

NSC

Riferimento al linguaggio di programmazione per la macchina virtuale di sistema

rev. Novembre 2006, Marzo 2007, Maggio 2007, Giugno 2007, Novembre 2007, Aprile 2008,
Giugno 2008, Ottobre 2008, Novembre 2008, Marzo 2009

Indice

Introduzione.....	8
Panoramica del sistema operativo.....	9
Pilotaggio ad eventi.....	9
Multitasking.....	10
Comprendere e gestire gli eventi.....	10
Ordine di esecuzione.....	11
Funzioni e rientranza.....	12
Protezioni di sistema.....	12
protezioni.....	13
Inizializzazione di un programma.....	14
Dimensione dei programmi.....	14
Limiti.....	15
Memoria.....	15
Funzioni ed eventi.....	15
Espressioni.....	15
Variabili ed argomenti.....	16
Superare i limiti.....	16
Convenzioni.....	18
Nomi.....	18
Commenti.....	18
Disposizione delle istruzioni.....	19
Struttura di un progetto.....	21
Namespace e visibilità.....	21
Spazio globale.....	22
Spazio globale e di inizializzazione.....	22
.....	22
Direttive.....	24
Inclusioni.....	24
Definizione.....	24
Pragma.....	25
Tipi di dato.....	27
Variabili.....	27
Variabili speciali (variabili oggetto).....	29
Arrays (matrici).....	29
Costanti numeriche.....	29
Operatori.....	31
Calcoli e priorità degli operatori.....	32
Priorità degli operatori.....	33
Funzioni, eventi, oggetti e classi.....	35
Oggetti.....	35
Classi.....	35

Eventi.....	35
Eventi pubblici.....	35
Funzioni.....	35
Parole chiave.....	36
Istruzioni.....	39
API Application programming interface specifico degli NSC.....	43
Funzioni.....	45
setTimer.....	45
getTimer.....	46
setDatetime.....	46
setweekday.....	47
setOut.....	47
set.....	48
dimmer.....	48
getIn.....	49
getOut.....	49
send.....	50
setAnalogTrigger.....	50
encoder.....	51
err.....	51
try.....	51
catch.....	52
OUT.....	52
wait.....	52
setbusy.....	53
resetbusy.....	53
setsync.....	53
sendOK.....	54
read.....	54
write.....	54
readSched.....	54
writeSched.....	55
insertSched.....	55
deleteSched.....	55
freezeScheduler.....	56
ResetAll.....	56
print.....	57
printchar.....	61
movecur.....	61
cls.....	61
clearline.....	62
showicon.....	62
glowlamp.....	62
rnd.....	63
Eventi.....	65
input.....	65

click.....	65
longclick.....	66
net.....	66
timer0 ...7.....	66
timer0 ...7.....	66
analog0 ...1.....	66
outchange.....	67
button.....	67
ZeroEncoder.....	67
KeyDetected.....	67
scheduler.....	68
Esempio schedulatore.....	69
powerdown.....	70
powerup.....	70
TChange.....	71
Costanti predefinite.....	73
Codici tastiera.....	78
Riconoscimenti.....	80

.....

§

Introduzione

Introduzione

Il presente documento fornisce informazioni di riferimento sul linguaggio per la programmazione dei dispositivi dotati di sistema operativo NSC con macchina virtuale incorporata. Il riferimento è al compilatore C simple usato per generare il codice macchina virtuale idoneo ad essere eseguito su tale piattaforma.

Osservazioni in merito al C semplificato.

In deroga allo standard ANSI, il linguaggio adotta una versione semplificata del C per la quale solo un sottoinsieme di funzionalità sono rese disponibili, mentre altre sono predefinite dal compilatore, quali l'inizializzazione automatica delle variabili a zero. Inoltre alcune funzionalità sono state aggiunte in estensione, per consentire di rendere il linguaggio idoneo e più facile per la programmazione dei dispositivi di automazione.

Panoramica del sistema operativo

Il sistema operativo NSC dispone di una macchina virtuale in grado di eseguire i programmi utente che caratterizzano il funzionamento del dispositivo sul quale opera; in questa panoramica verrà esaminato solo questo aspetto, trascurando gli altri sottosistemi che nel loro insieme costituiscono il sistema operativo.

Pilotaggio ad eventi

I programmi si caratterizzano per essere pilotati da eventi, la struttura fondamentale del programma infatti non prevede componenti le quali operino in continuazione, ma procedure elementari che vanno eseguite e poste a compimento in funzione del verificarsi di un evento per il quale si qualificano.

Gli eventi sono definiti da sistema e costituiscono la API (Application Programming Interface) con la quale il programmatore si confronta, e ciascun dispositivo espone propri eventi caratteristici.

Il programmatore deve essere edotto sul fatto che non dovrebbe prevedere esecuzioni di cicli infiniti all'interno di una qualsiasi procedura, in quanto essi contrastano con lo spirito della programmazione pilotata da eventi.

Gli eventi vengono registrati in autonomia e le procedure ad essi abbinate invocate autonomamente in modo asincrono, senza alcun intervento da parte del programmatore.

Multitasking

Anche se il sistema NSC contempla la possibilità di operare in multitasking, non tutti i dispositivi sono in grado di supportarlo. In questi ultimi casi essi operano in una modalità di multitask limitato.

Nelle versioni ove il multitasking è pienamente supportato nessun evento viene perduto né le procedure interrotte, viene inoltre stabilito un valore di priorità real-time per task critici, gestita in autonomia in modo automatico dal sistema mediante frammentazione e rinegoziazione del time-slice.

Nelle versioni con multitask limitato, gli eventi vengono eseguiti in sequenza con l'ordine in cui essi si sono verificati ed in caso di contemporaneità con l'ordine di organizzazione interna della catena di eventi. Anche se il task è impegnato in un processo altri eventi possono comunque essere registrati ed in seguito eseguiti; tuttavia per un certo evento in corso di esecuzione non è possibile accettarne un secondo fino a quando il primo non sia stato soddisfatto, ovvero concluso il suo processo.

Da questa caratteristica deriva la definizione di multitask limitato. Alcuni task però sono comunque considerati ed eseguiti, essi sono i task legati ad eventi prioritari.

Essi sono il tick time (il cronometro interno con cui vengono aggiornati i timer), l'orologio di sistema (se presente), i sincronizzatori di segnali ove presenti, la comunicazione (rete o bus).

Comprendere e gestire gli eventi in ambiente multitasking limitato

Se per esempio un messaggio di rete viene ricevuto ed il corrispondente evento viene generato e, mentre il processo di questo evento risulta ancora in corso, giunge un successivo messaggio, questo viene gestito immediatamente in modo indipendente dal processo dell'evento; tuttavia l'evento non sarà nuovamente generato in quanto ancora occupato dal processo precedente, e verrà perduto. Però se durante l'esecuzione del processo relativo all'evento, giunge un diverso evento questo viene registrato ed indipendentemente preparata la sua esecuzione, che però avverrà appena il precedente processo sarà concluso.

Per riassumere. Gli eventi vengono sempre registrati e preparati all'esecuzione. Fanno eccezione gli eventi successivi di eventi già registrati e pronti all'esecuzione o in corso di esecuzione, tali eventi vengono perduti.

Alcuni eventi (prioritari) sono comunque gestiti ed eseguiti, eventualmente interrompendo il processo in corso di un altro evento con minore priorità.

Normalmente, per le tipiche applicazioni cui viene destinato il dispositivo, la conclusione di un processo di evento è sufficientemente breve da non pregiudicare l'elaborazione di eventi successivi. Verificare sulla scheda tecnica del prodotto i tempi di esecuzione. Alcuni casi di processi che prevedono l'interazione con unità remote possono vedere amplificati i tempi di esecuzione.

È importante che il programmatore comprenda da quanto esposto sopra che l'adozione di processi ciclici infiniti sono in contraddizione con il paradigma della programmazione pilotata da eventi e caratteristico del sistema.

Al limite anche processi condizionati da tempi esterni al sistema possono essere compromettenti nel regolare funzionamento del programma utente.

In ogni caso, a beneficio dell'affidabilità, il sistema non subisce alcuna influenza da tali comportamenti poco rispettosi.

Non fare affidamento all'ordine di esecuzione

Date le caratteristiche descritte, sia per le versioni a multitask limitato sia per quelle a multitask pieno, è importante non fare affidamento all'ordine di esecuzione degli eventi per determinare il valore di variabili globali, o essere dipendenti da tali valori.

L'esecuzione di un evento infatti può avvenire in un momento e quella di un altro in un diverso momento, i due momenti possono essere in un ordine o nell'ordine inverso, questo non è garantito.

Se per esempio si usa una variabile globale per registrare un valore che viene alterato da due diversi eventi, non è possibile fare affidamento sul fatto che tale valore indichi una certa sequenzialità di causa-effetto tra diversi eventi.

Resta valido l'impiego di una variabile globale per stabilire se un certo evento si è verificato o meno, ma ancora non è possibile stabilire quando tale evento si sia verificato e quante volte, se non essendo sicuri che l'evento di controllo sia di diversi ordini di grandezza temporale maggiore di quello con cui l'evento da controllare si verifica.

Funzioni e rientranza

Il linguaggio permette la creazione di funzioni che possono essere invocate in ogni punto del programma. Non vi sono limitazioni sul fatto che una funzione possa richiamare se stessa, utile ad esempio per elaborazioni ricorsive.

Questo comporta però il rischio di causare una ricorsione infinita. Se infatti la funzione non è ben progettata e non dispone di "vie di fuga" essa in poche frazioni di microsecondo causa l'occupazione di tutta la memoria disponibile, producendo una violazione di sistema.

Occorre dunque prestare attenzione allo sviluppo di funzioni ricorsive.

Protezioni di sistema

La macchina virtuale di sistema dispone di una protezione che impedisce ad istruzioni errate o a percorsi di esecuzione imprevisti di causare danni al sistema.

Quando un programma, per un errore di sviluppo o per un evento fortuito che abbia prodotto un percorso di esecuzione imprevisto, tenta di eseguire operazioni non valide la protezione interviene, arrestando il programma ed attivando un registro di errore.

Tale registro può essere interrogato per conoscere lo stato del dispositivo ed informazioni sul punto del programma in cui si verificò l'intervento della protezione.

La protezione attiva anche una procedura di sicurezza che disattiva le uscite rilasciandole, e tipicamente attiva un segnale (visivo o sonoro) per indicare lo stato del dispositivo.

La protezione può essere cancellata, ma il riavvio dei processi per ragioni di sicurezza ed affidabilità riparte sempre da un punto noto di startup, che per semplicità nei dispositivi in genere corrisponde al riavvio del sistema.

Le protezioni prevedono:

Protezioni bloccanti	Quando interviene una di queste protezioni il sistema blocca i processi ed attiva le sicurezze.
stack overflow, stack underflow	Traboccamento della pila. Si verifica se le operazioni richieste superano le capacità di immagazzinamento degli operandi nella pila (stack), si verifica nel caso di funzioni ricorsive sfuggite al controllo, o ad eccessivo carico di eventi per unità di tempo (i processi legati agli eventi non vengono evasi in tempo utile per soddisfare gli eventi pendenti, che quindi si accumulano), numero di operazioni matematiche eccessivo (per esempio se il compilatore non è stato in grado di individuarne un uso illegale).
Unrecognized	Codice sconosciuto nel bytecode del programma utente.
Out of memory	Memoria insufficiente.
Out of function stack	Pila funzioni insufficiente. Si può verificare con funzioni ricorsive o un numero eccessivo di annidamenti di funzioni. Sui dispositivi più piccoli il numero massimo è di 5 annidamenti.
Reprogramming error	Errore di riprogrammazione.
Flash failure	Errore di riprogrammazione, flash esaurita.
Access violation	Violazione di accesso. Un processo ha tentato di accedere ad aree di memoria a lui negate, si verifica se si usano indici scorretti.
Invalid program	Programma utente non valido.
Protezioni bloccanti di sistema	Quando interviene una di queste protezioni il sistema blocca i processi ed attiva le sicurezze.
System hardware stack overflow	Il sistema ha traboccato la pila hardware.
System stack overflow	Il sistema ha traboccato la pila software.
Protezioni non bloccanti condizionate	Quando intervengono non solo bloccanti solo se si attiva la gestione errori mediante l'istruzione try , nel cui caso non vengono arrestati i processi ma viene registrato l'errore che può essere notificato e gestito dai processi stessi; se non si attiva la gestione errori sono bloccanti e causano l'arresto dei processi e l'attivazione delle sicurezze
Division by zero	Divisione per zero.
Subscript out of range	Indice della matrice fuori della matrice.
Protezioni non bloccanti	Quando intervengono queste protezioni i processi non vengono arrestati ma viene registrato l'errore che può essere notificato e gestito dai processi stessi.
RPC failure	Richiesta esecuzione evento pubblico non locale fallita (remote procedure call).
Condizioni speciali	
Stop	Unità in stop. I processi sono arrestati e sono attivate le misure di sicurezza, ed autonotifica.
Virgin	Unità mai programmata. Simile a Stop.
Virgin unserialized	L'unità non è stata correttamente serializzata in fabbrica, contattare il servizio tecnico.

Inizializzazione di un programma

Un programma è costituito da eventi e funzioni (vedere il capitolo "Struttura di un progetto"), e nella sua parte iniziale, prima che inizi l'implementazione di qualsiasi funzione o evento, lo spazio globale.

Lo spazio globale, chiamato anche di inizializzazione, è una regione di programma che viene eseguita immediatamente non appena il sistema si avvia.

Il suo scopo è di inizializzare le variabili globali ed il dispositivo. Durante questa fase possono essere usate tutte le funzioni dichiarate localmente (all'oggetto ovvero al dispositivo) e gli eventi pubblici di altri oggetti, ma in questa fase nessun evento verrà processato.

Se per esempio si inserisce una istruzione che operi un ciclo infinito in questo punto, gli eventi non saranno mai gestiti (fanno eccezione quelli prioritari gestiti dal sistema).

Terminata questa fase vengono sbloccati gli eventi e da quel momento il programma verrà eseguito come descritto nei paragrafi precedenti.

Dimensione dei programmi

Il bytecode generato dal compilatore utilizza una tecnica di compressione per cui lo spazio occupato da un programma è nettamente inferiore a quanto sarebbe se fosse compilato con istruzioni macchina standard. Non vi è alcuna relazione diretta tra la dimensione del testo del programma sorgente e il programma compilato.

È bene comunque sapere che l'uso di costanti numeriche occupa più spazio di una variabile, e che alcune istruzioni sono sconvenienti se usate in modo inappropriato.

In particolare l'istruzione *switch* non è conveniente se usa meno di due casi ed al suo posto è preferibile usare due istruzioni *if*, in ogni caso il compilatore, dove possibile, fornisce degli avvisi nel caso rilevi un uso improprio o poco conveniente delle istruzioni.

Limiti

I limiti dimensionali sono caratterizzati dallo spazio di memoria disponibile, dal numero delle funzioni usate (incluso tra queste anche gli eventi usati), dalla dimensione delle singole espressioni e dalla presenza o meno di stringhe costanti, dal numero e dal tipo di argomenti che possono essere passati alle funzioni e agli eventi, dal numero e dal tipo di variabili che possono essere dichiarate.

Memoria. A seconda del dispositivo in cui fisicamente opera NSC si disporrà di una diversa quantità di memoria. Questo condiziona sia l'estensione del programma sia il numero di possibili chiamate a funzione annidate (o il numero di ricorsioni, nel caso di funzioni ricorsive). Tipicamente nei dispositivi modulari questo spazio consente di memorizzare un programma sufficientemente ampio da soddisfare le finalità del dispositivo stesso e sono possibili fino a 5 livelli di annidamento di chiamata a funzione.

Esempio:

```
int i = f1( f2( f3( f4() ) ) ); // il 5° livello si raggiunge con la chiamata a
                               // f4, perchè questa procedura sta già occupando
                               // il primo livello.
```

Funzioni ed eventi. La macchina virtuale è in grado di gestire fino a 254 funzioni, includendo tra queste anche gli eventi e gli eventi pubblici.

Espressioni. Ciascuna espressione non può contenere più di un certo numero di operandi ed operatori, il numero però varia con il tipo di operandi e di operatori usati nell'espressione, mediamente possono essere gestiti circa fino ad una ventina di operandi. Inoltre in una espressione non possono essere annidati (in modo non semplificabile) più di 10 operandi, il compilatore automaticamente individua i membri dell'espressione che sono semplificabili e li riduce con l'equivalente valore costante o con l'equivalente variabile o un'espressione alternativa più efficiente; in ogni caso è garantito che la riduzione non alteri l'algoritmo voluto nell'espressione.

Notare che l'uso di parentesi può modificare e forzare o meno la valutazione del grado di priorità e quindi il livello di annidamento dei membri dell'espressione.

Il seguente esempio riporta una espressione che raggiunge (ma non supera) il limite citato:

```
a=a+(b+c*(g+h*(n+g*(n+g))));
```

Ogni funzione (o evento) non può contenere oltre un certo numero di espressioni, il numero esatto non è definibile perchè ciascuna espressione può cambiare di dimensione in base ai suoi operandi ed operatori, dalla presenza di chiamate a funzione, ecc.

Variabili ed argomenti. Il numero di variabili che possono essere dichiarate dipende dal tipo di dispositivo. Nei dispositivi modulari tipicamente possono essere dichiarate fino a 32 variabili globali di tipo byte o 16 di tipo int (in numero quindi varia in base ai tipi usati fino al massimo di 32).

In ciascuna funzione ed evento possono essere dichiarate fino a 32 variabili locali di tipo byte, includendo in queste gli argomenti che la funzione o l'evento riceve.

Le matrici non possono superare i limiti del numero di variabili elementari (byte) disponibili, per esempio una matrice di 2 int occupa lo spazio di 4 variabili byte.

In tutti i casi il compilatore segnala con un errore il caso in cui una espressione, una funzione o l'intero programma superi i limiti stabiliti o se si supera il limite dello spazio per le variabili.

Superare i limiti

Se accade che un'espressione sia troppo lunga, spezzare l'espressione in due o più espressioni più semplici. Esempio:

```
/*  
  l'espressione:   a=a+(b+c*(g+h*(n+g*(n+g*(K+Y))));  
  può essere semplificata in:  
*/  
  
int r = n+g*(n+g*(K+Y));  
  
a=a+(b+c*(g+h*r));  
  
//oppure:  
  
function f_ng(int n, int g, int Z)  
{  
  return n+g*(n+g*Z);  
}  
  
a=a+(b+c*(g+h* f_ng(n,g,K+Y) ));
```

Se accade di superare lo spazio disponibile per una funzione provare a semplificarla spezzando le parti di programma che possiedono un significato proprio e metterle in una seconda funzione, chiamata dalla prima.

In particolare se una operazione viene eseguita allo stesso modo (magari con modalità diverse specificabili con un parametro) in diversi punti del programma, prevedere di isolare quella parte di istruzioni ed inserirle in una funzione che sarà chiamata nei vari punti in cui occorre le istruzioni originali. Questo tra l'altro riduce il lavoro di sviluppo e lo rende più efficiente facilitando eventuali manutenzioni o modifiche.

§

Convenzioni

Convenzioni

Nomi

I nomi degli identificatori possono essere lunghi fino a 40 caratteri, devono essere integri (nessuno spazio o ritorno a capo all'interno del nome), devono iniziare per una lettera dell'alfabeto tra "A" e "Z" e tra "a" e "z" (non sono ammesse accentate, umlaut, ecc.) oppure per il carattere di sottolineatura "_", può in seguito includere numeri.

Esempi di nomi validi:

```
alfa, Beta, pompal, Ev132H, _collettore, sala_attesa
```

Esempi di nomi non validi:

```
1, 34alfa, metrò, &collettore, sala attesa,  
Ev1234567890abcdefghijklmnopqrstuvwxyz1234567890abcdefghijklmnopqrstuvwxyz
```

Commenti

È possibile inserire commenti nel codice. Un commento può iniziare per "//" e termina con la riga (il primo ritorno a capo, non automatico) oppure iniziare per "/*" e termina con "*/", in questo caso possono essere inseriti ritorni a capo.

Esempio:

```
// commento su una sola riga  
...istruzioni...  
/*  
    commento  
    su più  
    righe  
*/
```

Disposizione delle istruzioni

Le istruzioni e le espressioni possono essere disposte come meglio si crede.

Esempio:

```
if(  
    a == b      /* controllo a con b */  
    &&  
    c == 1     /* controllo c con 1 */  
)  
{  
    eseguiQuesto(a,b,c);  
}
```

§

Struttura

Struttura di un progetto

Un progetto di programma si compone di una serie di blocchi di dichiarazioni di entità le quali contengono una o più istruzioni. Ogni blocco viene racchiuso tra parentesi graffe. La struttura è composta da classi, oggetti e direttive di preprocesso.

Classi ed oggetti incorporano il codice che a sua volta è formato da funzioni, eventi ed eventi pubblici. Ciascuno di questi blocchi include a sua volta le istruzioni e le variabili.

Gli eventi sono funzioni invocate al verificarsi di un evento predefinito dal sistema e dipendente dal dispositivo (dipende da cosa offre e da cosa dispone il dispositivo).

Le funzioni sono blocchi di istruzioni definite dall'utente programmatore.

Gli eventi pubblici sono funzioni definite dall'utente programmatore che non possono ritornare valori ma possono essere invocate da qualsiasi oggetto del progetto.

Le direttive di preprocesso sono istruzioni generali di preparazione del programma, tra esse figurano le definizioni che sono coppie nomeidentificatore-valore che vengono sostituite nel codice in fase iniziale di preprocesso della compilazione.

Namespace e visibilità

All'interno di ciascun blocco sopra specificato risultano visibili le variabili e le funzioni dichiarate in essi. Una funzione dichiarata in un oggetto non è visibile in altri oggetti ed il suo nome può anche essere riutilizzato senza conflitti. Ugualmente le variabili dichiarate all'interno di un blocco funzione o evento non saranno visibili all'esterno di esso.

Spazio globale e di inizializzazione

In ciascun oggetto (e classe) la parte iniziale di codice, che preceda qualsiasi funzione o evento o evento pubblico, è definita lo spazio globale dell'oggetto.

Il codice in questo punto sarà eseguito immediatamente (per le classi lo sarà solo nell'oggetto in cui la classe risulta implementata; le classi sono codice astratto).

Durante questa fase gli eventi non sono attivi ed il codice viene garantito che non sarà interrotto da alcun evento fino alla sua conclusione.

Le variabili dichiarate in questa area sono inoltre visibili in tutto l'oggetto e sono quindi definite globali.

Il seguente schema delinea questo spazio:

```
object nomeoggetto
{
  AREA GLOBALE e di INIZIALIZZAZIONE

  function nomefunzione()
  {
    AREA LOCALE AL BLOCCO (FUNZIONE)
  }
}
```

§

Direttive

Direttive

Le direttive consentono di impartire istruzioni al compilatore per preparare il programma in modo opportuno alla compilazione. Tra esse figurano le inclusioni, le definizioni e le istruzioni pragma.

Inclusioni

Sintassi:

```
#include "nomefile"
```

nomefile deve essere il nome, completo di percorso, di un file il cui testo sarà incluso nel punto in cui si trova la direttiva e diventerà parte integrante del codice sorgente del programma (nella fase di compilazione).

Nomefile (e le virgolette) possono essere sostituiti dal nome di un componente generato dinamicamente, nella forma:

```
#include <*codeobject:aa> // do not change or remove this line
```

Questa linea non deve essere modificata o rimossa (diversamente i componenti non saranno incorporati).

Definizione

Sintassi:

```
#define identificatore valore
```

Definisce un valore ad un identificatore, utile per specificare valori da configurare in vari punti del programma. Valore verrà sostituito con tutti i casi in cui identificatore viene trovato nel programma. Una definizione successiva può fare uso, come valore, di un identificatore definito precedentemente.

Il compilatore accetta fino a 4351 definizioni.

Pragma

Sintassi:

```
#pragma <specifica_pragma>
```

Inserisce una informazione speciale nel programma ed una istruzione speciale al compilatore. Specifica pragma dipende dal tipo di informazione.

Sono definiti i seguenti pragma:

resources specifica di includere delle risorse bitmap nel programma

Le risorse bitmap consentono di includere delle icone nel programma che possono essere richiamate e stampate mediante la funzione **showicon** (vedi API).

La direttiva è valida solo per dispositivi dotati di terminale video, display o stampante.

Sintassi:

```
#pragma resources( identificatoreicona [, identificatoreicona ] )
```

identificatoreicona è il nome dell'identificatore dell'icona da inserire, possono seguire diversi identificatori separati da virgola per includere diverse icone. Le icone devono essere realizzate con l'apposito programma o con lo strumento offerto da MaticStudio, e gli identificatori devono corrispondere ai nomi assegnati a ciascuna icona.

Esempio:

```
#pragma resources( stop , allarme , auto, manuale )
```

§

Tipi di dato

Tipi di dato

Variabili

Le variabili devono essere dichiarate prima di essere usate. In deroga al C ANSI non è obbligatorio dichiarare le variabili all'inizio del codice, inoltre è possibile assegnare subito un valore, anche se ritornato da una funzione.

<i>tipo</i>	<i>note</i>	<i>caratteristiche</i>
char byte var		sinonimi: char , byte , var variabile minima, dimensione 1 byte, senza segno
int		variabile intera, valori da 0 a +32767 o da 0 a -32768
float	1	variabile reale, valori da 0.0000 a +99999 o da 0.0000 a -99999
string	2	riferimento a stringa costante di caratteri Possono essere assegnate stringhe con massimo 254 caratteri (i caratteri speciali e macro vengono contati come un solo carattere ciascuno).

Note: 1) disponibile solo su dispositivi abilitati all'uso di variabili float. 2) disponibile solo su dispositivi abilitati all'uso di costanti stringa.

Sintassi di dichiarazione di una variabile o di un riferimento a costante.

```
tipo identificatore [ = costante ] / [ = funzione() ] ;
```

dove tipo deve essere uno dei tipi sopra indicati (byte, int, ecc.)

Note.

La variabile *float* è disponibile solo su particolari versioni, non sui dispositivi standard.

Il tipo *void* non è previsto e non viene richiesto (è implicito).

Il tipo *string* è disponibile solo su dispositivi con display o schermo o stampante, è ammesso il solo riferimento a stringhe costanti (una volta dichiarata non è modificabile), l'assegnazione del valore (la stringa di caratteri) va fatta subito dopo la dichiarazione dell'identificatore del riferimento.

Esempio:

```
string testo="assegnazione della stringa costante";
```

Le stringhe possono essere composte con la funzione `print` (vedere le API), e le costanti stringa possono essere indicizzate. Esempio:

```
string aperto="aperto", chiuso="chiuso",rubinetto="rubinetto ";  
  
function stampaStato(byte statorubinetto)  
{  
    print(rubinetto,aperto+statorubinetto);  
}
```

Nell'esempio vengono dichiarate tre costanti stringa, rispettivamente di nome *aperto*, *chiuso* e *rubinetto*, a cui vengono assegnati corrispondenti valori costanti.

L'uso della funzione `print` compone la parola "rubinetto " (notare lo spazio) con la parola "aperto" se il valore di *statorubinetto* è zero, o "chiuso" se il valore di *statorubinetto* è 1. Come si vede la costante *aperto* viene usata come base per indicizzare le costanti successive. Se per esempio si richiamasse la funzione *stampaStato* passando come argomento di *statorubinetto* il valore 2 verrebbe stampato il testo "rubinetto rubinetto " poiché il secondo membro partendo dalla base *aperto* indicizzerebbe fino alla costante *rubinetto* (la terza in ordine).

Variabili speciali (variabili oggetto)

<i>tipo</i>	<i>caratteristiche</i>
outbuf	<p>class-union dati di ingresso/uscita logica</p> <p>Uso con notazione puntata seguita da costante numerica da 0 a 15.</p> <p>Sintassi:</p> <pre>identificatore.costante</pre> <p>Esempio:</p> <pre>outbuf u; u.0 = ON;</pre>

Arrays (matrici)

È possibile dichiarare matrici fino alla massima dimensione concessa. La matrice va dichiarata inserendo la dimensione tra parentesi quadre.

Esempio:

```
byte a[5]; // dichiara una variabile byte di 5 elementi
```

La dimensione massima per i dispositivi NSC è tipicamente di 12 byte o 6 int.

Costanti numeriche

0-9	numeri da zero a nove
.	punto decimale, esempio: 1.29
0x	prefisso per valori esadecimali
0o	prefisso per valori ottali

Speciali per NSC:

0b	prefisso per valori binari
----	----------------------------

Nota: la base binaria è in estensione al C ANSI

§

Operatori e calcoli

Operatori

+	somma	
-	sottrazione	
*	moltiplicazione	
/	divisione	
%	modulo (resto della divisione intera)	
=	assegnazione	
==	comparazione uguaglianza	
!=	comparazione diversità	
	OR logico	
&&	AND logico	
<	minore	
<=	minore uguale	
>	maggiore	
>=	maggiore uguale	
^	XOR (binario) [▪]	
	OR binario [▪]	<i>NON DISPONIBILE nei modelli -S</i>
&	AND binario [▪]	<i>NON DISPONIBILE nei modelli -S</i>
<<	spostamento a sinistra [▪]	<i>NON DISPONIBILE nei modelli -S</i>
>>	spostamento a destra [▪]	<i>NON DISPONIBILE nei modelli -S</i>
!	NOT logico (complemento a 2)	
~	complemento a 1 [▪] (inversione binaria) (usare i tasti ALT+0126)	
()	parentesi di precedenza	
[]	indice array, esempio: a[2]	
{ }	blocchi istruzioni (usare i tasti MAIUSC+ALTGR+ “[” oppure “[”)	
;	terminazione istruzione, istruzione nulla, terminazione prototipo	
,	separatore di lista argomenti o variabili	

Speciali per NSC:

.0 ...15	selettore bit (richiede costante da 0 a 15 o nome definito a costante) utilizza la notazione puntata.
.	operatore a oggetto, esempio: <code>MioOggetto.pubevOggetto();</code>
:	implementa classe o oggetto, esempio: <code>object MioOggetto : MiaClasse</code>
<-	emulatore ladder serie, esempio: <code>u.1 = IN.2 <- IN.3;</code>
	emulatore ladder parallelo, esempio: <code>u.1 = IN2 IN.3;</code>

[▪] usa gli operandi come interi senza segno (unsigned int)

Calcoli e priorità degli operatori

In run time calcoli producono dei risultati che sono dipendenti dal tipo di variabili coinvolte, non esiste un limite o un errore di sovrafflusso [overflow] di una variabile, ovvero di superamento del contenimento, salvo il caso di assegnazione di un valore tramite una costante o una variabile di maggiore capacità già in fase di compilazione.

Le variabili float sono autolimitate al massimo della capacità ma non causano errore se si tenta di superarne il limite.

La seguente espressione, raggiunto il valore di 255 non produrrà alcun errore e la variabile `test` riprende da zero il conto:

```
byte test;

// ciclo infinito
for(;;) {
    test = test +1; // test assume i valori tra 0 e 255,
                  // al 256esimo passo si avvolge e ritorna a zero.
}
```

ovvero il valore si avvolge attorno al limite stabilito dal tipo di variabile.

Occorre notare che le variabili intere con segno (int) passano al segno opposto.

Nell'esempio che segue, `test` assume il valore -32768 il passo successivo al 32767esimo:

```
int test;

// ciclo infinito
for(;;) {
    test = test +1; // test assume il valore -32768 al 32768esimo passo
}
```

da quel punto inizia a decrescere (-32767, -32766...) fino a raggiungere zero, poi ricomincia a crescere fino a 32767, e così via.

Se si desidera impostare un blocco esplicito è necessario prevedere delle istruzioni apposite, per esempio nel seguente esempio si limita il conto a numeri positivi, avvolgendo il valore su 32767:

```
int test;

// ciclo infinito
for(;;) {
    test = test +1; // test assume il valore tra 0 e 32767
    if(test & 0x8000u) test=0; // se il MSb è 1 allora il numero è negativo
}
```

Notare che per le variabili `int` il trasferimento del valore ad un diverso sistema potrebbe consentire il conteggio di valori compresi tra 0 e 65535. Esempio:

```
send(server, test, 0, CTRLTYPEUSER); // server può gestire il valore come intero
senza segno
```

Priorità degli operatori.

Gli operatori hanno una loro priorità che ne determina la precedenza nei calcoli. Di seguito viene riportato l'elenco della priorità, a partire da quella maggiore alla minore. Gli operatori aventi pari priorità sono eseguiti a partire da sinistra verso destra.

<i>priorità</i>	<i>operatori</i>
1	- (negativo) = ++ -- ! ~ ^ (potenza 2) / (radice 2) chiamate a funzione
2	*
3	/
4	%
5	+ -
6	< > <= >=
7	== !=
8	& << >> ^
9	&&

Per modificare la priorità si possono usare le parentesi tonde, seguendo la formalità tipica della matematica. Esempio:

```
x = (1 + 5) * 2;  
y = 1 + 5 * 2;
```

x riceverà il valore 12, y riceverà il valore 11.

§

Strutturatori

Funzioni, eventi, oggetti e classi

Oggetti

Object rappresenta una classe implementata, ciascun dispositivo in un progetto costituisce una classe implementata, ovvero un oggetto, e va quindi inserito con la parola chiave object.

Classi

Le class sono classi astratte di codice, esse possono essere implementate da uno o più oggetti mediante l'operatore ":". Una classe può ereditare un'altra classe, usando l'operatore ":".

Eventi

Gli eventi sono funzioni predefinite dal sistema che vengono chiamate al verificarsi dell'evento. Gli eventi non dichiarati non vengono considerati e non esistono nel codice oggetto generato.

Eventi pubblici

Gli eventi pubblici (pubevent) sono funzioni senza ritorno di valore che possono essere invocate da qualsiasi punto del progetto. Un evento pubblico viene esposto dall'oggetto che lo implementa e può essere chiamato usando l'operatore "." per identificare nomeoggetto.nomeeventopubblico()

Gli eventi pubblici richiedono un numero preciso di argomenti, come mostrato dallo schema riportato di seguito. L'argomento "byte caller" è obbligatorio e deve essere sempre presente. La chiamata ad un evento pubblico localmente non deve necessariamente passare questo argomento e può sostituirlo con i tre punti.

Funzioni

Le funzioni possono definire un numero di argomenti arbitrario non superiore a 12 e comunque non superiore a 12 byte (per i dispositivi NSC). Le funzioni possono ritornare valori. In deroga al C ANSI non è richiesto di specificare il tipo void per le funzioni che non passano argomenti (o che non ritornano valori), esso è implicito. Inoltre le funzioni possono o meno ritornare un valore, anche se dichiarato.

I prototipi delle funzioni vanno dichiarati prima della loro invocazione, eccetto il caso in cui l'invocazione non avvenga dopo che la funzione risulti (nel codice sorgente) già dichiarata sopra.

Parole chiave

Le seguenti parole chiave identificano il blocco di codice che contengono, ciascuna di esse richiede un identificatore per invocare l'entità che rappresentano.

object classe implementata

sintassi:

```
object nomeoggetto [ : nomeclasse ]
{ ...codice oggetto... }
```

class classe astratta

sintassi:

```
class nomeclasse [ : nomeclasse ] { ...codice classe... }
```

function funzione

sintassi:

```
function nomefunzione ( argomento [, argomento,...][...] )
{ ...codice funzione... }
```

Una funzione deve essere dichiarata senza riportare il blocco istruzioni chiudendo con un punto e virgola, sintassi:

```
function nomefunzione ( argomento [, argomento,...][...] );
```

event evento

sintassi:

```
event nomeevento ( argomentievento ) { ...istruzioni... }
```

pubevent evento pubblico

sintassi:

```
pubevent nomeevento ( argomenti* ) { ...istruzioni... }
```

*nota argomenti: possono essere dichiarati i seguenti argomenti:

opzione	argomenti obbligatori (e ammessi)
no argomenti	byte caller
1 argomento	int arg1, byte caller
2 argomenti	int arg1, int arg2, byte caller
3 argomenti	byte arg1, byte arg2, int arg3, byte caller
4 argomenti	byte arg1, byte arg2, byte arg3, byte arg4, byte caller

Esempi:

```
// dichiara una classe
class general
{
    function sum(int a, int b)
    {
        return a+b;
    }
}

// dichiara un oggetto, lo implementa, e implementa la classe general ereditandola
object alfa : general
{
    // dichiarazione di prototipo (solo se si usa prima della implementazione)
    pubevent accedi(int stato, byte caller);

    // dichiara evento (intrinseco di sistema)
    event click(int Clicked)
    {
        int total = sum(Clicked, 2);
        accendi(total, ...); // chiama evento pubblico, localmente
    }

    // dichiara evento pubblico
    pubevent accendi(int stato, byte caller)
    {
        setOut(stato, 0xffffu, this);
    }
}

// dichiara un oggetto e lo implementa
object beta
{
    alfa.accendi(0b1111001); // chiama evento pubblico dal global space
}
```

§

Istruzioni

Istruzioni

break interrompe un ciclo for oppure while o esce da un caso **switch**

case seleziona un caso in una struttura **switch**, il valore che segue deve essere una costante.

Sintassi:

```
case const :
```

Esempio:

```
switch (n) {  
  case 1:  
    z=0;  
    break;  
  case 2:  
    z=1;  
    break;  
  default:  
    z=99;  
}
```

(vedi switch e default)

NOTA: non disponibile nelle versioni s

continue salta al prossimo ciclo di un ciclo for o while

Esempio:

```
for(i=0;i<10;i=i+1) {  
  if(i==3) continue;  
  r[i] = i;  
}
```

default caso considerato in mancanza di altri casi
(vedi case e switch)

do ciclo a condizione posticipata, sintassi:

```
do {  
  ... istruzioni ...  
} while (espressione);
```

Nota: può non essere disponibile su tutti i modelli.

else alternativa in una sequenza di condizioni if. Si può concatenare con una successiva condizione if.

Esempio:

```
if(espressione) {  
    ...istruzioni se espressione è vera...  
}  
else if(espressione1) {  
    ...istruzioni se espressione1 è vera...  
}  
else {  
    ...istruzioni se le condizioni sopra non sono vere...  
}
```

for ciclo iterativo.

sintassi 1:

```
for(espressioneinizializza; espressionecondizione;  
espressioneiterazione) istruzione;
```

sintassi 2:

```
for(espressioneinizializza; espressionecondizione;  
espressioneiterazione) {  
    ...istruzioni in blocco...  
}
```

Esempio:

```
char a[10]; // dichiara a di 10 elementi  
  
for(i=0; i<10; i=i+1) {  
    if( a[i] ) break; // scansiona tutti gli elementi di a,  
                    // se un elemento è non zero il ciclo  
                    // termina  
}
```

È possibile non specificare una o tutte le espressioni di inizializzazione, condizione e iterazione, esempio:

```
for(;;) { ...istruzioni eseguite all'infinito... }
```

goto dirotta il flusso del programma alla etichetta indicata.

sintassi:

```
goto etichetta ;  
.  
.  
.  
etichetta:
```

etichetta è un nome a piacere conforme alle nomenclature ammesse.

if istruzione condizionale di diramazione.

sintassi 1:

```
if( espressione ) istruzione ;
```

sintassi 2:

```
if( espressione ) {  
    ...istruzioni...  
}
```

Esegue la istruzione che segue o le istruzioni nel blocco tra graffe se espressione è vera.

return interrompe una funzione ed opzionalmente ritorna un valore,

sintassi 1:

```
return ;
```

sintassi 2:

```
return espressione ;
```

switch seleziona tra vari valori diramando il flusso di esecuzione.

sintassi:

```
switch ( espressione ) {  
case valore1:  
    ...istruzioni se valore1 è = a risultato di espressione...  
    break; // (opzionale)  
case valore2:  
    ...istruzioni se valore2 è = a risultato di espressione...  
default:  
    ...istruzioni se nessuno dei casi sopra è vero...  
}
```

Note:

Se non si usa **break** i casi successivi vengono comunque eseguiti

Non disponibile nelle versioni -S

while cicla fino a che l'espressione è vera.

sintassi 1:

```
while ( espressione ) istruzione;
```

sintassi 2:

```
while ( espressione ) {  
    ...istruzioni...  
}
```

Note: Introducendo l'istruzione **break** si interrompe il ciclo, mentre l'istruzione **continue** salta alla fine del blocco istruzioni e continua il ciclo.

§

API
application
programming
interface

API Application programming interface specifico degli NSC

In questa sezione sono presentate le funzioni specifiche del sistema NSC, essa è divisa in due capitoli: le funzioni e gli eventi.

Le funzioni sono quelle offerte intrinsecamente al sistema, e consentono l'accesso a procedure del sistema operativo, quali la gestione di temporizzatori e la comunicazione dati via rete.

Gli eventi sono funzioni che è possibile chiedere che il sistema notifichi qualora il relativo evento si verifichi. Gli eventi impostano un gancio nel sistema che provvede autonomamente a invocare l'evento richiesto quando si manifesta, richiamando quindi la funzione definita, solo gli eventi dichiarati ed implementati sono agganciati. Il sistema comunque non spende tempo di esecuzione per gestire le funzioni evento quando non ci sono i relativi eventi, garantendo così la massima efficienza e velocità di esecuzione.

La dichiarazione di un evento si fa come per una qualsiasi funzione, ma deve rispettare il nome e gli argomenti offerti esattamente come previsto dal prototipo intrinseco (come viene mostrato in questo documento, per ciascun evento disponibile). Solo gli eventi disponibili per un certo dispositivo possono essere usati.

§

API - Funzioni

Funzioni

setTimer

imposta un temporizzatore

sintassi:

```
setTimer( valore, id );
```

parametri:

valore espressione (int) valore del ritardo, espresso in decimi di secondo

id espressione (byte) id del temporizzatore
(vedi costanti TIMER0 a TIMER7)

ritorno: la funzione non ritorna alcun valore.

Note:

Dopo avere impostato il temporizzatore questo attiva un grilletto che scatta quando il tempo scade scatenando l'evento timer0...timer1 (in base all'id imposto)

Impostando un valore zero il temporizzatore viene annullato.

E' possibile impostare una costante prodotta da una stringa interpretata con la forma: \$(time h : m : s : d)

dove h,m,s,d sono rispettivamente ore, minuti, secondi e decimi di secondo.

Esempio:

```
setTimer( $( time 0:25:0:33 ), TIMER0 );
```

Il valore massimo impostabile è di 1h 48' 48".0

getTimer

legge il tempo restante di un temporizzatore, o ritorna un argomento della data e ora di sistema (nei dispositivi con orologio). Sintassi:

```
getTimer( id )
```

parametri:

id espressione (byte) id del temporizzatore
 (vedi costanti TIMER0 a TIMER7)
 oppure i valori costanti GET_MINUTE, GET_HOUR, GET_SECOND,
 GET_DAY_WEEK, GET_MONTH o GET_YEAR per ricavare un
 valore dall'orologio (solo dispositivi dotati di orologio).

ritorno: valore int del tempo restante, zero se il
 temporizzatore è scaduto.

Note: NON DISPONIBILE nelle versioni -S

Nota: Le costanti che ritornano il tempo reale è DISPONIBILE SOLO SUI MODELLI DOTATI DI OROLOGIO; vedi anche setDatetime, setweekday.

Le costanti che ritornano gli argomenti che formano la data-ora possono essere indicizzati, usando GET_SECOND come base, esempio:

```
// stampa dataora
for( ; i < 6; i=i+1) print(getTimer(GET_SECOND + i), 1);
```

l'ordine è Secondi, Minuti, Ora, SettGiorno, Mese, Anno.

Notare che DAY_WEEK ritorna il giorno della settimana nei primi 3 bit alti, 0 corrisponde a Lunedì, ed il giorno del mese nei restanti bit bassi.

setDatetime

imposta la data e l'ora. sintassi:

```
setDatetime(year, month, wday, hour, minute, second);
```

argomenti (tutti di tipo byte): anno, mese, wday: giorno e giorno della settimana, ora, minuti, secondi.

Note: Giorno e giorno della settimana possono essere un valore composto che può essere ottenuto dalla funzione *setweekday*, tuttavia con le versioni pari o superiori alla 1.12 l'eventuale argomento giorno settimana viene ignorato e viene comunque calcolato automaticamente, pertanto è possibile assegnare direttamente solo il giorno del mese.

Avvertenza: La funzione non esegue controlli di congruenza, è possibile impostare il mese al valore 13, anche se con il tempo sarà prodotta una data valida più prossima al valore errato fornito.

Vedi anche: getTimer, setweekday

Disponibile solo sui dispositivi dotati di orologio.

setweekday

dato un giorno ed il relativo giorno della settimana, ritorna il valore composto *wday*. Sintassi:

```
(byte) setweekday(byte weekday,byte day);
```

Argomenti:

weekday giorno della settimana, contando da zero con il Lunedì.

day giorno del mese, valore ammesso da 1 a 31.

Attenzione: La funzione non esegue controlli, e si attende di ricevere valori corretti. Se si imposta il giorno 30 (per Febbraio) in combinazione con la funzione *setDatetime*, al cambio del giorno sarà cambiato nel 1 Marzo.

Vedi anche: *setDatetime*, *getTimer*

Disponibile solo sui dispositivi dotati di orologio.

setOut

imposta le uscite logiche, sintassi:

```
setOut( valore_data, valore_mask, target )
```

parametri:

valore_data espressione (int) valore da impostare, ogni bit corrisponde alla relativa uscita, esempio 0b0001 imposta l'uscita 0 (la prima) mentre 0b0010 imposta l'uscita 1 (la seconda).

valore_mask espressione (int) uscite da impostare, ogni bit corrisponde alla relativa uscita (vedi sopra).

target oggetto di destinazione dell'impostazione, usare *this* per l'oggetto in cui si trova l'istruzione.

ritorno: la funzione non ritorna alcun valore.

Note: Solo le uscite cui corrispondono i bit impostati in *valore_mask* saranno effettivamente riportati nelle uscite, azzerandola se il corrispondente bit in *valore_data* è a zero o eccitandola se il corrispondente bit in *valore_data* è a uno.

Vedi anche: *set*

set

imposta le uscite logiche, sintassi:

```
set( valore, target );
```

parametri:

valore variabile di tipo outbuf contenente il valore da impostare

target oggetto di destinazione dell'impostazione, usare this per l'oggetto in cui si trova l'istruzione.

ritorno: la funzione non ritorna alcun valore.

Note: Vengono impostate solo le uscite che sono effettivamente state impostate nella variabile valore (in modo automatico),

Esempio:

```
outbuf u;  
u.1 = ON;  
u.2 = OFF;  
set( u, this ); /* vengono impostate le uscite OUT1 e  
OUT2, mentre le altre uscite restano immutate. */
```

Vedi anche: *setOut*

dimmer

imposta il valore di uscita di un dimmer. sintassi:

```
dimmer(int value,int dimmerChannel,byte target);
```

argomenti:

value valore di uscita da impostare, può essere compreso tra zero e 20 (o 40 con dimmer hi-res).

dimmerChannel canale dimmer da impostare, questo valore deve essere BASEDIMMERCH+n dove n è il canale, oppure DIMMERCH0, DIMMERCH1, DIMMERCH2, DIMMERCH3 per i canali 0,1,2,3 rispettivamente.

target nome o indirizzo dell'oggetto destinatario dell'azione.

Note: Disponibile solo sui dispositivi che dispongono della funzione dimmer.

getIn

legge gli ingressi, sintassi:

```
(int) getIn( (byte) [ id | object ] );
```

parametri:

- id (o object) id o oggetto da cui leggere gli ingressi logici. Usare id: ANALOG0, ANALOG1 per avere il valore di ingresso analogico; DGTDATA per ottenere il dato digitale dal primo canale; Usare nome oggetto o *this* per l'oggetto stesso di cui avere lo stato degli ingressi logici.
- ritorno: valore int dello stato di ingressi logici o dell'ingresso analogico selezionato.

Nota: Non è possibile leggere un valore analogico o da canale digitale di un diverso oggetto.

Nota: Il canale dati digitale è disponibile solo in alcuni modelli.

Nota: Se non è possibile ottenere il valore richiesto la funzione ritorna zero e viene elevato un errore _ERRRPCFAILED (vedi RPC failure) leggibile con la funzione *catch()* oppure *err()* .

getOut

ritorna lo stato delle uscite. Sintassi:

```
(int) getOut( (byte) object );
```

argomenti:

object nome oggetto di cui si vuole ottenere lo stato delle uscite.

Nota: Se non è possibile ottenere il valore richiesto la funzione ritorna zero e viene elevato un errore _ERRRPCFAILED (vedi RPC failure) leggibile con la funzione *catch()* oppure *err()* .

send

invia dati ad un partner remoto, sintassi:

```
(byte) send( target, data1, data2, type );
```

parametri:

target espressione (byte) oggetto cui è destinato il messaggio

data1 espressione (int) valore data1

data2 espressione (int) valore data2

type espressione (byte) tipo messaggio (vedi note)

ritorno: la funzione ritorna SNDBUSY se il canale è occupato, SNDFAIL se ha fallito o SNDOK se ha avuto successo.

Note: NON DISPONIBILE nelli dispositivi -S

Il membro type può ricevere i valori compresi tra CTRLTYPEUSER fino a CTRLTYPEUSER+84 per invio messaggi connessi pacchetto controllo, e tra CTRLTYPEUSER_DGRAM fino a CTRLTYPEUSER_DGRAM+84 per messaggi non connessi (datagrammi) pacchetto controllo, o il valore zero per invio di messaggi come dato (pacchetto dato).

Altri valori sono riservati e non devono essere usati.

setAnalogTrigger

imposta un grilletto analogico, sintassi:

```
setAnalogTrigger( lowerLimit, higherLimit, channel )
```

parametri:

lowerLimit espressione (char) limite di soglia inferiore

higherLimit espressione (char) limite di soglia superiore

channel espressione (char) selettore canale analogico su cui impostare il grilletto, possono essere usate le costanti ANALOG0 e ANALOG1

ritorno: la funzione non ritorna alcun valore

Note: dopo avere impostato il grilletto, esso scatterà qualora il valore analogico esca dai margini inferiore e superiore impostati scatenando l'evento analog0 o analog1.

DISPONIBILE SOLO SU ALCUNI MODELLI con ingresso analogico

encoder

legge e ripristina il contatore encoder, sintassi:

```
(int) encoder( option );
```

parametri:

option tipo operazione:

READ	legge il valore corrente dell'encoder
RESETONZERO	legge il valore corrente dell'encoder e imposta il grilletto di azzeramento del contatore interno quando viene letto il prossimo impulso sul canale Z degli ingressi encoder
RESETNOW	azzerava immediatamente il contatore interno encoder, ritorna zero

ritorno: La funzione ritorna un valore int (con segno) del valore letto dall'encoder.

Note: **DISPONIBILE SOLO SUI MODELLI /E**

Dopo avere innescato il grilletto su azzeramento da impulso canale Z quando viene rilevato tale impulso viene anche scatenato l'evento zeroEncoder.

err

ritorna lo stato di errore gestito, sintassi:

```
(byte) err( );
```

parametri: nessuno

ritorno: la funzione ritorna il codice di errore se presente, altrimenti se non vi sono errori ritorna zero.

Note: **NON DISPONIBILE** nei dispositivi -S
Dopo avere invocato la funzione viene azzerato l'errore.

La funzione err è utilizzabile attraverso i gestori di errore **try** e **catch** (vedi)

try

istruzione try, sintassi:

```
try;
```

Attiva il gestore di errori, gli errori gestibili (ad esempio una divisione per zero) non causano una eccezione bloccando il programma e mettendo in sicurezza le uscite ma registrano l'errore che può essere letto o catturato rispettivamente con le funzioni **err()** o **catch()** (vedi)

E' importante disattivare il gestore di errori dopo il suo uso mediante la funzione **catch()**.

catch

cattura un errore ed annulla il gestore di errori, sintassi:

```
catch( );
```

parametri: nessuno.

ritorno: la funzione ritorna il codice di errore o zero se nessun errore si era generato.

Valori possibili:

`_ERRDIVBYZERO` divisione per zero

`_ERRRPCFAILED` esecuzione evento pubblico fallita

`_ERRSUBSCRIPT` accesso ad elemento di matrice non esistente

Note: NON DISPONIBILE nei dispositivi -S

La funzione disattiva il gestore di errori attivato con `try`, inoltre azzerava l'errore.

OUT

ritorna lo stato delle uscite, sintassi:

```
(outbuf) OUT( );
```

parametri: nessuno.

ritorno: la funzione ritorna un valore con tipo `outbuf`.

Esempio:

```
if( OUT().2 ) ;
```

wait

Istruzione. Attesa responso chiamata remota. Sintassi:

```
wait;
```

Note: Questa istruzione è disponibile solo a partire dalla versione 1.8 del firmware e 1.0.37 del compilatore.

Descrizione: Se risulta imposto lo stato di occupato il processo quando incontra questa istruzione si arresta (ma altri eventi possono nel frattempo essere processati) e attende la ricezione della conferma di esecuzione dal partner remoto entro un tempo di 2,24 secondi, trascorso inutilmente il quale il processo riprende ma viene attivato l'errore `_ERRRPCFAILED` (vedi RPC failure) che può essere letto usando la funzione `catch()`;

Il partner remoto deve inviare il messaggio `CTRLTYPE_OK` per sbloccare l'attesa senza fare cadere il timeout (vedi funzione `sendOK`).

Lo stato di occupato viene imposto automaticamente dopo una esecuzione remota oppure esplicitamente usando l'istruzione `setbusy`.

setbusy

Istruzione. Imposta lo stato di occupato. Sintassi:

```
setbusy;
```

Note: Questa istruzione è disponibile solo a partire dalla versione 1.10 del firmware e 1.0.44 del compilatore.

Lo stato occupato viene usato dall'istruzione *wait* per bloccare un processo in attesa di una risposta di rete con messaggio CTRLTYPE_OK (vedi funzione *sendOK*) oppure lo scadere del timeout.

resetbusy

Istruzione. Rimuove lo stato di occupato. Sintassi:

```
resetbusy;
```

Note: Questa istruzione è disponibile solo a partire dalla versione 1.10 del firmware e 1.0.44 del compilatore.

setsync

Istruzione. Imposta la sincronizzazione della chiamata RPC che immediatamente deve seguire, va usato in combinazione a *wait*. Sintassi:

```
setsync;
```

La sincronizzazione è disponibile solo per dispositivi dotati di firmware 1.10 o successivo e non per le versioni SMALL.

La sincronizzazione consente di attendere che l'esecuzione remota sia completata prima di procedere offrendo in questo modo una grande flessibilità potendo lanciare esecuzioni asincrone e sincrone, e permette di guidare situazioni che potrebbero creare picchi di traffico dati difficili da gestire.

Il partner che viene chiamato, se non ha il firmware aggiornato può inviare mediante la funzione send il tipo messaggio CTRLTYPE_OK per sincronizzare il chiamante (non serve per le versioni dalla 1.10+). Se il partner remoto non risponde, dopo un timeout di circa 4 secondi l'esecuzione procede (vedi istruzione *wait* per dettagli).

Se dopo setsync viene eseguita una funzione locale (o un evento pubblico che si trova però nello stesso oggetto) la sincronizzazione viene automaticamente annullata, allo stesso modo che la chiamata non fosse locale.

Esempio:

```
event click(int Clicked) {
    setsync; // imposta la sincronizzazione
    remote_partner.azione(...); // esegue la chiamata
                                // remota (o locale)
    wait; // attende la fine dell'esecuzione remota,
          // se la funzione o evento pubblico è locale
          // wait viene ignorato
}
```

sendOK

Funzione. Invia al destinatario specificato il messaggio CTRLTYPE_OK. Usato in combinazione con *wait* permette di sbloccare il processo in attesa di esito. Sintassi:

```
(void) sendOK(byte target);
```

Argomenti: target oggetto destinatario del messaggio.

Note: Questa istruzione è disponibile solo a partire dalla versione 1.10 del firmware e 1.0.44 del compilatore.

Nota: Questa funzione in genere non serve se si usa setsync, ha utilità solo per sincronizzazioni manuali tra processi usando le istruzioni *setbusy* e *wait*.

read

Legge dati nella memoria statica. Sintassi:

```
(int) read(byte location);
```

argomenti:

location posizione nel file dei dati, valori validi sono tra zero e MAXMEM-1 che dipende dal tipo di dispositivo (esempio se il dispositivo dispone di 980 byte di memoria statica MAXMEM corrisponde a tale valore). Se location viene combinato con *_RW_INT_* viene letto un valore int, diversamente viene letto un valore byte. L'allineamento deve essere rispettato in caso di scrittura.

ritorno La funzione ritorna il valore richiesto

write

Scrive dati nella memoria statica. Sintassi:

```
write(int value,byte location);
```

argomenti:

value dato da scrivere

location posizione nel file del dato, valori validi sono tra zero e MAXMEM-1 che dipende dal tipo di dispositivo (esempio se il dispositivo dispone di 980 byte di memoria statica MAXMEM corrisponde a tale valore). Se location viene combinato con *_RW_INT_* viene scritto un valore int, diversamente viene scritto un valore byte. L'allineamento deve essere rispettato in caso di lettura.

readSched

Legge un byte dalla schedulazione. Sintassi:

```
(byte) readSched(int location);
```

argomenti:

location posizione nella schedulazione del byte da leggere. Fare riferimento alla documentazione relativa allo schedulatore.

writeSched

Scrive un byte nella schedulazione. Sintassi:

```
writeSched(byte value, int location);
```

argomenti:

value valore da scrivere nella posizione specificata da location.

location posizione nella schedulazione in cui scrivere il byte.

Nota: per valori e posizioni fare riferimento alla documentazione dello schedulatore.

insertSched

Inserisce un evento temporale nello schedulatore. Sintassi:

```
insertSched(byte value, byte hour, byte minute,  
             byte stype, byte days);
```

argomenti:

value valore dell'evento (dipendente dalla gestione che se ne fa),
 esempio: valore numerico, valore di impostazione, ecc.

hour, minute ora e minuti di scadenza dell'evento

stype tipo della schedulazione (può essere da 0 a 3)

days giorni in cui l'evento va considerato, il bit zero è Lunedì,
 il bit 6 Domenica.

Note: la funzione provvede ad inserire in modo ordinato temporalmente la scadenza, se un'altra scadenza esiste alla stessa ora, stesso tipo e stessi giorni, aggiorna il valore; se la scadenza ha stesso valore, stessa ora, stesso tipo, aggiorna i giorni; la scadenza non viene inserita se ne esiste una di uguale identica.

Note: Disponibile solo sui dispositivi dotati di schedulatore e orologio (cronogestori), versione firmware 1.12 o successiva.

deleteSched

Elimina un evento temporale dallo schedulatore. Sintassi:

```
deleteSched(byte index);
```

argomenti:

index indice dell'evento temporale da eliminare, se l'evento non
 esiste, la funzione non fa niente.

Note: Disponibile solo sui dispositivi dotati di schedulatore e orologio (cronogestori), versione firmware 1.12 o successiva.

freezeScheduler

blocca lo schedulatore. Sintassi:

```
freezeScheduler(byte type,byte set);
```

argomenti:

type tipo di schedulazione, può assumere i valori da zero a 3

set booleano, true congela la schedulazione indicata da type, false la riattiva.

Note: Disponibile solo sui dispositivi dotati di schedulatore e orologio (cronogestori).

Commenti

La schedulazione congelata diventa inattiva ma non si arresta.

Se mentre la schedulazione è congelata si verifica un evento questo viene ignorato e l'evento *scheduler* per quel tipo di schedulazione non viene emesso.

Riattivando la schedulazione, se nel frattempo era scaduto un valore schedulato questo sarà perduto (fino al giorno dopo, salvo eccezioni), e l'evento *scheduler* non sarà emesso per quel tipo di schedulazione fino al successivo evento.

Vedi anche: esempio schedulatore

ResetAll

Reset generale di tutto il sistema. Sintassi:

```
ResetAll( );
```

Note: Il reset generale investe tutti i dispositivi collegati, viene cancellato l'eventuale stato di errore e riavviato il sistema.

print

Stampa su video, display o stampante (dipendentemente dal dispositivo) testi e valori numerici formattati. Sintassi:

```
print( [arg] [,arg [,... ] ] );
```

Argomenti: La funzione accetta costanti stringa e numeri, vedi descrizione.

Ritorno: Nessun valore.

La funzione print può accettare un numero variabile di argomenti sia di tipo string (costante) che di tipo numerico (byte, char, int, float), fino ad un massimo di 8 argomenti.

Argomenti numerici

Se l'argomento è un numero è atteso un successivo argomento che ne specifica la formattazione. Sono possibili le seguenti forme (i valori possono essere tra loro combinati, salvo incompatibilità - vedi note):

NULL default, il numero viene mostrato così com'è.

fmt_ABS rappresentazione assoluta, il segno non viene mostrato ed il suo spazio rimosso, allineando il numero alla sinistra senza spazi. Se combinato con un numero di digit maggiore di zero gli spazi a sinistra non occupati vengono riempiti con zeri. Esempio: fmt_ABS | 3 rappresenta almeno 3 cifre, se il numero da rappresentare è 18 esso appare "018"

fmt_byte10 applicabile solo per valori byte, rappresenta il numero convertendo il valore byte da 0 a 255 in un numero da 0 e 999. Utile per mostrare valori in tensione letti dalle analogiche o esprimere percentuali sul livello espresso con un byte.

fmt_byte5 applicabile solo per valori byte, come fmt_byte10 ma rappresenta il valore byte convertendolo con segno: da 0 a 127 con valori da 0 a 500, e da 128 a 255 con valori da 510 a 999. Utile per mostrare valori in tensione sia positiva che negativa.

fmt_1decimal mostra il numero con 1 decimale (come se il valore fosse diviso per 10).

fmt_2decimals mostra il numero con 2 decimali (come se il valore fosse diviso per 100).

digits: da 1 a 7 numero di cifre fisse da visualizzare. Se il numero è più piccolo esso viene allineato a destra e a sinistra sono posti degli spazi (cancellando l'eventuale sfondo) per il numero di digits richiesto. Notare che il numero di digits include il segno, se ammesso (vedi fmt_ABS), quindi se si usa fmt_ABS usare un digit in meno.

print (continua)

Esempio:

```
byte n=2,nn=5; print( n, fmt_2decimals | 2, nn, 1);
```

Note: *Diversi modi possono essere combinati tra loro ma `fmt_1decimal` e `fmt_2decimals` sono tra loro mutualmente esclusivi, i formati `fmt_byte5` e `fmt_byte10` sono incompatibili con valori maggiori di 255.*

Se non si specifica `digits` (zero `digits`) nessuna spaziatura viene fatta a sinistra del numero ed esso viene allineato a sinistra, ed eventuali cifre sulla destra (se il numero precedentemente visualizzato era di più cifre) verranno abbandonate e restano visibili (creando un numero non vero e confuso).

Argomenti stringa

Possono essere passate solo stringhe costanti (dichiarate in spazio globale). Vengono rappresentati i caratteri ASCII da 32 a 192 (ASCII192) ed i caratteri di controllo: tabulazione, backspace (ritorna indietro di 1 carattere), ritorno a capo (CR) e avanzamento riga (LF) hanno lo stessa funzione e si consiglia di usare solo uno dei due.

I riferimenti a stringhe costanti possono essere dichiarati solo nello spazio globale, è possibile specificare stringhe di lunghezza massima di 254 caratteri. I caratteri speciali e quelli comando o macro contano come un solo carattere ciascuno. La stringa va racchiusa tra virgolette, eventuali virgolette facenti parte della stringa stessa devono essere precedute dalla barra rovescia (vedi caratteri speciali).

Note:

Il riferimento ad una stringa costante è in effetti come un indice che può essere coinvolto in operazioni matematiche, per indicizzare una stringa diversa ad esempio. L'uso di un indice errato causa un errore di indice della matrice non valido.

Anche se il sistema fa uso di un set di caratteri ASCII192, il compilatore dispone di un convertitore che permette di inserire direttamente caratteri ANSI. La conversione non viene eseguita sui caratteri con escape.

print (continua)

Caratteri speciali

Per inserire i caratteri speciali usare il carattere di escape “\”:

\t	tabulazione
\r e \n	ritorno a capo
\b	backspace
\x##	dove ## è un numero esadecimale <u>di 2 cifre</u>

Sono inoltre usati i seguenti escape per caratteri standard:

\\	barra rovescia “\”
\'	apice
\"	virgolette

escape speciale per comandi carattere:

`\$comando;`

Comandi carattere

I comandi carattere consentono di inserire dei caratteri speciali o macro, un comando deve iniziare per “\\$” e terminare con “;” e contenere il nome del comando, esempio: `\$euro;` stampa il carattere €. Sono disponibili i seguenti comandi:

copy	©
reg	®
°C	°C
ohm	Ω
on	☑
off	☐
dots	...
euro	€
SS	ß
watch	🕒
micro	μ
+-	±
:-	÷
1/2	½
sound	🎵
home	🏠
left	←
right	→
up	↑
down	↓
alert	⚠
lamp	💡
dot	•

print (continua)

Macro:

`\$time`; visualizza l'ora corrente nella forma: *hh:mm:ss*

`\$date`; visualizza la data corrente nella forma: *We, gg/mm/aa*

dove *We* è il giorno della settimana abbreviato (es. Lu, Ma, Me, ecc.)

`\$invert`; visualizza invertito (in negativo).

`\$normal`; visualizza normale (rimuove anche il grassetto e il sottolineato).

`\$version`; visualizza la versione del sistema operativo

`\$bold`; visualizza i caratteri in grassetto (disabilitare prima di stampare immagini)

`\$underline`; visualizza i caratteri sottolineati (disabilitare prima di stampare immagini)

Nota: L'uso di `\$invert`; e `\$normal`; ha effetto su tutti i testi e immagini successivamente disegnati sullo schermo.

Esempio

```
string testo="ciao", invertito="\$invert;";
print(testo,invertito,testo); // stampa ciaociao
                               // il secondo ciao in negativo
```

Posizionamento del testo

Il testo viene scritto nel punto correntemente specificato dal cursore nel momento in cui viene eseguita la funzione *print*. Il cursore può essere riposizionato usando la funzione *movecur*. Dopo l'esecuzione della funzione *print* il cursore viene riposizionato alla fine del testo scritto.

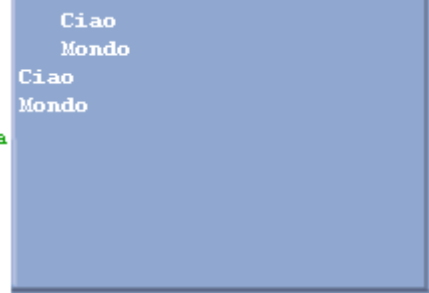
Ritorni a capo

Se il testo non ci sta in una riga questo viene rimandato a capo, nella colonna più a sinistra, automaticamente; se ciò accade nell'ultima riga disponibile, questo viene riportato nella prima riga in alto.

È anche possibile inserire ritorni a capo specifici nel testo, inserendo il carattere speciale `\n`. In questo caso il testo ritorna a capo relativamente alla colonna della posizione del cursore al momento in cui è stata eseguita la funzione *print*.

Esempio:

```
string s1="Ciao\nMondo";
movecur(0,3); //4a colonna
print(s1); //scrive Ciao Mondo
movecur(0,0); //allinea a sinistra
print(s1);
```



```
Ciao
Mondo
Ciao
Mondo
```

print (continua)

Tabulazioni

Sono disponibili punti di tabulazione, distanti 4 caratteri ciascuno. Su unità a display da 64x128 pixels essi corrispondono ai seguenti punti:

```

                                01234567890123456789012
tabulazione:                    ...^...^...^...^...^...
pixels:                          24  48  72  96 120
    
```

Le tabulazioni sono ottenute con il carattere speciale `\t`

set di caratteri ASCII192 (8x6 pixels): (parte relativa ai soli caratteri visualizzati)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	!	"	#	\$	%	&	'	()	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~	€
128	Ç	ü	é	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š
160	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š	š

Nota: Disponibile solo sui modelli dotati di stampa a video o altra periferica

printchar

stampa il carattere specificato dall'indice dato rispetto la tabella caratteri

```
printchar(char character);
```

argomenti:

character indice del carattere da stampare, riferito alla tabella caratteri.

Nota: Se il carattere ha un valore inferiore a 32 viene stampato uno spazio.

movecur

sposta il cursore di stampa. Sintassi:

```
movecur(byte row, byte col);
```

argomenti:

row riga in cui posizionare il cursore, per dispositivi con display 64x128 questo membro può assumere il valore da zero (prima riga in alto) a 7 (ultima riga in basso)

col colonna in cui posizionare il cursore, per dispositivi con display 64x128 e caratteri da 8x6 questo membro può assumere il valore da zero (prima colonna a sinistra) a 21 (ultima colonna a destra).

Nota: Disponibile solo sui modelli dotati di stampa a video o altra periferica

cls

cancella lo schermo o il display. Sintassi:

```
cls();
```

Note: Dopo avere usato la funzione cls il cursore viene riportato nell'angolo in alto a sinistra, e viene ripristinata la modalità di scrittura normale (in positivo).

Nota: Disponibile solo sui modelli dotati di stampa a video o altra periferica

clearline

cancella una riga. Sintassi:

```
clearline(byte startX, byte endX, byte row);
```

Argomenti:

startX	inizio della cancellazione lungo la riga
endX	posizione finale di cancellazione lungo la riga
row	riga da cancellare

Note: Se la funzione clearline viene usata dopo che è stata usata la macro `\$invert`; la riga viene cancellata al negativo. Il cursore viene spostato nella riga specificata, nella posizione finale indicata da endX.
Per display da 64x128 valori validi per startX ed endX sono da zero a 127, per row da zero a 7.

Nota: Disponibile solo sui modelli dotati di stampa a video o altra periferica

showicon

mostra una icona sullo schermo o display. Sintassi:

```
showicon(int ID, byte x, byte y);
```

argomenti:

ID	Indice o identificatore della icona da mostrare
x	posizione lungo l'asse x (orizzontale) espressa in pixels
y	posizione lungo l'asse y (verticale) espressa in pixels

Note: Le coordinate (x=0,y=0) corrispondono all'angolo in alto a sinistra.
Lo ID è un puntatore all'inizio del file dell'icona, l'uso di un puntatore errato causa un errore di violazione di accesso.
Il cursore viene posizionato nell'angolo in basso a destra dell'icona.

Nota: Disponibile solo sui modelli dotati di stampa a video o altra periferica grafica

glowlamp

accende la luce del display (se disponibile). Sintassi:

```
glowlamp();
```

Argomenti: nessuno. Ritorno: nessuno.

Note: La luce si spegna automaticamente dopo un certo periodo di tempo.

Attenzione: Se si usa questa funzione su dispositivi che non la prevedono si causa un errore di programma con effetti imprevedibili.

Nota: Disponibile solo sui modelli dotati di illuminatore o retroilluminazione (esempio display)

rnd

Funzione. Ritorna un numero pseudocasuale compreso tra 0 e 2040 (nei modelli privi di ingresso analogico ritorna un valore compreso tra 0 e 255)
Sintassi:

```
int x = rnd();
```

Nota. Il numero generato è sincronizzato dal clock interno. Se si deve ottenere un numero casuale a tempi regolari, per esempio nell'evento timer, si ottengono gli stessi valori a rotazione, per evitare questo usare due timer come nel seguente esempio:

```
byte rand;
setTimer(10,TIMER0); // preset timer 1 secondo
setTimer(10,TIMER1);

event timer0()
{ // cambio il punto di lettura di rnd in tempi variabili
  setTimer((rand=rnd())%13+1,TIMER0);
}

event timer1
{ // combina i valori generando uno pseudocasuale
  send(BROADCAST,rand+rnd(),0,CTRLTYPE_USERD);
  setTimer(10, TIMER1); // ricarica il timer periodico
}
```

Nell'esempio, il valore usato ad ogni rintocco del timer periodico (timer 1) può assumere un valore compreso tra zero e +2295.

§

API - Eventi

longclick

avviene dopo che un ingresso è transito da zero a uno (o un pulsante è stato premuto) ed è rimasto tale per almeno tre secondi circa.

Parametri:

(int) Clicked ingressi interessati dall'evento

NOTE: Verificare la disponibilità dell'evento, e per quali ingressi, in base al modello. Nei dispositivi dotati di tastiera l'argomento Clicked ritorna il codice di tasto premuto o il tasto premuto come bit (solo per dispositivi con ingressi e tasti in misura totale non superiore a 8), in questi ultimi il tasto può essere individuato usando la costante `__KEY0 ... __KEY3`, oppure mediante la macro `$(keypress)`.

L'evento input viene comunque segnalato.

Esempio:

```
event longclick(int Clicked)
{
    if(Clicked.2) {           // vera solo se l'ingresso IN2
                            // è in longclick
    }
}
```

net

scatenato quando arriva un messaggio di rete

parametri:

(int) par1 primo parametro

(int) par2 secondo parametro

(byte) type tipo di messaggio

(byte) sender indirizzo logico del mittente (o il nome dell'oggetto mittente)

Nota: Non disponibile nei modelli -S

timer0 ...7

scatenato quando scade un temporizzatore caricato (da timer0 a timer7)

parametri: nessuno.

Nota: Il timer relativo all'evento resterà bloccato (anche se riarmato) fino a quando la procedura dell'evento stesso non viene completata.

analog0 ...1

scatenato quando scatta il grilletto analogico impostato (vedi **setAnalogTrigger**)

parametri: nessuno.

outchange	<p>chiamato ad ogni cambiamento dello stato delle uscite.</p> <p>argomenti forniti:</p> <p>int changed stato delle uscite che hanno subito un cambiamento.</p> <p>Ogni bit reso rappresenta il cambiamento dello stato di una uscita, il bit 0 l'uscita 0, il bit 1 l'uscita 1 e così via. Se il bit è alto l'uscita ha subito un cambiamento.</p>
button	<p>chiamato quando un pulsante viene premuto dall'utente.</p> <p>argomenti forniti:</p> <p>char btn codice del pulsante premuto</p> <p>Nota: disponibile solo su alcuni dispositivi dotati di tastiera, ove non gestito, altri dispositivi dotati di tastiera elevano gli eventi click e longclick.</p>
ZeroEncoder	<p>segnalato quando l'ingresso Z vede il passaggio per lo zero dell'encoder.</p> <p>argomenti forniti: nessuno</p> <p>Note: L'evento è disponibile solo sui modelli con ingresso encoder.</p>
KeyDetected	<p>segnalato quando la chiave viene rilevata nell'apposito ricettacolo.</p> <p>argomenti forniti:</p> <p>byte k0, byte k1, byte k2, byte k3, byte k4, byte k5 k0-k5 è un blocco di byte relativo al numero di serie della chiave.</p> <p>Esempio:</p> <pre>event KeyDetected(byte k0,byte k1,byte k2,byte k3,byte k4,byte k5) { byte i, u, k; for(;u<MaxKey;u=u+6) { switch(u) { case 5:k=k5;break; case 4:k=k4;break; case 3:k=k3;break; case 2:k=k2;break; case 1:k=k1;break; default: k=k0; } if(read(u)!=k) return; // invalid key } /* key found */ OpenDoor(); } </pre> <p>Note: L'evento è disponibile solo per dispositivi dotati di ingresso per chiave elettronica.</p>

scheduler

chiamato quando un evento di schedulazione è scaduto.

argomenti forniti:

- byte type_exc tipo di schedulazione e se dovuta ad eccezione: nei 4 bit superiori sono riportate le eccezioni, in quelli inferiori le schedulazioni giornaliere-settimanali. Un bit alto indica che la corrispondente schedulazione (o sua eccezione) sono scadute. Il bit zero indica la schedulazione tipo zero, il bit 1 la schedulazione tipo uno, ecc.
Il bit 4 indica l'eccezione zero, il bit 5 l'eccezione uno, ecc.
- byte type_clr tipo di schedulazione la cui eccezione è terminata. I 4 bit meno significativi non sono usati, i 4 bit più significativi indicano le rispettive schedulazioni, da zero a 3. Se il bit è alto significa che l'eccezione al corrispondente tipo di schedulazione è terminata.
- byte value0 valore della schedulazione prevista per tipo zero. Non ha significato se nessuno dei bit della corrispondente schedulazione in type_exc (scadenza giornaliera-settimanale o eccezione) non sono alti.
- byte value1 come per value0 ma relativo alla schedulazione tipo 1
- byte value2 come per value0 ma relativo alla schedulazione tipo 2
- byte value3 come per value0 ma relativo alla schedulazione tipo 3

rappresentazione delle schedulazioni in type_exc:

(MSb) bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0 (LSb)
eccezione sched. 3	eccezione sched. 2	eccezione sched. 1	eccezione sched. 0	scadenza sched. 3	scadenza sched. 2	scadenza sched. 1	scadenza sched. 0

rappresentazione delle schedulazioni in type_clr:

eccezione sched. 3 terminata	eccezione sched. 2 terminata	eccezione sched.1 terminata	eccezione sched. 0 terminata	-	-	-	-
------------------------------	------------------------------	-----------------------------	------------------------------	---	---	---	---

Note: Disponibile solo sui dispositivi dotati di schedulatore.

Sono possibili fino a 4 tipi di schedulazioni temporali indipendenti, ciascuna con scadenze giornaliere-settimanali o eccezioni periodiche.

Quando una termina l'eccezione ad una schedulazione, il rispettivo valore (passato con l'argomento value*n*) corrisponde a quello che c'era prima della eccezione.

Esempio schedulatore

```
object Aggeggio
{
    // in area globale
    byte valorecorrente0;
    setTimer($(time 0:0:10:0),TIMER0); // carica timer a 10 secondi

    event scheduler(byte type_exc,byte type_clr,byte value0,byte value1,byte value2,byte value3)
    {
        if(type_exc & 0x1) { // test scadenza 0
            valorecorrente0 = value0; // ad ogni evento scaduto tiene il nuovo valore
        }
    }

    event timer0()
    {
        // invia il valore ogni 10 secondi
        setTimer($(time 0:0:10:0),TIMER0);
        send(BROADCAST,valorecorrente0,0,CTRLTYPE_USERDGRAM);
    }

    pubevent bloccaSchedulatore(int arresta, byte sender)
    {
        // funzione richiamabile da altri oggetti, blocca riattiva la schedulazione 0
        freezeScheduler(0, arresta);
    }
}
```

powerdown

chiamato quando il dispositivo rileva una condizione di power down (la tensione scende sotto la soglia minima) o riceve un messaggio di powersave.

Argomenti: nessuno

Note: L'evento viene scatenato solo se il dispositivo rileva un abbassamento della tensione (o riceve un messaggio di rete di powersave) e non si trova già in stato di power save, nel cui caso l'evento non viene generato.

Una volta emesso questo evento il dispositivo risulta già in stato di power save. In tale condizione le uscite sono rilasciate, ma viene mantenuto internamente lo stato logico di esse. Qualsiasi mutamento delle uscite sarà registrato e ripristinato quando cessano le condizioni di power save. (vedi powerup event)

Attenzione! Nello stato di powersave il dispositivo non risponde alle notifiche di cambio di stato delle uscite (solo le notifiche corrispondenti al rilascio dei relè e alla ripresa degli stessi viene inviata).

Nota: La soglia di tensione minima a cui scatta il power down è tipicamente 18V, mentre la soglia di tensione minima di blocco del dispositivo è generalmente sotto agli 11V (il blocco inibisce ogni attività, e tutto ciò che non è stato memorizzato nella memoria statica viene perduto).

powerup

chiamato quando il dispositivo rileva la condizione per il ripristino della potenza (la tensione risale sopra la soglia minima e vi rimane per almeno un tempo minimo predefinito - tipico 15 secondi) oppure riceve un messaggio di rete di powerrestore.

Argomenti: nessuno

Nota: l'evento viene scatenato solo se il dispositivo rileva le condizioni per un ripristino della potenza (o riceve un messaggio di ripristino) e sia ancora in stato di powersave. Quando questo evento viene generato il dispositivo è già in stato di potenza ripristinata. Quando questo si verifica le uscite riprendono lo stato che avrebbero avuto in assenza di power save (e non lo stato che c'era prima del power save), ovvero tengono fede ad eventuali operazioni eseguite anche durante la condizione di power save.

Attenzione! L'eventuale notifica dei cambiamenti dello stato delle uscite avviene in relazione allo stato ripristinato.

TChange

chiamato ad ogni cambiamento rilevato sulla sonda di temperatura.

argomenti forniti:

(int) temperature	valore temperatura letta in quel momento
(byte) item	senore, questo valore è sempre zero

Note:

L'evento è disponibile solo per i dispositivi dotati di sensore termico o di ingresso per sonda termica one-wire. Il valore reso deve essere convertito in base al sensore. Per sensori DS18S20 dividere per due il valore per ottenere il dato in °C con risoluzione di mezzo grado.

Nel caso siano disponibili più sensori l'evento viene sparato se è stata rilevata una variazione di temperatura su uno qualsiasi dei sensori, ma il valore passato dall'evento è **sempre riferito al primo sensore** (sensore zero) **anche se questo non ha avuto variazioni di temperatura.**

In questo caso è necessario usare la funzione **getIn** passando come argomento DGTDATA +c dove c è il canale da leggere (da zero a N-1).

Se il sensore non è collegato, o non risponde o è guasto, viene reso il valore costante 170 (0xaa) che corrisponde a +85°C.

Tenere conto che il campionamento del sensore avviene con un intervallo di circa 20 secondi tra una lettura e la successiva, usare ripetutamente l'istruzione getIn(DGTDATA) prima dello scadere dell'intervallo ritorna sempre l'ultimo valore letto o il valore 170 in caso di errore.

Con sensori DS18S20 i valori resi vanno da -110 (-55°C) a 170 (+85°C).

§

API - Costanti predefinite

Costanti predefinite

Queste sono costanti predefinite per l'uso con le funzioni o eventi definiti dall'API

<i>nome costante</i>	<i>descrizione</i>	<i>applicabile a:</i>
<i>- generici -</i>		
this	speciale, indica l'oggetto stesso	riferimenti a oggetto
false		
true		
OFF		
ON		
<i>- per analogica e ingressi -</i>		
ANALOG0	selettore analogica 0 (ingresso o uscita)	funzioni getIn, setAnalogTrig
ANALOG1	selettore analogica 1 (ingresso o uscita)	funzioni getIn, setAnalogTrig
DIGITAL	seleziona gli ingressi logici (in this)	funzione getIn
DGTDATA	selettore canale dati digitale	funzione getIn
<i>- per timer -</i>		
TIMER0	selettore temporizzatore 0	funzione setTimer, getTimer
TIMER1	selettore temporizzatore 1	funzione setTimer, getTimer
TIMER2	selettore temporizzatore 2	funzione setTimer, getTimer
TIMER3	selettore temporizzatore 3	funzione setTimer, getTimer
TIMER4	selettore temporizzatore 4	funzione setTimer, getTimer
TIMER5	selettore temporizzatore 5	funzione setTimer, getTimer
TIMER6	selettore temporizzatore 6	funzione setTimer, getTimer

<i>nome costante</i>	<i>descrizione</i>	<i>applicabile a:</i>
TIMER7	selettore temporizzatore 7	funzione setTimer, getTimer
<i>- per encoder -</i>		
READ	lettura	funzione encoder
RESETONZERO	legge e reset su impulso canale Z	funzione encoder
RESETNOW	legge e reset immediato	funzione encoder
<i>- per send o chiamate evento tra oggetti -</i>		
SNDOK	esecuzione avvenuta con successo	
SNDBUSY	esecuzione non avvenuta, canale occupato	
SNDFAIL	esecuzione non avvenuta o fallita, destinazione irraggiungibile	
BROADCAST	da usare come target	funzioni send, set e setOut
CTRLTYPEUSER	tipo messaggio privato per pacchetto controllo connesso, valori validi da CTRLTYPEUSER a CTRLTYPEUSER+84	funzione send
CTRLTYPEUSER_DGRAM	tipo messaggio privato per pacchetto controllo non connesso, valori validi da CTRLTYPEUSER_DGRAM a CTRLTYPEUSER_DGRAM+84	funzione send
<i>- array indicatori stato ingressi o uscite -</i>		
_MASK	selettore ingressi o uscite modificati per variabili outbuf	
_DATA	selettore valori ingressi o uscite per variabili outbuf esempio: <pre> outbuf u; u.1 = ON; send(station,u[_DATA],u[_MASK],0); </pre>	
<i>- errori -</i>		
_ERRDIVBYZERO	errore divisione per zero	funzione catch, query stato dispositivo

<i>nome costante</i>	<i>descrizione</i>	<i>applicabile a:</i>
_ERRRPCFAILED	esecuzione evento pubblico fallita	funzione catch, query stato dispositivo
_ERRSUBSCRIPT	accesso ad elemento di matrice non esistente	funzione catch, query stato dispositivo
_ERRSTOPPED	dispositivo in stato di stop	query stato dispositivo
<i>- dimmer - (nota: verificare le capacità del dispositivo)</i>		
DIMMERCH0	selezione canale dimmer zero	funzione setOut
DIMMERCH1	selezione canale dimmer uno	funzione setOut
DIMMERCH2	selezione canale dimmer due	funzione setOut
DIMMERCH3	selezione canale dimmer tre	funzione setOut
BASEDIMMERCH	numero di base per selezione canale dimmer (uso: BASEDIMMERCH+n)	funzione setOut
<i>- lettura/scrittura dati -</i>		
_RW_INT_	legge o scrive il dato come int invece che byte	funzioni read, write
<i>- orologio - (solo dispositivi dotati di orologio)</i>		
GET_SECOND, GET_MINUTE, GET_HOUR, GET_DAY_WEEK, GET_MONTH, GET_YEAR	ritorna rispettivamente i secondi, i minuti, l'ora, il giorno e giorno della settimana, il mese e l'anno corrente dell'orologio in tempo reale. Usare la macro \$(getWeekDay) e \$(getDay) per ottenere dal valore reso con GET_DAY_WEEK il giorno della settimana e il giorno. Vedi anche __FILTERDAY, __FILTERWEEK	funzione getTimer
__FILTERWEEK	usato con il valore reso da getTimer(GET_DAY_WEEK) estrae il giorno della settimana, dove Lunedì corrisponde a __FILTERWEEK, Martedì a __FILTERWEEK+1, ecc.	getTimer

<i>nome costante</i>	<i>descrizione</i>	<i>applicabile a:</i>
<code>__FILTERDAY</code>	usato con il valore reso da <code>getTimer(GET_DAY_WEEK)</code> estrae il giorno.	<code>getTimer</code>

§

Appendice

Codici tastiera

I seguenti codici tastiera sono resi dagli argomenti *IN* e *Clicked* per i dispositivi dotati di tastiera senza ingressi logici, oppure di tastiera e ingressi logici in numero totale (ingressi + tasti) superiore a 8.

12 tasti (codici illustrati con base esadecimale)

	<i>a</i>	<i>b</i>	<i>c</i>
4	7 <small>80</small>	8 <small>81</small>	9 <small>82</small>
3	4 <small>40</small>	5 <small>41</small>	6 <small>42</small>
2	1 <small>20</small>	2 <small>21</small>	3 <small>22</small>
1	0 <small>10</small>	x <small>11</small>	✓ <small>12</small>

14 tasti (codici illustrati con base esadecimale)

	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>
4	7 <small>80</small>	8 <small>81</small>	9 <small>82</small>	*
3	4 <small>40</small>	5 <small>41</small>	6 <small>42</small>	
2	1 <small>20</small>	2 <small>21</small>	3 <small>22</small>	#
1	0 <small>10</small>	x <small>11</small>	✓ <small>12</small>	

4 tasti

__KEY0	__KEY1	__KEY2	__KEY3
10h	20h	40h	80h
↑	↓	✓	x

NSC Riferimento al linguaggio di programmazione, di Claudio Ghiotto

Riconoscimenti

Sistema operativo a tecnologia distribuita retificata NSC, ©2002-2005
Soft&Media technologies, Ghiotto Claudio

Tecnologia Networked Shared Control (Networked Smart Controller): brevetto depositato; tecnologia componenti virtuali: brevetto depositato.

Protocollo di trasmissione dati ad atomi per bus di campo e reti Simple FieldBus Protocol (SFBP), ©2002 Claudio Ghiotto e Paolo Marchetto (open document license), riferimenti: www.smartcontroller.org

C semplificato ©2002-2003 Soft&Media technologies, autore Claudio Ghiotto

©2006, 2007, 2008 ByBus srl

Stampa Novembre 2006

Ristampe: Aprile 2008

ByBus s.r.l. via S. Bernardino 44,

IT 36075 Montecchio Maggiore, Vicenza

www.bybus-italia.com - www.automazione-domotica.it

tel. +39 0444 490370 fax +39 0444 1830 405

Made in EU