Key

`Cicode commands` are in green text
`[optional parameter]` do not type the []s
Replace *italics* with a name or value.
Explorer | File | Run denotes application, menu and submenu.
(v*x*.*x*SP*x*) lists the minimum required version/service pack for a new feature.
Q*xxxx* refers to a Citect Knowledgebase article.
[*section*]*parameter* refers to a Citect.ini or project parameter.

ActiveX

Insertable ActiveX controls can be created and assigned to an Object variable using the CreateObject(), CreateControlObject() or DspAnCreateControlObject() function.

To handle events, declare a non-private function: `FUNCTION EventClass_EventName(OBJECT hThis[,event argument list])`

Some properties return objects with their own properties/methods. These must be copied into other object variables to access sub-properties/methods.

Arrays

Declaration: `datatype name[size] [= value1[, value2, ...]];`

First element number is always 0. Must be GLOBAL or MODULE scope—no local arrays.

Arrays can have up to 3 dimensions.

Example: `STRING sFields[2][3][5] // 30 element 3-d array`

Max Array Size: 32,767 elements (v7.5). See Q1128

Cicode Variables

Every Cicode variable must be declared before it is used. Multiple variables may be declared in one statement if they are the same datatype.

If an initial value is specified, it will be set on startup for global and module vars or when the function starts for local vars. Otherwise, values are set to the Default value (see table below).

Declaration: `<datatype> varName [=<initial value>];`
`<datatype> var1 [=<init val>], var2 [=<init val>];`

Data types	Default	Prefix	Size	Range
INT	0	i or n	32 bits	-2,147,483,648 to 2,147,483,647
REAL	0	r	64 bits	1.79D-308 to 1.79D+308 (15 digits of precision)
STRING	""	s	255 bytes	
QUALITY	Bad	q	64 bits	
TIMESTAMP	invalid	t	64 bits	January 1, 1601 to 30,828AD
OBJECT	invalid	o or h	64 bits	

Before v7.20SP5 module/global STRINGs were 128 bytes. Before v7.20r0 module/global REALs were 32 bits (-3.4E38 to 3.4E38 with 7 digits precision).

CiVBA

Calling CiVBA procedures from Cicode. See [Code]VBASupport.

Syntax: `val = VbCallReturn(VbCallRun(VbCallOpen("vbfn", args())));`

If the return value is not needed VbCallReturn may be omitted.

Comments

Comments may be a full line or at the end of a line of code. ! and // are equivalent

```
//Sample full line comment
!Alternate full line comment
/* block comment can
span multiple lines */
Message("Hello world", "", 0); //Inline comment
```

Conditional Statements

WHILE statement executes a block of statements repeatedly while the condition is true.

```
WHILE condition DO
    statement(s)...
END
```

FOR statement executes a block of statements repeatedly, incrementing the counter by 1 each loop until it exceeds the finish value. To step by values other than 1, use a WHILE loop. To exit early, set counter to finish+1

```
FOR counter = start TO finish DO
        statement(s)...
END
```

IF statement executes a block of statements if a condition is true with an optional Else statement.

```
IF condition Then
        statement(s)...
[ELSE
        Statement(s)…]
END
```

SELECT CASE statement executes different statements for each case of a value. ELSE should be the last case. Only the first matching case is executed. Values may be matched to any combination of variables, numbers, strings, ranges and relational operators. Ranges and inequalities with strings compare alphabetically (case insensitive).

```
SELECT CASE value
CASE tagname, 1, 15        //individual values
    statement(s)...
CASE x To y, Is >= 5       //ranges and inequality
    statement(s)...
CASE ELSE
    statement(s)...
END SELECT
```

Conditional statements may be nested.

Constants

Constants may be defined using Project Editor | System menu | Labels.

Pseudo-constants use global, module, or local Cicode variables with initial values as constants. The programmer must remember to not write to them.

Conversions

Data type conversions are made automatically if possible.

The Addition operator causes a compile error if left and right values are not both numeric or both string types.

With INT to digital or logical conversion, 0 is FALSE and non-zero values become TRUE (1).

Debugger

Cicode Editor | Debug | Start Debugging connects to the runtime's client process (and starts it if needed). See [Debug]CodeDebug for server processes.
🟥 Debugging is disabled. 🟩 Debugging is enabled.

Debug | Insert Breakpoint sets a breakpoint ● on the current line (must contain a Cicode statement).

View | Step Into executes the next line of code and ⇨ indicates the next statement to be executed. If the line is calling a function written in Cicode, that function is displayed and stepped through. Use Step Over to execute the code without jumping into called functions or Step Out to execute the current function and return to the calling function.

Debugging

Use the `TraceMsg()` command to display diagnostic messages in the Kernel window.

Use `ErrLog()` or `DebugMsg()` to write to the Kernel and syslog.dat.

Example: `ErrLog("MyLog() Failed to open device");`

To view running tasks, go to the Kernel's Main window and type: PAGE TABLE CICODE <Enter>

Use `DspKernel(1)` to open the Kernel. See [Debug]Kernel, [Debug]Menu.

DLL Calls

When a DLL function is called the entire Citect runtime process will stop running until the DLL call completes.

The data types for arguments and the return value must match the DLL function's documentation.

If the DLL crashes, it can crash the Citect process. See [Code]DllCallProtect.

Error Handling

Many functions can cause a fatal error if they fail (such as DevOpen() with an invalid device name). A fatal error will immediately terminate the task and cause a hardware alarm.

Use ErrSetLevel() to disable automatic error checking and task termination. This allows you to check the value returned from the failed function call. If the function does not return the error code use IsError() to get the last error and reset the value to 0. See also [Code]HaltOnError.

Events

When certain predefined events occur (such as opening a page or shutting down the runtime) or user defined events occur, Citect can call a custom Cicode function to handle the event. See OnEvent().

Multiple handlers can be chained together with GetEvent() and ChainEvent().

The event handler should not have any parameters but should return 0. Some events can be cancelled by returning a non-zero value.

Foreground Tasks

A foreground task is one that needs to execute quickly, normally in one timeslice. For example, all expressions used to animate objects on a page execute in the page's foreground task and must complete before the page can refresh or a 'ForeGround Cicode Cannot Block' error occurs. See Q1077.

Some functions normally do not complete in one timeslice, such as reading from disk or getting user input, and cannot be called from a foreground task. They are normally documented as blocking functions. See TaskNew().

Format Operator

Converts a numeric value into a formatted string.

Examples: `tagname:###.#EU` or `(nVal * 5):#0##` or `rVal:#S##`

Format specifiers (formats must start with `#`):

`#` one digit	`.` location of the decimal point
`0` add leading zeros	`-` left justify
`S` scientific (exponential) notation	`EU` eng units for variable tags

Short notation uses numbers in place of #s, like `tagname:3.1`

Files/Functions

All Cicode files (*.ci) in a project are automatically compiled when the project is compiled and may contain any number of functions.

The number of lines of code in a function is limited by the compiled size (64k). Large functions can be split into smaller functions that call each other.

User functions can't write back to variables passed to it, but some functions built in to the product can, like QueRead() and _ObjectCallMethod(), requiring a Cicode variable, not an expression, to be passed.

Multi-Process Mode

If Multi-Process mode is selected in the Computer Setup Wizard (recommended), each runtime process (client, alarm, report, trend, I/O server) has its own copy of all global and module variables and local variable tags.

Use ServerRPC() or MsgRPC() to start Cicode tasks in other processes and optionally get return values. A client process cannot call directly to another client process or to a server process that it isn't currently connected to.

Multitasking

Multiple Cicode functions can run simultaneously in separate threads, regardless of whether Citect is in Multi-Process mode. See [Code]Threads.

Each thread is allowed to run for up to 100ms (see [Code]Timeslice and TimeSlicePage) before being pre-empted so the next thread can run.

A thread can execute a command that takes more than one time slice to return (such as a DLL call) which can't be preempted and will hold up all other threads in that runtime process.

Command buttons, events, the TaskNew() command, and other items that execute Cicode will create new threads. Some items like alarm on/off/ack actions run under a single thread. See Foreground Tasks.

Naming Rules

Function, variable, and label names must consist of A-Z, a-z, "_", and 0-9, beginning with a letter. See [General]TagStartDigit.

Do not use Cicode keywords or built-in function names. Function names may be 250 characters long (v7.1SP2). Names are not case sensitive.

Operators

Mathematical Operators: `+ - * / MOD`

Logical Operators: `AND OR NOT BITAND BITOR BITXOR`

Relational Operators: `< <= = >= > <>`

Format Operator: `:`

String Assignment Operator: `=`

Operator Precedence (highest to lowest). Use parentheses to override the default precedence.

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

Logical operators convert the values to boolean and return True (1) or False (0). Bit operators work with each individual bit in an INT or LONG.

Relational operators used with strings compare case-insensitively. See [Code]IgnoreCase. For inequalities, the strings are compared alphabetically using the ASCII table—non-English strings don't compare correctly (Q3750).

Queues

Queues consist of an unlimited number of elements (default 32,768—see [Code]Queue), each containing a 32-bit signed integer and a 255 character string.

Queues are meant to be processed as a first in, first-out buffer with QueRead() and QueWrite(), but QuePeek() can search the queue by element number (0 is the oldest element), integer value, or string value (case sensitive), optionally removing the found element.

Queues can only be accessed by the runtime process they were created in and can be viewed in the kernel with the commands PAGE TABLE USRQUE and PAGE QUEUE. Press Page Down to find the queue.

Recursion

Recursion means a function may call itself. Each call has its own local variables, but runs in the same thread. The local variables are stored in the Cicode Stack. If it runs out of space, the task will be terminated and cause a 'Cicode Stack Overflow' hardware alarm. See [Code]Stack.

Scope

Variables declared inside of a function are local scope. Variables declared outside of a function are MODULE scope unless prefixed by the GLOBAL keyword (MODULE keyword is optional).

Local variables are only associated with a single instance of the function and are destroyed when the function ends or returns.

Module variables are accessible by any function in the same Cicode file running in the same runtime process.

Global variables are accessible in any Cicode function within the same runtime process (but not by Cicode commands or expressions).

Functions are public scope by default, unless prefixed by `PRIVATE`.

Private functions can only be called by another function in the same Cicode file. TaskNew(), TaskCall(), and Cicode Forms can call private functions (v7.2SP5), but TagSubscribe() cannot.

Statement Termination

Each statement that does not end in a keyword (colored blue in the Cicode Editor) should end with a semicolon.

Multiple statements may be included on one line if separated by semicolons.

A statement may be split across multiple lines with a semicolon at the very end.

Variable Tags

Variable tags and local variable tags can be read and written like Cicode variables, but do not need to be declared. Their values are subscribed with an update rate of 250ms while executing the code. See [Code]TimeData.

Variable tag writes are sent to the I/O server asynchronously.

Use TagRead(), TagReadEx(), and TagWrite(), to read/write tags without hard-coding their names.

Each Cicode file, graphic page, or subsystem has its own tag value cache. See [Code]WriteLocal.

Tag writes that are outside of the engineering scale of the tag will fail, terminate the task, and cause a hardware alarm. See [Code]ScaleCheck.