fram Mathematica
vare di riferimento per la Didattica, la Ricerca e lo Sviluppo

Distributore di Wol
www.ad

Math 13

Costruire GUI interattive

Roberto Cavaliere
Mathematica Technical Sales Manager
Adalta – Software per la Scienza e per il Business

Note:
- Il materiale utilizzato durante il seminario sarà disponibile per il download, vi invieremo il link via email.
- Il documento è un notebook di *Mathematica* e si apre anche con il player Wolfram CDF Player (gratuito) http://www.wolfram.com/cdf-player/
- Per un miglior svolgimento del WebSeminar siete pregati di inviare eventuali domande tecniche e richieste di quotazione a wolfram@adalta.it.
- Per una migliore visione ingrandire lo schermo mediante il pulsante in alto a destra “Schermo intero” (su PC Windows con F11 si può portare il browser a tutto schermo).

Wed 16 Sep 2015 14:54:52

Agenda

Introduzione

- Cenni storici
- Il front end, i notebook e le celle

Personalizzare i notebook

- Presentazioni SlideShow
- I fogli di stile
- Le palette
- La DockedCells

Applicazioni

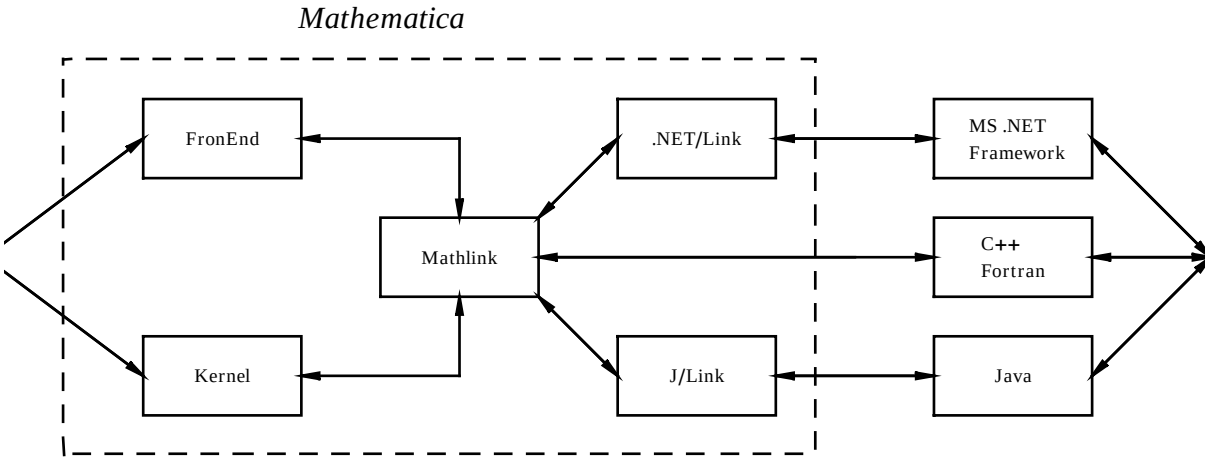
- Elementi di base
- Manipulate & Dynamic

Conclusioni

Introduzione

■ Cenni storici

L'architettura interna di *Mathematica* consiste di due moduli separati: **front end** e **kernel**. I due moduli sono in collegamento attraverso il *MathLink*, una libreria in grado di far dialogare il kernel con diversi altri sistemi.



Il front end rappresenta l'interfaccia utente mentre il kernel è il motore di calcolo. Pur essendo sempre stata una interfaccia di tipo grafico (GUI) il front end per molti anni ha avuto pochi vantaggi in più rispetto ad una tradizionale e sicuramente più spartana interfaccia a riga di comando. Per questo ancora oggi molti utenti sono abituati ad usare *Mathematica* semplicemente scrivendo le espressioni una dopo l'altra nel notebook (il documento di base su cui lavora il front end). Addirittura molti non si sono nemmeno mai spinti al punto di scrivere un package, che semplificando si potrebbe dire che trattasi di raccogliere tutte le righe di comando in un unico file di testo, assegnarli il nome di un package (con l'aggiunta di pochi costrutti) e salvarlo come file .m e non come file .nb. Questo comporta che ancora oggi molti utenti sono convinti che *Mathematica* sia esclusivamente uno strumento di calcolo. In effetti è molto di più, è un ambiente integrato di sviluppo di applicazioni di calcolo, oltre che una potente calcolatrice.

Un primo significativo progresso avvenne con la versione Mathematica 3, che introduceva elementi di grafica avanzata per l'interfaccia, come i bottoni ed i collegamenti ipertestuali (ButtonBox), le pulsantiere (Palettes), i fogli di stile avanzati (Stylesheet), ma soprattutto una ampia gamma di funzioni per la manipolazione dei notebook come espressioni di *Mathematica*.

Con Mathematica 6 si è sono aggiunge caratteristiche di dinamismo e modularità dei componenti di interfaccia che hanno fatto strada ad una nuova serie di applicazioni sviluppabili con *Mathematica*, come testimoniano le oltre 8000 demonstrations scaricabili (comprenso il sorgente) dal sito demonstrations.wolfram.com.

Infine, Mathematica 8 e la tecnologia CDF rilasciata lo scorso maggio insieme al plugin Wolfram CDF Player hanno definitivamente consegnato a *Mathematica* il primato di essere l'unico strumeno di calcolo con una elevata affidabilità in ambito tecnico scientifico che garantisce anche una completa programmabilità delle interfacce al fine di consentire lo sviluppo di applicazioni end-user in qualsiasi contesto scientifico e non, di ricerca, di sviluppo o anche di educazione e formazione.

Introduzione

■ Il front end, i notebook e le celle

Dunque, le componenti che contribuiscono alla creazione e funzionamento di una interfaccia in *Mathematica* sono molteplici e non è facile delineare una mappa precisa di ruoli, funzioni e interazioni tra tali elementi. Cerchiamo di semplificare il tutto elencando una serie di elementi che sicuramente giocano un ruolo importante nella creazione e gestione delle interfacce.
Non sbagliamo se consideriamo come fondamentali i seguenti: il **front end** ed il suo ampio set di istruzioni, i **notebook** e la loro organizzazione in celle, i fogli di stile **stylesheet**, il **kernel** che consente sia di programmare porzioni di interfacce sia di collegare funzionalità di calcolo alle funzioni di interfaccia.

Vediamo adesso, in maniera molto semplice, quali sono le principali relazioni tra questi componenti.

Quando si parla di interfacce in *Mathematica* si parla di notebook, perchè tutte le finestra con le quali noi utenti interagiamo sono dei notebook. Il front end gestisce tutte le parti di un notebook, delle sue celle e degli elementi in esse contenute. Il front end è completamente **personalizzabile** e **programmabile**.

Quando si dice personalizzabile si intende dire che possiamo intervenire manualmente tramite i menu o tramite altri meccanismi per modificare la formattazione o il comportamente di un elemento di front end.

Ad esempio selezioniamo questa cella cliccando sulla sua cell bracket a destra e poi andiamo nel menu Format → Background Color → Light Blue

Quando invece si dice che il front end è programmabile si intende dire che possiamo scrivere del codice che esegue delle impostazioni di front end, come è stato fatto nei due bottoni:

Cambia sfondo al notebook | Annulla modifiche

Nota: il linguaggio del front end non è un linguaggio a se stante, ma rappresenta un'estensione di quello del kernel, dunque la sintassi e tutte le regole di programmazione si applicano tanto alle funzioni del kernel quanto a quelle del front end. Questo è il grande vantaggio della rappresentazione simbolica di *Mathematica*: everything is an expression

Scrivi

Scrivi

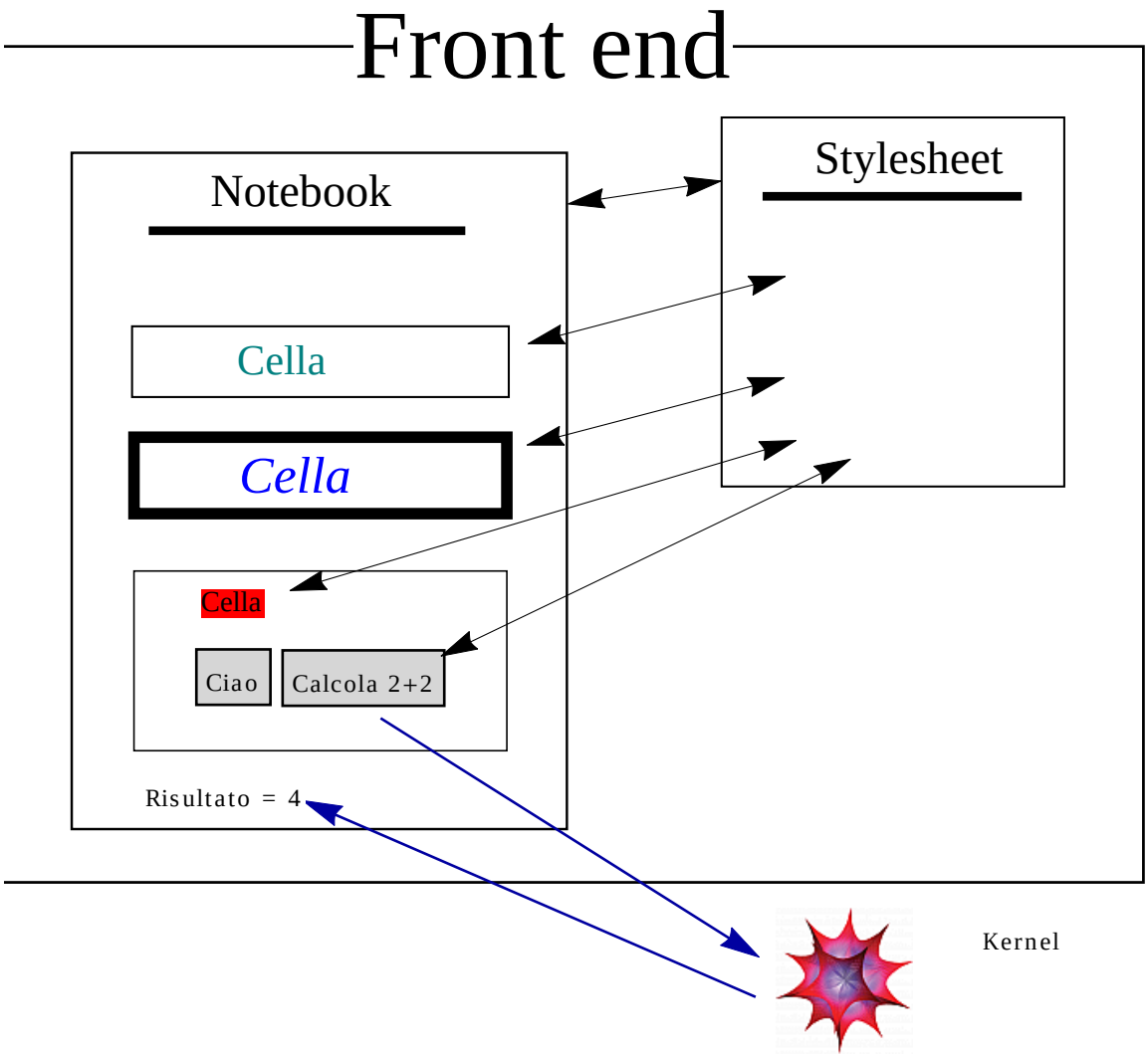
Facendo attenzione alla riga di codice di sopra, notiamo che in effetti la valutazione viene eseguita dal kernel, ma l'effetto (la colorazione dello sfondo della cella) viene gestito dal front end. Questo è un primo esempio di intreccio tra funzionalità e moduli, il front end viene pilotato anche tramite il kernel. E questo è ancora un aspetto della programmabilità del front end. Se vogliamo ottenere un risultato simile (il fondo grigio di una cella), agendo sul versante della personalizzazione possiamo usare ancora un altro modo, tramite i fogli di stile. Questo esempio lo vedremo tra un momento.

Da questa prima introduzione, seppure non siamo andati nel dettaglio tecnico di certe operazioni, dovrebbe già risultare chiaro che *Mathematica* ha una architettura di interfaccia utente molto potente, versatile e programmabile in diversi modi. Questo non deve creare confusione, anzi il fatto che le stesse operazioni possono essere fatte in modo diverso, ma tutte con la stessa logica e soprattutto la stessa sintassi, deve rappresentare un vantaggio: ognuno sceglie il modo più idoneo alla propria conoscenza.

Introduzione

■ Il front end, i notebook e le celle

Cerchiamo di fare un po' di chiarezza. La figura che segue mostra le relazioni (semplificate) tra front end, notebook, celle, stylesheet e kernel



Il front end gestisce la rappresentazione ed il coordinamento di tutti gli elementi a partire dal notebook. Il notebook si basa sul foglio di stile soprattutto per la formattazione (layout) sia del documento nel complesso sia delle singole celle. Il front end mette a disposizione anche una serie di costrutti che ci permettono di creare oggetti di interfaccia, ad esempio i bottoni, che a loro volta possono eseguire codice kernel. I risultati del kernel vengono visualizzati ancora nel notebook e pertanto gestiti dal front end.

Nota: tutti gli elementi delle interfacce di *Mathematica* sono interattivi ed interscambiabili, il che significa che ciascun elemento può essere incluso

all'interno di un altro, come ad esempio il bottone Calcola 2+2 inserito nella figura di sopra, che funziona anche se incassato nell'oggetto grafico.

Le interfacce che vedremo nel seguito di questo documento altro non sono che particolari tipi di notebook, dove alcune proprietà sono definite nello stylesheet, altre "a bordo" del documento, altre ancora a bordo delle celle o ancora degli elementi interni alle celle. Tutte le proprietà, a qualsiasi livello, sono definite tramite il meccanismo delle options, lo stesso impiegato dal kernel per consentire la personalizzazione del funzionamento dei comandi. Un modo semplice per modificare manualmente le opzioni di un notebook è tramite la funzione SetOptions.

```
?SetOptions
```

questa funzione prende in input come primo argomento l'oggetto a cui bisogna impostare l'opzione. Un tipo di oggetto potrebbe essere appunto il notebook. Esistono diverse funzioni che ci restituiscono un oggetto di tipo notebook, per il momento ci interessa

```
?InputNotebook
```

Scrivi

Scrivi

Introduzione

■ Il front end, i notebook e le celle

Proviamo a scoprire il codice interno usato dal front end per rappresentare un oggetto di tipo Cell al quale abbiamo applicato alcune personalizzazioni tramite il menu Format.
Selezionare la cella che segue e scegliere la voce di menu Cell → Show Expression

cella di esempio

si visualizzerà così l'espressione interna della cella (vedi Figura) che mostra un oggetto di tipo Cell che si basa sullo stile "Text" e che include una serie di opzioni che modificano la formattazione di tipo Text. Basta creare una nuova cella di tipo Text e confrontarla con quella di sopra.

Le stesse opzioni possono essere usate dal kernel per qualsiasi altro scopo, ad esempio modificare il testo dell'etichetta dentro un grafico

Scrivi

Dunque, un'altra conferma che il linguaggio *Mathematica* ha una sola logica e tutte le istruzioni seguono la stessa sintassi.

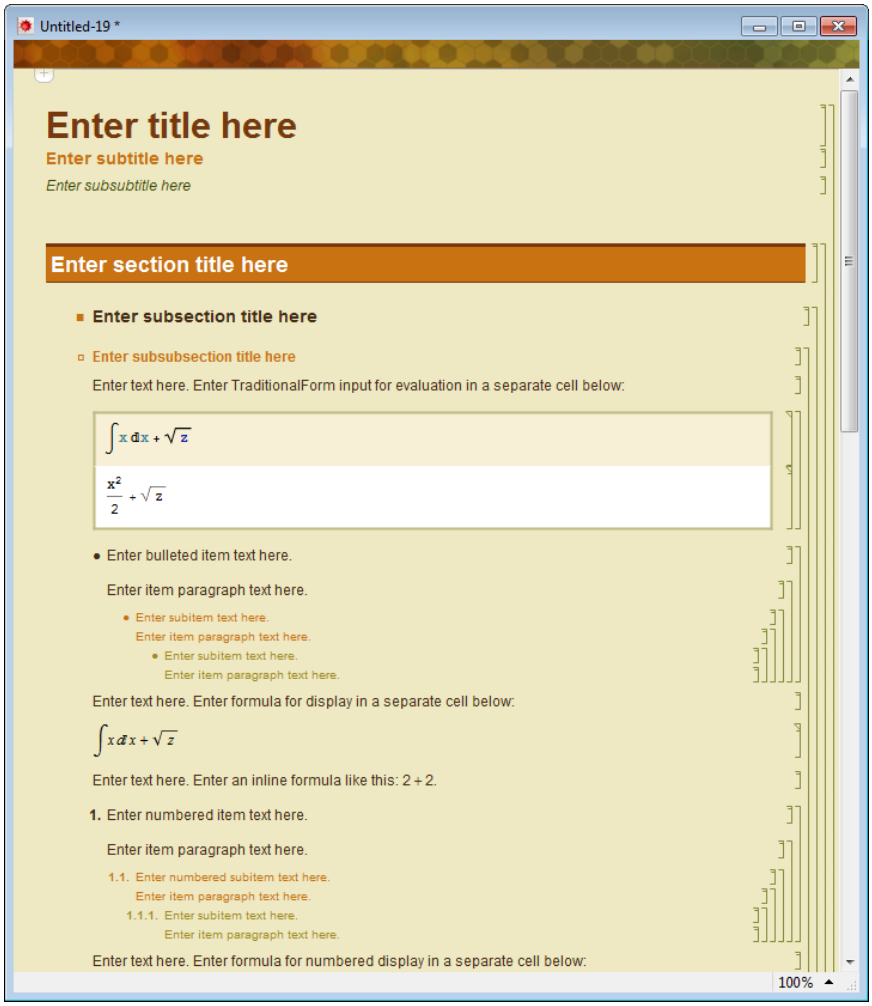
Personalizzare i notebook

I fogli di stile

Abbiamo già introdotto il foglio di stile come quel documento che definisce tutti gli stili (formattazione e comportamento) di un determinato notebook. Vediamo quanche esempio più in dettaglio del loro utilizzo per la creazione di report o parte dell'interfaccia.

Ancora una volta possiamo utilizzare una palette di sistema per operare sui fogli di stile. Dal menu Palettes → Stylesheets apriamo il documento come in Figura

Se muoviamo il mouse su uno qualsiasi dei riquadri notiamo che ciascuno di essi è suddiviso in due parti, una con la nota "Apply" e l'altra con la nota "New". In pratica, una volta scelto quale foglio di stile vogliamo usare, possiamo decidere se creare un nuovo documento o applicare quello stile al documento aperto e selezionato. In questo caso usiamo New ed otteniamo, ad esempio, il seguente nuovo documento



Come si nota, questo notebook ha tutta una serie di personalizzazioni, quali il colore dello sfondo, la DockedCell colorata, i colori delle celle di tipo titolo, testo ed input hanno caratteri molto diversi dal notebook di default che siamo abituati a vedere. Ebbene, tutte queste indicazioni sono scritte nel foglio di stile.
Come fare per modificare una particolare proprietà che non ci piace? Il foglio di stile si può editare e modificare completamente.

Ad esempio, non ci piace lo sfondo della cella di tipo Section. Possiamo modificarlo editando il foglio di stile dal menu Format → Edit Style sheet...

A questo punto si apre un altro notebook, come in Figura che indica le definizioni di tutti gli stili cella di notebook vuoto appena creato. Dal bottone "Choose a style" scegliamo lo stile "Section" e vediamo che compare una cella di definizione già evidenziata. Lasciando tale cella selezionata, andiamo nuovamente in Format → Background Color → Light Gray. Chiudiamo il foglio di stile e notiamo che ora la sezioni (Section) ha il fondo grigio chiaro. Se vogliamo modificare altre formattazioni ripetiamo questa operazione con altri stile di celle o altre proprietà accessibili dal menu Format.

◄ | ►

Personalizzare i notebook

■ Presentazioni SlideShow

Il modo più semplice per realizzare una presentazione SlideShow è fornito dal menu New → Slide Show.
Compare un nuovo documento come in Figura e contemporaneamente si apre una pulsantiere (Palette) con comandi dedicati proprio alla gestione SlideShow (vedi Figura).

Il documento si presenta come una sequenza di celle precompilate di cui alcune stranemente sottili e grige. Il documento si trova nella modalità "scrittura" ossia in un formato che ci consente di editare le singole slide e preparare la nostra presentazione.

Il meccanismo delle slide nel notebook è molto semplice e simile a quello di altri software. In effetti il documento è unico solo che tra una pagina e l'altra bisogna interporre una cella particolare (quella striscia grigia che si vede nel template) che serve a separare le slide. Questo si rende necessario perchè a differenza di altri strumenti di presentazione, in *Mathematica* la pagina non ha una lunghezza prefissata e dunque bisogna mettere uno "slide page break" per indicare quando separare le celle tra due pagine.
Per gestire lo scorrimento delle pagine abbiamo bisogno di una pulsantiera, con il classico tasto avanti, dietro, inizio e fine. Ora interviene un elemento interessante, che vale la pena approfondire perchè utile in molte interfacce: la cella di tipo DockedCell.

Vediamola prima in azione e poi la commentiamo in dettaglio. Se usiamo la palette Slide Show che è comparsa automaticamente quando abbiamo creato il nuovo documento di tipo Slide Show, usiamo il bottone "View Environment" nella sezione Settings ed impostiamolo a SlideShow (facciamo attenzione a che il notebook selezionato sia il template delle slide appena creato). Si noterà subito che questa impostazione modifica la visualizzazione del nuovo documento mostrandolo in modalità presentazione SlideShow (vedi Figura). La DockedCell è quella cella grigia in alto al notebook con una serie di bottoni (vedi Figura). La sua principale caratteristica, da cui prende il nome di "Docked", è che rimane fissa in cima al notebook anche se lo scorriamo verso il basso. Perciò rappresenta una parte fissa dove possiamo inserire tutti i nostri comandi per gestire un documento, in questo esempio le slide.

Il più è fatto! Se vogliamo proseguire nella creazione della presentazione non dobbiamo fare altro che cominciare a scrivere i nostri testi e le righe di codice per i calcoli (se ce ne sono). Se abbiamo bisogno di altre slide le possiamo

aggiungere usando i pulsanti della palette. Infine, possiamo anche creare un indice delle slide, tramite il bottone “Table of Contents from Slide Show” che crea un nuovo notebook con un indici attivo (fatto di bottoni) che ci permette di scorrere le slide in maniera non lineare.

Personalizzare i notebook

■ Le palette

Già nelle precedenti slide abbiamo utilizzato alcune palette native di *Mathematica*, che al momento sembrano degli speciali oggetti di sistema. In realtà, la palette è un notebook con una particolare serie di opzioni che lo rendono sia nell'aspetto che nelle funzioni, ottimizzato per un uso in modalità di pulsantiera. Facciamo una piccola prova.

Scrivi

Una prova che la modalità "Palette" è ancora una volta riconducibile ad una opzione di formato per il notebook

Scrivi

Personalizzare i notebook

■ Le palette

Ovviamente, la palette assume un ruolo significativo soprattutto se la riempiamo con una serie di bottoni che ci propongono delle operazioni frequenti o semplicemente che preferiamo tenere disponibili in qualsiasi momento. I bottoni (Button) sono particolari oggetti molto versatile e potente, alla base della creazione di GUI con *Mathematica*.

La struttura del bottone è semplicissima:

Button["testo", codice]

```
Button["Cliccami", Print["Ciao"]]
```

ovviamente il codice può contenere qualsiasi espressione di *Mathematica*.

Scrivi

Scrivi

Scrivi

Una cosa importante da dire subito è l'importanza dell'opzione **Method** di Button. Poichè il bottone può mandare in esecuzione qualsiasi codice del kernel, il front end potrebbe rimanere inattivo per un tempo indeterminato. Per evitare questo, di default i bottoni hanno l'opzione Method→"Preemptive" che, semplificando, significa che il front end fissa un timeout di 5 secondi oltre il quale disabilita il link tra il bottone ed il kernel. Pertanto se notiamo che il nostro bottone non funziona come dovrebbe, provare sempre a mettere l'opzione Method→"Queued" (nessun timeout).

Ritornando alle palette, proviamo a crearne qualcuna con i bottoni.

Esempio 1

Facciamoci elencare i file che si trovano in due cartella

```
Button["Root Mathematica",
 CreateDocument[FileNames["*", $InstallationDirectory]]]
```

```
Button["Desktop", CreateDocument[FileNames["*",
   FileNameJoin[{$HomeDirectory, "Desktop"}]]]]
```

```
Grid[{{(* copiare qui il primo bottone *)},
  {(* copiare qui il secondo bottone *)}}]
```

```
CreatePalette[];
```

Esempio 2

Usiamo un bottone di salvataggio del file sul quale stiamo lavorando (presumiamo sia InputNotebook)

```
CreatePalette[Button["Salva notebook di lavoro",
   NotebookSave[InputNotebook[]]]];
```

Esempio 3

Un esempio più articolato, per il quale non commentiamo il codice ma ne vediamo solo il risultato, per evidenziare come le palette possono contentere qualsiasi cosa.

times
calendario

Esempio 4

Un problema che spesso si riscontra è quello di voler selezionare tutte le celle di un certo tipo in un notebook per poterle cancellare. Se si tratta di celle di "Output" esiste una voce di menu in Cell → Delete All Output. Ma se vogliamo cancellare tutte le celle "Text" o vogliamo eseguire una particolare operazione, ad esempio convertire tutte le immagini in bitmap così da ridurre le dimensioni del notebook, allora potremmo crearci un nostro bottone da tenere in una palette di servizio.

Creiamo un notebook con una serie di celle ad hoc

```
nb = CreateWindow[DocumentNotebook[
     {CellGroup[{TextCell["Gruppo di celle", "Section"],
        TextCell["Cella di commento", "Text"],
        TextCell["Altra cella di commento", "Text"]}],
      CellGroup[{TextCell["Gruppo di grafici", "Section"],
        Apply[Sequence, Table[ExpressionCell[Plot[Exp[
             -x^n], {x, -3, 3}], "Output"], {n, 5}]]}]}]];
```

selezioniamo le celle che ci interessanto, ad esempio gli output

```
NotebookFind[nb, "Output", All, CellStyle];
```

oppure Text

```
NotebookFind[nb, "Text", All, CellStyle];
```

e poi le cancelliamo

```
NotebookDelete[nb];
```

ora se vogliamo creare una palette per un uso sistematico, non possiamo lavorare sull'oggetto notebook salvato nella variabile, però possiamo usare InputNotebook[]. Inoltre, mettiamo la funzione in un bottone e poi creiamo la palette

```
Button["Cancella output",
 NotebookFind[InputNotebook[], "Output", All, CellStyle];
 NotebookDelete[InputNotebook[]],
 Method → "Queued"]
```

```
CreatePalette[(* copiare qui il bottone *)];
```

ora creiamo un notebook di prova

```
CreateWindow[DocumentNotebook[
   {CellGroup[{TextCell["Gruppo di celle", "Section"],
      TextCell["Cella di commento", "Text"],
      TextCell["Altra cella di commento", "Text"]}],
    CellGroup[{TextCell["Gruppo di grafici", "Section"],
      Apply[Sequence, Table[ExpressionCell[Plot[Exp[
           -x^n], {x, -3, 3}], "Output"], {n, 5}]]}]}]];
```

assicuriamoci di selezionarlo prima di cliccare sul bottone che elimina gli output.

Proviamo a migliorare questa palette "distruttiva" anticipando una richiesta di conferma prima di cancellare effettivamente le celle di tipo Output

```
DialogInput[
 Column[{"Confermi la cancellazione di tutte le
     celle Output del notebook",
   FileNameTake[NotebookFileName[InputNotebook[]]],
   Row[{DefaultButton[DialogReturn[True]],
     CancelButton[DialogReturn[False]]}]}],
 WindowTitle → "Attenzione: operazione non reversibile"]
```

```
Button["Cancella output",
 NotebookFind[InputNotebook[], "Output", All, CellStyle];
 If[DialogInput[
   Column[{"Confermi la cancellazione di tutte le
       celle Output del notebook", FileNameTake[
      NotebookFileName[InputNotebook[]]], Row[
      {DefaultButton[DialogReturn[True]], CancelButton[
        DialogReturn[False]]}]}], WindowTitle →
    "Attenzione: operazione non reversibile"],
  NotebookDelete[InputNotebook[]]],
 Method → "Queued"]
```

```
CreatePalette[(* copiare qui il bottone *)];
```

Ultimo dettaglio: per poter usare NotebookFileName[InputNotebook[]] bisogna prima salvare il notebook altrimenti il file ancora non esiste e dunque non esiste il suo nome.

```
NotebookSave[CreateWindow[DocumentNotebook[
    {CellGroup[{TextCell["Gruppo di celle", "Section"],
       TextCell["Cella di commento", "Text"],
       TextCell["Altra cella di commento", "Text"]}],
     CellGroup[{TextCell["Gruppo di grafici",
        "Section"], Apply[Sequence,
        Table[ExpressionCell[Plot[Exp[-x^n], {x, -3, 3}],
          "Output"], {n, 5}]]}]}]],
  "C:\\tempmath\\test.nb"];
```

Personalizzare i notebook

- La DockedCells

Abbiamo già introdotto questa particolare cella nelle prime slide. Ora che abbiamo visto alcuni esempi di bottoni molto utili, potremmo pensare di collocarne alcuni direttamente nel notebook. Prendiamo per semplicità uno dei bottoni più semplici ma al contempo molto utile.

CreatePalette[Button["Salva notebook di lavoro", NotebookSave[InputNotebook[]]]];

Ora vediamo come poter impostare una cella DockedCells

```
CreateWindow[DockedCells → Cell["XXXX", "DockedCell"]];
```

dunque possiamo scrivere

```
CreateWindow[DockedCells → Cell[
    BoxData[ToBoxes[Button["Salva notebook di lavoro",
       NotebookSave[InputNotebook[]]]]], "DockedCell"]];
```

possiamo anche modificare alcune proprietà della cella DockedCells

```
CreateWindow[DockedCells → Cell[
    BoxData[ToBoxes[Button["Salva notebook di lavoro",
       NotebookSave[InputNotebook[]]]]], "DockedCell",
    Background → LightYellow, TextAlignment → Center,
    CellMargins → {{0, 0}, {100, 0}}]];
```

Esempio 1

Possiamo fare di più. Se vogliamo aggiungere questo bottone a tutti i notebook che creiamo, possiamo creare una palette che crea il bottone nella DockedCell

```
CreatePalette[
  Button["Imposta bottone", SetOptions[
    InputNotebook[], DockedCells → Cell[BoxData[ToBoxes[
        Button["Salva notebook di lavoro", NotebookSave[
          InputNotebook[]]]]], "DockedCell"]]]];
```

Esempio 2

Inserire un motore di ricerca testuale nel proprio notebook

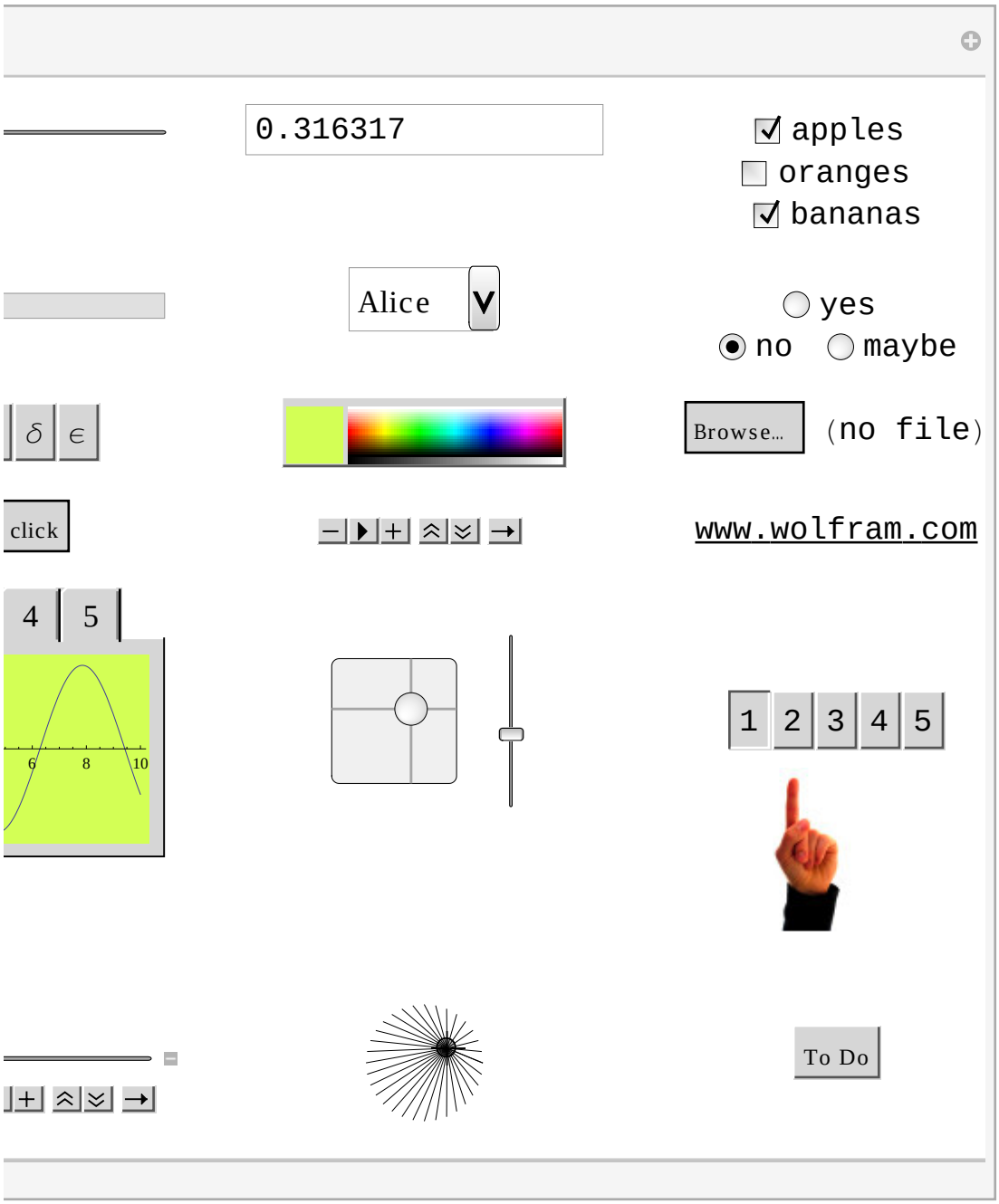
```
cerca = DynamicModule[{testo = ""},
   Row[{InputField[Dynamic[testo], String],
     Button["Successivo",
      NotebookFind[ButtonNotebook[], testo]],
     Button["Precedente", NotebookFind[
       ButtonNotebook[], testo, Previous]]
    }]];
nb = CreateDocument[{}, DockedCells →
    {Cell[BoxData[ToBoxes[cerca]], "DockedCell"]}];
```

◄ | ►

Applicazioni

Elementi di base

Le interfacce si possono costruire grazie alla disponibilità di decine e decine di controlli flessibili e potenti. Eccone alcuni



Esempio

I controlli possono essere interdipendenti e nidificati tra loro

```
Manipulate[Grid[Take[data, h, w]],
 {{data, RandomInteger[{0, 1}, {20, 30}]},
  ControlType → None},
 {{h, 5}, 1, 20, 1},
 {{w, 10}, 1, 30, 1},
 Panel[
  Dynamic[Grid[Table[With[{i = i, j = j}, Checkbox[Dynamic[
        data[[i, j]]], {0, 1}]], {i, h}, {j, w}]]]]]
```

◄ | ►

Applicazioni

- Manipulate & Dynamic

Mathematica 6 ha introdotto un nuovo set di comandi, come Manipulate e Dynamic ad esempio, che hanno ampliato il concetto di interfaccia grafica consentendo di realizzare applicazioni con moduli interattivi e dinamici. Interattivi perchè consentono all'utente di intervenire tramite una serie di controlli, dinamici perchè le conseguenze di questi interventi, ad esempio il ricalcolo di un certo algoritmo, avvengono dinamicamente, ossia in tempo reale.

In termini di interfacce per applicazioni, anche complesse, tramite la combinazione di Dynamic, Manipulate e controller vari si può realizzare praticamente qualsiasi cosa.

Esempio 1

Gestione dei controller tramite mouse o altri dispositivi esterni, quali gamepad

```
CreatePalette[Dynamic[CurrentImage[]]];
```

ora al posto di Manipulate uso ControllerManipulate, analogo della Manipulate ma già predisposto per funzionare con controller esterni di qualsiasi tipo e non mostra i controlli direttamente nell'applicazione

```
ControllerManipulate[Plot[{Sin[x + a] + b, Sinc[x + c] + d},
  {x, -2 Pi, 2 Pi}, PlotRange → 2],
 {a, 0, 10}, {b, 0, 1}, {c, 0, 10}, {d, 0, 1}]
```

```
ControllerInformation[]
```

Anche Manipulate intercetta i controller

```
Manipulate[Graphics[
  {If[c, Red, Blue], First[Plot[a * Sin[b * x], {x, 0, 2 Pi},
     PlotRange → {-2, 2}, PlotStyle → Thick]]}],
 {a, 1, 3}, {b, 1, 4}, {c, {True, False}}]
```

Esempio 2

Un esempio creato dalla Wolfram che illustra la completezza delle funzionalità di *Mathematica* nella realizzazione di interfacce interattive.

esempio

Esempio 3

Un'applicazione ancora "work in progress"

esempio

Curiosità

Vediamo come sono stati realizzati alcuni elementi di questo notebook.

Il bottone Scrivi

Scrivi

ci sono tanti modi per creare questi bottoni, io uso il seguente. Creo la cella con il codice input che voglio far scrivere

```
Plot[Cos[x], {x, -2, 2}, Background → LightGray]
```

poi faccio Cell → Show Expression e vedo il suo codice interno

```
Cell[BoxData[
 RowBox[{"Plot", "[",
  RowBox[{
   RowBox[{"Cos", "[", "x", "]"}], ",",
   RowBox[{"{",
    RowBox[{"x", ",",
     RowBox[{"-", "2"}], ",", "2"}], "}"}], ",",
   RowBox[{"Background", "→", "LightGray"}]}], "]"}]], "Input"]
```

lo copio e lo metto dentro una CellPrint che si trova dentro una Button

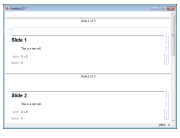
```
Button["Scrivi",
 CellPrint[Cell[BoxData[RowBox[{"Plot", "[",
       RowBox[{RowBox[{"Cos", "[", "x", "]"}],
         ",", RowBox[{"{", RowBox[{"x", ",",
             RowBox[{"-", "2"}], ",", "2"}], "}"}], ",",
         RowBox[{"Background", "→", "LightGray"}]}],
       "]"}]], "Input"]]]
```

Scrivi

Il bottone Figura

Figura

questo è molto più semplice, si tratta di un PopupWindow

```
PopupWindow[Panel["Figura", FrameMargins → 0,
  BaselinePosition → Axis],  , WindowSize → FitAll]
```

◄ | ►

Conclusioni

Creare interfacce utente avanzate è fondamentale per arricchire le applicazioni di calcolo con strumenti di controllo e gestione atti a semplificare l'utilizzo delle applicazioni stesse. Il chiaro vantaggio è che l'utente finale, sia che siamo noi stessi o nostri utenti/clienti, non vede più il codice *Mathematica*, non è costretto a conoscerlo o impararlo, non deve più mandare in esecuzione manualmente il codice prima di poterne vedere l'effetto. Tramite semplici maschere (notebook) e strumenti di controllo (bottoni, slider, ecc.) può accedere a tutte le funzionalità dell'applicazione senza scendere nel dettaglio tecnico di implementazione del codice *Mathematica*.