Uil Functions

Uil

UilDumpSymbolTable

Uil (library call)

NAME

Uil -Invokes the UIL compiler from within an application

SYNOPSIS

#include <uil/UilDef.h>

Uil_status_type **Uil**(
Uil_command_type **command_desc*,
Uil_compile_desc_type ***compile_desc*,
Uil_continue_type (**message_cb*) (),
char **message_data*,
Uil_continue_type (**status_cb*) (),
char **status_data*);

DESCRIPTION

The **Uil** function provides a callable entry point for the UIL compiler. The **Uil** callable interface can be used to process a UIL source file and to generate UID files, as well as return a detailed description of the UIL source module in the form of a symbol table (parse tree).

command_desc

Specifies the **uil** command line.

compile_desc

Returns the results of the compilation.

message_cb

Specifies a callback function that is called when the compiler encounters errors in the UIL source.

message_data

Specifies user data that is passed to the message callback function (*message_cb)*. Note that this argument is not interpreted by UIL, and is used exclusively by the calling application.

status_cb

Specifies a callback function that is called to allow X applications to service X events such as updating the screen. This function is called at various check points, which have been hard coded into the UIL compiler. The *status_update_delay* argument in *command_desc* specifies the number of check points to be passed before the *status_cb* function is invoked.

status_data

Specifies user data that is passed to the status callback function (*status_cb).* Note that this argument is not interpreted by the UIL compiler and is used exclusively by the calling application.

Following are the data structures **Uil_command_type** and **Uil_compile_desc_type**:

```
typedef struct Uil_command_type {
char *source_file;
    /* single source to compile */
char *resource_file; /* name of output file */
char *listing_file; /* name of listing file */
unsigned int *include_dir_count;
    /* number of dirs. in include_dir */
```

```
    char *((*include_dir) []);
        /* dir. to search for include files */
    unsigned listing_file_flag: 1;
        /* produce a listing */
    unsigned resource_file_flag: 1;
        /* generate UID output */
    unsigned machine_code_flag: 1;
        /* generate machine code */
    unsigned report_info_msg_flag: 1;
        /* report info messages */
    unsigned report_warn_msg_flag: 1;
        /* report warnings */
    unsigned parse_tree_flag: 1;
        /* generate parse tree */
    unsigned int status_update_delay;
        /* number of times a status point is */
        /* passed before calling status_cb */
        /* function 0 means called every time */
    char *database;
        /* name of database file */
    unsigned database_flag: 1;
        /* read a new database file */
    unsigned use_setlocale_flag: 1;
        /* enable calls to setlocale */
    };
    typedef struct Uil_compile_desc_type {
    unsigned int compiler_version;
        /* version number of compiler */
    unsigned int data_version;
        /* version number of structures */
    char *parse_tree_root; /* parse tree output */
    unsigned int message_count [Uil_k_max_status+1];
    /* array of severity counts */
    };
```

Following is a description of the message callback function specified by *message_cb*:
Uil_continue_type (**message_cb*) (*message_data, message_number, severity, msg_buffer, src_buffer, ptr_buffer, loc_buffer, message_count*)

char **message_data*;
int *message_number*;
int *severity*;
char **msg_buffer*, **src_buffer*;
char **ptr_buffer*, **loc_buffer*;
int *message_count*[];

This function specifies a callback function that UIL invokes instead of printing an error message when the compiler encounters an error in the UIL source. The callback should return one of the following values:

Uil_k_terminate

Terminate processing of the source file

Uil_k_continue

Continue processing the source file

The arguments are

message_data

Data supplied by the application as the *message_data* argument to the **Uil** function. UIL does not interpret this data in any way; it just passes it to the callback.

message_number

An index into a table of error messages and severities for internal use by UIL.

severity

An integer that indicates the severity of the error. The possible values are the status constants returned by the **Uil** function. See *Return Value* for more information.

msg_buffer

A string that describes the error.

src_buffer

A string consisting of the source line where the error occurred. This string is not always available. In this case, the argument is NULL.

ptr_buffer

A string consisting of whitespace and a printing character in the character position corresponding to the column of the source line where the error occurred. This string may be printed beneath the source line to provide a visual indication of the column where the error occurred. This string is not always available. In this case, the argument is NULL.

loc_buffer

A string identifying the line number and file of the source line where the error occurred. This is not always available; the argument is then NULL.

message_count

An array of integers containing the number of diagnostic messages issued thus far for each severity level. To find the number of messages issued for the current severity level, use the *severity* argument as the index into this array.

Following is a description of the status callback function specified by *status_cb*:

```
Uil_continue_type (*status_cb) (status_data, percent_complete,
    lines_processed, current_file, message_count)
    char *status_data;
    int percent_complete;
    int lines_processed;
    char *current_file;
    int message_count[];
```

This function specifies a callback function that is invoked to allow X applications to service X events such as updating the screen. The callback should return one of the following values:

Uil_k_terminate

Terminate processing of the source file

Uil_k_continue

Continue processing the source file

The arguments are

status_data

Data supplied by the application as the *status_data* argument to the **Uil** function. UIL does not interpret this data in any way; it just passes it to the callback.

percent_complete

An integer indicating what percentage of the current source file has been processed so far.

lines_processed

An integer indicating how many lines of the current source file have been read so far.

current_file

A string containing the pathname of the current source file.

message_count

An array of integers containing the number of diagnostic messages issued thus far for each severity level. To find the number of messages issued for a given severity level, use the severity level as the index into this array. The possible severity levels are the status constants returned by the **Uil** function. See *Return Value* for more information.

RETURN

This function returns one of the following status return constants:

Uil_k_success_status

The operation succeeded.

Uil_k_info_status

The operation succeeded. An informational message is returned.

Uil_k_warning_status

The operation succeeded. A warning message is returned.

Uil_k_error_status

The operation failed due to an error.

Uil_k_severe_status

The operation failed due to an error.

RELATED

UilDumpSymbolTable(3) and **uil**(1) .

UilDumpSymbolTable (library call)

NAME

UilDumpSymbolTable – Dumps the contents of a named UIL symbol table to standard output

SYNOPSIS

#include <uil/UilDef.h>

void **UilDumpSymbolTable**(
sym_entry_type **root_ptr*);

DESCRIPTION

The **UilDumpSymbolTable** function dumps the contents of a UIL symbol table pointer to standard output.

root_ptr

Specifies a pointer to the the symbol table root entry. This value can be taken from the *parse_tree_root* part of the *Uil_compile_desc_type* data structure returned by **Uil**.

By following the link from the root entry, you can traverse the entire parse tree. Symbol table entries are in the following format:

hex.address symbol.type symbol.data prev.source.position source.position modification.record

where:

hex.address

Specifies the hexadecimal address of this entry in the symbol table.

symbol.type

Specifies the type of this symbol table entry. Some possible types are *root*, *module*, *value*, *procedure*, and *widget*.

symbol.data

Specifies data for the symbol table entry. The data varies with the type of the entry. Often it contains pointers to other symbol table entries, or the actual data for the data type.

prev.source.position

Specifies the end point in the source code for the previous source item.

source.position

Specifies the range of positions in the source code for this symbol.

The exact data structures for each symbol type are defined in the include file `UilSymDef.h`. Note that this file is automatically included when an application includes the file `UilDef.h`.

RELATED

Uil(3)