageInfo](#)

Example

```
WinMove(100,50,500,300);
/* Moves the top-left corner of the active window to the pixel
coordinate 100,50 and size the window to 500 x 300 pixels. */
```

See Also

[Window Functions](#)

WinNew

Opens a new display window, with a specified page displayed. The window can later be destroyed with the WinFree() function.

You can also specify if the displayed page operates within the context of a particular cluster in a multiple cluster project. When the page is displayed during runtime, the ClusterName argument is used to resolve any tags that do not have a cluster explicitly defined.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinNew(*Page*,*ClusterName*)

Page:

The name or page number of the page to display (in quotation marks ""). Can be prefixed by the name of a host cluster, that is "ClusterName.Page". This will take precedence over the use of the ClusterName parameter if the two differ.

ClusterName:

The name of the cluster that will accommodate the page at runtime. This is optional if you have one cluster or are resolving the page via the current cluster context. The argument is enclosed in quotation marks "". If the Page parameter is prefixed with the name of a cluster, this parameter will not be used.

Return Value

The window number of the window, or -1 if the window cannot be opened. Be aware that this is not the same as the window handle returned from the WndFind() function.

Related Functions

[WinFree](#), [WinNewAt](#)

Example

```
! If the display window being opened is window number 2:  
Window=WinNew("Alarm");  
! Displays the Alarm page and sets Window to 2.
```

See Also

[Window Functions](#)

WinNewAt

Opens a new display window at a specified location, with a selected page displayed. The window can later be destroyed with the WinFree() function.

You can also specify if the displayed page operates within the context of a particular cluster in a multiple cluster project. When the page is displayed during runtime, the ClusterName argument is used to resolve any tags that have a cluster omitted.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinNewAt(*Page*, *X*, *Y*, *Mode*, *ClusterName*)

Page:

The name or page number of the page to display (in quotation marks ""). Can be prefixed by the name of a host cluster, that is "ClusterName.Page". This will take precedence over the use of the ClusterName parameter if the two differ.

X:

The x pixel coordinate of the top left corner of the window.

Y:

The y pixel coordinate of the top left corner of the window.

Mode:

The mode of the window:

0 - Normal page.

1 - Page child window. The window is closed when a new page is displayed, for example, when the PageDisplay() or PageGoto() function is called.
The parent is the current active window.

2 - Window child window. The window is closed automatically when the parent window is freed with the WinFree() function. The parent is the current active window.

4 - No re-size. The window is displayed with thin borders and no maximize/minimize icons. The window cannot be re-sized.

8 - No icons. The window is displayed with thin borders and no maximize/minimize or system menu icons. The window cannot be re-sized.

16 - No caption. The window is displayed with thin borders, no caption, and no maximize/minimize or system menu icons. The window cannot be re-sized.

32 - Echo enabled. When enabled, keyboard echo, prompts, and error messages are displayed on the parent window. This mode should only be used with child windows (for example, Mode 1 and 2).

64 - Always on top.

128 - Open a unique window. This mode helps to prevent this window from being opened more than once.

256 - Display the entire window. This mode commands that no parts of the window will appear off the screen

512 - Open a unique Super Genie. This mode helps to prevent a Super Genie from being opened more than once (at the same time). However, the same Super Genie with different associations can be opened.

1024 - Disables dynamic resizing of the new window, overriding the setting of the [Page]DynamicSizing parameter.

4096 - Allows the window to be resized without maintaining the current aspect ratio. The aspect ratio defines the relationship between the width and the height of the window, which means this setting allows you to stretch or compress the window to any proportions. This option overrides the setting of the [Page]MaintainAspectRatio parameter.

8192 - Text on a page will be resized in proportion with the maximum scale change for a resized window. For example, consider a page that is resized to three times the original width, and half the original height. If this mode is set, the font size of the text on the page will be tripled (in proportion with the maximum scale). This option overrides the setting of the [Page] ScaleTextToMax parameter.

16384 - Hide the horizontal scroll bar.

32768 - Hide the vertical scroll bar.

65536 - Disable horizontal scrolling.

131072 - Disable vertical scrolling.

You can select multiple modes by adding modes together (for example, set Mode to 9 to open a page child window without maximize, minimize, or system menu icons).

ClusterName:

The name of the cluster that will accommodate the page at runtime. This is optional if you have one cluster or are resolving the page via the current cluster context. The argument is enclosed in quotation marks "". If the Page parameter is prefixed with the name of a cluster, this parameter will not be used.

Return Value

The window number of the window, or -1 if the window cannot be opened. Be aware that this is not the same as the window handle returned from the WndFind() function.

Related Functions

[WinFree](#), [WinNew](#)

Example

Buttons	
Text	Mimic Page
Com- mand	WinNewAt("Mimic", 100, 20, 0)

Comment	Display the mimic page in a new window at coordinate 100, 20.
Buttons	
Text	Pop Page
Com- mand	WinNewAt("Popup", 100, 200, 2)
Comment	Display the popup page in a child window at coordinate 100, 200
Buttons	
Text	Pop Page
Com- mand	WinNewAt("Popup", 100, 200, 4)
Comment	Display the popup page in a new window with no maximize and minimize icons
System Keyboard	
Key Sequence	Pop ##### Enter
Com- mand	WinNewAt(Arg1, 100, 200, 2)
Comment	Display a specified popup page in a child window at coordinate 100, 200
System Keyboard	
Key Sequence	Pop ##### Enter
Com- mand	WinNewAt(Arg1, 100, 200, 4)
Comment	Display a specified popup page in a new window with no maximize and minimize icons

See Also

[Window Functions](#)

Makes the next window (in order of creation) active.

Syntax

WinNext()

Return Value

The window number of the window, or -1 if there is no next window. Be aware that this is not the same as the window handle returned from the WndFind() function.

Related Functions

[WinNew](#), [WinPrev](#)

Example

```
! If the display window being made active is window number 2:  
Window=WinNext();  
! Makes the next window active and sets Window to 2.
```

See Also

[Window Functions](#)

WinNumber

Gets the window number of the active CitectSCADA window. This number can be used with other functions to control the window.

Syntax

WinNumber([*sName*])

sName:

String name previously associated with a window number using WinSetName().

Return Value

Window Number associated with the Name provided. If no Name is specified then the active window number is returned. If there isn't a valid window number associated with the name provided then -1 is returned.

Related Functions

[WinNew](#), [WinGoto](#), [WinSetName](#)

Example

```
! Create a new window, but keep the active window the same:
Window=WinNumber();
WinNew("Alarm");
WinGoto(Window);
```

See Also[Window Functions](#)**WinPos**

Moves the active window to a new location. You use PageInfo() to get the current window position.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax**WinPos(X, Y)**

X, Y:

The new x and y pixel coordinates of the top-left corner of the active window.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions[WinSize](#), [WinMove](#), [PageInfo](#)**Example**

```
WinPos(100,50);
/* Moves the top-left corner of the active window to the pixel
coordinate 100,50. */
```

See Also[Window Functions](#)**WinPrev**

Makes the previous window (in order of creation) active.

Syntax

WinPrev()

Return Value

The window number of the window, or -1 if there is no next window. Be aware that this is not the same as the window handle returned from the WndFind() function.

Related Functions

[WinNext](#)

Example

```
! If the display window being made active is window number 2:  
Window=WinPrev();  
! Makes the previous window active and sets Window to 2.
```

See Also

[Window Functions](#)

WinPrint

Sends the graphics image of the active window to a printer.

Notes:

- This function might not work as expected if called directly from the Kernel; instead, this function should be called from a graphics page.
- This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinPrint(*sPort* [, *xScale*] [, *yScale*] [, *bSwapBlackWhite*] [, *sMap*])

sPort:

The name of the printer port to which the window will be printed. This name needs to be enclosed within quotation marks "". For example "LPT1:", to print to the local printer, or "\\\Pserver\canon1" using UNC to print to a network printer. *sPort* may not contain spaces.

xScale:

The x scaling factor for the print. The default value of 0 (zero) automatically scales the print to fit the page. A value of 1 is used to maintain the current horizontal scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the width of the image (optional).

yScale:

The y scaling factor for the print. The default value of 0 (zero) automatically scales the print to fit the page. A value of 1 is used to maintain the current horizontal scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the width of the image (optional).

bSwapBlackWhite:

Swaps the colors black and white for the purpose of printing pages with a lot of black content. Use the default value of 1 to swap black and white (optional).

sMap:

The file name or path of a text based map file used to specify colors to swap when printing. By default CitectSCADA will look in the bin directory for the map file. The format for the map file is:

```
RRR GGG BBB RRR GGG BBB [HHH] [SSS] [LLL] //
RRR GGG BBB RRR GGG BBB [HHH] [SSS] [LLL] //
```

Where:

RRR is a decimal red intensity value between 000 and 255.

GGG is a decimal green intensity value between 000 to 255.

BBB is a decimal blue intensity value between 000 to 255.

HHH, *SSS* and *LLL* are optional tolerance values to enable swapping a range of colors. Default values are shown below.

HHH is a decimal value representing the Hue tolerance. Valid range is 0 to 360. Default = 0.

SSS is a decimal value representing the Saturation tolerance. Valid range is 0 to 255. Default = 2 * Hue tolerance.

LLL is a decimal value representing the Luminance tolerance. Valid range is 0 to 255. Default = 2 * Hue tolerance.

Leading zeros are not required, however they aid readability.

The first three RGB values specify the FROM color, and the second three RGB values specify the TO color. The tolerance values are applied to the FROM color when replacing individual pixels in the image.

Comments may be placed at the end of each line, or on individual lines, and needs to be proceeded with // or !.

Example map file to swap the CSV_Include project:

```
043 043 255 155 205 255 025 000 025
//Change dark blue to light blue using Hue and Luminance tolerance of 25. (Converts
the CSV title bar)
000 000 000 255 255 255 //Swap black to white with no tolerance
```

Note:If swap color ranges overlap, the behavior is undefined (i.e. a color that falls in both ranges may end up as either color)

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinPrintFile](#)

Example

```
WinPrint("LPT3:",0,0,0);
! Prints the active window on printer "LPT3". The print will be
scaled to fit the largest possible page area and will retain the
orientation of the printer, aspect ratio and colors that are
displayed on screen.
```

```
WinPrint("LPT3:");
!Prints the page as in the first example, but swaps black and
white on the printout.
```

See Also

[Window Functions](#)

WinPrintFile

Prints a file to the system printer. The file needs to be saved with the WinFile() function.

Note: This function might not work as expected if called directly from the Kernel; if the WinFile function does not succeed, WinPrintFile either doesn't print anything or prints a previously saved page. This function should be called from a graphics page.

Syntax

WinPrintFile(*sFile*, *sPort* [, *xScale*] [, *yScale*] [, *bSwapBlackWhite*] [, *fromColor*] [, *toColor*])

sFile:

The file name.

sPort:

The name of the printer port to which the window will be printed. This name needs to be enclosed within quotation marks "". For example "LPT1:", to print to the local printer, or "\Pserver\canon1" using UNC to print to a network printer. *sPort* may not contain spaces

xScale:

The x scaling factor for the print. The default value of 0 (zero) automatically scales the print to fit the page. A value of 1 is used to maintain the current horizontal scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the width of the image (optional).

yScale:

The y scaling factor for the print. The default value 0 (zero) automatically scales the print to fit the page. A value of 1 is used to maintain the current horizontal scaling of the image. The outcome is proportional to 1; for example, 0.5 halves the width of the image (optional).

bSwapBlackWhite:

Swaps the colors black and white for the purpose of printing pages with a lot of black content. Use the default value of 1 to swap black and white (optional).

fromColor

The hex RGB color value (0xRRGGBB) to change into *toColor*. The default value of -1 results in no color change (optional).

toColor

The hex RGB color value (0xRRGGBB) into which *fromColor* is changed. The default value of -1 results in no color change (optional).

Note: To change a color, a value needs to be specified for both *fromColor* and *toColor*.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinPrint](#)

Example

```
! Save image to disk then print.
WinFile("temp");
WinPrintFile("temp", "LPT3:", 0, 0, 0);
```

```
! Prints the file "temp" on printer "LPT3". The print will be
scaled to fit the largest possible page area and will retain the
orientation of the printer, aspect ratio and colors that are
displayed on screen.
WinPrintFile("temp","LPT3:");
! Prints the page as in the first example, but swaps black and
white on the printout.
WinPrintFile("temp","LPT3:",0,0,0,0x00FF00,0xFF0000);
! Changes green to red.
WinPrintFile("temp","LPT3:",0,0,1,0x00FF00,0xFF0000);
! Changes green to red and black to white.
```

[Window Functions](#)

WinSelect

Selects a window to make active. This function only affects the output of Cicode functions. It does not change the screen focus of the windows, or move a background window to the foreground.

Always re-select the original window if it is called from a Page database (Page Numbers, Page Symbols, and so on), because other Cicode tasks will assume it is the correct window. This function only changes the active window for the Cicode task that called it.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinSelect(*Window*)

Window:

The window number to select. Be aware that this is not the same as the window handle returned from the WndFind() function.

Return Value

The old window number.

Related Functions

[WinGoto](#), [WinNumber](#)

Example

```
OldWindow=WinSelect(1);
! Selects window number 1.
Prompt("Message to Window 1");
! Sends message to window number 1.
WinSelect(2);
! Selects window number 2.
Prompt("Message to Window 2");
! Sends message to window number 2.
WinSelect(OldWindow);
! Selects original window.
```

See Also

[Window Functions](#)

WinSetName

Associates a name with a particular window by its window number. An association with a window is removed when the window is free.

Syntax

WinSetName(*sName* [, *iWinNum*])

sName:

String name to associate with window number *iWinNum*.

iWinNum:

An optional parameter which specifies the window number with which to associate Name. If no number is specified the name is associated with the currently selected window number..

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinNumber](#)

See Also

[Window Functions](#)

WinSize

Sizes the active window. The origin of the window does not move.

Note: This function is not supported in the server process in a multiprocessor environment. Calling this function from the server process results in a hardware alarm being raised.

Syntax

WinSize(Width, Height)

Width, Height:

The new width and height of the window, in pixels.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinMove](#), [WinPos](#)

Example

```
WinSize(200,100);  
! Sizes the active window to 200 pixels wide x 100 pixels high.
```

See Also

[Window Functions](#)

WinStyle

Switches on and off scrolling and scrollbar features for existing windows.

Syntax

WinStyle(Style, Mode)

Style:

One of the following:

- 1 - Hide horizontal scroll bars.
- 2 - Hide vertical scroll bars.
- 3 - Disable horizontal scrolling.
- 4 - Disable vertical scrolling.

Mode:

The mode of the option:

-
- 0 - Turn option off.
 - 1 - Turn option on.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinNewAt](#)

Example

To turn horizontal and vertical scroll bars off for the current window:

```
WinStyle(1, 1);  
WinStyle(2, 1);
```

See Also

[Window Functions](#)

WinTitle

Sets the title of the active window.

If a window title has been set with the [Page]WinTitle parameter, CitectSCADA uses this title when it refreshes the page (overriding the window title set with the WinTitle() function). To minimize the likelihood of CitectSCADA from overriding the title, set the parameter [Page]WinTitle to *.

Syntax

WinTitle(*sTitle*)

sTitle:

The new title for the window.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WinNew](#), [GetWinTitle](#)

Example

```
WinTitle(Time() + " " + Date());  
! Places the current time and date into the window title.
```

See Also

[Window Functions](#)

WndFind

Gets the Windows handle of any window of any application, so that the window can be manipulated. The window handle is not the same as the CitectSCADA window number and cannot be used with functions that expect the CitectSCADA window number (the *Win...* functions).

The window title (caption) needs to be an exact match of the window name (including any blank spaces) for this function to find the window. You should therefore check that the other application does not change the title of the window during execution.

Be aware that if the title banner of a CitectSCADA window is set with the CitectSCADA parameter [Page] WinTitle, you should not specify justification (for example, use {TITLE,32,N}). If justification is not disabled (that is the N is omitted), you need to pass the full title of the window (including leading and trailing blanks) to this function.

Syntax

WndFind(*sTitle*)

sTitle

The title (caption) of the window.

Return Value

The window handle. Be aware that this is not the same as a CitectSCADA window number returned from the WinNumber() function.

Related Functions

[WinNew](#)

Example

```
hWndExcel=WndFind("Microsoft Excel - Book1");  
! Gets the Windows number of the window titled "Microsoft Excel -
```

```
Book1"
```

See Also

[Window Functions](#)

WndGetFileProfile

Gets a profile string from any .ini file.

Syntax

WndGetFileProfile(*sGroup*, *sName*, *sDefault*, *sFile*)

sGroup:

The name of the [group].

sName:

The name of the variable.

sDefault:

The default value.

sFile:

The .ini file name.

Return Value

The profile string from *sFile*.

Related Functions

[WndPutFileProfile](#)

Example

```
! get this user startup page from USER.INI File
sStartup =
WndGetFileProfile(Name(), "Startup", "menu", "[Run] : \USER.INI");
PageDisplay(sStartup);
```

See Also

[Window Functions](#)

WndHelp

Invokes the Windows Help application (WinHlp32.EXE) to display a specific topic from a specific help file.

Syntax

WndHelp(*sHelpFile*, *Command*, *Data*)

sHelpFile:

The help file to display.

Command:

The type of help:

- 1 - Displays the help topic identified by the context string/number in the Data field. The context string/number needs to be defined in the [MAP] section of the help's .HPJ file.
- 2 - Closes the Help application. Enter an empty string for the Data argument.
- 3 - Displays the help contents topic defined by the CONTENTS option in the [OPTIONS] section of the .HPJ file.
- 4 - Displays the contents topic of the designated How to Use Help file. The context string/number (specified in the Data field) needs to be defined in the [MAP] section of the .HPJ file.
- 5 - Changes the current help contents topic to match the context string/number specified in the Data field. This topic is used instead of the one defined by the CONTENTS option in the [OPTIONS] section of the .HPJ file. This will affect Command 3 (see above). The context string/number needs to be defined in the [MAP] section of the help's .HPJ file, and the help file needs to already be open. The change will last only until the help file is closed.
- 8 - Displays, in a pop-up window, the help topic identified by the context string/number in the Data field. The context string/number needs to be defined in the [MAP] section of the .HPJ file.
- 9 - Tests that the correct help file is displayed. If the correct help file is currently displayed, this command merely makes the help the active window. If the incorrect help file is displayed, WinHelp opens the correct file, and displays the help contents topic defined by the CONTENTS option in the [OPTIONS] section of the .HPJ file.

Note: This command will not distinguish between two files of the same name, regardless of their paths.

- 11 - Displays the CitectSCADA Help Topics with either the Contents, the Index, or the Find tab selected, depending on which one was last used. Enter an empty string for the Data argument.
- 257 - Searches the help index for your keyword (as specified in the Data field) and displays the first topic in the index with an identical match. If there is no match, displays the index with your keyword already entered. To display the index without passing a keyword, enter an empty string for the Data argument.
- 258 - Executes the Help macro string specified in the Data field. Help needs to be running and the help file needs to be open, or the message is ignored.
- 260 - Displays, in a pop-up window, the help topic identified by the context string/number in the Data field.
- 261 - Searches the help index for your keyword (as specified in the Data field) and displays the first topic in the index with an identical match. If there is no match, displays the index with your keyword already entered. To display the index without passing a keyword, enter an empty string for the Data argument.

Data:

The context string/number or keyword of the help topic that is requested.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WndViewer](#)

Example

```
WndHelp("MyHelp.HLP", 3, 1);
! Displays the "MyHelp" Contents page.
WndHelp("C:\Help\Process.HLP", 8, 239);
! Displays topic labelled "239" in the "Process" help file.
```

See Also

[Window Functions](#)

[WndInfo](#)

Gets information on the window system (such as the widths and heights of the various elements displayed by Windows). WndInfo() can also return flags that indicate whether the current version of the Windows operating system is a debugging version, whether a mouse is present, or whether the functions of the left and right mouse buttons have been exchanged.

Syntax

WndInfo(*iType*)

iType:

The system measurement to be retrieved. Measurements are in pixels. The system measurement needs to be one of the following values:

- 0 - Width of the screen.
- 1 - Height of the screen.
- 2 - Width of the arrow bitmap on a vertical scroll bar.
- 3 - Height of the arrow bitmap on a horizontal scroll bar.
- 4 - Height of the window title. This is the title height plus the height of the window frame that cannot be sized (SM_CYBORDER).
- 5 - Width of the window frame that cannot be sized.
- 6 - Height of the window frame that cannot be sized.
- 7 - Width of the frame when the window has the WS_DLGFRADE style.
- 8 - Height of the frame when the window has the WS_DLGFRADE style.
- 9 - Height of the scroll box on vertical scroll bar.
- 10 - Width of the scroll box (thumb) on horizontal scroll bar.
- 12 - Height of the icon.
- 13 - Width of the cursor.
- 14 - Height of the cursor.
- 15 - Height of a single-line menu bar. This is the menu height minus the height of the window frame that cannot be sized (SM_CYBORDER).
- 16 - Width of the window client area for a fullscreen window.
- 17 - Height of the window client area for a fullscreen window (equivalent to the height of the screen minus the height of the window title).
- 18 - Height of a Kanji window.
- 19 - Non-zero if the mouse hardware is installed.
- 20 - Height of arrow bitmap on a vertical scroll bar.
- 21 - Width of arrow bitmap on a horizontal scroll bar.
- 22 - Non-zero if the Windows version is a debugging version.
- 23 - Non-zero if the left and right mouse buttons are swapped.

24-27 - Not Used
 28 - Minimum width of the window.
 29 - Minimum height of the window.
 30 - Width of bitmaps contained in the title bar.
 31 - Height of bitmaps contained in the title bar.
 32 - Width of the window frame that can be sized.
 33 - Height of the window frame that can be sized.
 34 - Minimum tracking width of the window.
 35 - Minimum tracking height of the window.
 76 - x co-ordinate of upper left corner of virtual screen
 77 - y co-ordinate of upper left corner of virtual screen
 78 - width of virtual screen
 79 - height of virtual screen
 80 - number of monitors available

Return Value

The system metric information.

Example

```

width = WndInfo(0);           ! get width of screen
height = WndInfo(1);          ! get height of screen
WinPos(width/2, height/2);    ! move window to centre of screen
monitors = WndInfo(80);       ! get number of monitors available

```

See Also

[Window Functions](#)

WndMonitorInfo

Returns information about a particular monitor.

Syntax

WndMonitorInfo(*iMonitor*, *iType*)

iMonitor:

Monitor Number 1 to n, where n is the number of monitors returned from WndInfo(80). Using 0 will return the value for the virtual screen. Using an unsupported monitor number (0 or n) will return -1 for all values.

iType:

Type The monitor measurement to be retrieved:

0 - x co-ordinate of upper left corner of monitor.

1 - y co-ordinate of upper left corner of monitor.

2 - width of monitor.

3 - height of monitor.

4 - x co-ordinate of upper left corner of monitor working area.

5 - y co-ordinate of upper left corner of monitor working area.

6 - width of monitor working area.

7 - height of monitor working area.

Return Value

Requested information about the selected monitor.

See Also

[Window Functions](#)

WndPutFileProfile

Puts a profile string into any .INI file.

Syntax

WndPutFileProfile(*sGroup*, *sName*, *sData*, *sFile*)

sGroup:

The name of the [group].

sName:

The name of the variable.

sData:

The variable data.

sFile:

The .INI file name.

Return Value

0 (zero) if successful, otherwise an [error](#) is returned.

Related Functions

[WndGetFileProfile](#)

Example

```
WndPutFileProfile(Name(), "What", "100", "USER.INI");
```

See Also

[Window Functions](#)

WndShow

Sets the display mode of any window of any application.

Syntax

WndShow(*hWnd*, *nMode*)

hWnd:

The Windows handle of the window (returned from the WndFind() function). Be aware that this is not the same as a CitectSCADA window number returned from the WinNumber() function.

nMode:

The window mode:

0 - Hide the window.

1 - Activate the window in normal mode.

2 - Activate the window in an iconized state.

3 - Activate the window in a maximized state.

4 - Display the window in its previous state without activating it.

5 - Activate the window in its current state.

6 - Iconize the window.

7 - Display the window in an iconized state without activating it.

8 - Display the window in its current state without activating it.

9 - Activate the window in its previous state.

Return Value

0 (zero) if successful, otherwise an error is returned.

Related Functions

[WndFind](#)

Example

```
WndShow(WndFind("Microsoft Excel"), 0);  
! Hides the "Microsoft Excel" window.
```

See Also

[Window Functions](#)

WndViewer

Invokes the Microsoft Multimedia application.

Syntax

WndViewer(*sViewerFile*, *Command*, *Data*)

sViewerFile:

The Multimedia Viewer file to display.

Command:

The type of help:

1 - Displays a Viewer topic (specified in the Data field) in the main Viewer window.

2 - Displays a Viewer topic (specified in the Data field) in a pop-up window.

Data:

The context string of the Multimedia Viewer topic.

Return Value

0 (zero) if successful, otherwise an error is returned.

Note: CitectSCADA cannot test if the topic has been found or displayed correctly. For example, if you pass an invalid topic, the viewer will open with "Viewer topic does not exist" - but this function will return 0.

Related Functions

[WndHelp](#)

Example

```
WndViewer("MyFile.MVB",1, "Contents");
! Displays the contents topic in the Multimedia file "MyFile.MVB".
WndViewer("HelpFile.MVB",2, "HelpTip");
! Displays the HelpTip topic in the Multimedia file "HelpFile.MVB"
in a popup.
```

See Also

[Window Functions](#)

Part: 6

Technical Reference

This section contains information for Users and describes the following:

[Cicode Errors](#)

[Browse Function Field Reference](#)

Chapter: 59 Cicode Errors

This section describes the error codes used by CitectSCADA in the following situations:

- [Hardware related errors](#)
- [Cicode and general errors](#)
- [Mail API errors](#)

Hardware/Cicode Errors

CitectSCADA 'traps' system errors automatically. When CitectSCADA detects a system error, it generates a hardware alarm, and the corresponding error message is placed in the alarm description. Each error has an associated (unique) error number.

You can use the IsError() function to get the number of the last error. Alternatively, you can trap and process errors within your user functions. Use the ErrSet() function to enable or disable error trapping. CitectSCADA 'traps' system errors automatically. When a system error occurs, CitectSCADA generates a hardware alarm, and the corresponding error message is placed in the alarm description. Each error has an associated (unique) error number. Number 0 means no error, errors start from 1.

Range	Source	Cause
1 - 31	PLC or I/O Device Generic errors	The I/O Device is reporting an error, or CitectSCADA is experiencing the reported error trying communicate with an I/O Device. Often caused by incorrect configuration or poor cabling. For a detailed list of these errors see Generic Errors .
256 -511	General	<p>General errors are wide ranging, from animation to server problems. However, there are two main causes of general errors:</p> <ol style="list-style-type: none">1. Device External devices such as printers, databases, and files can cause many different hardware errors since they are beyond the control of CitectSCADA. Often the device itself is improperly configured or non-existent.2. Cicode

Cicode errors are generated when your project configuration calls a Cicode function in an invalid way, or when a Cicode function returns an error or does illegal operations.

For a detailed list of these errors see [Cicode and General Errors](#).

382 - 383	Invalid tag data	This will indicate that the tag data validation process has identified a discrepancy with the tag index values. See Validating distributed project data for tag-based drivers for more information.
-----------	------------------	---

You can use the `IsError()` function to get the number of the last error. Alternatively, you can trap and process errors within your user functions. Use the `ErrSet()` function to enable or disable error trapping.

Cicode and General Errors

The table below describes the general errors in Cicode.

Error No.	Error title	Description
55	Tag value is limited	Tag value is limited to be within range.
256	General software error	An internal CitectSCADA software error is detected. Contact Technical Support for this product and provide details on what causes the error.
257	Value is out of range	A numeric value is out of range. An out-of-range value has been passed to a function, or an array index is off the end of an array, or a value that is outside of the specified engineering scale has been assigned to a I/O Device variable. You can disable range checking on PLC variables with the <code>CodeSetMode()</code> function.
258	Buffer has been overrun	A buffer has been overrun. More data has been passed to a function than it can write to its temporary buffers. Try again by calling the function twice, with half the data in each call.
259	Array has been overrun	An array passed to a function is too small for the data requested. Define a larger array or reduce the maximum data size requested.

Error No.	Error title	Description
260	Path does not exist	The specified path to a device or file does not exist. During a function call (that tried to open a file), a non-existent path was specified. Call the function again with the correct path.
261	File does not exist	The specified file or device does not exist. During a function call (that tried to open a file), the file could not be found. Call the function again with the correct file name. This error will also be detected if you try to call TrnDelHistory() on a file that has not been added using TrnAddHistory().
262	Cannot open file	<p>The specified file cannot be opened. During a function call (that tried to open a file), the file could not be opened. There may be a mode error (for example, from trying to open a read-only file in write mode), or the file may be open by others, or the operating system resources may be too low to open the file.</p> <p>Check that the file does exist (use File Manager), and that you have the correct rights to open it. (Check with your network supervisor that you have correct rights to open the file).</p>
263	Cannot read file	The specified file cannot be read. Either an error has been detected during a read operation, or the end of file was unexpectedly found, or a disconnection from the file server occurred, or the operating system is out of resources.
264	Cannot write to file	The specified file cannot be written to. During a function call (that tried to write to a file), a write error has been detected. There could be a disk full error, or a disconnection from the file server may have occurred, or the operating system is out of resources.
265	Invalid file type	An attempt was made to open a file of the wrong type, for example, you tried to open an ASCII file as a dBASE file.
266	Field not found in file	The specified field does not exist in the

Error No.	Error title	Description
		device or database. A function that is trying to access an individual field in a database cannot find that field. Check that you have specified the correct field name and database name.
267	File mode is invalid	An operation has been attempted on a file or device that is of the wrong mode, for example, you tried to perform a seek on a printer device.
268	Key not found in file	The requested key was not found when a key search was performed on a database device, that is the record specified on an indexed search cannot be found. Either the record does not exist or you have specified the wrong key.
269	Bad handle specified	A bad handle has been passed to a function. You have called a function that requires a device handle, font handle, window handle, etc., but you passed a number that does not associate with a read device, font, or window (for example, you called WinGoto(100) when no window with the handle "100" exists). Check where the handle or number was retrieved from, and verify that it is the same handle. This error may also be detected if you have closed or destroyed the resource and you then try to access it.
271	No more free handles left	All the available file handles have been used, that is too many files or databases are open at the same time. Open fewer files at one time or increase the number of file handles in the [CtEdit]DBFiles parameter.
272	Out of memory	CitectSCADA is out of memory. Increase the amount of memory in the computer or use smaller databases.
273	Divide by zero	An attempt has been made to divide a number by zero.
274	Invalid argument passed	An invalid argument has been passed to a Cicode function. This is a general error message and is generated when arguments passed to a function are out of range or are

Error No.	Error title	Description
		invalid. Check the value of arguments being passed to the function. If arguments are input directly from the operator, you should check that the correct arguments are being passed to the function.
275	Overflow	A calculation has resulted in a numeric value overflow. Check for operations that will generate large numbers.
276	No privilege for operation	A user has requested an operation for which he or she has no privilege.
277	Not in correct area	A user has requested data that does not belong to the current user area.
278	Report already busy	A request has been made to run a report that is already running. You can get the current state of a report with the RepGetControl() function. You can ignore this error message (because the report is already running).
279	Report is late for execution	The report cannot run at the rate requested in the configuration. An attempt could have been made to run a report too frequently, and the required data cannot be read from the I/O Device(s) in time for the next report.
280	Invalid report ID specified	The specified report name does not exist, or the user has no privilege to run the report, or the report is not in the current user area. Check the name of the report and the current user's privilege and areas.
281	No server could be found	The specified CitectSCADA server cannot be found. Either the server is not running or there is some disturbance in the network. Check that the network is set up correctly, and you are using the same Server Name on both the client and server.
282	Foreground Cicode cannot block	You cannot block the foreground Citect-SCADA task. You may have called a blocking function from one of the Page animation databases.

Error No.	Error title	Description
283	Trend has missed samples	The trend cannot run at the rate requested in the configuration. An attempt could have been made to trend the data too frequently, and the required data cannot be read from the I/O Device(s) in time for the next trend. Either increase the performance of the communication link to the PLCs or slow the rate of trend data acquisition.
284	Device is disabled	An attempt was made to access a device that is disabled. You can disable any devices (printers and other logging devices) with the DevDisable() function. When CitectSCADA (or your Cicode function) tries to access a disabled device, this message returns and all output is lost.
285	Foreground Cicode run is too long	The foreground Cicode task is taking too long to animate the display page. The Cicode is too complex and is taking too long to execute. Simplify the Cicode that is animating the page, or increase the [Code] TimeSlice parameter. If you cannot simplify the Cicode, you can create a separate task using TaskNew() to calculate your complex operation, and then use the Display functions to display the results. Cicode running from a TaskNew() call is in background mode and can run as long as required.
286	Out of Cicode threads	CitectSCADA has run out of Cicode tasks. Run fewer tasks (for example, reports, key commands, and Cicode tasks) in parallel, or increase the number of tasks with the [Code]Threads parameter. This error can be caused by a configuration error if you keep creating tasks that do not finish.
287	Floating point exception trap	An invalid floating-point number has been found. Check the floating-point data from the I/O Device.
288	Out of buffers	CitectSCADA is out of dynamic buffers. You have called a function that requests buffer space but no buffers are available. Check which function is causing the error and increase the associated buffers, or slow the rate of transfer to that function. If the error occurs on a server or LAN device, increase

Error No.	Error title	Description	
	the number of buffers with the [Lan]ReadPool parameter.	This error can also be detected if something is stopping the release of the buffers, for example, if network communication has stopped or a PLC has just come off-line. The error 'Out of buffers' can also be generated in the following ways:	
	Calling QueWrite() when the queue functions have run out of buffers. You can increase the number of queue buffers with the [Code] Queue parameter.	Calling WinFree() to free the last Cicode window. If WinFree() did free the last window, CitectSCADA would have no windows left.	
	To verify which function is causing the hardware error, display the {ERRPAGE} and {ERRDESC} fields on the hardware alarm.	289 Name does not exist	The specified name does not exist in this context. You are probably using the wrong name.
290 Not finished	A request has been made for trend data that has not yet finished trending.	291 File not CitectSCADA format	The specified file is not in CitectSCADA format. The file (trend, graphic, or any other file) is in an invalid format. Check that the name of the file is valid or that the file has not become corrupted.
292 Invalid function	The specified function name does not exist. You have tried to create a task, or called a remote procedure, or set an event function that does not exist.	293 File error	A general file error has been detected. Either a general hardware error has occurred, or the operating system is out of resources, or the file server is down.
294 File EOF	The end of the file was found. An attempt was made to read data off the end of a file or database.		

Error No.	Error title	Description
295	Cicode stack overflow	<p>A Cicode evaluation stack overflow has occurred. There are too many local function variables or nested function calls. Reduce the number of local variables or increase the [Code] Stack parameter.</p>
		<p>The Cicode stack is used to store local function variables and function calls. If you have many nested functions and a large number of local function variables, the Cicode stack may overflow. When the Cicode stack overflows, the Cicode that caused the overflow is halted.</p>
		<p>You can estimate the size of the stack by counting the maximum number of local function variables in the deepest function calls. For example, if function A has 10 variables and calls function B with 30 variables, which calls function C with 40 variables, the stack needs to be $10 + 30 + 40 = 80$ deep.</p>
296	Queue empty	<p>An attempt has been made to read an element from an empty queue.</p>
297	Semaphore owner died	<p>The owner of a Cicode semaphore was halted, killed, or returned without releasing the semaphore. Reset the shared resource back to a known state (because the task that died may have left it in a mess), and then continue. For example, if you are sharing a printer, do a form feed.</p>
298	Semaphore timeout	<p>The requested semaphore was still in use after the specified timeout. Either try to get the semaphore again or abort the operation and tell the operator of the detected error.</p>
299	Cancelled	<p>The specified form or command was cancelled. This error is returned when a user presses the Cancel button on a form. The normal procedure is to abort the operation.</p>
300	Trend not found	<p>The trend does not exist at the specified AN and page. A trend function may have been called when the trend is not defined for that AN.</p>
301	Trend pen not found	<p>The required trend pen name does not exist</p>

Error No.	Error title	Description
		in the Trends database or is not in the current user area. Check that the pen name exists and check the current user's privilege and area.
302	Trend data not valid	The requested trend data is not valid. Either the I/O Device data was bad, or the CitectSCADA trend server was shut down, or the trend data was disabled.
303	Invalid animation number	The AN specified in the function is not defined. You called one of the DspXXX animation functions, but you specified an animation number that was out of range or that had been deleted.
304	File server inoperative, stand-by active	CitectSCADA has detected that a file server has become inoperative, and will switch to the standby file server. The file server's inoperative condition is due to errors of the network or of the file server computer. This error is displayed only if you have enabled redundant file servers. If a redundant file server is not enabled, CitectSCADA and Windows become inoperative when the file server becomes inoperative. You should report this error to the operators to fix the file server.
305	Conflicting types of animation	The same AN is being used for two different types of animation. This error is detected if you try to display two (or more) incompatible types of animation on the same AN (for example, you try to display a symbol at a AN where a bar graph is already displayed). Check the configuration. If you need to display a new animation, you need to first delete the old animation with the DspDel() function.
306	SQL field value truncated	A maximum of 1000bytes (1Kb) can be returned from a single field call. If the field data is larger than this limit, it is truncated. You have tried to access a database where one of the fields is greater than 1000 bytes in size. Change the database field size to less than 1000 bytes so it can be accessed. In fact, you should change the field size to less than 256 bytes, the maximum allowable length of a Cicode string.

Error No.	Error title	Description
307	SQL database error	A general SQL error. Call the SQLErrMsg() function for details of the detected error.
308	SQL null field data returned	Data has been requested from a field that contained no data, or the SQL server does not support this type of field data. CitectSCADA will return an empty string. Call the SQLFieldInfo() function to list the fields in the database.
309	Trend data is gated	You have requested trend data that was gated (set to logging disabled) by the trigger expression (that is when it was acquired). The data is returned with the gated values set to 0.
310	Incompatible server version	Two servers are running incompatible versions of the CitectSCADA software. Install the latest version on each server. Contact Citect Technical Support to arrange for an upgrade.
311	Alarm tag synchronize error	When the Alarm server shuts down it writes an alarm save file. If the alarm server is in tag mode (rather than record mode) this message will display. You can set the mode with the [Alarm]SaveStyle parameter. You can ignore this message as it is an alert only.
312	MAPI generic error	A generic MAPI error has been detected. Call the MailError() function to retrieve the MAPI error.
313	No MAPI	The MAPI mail system is not installed, or incorrectly installed on the computer.
314	MAPI offline	The computer is not logged on to the MAPI mail system. Call the MailLogon() function to log on to the MAPI mail system.
315	MAPI no mail	No mail was available. This message is returned from the MailRead() function if no mail is available.
316	dBASE record locked by another	The dBASE file is being accessed by another

Error No.	Error title	Description
		<p>user. Check if the dBASE file has been opened in exclusive mode by the other user. This error can also be detected if another user is updating the dBASE file, and will usually occur if it is an indexed database, and the file is on a slow file server. You can adjust dBASE access with the [General]LockRetry and [General]LockDelay parameters.</p>
317	Not in this version	<p>The operation is not supported in this version of CitectSCADA. You need to upgrade to a higher version.</p>
318	Invalid page function	<p>You have called the PageGoto(), PageNext(), PagePrev(), PageDisplay(), or PageLast() as an exit command in the Pages database.</p>
319	Low physical memory	<p>CitectSCADA is low on physical memory. Increase available physical memory (not virtual memory). Reduce the size of SMARTDRV cache, close any other windows programs that are running, or add more RAM to your computer. You can set the minimum size of memory required by CitectSCADA with the [Memory]MinPhyK parameter. This parameter sets a value for the minimum physical memory before CitectSCADA will generate this error message.</p> <p>This error may also be detected if your swap file is large (that is greater than 20 Mb). Reduce the size of your swap file. The swap file is configured with the Windows Control Panel (386 Enhanced icon).</p>
320	Cannot free window	<p>The WinFree() function has been called but CitectSCADA has no windows left. (Be aware that the last window and any child windows owned by the last window cannot be removed.)</p>
321	Font cannot be found	<p>The specified font cannot be found. Check the font name.</p>
322	Cannot connect to LAN	<p>CitectSCADA has detected an error on the network.</p>

Error No.	Error title	Description
323	Super Genie not Associated	A Super Genie variable has not been associated correctly. This error can be detected if a variable passed to the Super Genie is the wrong data type or the variable does not exist. The error will also be detected if the Ass() function has not been called for the variable.
325	Project is not compiled	Changes have been made to the project while the system was running. Either restart the system or shutdown and re-compile.
326	Could not run the CitectSCADA compiler	The CitectSCADA compiler could not be found. Either the computer has run out of memory, or the compiler has been removed from its directory.
327	User type not found	An attempt was made to create a user of a type that has not been defined in the users database.
328	User already exists	An attempt was made to create a new user with the same name as an existing user.
329	Cannot have mixed trends	An attempt was made to display both a periodic trend and an event trend in the same trend window. Check the project configuration (Trend Tags and Page Trends databases) for mixed trends displayed in a trend window.
336	Event type trend is expected	One of the arguments passed to this trend function is only valid for event type trends.
337	Trend in file does not exist	The trend name inside the trend file does not exist in the trend database. It is likely that the trend file belongs to a trend which is deleted from the system configuration.
338	Plot Functions Sequence Mismatch	Plot functions are to be written in sequence since they depend on the data set up by other plot functions. Please refer to the description section for each Plot Function for the order of plot functions.
339	Plot Marker is not Defined	An undefined plot marker symbol has been used. Use the PlotSetMarker function before

Error No.	Error title	Description
		the PlotLine, PlotXYLine or PlotMarker functions.
340	Invalid subgroup size	The subgroup size specified for this SPC trend is not valid.
341	Trend Cursor Disabled	The trend cursor is currently disabled.
342	Debug break	The DebugBreak() Cicode function has been called. This indicates an invalid condition detected in user written Cicode. Enable the Cicode debugger to find the cause of the problem.
343	Foreground Cicode cannot break	A breakpoint has been hit in foreground Cicode. Foreground Cicode cannot be blocked. You can disable this error message in Debug Options, accessed through the Debug menu in the Cicode Editor.
344	Format overflow	This error occurs when the string being used to return an error message is not long enough for the information to go into it.
345	Trend data not ready	This alert is returned when the trend data is not ready to be returned. Try again later.
346	Dynamic Out of licence points	The dynamic point count has exceeded the point limit. See CitectSCADA Licence Point Count.
347	Assertion did not succeed in user Cicode	An assertion in your Cicode has been proven FALSE, and the task terminated. Assertions are made using the Assert() function. If you set the [Code]DebugMessage parameter to 1, the assertion is logged and the operator prompted.
348	Property does not exist	The tag property does not exist.
350	RDB file not found	A RDB file is not found. Try a full compile and re-start.
353	Cannot connect to FTP	Problems connecting to the FTP server for file copy. Check that the service is running correctly by using a 3rd party tool outside CitectSCADA.

Error No.	Error title	Description
354	Unrecognized object class	The automation object to be used is not known or registered.
355	Object has no interface	The automation objects interface no longer exists. This is either a logic or code error.
356	Object automation exception	Generic automation exception code for logging issues.
357	Too many arguments	Formatting issues likely due to too many arguments (automation)
358	Too few arguments	Less arguments available than expected (automation)
359	Named object already exists	An object tried to be created when it was already existed and it was not expected to exist. Check your cicode (automation)
360	Unrecognized named object	The name of an object did not match the internal records. Check your cicode (automation).
361	Page CTG/RDB record mismatch	An animation object id was not found in the records. Try a full compile and re-start.
362	Object event queue flooded	CitectSCADA has run out of Cicode tasks while processing ActiveX events. Create fewer concurrent ActiveX events, or increase the number of tasks with the [Code]Threads parameter.
363	Incorrect number of arguments	Internal error in cicode handling of objects. Try a full recompile and restart.
364	No 'this' argument	Internal error in cicode handling of objects. Try a full recompile and restart.
367	File cannot be printed	Unable to print the specified file on the specified port.
368	Animation number invalid	Unable to display the animation at the given AN.

Error No.	Error title	Description
369	Wrong type for text display	Unable to display the given animation as text.
370	No FileFind instances to close	There are no FileFind instances to close.
371	No dialback response returned	The retry count for the dialback response has been exceeded.
372	Unrecognised ActiveX object	The ActiveX graphics data was unable to be loaded as it is an unrecognised object.
374	Date Time Conflict	A Date and/or time conflict has been detected. If you are attempting a TrnAddHistory(), verify that the file you are adding does not have a conflicting time or date with existing trend files.
375	Password Expired	The password has expired. Change password or adjust the [General]PasswordExpiry settings.
376	Error Writing to Files of Trend	An error has occurred while attempting to write to a trend file.
377	Error Reading Files of Trend	An error has occurred while attempting to read from a trend file.
379	Cannot modify field	The Users.DBF file is protected from direct user manipulation, so cannot be modified.
380	Name Exists	The user name specified already exists in the Users.DBF.
381	File Format Error	The file is not in the desired format. It may be corrupted.
382	Page data / variable tag data mismatch	Page data / variable tag data mismatch with tag-based driver.
384	The code this queue/semaphore was waiting for exited due to an error	Cicode task has been flagged as locked or inactive. This will not apply if a Cicode task was killed deliberately by the Cicode logic.
385	Cannot download web-	Unable to download the webclient signature

Error No.	Error title	Description
	signature.xml	file.
388	OID check disabled	Attempting to open a DBF file without checking to see if it has changed.
391	Unsupported Cicode function in Citect Web Client	A number of Cicode functions can only be run in a non Web context. This error is flagging that such a function was tried from a Web client.
392	Timeout from Rnd Alarm Server	The timeout for receiving alarm data from the redundant alarm server has been exceeded.
393	RDB and associated page mismatch	The RDB page version does not match the one expected after compilation.
394	Remote license lost	The remote licence can not be found.
400	Project or file is Read-Only	The operation could not be completed because the project or file is read only.
401	Redundant server not found	The redundant server is current not available or was not found.
402	Cluster not specified	Cicode call is ambiguous because no cluster was specified.
403	Cluster not found	The provided cluster name was not seen as a valid cluster name.
404	Cluster name and tag mismatch	A cluster name and a tag with a cluster name prefix were both passed, but they do not match each other.
405	Cluster not connected	The operation cannot continue as the cluster is not connected.
406	Cluster already connected	The operation cannot continue as the cluster is already connected.
407	Databrowse pending	Databrowse session currently being connected and is not yet available for further commands.

Error No.	Error title	Description
408	Databrowse not supported	This normally means that data has been requested from the wrong client / server type, that is a client request on a server or vice versa.
409	Databrowse type not found	Browse type unknown. Check that the versions of citect32 are the same.
410	Databrowse session not found	iSession handle is invalid.
411	Databrowse session exists	Cannot open a session as it already exists.
412	Databrowse session EOF	Cannot browse beyond end of file. No more records left in the browse.
413	Databrowse field not found	The specified field is not present in the file.
414	Databrowse invalid field	The specified field is not valid for this browse function.
415	Databrowse no command	Browse command unknown. Check that the versions of citect32 are the same.
416	Databrowse no next cluster	Internal flag that all clusters have been processed. Users use this in cicode to determine that there are no more clusters.
417	Cluster required for operation	The operation cannot continue as the cluster is required for operation of a server.
418	No server of type on cluster	There is no server of the required type configured on the server.
419	Client / Server ID mismatch	An ID given to a Cicode function is not of the correct type.
420	Not available outside process	This functionality is not available outside a process boundary for example, AlarmFirstCatRec is only able to be called from inside the Alarm Server process.
421		

Error No.	Error title	Description
	Invalid tran handle detected	Citect's internal communications subsystem detected an error while trying to send a message.
422	Dial call did not succeed	An attempt to make a call to a remote I/O device did not succeed. You can find more information on the cause of this problem (if it is reproducible) by setting [Dial]Debug=1 and [Dial]DebugLevel=3 and looking in the syslog.dat.
423	Waiting for initial data	Occurs when attempted to read the value before the initial value has been retrieved from the device.
424	Tag not found	Occurs when attempt to read or subscribe to a tag that does not exist.
425	No connector	Not used.
426	Write to named pipe did not succeed	An attempt to send a message via a name pipe has did not succeed.
427	Redirect from non-Cicode task	A function cannot be redirected to another component as a proxy RPC unless it is in the context of a Cicode call.
428	Data browse record is invalid	The specified record is not valid for this browse function.
429	Can't plot symbol to printer	The Cicode PlotMarker() function can only plot a symbol to the display. This error occurs when the PlotMarker() Cicode function is used with a user-defined marker and a printer was specified as the output when PlotOpen() was called.
430	Alarm sort parameters mismatch	The [Alarm]SortMode parameter value on client is different as compared to the value on the alarm server. This value should be same in order to display alarms in correct order on the client.
431	Summary sort parameters mismatch	Values for [Alarm]SummarySort and [Alarm]SummarySortMode parameters on client are different as compared to the values on the alarm server. These values should be same in order to display alarm

Error No.	Error title	Description
		summary in correct order on the client.
432	Property not ready	A specific tag property has been read in synchronous mode, before the value has been retrieved from the server. (similar to subscription value is pending)
433	Cicode type mismatch	Some server side changes are now allowed for cicode, without restarting the client. This means that now it is possible to change a tag type to one that now cannot be converted as it was before. Eg val = TagA - TagB, If TagA has been converted from Integer to String, then the cicode will raise a cicode type mismatch as the '-' operation is not supported for strings
434	Invalid data conversion	Attempt to convert a value to a type that cannot store the value for example, UInt to Long, and the value is greater than the Long type can handle.
436	User password invalid	The password is incorrect. Check that the user name and password are typed in correctly.
437	Unable to load security provider	Cannot load or initialize the Windows security provider. This indicates that the installation of operating system may be corrupted or the Windows security provider was not included.
439	Authentication did not succeed	Cannot authenticate the given Windows user name and password. Check that the user name and password are typed in correctly
440	Authentication session does not exist	Cannot find the authentication session during the authentication process for the Windows user. This indicates that there was mis-communications in between client and server.
441	Role checking did not succeed	Cannot perform the role-check process for the Windows user. Check that the roles database in the project is not corrupted.
442	No linked role	The given Windows user is not linked to any role defined in the project.

Error No.	Error title	Description
443	Acquire user credentials did not succeed	Cannot obtain the user credential handle from Windows.
444	Acquire user access token did not succeed	Cannot obtain the user access token. This indicates that the security context of the Windows user was invalid.
445	Encrypt data did not succeed	Cannot encrypt the user information during the Windows user authentication process. This indicates that the security context of the Windows user was invalid.
446	Decrypt data did not succeed	Cannot decrypt the user information during the Windows user authentication process. This indicates that the security context of the Windows user was invalid.
447	The name of the table can not be found in the kernel.	Cicode UsrKernelTableInfo() has asked for a table that does not exist.
449	Invalid logout	The logout was called when there is no one logged in or the logged on user is system default user.
450	Invalid tag data	Invalid tag data has been detected.
453	Subscription value is pending	Occurs when a tag has been subscribed to, and attempted to read the value before the initial value has been retrieved from the server.
512	Time out error	Did not execute requested action on time.
513	Access denied error	Access denied to perform requested action.
514	Write unsuccessful	Can not write to view-only client. User needs to login with valid user credentials to get write access.
517	Can't swap pages from foreground	Cannot swap the page on foreground cicode.
518	Too Many arguments for function	Too many arguments have been passed to a Cicode function. Check the number of arguments being passed to the function. If argu-

Error No.	Error title	Description
		ments are input directly from the operator, you should check that the correct number of arguments are being passed to the function.
524	ComBreak is obsolete parameter	[Page]ComBreak and [Page]ComBreakText are obsolete parameters.
525	Tag in Last Usable Value	Tag quality is uncertain and its value is the last usable value.
526	Tran authentication unsuccessful	Login did not authenticate correctly.
527	High watermark reached on Tran	High watermark is reached on transaction, network is overloaded.
528	Reduced size of resized page	The specified resized page has been reduced in size to fit the constraints of the maximum height and width of 4096.
529	Too few arguments	Too few arguments have been passed to a Cicode function. Check the number of arguments being passed to the function. If arguments are input directly from the operator, you should check that the correct number of arguments are being passed to the function.
530	Null pointer as an argument	A string with invalid pointer has been passed to SCADA execution system.

Windows Socket Error Codes

The following CitectSCADA error codes are mapped to standard Windows socket errors. Generally these errors are beyond CitectSCADA's control, that is, there is an external reason why CitectSCADA cannot use the TCP/IP connection (assuming there are no Citect configuration issues).

Error No.	Error title	Description
1330	WSAENETDOWN	(10050) Network is down.
1331	Socket unreachable	(10051) Network is unreachable.
1332	Network reset	(10052) Network dropped connection on reset.

Error No.	Error title	Description
1333	Connection aborted	(10053) Software caused connection abort.
1334	Connection reset	(10054) Connection reset by peer.
1335	No buffers available	(10055) No buffer space available.
1340	Socket timeout	(10060) Connection timed out.
1341	Connection refused	(10061) Connection refused.
1343	Name too long	(10063) Item Name too long.
1344	Host down	(10064) Host is down.
1345	Host unreachable	(10065) No route to host.

MAPI Errors

The table below describes the MAPI errors in Cicode.

Error number	Error title	Description
0	SUCCESS_SUCCESS	The command completed successfully.
1	MAPI_USER_ABORT	The command was aborted by the user.
2	MAPI_E_FAILURE	General MAPI error.
3	MAPI_E_LOGIN_FAILURE	The login did not succeed, either the login user is unknown, misspelt, or the password is incorrect.
4	MAPI_E_DISK_FULL	The disk is full. The mail system will copy files into the temporary directory when mail is read. This can fill up the local hard disk.
5	MAPI_E_INSUFFICIENT_MEMORY	Insufficient memory to complete the command.

INSUF- FICIENT_MEM- ORY		
6	MAPI_E_ACCESS_DENIED	More privilege is required to complete the requested command.
8	MAPI_E_TOO_MANY_SESSIONS	You have tried to open too many sessions to the mail system.
9	MAPI_E_TOO_MANY_FILES	Too many attached files in a message.
10	MAPI_E_TOO_MANY_RECIPIENTS	Too many recipients for a mail message.
11	MAPI_E_ATTACHMENT_NOT_FOUND	Cannot find the attached file.
12	MAPI_E_ATTACHMENT_OPEN_FAILURE	Cannot open the specified attached file. Often because the file does not exist.
13	MAPI_E_ATTACHMENT_WRITE_FAILURE	Cannot write the attached file.
14	MAPI_E_UNKNOWN_RECIPIENT	Recipient of the mail message is unknown. Check if the user is configured in the mail system or check the spelling of the user's name.
15	MAPI_E_BAD_RECIPIENTTYPE	Unknown recipient type.
16	MAPI_E_NO_MESSAGES	No new messages to read.
17	MAPI_E_INVALID_MESSAGE	The mail message is invalid.
18	MAPI_E_TEXT_TOO_LARGE	Text message is too large to be sent. If you want to send large text messages, write the text to a file and attach the file to the mail message.

19	MAPI_E_INVALID_SESSION	Mail session is invalid. You should not get this error message; contact Technical Support for this product.
20	MAPI_E_TYPE_NOT_SUPPORTED	You should not get this error message; contact Technical Support for this product.
21	MAPI_E_AMBIGUOUS_RECIPIENT	The recipient of the mail message is ambiguous. Specify the exact user name to which the mail is to be sent.
22	MAPI_E_MESSAGE_IN_USE	Mail message is in use. You should not get this error message; contact Technical Support for this product.
23	MAPI_E_NETWORK_FAILURE	Mail cannot be sent or received as the network has experienced an error.
24	MAPI_E_INVALID_EDIT_FIELDS	You should not get this error message; contact Technical Support for this product.
25	MAPI_E_INVALID_RECIPS	You should not get this error message; contact Technical Support for this product.
26	MAPI_E_NOT_SUPPORTED	Command is not supported by this implementation of MAPI.

Chapter: 60 Browse Function Field Reference

See [Server Browse Function Fields](#) for Server fields.

Field Name	Description	Length	Values
ACKDATE	Event acknowledge date	12	short: dd-mm-yy [default] extended: dd-mm-yyyy Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise "". Note: The format can be changed by setting ExtendedDate to FALSE under the section [Alarm] in the citect.ini file.
ACKDATEEXT	Extended Event Acknowledge date	12	For non-Hardware alarms: dd-mm-yyyy
ACKTIME	Event acknowledge time	12	hh:mm:ss [am pm]. Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise ""
ACKUTC	Event Acknowledgment Time	12	Integer that represents a "time" value - the number of seconds since Jan 1, 1970 in UTC format not Local.
ACQERROR	Acquisition Error	6	Numeric value (integer)
ALARMTYPE	Alarm type (text)	16	"Digital", "Analog", "Advanced", "Multi-Digital", "Argyle Analog", "Time Stamped", "Time Stamped Digital", "Time Stamped Analog"
ALMCOMMENT	Alarm comment	254	
AREA	Alarm area	16	Numeric value (integer)
CATEGORY	Alarm category	16	Numeric value (integer)

CLUSTER	The cluster the tag exists on	32	
COMMENT	Comment	64	If hardware alarm = "". Or configured event description or trend comment
CUSTOM1	Alarm custom field #1	64	
CUSTOM2	Alarm custom field #2	64	
CUSTOM3	Alarm custom field #3	64	
CUSTOM4	Alarm custom field #4	64	
CUSTOM5	Alarm custom field #5	64	
CUSTOM6	Alarm custom field #6	64	
CUSTOM7	Alarm custom field #7	64	
CUSTOM8	Alarm custom field #8	64	
DATE	Alarm Date in default format	12	short: dd-mm-yy [default] extended: dd-mm-yyyy Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise "". Note: The format can be changed by setting ExtendedDate to FALSE under the section [Alarm] in the citect.ini file.
DATEEXT	Extended Alarm date	12	dd-mm-yyyy
DEADBAND		12	

	Alarm dead-band		For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
DELTATIME	Event on duration	12	hh:mm:ss
DESC	Alarm description	254	For Analog Alarms "OFF", "ON", "ON State 2", "ON State 3", "ON State 4", "ON State 5", "ON State 6", "ON State 7", "DEVIATION", "RATE OF CHANGE", "LOW", "HIGH", "LOW LOW", "HIGH HIGH", "Invalid". For others - configured descriptions.
DEVIATION	Alarm deviation	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
ENG_FULL	The engineering full scale for this value	11	Numeric value (real)
ENG_UNITS	Engineering units	8	The predefined values (see below) are in the Help.dbf of the bin directory but the user may specify it as a free text: %, Amps, deg, ft, ft/min, ft/s, ft/sec, gal, gal/h, gal/min, gal/s, gal/sec, Hz, kg, kg/h, kg/min, kg/s, kg/sec, km/h, kPa, kW, litres, lt, lt/h, lt/min, lt/s, lt/sec, m/min, m/s, m/sec, metres, Rev, Rev/h, RPM, t, t/h, Tonnes, Volts, Watts.
ENG_ZERO	The engineering zero scale for this value	11	Numeric value (real)
ERRDESC	Extra Citect info on alarms	80	For Hardware alarms: error description according to the configuration.

ERRPAGE	Alarm from this Citect page	80	For Hardware alarms: the Citect page the alarm is assigned to.
EXPRESSION	Logged variable name	65	The logged variable name (clustername.tagname). If the expression contains simple Cicode containing a single tag name, such as "LT131" or "LT131<1000", the tag name is returned (in both these cases LT131). If the expression contains a function call or more than one tag name, an empty string is returned.
FILENAME	File where the trend data is to be stored	231	
FILES	The number of history files stored on the hard disk (for this tag)	4	
FORMAT	Alarm format	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
FULLNAME	Event user full name	20	For non-Hardware alarms the configured full name of the event user if exists. Otherwise "System".
GROUP	Alarm group	16	For Argyle Digital alarms: Numeric value
HELP	Alarm help	254	The name of the help page.
HIGH	Alarm high	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
HIGHHIGH	Alarm highhigh	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)

LOCAL-TIMEDATE	Alarm Date and Time	24	yyyy-mm-dd hh:mm:ss[.ttt]. Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise ""
LOGSTATE	Log state text	13	"ACTIVE", "INACTIVE", "DISABLED", "ENABLED", "ACKNOWLEDGED", "LOW", "HIGH", "LOW LOW", "HIGH HIGH", "RATE", "DEVIATION", "Cleared", "UNACKNOWLEDGED"
LOW	Alarm low	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
LOWLOW	Alarm lowlow	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
LSL	Lower specification limit	16	
MILLISEC	Alarm milliseconds	6	Numeric value (integer)
NAME	Alarm name	80	
NATIVE_COMMENT	Event Comment	64	For non-Hardware alarms: the event description.
NATIVE_DESC	Alarm description	80	Analog alarm - Alarm state text. Otherwise - Configured alarm description.
NATIVE_NAME	Alarm name	80	Configured alarm name.
NATIVE_SUM-DESC	Event description text	80	For non-Hardware Analog alarms: the event state string. For non-Hardware other alarms: the event [or if not exists the alarm] description
NODE	Local node name	32	

OFFDATE	Event on date	12	short: dd-mm-yy [default] extended: dd-mm-yyyy Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise "". Note: The format can be changed by setting ExtendedDate to FALSE under the section [Alarm] in the citect.ini file.
OFFDATEEXT	Extended Event off date	12	For non-Hardware alarms: dd-mm-yyyy
OFFMILLI	Alarm summary milliseconds	6	For High Resolution, Times-tamped Digital and Timestamped Analog alarms it is the milliseconds part of the off time
OFFTIME	Event off time	12	hh:mm:ss [am pm]. Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise ""
OFFTIMEDATE	Special SQL formatted date and time	12	For non-Hardware alarms: HH:MM:SS. Note: The format can be configured in the citect.ini file by specifying TimeDate under the section [Alarm].
OFFUTC	Event off time		Integer that represents a "time" value - the number of seconds since Jan 1, 1970 in UTC format not Local.
OLD_DESC	Argyle old state desc.	8	"Invalid", "OFF", "ON", "ON State 2", "ON State 3", "ON State 4", "ON State 5", "ON State 6", "ON State 7"
ONDATE	Event on date	12	short: dd-mm-yy [default] extended: dd-mm-yyyy Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise "". Note: The format can be changed

			by setting ExtendedDate to FALSE under the section [Alarm] in the citect.ini file.
ONDATEEXT	Extended Event on date	12	For non-Hardware alarms: dd-mm-yyyy
ONMILLI	Alarm summary milliseconds	6	For High Resolution, Times-tamped Digital and Timestamped Analog alarms it is the milliseconds part of the on time
ONTIME	Event on time	12	hh:mm:ss [am pm]. Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise ""
ONTIMEDATE	Special SQL formatted date and time	12	For non-Hardware alarms: HH:MM:SS. Note: The format can be configured in the citect.ini file by specifying TimeDate under the section [Alarm].
ONUTC	Event on time		Integer that represents a "time" value - the number of seconds since Jan 1, 1970 in UTC format not Local.
ORATODATE	Special SQL formatted time	12	The first 12 characters of the string: "TO_DATE('<sql_time_date>', '<sql_time_format>')."
ORA-TOOFFDATE	Special SQL formatted time	12	For non-Hardware alarms: The first 12 characters of the string: "TO_DATE('<sql_off_time_date>', '<sql_time_format>')."
ORA-TOONDATE	Special SQL formatted time	12	For non-Hardware alarms: The first 12 characters of the string: "TO_DATE('<sql_on_time_date>', '<sql_time_format>')."
PAGING	Alarm paged flag	8	A flag to indicate that the alarm is going to be paged. Values are 1 (TRUE) or 0 (FALSE).

PAGINGGROUP	Paging group for alarm	80	A freeform text field indicating the sequence of people to notify in the event the alarm occurred.
PERIOD	The period of the history file	32	In hh:mm:ss (hours:minutes:seconds).
PRIORITY	Alarm category priority	4	Numeric value (integer)
PRIV	Alarm privilege	16	Numeric value (integer)
RANGE	Process range	16	
RATE	Alarm rate	12	For Analog, Argyle Analog and Timestamped Analog alarms: Numeric value (real)
RAW_FULL	The raw full scale for this value	11	Numeric value (real)
RAW_ZERO	The raw zero scale for this value	11	Numeric value (real)
RUNNING	Running time in seconds	16	Numeric value (real)
SAMPLEPER	Sample period of the trend in seconds	16	Numeric value (real)
SDEVIATION	Standard deviation	16	
SPCFLAG	Indicates tag is a Statistical Process Control (SPC) tag		
STARTS	Number of starts	16	Numeric value (real)
STATE	Alarm state string	16	"OFF", "ON", "ON State 2", "ON State 3", "ON State 4", "ON State 5", "ON State 6", "ON State 7",

			"DEVIATION", "RATE OF CHANGE", "LOW", "HIGH", "LOW LOW", "HIGH HIGH", "Invalid"
STATE_DESC	Argyle state description	8	"Invalid", "OFF", "ON", "ON State 2", "ON State 3", "ON State 4", "ON State 5", "ON State 6", "ON State 7"
STATE_DESC0	Argyle state 0	64	For Argyle Digital alarms: "OFF". For others: "INVALID"
STATE_DESC1	Argyle state 1	64	For Argyle Digital alarms: "ON". For others: "INVALID"
STATE_DESC2	Argyle state 2	64	For Argyle Digital alarms: "ON state 2". For others: "INVALID"
STATE_DESC3	Argyle state 3	64	For Argyle Digital alarms: "ON state 3". For others: "INVALID"
STATE_DESC4	Argyle state 4	64	For Argyle Digital alarms: "ON state 4". For others: "INVALID"
STATE_DESC5	Argyle state 5	64	For Argyle Digital alarms: "ON state 5". For others: "INVALID"
STATE_DESC6	Argyle state 6	64	For Argyle Digital alarms: "ON state 6". For others: "INVALID"
STATE_DESC7	Argyle state 7	64	For Argyle Digital alarms: "ON state 7". For others: "INVALID"
STORMETHOD	Storage method for the SPC data	16	"Scaled" or "Floating Point"

SUBGRPSIZE	The size of each subgroup	8	
SUMDESC	Event description text	80	Analog alarm - Event state text. Otherwise - Configured event [or alarm in case event not defined] description
SUMSTATE	Event state text	16	"OFF", "ON", "ON State 2", "ON State 3", "ON State 4", "ON State 5", "ON State 6", "ON State 7", "DEVIATION", "RATE OF CHANGE", "LOW", "HIGH", "LOW LOW", "HIGH HIGH", "Invalid"
SUMTYPE	Event state	16	For non-Hardware alarms: "DISABLED", "UNACKNOWLEDGED", "ACKNOWLEDGED", "Cleared".
TAG	Tag name	80	
TAGEX	Extended Alarm tag	32	<cluster>.<tag> when cluster defined Otherwise <tag>.
TAGGENLINK	Indicates Tag was imported from an IO Device		Name of the IO Device from which this tag was generated
TIME	Alarm Time	12	hh:mm:ss [am pm]. Note: works between: UTC time 1/1/1980 - UTC time 31/12/2037, otherwise ""
TIMEDATE	Special SQL formatted time	24	HH:MM:SS. Note: The format can be configured in the citect.ini file by specifying TimeDate under the section [Alarm].
TOTALISER	Running total of value	16	Numeric value (real)
TRIGGER	Last trigger	5	0 OR 1

(accumulator fields)	value, used to check for rising edge of the trigger code		
TRIGGER (trend fields)	The variable tag that triggers data logging	65	<p>The variable tag (clustername.tagname) that triggers data logging.</p> <p>If the trigger contains simple Cicode containing a single tag name, such as "LT131<50", the tag name is returned (in this case LT131). If the expression contains a function call or more than one tag name, an empty string is returned.</p>
TYPE	Alarm or Trend type string	16	<p>For Alarms:</p> <p>"DISABLED", "UNACKNOWLEDGED", "ACKNOWLEDGED", "Cleared"</p> <p>For Trends:</p> <p>"1" - Periodic, "2" - Event, "3" - Periodic Event</p>
TYPENUM	Alarm type	4	Numeric value (integer)
USERDESC	Event user description	64	For non-Hardware alarms the configured user description if exists. Otherwise "".
USERNAME	Event user name	16	For non-Hardware alarms the configured name of the event user if exists. Otherwise "System".
USL	Upper specification limit	16	
VALUE (alarm fields)	Formatted alarm value	16	<p>Analog Alarms - numeric value formatted according to configuration.</p> <p>Others - "ON", "OFF"</p>
VALUE	The value to be	16	Numeric value (real)

(accumulator fields)	added to the totaliser while trigger is true
XDOUBLEBAR	Process mean

Server Browse Function Fields

Field Name	Description	Length	Values
CLUSTER	The name of the cluster to which the server belongs.	16	
COMMENT	Comment	48	
LEGA-CYPORT	The port number the server is listening on for legacy connections. If no legacy port number was configured, this field contains the default legacy port number for the server type.	16	Number between 1 and 65535.
MODE	Indicates whether the server is configured as a primary or standby.	16	For Alarm, Report and Trend servers: 1 - primary 0 - standby For I/O servers the MODE field is set to "0".
NAME	The name of the server	16	
NETADDR	The network addresses associated with the server.	70	A comma-separated list of one or more names. The field contains the network address names as configured, and not IP addresses.
PORT	The port number the server is listening on. If no port number was configured, this field contains the default port number for the server type.	16	Number between 1 and 65535.
TYPE	Specifies the purpose of the server	16	"Alarm", "I/O", "Report", "Trend"

Index

:

: operator 79

—_ObjectCallMethod Cicode function 158

—_ObjectGetProperty Cicode function 159

—_ObjectSetProperty Cicode function 159

A

Abs Cicode function 622

AccControl Cicode function 149

AccumBrowseClose Cicode function 150

AccumBrowseFirst Cicode function 151

AccumBrowseGetField Cicode function 152

AccumBrowseNext Cicode function 153

AccumBrowseNumRecords Cicode function 153

AccumBrowseOpen Cicode function 154

AccumBrowsePrev Cicode function 155

Accumulator Cicode functions 149

ActiveX Cicode functions 157

Alarm Cicode functions 171

AlarmAck Cicode function 175

AlarmAckRec Cicode function 178

AlarmActive Cicode function 179

AlarmCatGetFormat Cicode function 180

AlarmClear Cicode function 181

AlarmClearRec Cicode function 183

AlarmComment Cicode function 184

AlarmDelete Cicode function 185

AlarmDisable Cicode function 187

AlarmDisableRec Cicode function 189

AlarmDsp Cicode function 190

AlarmDspClusterAdd Cicode function 192

AlarmDspClusterInUse Cicode function 193

AlarmDspClusterRemove Cicode function 194

AlarmDspLast Cicode function 194

AlarmDspNext Cicode function 196

AlarmDspPrev Cicode function 197

AlarmEnable Cicode function 198

AlarmEnableRec Cicode function 200

AlarmEventQue Cicode function	201
AlarmFirstCatRec Cicode function	202
AlarmFirstPriRec Cicode function	204
AlarmFirstTagRec Cicode function	206
AlarmGetDelay Cicode function	207
AlarmGetDelayRec Cicode function	207
AlarmGetDsp Cicode function	209
AlarmGetFieldRec Cicode function	212
AlarmGetInfo Cicode function	215
AlarmGetOrderbyKey Cicode function	216
AlarmGetThreshold Cicode function	217
AlarmGetThresholdRec Cicode function	218
AlarmHelp Cicode function	219
AlarmNextCatRec Cicode function	220
AlarmNextPriRec Cicode function	221
AlarmNextTagRec Cicode function	223
AlarmNotifyVarChange Cicode function	224
AlarmQueryFirstRec Cicode function	226
AlarmQueryNextRec Cicode function	227
alarms, hardware	1217
AlarmSetDelay Cicode function	228
AlarmSetDelayRec Cicode function	229
AlarmSetInfo Cicode function	230
AlarmSetThreshold Cicode function	236
AlarmSetThresholdRec Cicode function	238
AlarmSplit Cicode function	239
AlarmSumAppend Cicode function	240
AlarmSumCommit Cicode function	242
AlarmSumDelete Cicode function	243
AlarmSumFind Cicode function	245
AlarmSumFirst Cicode function	247
AlarmSumGet Cicode function	248
AlarmSumLast Cicode function	250
AlarmSumNext Cicode function	251
AlarmSumPrev Cicode function	252
AlarmSumSet Cicode function	253
AlarmSumSplit Cicode function	255
AlarmSumType Cicode function	256
AlmSummaryAck Cicode function	257
AlmSummaryClear Cicode function	258
AlmSummaryClose Cicode function	258
AlmSummaryCommit Cicode function	259
AlmSummaryDelete Cicode function	260
AlmSummaryDeleteAll Cicode function	261

AlmSummaryDisable Cicode function	262
AlmSummaryEnable Cicode function	262
AlmSummaryFirst Cicode function	263
AlmSummaryGetField Cicode function	263
AlmSummaryLast Cicode function	265
AlmSummaryNext Cicode function	265
AlmSummaryOpen Cicode function	266
AlmSummaryPrev Cicode function	267
AlmSummarySetFieldValue Cicode function	268
AlmTagsAck Cicode function	269
AlmTagsClear Cicode function	270
AlmTagsClose Cicode function	270
AlmTagsDisable Cicode function	271
AlmTagsEnable Cicode function	272
AlmTagsFirst Cicode function	272
AlmTagsGetField Cicode function	273
AlmTagsNext Cicode function	274
AlmTagsNumRecords Cicode function	274
AlmTagsOpen Cicode function	275
AlmTagsPrev Cicode function	277
AnByName Cicode function	160
ArcCos Cicode function	623
ArcSin Cicode function	623
ArcTan Cicode function	624
AreaCheck Cicode function	660
argument structure	52
arguments	
default values	56
key sequences as	23
passing data to	31
arguments, string	31
Ass Cicode function	917
AssChain Cicode function	918
AssChainPage Cicode function	925
AssChainPopUp Cicode function	926
AssChainWin Cicode function	926
AssChainWinFree Cicode function	928
Assert Cicode function	660
Assert function	146
AssGetProperty Cicode function	930
AssGetScale Cicode function	932
AssInfo Cicode function	934
AssInfoEx Cicode function	936
AssMetadata Cicode function	919

AssMetadataPage Cicode Function	920
AssMetadataPopUp Cicode function	921
AssMetadataWin Cicode function	922
AssPage Cicode function	938
AssPopUp Cicode function	940
AssScaleStr Cicode function	942
AssTag Cicode function	944
AssTitle Cicode function	945
AssVarTags Cicode function	946
AssWin Cicode function	947
B	
background tasks	99
Beep Cicode function	661
bit operators	87
Breakpoint window	112
breakpoints	122
browse function field reference	1241
C	
calculations	
variables in	20
CallEvent Cicode function	475
caret (^) character	32
ChainEvent Cicode function	479
CharToStr Cicode function	887
Cicode Editor	
toolbar options	111
Cicode errors	1218
CitectColourToPackedRGB Cicode function	293
CitectInfo Cicode function	662
CitectVBA Watch window	116
Clipboard Cicode functions	281
ClipCopy Cicode function	281
ClipPaste Cicode function	282
ClipReadLn Cicode function	282
ClipSetMode Cicode function	283
ClipWriteLn Cicode function	284
cluster Cicode functions	285
ClusterActivate Cicode function	286
ClusterDeactivate Cicode function	286
ClusterFirst Cicode function	287
ClusterGetName Cicode function	287
ClusterIsActive Cicode function	288
ClusterNext Cicode function	289

ClusterServerTypes Cicode function	289
ClusterSetName Cicode function	290
ClusterStatus Cicode function	291
ClusterSwapActive Cicode function	292
code debugging	120
CodeSetMode Cicode function	1001
CodeTrace Cicode function	668
cohesion	139
color Cicode functions	293
ComClose Cicode function	299
comments	43
formatting	132
communication Cicode functions	299
ComOpen Cicode function	300
Compile Errors window	115
ComRead Cicode function	302
ComReset Cicode function	304
ComWrite Cicode function	304
constants, standards for	128
controlling tasks	100
converting integers to strings	79
copying	
variables	19
Cos Cicode function	625
coupling	140
CreateControlObject Cicode function	160
CreateObject Cicode function	163
D	
data	
displaying, using expressions	25
logging expression	26
returning	34
data operators	85
data types	49
Date Cicode function	1042
date functions	37
DateAdd Cicode function	1043
DateDay Cicode function	1044
DateInfo Cicode function	1045
DateMonth Cicode function	1046
DateSub Cicode function	1047
DateWeekDay Cicode function	1048
DateYear Cicode function	1049

DDE Cicode functions	307
DDEExec Cicode function	308
DDEhExecute Cicode function	309
DDEhGetLastError Cicode function	310
DDEhInitiate Cicode function	311
DDEhPoke Cicode function	313
DDEhReadLn Cicode function	313
DDEhRequest Cicode function	314
DDEhSetMode Cicode function	315
DDEhTerminate Cicode function	316
DDEhWriteLn Cicode function	317
DDEPost Cicode function	318
DDERead Cicode function	319
DDEWrite Cicode function	320
DebugBreak Cicode function	669
debugging	123
error trapping	145
debugging breakpoints	122
debugging code	120
debugging functions	44
DebugMsg Cicode function	670
DebugMsg function	145
DebugMsgSet Cicode function	671
decision-making	26
declaration standards, variable	126
declaration, function	50
declarations, formatting	129
default values for arguments	56
default variable values	64
defensive programming	141
DegToRad Cicode function	625
DelayShutdown Cicode function	671
DevAppend Cicode function	324
DevClose Cicode function	325
DevControl Cicode function	326
DevCurr Cicode function	327
DevDelete Cicode function	328
DevDisable Cicode function	329
DevEOF Cicode function	329
DevFind Cicode function	330
DevFirst Cicode function	331
DevFlush Cicode function	332
DevGetField Cicode function	333
DevHistory Cicode function	334

device Cicode functions	323
DevInfo Cicode function	334
DevModify Cicode function	336
DevNext Cicode function	338
DevOpen Cicode function	338
DevOpenGrp Cicode function	340
DevPrev Cicode function	341
DevPrint Cicode function	342
DevRead Cicode function	343
DevReadLn Cicode function	344
DevRecNo Cicode function	344
DevSeek Cicode function	345
DevSetField Cicode function	346
DevSize Cicode function	347
DevWrite Cicode function	348
DevWriteLn Cicode function	348
DevZap Cicode function	349
display Cicode functions	353
displaying data	25
DisplayRuntimeManager	672
DLL Cicode functions	447
DLLCall Cicode function	447
DLLCallEx Cicode Function	448
DLLOpen Cicode function	450
DriverInfo Cicode function	587
DspAnCreateControlObject Cicode function	358
DspAnFree Cicode function	359
DspAnGetArea Cicode function	359
DspAnGetMetadata Cicode function	360
DspAnGetMetadataAt Cicode function	361
DspAnGetPos Cicode function	362
DspAnGetPrivilege Cicode function	362
DspAnInfo Cicode function	363
DspAnInRgn Cicode function	364
DspAnMove Cicode function	365
DspAnMoveRel Cicode function	366
DspAnNew Cicode function	367
DspAnNewRel Cicode function	368
DspAnSetMetadata Cicode function	368
DspAnSetMetadataAt Cicode function	369
DspBar Cicode function	370
DspBmp Cicode function	371
DspButton Cicode function	372
DspButtonFn Cicode function	374

DspChart Cicode function	376
DspCol Cicode function	377
DspDel Cicode function	377
DspDelayRenderBegin Cicode function	378
DspDelayRenderEnd Cicode function	379
DspDirty Cicode function	380
DspError Cicode function	381
DspFile Cicode function	382
DspFileGetInfo Cicode function	383
DspFileGetName Cicode function	384
DspFileScroll Cicode function	385
DspFileSetName Cicode function	386
DspFont Cicode function	387
DspFontHnd Cicode function	388
DspFullScreen Cicode function	389
DspGetAnBottom Cicode function	390
DspGetAnCur Cicode function	391
DspGetAnExtent Cicode function	392
DspGetAnFirst Cicode function	393
DspGetAnFromPoint Cicode function	393
DspGetAnHeight Cicode function	395
DspGetAnLeft Cicode function	395
DspGetAnNext Cicode function	396
DspGetAnRight Cicode function	396
DspGetAnTop Cicode function	397
DspGetAnWidth Cicode function	398
DspGetEnv Cicode function	398
DspGetMouse Cicode function	399
DspGetMouseOver Cicode function	400
DspGetNearestAn Cicode function	400
DspGetParentAn Cicode function	401
DspGetSlider Cicode function	402
DspGetTip Cicode function	402
DspGrayButton Cicode function	403
DspInfo Cicode function	404
DspInfoDestroy Cicode function	406
DspInfoField Cicode function	407
DspInfoNew Cicode function	408
DspInfoValid Cicode function	410
DspIsButtonGray Cicode function	410
DspKernel Cicode function	411
DspMarkerMove Cicode function	412
DspMarkerNew Cicode function	413
DspMCI Cicode function	414

DspPlaySound Cicode function	415
DspPopupConfigMenu Cicode function	417
DspPopupMenu Cicode function	417
DspRichText Cicode function	420
DspRichTextEdit Cicode function	421
DspRichTextEnable Cicode function	422
DspRichTextGetInfo Cicode function	422
DspRichTextLoad Cicode function	423
DspRichTextPgScroll Cicode function	424
DspRichTextPrint Cicode function	425
DspRichTextSave Cicode function	426
DspRichTextScroll Cicode function	427
DspRubEnd Cicode function	428
DspRubMove Cicode function	428
DspRubSetClip Cicode function	429
DspRubStart Cicode function	430
DspSetSlider Cicode function	431
DspSetTip Cicode function	432
DspSetTooltipFont Cicode function	433
DspStatus Cicode function	434
DspStr Cicode function	435
DspSym Cicode function	436
DspSymAnm Cicode function	437
DspSymAnmEx Cicode function	438
DspSymAtSize Cicode function	439
DspText Cicode function	441
DspTipMode Cicode function	442
DspTrend Cicode function	443
DspTrendInfo Cicode function	445
DumpKernel Cicode function	672
E	
EngToGeneric Cicode function	674
EnterCriticalSection Cicode function	1002
EquipBrowseClose Cicode function	453
EquipBrowseFirst Cicode function	453
EquipBrowseGetField Cicode function	454
EquipBrowseNext Cicode function	455
EquipBrowseNumRecords Cicode function	456
EquipBrowseOpen Cicode function	457
EquipBrowsePrev Cicode function	458
EquipCheckUpdate Cicode function	458
EquipGetProperty Cicode function	459
Equipment Database Cicode functions	452

ErrCom Cicode function	461
ErrDrv Cicode function	462
ErrGetHw Cicode function	463
ErrHelp Cicode function	465
ErrInfo Cicode function	466
ErrLog Cicode function	466
ErrMsg Cicode function	467
error Cicode functions	461
error handling	142
error messages	1217
error trapping	145
errors	
Cicode	1218
general	1218
Hardware Cicode errors	1217
MAPI	1238
ErrSet Cicode function	468
ErrSetHw Cicode function	469
ErrSetLevel Cicode function	471
ErrTrap Cicode function	472
escape sequence character	32
escape sequences	83
evaluating functions	30
event Cicode functions	475
events	
handling	97
triggering	27
Exec Cicode function	674
ExecuteDTSPkg Cicode function	860
Exp Cicode function	626
expression data, logging	26
expressions	25
decision-making with	26
triggering events using	27
expressions, formatting	131
F	
Fact Cicode function	626
fields for browse functions	1241
file Cicode functions	493
file headers	42
FileClose Cicode function	494
FileCopy Cicode function	495
FileDelete Cicode function	496

FileEOF Cicode function	496
FileExist Cicode function	497
FileFind Cicode function	498
FileFindClose Cicode function	499
FileGetTime Cicode function	500
FileMakePath Cicode function	500
FileOpen Cicode function	501
FilePrint Cicode function	502
FileRead Cicode function	503
FileReadBlock Cicode function	504
FileReadLn Cicode function	505
FileRename Cicode function	505
FileRichTextPrint Cicode function	506
files	
include	21
using	16
Files window	116
FileSeek Cicode function	507
FileSetTime Cicode function	508
FileSize Cicode function	509
FileSplitPath Cicode function	509
FileWrite Cicode function	510
FileWriteBlock Cicode function	511
FileWriteLn Cicode function	512
FmtClose Cicode function	553
FmtFieldHnd Cicode function	554
FmtGetField Cicode function	555
FmtGetFieldCount Cicode function	556
FmtGetFieldHnd Cicode function	556
FmtGetFieldName Cicode function	557
FmtGetFieldWidth Cicode function	558
FmtOpen Cicode function	558
FmtSetField Cicode function	559
FmtSetFieldHnd Cicode function	560
FmtToStr Cicode function	561
for...do	92
foreground tasks	99
form Cicode functions	513
FormActive Cicode function	515
FormAddList Cicode function	515
format Cicode functions	553
format operator (:)	79
format templates	134

formatting	
comments	132
expressions	131
function headers	137
functions	132
simple declarations	129
text strings	81
formatting statements	130
FormButton Cicode function	516
FormCheckBox Cicode function	517
FormComboBox Cicode function	519
FormCurr Cicode function	520
FormDestroy Cicode function	521
FormEdit Cicode function	522
FormField Cicode function	523
FormGetCurrInst Cicode function	525
FormGetData Cicode function	526
FormGetInst Cicode function	527
FormGetText Cicode function	528
FormGoto Cicode function	528
FormGroupBox Cicode function	529
FormInput Cicode function	530
FormListAddText Cicode function	532
FormListBox Cicode function	533
FormListDeleteText Cicode function	534
FormListSelectText Cicode function	535
FormNew Cicode function	536
FormNumPad Cicode function	538
FormOpenFile Cicode function	539
FormPassword Cicode function	540
FormPosition Cicode function	541
FormPrompt Cicode function	542
FormRadioButton Cicode function	542
FormRead Cicode function	544
FormSaveAsFile Cicode function	545
FormSecurePassword	546
FormSecurePassword Cicode function	546
FormSelectPrinter Cicode function	547
FormSetData Cicode function	547
FormSetInst Cicode function	548
FormSetText Cicode function	549
FormWndHnd Cicode function	550
FTP Cicode functions	563
FTPClose Cicode function	563

FTPFileCopy Cicode function	564
FTPFileFind Cicode function	565
FTPFileFindClose Cicode function	566
FTPOpen Cicode function	567
FullName Cicode function	803
function headers	137
function libraries	41
functions	
arguments and	31
Assert	146
debugging	44
DebugMsg	145
declaration	50
evaluating	30
event handling	97
formatting	132
groups	41
keyboard	37
libraries	41
menu	639
miscellaneous	37
page	36
report	37
returning values	58
scope of	48
structure of	39
syntax	46
table	72
time and date	37
FuzzyClose Cicode function	569
FuzzyGetCodeValue Cicode function	570
FuzzyGetShellValue Cicode function	571
FuzzyOpen Cicode function	572
FuzzySetCodeValue Cicode function	573
FuzzySetShellValue Cicode function	573
FuzzyTech Cicode functions	569
FuzzyTrace Cicode function	574
G	
general errors	1218
GetArea Cicode function	676
GetBlueValue Cicode function	294
GetEnv Cicode function	676
GetEvent Cicode function	479

GetGreenValue Cicode function	294
GetLogging Cicode function	677
GetPriv Cicode function	803
GetRedValue Cicode function	295
GetWinTitle Cicode function	1179
Global variable window	113
group Cicode functions	577
groups of functions	41
GrpClose Cicode function	577
GrpDelete Cicode function	578
GrpFirst Cicode function	579
GrpIn Cicode function	580
GrpInsert Cicode function	580
GrpMath Cicode function	581
GrpName Cicode function	582
GrpNext Cicode function	583
GrpOpen Cicode function	584
GrpToStr Cicode function	585
H	
Halt Cicode function	1004
handling events	97
handling, error	142
hardware alarms	1217
Hardware Cicode errors	1217
headers, file	42, 136
headers, function	137
HexToStr Cicode function	887
HighByte Cicode function	627
HighWord Cicode function	627
HtmlHelp Cicode function	1179
HwAlarmQue Cicode function	278
I	
I/O device Cicode functions	587
if...then	91
IFDEF macro	73
IFDEFAdvAlm macro	74
IFDEFAnaAlm macro	75
IFDEFDigAlm	77
include files	21
InfoForm Cicode function	678
InfoFormAn Cicode function	679
Input Cicode function	680
input, runtime operator	23

integer data type	49
integers	79
IntToReal Cicode function	681
IntToStr Cicode function	888
IODeviceControl Cicode function	589
IODeviceInfo Cicode function	592
IODeviceStats Cicode function	597
IsError Cicode function	473
K	
KerCmd Cicode function	681
KernelQueueLength Cicode function	682
KernelTableInfo Cicode function	682
KernelTableItemCount Cicode function	683
KeyAllowCursor Cicode function	600
keyboard Cicode functions	599
keyboard functions	37
KeyBs Cicode function	601
KeyDown Cicode function	602
KeyGet Cicode function	602
KeyGetCursor Cicode function	603
KeyLeft Cicode function	604
KeyMove Cicode function	604
KeyPeek Cicode function	605
KeyPut Cicode function	606
KeyPutStr Cicode function	606
KeyReplay Cicode function	607
KeyReplayAll Cicode function	608
KeyRight Cicode function	608
KeySetCursor Cicode function	609
KeySetSeq Cicode function	610
KeyUp Cicode function	611
L	
labels, standards for	129
LanguageFileTranslate Cicode function	684
LeaveCriticalSection Cicode function	1004
libraries of functions	41
LLClose Cicode function	450
Ln Cicode function	628
Log Cicode function	629
logging expression data	26
logic	26
logical operators	89
Login Cicode function	805

LoginForm Cicode function	805
Logout Cicode function	806
LogoutIdle Cicode function	807
LowByte Cicode function	629
LowWord Cicode function	630
M	
macro arguments	78
mail Cicode functions	615
MailError Cicode function	615
MailLogoff Cicode function	616
MailLogon Cicode function	617
MailRead Cicode function	618
MailSend Cicode function	619
MakeCitectColour Cicode function	295
MAPI errors	1238
math/trigonometry Cicode functions	621
mathematical operators	85
Max Cicode function	631
menu Cicode functions	640
menu functions	639
MenuGetChild Cicode function	641
MenuGetFirstChild Cicode function	642
MenuGetGenericNode Cicode function	643
MenuGetNextChild Cicode function	643
MenuGetPageNode Cicode function	644
MenuGetProperty Cicode function	645
MenuGetParent Cicode function	645
MenuGetPrevChild Cicode function	645
MenuGetWindowNode Cicode function	646
MenuNodeAddChild Cicode function	646
MenuNodeGetProperty Cicode function	648
MenuNodeHasCommand Cicode function	648
MenuNodeIsDisabled Cicode function	649
MenuNodeIsHidden Cicode function	650
MenuNodeRemove Cicode function	651
MenuNodeRunCommand Cicode function	651
MenuNodeSetDisabledWhen Cicode function	652
MenuNodeSetHiddenWhen Cicode function	653
MenuNode SetProperty Cicode function	654
MenuReload Cicode function	655
Message Cicode function	685
messages, error	1217
Min Cicode function	631
miscellaneous Cicode functions	657

miscellaneous functions	37
modular programming	138-140
module variables	65
MsgBrdcst Cicode function	1005
MsgClose Cicode function	1006
MsgGetCurr Cicode function	1007
MsgOpen Cicode function	1008
MsgRead Cicode function	1010
MsgRPC Cicode function	1011
MsgState Cicode function	1013
MsgWrite Cicode function	1014
MultiMonitorStart Cicode function	1180
multiple arguments'arguments	
multiple	33
multiple statements	21
MultiSignatureForm	808
MultiSignatureTagWrite	809
multitasking	99-100
N	
Name Cicode function	810
naming standards, variables	127
O	
object data type	49
ObjectAssociateEvents Cicode function	165
ObjectAssociatePropertyWithTag Cicode function	166
ObjectByName Cicode function	167
ObjectHasInterface Cicode function	168
ObjectIsValid Cicode function	168
ObjectToStr Cicode function	169
OLEDateToTime Cicode function	1050
one-dimensional variable arrays	70
OnEvent Cicode function	483
operator precedence	90
operator, format	79
operators	
bit	87
logical	89
relational	88
operators, data	85
Output window	113
P	
PackedRGB Cicode function	296
PackedRGBToCitectColour Cicode function	297

page Cicode functions	711
page functions	36
PageAlarm Cicode function	713
PageBack Cicode function	714
PageDisabled Cicode function	715
PageDisplay Cicode function	716
PageFile Cicode function	718
PageFileInfo Cicode function	719
PageForward Cicode function	720
PageGetInt Cicode function	720
PageGetStr Cicode function	721
PageGoto Cicode function	722
PageHardware Cicode function	723
PageHistoryDspMenu Cicode function	724
PageHistoryEmpty Cicode function	725
PageHome Cicode function	725
PageInfo Cicode function	726
PageLast Cicode function	728
PageMenu Cicode function	729
PageNext Cicode function	730
PagePeekCurrent Cicode function	731
PagePeekLast Cicode function	732
PagePopLast Cicode function	732
PagePopUp Cicode function	733
PagePrev Cicode function	734
PageProcessAnalyst Cicode function	735
PageProcessAnalystPens Cicode function	737
PagePushLast Cicode function	738
PageRecall Cicode function	739
PageRichTextFile Cicode function	740
PageSelect Cicode function	741
PageSetInt Cicode function	742
PageSetStr Cicode function	743
PageSummary Cicode function	744
PageTask Cicode function	745
PageTransformCoords Cicode function	746
PageTrend Cicode function	747
ParameterGet Cicode function	686
ParameterPut Cicode function	687
passing data to arguments	31
PathToStr Cicode function	889
Pi Cicode function	632
plot Cicode functions	751
PlotClose Cicode function	752

PlotDraw Cicode function	752
PlotGetMarker Cicode function	754
PlotGrid Cicode function	755
PlotInfo Cicode function	757
PlotLine Cicode function	758
PlotMarker Cicode function	761
PlotOpen Cicode function	763
PlotScaleMarker Cicode function	765
PlotSetMarker Cicode function	767
PlotText Cicode function	768
PlotXYLine Cicode function	769
Pow Cicode function	632
pre-emptive multitasking	100
precedence, operator	90
Print Cicode function	350
PrintFont Cicode function	351
PrintLn Cicode function	352
private scope	48
Process Analyst Cicode functions	773
ProcessAnalystLoadFile Cicode function	773
ProcessAnalystPopup Cicode function	774
ProcessAnalystSelect Cicode function	776
ProcessAnalystSetPen Cicode function	777
ProcessAnalystWin Cicode function	778
ProcessIsClient Cicode function	688
ProcessIsServer Cicode function	688
ProcessRestart Cicode function	689
ProductInfo Cicode function	690
programming standards	125
programming, modular	138
ProjectInfo Cicode function	690
ProjectRestartGet Cicode function	691
ProjectRestartSet Cicode function	692
ProjectSet Cicode function	692
Prompt Cicode function	693
pseudocode	42
public scope	48
Pulse Cicode function	694
Q	
Quality Cicode functions	781
quality data type	49
QualityCreate Cicode function	781
QualityGetPart Cicode function	782

QualityIsBad Cicode function	784
QualityIsControlMode Cicode function	783
QualityIsGood Cicode function	785
QualityIsOverride Cicode function	786
QualityIsUncertain Cicode function	786
QualitySetPart Cicode function	787
QualityToStr Cicode function	788
QueClose Cicode function	1015
QueLength Cicode function	1016
QueOpen Cicode function	1016
QuePeek Cicode function	1017
QueRead Cicode function	1019
QueryFunction Cicode function	279
QueWrite Cicode function	1020
R	
RadToDeg Cicode function	633
Rand Cicode function	634
real data type	49
RealToStr Cicode function	889
relational operators	88
RepGetControl Cicode function	797
Report Cicode function	797
report Cicode functions	795
report functions	37
ReportGetCluster Cicode function	795
RepSetControl Cicode function	798
ReRead Cicode function	1021
returning data from functions	34
returning values from functions	58
Round Cicode function	634
runtime operator input	23
runtime operator input triggering	29
S	
scope	64, 126
scope, function	48
security Cicode functions	801
select case	94
SemClose Cicode function	1022
SemOpen Cicode function	1022
SemSignal Cicode function	1023
SemWait Cicode function	1024
SendKeys Cicode function	611
sequences, escape	83

SerialKey Cicode function	306
server Cicode functions	827
ServerBrowseClose Cicode function	828
ServerBrowseFirst Cicode function	829
ServerBrowseGetField Cicode function	831
ServerBrowseNext Cicode function	830
ServerBrowseNumRecords Cicode function	832
ServerBrowseOpen Cicode function	829
ServerBrowsePrev Cicode function	830
ServerGetProperty Cicode function	832
ServerInfo Cicode function	834
ServerInfoEx Cicode function	836
ServerIsOnline Cicode function	839
ServerReload Cicode function	840
ServerRestart Cicode function	841
ServerRPC Cicode function	1025
ServiceGetList Cicode function	694
SetArea Cicode function	695
SetEvent Cicode function	488
SetLanguage Cicode function	696
SetLogging Cicode function	698
setting	
variables	19
Shutdown Cicode function	699
ShutdownForm Cicode function	701
ShutdownMode Cicode function	702
Sign Cicode function	635
Sin Cicode function	635
Sleep Cicode function	1026
SleepMS Cicode function	1027
source file headers	136
SPC Cicode functions	843
SPCALarms Cicode function	844
SPCClientInfo Cicode function	845
SPCGetHistogramTable Cicode function	846
SPCGetSubgroupTable Cicode function	847
SPCPlot Cicode function	849
SPCProcessXRSGet Cicode function	850
SPCProcessXRSSet Cicode function	851
SPCSelLimit Cicode function	852
SPCSelLimitGet Cicode function	853
SPCSelLimitSet Cicode function	855
SPCSubgroupSizeGet Cicode function	856
SPCSubgroupSizeSet Cicode function	857

SQL Cicode functions	859
SQLAppend Cicode function	862
SQLBeginTran Cicode function	863
SQLCommit Cicode function	864
SQLConnect Cicode function	865
SQLDisconnect Cicode function	868
SQLEnd Cicode function	869
SQLErrMsg Cicode function	870
SQLExec Cicode function	871
SQLFieldInfo Cicode function	876
SQLGetField Cicode function	877
SQLInfo Cicode function	878
SQLNext Cicode function	879
SQLNoFields Cicode function	879
SQLNumChange Cicode function	880
SQLRollBack Cicode function	881
SQLSet Cicode function	882
SQLTraceOff Cicode function	883
SQLTraceOn Cicode function	883
Sqrt Cicode function	636
Stack window	114
standards, for constants, variables and labels	128
standards, naming, for variables	127
standards, programming	125
statements	
using multiple	21
statements, executable	130
stepping through code	123
StrCalcWidth Cicode function	890
StrClean Cicode function	890
StrFill Cicode function	891
StrFormat Cicode function	892
StrGetChar Cicode function	893
string arguments	31
string Cicode functions	885
string data type	49
strings	79
strings, formatting	81
StrLeft Cicode function	893
StrLength Cicode function	894
StrLower Cicode function	895
StrMid Cicode function	895
StrPad Cicode function	896
StrRight Cicode function	897

StrSearch Cicode function	897
StrSetChar Cicode function	898
StrToChar Cicode function	899
StrToDate Cicode function	900
StrToFmt Cicode function	901
StrToGrp Cicode function	901
StrToHex Cicode function	902
StrToInt Cicode function	903
StrToLines Cicode function	904
StrToLocalText Cicode function	905
StrToPeriod Cicode function	906
StrToReal Cicode function	906
StrToTime Cicode function	907
StrToValue Cicode function	908
StrTrim Cicode function	909
StrTruncFont Cicode function	910
StrTruncFontHnd Cicode function	911
structure of functions	39
structure, argument	52
StrUpper Cicode function	911
StrWord Cicode function	912
SubscriptionAddCallback Cicode function	958
SubscriptionGetAttribute Cicode function	959
SubscriptionGetInfo Cicode function	961
SubscriptionGetQuality Cicode function	962
SubscriptionGetTag Cicode function	962
SubscriptionGetTimestamp Cicode function	963
SubscriptionGetValue Cicode function	964
SubscriptionRemoveCallback Cicode function	965
SuperGenie Cicode functions	915
SwitchConfig Cicode function	703
syntax	45
SysTime Cicode function	1051
SysTimeDelta Cicode function	1052
T	
table (array) Cicode functions	951
TableLookup Cicode function	951
TableMath Cicode function	952
TableShift Cicode function	954
Tag Cicode functions	957
Tag Reference /TagReadEx() behaviour in Cicode Expressions	44
TagDebug Cicode function	965
TagEventFormat Cicode function	966

TagEventQueue Cicode function	967
TagGetProperty Cicode function	970
TagGetScale Cicode function	972
TagInfo Cicode function	973
TagInfoEx Cicode function	975
TagRamp Cicode function	977
TagRDBReload Cicode function	979
TagRead Cicode function	979
TagReadEx Cicode function	982
TagScaleStr Cicode function	984
TagSetOverrideBad Cicode function	985
TagSetOverrideGood Cicode function	987
TagSetOverrideQuality Cicode function	988
TagSetOverrideUncertain Cicode function	990
TagSubscribe Cicode function	992
TagUnsubscribe Cicode function	994
TagWrite Cicode function	995
TagWriteEventQue Cicode function	997
Tan Cicode function	637
task Cicode functions	999
TaskCall Cicode function	1029
TaskCluster Cicode function	1030
TaskGetSignal Cicode function	1031
TaskHnd Cicode function	1031
TaskKill Cicode function	1032
TaskNew Cicode function	1033
TaskNewEx Cicode function	1036
TaskResume Cicode function	1038
tasks	
controlling	100
tasks, foreground and background	99
TaskSetSignal Cicode function	1038
TaskSuspend Cicode function	1039
TestRandomWave Cicode function	703
TestSawWave Cicode function	704
TestSinWave Cicode function	705
TestSquareWave Cicode function	706
TestTriangWave Cicode function	707
text files	21
text strings, formatting	81
Thread window	114
threads	99
time and date Cicode functions	1041
Time Cicode function	1053

time functions	37
TimeCurrent Cicode function	1054
TimeHour Cicode function	1055
TimeInfo Cicode function	1055
TimeIntToTimestamp Cicode functions	1063
TimeMidNight Cicode function	1056
TimeMin Cicode function	1057
TimeSec Cicode function	1058
TimeSet Cicode function	1059
Timestamp Cicode functions	1063
timestamp data type	49
TIMESTAMP data type	62-63
TimestampAdd Cicode function	1065, 1070
TimestampCreate Cicode function	1066
TimestampCurrent Cicode function	1066
TimestampDifference Cicode function	1068
TimestampFormat Cicode function	1069
TimestampSub Cicode function	1071
TimestampToStr Cicode function	1073
TimestampToTimeInt Cicode function	1074
TimeToOLEDate Cicode function	1060
TimeToStr Cicode function	1060
TimeUTCOffset Cicode function	1062
Toggle Cicode function	708
TraceMsg Cicode function	709
trapping, error	145
trend Cicode functions	1079
TrendDspCursorScale Cicode function	1083
TrendDspCursorTag Cicode function	1084
TrendDspCursorTime Cicode function	1085
TrendDspCursorValue Cicode function	1086
TrendGetAn Cicode function	1086
TrendPopUp Cicode function	1087
TrendRun Cicode function	1088
TrendSetDate Cicode function	1089
TrendSetScale Cicode function	1089
TrendSetSpan Cicode function	1090
TrendSetTime Cicode function	1091
TrendSetTimebase Cicode function	1092
TrendWin Cicode function	1092
TrendZoom Cicode function	1094
triggering	
runtime operator input	29
triggering events	27

TrnAddHistory Cicode function	1095
TrnBrowseClose Cicode function	1096
TrnBrowseFirst Cicode function	1097
TrnBrowseGetField Cicode function	1097
TrnBrowseNext Cicode function	1098
TrnBrowseNumRecords Cicode function	1099
TrnBrowseOpen Cicode function	1100
TrnBrowsePrev Cicode function	1101
TrnClientInfo Cicode function	1101
TrnComparePlot Cicode function	1103
TrnDelete Cicode function	1105
TrnDelHistory Cicode function	1105
TrnEcho Cicode function	1106
TrnEventGetTable Cicode function	1107
TrnEventGetTableMS Cicode function	1109
TrnEventSetTable Cicode function	1111
TrnEventSetTableMS Cicode function	1113
TrnExportClip Cicode function	1115
TrnExportCSV Cicode function	1117
TrnExportDBF Cicode function	1119
TrnExportDDE Cicode function	1122
TrnFlush Cicode function	1125
TrnGetBufEvent Cicode function	1126
TrnGetBufTime Cicode function	1126
TrnGetBufValue Cicode function	1127
TrnGetCluster Cicode function	1128
TrnGetCursorEvent Cicode function	1129
TrnGetCursorMSTime Cicode function	1130
TrnGetCursorPos Cicode function	1131
TrnGetCursorTime Cicode function	1131
TrnGetCursorValue Cicode function	1132
TrnGetCursorValueStr Cicode function	1133
TrnGetDefScale Cicode function	1134
TrnGetDisplayMode Cicode function	1135
TrnGetEvent Cicode function	1136
TrnGetFormat Cicode function	1137
TrnGetGatedValue Cicode function	1138
TrnGetInvalidValue Cicode function	1139
TrnGetMode Cicode function	1140
TrnGetMSTime Cicode function	1140
TrnGetPen Cicode function	1142
TrnGetPenComment Cicode function	1143
TrnGetPenFocus Cicode function	1143
TrnGetPenNo Cicode function	1144

TrnGetPeriod Cicode function	1145
TrnGetScale Cicode function	1146
TrnGetScaleStr Cicode function	1147
TrnGetSpan Cicode function	1148
TrnGetTable Cicode function	1149
TrnGetTime Cicode function	1151
TrnGetUnits Cicode function	1153
TrnInfo Cicode function	1153
TrnIsValidValue Cicode function	1155
TrnNew Cicode function	1156
TrnPlot Cicode function	1157
TrnPrint Cicode function	1159
TrnSamplesConfigured Cicode function	1160
TrnScroll Cicode function	1160
TrnSelect Cicode function	1162
TrnSetCursor Cicode function	1163
TrnSetCursorPos Cicode function	1164
TrnSetDisplayMode Cicode function	1164
TrnSetEvent Cicode function	1166
TrnSetPen Cicode function	1167
TrnSetPenFocus Cicode function	1169
TrnSetPeriod Cicode function	1170
TrnSetScale Cicode function	1170
TrnSetSpan Cicode function	1172
TrnSetTable Cicode function	1172
TrnSetTime Cicode function	1174
U	
UserCreate Cicode function	811
UserCreateForm Cicode function	812
UserDelete Cicode function	812
UserEditForm Cicode function	813
UserInfo Cicode function	814
UserLogin Cicode function	815
UserPassword Cicode function	817
UserPasswordExpiryDays Cicode function	818
UserPasswordField Cicode function	819
UserSetStr Cicode function	819
UserUpdateRecord Cicode function	820
UserVerify Cicode function	821
using	
files	16
multiple statements	21

V

variable arrays	
functions	72
one-dimensional	70
variable scope	64
variable tags, standards for	128
VariableQuality Cicode function	792
variables	61, 83
copying	19
declaration standards	126
default values for	64
in calculations	20
module	65
naming standards	127
scope standards	126
setting	19
VariableTimestamp Cicode function	1075
VerifyPrivilegeForm	822
VerifyPrivilegeTagWrite	823
Version Cicode function	709

W

while...do	93
WhoAmI Cicode function	825
WinCopy Cicode function	1180
WinFile Cicode function	1182
WinFree Cicode function	1184
WinGetFocus Cicode function	1185
WinGetWndHnd Cicode function	1186
WinGoto Cicode function	1186
WinMode Cicode function	1187
WinMove Cicode function	1188
WinNew Cicode function	1189
WinNewAt Cicode function	1190
WinNext Cicode function	1194
WinNumber Cicode function	1194
WinPos Cicode function	1195
WinPrev Cicode function	1196
WinPrint Cicode function	1196
WinPrintFile Cicode function	1198
WinSelect Cicode function	1200
WinSetName Cicode function	1201
WinSize Cicode function	1201
WinStyle Cicode function	1202

WinTitle Cicode function	1203
WndFind Cicode function	1204
WndGetFileProfile Cicode function	1205
WndHelp Cicode function	1206
WndInfo Cicode function	1208
WndMonitorInfo Cicode function	1209
WndPutFileProfile Cicode function	1210
WndShow Cicode function	1211
WndViewer Cicode function	1212
writing functions	41

