

[Help](#)

```

#if defined(PremiaCurrentVersion) && PremiaCurrentVersion < (2008+2) //The "#els
#else
/*****
*   CPS - A simple C PDE solver                               *
*                                                           *
*   Copyright (c) 2007,                                       *
*   Maya Briani      <m.briani@iac.rm.cnr.it>,               *
*   Francesco Ferreri <francesco.ferreri@gmail.com>,         *
*   Roberto Natalini <r.natalini@iac.rm.cnr.it>,             *
*   Marco Papi       <m.papi@iac.rm.cnr.it>                  *
*                                                           *
*****/
#include "cps_boundary_description.h"
#include "cps_function.h"
#include "cps_grid.h"
#include "cps_grid_node.h"
#include "cps_utils.h"
#include "cps_assertions.h"

#define VALID_BOUNDARY_TYPE(T) \
(T == BOUNDARY_INITIAL || \
T == BOUNDARY_LEFT || \
T == BOUNDARY_INITIAL )

int boundary_description_create(boundary_description **descr)
{
    unsigned int dim;
    STANDARD_CREATE(descr, boundary_description);
    (*descr)->initial = NULL;
    for (dim = X_DIM; dim < MAX_DIMENSIONS; dim++)
    {
        (*descr)->left[dim] = NULL;
        (*descr)->right[dim] = NULL;
    }

    return OK;
}

```

```
int boundary_description_destroy(boundary_description **descr)
{
    /* destroy boundary, related functions cannot be destroyed
       altogether, since multiple pointers to same area can be present */

    STANDARD_DESTROY(descr);
    return OK;
}
```

```
int boundary_description_set_left(boundary_description *descr, unsigned int dim,
{
    /* set left boundary function */
    REQUIRE("decription_not_null", descr != NULL);
    REQUIRE("function_not_null", f != NULL);
    REQUIRE("valid_dimension", dim >= X_DIM && dim < MAX_DIMENSIONS);

    descr->left[dim] = f;

    return OK;
}
```

```
int boundary_description_set_right(boundary_description *descr, unsigned int dim,
{
    /* set right boundary function */
    REQUIRE("decription_not_null", descr != NULL);
    REQUIRE("function_not_null", f != NULL);
    REQUIRE("valid_dimension", dim >= X_DIM && dim < MAX_DIMENSIONS);

    descr->right[dim] = f;

    return OK;
}
```

```
int boundary_description_set_initial(boundary_description *descr, function *f)
{
    /* set payoff boundary function */
    REQUIRE("decription_not_null", descr != NULL);
    REQUIRE("function_not_null", f != NULL);

    descr->initial = f;
}
```

```
    return OK;
}

double boundary_description_evaluate(boundary_description *descr, const grid *gr
{
    int dim;
    double result = 0.0;

    /* evaluate boundary on given side */
    REQUIRE("description_not_null", descr != NULL);
    REQUIRE("node_belongs_to_boundary_or_is_initial", grid_node_is_boundary(node)

    if (grid_node_is_initial(node))
    {
        result = cps_function_evaluate(descr->initial, node);
        return result;
    }

    for (dim = X_DIM; dim <= grid->space_dimensions; dim++)
    {
        if (grid_node_is_left_boundary(node, dim))
        {
            result = cps_function_evaluate(descr->left[dim], node);
            return result;
        }
        else if (grid_node_is_right_boundary(node, dim))
        {
            result = cps_function_evaluate(descr->right[dim], node);
            return result;
        }
    }
    return result;
}

/* end -- boundary_description.c */

#endif //PremiaCurrentVersion
```