Služby Kernelu

Jan Krček (jan.krcek@matfyz.cz)

Úvod

Tato referenční příručka obsahuje popis služeb kernelu, metody a atributy známých jmen a popis konstant.

S novými verzemi kernelu mohou přibývat nové služby a známá jména.

Služby Kernelu

Práce s kernelem

Jména

```
int @IsFNMemberOfSN(name first,name second)
int @IsFOMemberOfSN(objptr first_obj, name second)
int @IsFNMemberOfSO(name first, objptr second_obj)
int @IsFOMemberOfSO(objptr first_obj, objptr second_obj)
```

Služby zjišťují závislosti mezi jmény. Všechny vyhodnocují relaci `first <= second`. Je možno kombinovat objekty a jména. Když je uveden objekt, zjistí se jeho typ a ten se pak porovnává.

Ladění

```
void @LogUserError(int group, int info1 = 0, char *str = 0)
void @LogDebugInfo(int num , int info1 = 0, char *str = 0)
```

`group` říká, do které skupiny má error patřit (`@eKEGPanicError`, `@eKEGWarning`, ...), `num` si uživatel může zvolit. Další parametry by měly obsahovat popis chyby, jsou nepovinné.
Vyvolání erroru ze skupiny `@eKEGPanicError` způsobí okamžité ukončení kernelu a tím i skriptů. I některé jiné errory mohou vyvolat ukončení kernelu (záleží na nastavení).

Objekty a objektové tagy

```
int @ExistsObj(objptr obj)    // zjistí, zda objekt existuje (zda je živý)
```

Každý objekt má skrytý atribut TAG. Přistupovat k němu lze pomocí následujících služeb. Standardní objekt Mapa pro hru Krkal tento tag využívá a předpokládá, že je nastaven na 0. (0 je iniciální hodnota tagu)

```
void @SetTag(objptr obj)      // nastaví tag na 1
void @ClearTag(objptr obj)    // vynuluje tag
int @CheckTag(objptr obj)     // vrátí hodnotu tagu
```

Hra

```
void @TerminateKernel()       // na konci kola bude ukončen kernel
void @GameVictory()           // Nahlásí kernelu,že level byl úspěšně dohrán
                              // ve hře je možno pokračovat, dokud nebude
                              // ukončena
int @GetKernelTime()          // vrátí kernelí čas. Během taktu se nemění.
```

```
int @GetRunMode()             // vrací typ hry: @NORMAL_RUN, @EDITOR_RUN
int @IsGame()                 // Zda je typ hry @NORMAL_RUN
int @IsEditor()               // Zda je typ hry EDITOR_RUN
```

Kopírování

```
inta @CopyIntA(inta a);
chara @CopyCharA(chara a);
doublea @CopyDoubleA(doublea a);
objptra @CopyObjPtrA(objptra a);
namea @CopyNameA(namea a);
```

Služby vytvoří a vrátí kopii kernelího pole. Pokud pole bylo Null, vrátí Null.

```
objptr @CopyObject(objptr object);
```

Služba vytvoří objekt, zkopíruje do něj data a zavolá `cconstructor`. Pak vrátí `objptr` na okopírovaný objekt.

Práce s Mapou

Umísťování a odebírání

```
int @IsObjInMap(objptr obj)         // zjisti zda je objekt umisten v mape
void @PlaceObjToMap(objptr obj);    // umisti objekt do mapy
void @PlaceObjToMapKill(objptr obj);
      // před umístěním zničí všechny kolidující objekty. Umístí objekt do mapy.
void @RemoveObjFromMap(objptr obj); // odebere objekt z mapy
```

Pohyby

```
void @MoveObjTo(objptr obj, int x, int y, int z)
      // okamzite posune objekt na nove absolutni souradnice.
void @MoveObjRel(objptr obj, int dx, int dy, int dz)
      // okamzite posune objekt relativne, zadanym smerem.
void @InitMoveTo(objptr obj, int time, int dx, int dy, int dz=0)
      // Nastartuje plynulý pohyb. Ten bude trvat po čas time.
      // Souřadnice jsou zadávány relativně
```

První dvě funkce ruší případný probíhající plynulý pohyb

```
void @MvConnectObjs(objptr obj1, objptr obj2);
      // propojí objekty tak, že se budou hýbat spolu. Plynulý pohyb se kopíruje z druhé skupiny na první
void @MvDisconnectObj(objptr obj);        // odpojí objekt od skupiny.
                                          //Odebrání z mapy automaticky odpojí objekt
```

Kolize

```
int @IsCellInMap(int cx, int cy, int cz)       // 1 Ano, 0 Ne
int @IsPixelInMap(int x, int y, int z)         // 1 Ano, 0 Ne
void @DeleteOutOfMapCells(inta Cells);
      // z pole odstraní buňky, které jsou mimo mapu. Buňky jsou popsány trojicemi souřadnic x,y,z
```

Následující funkce testují kolize s objektem. Parametry `dx`, `dy`, `dz`, nastavujeme relativní posun objektu. Ptáme se pak, jak objekt koliduje, kdyby byl posunut o tyto relativní souřadnice. Testovat kolize můžeme i u objektů neumístěných v mapě.

```
inta @FindCollidingCells(objptr Obj, inta OutPutBuff, int dx=0, int dy=0,
int dz=0);
      // Služba v poli OutPutBuff (je třeba zadat vytvořené pole) vrátí seznam buněk (trojice souřadnic x y z),
      // do kterých objekt zasahuje.
```

```
int @IsObjInCollision(objptr obj, int dx=0, int dy=0, int dz=0);  //1 Ano
objptra @FindCollidingObjs(objptr obj, objptra ret=0, int dx=0, int dy=0,
int dz=0)
      // Vrati seznam objektu se kterymi objekt koliduje. V promenne ret
      // muzes urcit, kam se bude vracet, jinak fce pole alokuje
```

Souřadnice

```
int @FindObjCell(objptr obj, int *cx, int *cy, int *cz)
```
// ze souřadnic objektu zjistí jeho primární buňku. Vráti 0, když je buňka mimo mapu.
```
int @FindCell(int x, int y, int z, int *cx, int *cy, int *cz)
```
// Převede souřadnice (x,y,z) v pixelech na souřadnice (cx,cy,cz) v buňkách.
// Vráti 0, když je buňka mimo mapu.
```
void @FindCellPosition(int cx, int cy, int cz, int *x, int *y, int *z)
```
// V parametrech (x,y,z) vrátí centrální pozici buňky (cx,cy,cz)

```
void @ReadObjCoords(objptr obj, int *x, int *y, int *z)
      // precte z objektu souradnice ma-li je (jinak vraci nuly).
void @WriteObjCoords(objptr obj, int x, int y, int z=0);
```
// změní souřadnice objektu, má-li je. Umístěný objekt v mapě bude přemístěn voláním `@MoveObjTo`

Automatismy

```
void @ResetAuto(objptr obj, int ResetRandom=0, int ResetConnection=0)
```
// Řekne kernelu, aby přepočítal automatismus. `ResetRandom` říká, že se automatismus nemá
// rozhodovat podle stejných náhodných čísel, jako se rozhodoval posledně.
// `ResetConnection` donutí objekt, aby se znovu pokusil navázat na vše kolem
```
name @GetActivAuto(objptr obj)       // Vrátí jméno grafiky, kterou objekt aktuálně používá
name @GetDefaultAuto(objptr obj)           // Vrátí defaultní automatismus
name @GetDefaultAuto2(name ObjType)        // Vrátí defaultní automatismus
```

Vyhledávání objektů v oblasti

Následující funkce pracují na buňkách. Prohledávají obdélníkovou oblast buněk. Ta může být zadaná buď relativně k objektu `obj`, pokud je `obj` zadán. Pokud je `obj` null, tak se oblast bere absolutně.

Funkce vyhledají na oblasti všechny objekty z množiny `set`. Pokud ta není zadána nebo je null, tak se vyhledají všechny objekty.

```
void @FindObjsInArea(objptra output, objptr obj, name set=0, int x1=0, int
y1=0, int x2=0, int y2=0, int z=0)
```
// najde objekty, které jsou z množiny set, v buňkách na dané oblasti. set nemusí být zadán.
```
void @AreaCall(name Method, objptr obj, name set=0, int x1=0, int y1=0, int
x2=0, int y2=0, int z=0)
```
// naleznou se objekty v oblasti a zavolá se na ne metoda `Method`
```
void @AreaMessage(name Method, objptr obj, name set=0, int x1=0, int y1=0,
int x2=0, int y2=0, int z=0);
```
// naleznou se objekty v oblasti a pošle se jim zpráva

Pro objekt Mapa

```
void @RegisterMap(int CellType, int leftx, int lefty, int rightx, int
righty, int lowerlevel, int upperlevel, int cellsizeX, int cellsizeY, int
cellSizeZ)
```

Tuto funkci volá objekt Mapa ve svém konstruktoru a předává v ní kernelu informace o mapě. `CellType` určuje typ buňky (zatím jsou přípustné jen čtverce). `leftx`, `lefty` je levý horní roh mapy v pixelech. `rightx`, `righty` je pravý dolní roh. `lowerlevel` a `upperlevel` určují nejnižší a nejvyšší patro v buňkách. `cellsizeX`, `cellsizeY` a `cellSizeZ` udávají rozměry buňky v pixelech.

```
void @MapGetNumberOfCells(int *sizex, int *sizey, int *startx, int *starty)
       // vrati indexy prvni bunky a pocty bunek. vse jen v osach x a y
```

Obecné služby

```
double @rand(double mez)        // reálné náhodné číslo z intervalu <0,mez>
double @randExc(double mez)     // reálné náhodné číslo z intervalu <0,mez)
int @randInt(int mez)           // přirozené náhodné číslo z intervalu <0,mez>

double @sqrt(doubla a)          // spočítá odmocninu z a
int @round(double a)            // zaokrouhlí a na celá čísla
```

Světla, zvuky, vstup, scrolling

```
int @AddLight(int x, int y, int z, char ir, char ig, char ib, int radius )
       // (x,y,z) pozice, (ir,ig,ib) intenzita, radius poloměr
       // vrátí číslo světla, které je třeba si zapamatovat pro případ, že bychom světlo chtěli zrušit.
void @DeleteLight(int lightNum);      // Ruší světlo daného čísla
void @SetTopLightIntenzity(char r, char g, char b);   // Nastaví barvu globálního světla
void @GetTopLightIntenzity(char &r, char &g, char &b);// Zjistí barvu globálního světla

int @PlaySound(name soundname, int x, int y, double volume=1)
       // přehraje zvuk, volume je 0..1, x,y souřadnice v pixelech, kde má zvuk hrát.
       // soundname určuje jméno zvuku. Musí to být jméno typu void a v konfiguračním souboru zvuků
       // musí být k tomuto jménu přiřazen příslušný zvuk (cesta k souboru se zvukem v nějaké pakáži).

int @IsKeyDown(name klavesa)
       // Test, zda je zmáčknuta klávesa. 1 ano, 0 ne.
       // klavesa určuje jméno klávesy. Musí to být jméno typu void a v konfiguračním souboru
       // keyboard.cfg musí být k tomuto jménu přiřazen kód příslušné klávesy
int @IsAnyKeyDown()       // Je nějaká klávesa zmáčklá? 1 ano, 0 ne.

void @SetScrollCenter(int x, int y);
       // nascrolluje tak, aby bod x,y bul ve středu herního okna
void @WindowScroll(int dx, int dy,int time);
       // zapne plynulé scrollovaní okna. Za čas time se okno posune na nový střed, posunutý o dx, dy
```

Poznámka: Zadávaný střed nemusí být středem skutečným. Scrolling se zaráží na okraji mapy. Jinak střed znamená to místo, kolem kterého chceme obraz vycentrovat. Jen na okrajích to dopadne jinak.

```
void @CalcScrollDistance(int x, int y, int *dx, int *dy);
       // Vstup x,y je počáteční bod scrollingu, cílový bod je posunut o *dx, *dy.
       // Protože scrollovani se zaráží na okrajích levlu, tak ve skutečnosti může být scrollovací vzdálenost
       // kratší než zadané *dx a *dy. Funkce tedy případně *dx a *dy zmenší.
```

Save Load

Metody pro nahrávání dat ze streamu se dají volat jen v `lconstructoru` a v metodách, které zavolal přímým voláním. Metody pro ukládání zase jen z funkce `@ESaveMe`. Metody uloží nebo nahrají příslušný datový typ a zároveň nastaví pozici ve streamu na následující položku. Lze tedy číst a zapisovat sekvenčně.

Práce se streamem

```
void @SLSeek(int pos);  // Nastaví pozici ve streamu. počítá se v Bytech a první data mají index 0
void @SLSeekToEnd();    // Nastaví pozici za poslední data
int @SLGetPos();        // Vrátí aktuální pozici
int @SLEof();           // Vrátí jedna pokud už jsem na konci dat (nemůžu už nic číst)
```

Save

```
void @SaveInt(int a);
void @SaveChar(char a);
void @SaveDouble(double a);
void @SaveObjPtr(objptr obj);
void @SaveName(name jmeno);
void @SaveString(string str);

void @SaveIntA(inta a);
void @SaveCharA(chara a);
void @SaveDoubleA(doublea a);
void @SaveObjPtrA(objptra a);
void @SaveNameA(namea a);
```

Poznámka: Pole mohou být uložena jako prázdná (ale vytvořená), plná, či nevytvořená (null). Při nahrávání kernel pole pro objekt vytvoří a předá mu ho. Pokud bylo null, předá null.

Load

```
int @LoadInt();
char @LoadChar();
double @LoadDouble();
objptr @LoadObjPtr();
name @LoadName();
void @LoadString(string ret str);

inta @LoadIntA();
chara @LoadCharA();
doublea @LoadDoubleA();
objptra @LoadObjPtrA();
namea @LoadNameA();
```

Herní menu

Služby vrací 1 v případě úspěchu nebo 0 v případě neúspěchu

```
int @MnuSetBar(int index, int progress);
      // nastaví pruh s indexem 0 nebo 1 na progress, což je číslo od 0 do 100 procent
int @MnuSetItem(int index, int count);
      // nastaví počet count u položky s indexem index
void @MnuRefresh();
      // Je třeba zavolat na konci přidávání, ubírání položek, po přidání pruhů
int @MnuDeleteItem(int index);
      // Smaže položku daného indexu
int @MnuAddItemN(name item);
      // Přidá položku a vrátí její index. item musí být jméno objektového typu. Z toho služba zjistí obrázek
int @MnuAddItemO(objptr obj);
      // Přidá položku a vrátí její index. Obrázek položky je zjištěn z grafiky objektu obj
      // při hledání grafiky se počítá automatismus. Ten bere ohled na případné atributy
      // ovlivňující automatismus.
int @MnuAddBars(int color1, int color2);
      // přidá dva pruhy příslušných barev ve formátu 0xAARRGGBB
```

Programování editačních položek

Následující služby lze volat jen z konstruktoru `scripted` položky a z metod, které zavolal button, jako svou obsluhu, když byl zmáčknut. Přípustné je i volání z metod vnořených přímým voláním.

Není-li uvedeno jinak, tak služby vrací 1 – úspěch, 0 – neúspěch.

Editační pole

Následující služby položku připraví. Nejprve je třeba ji vytvořit, pak je možné ji modifikovat a nakonec je třeba ji umístit. Funkce musí jít postupně po sobě a musí pracovat stále s tou stejnou položkou, dokud není umístěna.

vytvoření

```
void @ECreateInt(int *data, char *label, char *comment=0)
void @ECreateChar(char *data, char *label, char *comment=0)
void @ECreateDouble(double *data, char *label, char *comment=0)
void @ECreateName(name *data, char *label, char *comment=0)
void @ECreateObjPtr(objptr *data, char *label, char *comment=0)

void @ECreateIntA(inta *data, char *label, char *comment=0)
void @ECreateCharA(chara *data, char *label, char *comment=0)
void @ECreateDoubleA(doublea *data, char *label, char *comment=0)
void @ECreateNameA(namea *data, char *label, char *comment=0)
void @ECreateObjPtrA(objptra *data, char *label, char *comment=0)
```

Službám se předává adresa dat, které má editační pole modifikovat. Za dobu existence editačního pole nelze s daty hýbat. `label` určuje popisek editační položky. A `comment` volitelnou nápovědu (tooltip)

modifikace

```
void @ESetVarTags(int tags,int filter=-1);
      // tags ovlivňuje chování editačního pole a přípustnost určitých hodnot.
      // Viz konstanty začínající @eKVUB. filter je pro KSID jména, filtruje jejich typ.
```

umístění

```
int @EPlaceItem(int where=0, int before=0);
      // umístí vytvořenou položku na index where. before 0 říká umísti za index,
      // before 1 říká umísti před index
      // where==0 && before==0 => pridani na konec seznamu
      // where==0 && before==1 => pridani na zacatek seznamu
      // Vrátí index nové položky
```

Další ovládací prvky

```
int @ERefresh();                                  // nepoužívat, zastaralé
int @EAddGap(int where=0, int before=0);  // přidá oddělovací čáru, vrací index
int @EDeleteItem(int index);                  // smaže položku na indexu
int @EAddButton(int where, int shift, char* label, char* help, int userID,
name scriptFunction);
      // Služba vytvoří button na pozici where. shift může být -1 před, 0 na, 1 za
      // userID je číslo, které bude dostávat obslužná metoda při zmáčknutí tohoto bottonu
      // scriptFunction je KSID jméno obslužné metody.
      // Služba vrací buttonID
int EDeleteButton(int buttonID);     // odstranění buttonu buttonID
int @EAddText(char* text, char* help=0, int where=0, int before=0)
      // přidá text text, vrací index
```

Práce s podstromy

```
int @EAddGroupItem(char* label, char* help, int where=0, int before=0);
      // vytvori novou skupinu (novy "adresar" pro editacni polozky)
      // urceni pozice funguje stejne jako u EPlaceItem
      // vrati index skupiny (groupIndex) - 1..n
      // groupIndex==0 ma korenova polozka
int @EDeleteGroupItem(int groupIndex);
      // smaze skupinu s groupIndex vcetne vsech jejich polozek
```

```
int @ESelectGroupItem(int groupIndex);
      // vybere skupinu, se kterou se bude dale pracovat
      // groupIndex==0 znamena praci v korenove polozce
int @EDeleteAllGroupItems(int groupIndex, int buttonsLet=0);
      // smaze vsechny polozky ve skupine groupIndex
      // buttonsLet urcuje, zda ponecha tlactika (==1) nebo nikoliv (==0, tzn. smaze vsechno)
      // samotna skupinova polozka (adresar) se nemaze
      // groupIndex==0 => smaze vsechno krome korenove polozky odpovidajici skriptovano promenne
```

Vytvoření skupinové položky

Funkce slouží pro vytvoření editační položky na zadávání skupin. (souřadnic, oblastí, ...) Položka je nejen vytvořena, ale i umístěna. Funkce vrací index.

```
int @EAdd2DCell(int *x, int *y, char *label, char *comment=0,int where=0,
int before=0)
int @EAdd3DCell(int *x, int *y, int *z, char *label, char *comment=0,int
where=0, int before=0)
int @EAdd2DPoint(int *x, int *y, char *label, char *comment=0,int where=0,
int before=0)
int @EAdd3DPoint(int *x, int *y, int *z, char *label, char *comment=0,int
where=0, int before=0)
int @EAdd2DAreaP(int *x1, int *y1, int *x2, int *y2, char *label, char
*comment=0,int where=0, int before=0)
int @EAdd3DAreaP(int *x1, int *y1, int *z1, int *x2, int *y2, int *z2, char
*label, char *comment=0,int where=0, int before=0)
int @EAdd2DAreaC(int *x1, int *y1, int *x2, int *y2, char *label, char
*comment=0,int where=0, int before=0)
int @EAdd3DAreaC(int *x1, int *y1, int *z1, int *x2, int *y2, int *z2, char
*label, char *comment=0,int where=0, int before=0)
```

Metody Známých jmen

Všechny metody známých jmen jsou typu `safe`.

```
Constructor()           // volá se při vytvoření nového objektu
LoadConstructor()       // volá se při nahrávání objektu z levlu.
                        // Objekt má otevřen stream pro nahrávání dat
CopyConstructor()       // Volá se u kopie po okopírování objektu.
                        // Kopie má vyplněny atributy stejně jako originál
Destructor()            // Volá se před zničením objektu
void @ESaveMe()         // Volá se během ukládání levlu
                        // Objekt má otevřen stream pro ukládání dat

void @MapPlaced()       // Objekt byl právě umístěn do mapy. Volá se jako zpráva.
void @MapRemoved()      // Volá se těsně před odebráním objektu z mapy

int retor @TestCollision(objptr @Object)
      // Kernel volá tuto metodu, když chce zjistit, zda objekt @Object ze skupiny @clzFceGr
      // koliduje 1, či ne 0
void @CollisionKill()
      // tato metoda je volána objektům, které chce kernel zabít kolizí. Viz služba @PlaceObjToMapkill.
      // Pokud objekt tuto metodu nemá implementovánu, bude převedena na Destructor.

void @MoveEnded()       // Volá se, když skončil plynulý pohyb
```

```
void @TriggerOn(objptr @Object, name @ObjType)
void @TriggerOff(objptr @Object, name @ObjType)
      // Toto jsou zprávy, které posílá trigger. @Object je pointer na objekt, který událost vyvolal
      // @ObjType je typ objektu, který událost vyvolal

void <konstruktor scripted polozky>(int @ItemID)
      // Volá editor, když chce, aby skript zadal editační položky. @ItemID je uživatelský parametr, který se
      // zadává ve scripted položce
void <reakce na button>(int @ButtonID, int @ButtonUserID, int @GroupID)
      // Volá GUI, když byl zmáčknut button. Předává jeho @ButtonID a číslo stromové skupiny
      // @GroupID. @ButtonUserID přiřadil k buttonu uživatel při jeho konstrukci.
```

Metody povinné pro objekt Mapa

```
void @MPlaceObjToMap(objptr @Object, inta @CellsArray)
```

Tuto metodu volá kernel, když přidává objekt `@Object` do mapy. V `@CellsArray`, předává kernel mapě seznam buněk, do kterých objekt zasahuje. V poli jsou předávány souřadnice buněk, pro každou buňku je tam trojice hodnot: x, y, z.

```
void @MRemoveObjFromMap(objptr @Object, inta @CellsArray)
```

Tuto metodu volá kernel, když oznamuje mapě, oby si objekt odebrala ze své datové struktury. V `@CellsArray` je opět seznam buněk, ve kterých se objekt nachází.

```
void @MMoveObjInMap(objptr @Object, inta @RemoveCellsArray, inta
@KeepCellsArray, inta @PlaceCellsArray)
```

Tuto metody volá kernel poté, co objektem `@Object` bylo posunuto na nějaké nové souřadnice. V `@RemoveCellsArray` jsou buňky, které objekt opustil; v `@KeepCellsArray` jsou buňky, kde objekt zůstává, a v `@PlaceCellsArray` jsou buňky, které objekt nově zabral. V budoucnu plánuji této funkci předávat i staré a nové souřadnice objektu. Tohle, ale zatím implementováno není.

```
objptra @MGetObjects(inta @CellsArray, objptra @ObjectArray)
```

Tato funkce slouží ke zjišťování objektů na daných buňkách. Funkce by měla vrátit seznam všech objektů, které se vyskytují na buňkách z `@CellsArray`. Pokud je `@ObjectArray` null, funkce by měla vytvořit pole typu `objptra` a to vrátit. Jinak by funkce měla pole z `@ObjectArray` vyprázdnit, uložit do něj výsledek a vrátit `@ObjectArray`.

```
void @MResizeMap()
```

Tuto funkci volá kernel, pokud v editoru dojde ke změně velikosti mapy. Mapa by měla zrušit svou datovou strukturu. Vytvořit ji znova prázdnou a znovu se zaregistrovat. Další krok provádí kernel. Vezme všechny objekty, které byly umístěny do mapy. Pokud je stále možné umístit je do mapy, umístí je (volá `@MPlaceObjToMap`. Objekt se o této změně nedozví, `@MapPlaced` zavolána není). Objekty, které se ocitly mimo mapu, kernel zničí.

Atributy Známých jmen

```
void @nonevar

char @CollisionCfg      // Informace o tom, jak vypadá kolizní oblast.
                        // Viz konstanty začínající @eKCC
                        // defaultní hodnota je @eKCConeCell | @eKCCwall

int @NumCellZ           // U obdélníkové kolizní oblasti: Rozměry obdélníka v buňkách
int @NumCellX
int @NumCellY
```

```
int @BCubeX1            // U kolizní krychle: Relativní umístění krychle kolem objektu. V pixelech
int @BCubeY1
int @BCubeZ1
int @BCubeX2
int @BCubeY2
int @BCubeZ2

int @ObjPosX            // Objektové souřadnice v pixelech
int @ObjPosY
int @ObjPosZ
```

Poznámka: Nastavení kolizní oblasti a souřadnice se nedají měnit, když je objekt umístěn. (Tohle pravidlo neplatí pro kolizní skupiny)

```
name @clzAddGr          // Množina se kterou objekt koliduje. Defaultně @Everything
name @clzSubGr          // Množina se kterou objekt nekoliduje. Defaultně null
name @APicture          // Přiřazená automatická grafika. Defaultně @DefaultAuto
int @ANoConnect         // Bitová maska. Na které směry grafika objektu nenavazuje?
                        // defaultně 0, objekt navazuje všemi směry
name @clzFceGr          // U kterých objektů je nutno kolizi ještě dodatečně testovat funkcí
                        // @TestCollision? Defaultně null

objptr @MsgRedirect     // U triggerů. Kam mají být přesměrovány zprávy typu @TriggerOn,
                        //@TriggerOff? Defaultně null.
```

Objekty Známých jmen

`@DefaultObject`

Kernel vytvoří objekt tohoto typu, když měl tvořit objekt typu null nebo neplatného typu

`@ScrollObj`

Scrollovat můžeme buď přímo přes služby kernelu a nebo můžeme vytvořit a umístit do mapy speciální objekt typu `@ScrollObj`. Jak se tento objekt bude pohybovat po mapě, bude s ním scrollovat i okno. A to tak, aby se objekt držel ve středu okna. Nejlepší je nastavit `@ScrollObj` tak, aby s ničím nekolidoval, aby byl neviditelný (nadat mu grafiku nebo nastavit do kolizní konfigurace `@eKCCinvisible` bit) a pak ho propojit s postavičkou, která se má držet ve středu okna.

Konstanty

Konstanty typu name

```
@DefaultAuto        // jméno pro defaultní grafiku. Jaká to bude konkrétně, záleží na objektu a na
                    // stylu levlu. Každý objekt s grafikou má nějakou defaultní grafiku.
                    // Defaultní grafika také bývá nejčastěji používána
@Everything         // Množina všeho. Jen @Everything nepatří do @Everything.
@Nothing            // Prázdná množina
```

Číselné konstanty

Nastavení typu kolize

```
@eKCCpoint          // Jeden bod. Objekt koliduje jen pokud je bod ostře uvnitř jiného objektu
@eKCConeCell        // Jedna buňka
@eKCCrect1          // Obdélník n x m buněk
```

```
@eKCCcolCube        // Kolizní krychle, zadávají se její rohy v pixelech a relativně k objektu
@eKCCoutOfMap       // Objekt nekoliduje ani s mapou ani s jinými objekty, přesto může mít grafiku
                    // a může být umísťován na libovolné souřadnice (i za hranicemi mapy)
@eKCCtriggerBit     // Zda se jedná o trigger
@eKCCwall           // zapíná kolizi se stěnami
@eKCCfloor          // zapíná kolizi s podlahami
@eKCCcell           // koliduje i se stěnami i s podlahami
@eKCCnothing        // nekoliduje s žádnými objekty jen s mapou
@eKCCcenterColBit   // Pro triggery. U druhého objektu se netestuje jeho kolizní oblast,
                    // ale pouze bod – jeho souřadnice
@eKCCinvisible      // Pro netriggery. Objekt je neviditelný.
```

Typ běhu kernelu

```
@NORMAL_RUN
@EDITOR_RUN
```

Tvary buněk v mapě. Zatím jsou přípustné jen čtverce

```
@eKCTctverce
@eKCTkosoctverce  // Nepoužívat
@eKCTplocheHexy   // Nepoužívat
@eKCTspicateHexy  // Nepoužívat
```

Druhy běhových chyb

```
@eKEGPanicError   // Způsobuje okamžité ukončení kernelu
@eKEGFatalError   // Podle běžného nastavení, těchto chyb může být nejvýše 10. Pak ukončení kernelu.
@eKEGError
@eKEGWarning
@eKEGInfo
@eKEGParamConversionError
@eKEGParamAssignationError
@eKEGCallingError
@eKEGDebug
@eKEGMapError
```

Konstanty ovlivňující editaci proměnných

```
// Tyto bity ovlivňují editaci proměnné
@eKVUBexclusive         // Intervaly jsou neostré
@eKVUBplannarNames      // jména se neřadí do stromu, ale jsou vidět všechny najednou
@eKVUBincludeNull       // Jako hodnotu dovolím zadávat i 0 a null
@eKVUBobjInMap          // Pro objekty a jména typu object. Zda musí mít nastaven InMap tag
@eKVUBobjOutMap         // Pro objekty a jména typu object. Zda musí mít nastaven OutMap tag

// konstanty, jejichž skládáním zle konfigurovat filtr na hodnoty typu name
@eEdNTvoid
@eEdNTobject
@eEdNTmethod
@eEdNTparam
@eEdNTeverything

// Konfigurace intervalu. zatím nevyužíváno
@eKLCnone
@eKLCinterval
@eKLClist
@eKLCup
@eKLCdown
```