Training per esperti

TECNICHE AVANZATE DI I/O SU FILE

Spesso, la decisione di dividere in processi differenti la produzione dei dati dall'uso degli stessi viene presa perché occorre scrivere i dati su file man mano che sono acquisiti

a cura di Matteo Foini

Al loro livello più basso, tutti i file scritti sull'hard disk del vostro computer sono una serie di bit binari. Tuttavia, sono disponibili molti formati per l'organizzazione e la rappresentazione dei dati. In LabVIEW, tre delle tecniche più comuni di memorizzazione dei dati sono il formato di file Ascii, la memorizzazione diretta in binario e il formato di file TDM.
Ognuno di questi formati ha dei vantaggi e alcuni formati lavorano meglio per immagazzinare certi tipi di dato piuttosto che altri.
Questa lezione spiega i formati di file Ascii e Binario mentre il Text Data Exchange (TDM) verrà trattato dettagliatamente nel prossimo numero della rivista.

QUANDO USARE FILE DI TESTO (ASCII)

Usate i file in formato testuale per i vostri dati per renderli disponibili ad altri utenti o applicazioni se lo spazio sul disco e la velocità di I/O del file non sono cruciali, se non avete bisogno di effettuare lettura e scrittura ad accesso casuale e se la precisione numerica non è importante.
I file di testo sono il formato più facile da usare e condividere.
Quasi ogni computer può leggere o scrivere su un file di testo. Una gran varietà di programmi basati su testo possono leggere i file di testo. Memorizzate i dati in file di testo quando volete accedervi da un'altra applicazione, come un programma di scrittura o un'applicazione del tipo foglio elettronico.
Per immagazzinare i dati in formato di testo, usate le funzioni **String** per convertire tutti i file in stringhe di testo.
I file di testo possono contenere informazioni su differenti tipi di dato. I file di testo tipicamente occupano più memoria dei file binari e datalog se il dato non è originariamente in forma testuale, come i dati di un grafico, perché la rappresentazione Ascii dei dati usualmente è più grande dei dati stessi. Per esempio, il numero –123,4567 può essere memorizzato in 4 byte come numero a virgola mobile a singola precisione. Tuttavia, la sua rappresentazione Ascii richiede 9 byte, uno per ogni carattere. Inoltre, è difficile accedere casualmente ai dati numerici nei file di testo. Sebbene ogni carattere di una stringa occupi esattamente 1 byte di spazio, lo spazio richiesto per esprimere un numero come testo tipicamente non è fisso. Per trovare il nono numero in un file di testo, LabVIEW deve prima leggere e convertire i precedenti otto numeri. Potreste perdere in precisione memorizzando dati numerici in un file di testo.
I computer conservano i dati numerici come dati binari e tipicamente voi scrivete dati numerici su un file di testo nella notazione decimale. Si potrebbe avere una perdita di precisione quando scrivete i dati sul file di testo. La perdita di precisione non è un problema con i file binari.

QUANDO USARE FILE BINARI

La memorizzazione dei dati binari, come un numero intero, usa un numero fisso di byte sul disco. Per esempio, immagazzinare ogni numero da 0 a 4 miliardi in formato binario, come 1, 1.000 o 10.000, richiede fino a 4 byte per ogni numero. Usate i file binari per salvare dati numerici e per accedere a numeri specifici da un file o per accedervi casualmente. I file binari sono leggibili solo dalle macchine, diversamente dai file di testo, che sono leggibili dagli umani.
Potete usare più tipi di dati nei file binari, ma questo è insolito. I file binari sono più efficienti perché usano meno spazio sul disco e perché non avete bisogno di convertire i dati in e da un rappresentazione di testo quando conservate e recuperate i dati.
Un file binario può rappresentare 256 valori in un byte di spazio su disco. Spesso, i file binari contengono un'immagine byte per byte del dato come è stato memorizzato, tranne in casi come i valori numerici estesi e complessi. Quando il file contiene un'immagine byte per byte del dato come è stato conservato in memoria, la lettura del file è più veloce perché non c'è bisogno di conversione.

FILE DATALOG

Uno specifico tipo di file binario, conosciuto come datalog file, è il modo più semplice per registrare dati cluster su file. I file datalog conservano array di cluster con una rappresentazione binaria.
I file datalog forniscono un'efficiente memorizzazione dei dati e accesso casuale, tuttavia il formato di memorizzazione dei file datalog è complesso e perciò è di difficile accessibilità in tutti gli ambienti diversi da LabVIEW. Inoltre, per accedere ai contenuti di un file datalog, dove-

te conoscere i contenuti del tipo di cluster memorizzato nel file. Se perdete la definizione del cluster, il file diventa veramente difficile da decodificare. Per questa ragione i file datalog non sono raccomandati per condividere i dati con altri o per immagazzinare i dati di grandi organizzazioni in cui potreste perdere o mal posizionare la definizione del cluster.

FILE BINARI

Sebbene tutti i metodi di I/O dei file creino per lo più file binari, potete interagire direttamente con un file binario usando la funzione Binary File. Di seguito trovate alcune funzioni comuni che interagiscono con i file binari.

Open/Create/Replace File - Questa funzione apre un riferimento a un file nuovo o esistente per file binari o Ascii.

Write Binary File - Questa funzione scrive dati binari su un file. La funzione lavora in modo molto simile alla funzione Write to Text File, ma può accettare la maggior parte dei tipi di dati.

Read Binary File - Questa funzione legge dati binari cominciando dalla posizione di file corrente.
Dovete specificare alla funzione il tipo di dato da leggere. Utilizzate questa funzione per accedere ad un singolo elemento di dato o per collegare un valore all'ingresso di conteggio.
Ciò provoca la restituzione da parte della funzione di un array del tipo di dato specificato.

Get File Size - Questa funzione restituisce la dimensione del file in byte. Usate questa funzione in combinazione con la funzione Read Binary File quando volete leggere tutto il file binario.
Ricordate che se state leggendo elementi dei dati più grandi di un byte dovete regolare il conteggio da leggere.

Get/Set File Position - Queste funzioni rilevano e impostano la locazione nel file in cui bisogna leggere e scrivere. Usate queste funzioni per l'accesso casuale al file (Random File Access).

Close File - Questa funzione chiude un riferimento aperto a un file.

La fig. 1 mostra un esempio di scrittura di un array di numeri in doppia precisione su un file binario.
Fate riferimento alla sezione Array di questa lezione per avere maggiori informazioni sull'opzione **Prepend array or string size**?.

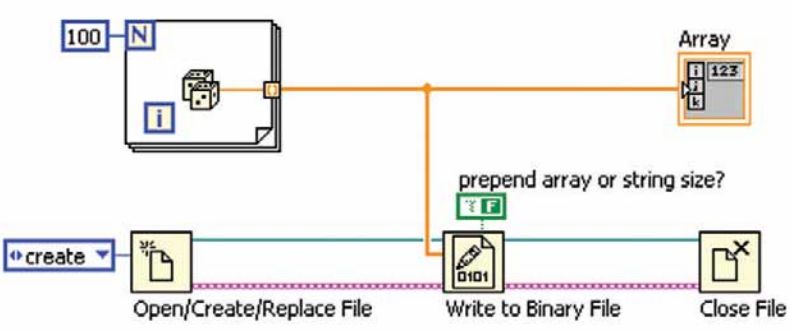

Fig 1 - Scrittura di un file binario

RAPPRESENTAZIONE BINARIA

Ogni tipo di dati in LabVIEW è rappresentato in un modo specifico quando viene scritto su un file binario.
Di seguito viene discussa la rappresentazione di ogni tipo e i problemi più importanti quando si ha a che fare con la rappresentazionie binaria di quel tipo.

Suggerimento Un bit è un valore binario singolo. Rappresentato da 1 o da 0, ogni bit è on o off. Un byte è una serie di 8 bit.

BOOLEANI

LabVIEW rappresenta i valori booleani con valori a 8 bit in un file binario. Un valore con tutti zero rappresenta False. Ogni altro valore rappresenta True.

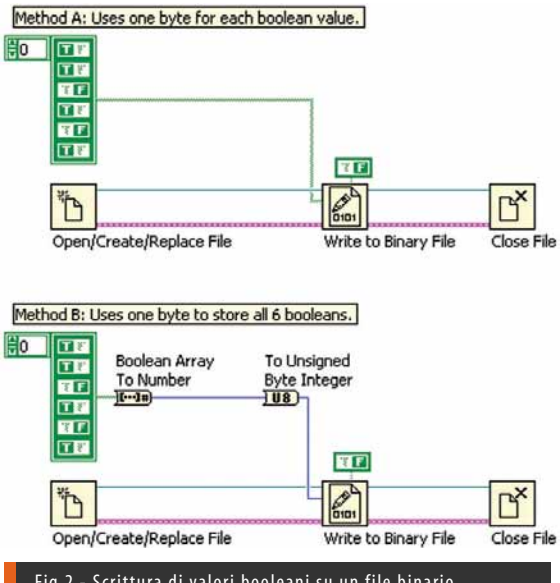

Fig 2 - Scrittura di valori booleani su un file binario

Questo divide i file in parti della dimensione di byte e semplifica la lettura e il trattamento dei file.
Per memorizzare efficientemente i valori booleani, convertite una serie di valori booleani in un numero intero usando la funzione Boolean Array To Number.
La Figura 2 mostra due metodi per scrivere sei valori booleani su un file binario.

La tab.1 visualizza una rappresentazione binaria dei contenuti del file che risulta dall'avviamento del programma della fig. 2.
Notate che il Metodo B è un metodo più efficiente di memorizzazione.

Metodo A	00000001 00000001 00000000 00000001 00000000 00000001
Metodo B	00101011

Tab. 1 - Risultati della fig. 2

NUMERI INTERI A 8 BIT

I numeri interi a 8 bit senza segno (U8) corrispondono direttamente ai byte scritti sul file.
Quando dovete scrivere dei valori di vario tipo su un file binario, convertite ogni tipo in un array di U8 usando le funzioni Boolean Array To Number, String to Byte Array, Split Number e Type Cast.
Successivamente potete concatenare i vari array di U8 e scrivere l'array risultante su un file.
Questo processo non è necessario quando scrivete su un file binario che contiene solo un tipo di dati.

Valore Binario	Valore U8
00000000	0
00000001	1
00000010	2
11111111	255

Tab. 2 - Rappresentazione U8

ALTRI NUMERI INTERI

I numeri interi multi-byte sono spezzati in due byte separati e immagazzinati in file con ordine di byte little endian o big endian. Usando il VI Write to Binary File, potete scegliere se immagazzinare i vostri dati nel formato little-endian o big-endian. L'ordine dei byte little-endian memorizza per primo il byte meno significativo e per ultimo il byte più significativo.
I Computer Macintosh tradizionalmente hanno usato l'ordine little endian e spesso rappresentano i dati internamente in LabVIEW.
L'ordine dei byte big-endian memorizza per primo il byte più significativo e per ultimo il byte meno significativo. La maggioranza dei programmi Windows utilizza il big-endian per memorizzare i dati sui file.

Valore U32	Valore little-endian	Valore big-endian
0	00000000 00000000	00000000 00000000
	00000000 00000000	00000000 00000000
1	00000001 00000000	00000000 00000000
	00000000 00000000	00000000 00000001
255	11111111 00000000	00000000 00000000
	00000000 00000000	00000000 11111111
65535	11111111 11111111	00000000 00000000
	00000000 00000000	11111111 11111111
4294 967 295	11111111 11111111	11111111 11111111
	11111111 11111111	11111111 11111111

Tab. 3 - Rappresentazioni di interi

NUMERI A VIRGOLA MOBILE

I numeri a virgola mobile sono memorizzati come descritto nello Standard Ieee 754 su Binary Floating-Point Arithmetic (Aritmetica binaria a virgola mobile). I numeri a singola precisione utilizzano ognuno 32 bit e i numeri a doppia precisione utilizzano ognuno 64 bit. La lunghezza dei numeri a precisione estesa dipende dal sistema operativo.

STRINGHE

Le stringhe sono memorizzate come serie di interi a 8 bit senza segno, ognuno dei quali è un valore della Tabella Ascii Character Code Equivalents. Questo significa che non c'è differenza tra scrivere stringhe con le funzioni Binary File e scriverle con le funzioni Text File.

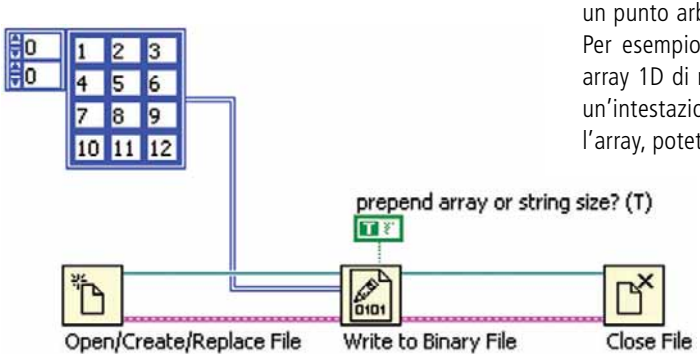

Fig. 3 - Scrittura di un array 2D di numeri interi senza segno su un file con intestazione

ARRAY

Gli array sono rappresentati come un elenco sequenziale di ognuno dei loro elementi.

La rappresentazione reale di ogni elemento dipende dal tipo di elemento.

Quando memorizzate un array su un file avete l'opzione di far precedere all'array un'intestazione.

Un'intestazione contiene un numero intero a 4 byte che rappresenta la grandezza di ogni dimensione.

Perciò un array bidimensionale con un'intestazione contiene due numeri interi, seguiti dai dati dell'array.

La fig. 3 mostra un esempio di scrittura di un array bidimensionale a numeri interi a 8 bit su un file con un'intestazione. Il terminale **prepend array or string size**? della funzione Write Binary File abilita l'intestazione.

Notate che il valore di default di questo terminale è True. Perciò le intestazioni sono aggiunte di default a tutti i file binari.

La tab. 4 mostra lo schema del file che il codice genera nella fig. 3. Notate che le intestazioni sono rappresentate da numeri interi a 32 bit anche se i dati sono interi a 8 bit.

4	3	1	2	3	4	5	6	7	8	9	10	11	12

Tab. 4 - Esempio di rappresentazione di array in file binario

CLUSTER

I file datalog rappresentano meglio i cluster nei file binari. Fate riferimento alla sezione *File Datalog* per avere maggiori informazioni.

CONFRONTO TRA ACCESSO SEQUENZIALE E CASUALE

Quando si legge un file binario, ci sono due metodi per accedere ai dati. Il primo è di leggere ogni voce in ordine, partendo dall'inizio di un file.

Questo è chiamato accesso sequenziale e lavora come la lettura di un file Ascii. Il secondo è di accedere ai dati in un punto arbitrario all'interno del file.

Per esempio, se sapete che un file binario contiene un array 1D di numeri interi a 32 bit che è stato scritto con un'intestazione e volete accedere alla decima voce dell'array, potete calcolare lo scostamento di byte di quell'elemento nel file e poi leggere solo quell'elemento. In questo esempio l'elemento ha uno scostamento di 4 (l'intestazione) +10 (l'indice dell'array) * 4 (il numero di byte in un I32) =44. L'accesso ai dati in questo modo è conosciuto come accesso casuale (random access).

ACCESSO SEQUENZIALE

Per accedere sequenzialmente a tutti i dati di un file, potete chiamare la funzione Get File Size e usare il risultato per calcolare il numero di voci del file, in base alla dimensione di ogni voce e alla configurazione del file.

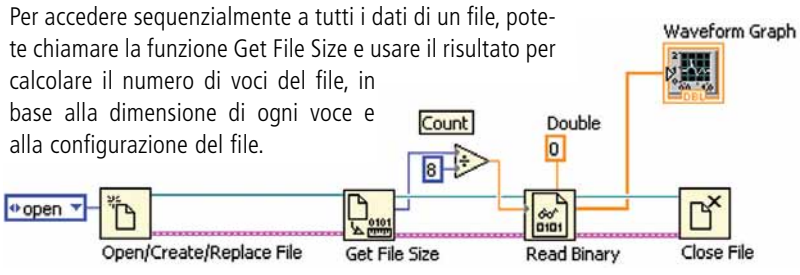

Fig. 4 - Lettura sequenziale di un intero file

Potete poi collegare il numero delle voci a un terminale di conteggio della funzione Read Binary. La fig. 4 mostra un esempio di questo metodo.

Alternativamente potete accedere sequenzialmente al file una voce alla volta chiamando la funzione Read Binary con conteggio di default pari a 1.

Ogni operazione di lettura aggiorna la posizione all'interno del file in modo che leggiate una nuova voce ogni volta che è chiamata la lettura.

Quando usate questa tecnica per accedere ai dati potete controllare l'errore End of File dopo aver chiamato Read Binary o calcolare il numero di letture necessarie per raggiungere la fine del file usando Get File Size.

ACCESSO CASUALE

Per accedere casualmente a un file binario, usate il VI Set Position per impostare l'offset della lettura dal punto del file da cui volete cominciare la lettura.

Notate che lo scostamento è in byte. Perciò dovete calcolare lo scostamento in base allo schema organizzativo del

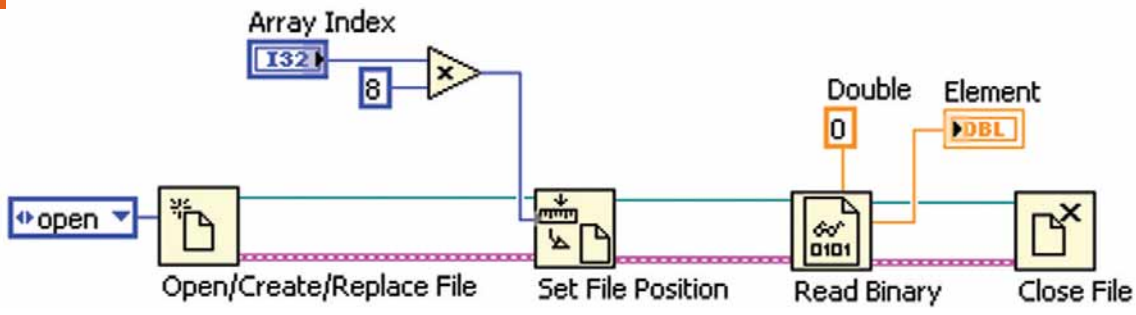

Fig. 5 - Accesso casuale a un file binario

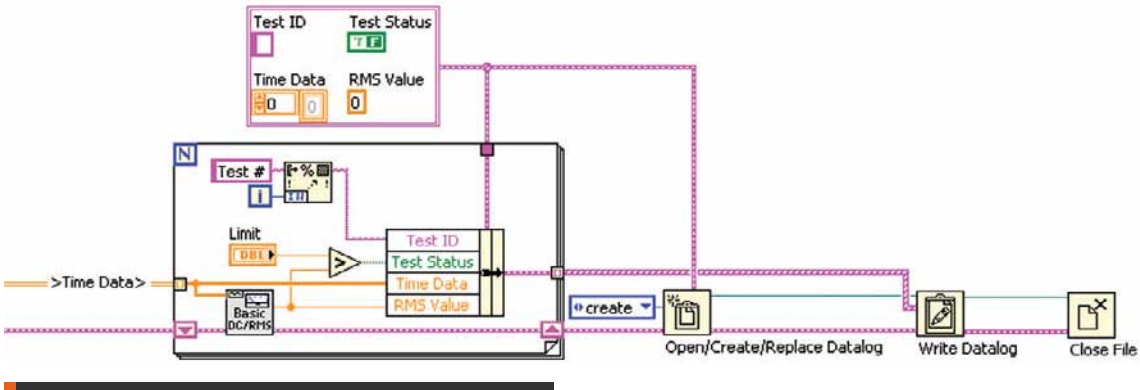

Fig. 6 - Scrittura di un file Datalog

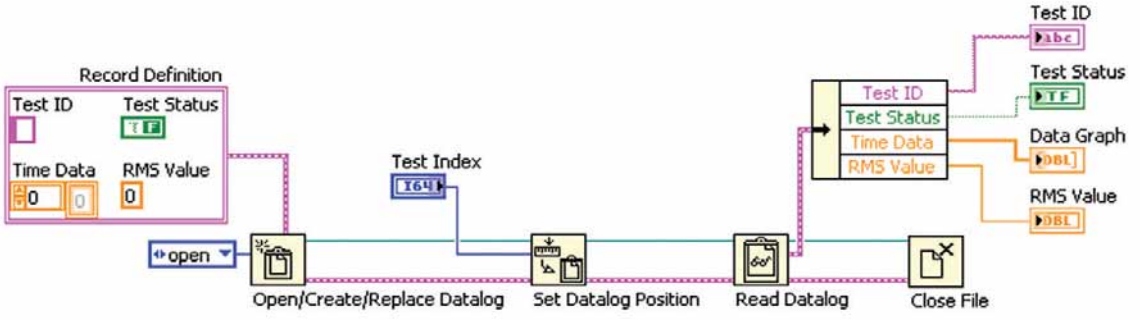

Fig. 7 - Lettura di un file Datalog

file. Nella fig. 5 il VI restituisce le voci dell'array secondo l'indice specificato, assumendo che il file sia stato scritto come array binario di numeri a doppia precisione senza intestazione, come quello scritto per esempio nella fig. 1.

FILE DATALOG

I file Datalog sono progettati per memorizzare un elenco di record su un file. Ogni record è rappresentato da un cluster e può contenere più parti di dati con qualsiasi tipo di dati. I file Datalog sono file binari, tuttavia usano un API diverso da tutti gli altri file binari. I VI Datalog vi consentono di leggere e scrivere array di cluster su file Datalog. Quando aprite un file Datalog per lettura o scrittura dovete specificare il tipo di registrazione usata dal file. Per fare questo collegate un cluster di tipo appropriato al VI Open/Create/Replace Datalog. Una volta aperto il file, potete programmare i file Datalog come ogni altro file binario. L'accesso casuale è disponibile, sebbene gli scostamenti siano specificati in record invece che in byte. La fig. 6 mostra un esempio di scrittura di un file datalog. Notate che il cluster unisce i dati e apre il file Datalog. La fig. 7 mostra un esempio di accesso casuale ad un file datalog. Notate che il cluster Record Definition concorda con il cluster usato per scrivere il file. Se il record type collegato al VI Open/Create/Replace Datalog non concorda con i record del file aperto, si verifica un errore. Invece di usare l'accesso casuale, potete leggere un intero file datalog collegando l'uscita della funzione Get Number of Records al terminale di conteggio della funzione Read Datalog.

readerservice.it n°112

Note sugli autori: Matteo Foini: National Instruments