Programování v systému Maple

autor: **Ing. Vladimír Žák**

email: zakyn@centrum.cz

web: www.vladimirzak.com/maple

recenzent: **Prof. RNDr. Jiří Hřebíček, CSc.**
Institut biostatistiky a analýz
Masarykova universita
hrebicek@iba.muni.cz

restart

Obsah kurzu:

Vzdělávací akce je určena uživatelům systému Maple, kteří chtějí získat podrobnější znalosti o programování v systému Maple. Cílem kurzu je naučit se vytvářet vlastní procedury, Maplety a interaktivní dokumenty v systému Maple. Účastníci se na závěr kurzu seznámí s technologií OpenMaple a možnostmi jejího využití a dále se systémem MapleNET a Maple T.A. Budou představeny i možnosti některých toolboxů dodávaných k systému Maple.

Javovské prostředí

- pokročilé techniky práce
- tvorba interaktivních dokumentů

Programování

- základy programování
- pokročilé techniky v procedurálním programování v Maple

Tvorba interaktivních dokumentů - užití vložených komponent

- užití vložených komponent

Vytváření Mapletů

- ručně
- s využitím MapletBuilder

Seznámení s technologií OpenMaple

- přednáška

Základy práce v MapleNET

- Možnosti MapleNET
- Základní použití

Ukázky některých Toolboxů systému Maple

- Maple Toolbox for Matlab
- apod.

Základy práce v Maple T.A.

- Možnosti MapleTA
- Základní použití

Standardní prostředí - javovské prostředí

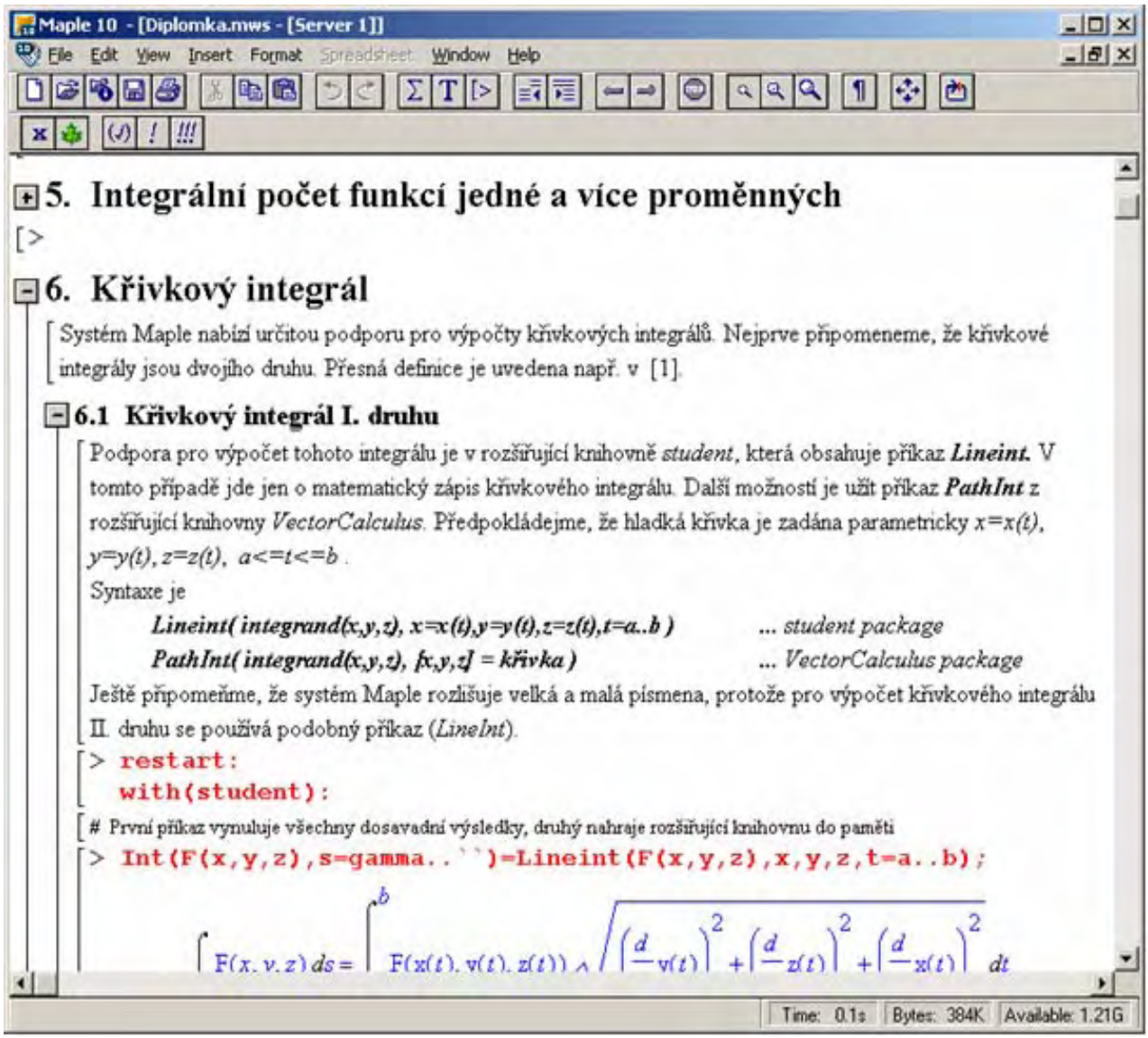
Úvod

Počítačový systém Maple je jednou z možných informačních a komunikačních technologií (ICT), kterou lze velmi efektivně zapojit jak do výzkumu, tak i do výukového procesu. Poskytuje nepřeberné množství funkcí pro vysvětlení základních i náročnějších matematických pojmů a velmi intuitivní formou poskytuje široké možnosti symbolických i numerických výpočtů včetně aplikací v superpočítačovém centrum Masarykovy university.

Obsahuje již i pokročilé nástroje pro tvorbu a vývoj grafických uživatelských rozhraní v rámci systému Maple, kdy uživatel již nemusí znát téměř žádné příkazy systému a vystačí si třeba jen s kontextovou nápovědou. V následujících odstavcích se zaměříme na klíčové vlastnosti systému Maple, vyzdvihneme jeho výhody a zmíníme se i o nevýhodách.

Prostředí Maple

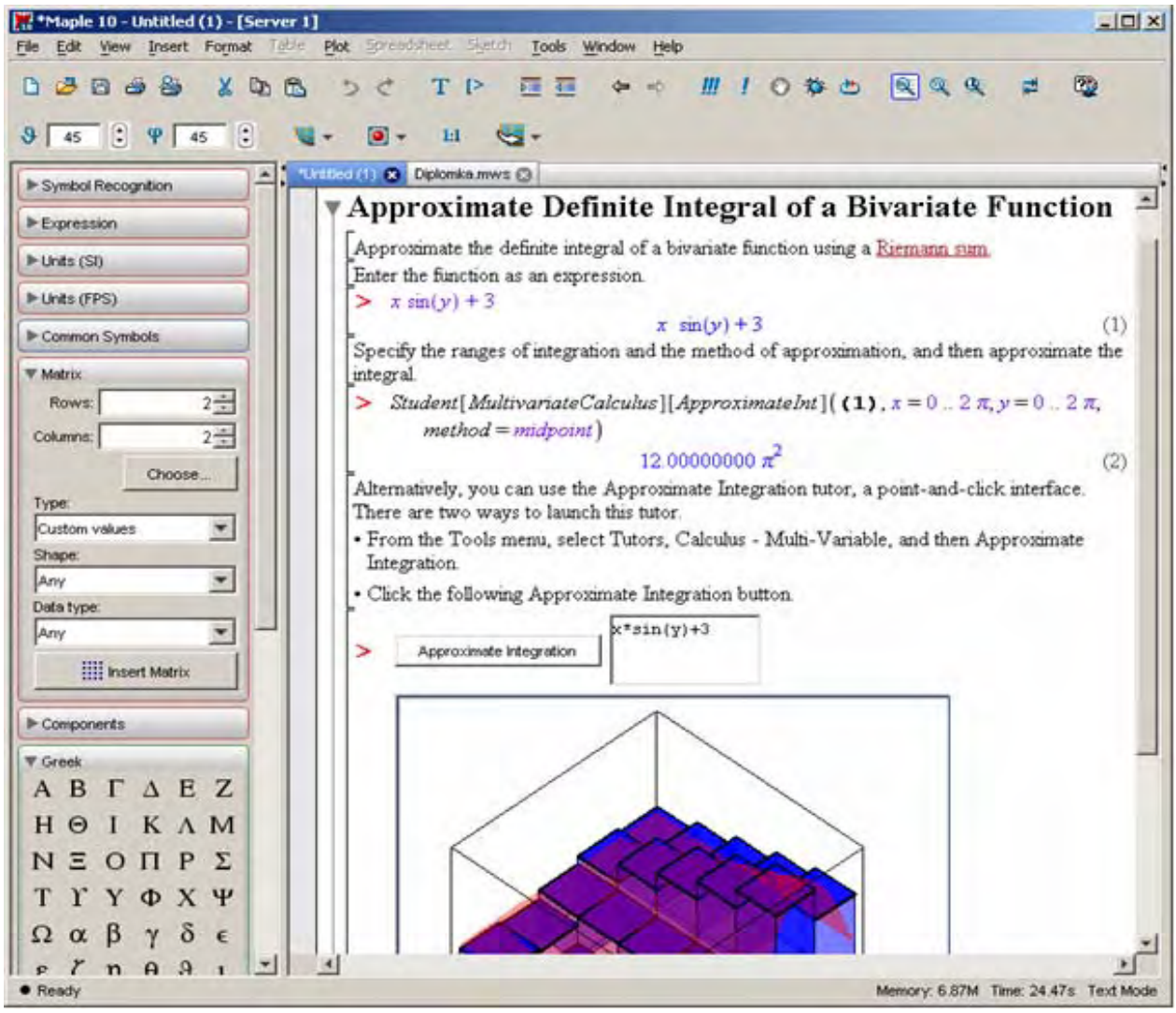
Používání systému Maple je založeno na **standardním grafickém rozhraní** (GUI) a **klasických zápisnících**. Toto grafické rozhraní bylo velmi dobře použitelné a přehledné, nicméně vyžadovalo poměrně hlubokou znalost programovacího jazyka systému Maple. Typický dokument, který byl napsán v tomto uživatelském grafickém rozhraní, je zobrazen na Obr. 1. Více informací lze nalézt na http://www.vladimirzak.com/maple/zakladyprace/popisprostredi.html.



Obr. 1 - dokument v klasickém grafickém rozhraní systému Maple do verze 9.5

Systém **Maple 9** přinesl před několika lety **dvě různá grafická uživatelská rozhraní**. Zůstalo již dříve užívané grafické rozhraní, které se nyní nazývá „**Classic Worksheet**“, a nově bylo **vytvořeno rozhraní napsané v jazyce Java**, které je má v budoucnu zcela nahradit. Toto nové rozhraní odráží jak připomínky uživatelů vzhledem k dřívějšímu GUI, které bylo někdy velmi těžkopádné, tak i současné trendy komunikace s uživateli.

Nové uživatelské grafické rozhraní (viz. Obr. 2) se velmi rychle začalo vyvíjet a přinášelo další a další zlepšení prospěšné pro uživatele Maple. Bohužel však bylo značně pomalé z důvodu použití jazyka Java. Toto rozhraní bylo výhodné používat jen na počítačích s rychlejšími procesory. Od verze Maple 10.03 se toto rozhraní podařilo velmi vylepšit a komunikaci zrychlit.



Obr. 2 – nové uživatelské rozhraní systému Maple od verze Maple 9

Změny ve verzi Maple10

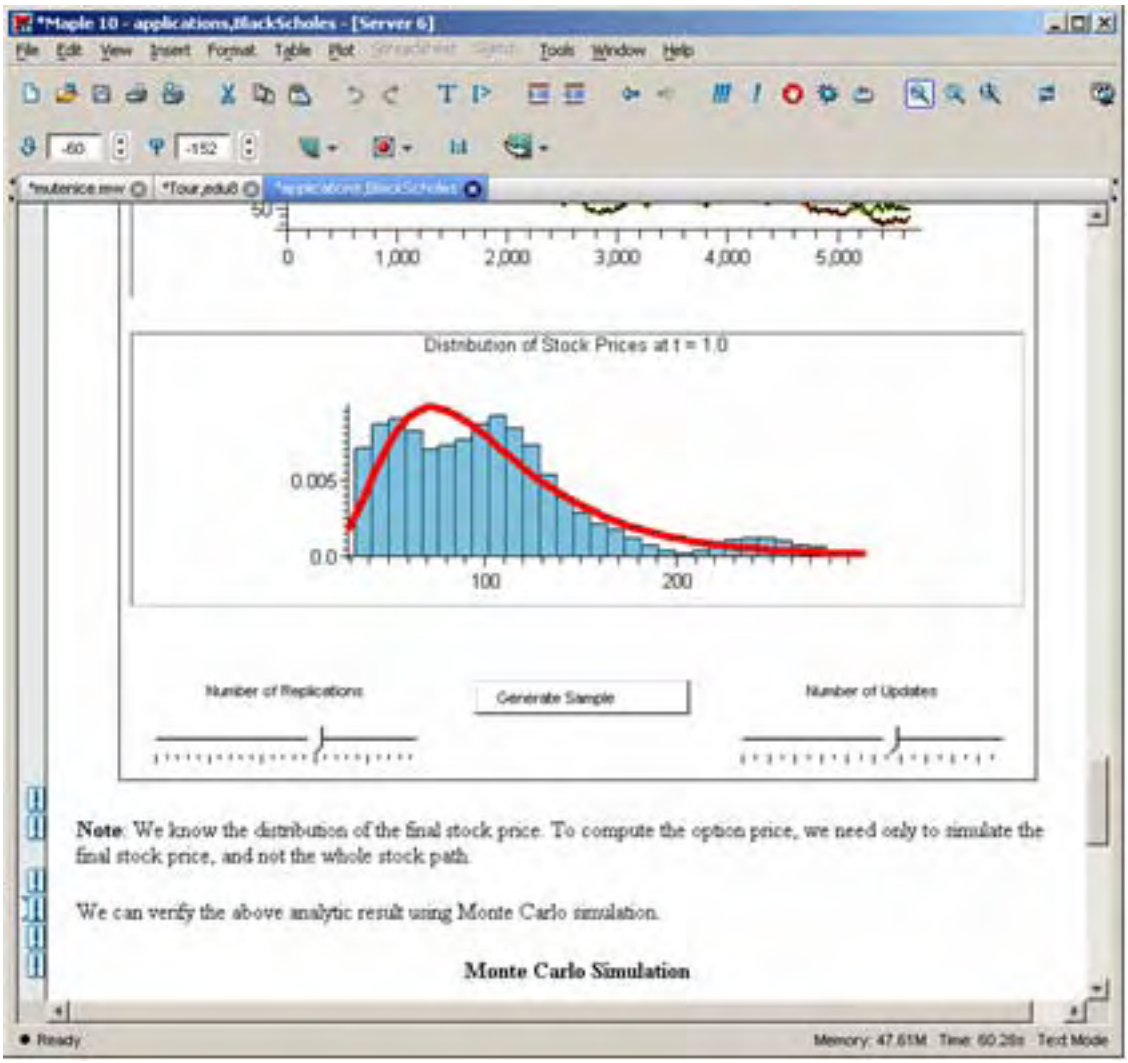
Nová verze systému **Maple 10** přinesla **revoluční změnu systému dokumentů**. Tento posun s sebou přinesl změnu celé filozofie používání systému Maple jako celku. Jeho předchozí použití bylo založeno na znalosti programovacího jazyka systému Maple, který sloužil k provádění matematických výpočtů. Uživatel se musel naučit základní datové struktury Maple, práci s nimi a dále byl nucen učit se velké množství příkazů pro jednotlivé operace. To však bylo v nové verzi systému Maple 10 opuštěno ve prospěch „pohodlí“ uživatele, který využívá kontextové nabídky kliknutím na pravé tlačítko myši.

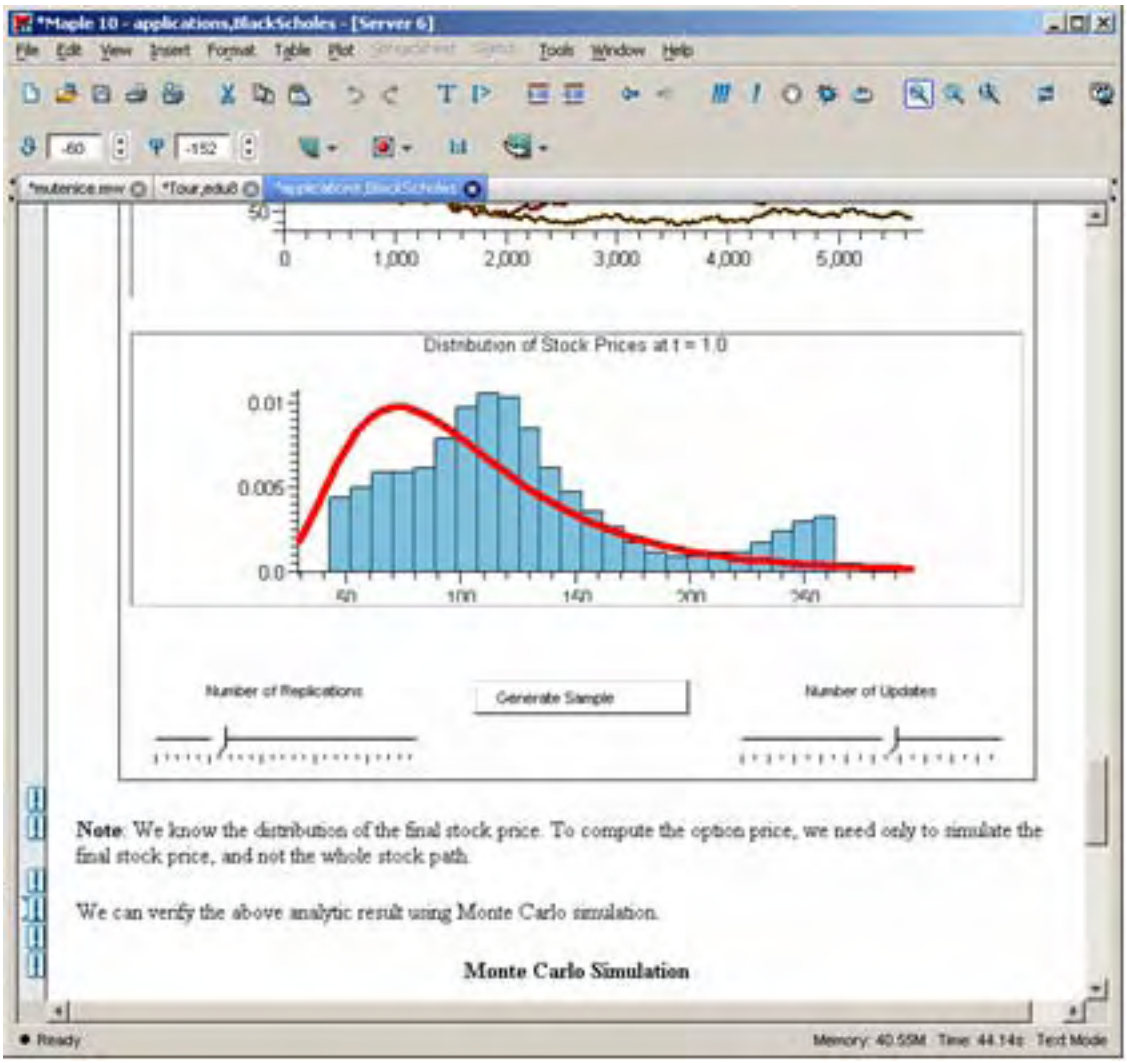
Byl vytvořen zcela nový typ dokumentu, který je označován jako „**Rich Technical Dokument** (RTD)“, který umožňuje **pracovat** se systémem Maple velmi intuitivně a interaktivně **pomocí kontextové nabídky**. Uživatel je schopen jen s pomocí základních počítačových dovedností vytvářet plně interaktivní a komplexní dokumenty, které mohou sloužit dokonce jako výstup technických aplikací, popř. jako dokumentace k řešenému matematickému problému. Tyto vlastnosti nelze nalézt v žádném obdobném CAS (Mathematica, MathCad, Derive, atd.). Maple je tedy v tomto ohledu zcela revoluční a jeho inovace přináší zlepšení možností využití celého systému nejen pro pokročilé uživatele, ale zejména pro začínající studenty.

Je nutné ještě poznamenat, že tradiční zápisník v Maple je uživatelům stále k dispozici, ale neumožňuje jednoduše vytvářet plně interaktivní dokumenty.

Nejdůležitější novinky v systému Maple 10

Zásadní inovací v systému Maple 10 je již zmíněný nový typ dokumentu, formátu **RTD**, který umožňuje vytvářet plně interaktivní dokumenty (viz. Obr. 3), které jsou zcela nové nejen svým obsahem, ale i formátováním a interaktivními komponentami.


*Maple 10 - applications,BlackScholes - [Server 6]
File Edit View Insert Format Table Plot Tools Window Help
50
0
1,000
2,000
3,000
4,000
5,000
Distribution of Stock Prices at t = 1.0
0.005
0.0
100
200
Number of Replications
Generate Sample
Number of Updates
Note: We know the distribution of the final stock price. To compute the option price, we need only to simulate the final stock price, and not the whole stock path.
We can verify the above analytic result using Monte Carlo simulation.
Monte Carlo Simulation
Ready
Memory: 47.61M Time: 60.28s Text Mode



Obr. 3 : Interaktivní dokument

Další předností tohoto typu dokumentu je možnost použití systému Maple bez znalosti jeho příkazů s využitím:

- palety nástrojů,
- šablon běžných problémů (tzv. „task templates“), - více informací
- rozšířené kontextové nabídky,
- nástrojů pro rozpoznávání znaků,
- interaktivních průvodců umožňujících např. import dat a jejich analýzu, atd.

Pro tvorbu **„mapletů“** (viz http://www.fi.muni.cz/~hrebicek/maple) je k dispozici tzv. **Maplet builder**. Maplety jsou velmi vhodné pro vytvoření grafického rozhraní pro řešení a lepšího pochopení některých matematických úloh. Jsou hojně využívány v nových rozšiřujících knihovnách Maplu.

Více informacích o novinkách Maple 10

Základy programování

Pro vytváření vlastních procedur v systému Maple je vytvořen speciální jazyk. Tato kapitola není úplným referenčním textem, je jen základem pro pochopení struktur vlastních procedur. Při programování se velmi často používají cykly a podmínky, a proto i systém Maple tyto řídící struktury obsahuje.

Zopakujme syntaxi pro vytvoření vlastní funkce (procedury). Struktura vypadá následovně

název_fce := **proc**(*param1*,*param2*,…)
    tělo funkce
end proc;

Nejprve zvolíme název funkce a pak pomocí klíčového slova proc vytvoříme funkci s parametry, které jsou uzavřeny do kulatých závorek. Dále následuje tělo funkce , ve kterém se většinou vyhodnotí zadané parametry a jejich typy, provedou se příslušné příkazy a konstrukce se ukončí pomocí end proc .

Podmínky - if - else -end if a if - elif - else - end if

Nyní se podívejme na řízení toku programu pomocí struktury (podmínky) ***if-then-else-end if*** . Přesná syntaxe je

if *podmínka1* **then** *příkazy1* **else** *příkazy2* **end if (fi)**

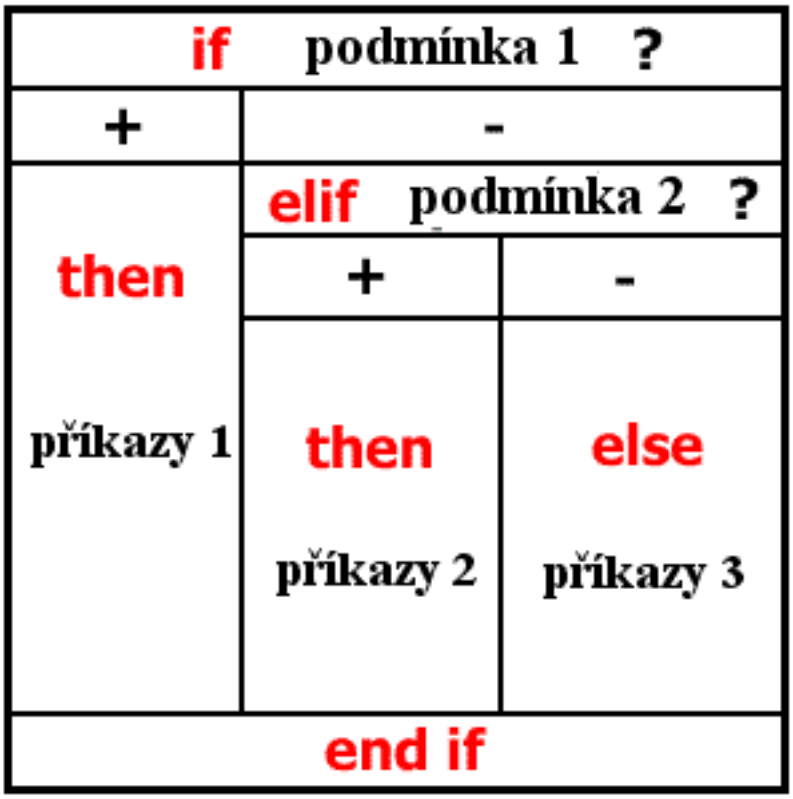
Následující obrázek znázornuje situaci

Je-li podmínka za if splněna (tj. +), jsou provedeny příkazy uvedené za then . Pokud splněna není (tj. -), provedou se příkazy uvedené za else . V obou případech se hned po provedení příkazů pokračuje v programu.

Existuje ještě jiná možnost zápisu podmínky s ***if*** , a to

if *podmínka1* **then** *příkazy1* **elif** *podmínka2* **then** *příkazy2* **else** *příkazy3* **end if**

Následující obrázek ukazuje, jak tato struktura pracuje.



Nyní si na příkladu hledání maxima ze dvou zadaných čísel ukažme použití těchto dvou struktur. Z pedagogických důvodů budeme požadovat i informaci o případné rovnosti čísel. Příklad později rozšíříme na hledání maxima z libovolného počtu zadaných čísel, a to pomocí cyklů typu ***for-do*** nebo ***while-do*** .

```
> max1:=proc(a,b)      # užití if-else-end if
    if a<>b then
      if a>b then
        return a;
      else
        return b;
      end if;
    else
      return a=b;
    end if;
  end proc:
>
```

Volání výše uvedené procedury je např. nasledující: (zkuste použít nápovědu výrazů - pomocí kláves CTRL + mezerník)
max1(−5, 6);

6 (3.1.1)

max1(5,−6)

5 (3.1.2)

max1(−6,−6)

$$-6 = -6 \tag{3.1.3}$$

Výše uvedenou proceduru lze samozřejmě zapsat i bez použití ***else*** , ale jako ukázka je vhodnější výše uvedený kód. Nyní s výhodou užijeme struktury s ***elif*** .

```
> max2:=proc(a,b)
  # užití if-elif-else-end fi
    if a>b then
      return a;
    elif a<b then
      return b;
    else
       return a=b;
    fi;    # alternativa k end if
  end proc:
>
```

Volání výše uvedené procedury je např. nasledující:
max2(−5, 6);

$$6 \tag{3.1.4}$$

max2(5,−6)

$$5 \tag{3.1.5}$$

max2(−6,−6)

$$-6 = -6 \tag{3.1.6}$$

V předchozím zdrojovém kódu si hlavně všimněte předposledního řádku, kde je uvedena alternativa ukončení k ***end if*** , tj. ***fi*** .

Nyní přistupme k dalším strukturám, půjde o cykly.

Cyklus while-do-end do

Začneme s cyklem ***while-do-end do*** . Přesnější syntaxe je

while *podmínka* **do** *příkazy* **end do**

Jde o cyklus s podmínkou, je-li pravdivá, jsou provedeny příkazy za klíčovým slovem ***do*** . Následující obrázek ukazuje, jak tento cyklus pracuje.

Vraťme se k proceduře hledání maxima. Nyní ji změníme pro libovolný počet zadaných parametrů s využitím cyklu ***while-do*** .

```
> max3:=proc()       # podmínka while-do-end do
    local i,n,max;
  # definice lokálních proměnných
    n:=nargs;
  # počet vložených parametrů do fce max3
    max:=args[1];
  # jako maximum nastavíme 1. parametr
    i:=2;
  # začneme porovnávat až od druhého parametru
    while i<=n do
      if args[i]>max then
        max:=args[i];
  # je-li parametr větší než max, přepíšeme max
      end if;
      i:=i+1;
  # přejdeme k následujícímu parametru
    end do;
    return max;
  # vypíšeme maximum
```

```
end proc:
```

Volání výše uvedené procedury je např. nasledující:

max3(5,−6)

5 (3.2.1)

max3(1)

1 (3.2.2)

max3(0,−65, 76);

76 (3.2.3)

max3(0,−6, 76,−98.25, 78, 78);

78 (3.2.4)

Přistupme k vysvětlení předchozí procedury. Pokud potřebujeme nějaké vlastní (lokální) proměnné, které budou pouze v těle procedury, je nutné užít klíčového slova ***local***. V případě, že tyto proměnné nejsou takto předem deklarovány, Maple ohlásí varování.

Podíváme-li se znovu na předchozí zdrojový kód, zjistíme, že nejsou zadány žádné parametry. To je ale omyl. Procedura počítá minimálně s jedním, a to při inicializaci proměnné ***max***. K zadaným parametrům se v Maplu přistupuje pomocí pole ***args[i]*** a celkový počet zadaných parametrů je obsažen v ***nargs***.

Vraťme se k předchozímu kódu. Dále následuje inicializace maxima prvním zadaným parametrem a pak pomocí cyklu ***while-do*** procházíme a porovnáváme zbylé parametry s hodnotou uloženou v ***max*** . Nakonec je maximum vypsáno.

Cyklus for

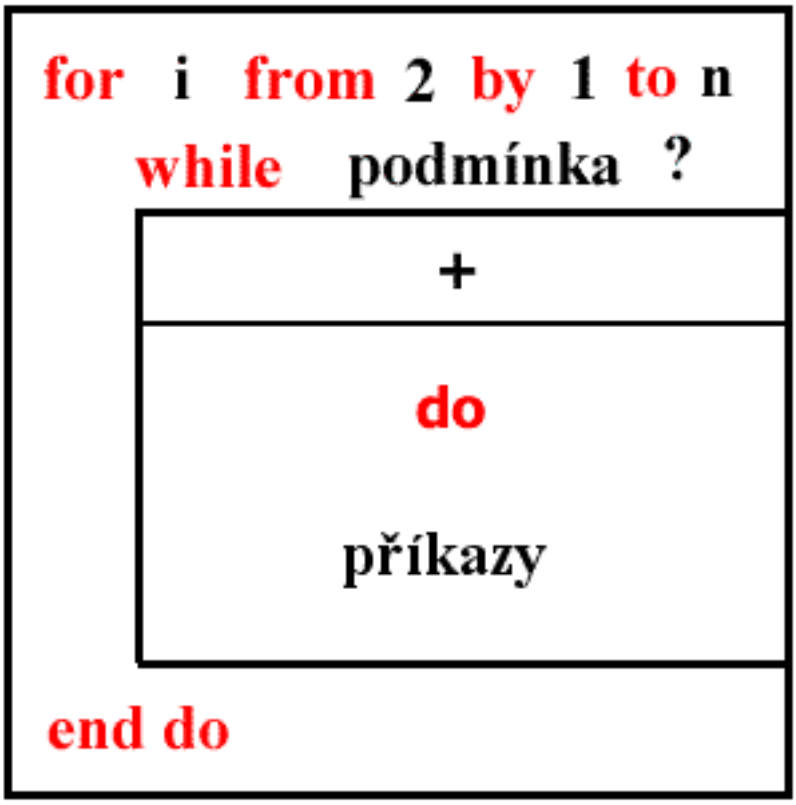
Pokud znáte jiný programovací jazyk, např. jazyk C, pravděpodobně byste v tomto případě použili jiný cyklus a to cyklus **for** . I zde je to možné, protože i Maple nabízí tento často užívaný cyklus. Podívejme se blíže na jeho dvě možné syntaxe.

for *proměnná* **from** *odkud* **by** *krok* **to** *kam* **while** *podmínka* **do** *příkazy* **end do**

nebo

for *proměnná* **in** *struktura* **while** *podmínka* **do** *příkazy* **end do**

V prvním příkazu se proměnná inicializuje hodnotou uvedenou za klíčovým slovem from , zvyšovat se bude o hodnotu kroku dokud nedosáhne hodnoty kam . Dále se vyhodnotí podmínka příkazu while a pokud je splněna, jsou provedeny příkazy . Následující obrázek vše vysvětlí lépe.



Ukažme si to na dalším možném zápisu naší známé procedury hledání maxima.

```
> max4:=proc()
    local i,max;
  # definice lokálních proměnných
    max:=args[1];
  # jako maximum nastavíme 1. parametr
    for i from 2 to nargs do
  # začneme porovnávat až od druhého parametru
       if args[i]>max then
  # je-li parametr větší než max, přepíšeme max
         max:=args[i];
       end if;
    end do;
    return max;      # vypíšeme maximum
  end proc:
>
```

Volání funkce.
max4(5,–6)

Následující volání způsobí chybu, protože se snaží přistupovat k prvnímu prvku pole argumentů, ale tento prvek neexistuje

```
max4( )
Error, (in max4) invalid subscript selector
```

```
max4(0,–65, 76);
```

$$76 \tag{3.3.2}$$

```
max4(0,–6, 76,–98.25, 78, 78);
```

$$78 \tag{3.3.3}$$

Jen pri uziti is(..)

```
max4(π, 5);
```

$$5 \tag{3.3.4}$$

```
max4(a, 5)
```

$$a \tag{3.3.5}$$

```
max4(5, a)
```

$$5 \tag{3.3.6}$$

Nyní se podívejme na cyklus **for-in-do** . Mějme seznam čísel, které chceme postupně sčítat, dokud nedosáhneme určité hodnoty.

```
cisla := [1, 4, 6, 7, 12, 15, 20]
```

$$[1, 4, 6, 7, 12, 15, 20] \tag{3.3.7}$$

```
> cisla1 := [1, 2];
```

$$cisla := [1, 2] \tag{3.3.8}$$

```
mmax := 0;
```

$$0 \tag{3.3.9}$$

```
for z in cisla while mmax ≤ 30 do
  mmax := mmax + z
end do
```

$$3 \tag{3.3.10}$$

```
> for z in cisla while mmax ≤ 30 do
    mmax := mmax + z;
  end do;
  mmax;
```

$$45 \tag{3.3.11}$$

```
>
```

Výše uvedený zdrojový kód je snáze čitelný. Všimněme si, že místo proměnné max je užita proměnná mmax , a to proto, že max je klíčovým slovem systému Maple. V předchozích zdrojových kódech mohla být proměnná max používána bez nebezpečí, protože byla deklarována jako lokální, pomocí local .

Pokročilé techniky v procedurálním programování v Maple

Klíčová slova a operátory

Systém Maple má podobně jako jakékoliv programovací systémy určitá slova, která v systému mají nějaký význam. Mluvíme zde o tzv. klíčových slovech.
Seznam klíčových slov lze nalézt v nápovědě pomocí
?*keywords*
Tato klíčová slova jsou chráněná a nelze je tedy užít v jiném významu.

Podobně je to i s operátory. Maple má operátory unární a binární.
?*operators*[*unary*]
?*operators*[*binary*]
?*operators*[*nullary*] (reeprezentuje jen %,%%,%%%)

Identifikátory a jména proměnných

Maple rozlišuje velká a malá písmena. Je tedy case-sensitive. Jména nesmějí začínat číslem. Identifikátor, který začíná podtržítkem je většinou identifikátor systému Maple s určitým významem. Proto nedoporučuji začínat vlastní identifikátory podtržítkem.

```
> m_name;
```

m_name (4.2.1)

```
> whattype(%);
```

symbol (4.2.2)

```
> 1m_nam
```

Identifikátory lze vytvořit i pomocí více slov. Udělá se to následovně

```
> `Toto je identifikator`;
```

Toto je identifikator (4.2.3)

```
> whattype(%);
```

symbol (4.2.4)

Maple má také identifikátory, které reprezentují určitá nastavení systému. Více nalezente v nápovědě.
?*ininames*
?*anames*
Jde např. o proměnnou Digits, která reprezentuje počet platných cifer
Digits

10 (4.2.5)

evalf(π)

$$3.141592654 \tag{4.2.6}$$

$Digits := 100$

$$100 \tag{4.2.7}$$

$evalf(\pi)$

$$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706\ 8 \tag{4.2.8}$$

$Digits := 10$

$$10 \tag{4.2.9}$$

$evalf(\pi)$

$$3.141592654 \tag{4.2.10}$$

$evalf(\pi, 100)$

$$3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706\ 8 \tag{4.2.11}$$

Poznámky:

$solve\left(\sin(x) = \frac{1}{2}\right)$

$$\frac{1}{6}\,\pi \tag{4.2.12}$$

Zobrazeni vsech reseni

$_EnvAllSolutions := true;$

$$true \tag{4.2.13}$$

$solve\left(\sin(x) = \frac{1}{2}\right)$

$$\frac{1}{6}\,\pi + \frac{2}{3}\,\pi\,\_B11\sim + 2\,\pi\,\_Z11\sim \tag{4.2.14}$$

Seznam identifikatoru v predchozim

$indets(\%, name)$

$$\{\pi, \_B11\sim, \_Z11\sim\} \tag{4.2.15}$$

```
> about(%)
{Pi, _Z11, _B11}:
  is used in the following assumed objects
  [_Z11] assumed integer
  [_B11] assumed OrProp(0,1)
  [Pi] assumed Pi
```

$_EnvAllSolutions := false;$

$$false \tag{4.2.16}$$

Zobrazení kódu procedury

K zobrazení kódu procedur je možné použít následující konstrukci.

```
interface(verboseproc = 2)
```

1 (4.3.1)

```
print(int)
```

```
proc( )                                                                    (4.3.2)
    option Copyright (c) 1997 Waterloo Maple Inc. All rights reserved.;
    local answer;
    if nargs = 2
     and type(args[2], = ('name', '{complex(float)..algebraic,
    algebraic..complex(float)}')) then
        try
            answer := evalf('int'(args));
            if type(answer, 'complex(float)') then
                return answer
            end if
        catch:

        end try
    end if;

    _EnvIntWarning :=
    evalb(kernelopts('level') <= 23
     or kernelopts('level') <= 49 and _Env_assuming_recursion_level = 0); if
        _EnvIntWarning then
        _EnvIntWarning := SimpleQueue( )
    end if;
    if has([args], {'AllSolutions', 'AllSolutions' = true}) then
        _EnvAllSolutions := true
    end if;
    answer :=
    int/int([args], Digits, _EnvCauchyPrincipalValue, _EnvAllSolutions,
        _EnvContinuous);
    if answer = FAIL then
        error "wrong number (or type) of arguments"
    end if;
    answer
```

```
end proc
```

Následující příklad ukazuje, že zdrojový kód některých funkcí zbrazit nelze, protože jsou přímo zkompilované v jádru.

```
print(lhs)
```

```
proc() option builtin; 244 end proc                                    (4.3.3)
```

```
eval(log)
```

```
proc(x::algebraic)                                                     (4.3.4)

    option Copyright (c) 1992 by the University of Waterloo. All rights reserved.;
    local a, x1;
    if nargs <> 1 then
        error "expecting 1 argument, got %1", nargs
    elif type(procname, 'indexed') then
        a := op(procname);
        if not type([a], ['algebraic']) then
            error "invalid index"
        elif type([a, x], ['∞', 'finite']) then
            if(hastype([a, x], 'float'), ln(evalf(x))/ln(evalf(a)), ln(x)/ln(a))
        elif hastype(a, {'∞', 'undefined'}) then
            if(hastype([a, x], 'nonreal'),
            if(hastype([a, x], 'float'), Float(undefined)
            + Float(undefined)*I, undefined + undefined*I), if(hastype([a, x], 'float'
            ), Float(undefined), undefined))
        elif type(a, 'complex(float)')
         or type(x, 'complex(float)') then
            a := evalf(a);
            x1 := evalf(x);
            evalf(evalf[Digits + 2]('ln'(x1)/'ln'(a)))
        else
            ln(x)/ln(a)
        end if
    else
        ln(x)
    end if
```

end proc

Práce s řetězci - string

Řetězce jsou posloupností znaků

"toto je text"

"toto je text" (4.4.1)

"Toto je řetězec"

"Toto je řetězec" (4.4.2)

do kterého nelze zapsat proměnnou. Srovnej s víceslovným identifikátorem.

"ahoj" := 5

```
Error, illegal use of an object as a name
```

"ahoj" := 5

`ahoj` := 5

5 (4.4.3)

Délka řetězce

length("Jak jsem dlouhý?")

16 (4.4.4)

Prázdný řetězec

""

"" (4.4.5)

length(%)

0 (4.4.6)

whattype(%%)

string (4.4.7)

Pro **spuštění řetězce** jako příkazu Maple se užívá příkaz ***parse***

"sin(Pi)"

"sin(Pi)" (4.4.8)

whattype(%);

string (4.4.9)

parse(%%)

0 (4.4.10)

Popř. pro přímý výpočet je někdy nutné použít

parse("sin(Pi)", *statement*)

0 (4.4.11)

Pro více informací využijte nápovědy

?*string*

?*StringTools*

?*searchstring*

$a := \text{"ahoj"};$

$$\text{"ahoj"} \tag{4.4.12}$$

$b := \text{"svete"};$

$$\text{"svete"} \tag{4.4.13}$$

Spojeni retezcu
$cat(a, \text{" "}, b)$

$$\text{"ahoj svete"} \tag{4.4.14}$$

$a \,||\, b \,||\, \text{" adasdf"};$

$$asvete\ adasdf \tag{4.4.15}$$

$v||(1..5);$

$$v1, v2, v3, v4, v5 \tag{4.4.16}$$

Obecná definice procedury

V předchozích kapitolách jsme užívali jen některé vlastnosti procedur. Tento odstavec je určen pro zkompletování informací o tvorbě procedur

```
jmeno_procedury:=proc(parametry)
   local L;
   global G;
   options O;
   description D

   telo_procedury
end proc;
```

Více informací naleznete v nápovědě
?**local**
?**global**
?**option**
?**description**

Typová kontrola parametrů

Dalším velmi důležitým aspektem při tvorbě sofistikovaných procedur je typová kontrola vstupních parametrů. Ta se provádí pomocí tzv. čtyřtečky.
Ukažme to na jednoduchém příkladu

```
> F:=proc(x::integer,y::integer)
     gcd(x,y);
  end proc;
```

$$F := \textbf{proc}(x::integer, y::integer)\ gcd(x, y)\ \textbf{end proc} \tag{4.6.1}$$

>
Volání bez chyby, zadáváme jen celá čísla.
$F(4, 6)$

$$2 \tag{4.6.2}$$

Nyní zkusíme zadat desetinné číslo
$F(4, 6.3)$

```
Error, (in F) invalid input: F expects its 2nd argument, y, to be
of type integer, but received 6.3
```

A je vypsána chyba, že druhý parametr musí být celé číslo.

Nyní se podívejme na typy systému Maple. Jejich seznam můžeme získat pomocí
$?type$

Pro zjištění typu se používá příkaz
$whattype(F)$

$$symbol \tag{4.6.3}$$

Případně pro porovnání se užívá
$type(F, procedure)$

$$true \tag{4.6.4}$$

$is(F, symbol)$

$$true \tag{4.6.5}$$

Ekvivalentem pro neurčení typu parametru v proceduře je typ ***anything***.

Návratový typ funkce se specifikuje stejným způsobem a zapíše se za závorky s parametry.

```
> F1:=proc(x::numeric)::integer;
  # navratova hodnota ma byt cele cislo
      x/2;
  end proc:
```

$$F1 := \mathbf{proc}(\,)\ \ \mathbf{end\ proc} \tag{4.6.6}$$

Následuje volání, které nás přesvědčí, že vše není zcela v pořádku.
$F1(3)$

$$\frac{3}{2} \tag{4.6.7}$$

Podle specifikace by měl být výsledek celým číslem a to není. Tak kde je chyba? Na chyby tohoto typu lze upozornit pomocí následující konstrukce.

$kernelopts(assertlevel = 2)$

$$2 \tag{4.6.8}$$

$F1(3)$

```
Error, (in F1) assertion failed: F1 expects its return value to
be of type integer, but computed 3/2
```

$$F1(3) \tag{4.6.9}$$

V tomto případě výsledek není vypsán, neboť nesplňuje výsledný typ.

Pro vrácení k původnímu nastavení užijeme

***kernelopts*(*assertlevel* = 1)**

$$2 \tag{4.6.10}$$

F1(3)

$$\frac{3}{2} \tag{4.6.11}$$

Druhou možností pro volání parametru funkce odkazem je následující konstrukce, kdy jako parametr bude vstupovat jméno proměnné, do které bude přiřazen výstup.

```
> Sqr:=proc(x,y::name)
      y:=x^2;
  end proc;
```

$$Sqr := \mathbf{proc}(x, y::name)\ y := x\text{^}2\ \mathbf{end\ proc} \tag{4.6.12}$$

Volání je pak následující - je nutné použít uvozovky !

Sqr(3,'*vystup*')

$$9 \tag{4.6.13}$$

vystup

$$9 \tag{4.6.14}$$

Jak je vidět, proměnná výstup byla inicializována uvnitř procedury.

Nakonec ještě poznamenejme, že kontrolu typu proměnných lze uvádět i pro definici lokálních ale ne globálních proměnných uvnitř procedury.

```
> F2:=proc(x::anything)
  local i::integer;
  global vystup;
  i:=3;
  vystup:=x+i:
  end proc:
```

F2(5)

$$8 \tag{4.6.15}$$

vystup

$$8 \tag{4.6.16}$$

i

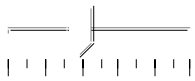
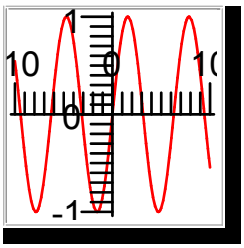
$$i \tag{4.6.17}$$

Lokální proměnná i není mimo tělo procedury dostupná, globální proměnná ano.

Tvorba interaktivních dokumentů - užití vložených komponent

Jedním ze základních předností nového typu dokumentu je možnost vkládání interaktviních komponent. Jejich seznam, anglický název a komponenta jsou zobrazeny v následující tabulce.

Název	Název komponenty	Komponenta	Nápověda
Tlačítko	Button	Button	?*buttonComponent*
Výběrové tlačítko	Toggle Button	ToggleButton	?*togglebutton*
Rozbalovací seznam	Combo Box	ComboBox	?*ComboBox*
Zaškrtávací políčko	Check Box	CheckBox	?*checkbox*
Text	Text Area	wefasdfsfd asdf	?*textarea*
Popisek	Label	La...	?*label*
Seznam	List Box	ListBox (new item) (new item) (new item) (new item) (new item) (new item)	?*listbox*

Posuvník	Slider		?*slider*
Graf	Plot		?*plotComponent*
Matematický zápis	Mathematical Expression	$x^2 + 3 +$	? *MathExpres\ sionCompo\ nent*

Tabulka - vložené komponenty

Tyto komponenty zajišťují interaktivitu v dokumentu. Nyní se podívejme na programování těchto komponent.
Nyní se podívejme na některé z výše uvedených komponent. U ostatních použijte nápovědy.

Tlačítko - Button

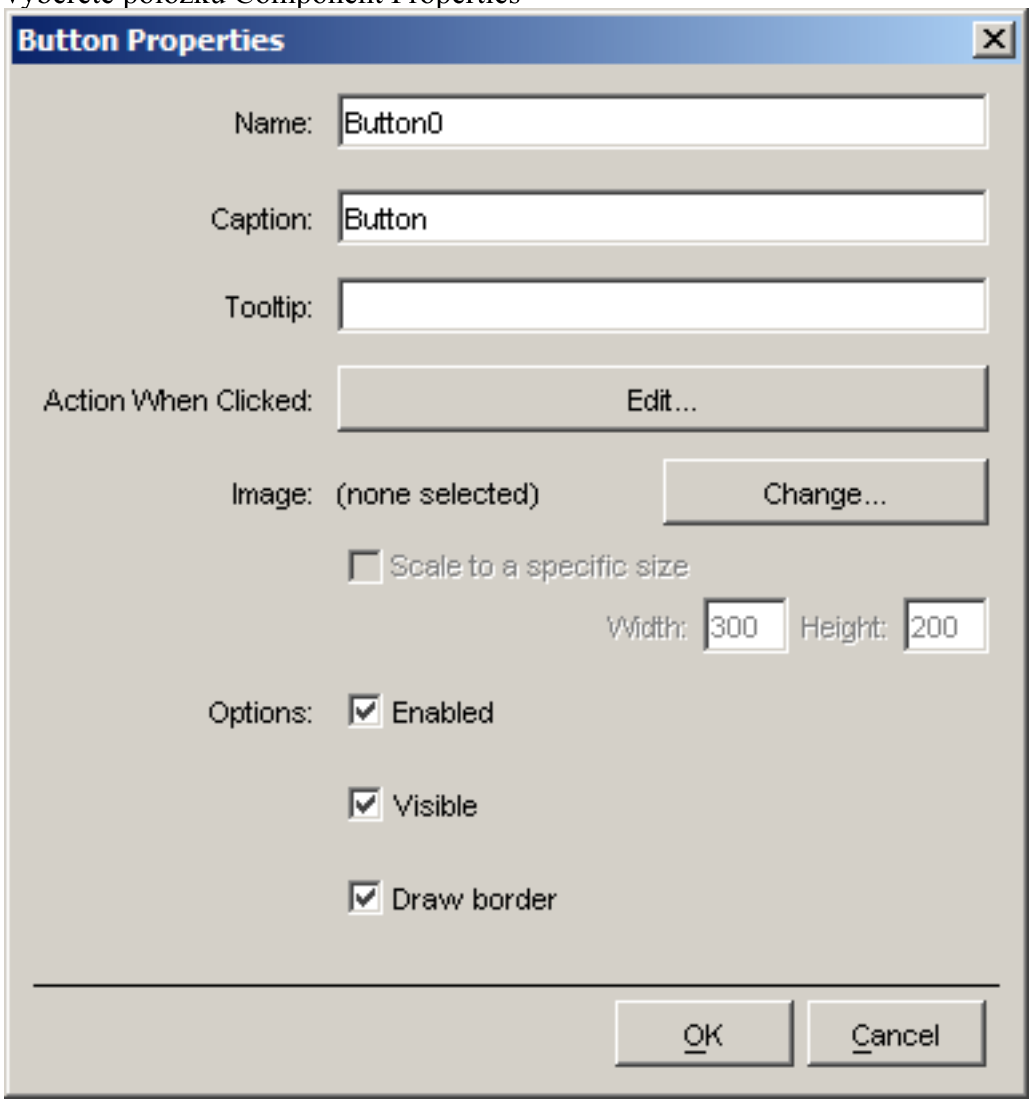
Jednou ze základních komponetn je tlačítko. Slouží zejména k provedení určité operace.

Postup práce je následující:

• vložte komponentu Button z palety Components

• klikněte pravým tlačítkem na komponentu

• vyberete položku Component Properties



• všechny položky jsou zřejmé

• pomocí tlačítka Edit lze vkládat zdrojový kód, který ovládá dané operace

Informace o komponentě Button naleznete v helpu systému Maple, např. stlačením tohoto tlačítka

Nápověda ke komponentě Button

Následující příklad ukáže, jak je možné pracovat s komponentou Button
Vytvoříme tlačítko, pomocí kterého spočítáme integerál obsažený v proměnné ***i1***.

```
> i1:=x^2;
```

$$i1 := x^2 \tag{5.1.1}$$

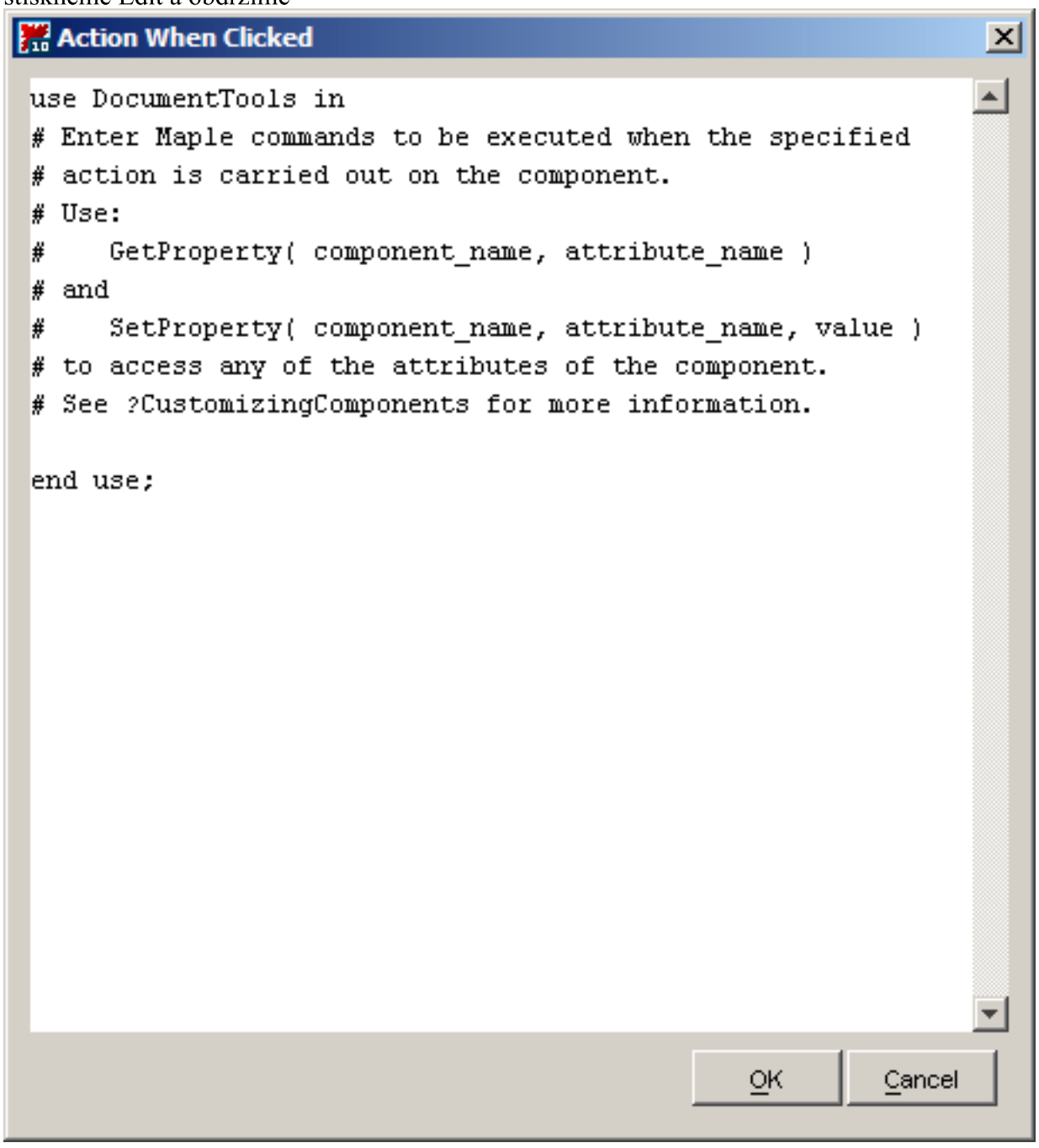
```
>
```

• Vložíme komponentu Button

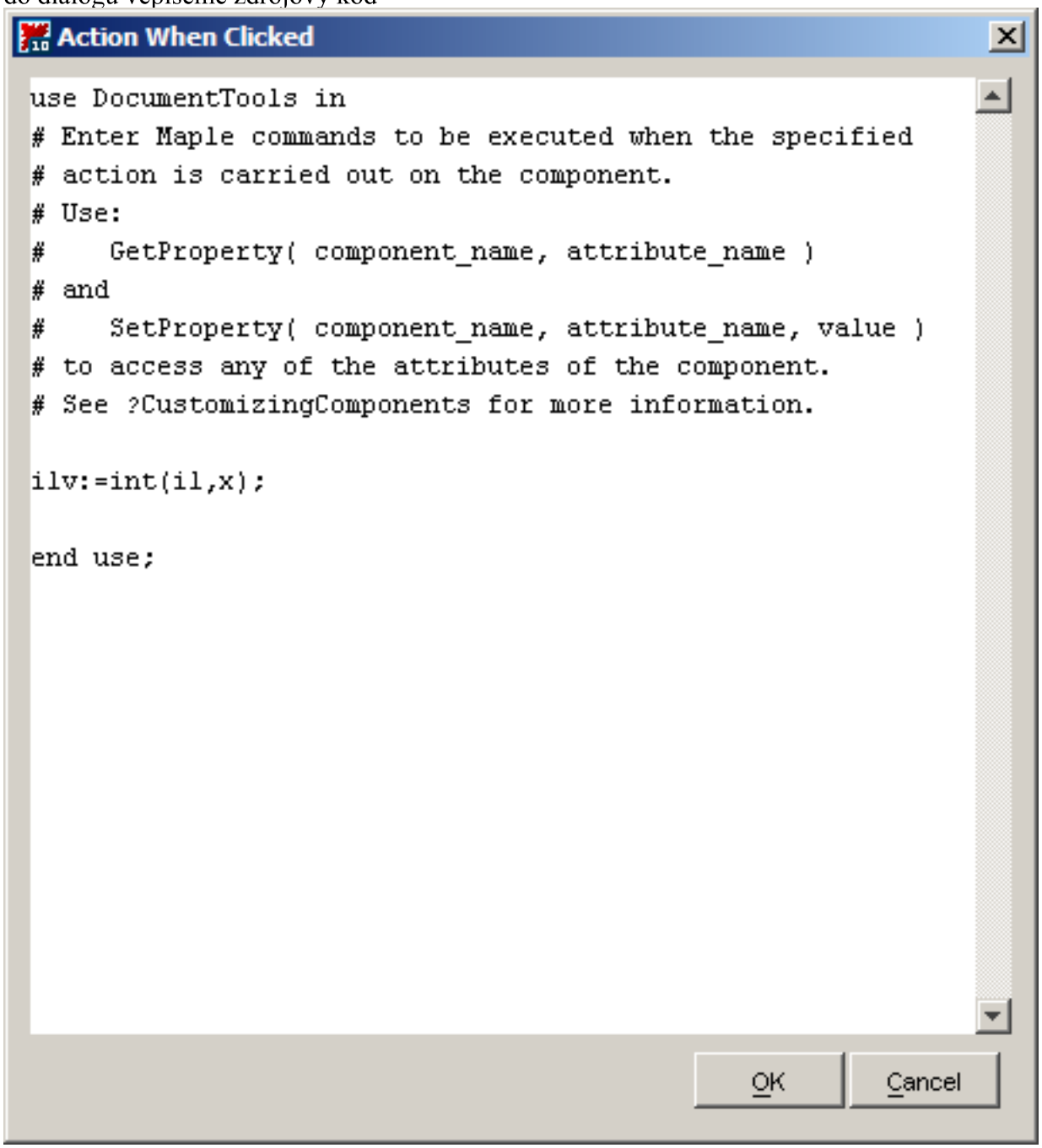
Button

• kontextová nabídka -> Component Properties

• stiskneme Edit a obdržíme



• do dialogu vepíšeme zdrojový kód



• otestujeme, zda byla integrace provedena

```
> ilv;
```

$$\frac{1}{3}\,x^3 \tag{5.1.2}$$

```
>
```

Jak jste si již možná všimli, v dialogu je v poznámkách napsáno, jakým způsobem je možné nastavovat a získávat hodnoty z daných komponet.
Používají se následující dvě funkce ***GetProperty*** a ***SetProperty*** z knihovny ***DocumentTools***.
?DocumentTools

?*DocumentTools, GetProperty*
?*DocumentTools, SetProperty*
Každá komponenta má určité vlastnosti, které lze najít v helpu. Komponenta Button má vlastnosti shrnuté následující tabulce

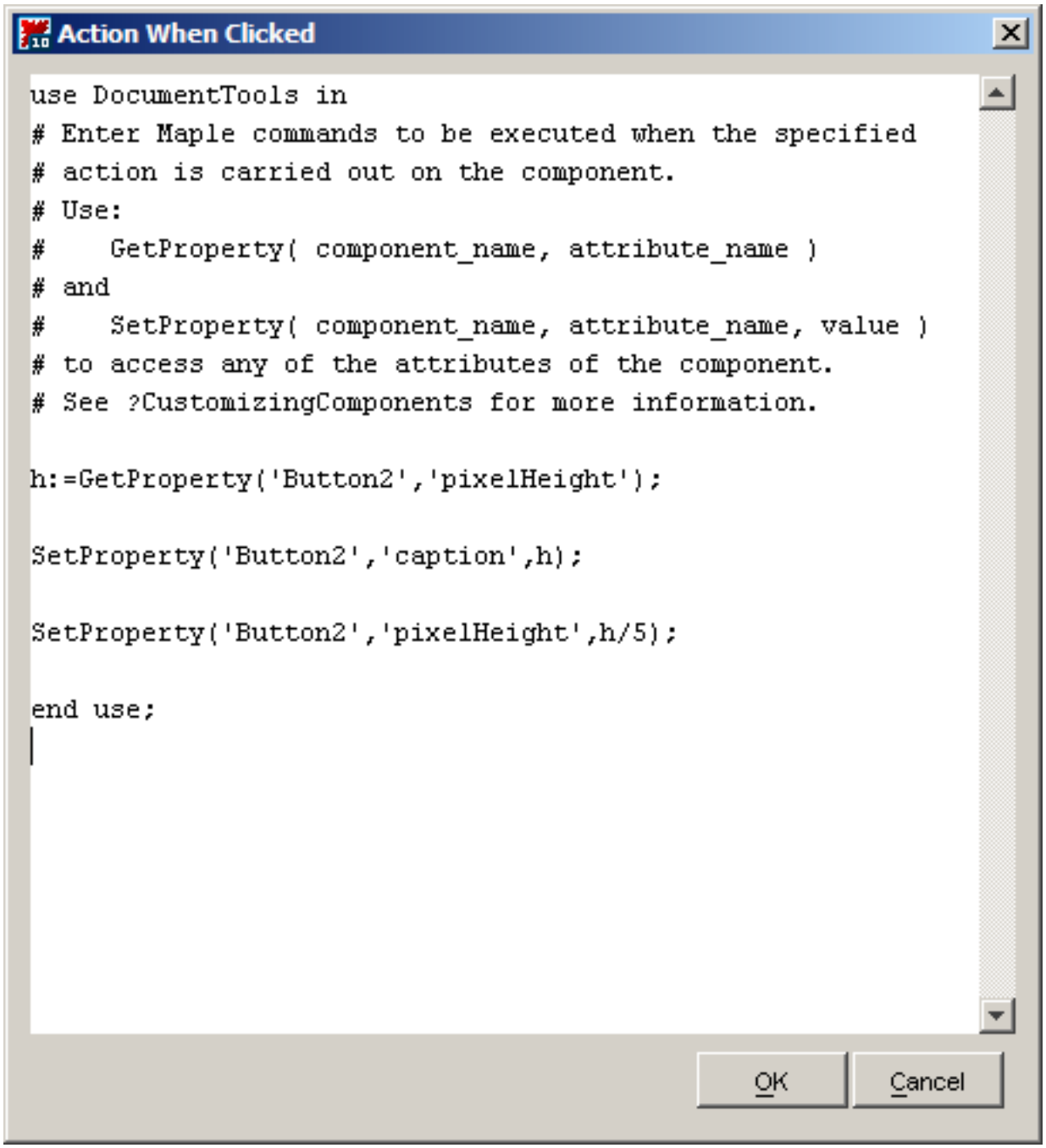
Tabulka vlastností komponenty Button

Název vlastnosti = hodnota	**Popis**
caption = string	Popisek, který je zobrazen na tlačítku
enabled = true or false	Určuje, zda je dotupný a nebo ne
image = name or string	Určuje zobrazení obrázku na tlačítku
pixelHeight = posint	Výška obrázku v pixelech
pixelWidth = posint	Šířka obrázku v pixelech
showBorders = true or false	Určuje, zda jsou zobrazeny okraje
tooltip = string	Zobrazuje tooltip u tlačítka
useSpecifiedSize = true or false	Určuje, zda je použita specifická velikost (viz help)
visible = true or false	Určuje, zda je komponenta viditelná a nebo ne

Na dalším příkladu ukážeme, jakým způsobem lze změnit komponentu. Změníme velikost tlačítka při jeho stisknutí.

• Vložte tlačítko, vložte jakýkoliv obrázek na tlačítko

• změnte kód při stisku tlačítka



• A nyní stiskněte několikrát tlačítko. Výška obrázku se bude vždy o 5 pixelů zvětšovat

Tlačítko Toggle Button

Tlačítko Toggle Button je anologické a proto jej nebudeme probírat.

Zkuste však vytvořit jednoduchou situaci, kdy do nějaké proměnné budete vypisovat, zda je tlačítko stisknuto a nebo ne

Rozbalovací seznam Combo Box a List Box

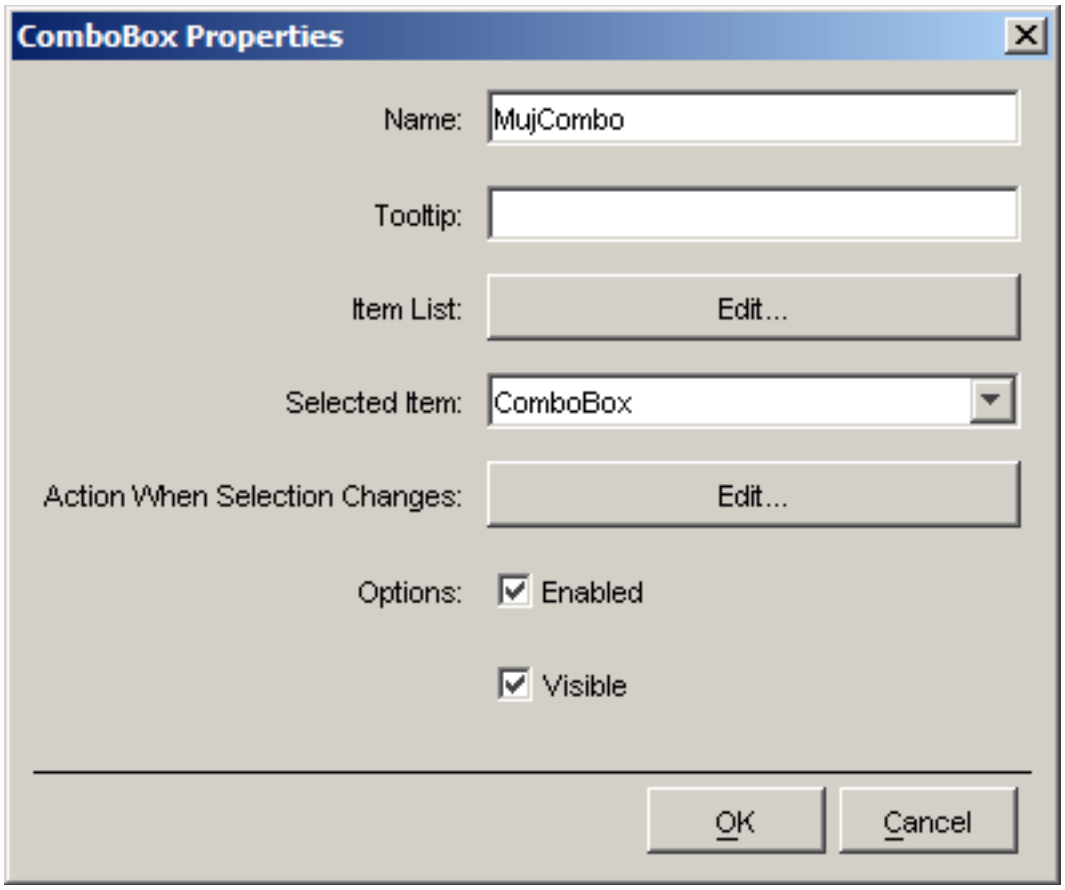
ComboBox a ListBox jsou dvě komponenty, které jsou si velmi podobné. Rozdíl je jen v zobrazení jednotlivých položek. Podívejme se na práci s těmito komponentami.

Přehled vlastností obou komponent

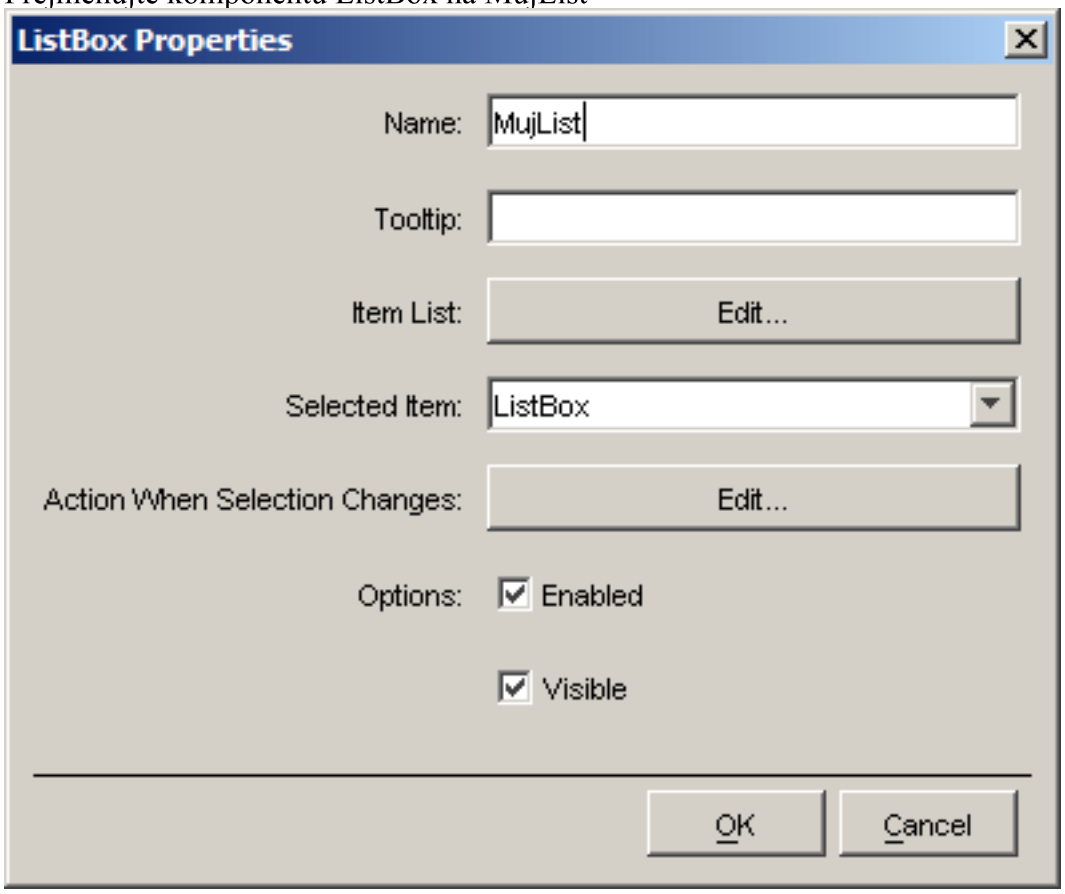
caption = string	Nadpis komponenty
enabled = true or false	Udává, zda je komponenta dostupná
itemlist = symbol (combobox) itemList = list (listbox)	Obsahuje seznam všech položek v množině a nebo seznamu.
tooltip = string	Nastavuje tooltip
value = string	Udává vybranou hodnotu
visible = true or false	Udává, zda je komponenta viditelná

• Vložte ComboBox, ListBox,TextArea a Button

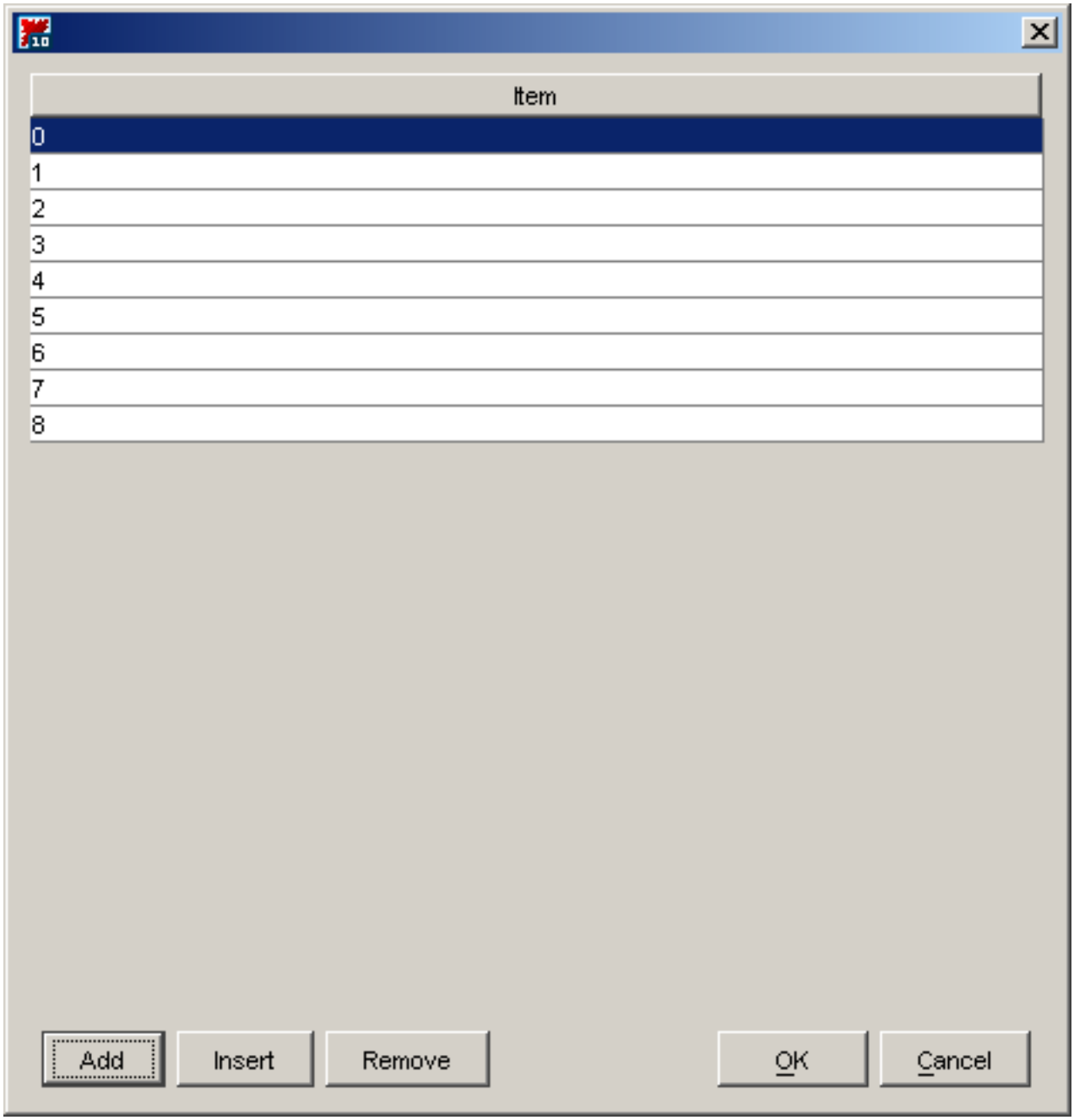
• Přejmenujte komponentu ComboBox na MujCombo



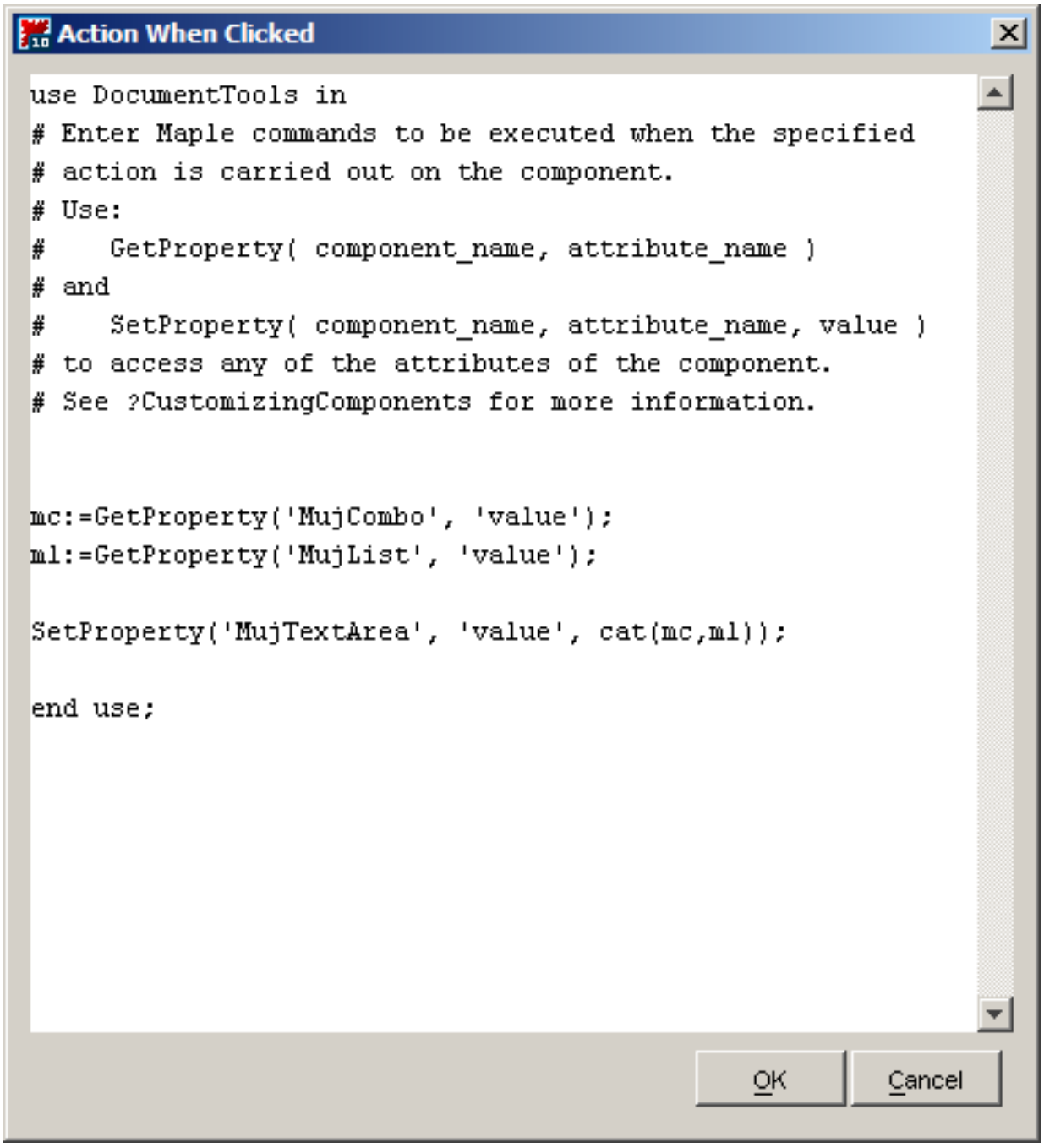
• Přejmenujte komponentu ListBox na MujList



• Přidejte do MujBox několik položek a to pomocí tlačítka Edit



• Nyní pomocí tlačítka budeme přesouvat hodnoty vybrané položky z MujCombo a z MujList do TextArea. Pomocí funkce ***cat*** spojíme oba řetězce dohromady.


Action When Clicked
use DocumentTools in
# Enter Maple commands to be executed when the specified
# action is carried out on the component.
# Use:
#   GetProperty(component_name, attribute_name)
# and
#   SetProperty(component_name, attribute_name, value)
# to access any of the attributes of the component.
# See ?CustomizingComponents for more information.
mc:=GetProperty('MujCombo', 'value');
ml:=GetProperty('MujList', 'value');
SetProperty('MujTextArea', 'value', cat(mc,ml));
end use;
OK
Cancel

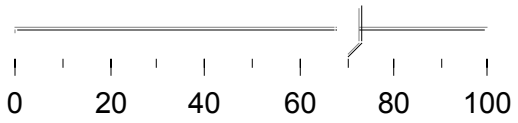
Posuvník a graf - Slider a Plot

Vysvětlení těchto dvou komponent spojíme do jedné kapitoly, neboť jejijch spojení velmi efektní a efektivní.

Podívejme se nejprve na jednotlivé komponenty.

Slider

Nápovědu získáte pomocí
?*slider*



Následující tabulka uvádí dostupné vlastnosti dané komponenty

Vlastnost	Popis
caption = string	Jen pro čtení - Slider
enabled = true or false	Udává, zda je komponenta aktivní.
filled = true or false	Udává, zda jsou zobrazeny body. Defaultní je true
lower = int	Nejnižší hodnota. Defaultní je 0.
majorTicks = posint	Interval mezi hlavními čárkami. Defaultní je 20.
minorTicks = posint	Interval mezi vedlejšími čárkami. Defaultní je 10.
showLabels = true or false	Indikuje, zda jsou zobrazeny popisky.
showTicks = true or false	Zobrazuje čárky.
snapToTicks = true or false	Indikuje, zda má jezdec přiskakovat k hodnotám značeným čárkami.

tooltip = string	Nápověda ke komponentě.
upper = int	Horní hranice.
value = posint	Aktuální hodnota.
vertical = true or false	Indikuje, zda je posuvník zobrazen svisle a nebo vodorovně.
visible = true or false	Indikuje, zda je komponenta viditelná.

Je nutné poznamenat, že některé vlastnosti jsou jen pro čtení a nebo pro zápis. Informace jsou dostupné v nápovědě k dané komponentě.

Příklad bude uveden níže.

Plot

Komponenta Plot je určena pro zobrazování grafických výstupů. Následující tabulka uvádí dostupné vlastnosti. Informace lze nalézt v nápovědě ?*PlotComponent*

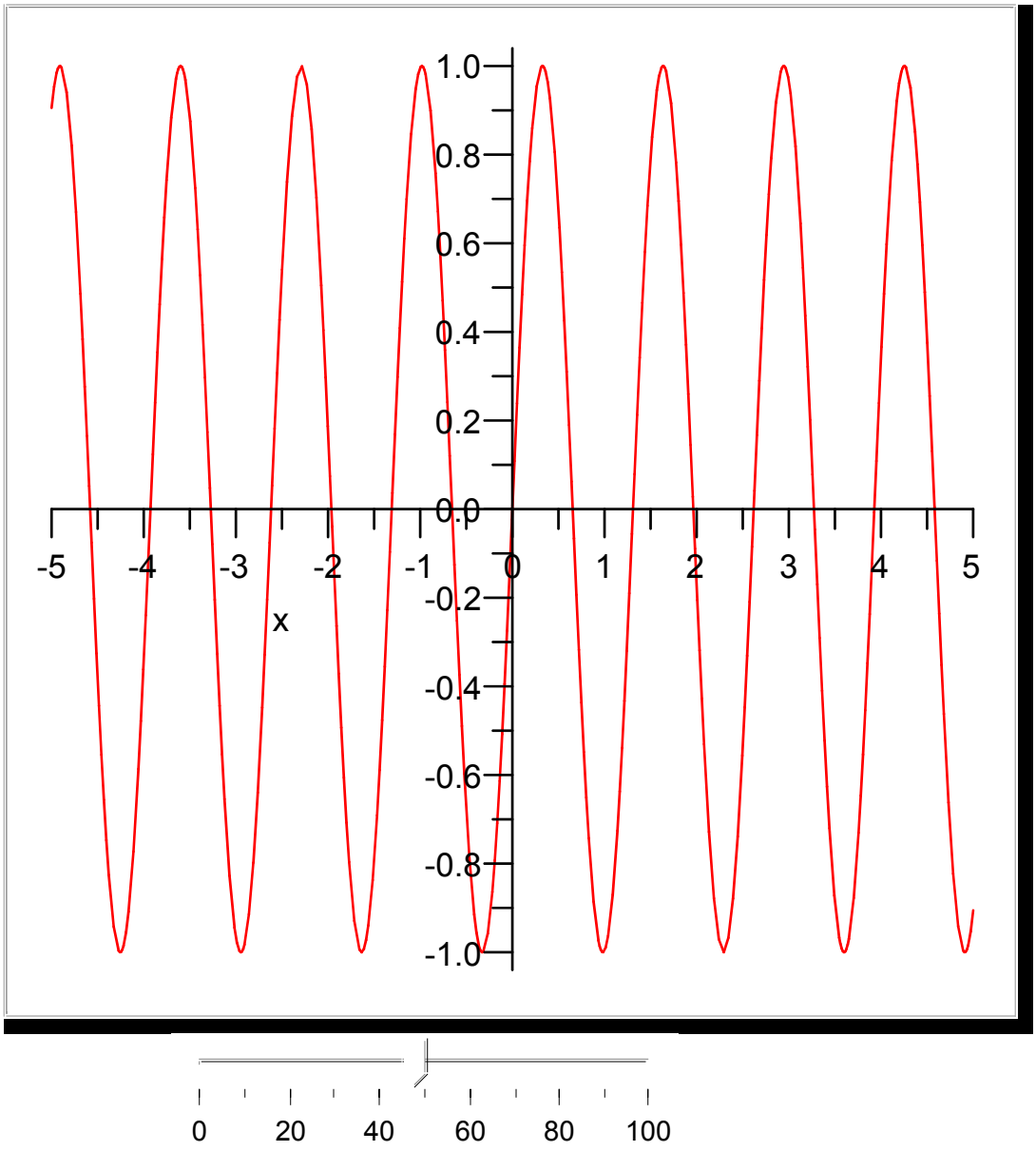
Vlastnosti	Popis
continuous = true or false	Indikuje, zda je animace přehrávána kontinuáně a nebo po snímcích.
delay = posint	Časová mezera v milisekundách mezi jednotlivými snímky. Defaultní hodnota je 100.
frame = posint	Aktuálně zobrazený snímek.
frameCount = posint	Počet zobrazených snímků.
frameBackwards = true or false	Pokud je true, je zobrazen předchozí snímek sekvence. Pokud je animace přehrávána, tak se zastaví.
frameForwards = true or false	Pokud je true, je zobrazen následující snímek sekvence. Pokud je animace přehrávána, tak se zastaví.
pause = true or	Nastavením true se animace

false	zastaví.
pixelHeight = posint	Výška obrázku v pixelech. Dafaultní je 400px.
pixelWidth = posint	Šířka obrázku v pixelech. Dafaultní je 400px.
play = true or false	Nastavení na true začne přehrávat animaci nebo ji pozastaví (pauza).
`stop` = true or false	Nastavením true je animace zastavena. Protože (stop) je také klíčové slovo, je nutné uzavřít ho do uvozovek (``)
toEnd = true or false	Nastavením na true se animace přesune na konec. V případě přehrávání animace se zastaví.
toStart = true or false	Nastavením na true se animace přesune na začátek. V případě přehrávání animace se zastaví.
value = plot command	Hodnota, která má být zobrazena (musí to být PLOT struktura), tj. příkazy plot, plot3d nebo nějaká z vykreslovacích funkcí popř. uživatelem definovaná struktura PLOT nebo PLOT3D
visible = true or false	Udává, zda je komponenta viditelná.

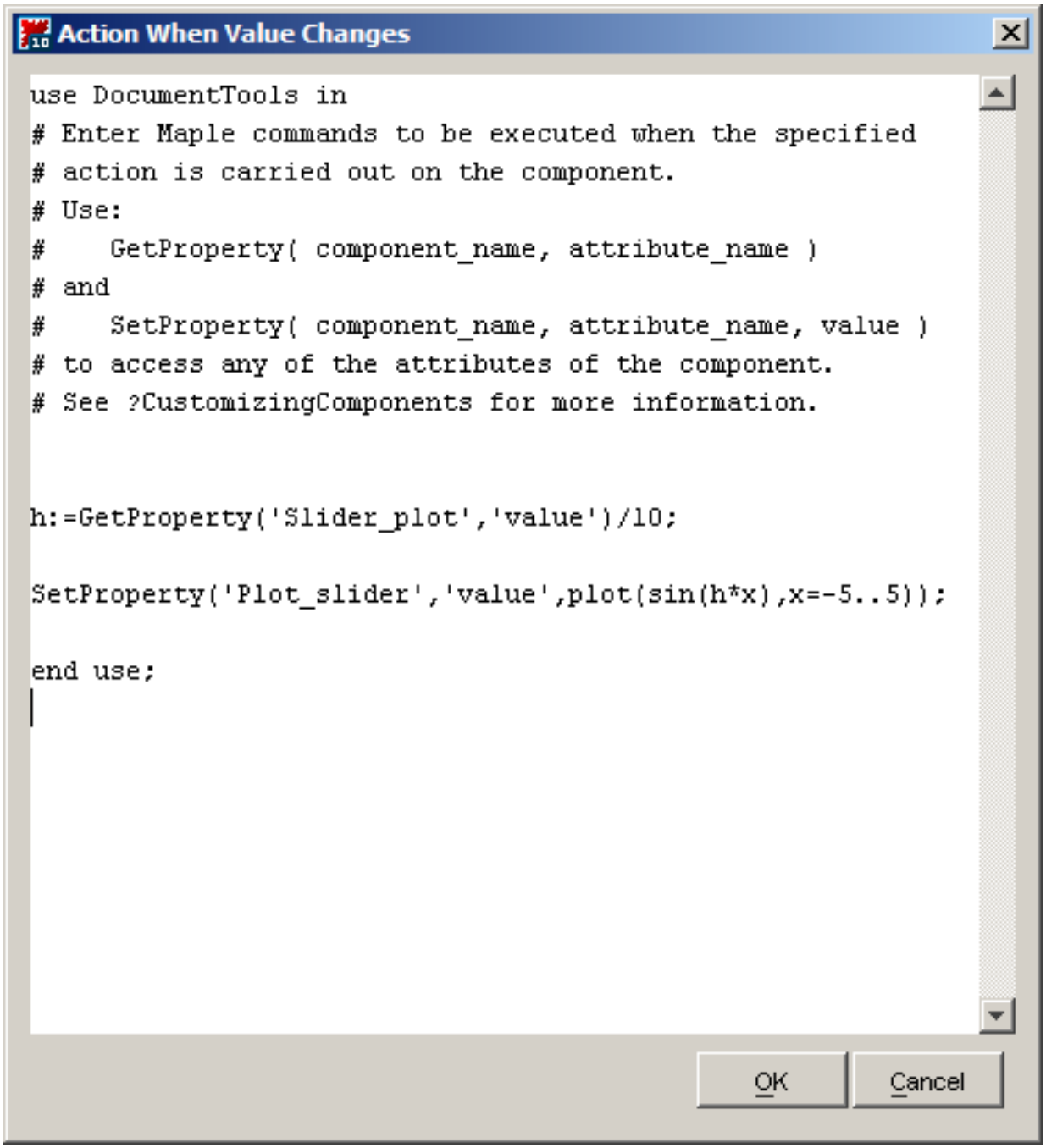
Více informací o vlastnostech a jejich dostupnosti lze najít v nápovědě.
?*PlotComponent*

Nyní se uvedeme velmi efektní aplikaci těchto dvou komponent. Aplikace bude vykreslovat křivku v závislosti na daném parametu. Příklad po té rozšíříme pomocí komponent Text Area a MathExpression.

• Vložme komponenty Slider a Plot a pojmenujme je jako Slider_plot a Plot_slider



• Pomocí tlačítka Edit z dialogu komponenty Slider_plot vepíšeme požadovanou funkci.



• Pomocí jezdce se nyní mění grafika v komponentě Plot_slider

$$h := \frac{GetProperty('Slider_{Plot}','value')}{10};$$

Label, TextArea a Mathematical Expression

Všchny tyto tři komponenty jsou určeny pro zobrazování určitých textů. Komponetna Label se hodí k velmi jednoduchým aplikacím a zobrazování spíšek popisek apod. Komponenta TextArea je určena k zadávání vstupů od uživatele. Poslední komponenta MathExpression je určena pro zobrazování matematických výstupů, které jsou vnitřně uloženy ve formátu MathML..

Práci s těmito komponentami ukážeme na příkladu. Vlasnosti všech tří komponent naleznete vždy v příslušné nápovědě:
?*LabelComponent*
?*TextAreaComponent*
?*MathExpressionComponent*

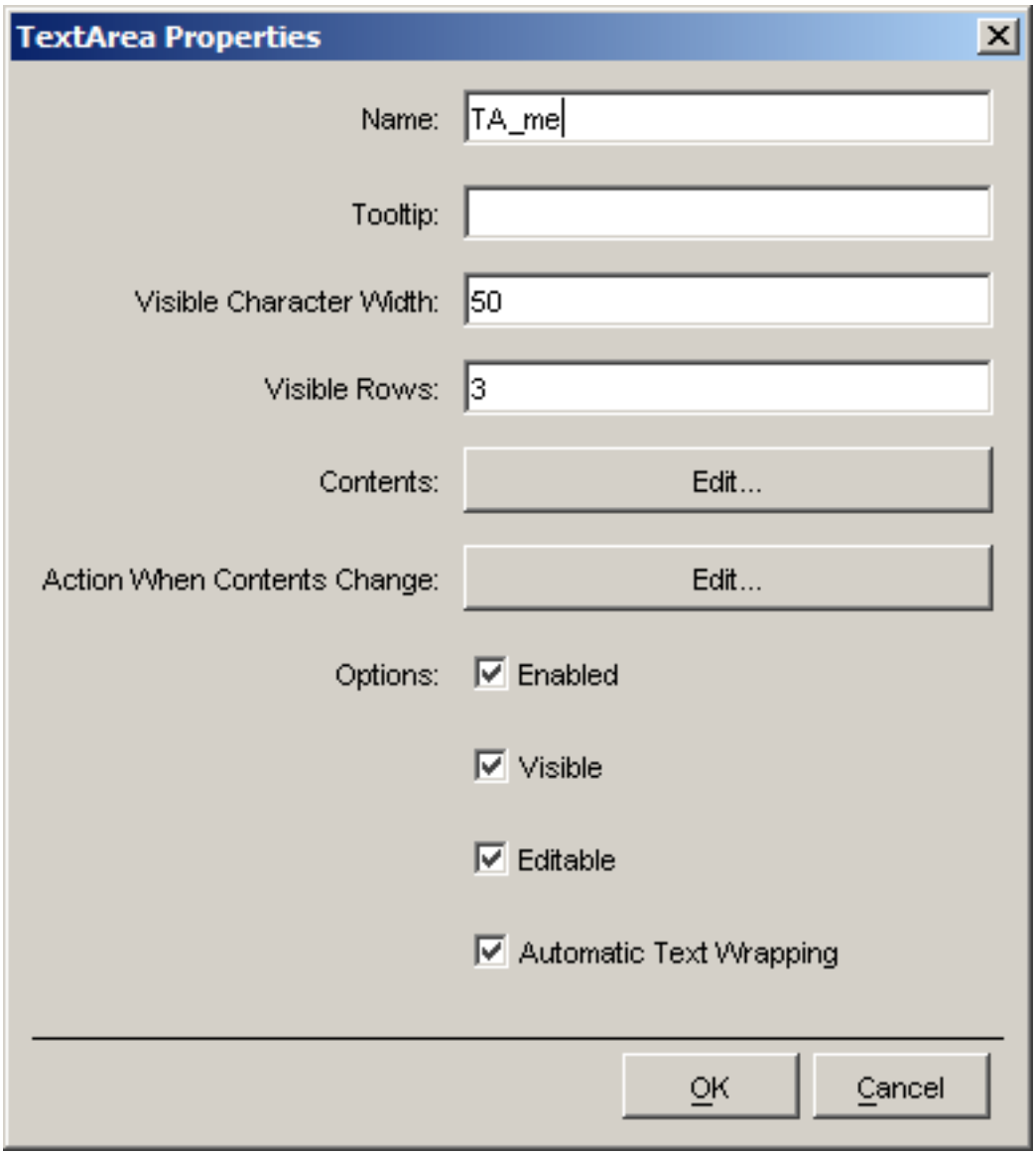
Vložme komponenty TextArea a MathExpression a přejmenujte je na ***TA_me*** a ***ME_ta***.Nakonec vložte tlačítko s názvem ***Button_TAME***.

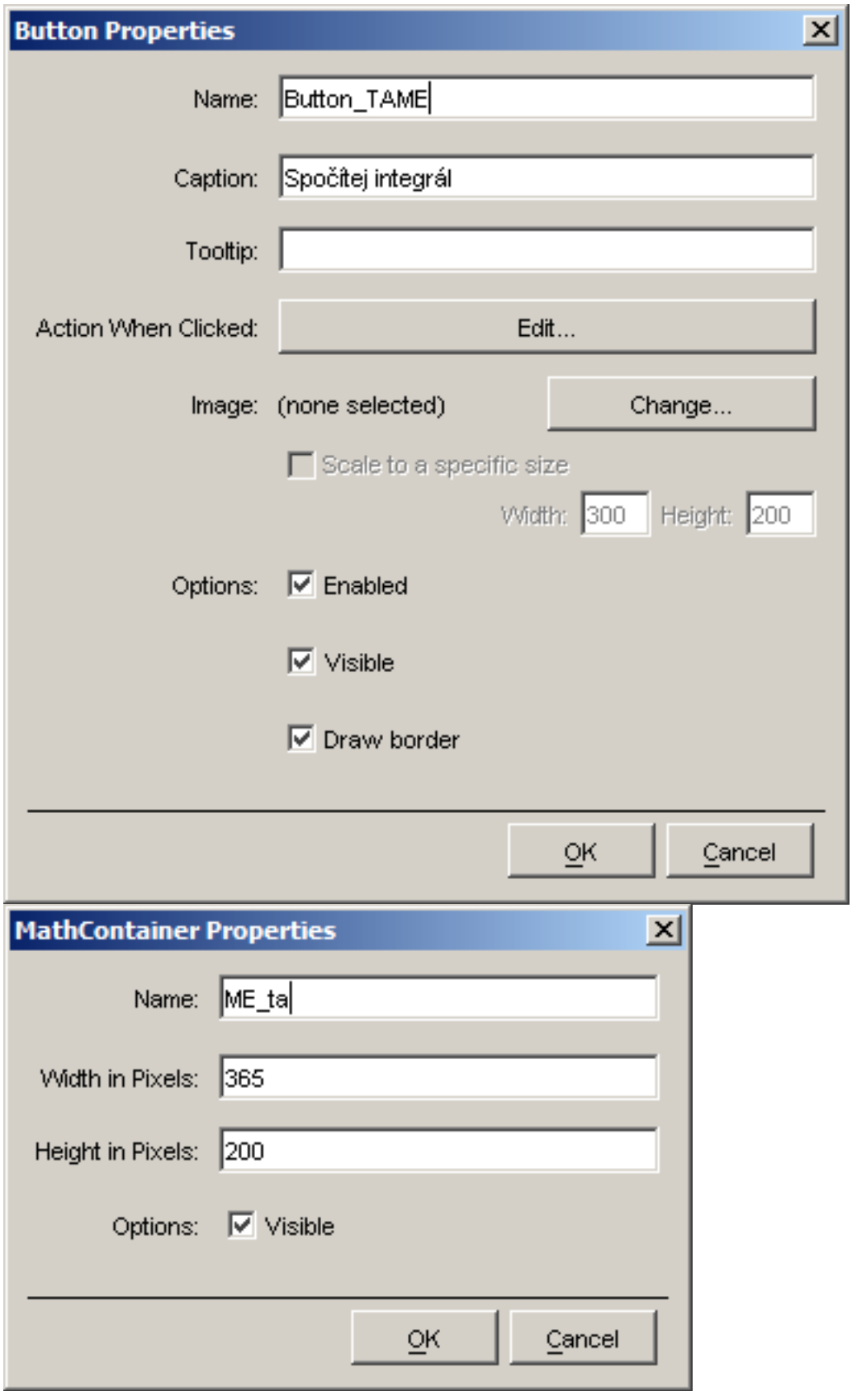
```
2*cos(x)*sin(x)+3*sin(x-6*x^2)
```

$$\int \left(2\cos(x)\sin(x)-3\sin\left(-x+6x^2\right)\right)dx$$
$$=\sin(x)^2-\frac{1}{4}\sqrt{2}\sqrt{\pi}\sqrt{6}\left(\cos\left(\frac{1}{24}\right)\right.$$
$$\left.\text{FresnelS}\left(\frac{1}{6}\,\frac{\sqrt{2}\sqrt{6}\left(6x-\frac{1}{2}\right)}{\sqrt{\pi}}\right)\right.$$
$$\left.-\sin\left(\frac{1}{24}\right)\text{FresnelC}\left(\frac{1}{6}\,\frac{\sqrt{2}\sqrt{6}\left(6x-\frac{1}{2}\right)}{\sqrt{\pi}}\right)\right)$$

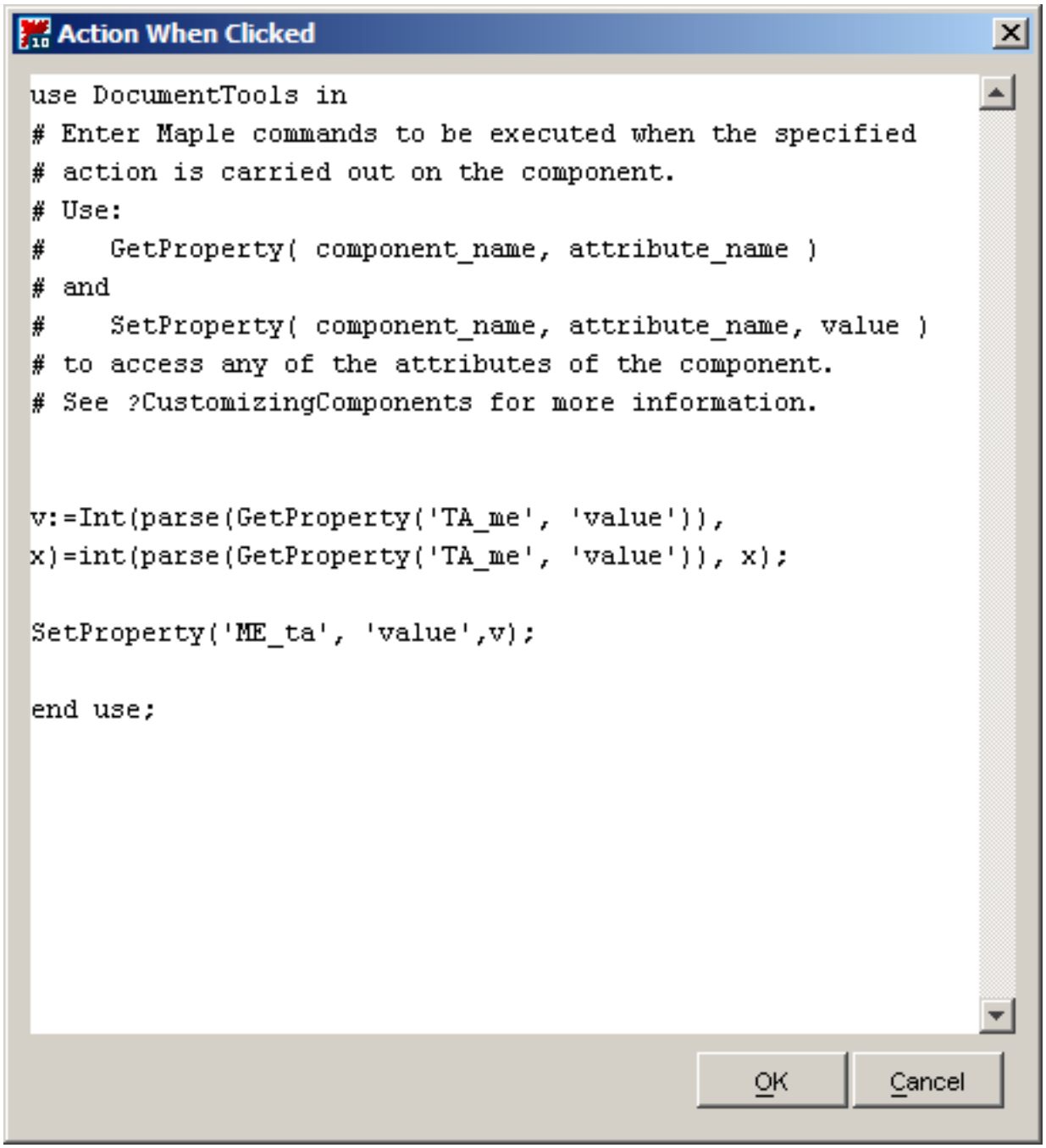
Spočítej integrál

• změníme názvy komponent


TextArea Properties
Name:
TA_me
Tooltip:
Visible Character Width:
50
Visible Rows:
3
Contents:
Edit...
Action When Contents Change:
Edit...
Options:
Enabled
Visible
Editable
Automatic Text Wrapping
OK
Cancel



• vložíme akci do tlačítka pomocí Edit



Pokud jde o ostatní neprobrané komponenty, práce s nimi a jejich vlastnosti jsou velmi podobné s těmi probranými.

Následující příklad ukazuje komplexní použití komponent.

Příklad

Tento příklad je ukázkou užití výše probraných komponent jako celku. Není zde vysvětlen postup práce, neboť je shodný s výše uvedenými postupy. Na konkrétní zdrojový kód jednotlivých komponent se podívejte pomocí Dialogu vlastností.

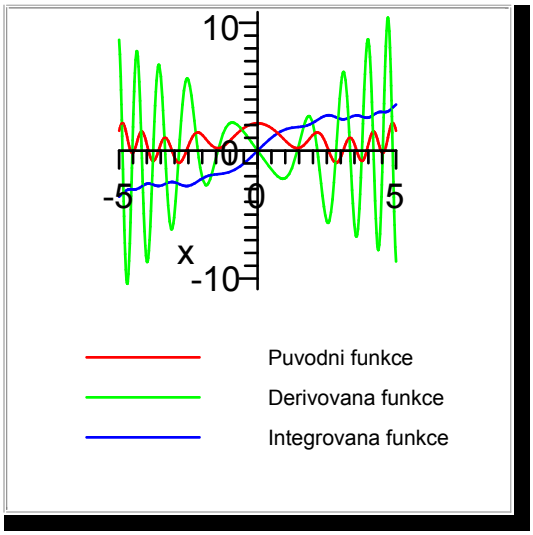
Pokročilý příklad užití kom...

```
sin(x^2+3)+cos(x)+1
```

Derivace | Integrace

Vykresli vše dohromady

$$\int (\sin(x^2+3)+\cos(x)+1)\,dx = \frac{1}{2}\sqrt{2}\sqrt{\pi}\left(\cos(3)\,\text{FresnelS}\left(\frac{\sqrt{2}\,x}{\sqrt{\pi}}\right) + \sin(3)\,\text{FresnelC}\left(\frac{\sqrt{2}\,x}{\sqrt{\pi}}\right)\right) + \sin$$



Technologie Maplet

Tato kapitola se zabývá technologií Maplet, která byla uvedena spolu se systémem Maple 8. Tato technologie postavená na Javě však byla již k dispozici jako dodatek k verzi Maple 7. Technologie Maplet je určena pro vytváření uživatelských dialogů či aplikací, které budou poskytovat uživateli velmi pohodlné a přehledné prostředí pro řešení daného problému. V současné době jsou Maplety součástí téměř každé nově přidané rozšiřující knihovny. Stežejní součástí jsou zejména ve všech částech knihovny Student (?*Student*)

Nebudeme se zabývat nějakou teorií a hned přejdeme ke tvorbě mapletů. Začneme nejprve s těmi nejjednoduššími, pak si představíme jednotlivé elementy a nakonec si ukážeme vytváření mapletů pomocí tzv. Maplet Builderu (?*MapletBuilder*). Příklady jsou analogické těm z nápovědy.

Pro práci s Maplety je určena knihovna Maplets, která obsahuje další podknihovny:

• Maplets[Elements] - ?*Maplets*[*Elements*]

• Maplets[Tools] - ?*Maplets*[*Tools*]

• Maplets[Utilities] - ?*Maplets*[*Utilities*]

Postupně si tyto knihovny a příkazy v nich probereme.

První Maplet

Tento první Maplet vypisuje tradičně pozdrav světu.
Nejvyšším elementem všech Mapletů musí být element Maplet z Maplets[Elements].

```
> restart:
> with(Maplets[Elements]);
```

[*Action*, *AlertDialog*, *Argument*, *BorderLayout*, *BoxCell*, *BoxColumn*, *BoxLayout*, *BoxRow*, *Button*, *ButtonGroup*, *CheckBox*, *CheckBoxMenuItem*, *CloseWindow*, *ColorDialog*, *ComboBox*, *ConfirmDialog*, *DropDownBox*, *Evaluate*, *FileDialog*, *Font*, *GridCell*, *GridCell2*, *GridLayout*, *GridRow*, *HorizontalGlue*, *Image*, *InputDialog*, *Item*, *Label*, *ListBox*, *Maplet*, *MathMLEditor*, *MathMLViewer*, *Menu*, *MenuBar*, *MenuItem*, *MenuSeparator*, *MessageDialog*, *PasswordField*, *Plotter*, *PopupMenu*, *QuestionDialog*, *RadioButton*, *RadioButtonMenuItem*, *Return*, *ReturnItem*, *RunDialog*, *RunWindow*, *SetOption*, *Shutdown*, *Slider*, *Table*, *TableHeader*, *TableItem*, *TableRow*, *TextBox*, *TextField*, *ToggleButton*, *ToolBar*, *ToolBarButton*, *ToolBarSeparator*, *VerticalGlue*, *Window*] (6.1.1)

```
>
```

Pro zobrazení mapletu se používá funkce ***Display***, která je přímo v knihovně ***Maplets***.

```
> with(Maplets);
```

[*Display*, *Elements*, *Examples*, *Tools*, *Utilities*] (6.1.2)

```
> Display(  Maplet(  ["Ahoj světe"]  ));
```

Výstupem je nejjednodušší maplet - 

.

Všimněte si seznamu uvnitř funkce Maplet. Více se dozvíte v dalším odstavci.

Seznam v Mapletu

Maplet je definován užitím tzv. box layout struktury. Tato struktura je založena na konceptu vložených seznamů do základního seznamu. Seznam je uzavřen do hranatých závorek [].

```
> seznam:=[1,5,7];
```

$$seznam := [1, 5, 7] \tag{6.2.1}$$

```
>
```

Vložený seznam do seznamu je např. následující

```
> sez:=[1, [1,3,5],[2,6],[2, [1,2,3],2],2];
```

$$sez := [1, [1, 3, 5], [2, 6], [2, [1, 2, 3], 2], 2] \tag{6.2.2}$$

```
> sez[4,2,1];
```

$$1 \tag{6.2.3}$$

Při psaní mapletu jsou texty, vstupy a další elementy definovany v hlavním seznamu, jak již bylo vidět na prvním příkladu

Vložení tlačítka

Nyní do předchozího mapletu vložíme tlačítko OK, které maplet zavře pomocé funkce ***Shutdown***, která je opět v knihovně *Maplet[Elemets]*

```
> restart:
> with(Maplets):with(Maplets[Elements]):
> Display(
    Maplet(   # spusteni mapletu - hlavni povinna komponenta
      [   # hlavni seznam
         "Ahoj světe" ,
         Button("OK", Shutdown())  # tlacitko OK
      ]
    )
```

```
  );
>
```

Protože je tlačítko i nápis v hlavním seznamu jsou zobrazeny podsebou. Pokud bychom je chtěli mít vedle sebe, museli bychom obě položky uzavřít do dalšího seznamu.

```
> restart:
> with(Maplets):with(Maplets[Elements]):
> Display(
    Maplet( # spusteni mapletu - hlavni povinna komponenta
       # pomoci window lze menit vlastnosti Maplet
       [       # hlavni seznam
          [    # vnitrni seznam
          "Ahoj světe" ,
          Button("OK", Shutdown())  # tlacitko OK
          ]
       ]
    )
  );
>
```

Přidání nadpisu do titulku mapletu (dialogu)

Nyní do předchozího mapletu přidáme nadpis dialogu. Pro určení specifických možností dialogu je nutné užít elementu Window, který však není vložen do hlavního seznamu, nebot jde o specifikaci hlavního okna mapletu. Vše je zřejmé z následujícího kódu.

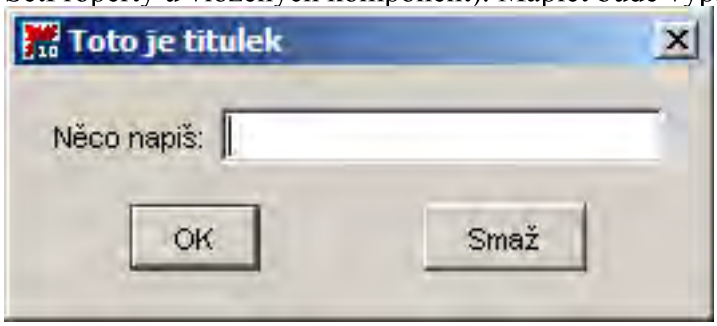
```
> restart:
> with(Maplets):with(Maplets[Elements]):
> Display(
    Maplet( # spusteni mapletu - hlavni povinna komponenta
      Window
      (     # pomoci window lze menit vlastnosti Maplet
         'title'="Toto je titulek",
         'width'=300,
         'height'=500,
         [       # hlavni seznam
            [    # vnitrni seznam
            "Ahoj světe" ,
```

```
            Button("OK", Shutdown())   # tlacitko OK
            ]
        ]
    )
  )
);
```

Přidání editačního pole

Následující příklad přidá do mapletu další tlačítko a editační pole. Tlačítko editační pole smaže. Text ahoj světe bude nahrazen textem "Něco napiš"
Editační pole se vkládá jako ***TextField*** element s názvem *TF1* a bude prázdné. Komponenta *TextField* je jednořádkové textovém pole. Víceřádkové je vytvářeno pomocí komponenty ***TextBox***.. Odpovídající kód tedy je ***TextField['TF1']().***

K textovému poli bude druhé tlačítko přistupovat pomocí funkce ***SetOption*** (analogie s funkcí SetProperty u vložených komponent). Maplet bude vypadat jako



Z toho důvodu je nutné uzavřít jak popisek a textové pole do vnitřního seznamu, stejně jako obě tlačítka - tedy konstrukce s hlavním seznamem bude vypadat
[[popisek, textovepole], [tlacitko ok, tlacitko smaz]]

Dále budeme požadovat vrácení hodnoty v textovém poli při zavření pomocí tlačítka OK. Toho docílíme tím, že do funkce Shutdown složíme seznam s názvy komponent, ze kterých chceme hodnoty vrátit. Tedy v našem případě ***Button("OK", Shutdown(['TF1']));***

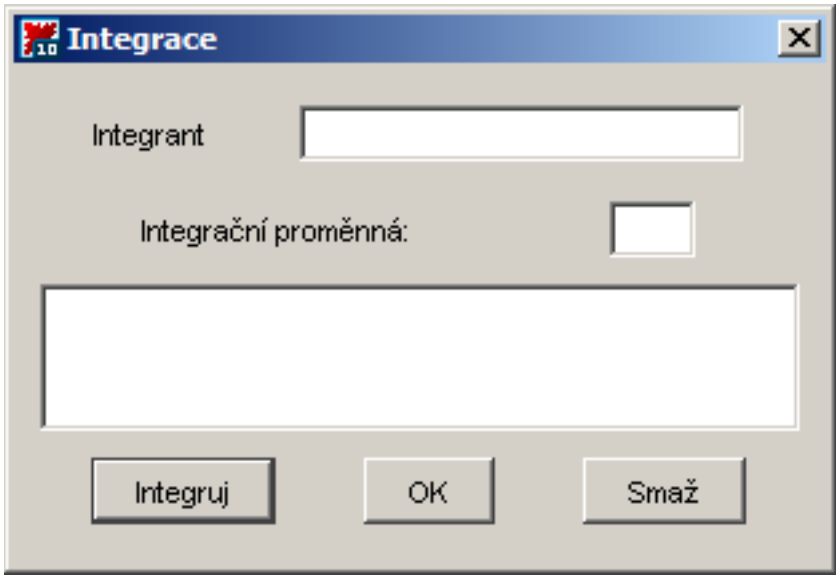
Kompletní výpis

```
> restart:
> with(Maplets):with(Maplets[Elements]):
> Display(
    Maplet( # spusteni mapletu - hlavni povinna komponenta
      Window
      (     # pomoci window lze menit vlastnosti Maplet
        'title'="Toto je titulek",
        [       # hlavni seznam
```

```
            [    # vnitrni seznam - pro seskupeni elementu
              "Něco napiš:" ,
              TextField['TF1']()
            ],
            [    # vnitrni seznam - pro seskupeni elementu
              Button("OK", Shutdown(['TF1'])),  # tlacitko OK s
navratovou hodnotou
              Button("Smaž", SetOption('TF1' = ""))  # tlacitko
smaz
            ]
          ]
        )
      )
);
                                    [""]                              (6.5.1)
>
```

Zaslání výpočtu do mapletu

Tento příklad ilustruje zaslání výpočtu do mapletu. Maplet bude vypadat jako na obrázku. Bude využita komponenta ***TextBox***, která je víceřádkovým ekvivalentem *TextField*u.



Komponentami budou (zhora a zleva):

- Popisek - Integrant
- TextField - TF1 - reprezentuje zadaný řetězec
- Popisek - Integrační proměnná

• TextField - TF2 - reprezentuje zadanou proměnnou

• TextBox - TB1 - do něj bude zobrazen výstup ve formě integrálu - bude vypočítán

• Button - Integruj

• Button - OK

• Button - Smaž

TextBox má velikost 3x40 a uživatel do něj nemůže přistupovat, neboť není povolena editace a to pomocí **'editable'='false'**.

Další novou funkcí je funkce Evaluate, pomocí které se vyhodnocuje výraz. Konstrukce ***Evaluate ('TB1' = 'int(TF1,TF2)')*** spočítá integrál ze zadené funkce a neznámé proměnné a hodnotu vloží do TextBoxu.

```
> restart:
> with(Maplets):with(Maplets[Elements]):
> Display(
    Maplet( # spusteni mapletu - hlavni povinna komponenta
      Window
      (     # pomoci window lze menit vlastnosti Maplet
         'title'="Integrace",
         [       # hlavni seznam
            [    # vnitrni seznam - pro seskupeni elementu - 1.
radek
              "Integrant" ,
              TextField['TF1']()
            ],
            [    # vnitrni seznam - pro seskupeni elementu - 2.
radek,
              "Integrační proměnná: ",
              TextField['TF2'](3)
            ],
            TextBox['TB1']('editable'='false',3..40), # 3.radek
                 # pole pro vypis, needitovatelne, rozmery - 3
radky, 40 znaku
            [    # vnitrni seznam - pro seskupeni elementu - 4.
radek
              Button("Integruj", Evaluate( 'TB1' = 'int(TF1,TF2)
')),
              Button("OK", Shutdown(['TF1','TF2','TB1'])),  #
tlacitko OK s navratovou hodnotou
              Button("Smaž", SetOption( 'TF1' = "" ))  #
tlacitko smaz
            ]
```

```
          ]
        )
      )
    );
```

$$["x", "x", "1/2*x^2"] \tag{6.6.1}$$

```
> parse(%[3]);  # z retezce udelame prikaz
```

$$\frac{1}{2}\,x^2 \tag{6.6.2}$$

```
>
```

Ošetření chyb - spuštění vlastní procedury

Předchozí příklad měl však jednu zásadní chybu. Nebyly ošetřeny vstupy od uživatele. K jejich ošetření a k ukázce volání vlastní procedury z mapletu nám bude sloužit následující příklad. Vytvoříme vlastní proceduru, která bude pomocí funkce ***Get*** z ***Maplet[Tools]*** získávat hodnoty z obou TextFieldu a pomocí čtyřtečky jim přiřadíme typy. Funkce ***Get('TF1'::algebraic)*** získá hodnotu z TF1 a zjistí typ vstupu. Pokud typ neodpovídá, je vypsána chyba. Je také možné pomocí dalšího parametru ***corrections=true*** vynucení si zadání hodnoty. Tuto možnost využijeme jen ve druhém ***TextFieldu***, který reprezentuje neznámou proměnnou.

```
> restart:
> with(Maplets):with(Maplets[Elements]):
>
> VlastniFce:=proc()
    local integrand, promenna;

    use Maplets[Tools] in
        integrand := Get('TF1'::algebraic);  #
        promenna:=Get('TF2'::name,corrections=true);
    end use;

    int(integrand,promenna);
  end proc:
>
> Display(
    Maplet( # spusteni mapletu - hlavni povinna komponenta
      Window
      (     # pomoci window lze menit vlastnosti Maplet
        'title'="Integrace",
        [       # hlavni seznam
```

```
            [    # vnitrni seznam - pro seskupeni elementu - 1.
radek
              "Integrant" ,
              TextField['TF1']()
            ],
            [    # vnitrni seznam - pro seskupeni elementu - 2.
radek,
              "Integrační proměnná: ",
              TextField['TF2'](3)
            ],
            TextBox['TB1']('editable'='false',3..40), # 3.radek
                 # pole pro vypis, needitovatelne, rozmery - 3
radky, 40 znaku
            [    # vnitrni seznam - pro seskupeni elementu - 4.
radek

# --> zde jsme kod zmenili a to z 'TB1' = 'int(TF1,TF2)' na
'TB1' = 'VlastniFce'
              Button("Integruj", Evaluate( 'TB1' = 'VlastniFce')
),  # do TB1 vlozime vysledek funkce VlastniFce
# --> konec zmeny

              Button("OK", Shutdown(['TF1','TF2','TB1'])),  #
tlacitko OK s navratovou hodnotou
              Button("Smaž", SetOption( 'TF1' = "" ))  #
tlacitko smaz
            ]
        ]
    )
  )
);
```

["x", "", "1/2*x^2"] (6.7.1)

```
>
```

Přidání dialogu s nápovědou a grafu

Tento příklad demostruje, jakým způsobem se dá přidat dialog s nápovědou, graf a posuvník.

Nejprve než se pustíme do vytváření tohot mapletu se podíváme, jakým způsobem je možné po ošetření vstupních dat zadaná data uložit do vstupních elemetů. V naší proceduře užijeme funkci ***Set***, která je přesným opakem funkce ***Get*** a je podobná funkce ***SetOptions***, která se užívá u

vložených komponent. Její použití je jasné ze zdrojového kódu. Více informací můžete nalézt v nápovědě (?*Maplets*, *Tools*, *Set*).

```
> restart:
> with(Maplets):with(Maplets[Elements]):
>
> VlastniFce:=proc()
    local integrand, promenna;

    use Maplets[Tools] in
        integrand := Get('TF1'::algebraic,corrections=true);  #
        promenna:=Get('TF2'::name,corrections=true);

        Set('TF1'=integrand);
        Set('TF2'=promenna);
    end use;

    int(integrand,promenna);
  end proc:
```

Do mapletu přidáme graf pomocí komponenty ***Plotter*** (více v nápovědě ?*Maplets*, *Elements*, *Plotter*). Graf bude zobrazen když uživatel stiskne tlačítko Plot. Dále posuvník, který je zastoupen komponentou Slider (více ?*Maplets*, *Elements*, *Slider*). Přidáme také dialogové okno pomocí funkce MessageBox (více ?*Maplets*, *Elements*, *MessageDialog*). Ještě vysvětleme funkci ***Action***. Ta je určena pro provádění váce operací najednou. Vše ostatní je jasné ze zdrojového kódu.

```
> Display(
    Maplet( # spusteni mapletu - hlavni povinna komponenta
      Window
      (     # pomoci window lze menit vlastnosti Maplet
         'title'="Integrace a vykrelesní",
         BoxColumn
         ( # hlavni seznam je vymenen za BoxColumn - sloupec
  komponent
           'vscroll'='always',
           [    # vnitrni seznam - pro seskupeni elementu - 1.
  radek
             "Integrant" ,
             TextField['TF1']()
           ],
           [    # vnitrni seznam - pro seskupeni elementu - 2.
  radek,
             "Integrační proměnná: ",
```

```
            TextField['TF2'](3)
          ],
          TextBox['TB1']('editable'='false',3..40), # 3.radek
               # pole pro vypis, needitovatelne, rozmery - 3
radky, 40 znaku

          Plotter['PL1'](), # graf

          Slider['SL1'](0..20,5,'showticks','majorticks'=5,
'minorticks'=1,'visible'='false',
                        Evaluate('PL1'='plot([TF1,TB1],TF2=0..
SL1)')),
                            # posuvnik

          [    # vnitrni seznam - pro seskupeni elementu - 4.
radek
            Button("Integruj",
                  Action(
                         Evaluate( 'TB1' = 'VlastniFce'),  #
do TB1 vlozime vysledek funkce VlastniFce
                         SetOption('B1'(enabled)='true')
                        )
                  ),
            Button("OK", Shutdown(['TF1','TF2','TB1'])),  #
tlacitko OK s navratovou hodnotou
            Button("Smaž",
                  Action(
                        SetOption('TF1' = "" ),
                        SetOption('TF2' = "" ),
                        SetOption('TB1' = "" ),
                        SetOption('B1'('enabled') = 'false'),
                        SetOption('SL1'('visible') = 'false')
,
                        Evaluate('PL1' = 'plot(undefined,x=0.
.SL1)' )
                        )
                ),  # tlacitko smaz
            Button("Nápověda",RunDialog('MD1')),
            Button['B1']("Kresli",'enabled'='false',
                        Action(
                               SetOption('SL1'('visible')=
'true'),
```

```
                                    Evaluate('PL1'='plot([TF1,TB1]
,TF2=0..SL1)')
                                )
                            )
          ] # konec vnitrni seznam 4. radek
       ) # konec BoxColumn
    ), # konec window

    MessageDialog['MD1'](
                    "Zadejte výraz pro provedení integrace.",
'title'="Informace",'type'='information')

  ) # konec maplet
); # konec display
```

["", "", ""] (6.8.1)

```
>
```

Další komponenty

V knihovně Maplets je mnoho dalších komponent. Nyní si je rychle probereme. Příklady jsou podobné těm z nápovědy.

```
> restart;
> with(Maplets[Elements]):
```

Elementy okna mapletu

Button - tlačítko

Tlačítko je určeno pro vyvolání určité funkce pomocí události onclick. Tlačítko má nápis, ale žádnou hodnotu narozdíl od tlačítka ToggleButton.

```
> Maplets[Display]( Maplet( [
      Button("OK", Shutdown())
  ] ) );
```

CheckBox

Tato komponenta je určena k vybírání dané položky pomocí zaškrtávacího tlačítka.

```
> Maplets[Display]( Maplet( [
      [
        CheckBox['ChB1']( 'caption'='Zaškrtni','value' =
  'true' )
      ],
      [
        Button("OK", Shutdown(['ChB1']))
      ]
  ] ) );
```

["true"] (6.9.1.2.1)

ComboBox

Rozbalovací seznam je určen k vybrání řetězce ze seznamu. Hodnoty jsou uloženy v seznamu v promněné **value**. Více informací lze najít v nápovědě.

```
> Maplets[Display]( Maplet( [
      ComboBox['CoB1']( 'value' = "modrá", ["červená",
  "oranžová", "žlutá", "zelená", "modrá", "fialová"] ),
      Button("OK", Shutdown(['CoB1']))
  ] ) );
```

["fsdfgsdgf"] (6.9.1.3.1)

Na rozdíl od ***DropDownBoxu*** je možné zadat i hodnotu, která není v seznamu.

DropDownBox

DropDown menu obsahuje seznam řetězců ze kterých může uživatel vybírat.

```
> Maplets[Display]( Maplet( [
      DropDownBox['DDB1']( 'value' = "Victoria", [
          "Victoria", "Edmonton", "Regina", "Winnipeg",
  "Toronto", "Quebec City",
          "Fredericton", "Halifax", "Charlottetown", "St.
  John's", "Whitehorse",
          "Yellowknife", "Iqaluit"] ),
      Button("OK", Shutdown(['DDB1']))
  ] ) );
```

["Fredericton"] (6.9.1.4.1)

Label

Popisek může obsahovat text a nebo obrázek, text ani obrázek nemůže být uživatelem změněn ani vybrán. Retězce jsou automaticky zalomeny.

```
> Maplets[Display]( Maplet( [
      "Standard text",
      Label( "Italicized text", 'font' = Font( helvetica,
  italic, 25 ) ),
      Button("OK", Shutdown())
  ] ) );
```

ListBox

ListBox umožňuje uživateli vybrat žadnou a nebo více položek ze seznamu. Pro vícenásobný výběr uživatelé podrží klávesy CTRL a nebo SHIFT. Výsledkem je seznam s jedním prvkem obsahující vybrané hodnoty. Pro jejich konverzi do prvků seznamu se použije funkce ***Maplets[Tools][ListBoxSplit]***.

```
> result := Maplets[Display]( Maplet( [
      ListBox['LB1']( 'value' = "Victoria", [
          "Victoria", "Edmonton", "Regina", "Winnipeg",
  "Toronto", "Quebec City",
          "Fredericton", "Halifax", "Charlottetown", "St.
  John's", "Whitehorse",
          "Yellowknife", "Iqaluit"] ),
      Button("OK", Shutdown(['LB1']))
  ] ) );
```

result := ["Regina,Halifax,Yellowknife"] (6.9.1.6.1)

```
> if result <> NULL then
      Maplets[Tools][ListBoxSplit]( result[1] );
  end if;
```

["Regina", "Halifax", "Yellowknife"] (6.9.1.6.2)

MathML Viewer

Tato komponenta je určena pro zobrazení MathML. Zatímco defaultní hodnota může být jiný výraz než validní MathML, každé další vložení však musí být validní MathML řetězec. K tomu tedy využije knihovnu ***MathML*** (?*MathML*). Pro export výrazu se používá funkce ***Export*** a pro import se používá ***Import***.

```
> Maplets[Display]( Maplet( [
      "Vložte výraz, který bude zobrazen jako MathML:",
      TextField['TF1'](),
      MathMLViewer['MMLV1']( 'value' = x^2 - 4*x + 3 ),
      [Button("Zobraz", Evaluate( 'MMLV1' = 'MathML[Export]
  (TF1)' ) ), Button("OK", Shutdown(['MMLV1']))]
  ] ) );
```

[(6.9.1.7.1)

```
""<math
    xmlns='http://www.w3.org/1998/Math/MathML'><semantics><mro\
    w xref='id5'><mo>∫</mo><mfenced><mrow xref='id4'><mi
    xref='id2'>x</mi><mo>+</mo><mi
    xref='id3'>y</mi></mrow></mfenced><mo>
    </mo><mrow><mo>d</mo><mi
    xref='id1'>x</mi></mrow></mrow><annotation-xml
    encoding='MathML-Content'><apply id='id5'><int/><bvar><ci
    id='id1'>x</ci></bvar><apply id='id4'><plus/><ci
    id='id2'>x</ci><ci
    id='id3'>y</ci></apply></apply></annotation-xml><annotatio\
    n
    encoding='Maple'>Int(x+y,x)</annotation></semantics></math>\
    ""]
```

Plotter

Tato komponenta je určena pro zobrazení 2-D a nebo 3-D grafu nebo animace.

```
> Maplets[Display]( Maplet( [
      "Vlož výraz, který bude vykreslen na x=0..10:",
```

```
    TextField['TF1']( 'value' = x^2 - 4*x + 3),
    Plotter['PL1']( 'value' = plot( x^2 - 4*x + 3, x=0..10
) ),
    [Button("Vykresli", Evaluate( 'PL1' = 'plot(TF1, x=0.
.10)' ) ), Button("OK", Shutdown())]
] ) );
```

RadioButton

Podobně jako CheckBoxLike uchovává hodnoty true nebo false. Jestliže je součástí tlačítkové skupiny (definované pomocí ButtonGroup), pak může mít pouze jedno hodnotu true.

```
> Maplets[Display]( Maplet( [
    [RadioButton['RB1']( "První", true, 'group' = BG1 ),
RadioButton['RB2']( "Druhé", 'group' = 'BG1' )],
    [Button("OK", Shutdown(['RB1', 'RB2']))]] ,
ButtonGroup['BG1']()) );
```

["false", "true"] (6.9.1.9.1)

Slider

Posuvník je určen pro ukládání a nastavování celých čísel. Vlastnost majorticks definuje zobrazení větších čar a vlastnost minorticks definuje zobrazení malých čar.

```
> Maplets[Display]( Maplet( [
    "Vyber celé číslo:",
    Slider['SL1']( 10, 0..20, 'majorticks'=10,
'minorticks'=2, 'showticks' ),
    Button("OK", Shutdown(['SL1']))
] ) );
```

["16"] (6.9.1.10.1)

Table

Tabulka je určena pro zobrazení textových informací v tabulce.

```
> IL := [sin(x), cos(x), tan(x), sec(x), csc(x), cot(x)]:

  Maplets[Display]( Maplet( [
    Table( ["Integrand", "Integral"], [seq( [i, int( i, x
)], i = IL )], 'width'=400 ),
    Button("OK", Shutdown())
] ) );
```

Text Box

TextBox je víceřídkové textové pole pro vstup, výstup nebo popisek. Obsah může být vybrán.

```
> restart;
> with(Maplets[Elements]):
  maplet := Maplet([
     ["Vlož text:  ", BoxCell(TextBox['IB1'](3..30),
  'as_needed')],
     [Button("OK", Shutdown(['IB1'])), Button("Cancel",
  Shutdown())]
  ]):
  Maplets[Display](maplet);
```

["as"] (6.9.1.12.1)

TextBox může být použit pro vstup a zobrazení textu.

```
> restart;
> with(Maplets[Elements]):
> Maplets[Display]( Maplet( [
      "Vlož výraz pro integraci dle x:",
      [TextBox['TF1']('value' = x^2 - 4*x + 3)],
      [Button("Integruj", Evaluate( 'TF1' = 'int(TF1, x)' )
  ), Button("OK", Shutdown(['TF1']))]
  ] ) );
```

["1/3*x^3-2*x^2+3*x"] (6.9.1.12.2)

Pro vytvoření jen výstupu užijeme vlastnost ***editable*** a nastavíme ji na false.

```
> restart;
> with(Maplets[Elements]):
  maplet := Maplet([
     [TextBox['IB1']('editable'='false', "Tento text je
  uvnitř TextBox. Nemůžete jej měnit.")],
     [Button("OK", Shutdown(['IB1'])), Button("Cancel",
  Shutdown())]
  ]):
  Maplets[Display](maplet);
```

["Tento text je uvnitř TextBox. Nemůžete jej měnit."] (6.9.1.12.3)

Text Field

Toto jednořádkové textové pole slouží pro vstup uživatele nebo zobrazení informací.

```
> restart;
> with(Maplets[Elements]):
> Maplets[Display]( Maplet( [
```

```
    "Vložte výraz pro integraci vzhledem k x:",
    TextField['TF1']( 'value' = x^2 - 4*x + 3),
    [Button("Integruj", Evaluate( 'TF1' = 'int(TF1, x)' )
), Button("OK", Shutdown(['TF1']))]
] ) );
```

["x^2-4*x+3"] (6.9.1.13.1)

Toggle Button

Podobně jako zaškrtávací políčko, přepínač i toggle button se využívá pro rozhodování mezi hodnota true a false.

```
> Maplets[Display]( Maplet( [
    [ToggleButton['RB1']( "1st", true ), ToggleButton
['RB2']( "2nd" )],
    Button("OK", Shutdown(['RB1', 'RB2']))]) );
```

["true", "false"] (6.9.1.14.1)

Dialogové elementy

Dialogy Mapletů mají předdefinovaný vzhled a není možné je nijak měnit (vzhledově ani měnit komponenty).

Alert Dialog

Dialog Alert umožňuje uživateli odpovědět na danou otázku bud pomocé OK a nebo Cancel. Je vrácena odpovídající hodnota true/FAIL.

```
> Maplets[Display]( Maplet( AlertDialog(
    "Omez x na kladná čísla - x > 0",
    'onapprove' = Shutdown('true'),
    'oncancel' = Shutdown("FAIL")
) ) );
```

"true" (6.9.2.1.1)

Color Dialog

Tento dialog umožňuje uživateli zvolit barvu a její hodnotu pak vrací.

```
> Maplets[Display]( Maplet( ColorDialog['CD1'](
    'onapprove' = Shutdown(['CD1']),
    'oncancel' = Shutdown()
) ) );
```

["#FF3399"] (6.9.2.2.1)

Confirm Dialog

Potvrzovací dialog je určen k vybrání hodnot Ano, Ne neno zruš (true/false/FAIL).

```
> Maplets[Display]( Maplet(ConfirmDialog( 'question', "Je x
  > 0 ?",
     'onapprove' = Shutdown('true'),
     'ondecline' = Shutdown('false'),
     'oncancel' = Shutdown("FAIL")
  ) ) );
```

"true" (6.9.2.3.1)

File Dialog

Tento dialog je určen k výběru souboru.

```
> Maplets[Display]( Maplet( FileDialog['FD1'](
     'onapprove' = Shutdown(['FD1']),
     'oncancel' = Shutdown()
  ) ) );
```

["C:\Documents and Settings\vladimir\Dokumenty\Bac3C4.bmp"] (6.9.2.4.1)

Input Dialog

Vstupní dialog je určen pro vložení hodnoty, která je vrácena jen v případě stisknutí OK.

```
> Maplets[Display]( Maplet(
     InputDialog['ID1']("Vlož číslo",
        'onapprove'=Shutdown(['ID1']),
        'oncancel'=Shutdown(),
        'title'="Titulek"
     )
  ) );
```

["5"] (6.9.2.5.1)

Message Dialog

Je určen k zobrazení nějakého nápisu.

```
> Maplets[Display]( Maplet( MessageDialog(
     warning,
     "Proměnná `x` bude smazána",
     'onapprove'=Shutdown(),
     'title'="Titulek"
```

```
) ) );
```

Question Dialog

Dialog pro výběr odpovědi Ano/Ne.

```
> Maplets[Display]( Maplet( QuestionDialog("Je x > 0?",
    'onapprove'=Shutdown('true'),
    'ondecline'=Shutdown('false'),
    'title'="Titulek"
) ) );
```

"true" (6.9.2.7.1)

Elementy pro tvorbu menu

Menu může být vloženo do komponenty Window. Následující příklad demostruje menu s několika položkami a oddělovači.

```
> restart;
> with(Maplets[Elements]):
> Maplets[Display]( Maplet(
    Window('title'="Integrace a derivace", 'menubar'='MB1',
["Vlož výraz a z menu vyber operaci:",[TextField['TF1']()],
Button("Konec", Shutdown("Zavřeno tlačítkem", ['TF1']))]),
    MenuBar['MB1'](
        Menu("Soubor", MenuItem("Close", Shutdown("Zavřeno z
menu", ['TF1']))),
        Menu("Operace",
            MenuItem("Integrace", Evaluate('TF1' = 'int(TF1,
x)') ),
            MenuSeparator(),
            MenuItem("Derivace", Evaluate('TF1' = 'diff(TF1,
x)'))
        )
    )
) );
```

"Zavřeno z menu", ["c"] (6.9.3.1)

Elementy pro nástrojovou lištu

Nástrojová lišta můžeme být vložena do komponenty Window. Následující ukázka obsahuje nástrojovou lištu tlačítek a oddělovačů..

```
> Maplets[Display]( Maplet(
      Window('title'='Titulek','toolbar'='TB1', [TextField
  ['TF1']()), Button("OK", Shutdown(['TF1']))]),
      ToolBar['TB1'](
          ToolBarButton("Integruj", Evaluate('TF1' = 'int(TF1,
  x)') ),
          ToolBarSeparator(),
          ToolBarButton("Derivuj", Evaluate('TF1' = 'diff(TF1,
  x)'))
      )
  ) );
```

["g*x"] (6.9.4.1)

Elementy pro provádění určité akce

K dispozici jsou také elementy, které jsou určeny pro provádění nějaké akce.

Close Window

Zavře okno.

```
> restart;
> with(Maplets[Elements]):
  maplet := Maplet('onstartup' = 'A1',
     Window['W1']("1",
        [Button("Otevři nové okno", RunWindow('W2')),
         Button("Konec", Shutdown("1"))]
     ),
     Window['W2']("2",
        [Button("Zavři toto okno", CloseWindow('W2')),
         Button("Konec", Shutdown("2"))]
     ),
     Action['A1'](RunWindow('W1'))
  ):
  Maplets[Display](maplet);
```

"1" (6.9.5.1.1)

Evaluate

Provede příkaz systému Maple.

```
> Maplets[Display]( Maplet( [
     ["Vložte výraz", TextField['TF1']('width' = 30)],
     [
        "Derivace vzhledem k x:",
        Button("Spočítej", Evaluate('TF1' = 'diff(TF1, x)'))
,
        Button("OK", Shutdown(['TF1']))
     ]
  ] ) );
```

["1"] (6.9.5.2.1)

RunDialog

Zobrazí dialog.

```
> Maplets[Display]( Maplet(
```

```
   Window([
      [TextField['TF1']()],
      [
         Button("Derivace vzhledem k x", Evaluate('TF1' =
'diff(TF1, x)')),
         Button("Nápověda", RunDialog('MD1')),
         Button("OK", Shutdown(['TF1']))
      ]
   ]),
   MessageDialog['MD1']("Pro více informací napište ?
diff",'title'="Titulek")
) );
```

["", ""] (6.9.5.3.1)

RunWindow

Zobrazí nové okno.

```
> restart;
> with(Maplets[Elements]):
  maplet := Maplet('onstartup' = 'A1',
     Window['W1']('title' = "Vyber metodu", 'layout' =
  'BL0'),
     BoxLayout['BL0'](
        BoxColumn(
           BoxRow("Vyber metodu"),
           BoxRow(
              Button("Derivace", RunWindow('W2')),
              Button("Integrace", RunWindow('W3'))
           )
        )
     ),
     Window['W2']('title'="Derivace", [
        [
           "Vlož výraz:",
           TextField['TF1']()
        ],
        [
           Button("Derivace vzhledem k proměnné x", Evaluate
  ('TF1' = 'diff(TF1, x)')),
           Button("Konec", Shutdown(['TF1']))
        ]
     ]),
```

```
   Window['W3']('title'="Integrace", [
      [
         "Vložte integrand:",
         TextField['TF2']()
      ],
      [
         Button("Integrace dle proměnné x", Evaluate('TF2'
= 'int(TF2, x)')),
         Button("Konec", Shutdown(['TF2']))
      ]
   ]),
   Action['A1'](RunWindow('W1'))
):
Maplets[Display](maplet);
```

["1"] (6.9.5.4.1)

SetOption

Nastavuje hodnotu komponenty mapletu na hodnotu ***value***.

```
> Maplets[Display]( Maplet( [
      "Vepiš nějaký text:",
      TextField['B1'](20),
      TextField['B2'](20, 'editable'='false'),
      [
         Button("Smaž první pole", SetOption('target' = 'B1',
'value' = "")),
         Button("Zkopíruj druhé pole", SetOption('target' =
'B2', Argument('B1'))),
         Button("Vrať hodnotu do 2. pole", Shutdown(['B2']))
      ]
] ) );
```

Shutdown

Funkce je určena k ukončení mapletu.

```
> Maplets[Display](Maplet([[Button("Zavři", Shutdown())]]));
```

Elementy pro uspořádání

Box Layout

Box layout umožňuje dávat objekty do řádků a nebo sloupců. V podstatě jde o tabulku. BoxLayout lze také zapisovat jen pomocí seznamu.

```
> Maplets[Display]( Maplet(
      Window(["A", [["B", "C"], "D", [["E", "F", "G"], "H",
"I"], "J"], Button("OK", Shutdown())])
) );
```

Grid Layout

Grid layout umožňuje zarovnání jak horizontální tak vertikální..

```
> Maplets[Display]( Maplet(
      Window([GridLayout( [["A", "B"], ["C", "D"], ["E",
"F"]] ), Button("OK", Shutdown())])
) );
```

Tvorba pomocí Maplet Builderu

Jednodušší Maplety lze vytvářet pomocí Maplet Builderu, který je k dispozici v nabídce Tools -> Assistants -> Maplet Builder. Jde o aplikaci, která poskytuje vývojářům prostředí pro vývoj mapletů. Lze v něm vytvářet kompletní layout a všechny klíčové komponenty. Jen jsou tam některá omezení.