Section 7 - UIL Functions

This page describes the format and contents of each reference page in Section 7, which covers the User Interface Language (UIL) functions.

Name

Function – a brief description of the function.

Synopsis

This section shows the signature of the function: the names and types of the arguments, and the type of the return value. The header file <*uil/UilDef.h*> declares both of the public UIL functions.

Inputs

This subsection describes each of the function arguments that pass information to the function.

Outputs

This subsection describes any of the function arguments that are used to return information from the function. These arguments are always of some pointer type, so you should use the C address-of operator (**&**) to pass the address of the variable in which the function will store the return value. The names of these arguments are sometimes suffixed with *_return* to indicate that values are returned in them. Some arguments both supply and return a value; they will be listed in this section and in the "Inputs" section above. Finally, note that because the list of function arguments is broken into "Input" and "Output" sections, they do not always appear in the same order that they are passed to the function. See the function signature for the actual calling order.

Returns

This subsection explains the return value of the function, if any.

Description

This section explains what the function does and describes its arguments and return value. If you've used the function before and are just looking for a refresher, this section and the synopsis above should be all you need.

Usage

This section appears for most functions and provides less formal information about the function: when and how you might want to use it, things to watch out for, and related functions that you might want to consider.

Example

This section provides an example of the use of the function.

Structures

This section shows the definition of any structures, enumerated types, typedefs, or symbolic constants used by the function.

Procedures

This section shows the syntax of any prototype procedures used by the function.

See Also

This section refers you to related functions, clients, and UIL data types. The numbers in parentheses following each reference refer to the sections of this book in which they are found.

Name

Uil – call the UIL compiler from 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)
```

Inputs

command_desc	Specifies a structure containing the compilation options.
message_cb	Specifies a callback function that is called when error, warning and informational messages are generated by the compiler.
message_data	Specifies data that is passed to the message_cb function.
status_cb	Specifies a callback function that is called periodically during the compilation to indicate progress.
status_data	Specifies data that is passed to the status_cb function.

Outputs

compile_desc	Returns a structure containing the results of the compilation.

Returns

Uil_k_success_status on success and if no problems are detected,
Uil_k_info_status on success and if informational messages are generated,
Uil_k_warning_status on success and if warning messages are generated,
Uil_k_error_status on failure and if error messages are generated, and
Uil_k_severe_status on failure and if the compilation stopped prematurely.

Description

`Uil`() invokes the UIL compiler from within an application. Options for the compiler, including the input, output and listing files, are provided in the *command_desc* argument. The calling application can supply a message handling function in *message_cb* that displays compiler messages in an application-defined manner. The application can also supply a status-monitoring function in *status_cb*. This function is called periodically by the compiler to report progress. Upon completion, the `Uil`() function fills in the *compile_desc* structure with information about the compilation and returns the status of the compilation.

Usage

An application that calls `Uil`() is responsible for allocating the *command_desc* and *compile_desc* arguments. The application must initialize all members of the

command_desc structure. Members of the *compile_desc* structure are set by the compiler. If the *parse_tree_flag* in *command_desc* is set, the compiler returns a pointer to the root of the parse tree in the *parse_tree_root* field of the *compile_desc*. This parse table cannot be freed by the calling application. Therefore, you should not set the *parse_tree_flag* unless you plan to use the parse tree. To limit memory consumption, if you set the *parse_tree_flag*, invoke the `Uil`() routine once and exit soon thereafter.

An application can specify a function for handling compiler generated messages in the *message_cb* argument. You can specify NULL for this argument if you want to use the default message handling routine. This routine prints all messages to *stderr*. If you specify a function, the value of *message_data* is passed to each invocation of the function.

An application can also specify a function for monitoring the status of the compilation in the *status_cb* argument. You can specify NULL to indicate that no status function should be called. If you specify a function, the value of *status_data* is passed to each invocation of the function. In addition to monitoring progress, the function can also be used to process X events in an X application.

The `Uil`() function installs signal handlers for SIGBUS, SIGSYS, and SIGFPE with no regard for application installed handlers. These installed handlers remain set after the function returns, so you may wish to change them.

Applications that call the `Uil`() function must be linked with the UIL library, *libUil.a*, in addition to the Mrm, Motif, Xt, and X libraries.

Structures

The *Uil_command_type* is defined as follows:

```
typedef struct {
    char          *source_file;              /* name of UIL source file */
    char          *resource_file;            /* name of UID output file */
    char          *listing_file;             /* name of listing file */
    unsigned int  include_dir_count;         /* length of include_dir array */
    char          **include_dir;             /* array of include file directories */
    unsigned int  listing_file_flag : 1;     /* write listing file flag */
    unsigned int  resource_file_flag : 1;    /* write UID file flag */
    unsigned int  machine_code_flag : 1;     /* write machine code flag */
    unsigned int  report_info_msg_flag : 1;  /* report informational mes-
    sages */
    unsigned int  report_warn_msg_flag : 1; /* report warning messages */
    unsigned int  parse_tree_flag : 1;       /* generate parse tree flag */
    unsigned int  issue_summary : 1;         /* write diagnostic summary flag
    */
```

```
    unsigned int    status_update_delay;    /* delay between status_cb calls */
    char            *database;              /* WML database filename */
    unsigned int    database_flag : 1;      /* read WML database flag */
    unsigned int    use_setlocale_flag : 1; /* parse strings in locale flag */
} Uil_command_type;
```

Uil_command_type describes the compilation options for the `Uil`() routine. *source_file* is the name of the UIL module to compile. *resource_file* is the name of the UID file that is output if *resource_file_flag* is set. *listing_file* is the name of the compilation listing file that is output if *listing_file_flag* is set. Setting *machine_code_flag* causes the compiler to output a binary description of the UID file when a listing is generated.

include_dir specifies an array of *include_dir_count* directory names that the compiler searches for UIL include files. If set, *report_info_msg_flag*, *report_warn_msg_flag*, and *issue_summary* cause the compiler to generate informational messages, warning messages, and a summary message, respectively.

If *parse_tree_flag* is set, it instructs the compiler to return a pointer to the parse tree of the module in the *compile_desc* structure. *status_update_delay* specifies how many status check points must be passed before the *status_cb* callback is called. If the field is set to zero, the function is called at every check point.

use_setlocale_flag directs the UIL compiler to parse double-quoted strings in the current locale. (See the UIL string type man page for more information.) *database* specifies the name of a Widget Meta-Language (WML) description file that the compiler loads if *database_flag* is set.

The *Uil_compile_desc_type* is defined as follows:

```
typedef struct _Uil_comp_desc {
    unsigned int    compiler_version;   /* UIL compiler version */
    unsigned int    data_version;       /* UIL structures version */
    char            *parse_tree_root;   /* parse tree for module */
    unsigned int    message_count[];    /* status messages counts */
} Uil_compile_desc_type;
```

Uil_compile_desc_type describes the return data for the `Uil`() routine. *compiler_version* specifies the version of the UIL compiler, while *data_version* specifies the version of the structures used by the compiler. If *parse_tree_flag* is set in the *command_desc* argument, *parse_tree_root* contains a pointer to a compiler-generated parse tree if the compilation succeeds. *message_count* is an array of integers that contains the number of each type of compiler message generated by the routine. Valid indices to the array are Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, and Uil_k_severe_status.

Procedures

A message_cb function has the following syntax:

```
Uil_continue_type *message_cb (char   *message_data,
                               int    message_number,
                               int    severity,
                               char   *message_string,
                               char   *source_text,
                               char   *column_string,
                               char   *location_string,
                               int    message_count[])
```

A *message_cb* function takes eight arguments. The first argument, *message_data*, is the value of the *message_data* argument passed to the `Uil`() function. *message_number* is the internal index of the message, which is used by the UIL compiler. *severity* specifies the severity of the message, which is one of Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, or Uil_k_severe_status.

message_string is a string describing the problem. *source_text* is a copy of the source line to which the message refers, with a tab character prepended. If the source line is not available, *source_text* is the empty string. *column_string* is a string that consists of a leading tab character followed by zero or more spaces and an * (asterisk) in the same column as the problem in the source line. This string is suitable for printing beneath *message_string* to indicate the location of the problem. If the column that contains the error or the source line is not available, *column_string* is the empty string.

location_string describes the location where the problem occurred. The format of this string is "\t\t line: %d file: %s" if both source and column number are available, or if no column number is available. If the column number, but no source line is available, the format is "\t\t line: %d position: %d file: %s". If the location is unavailable, the value of *location_string* is the empty string. If an application does not specify a *message_cb* routine, the compiler prints *source_text*, *column_string*, *message_string*, and *location_string* in that order.

message_count is an array of integers that contains the number of each type of compiler message generated by the routine so far. Valid indices to the array are Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, and Uil_k_severe_status.

A *message_cb* function should return Uil_k_continue if the compilation can continue or Uil_k_terminate if the compilation should be terminated.

A status_cb function has the following format:

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

A *status_cb* function takes five parameters. The first argument, *status_data*, is the value of the *status_data* argument passed to the `Uil`() function. *percent_complete* specifies an estimate of the percentage of the compilation that has been completed. The value of this field falls within a fixed range of values for each step of the compilation. The value ranges from 0 to 50 while *source_file* is being parsed, from 60 to 80 while the *resource_file* is written, and from 80 to 100 while the *listing_file* is generated. Some versions of the UIL compiler may only report percent-complete values on the boundaries of these ranges. *lines_processed* indicates the number of lines that have been read from the input file.

When the UIL compiler is invoked, it parses the *source_file*, writes the resource_file, and then generates the *listing_file*, based on the settings of the *command_desc* argument. The *current_file* field changes to reflect the file that the compiler is accessing.

message_count is an array of integers that contains the number of each type of compiler message generated by the routine so far. Valid indices to the array are Uil_k_info_status, Uil_k_warning_status, Uil_k_error_status, and Uil_k_severe_status.

A *status_cb* function should return Uil_k_continue if the compilation can continue or Uil_k_terminate if the compilation should be terminated.

The frequency with which the compiler calls the *status_cb* function at check points is based on the value of *status_update_delay* field in *command_desc*. A check point occurs every time a symbol is found during the parsing of *source_file*, every time an element is written to the *resource_file*, and every time a line is written to the *listing_file*.

Example

The following routines illustrate the use of the `Uil`() routine in a very basic way:

```
#include <uil/UilDef.h>
#include <stdio.h>

static char *last_current_file;
static char *status_string_list[Uil_k_max_status] = { NULL };
```

```
Uil_continue_type message_cb ( char    *message_data,
                               int     message_number,
                               int     severity,
                               char *message_string,
                               char    *line_text,
                               char    *error_col_string,
                               char    *line_and_file_string,
                               int     *message_count)
{
    if (*line_text != ' ')
        puts (line_text);
    if (*error_col_string != ' ')
        puts (error_col_string);
    if (*message_string != ' ')
        printf ("%s: %s\n", status_string_list[severity], message_string);
    if (*line_and_file_string != ' ')
        puts (line_and_file_string);

    return (Uil_k_continue);
}

Uil_continue_type status_cb ( char    *status_data,
                              int     percent_complete,
                              int     lines_processed,
                              char    *current_file,
                              int     *message_count)
{
    if (last_current_file == NULL || strcmp (last_current_file, current_file) != 0)
    {
        fprintf (stderr, "Working on file %s...\n", current_file);
        last_current_file = current_file;
    }

    return (Uil_k_continue);
}

Uil_compile_desc_type * compile (char *filename)
{
    Uil_command_type                command_desc;
    static Uil_compile_desc_type    compile_desc;
    Uil_status_type                 status;

    if (status_string_list[Uil_k_success_status] == NULL) {
        status_string_list[Uil_k_success_status]       = "Success";
```

```
        status_string_list[Uil_k_info_status]       = "Informational";
        status_string_list[Uil_k_warning_status]    = "Warning";
        status_string_list[Uil_k_error_status]      = "Error";
        status_string_list[Uil_k_severe_status]     = "Severe Error";
    }
    command_desc.source_file                = filename;
    command_desc.resource_file              = "a.uid";
    command_desc.listing_file               = "uil.lst";
    command_desc.include_dir_count          = 0;
    command_desc.include_dir                = NULL;
    command_desc.listing_file_flag          = TRUE;
    command_desc.resource_file_flag         = TRUE;
    command_desc.machine_code_flag          = FALSE;
    command_desc.report_info_msg_flag       = TRUE;
    command_desc.report_warn_msg_flag       = TRUE;
    command_desc.parse_tree_flag            = FALSE;
    command_desc.issue_summary              = TRUE;
    command_desc.status_update_delay        = 0;
    command_desc.database                   = NULL;
    command_desc.database_flag              = FALSE;
    command_desc.use_setlocale_flag         = FALSE;

    last_current_file                       = NULL;

    status = Uil (&command_desc, &compile_desc, message_cb, NULL,
    status_cb, NULL);

    if (status == Uil_k_error_status || status == Uil_k_severe_status)
        return (NULL);

    return (&compile_desc);
}

int main (int argc, char **argv)
{
    Uil_compile_desc_type *compile_desc;

    if (argc != 2) {
        printf ("usage: Uil filename\n");
        exit (1);
    }

    compile_desc = compile (argv[1]);

    if (compile_desc != NULL)
        fprintf (stderr, "Compilation Successful.\n");
```

```
    else
        fprintf (stderr, "Compilation Failed.\n");
}
```

See Also

`uil`(4), `string`(6), `UilDumpSymbolTable`(7).

Name

UilDumpSymbolTable – produce a listing of a UIL symbol table.

Synopsis

#include <uil/UilDef.h>

void UilDumpSymbolTable (sym_entry_type **parse_tree_root*)

Inputs

parse_tree_root Specifies a pointer to the root entry of a symbol table.

Description

`UilDumpSymbolTable`() prints a listing of the symbols parsed in a UIL module to stdout. A parse tree is generated by a call to `Uil`(). If the *parse_tree_flag* of the *Uil_command_type* structure passed to the routine is set and the compilation is successful, the `Uil`() routine returns a pointer to the root of the parse tree in the *parse_tree_root* member of the *Uil_compile_desc_type* structure. If the compilation is unsuccessful, the *parse_tree_root* field is set to NULL.

Usage

`UilDumpSymbolTable`() generates a listing of the internal representation of UIL structures and symbols, which is really only useful for people who are quite familiar with the internals of the UIL compiler. The -m option of the `uil` command, or the *machine_code_flag* option of the `Uil`() routine, generates far more useful information for most users of UIL.

Instead of calling `UilDumpSymbolTable`(), an application can examine the parse tree directly. The structures used in the parse tree are defined in the file <*uil/UilSymDef.h*> and definitions of constants used in the structures are in <*uil/UilDBDef.h*>. Both of these files are included by <*uil/UilDef.h*>.

The parse table generated by the `Uil`() routine cannot be freed by the calling application. Therefore, you should not set the *parse_tree_flag* unless you plan to use the parse tree. To limit memory consumption, if you set the *parse_tree_flag*, invoke the `Uil`() routine once and exit soon thereafter.

See Also

`uil`(4).
`Uil`(7).