```
/*
```

ORKID OPERATION DECLARATIONS

This section of the the **ORKID** C language binding is the largest and contains function declarations for all the operations defined in the main **ORKID** standard, and is subdivided according to the subsections in this standard.

Each subdivision contains a list of function declarations and a list of symbol definitions. The function names have been kept to six characters for the sake of linker compatibility. Of these six characters, the first two are always 'OK', and the third designates the **ORKID** object type on which the operation works. The symbol definitions link the full names of the operations given in the **ORKID** standard (in lower case) to the appropriate abbreviation.

The lists of function declarations are split in two. If the symbol __ANSI__ has been defined, then all the functions are declared to the ANSI C standard using full prototyping, with parameter names also included. This latter is not necessary, but not illegal. It shows the correspondence between arguments in this and the main **ORKID** standard, the names being identical. If the symbol __ANSI__ has not been defined, then the functions are declared without prototyping.

The correspondence between the C types and arguments and those defined in the **ORKID** standard are mostly obvious. However, the following comments concerning task_start/restart and exception_catch are perhaps necessary.

A task start address is translated into a function with one argument -a pointer to anything. The task's startup arguments are given as a pointer to anything and a length. The actual arguments will be contained in a programmer defined data type, a copy of which will be passed to the new task. The following is an example of a declaration of a task's main program and a call to start that task (the necessary task creation call is not included):

```
typedef struct { int arg1, arg2, arg3 } argblock ; /*can contain
argblock *argp ;                                    anything*/

void taskmain( argblock *taskargs ) { ... } ;       /*main task program*/

status = oktsta( tid, taskmain, *argp, size_of( argblock ) ) ;

/*start the task*/
```

An XHR address also becomes a function with one argument - this time a bitfield. The previous XHR address output parameter becomes a pointer to such a function. The following is an example of the declaration of an XHR and a call to exception_catch to set it up:

```
void taskxhr( bit_field exceptions_caught ) { ... } ;  /*XHR
                                                        declaration*/
void (*prevxhr)() ;

status = okxcat( taskxhr, NOXHR, prevxhr ) ;  /*set up taskxhr as XHR*/
*/
```

```
/*    Task Operations    */

#ifdef __ANSI__

extern int oktcre( char *name, prio priority, int stacksize, bit_field
                   mode, bit_field options, task_id *tid ) ;
extern int oktdel( task_id *tid ) ;
extern int oktidt( char *name, node_id node, task_id *tid ) ;
extern int oktsta( task_id *tid, void start(void *), void *arguments,
                   int arg_length ) ;
extern int oktrst( task_id *tid, void *arguments, int arg_length ) ;
extern int oktsus( task_id *tid ) ;
extern int oktrsm( task_id *tid ) ;
extern int oktspr( task_id *tid, prio new_prio, prio *prev_prio ) ;
extern int oktsmd( bit_field mode, bit_field mask, bit_field
                   *prev_mode ) ;
extern int oktrdl( task_id *tid, lnum loc_number, int *loc_value ) ;
extern int oktwrl( task_id *tid, lnum loc_number, int loc_value ) ;

#else

extern int oktcre( ) ;
extern int oktdel( ) ;
extern int oktidt( ) ;
extern int oktsta( ) ;
extern int oktrst( ) ;
extern int oktsus( ) ;
extern int oktrsm( ) ;
extern int oktspr( ) ;
extern int oktsmd( ) ;
extern int oktrdl( ) ;
extern int oktwrl( ) ;

#endif

#define task_create           oktcre
#define task_delete           oktdel
#define task_ident            oktidt
#define task_start            oktsta
#define task_restart          oktrst
#define task_suspend          oktsus
#define task_resume           oktrsm
#define task_set_priority     oktspr
#define task_set_mode         oktsmd
#define task_read_location    oktrdl
#define task_write_location   oktwrl
```

```
/*    Region Operations    */

#ifdef __ANSI__

extern int okrcre( char *name, void *addr, int length, int granularity,
                   bit_field options, region_id *rid ) ;
extern int okrdel( region_id *rid, bit_field options ) ;
extern int okridt( char *name, region_id *rid ) ;
extern int okrgsg( region_id *rid, int seg_size, void **seg_addr ) ;
extern int okrrsg( region_id *rid, void *seg_addr ) ;

#else

extern int okrcre( ) ;
extern int okrdel( ) ;
extern int okridt( ) ;
extern int okrgsg( ) ;
extern int okrrsg( ) ;

#endif

#define region_create    okrcre
#define region_delete    okrdel
#define region_ident     okridt
#define region_get_seg   okrgsg
#define region_ret_set   okrrsg
```

```
/*    Partition Operations    */

#ifdef __ANSI__

extern int okpcre( char *name, void *addr, int length, int block_size,
                   bit_field options, part_id *pid ) ;
extern int okpdel( part_id *pid, bit_field options ) ;
extern int okpidt( char *name, node_id nid, part_id *pid,
                   int block_size ) ;
extern int okpgbl( part_id *pid, void **blk_addr ) ;
extern int okprbl( part_id *pid, void *blk_addr ) ;

#else

extern int okpcre( ) ;
extern int okpdel( ) ;
extern int okpidt( ) ;
extern int okpgbl( ) ;
extern int okprbl( ) ;

#endif

#define partition_create      okpcre
#define partition_delete      okpdel
#define partition_ident       okpidt
#define partition_get_blk     okpgbl
#define partition_ret_blk     okprbl
```