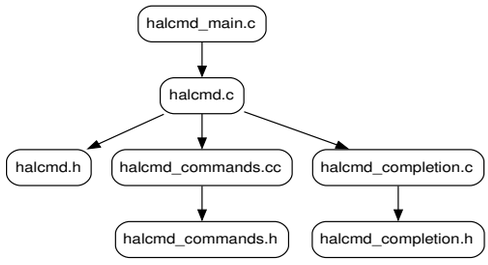
Herramientas HAL

Halcmd



https://github.com/LinuxCNC/linuxcnc/blob/master/src/hal/utils/halcmd_completion.c

Descripción: Este archivo está vinculado al sistema de autocompletado de comandos de halcmd.

INCLUDE

```
#include "halcmd_completion.h"
#include "config.h"
#include "rtapi.h"           /* RTAPI realtime OS API */
#include "hal.h"             /* HAL public API decls */
#include "../hal_priv.h"     /* private HAL decls */
#include <rtapi_mutex.h>
#include <string.h>
#include <ctype.h>
#include <dirent.h>
```

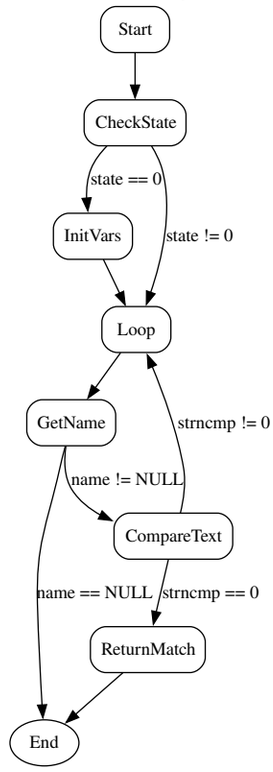
DEFINICIONES

```
static int argno;

static const char *command_table[] = {
    "loadrt", "loadusr", "unload", "lock", "unlock",
    "linkps", "linksp", "linkpp", "unlinkp",
    "net", "newsig", "delsig", "getp", "gets", "setp", "sets", "ptype", "stype",
    "addf", "delf", "show", "list", "status", "save", "source",
    "start", "stop", "quit", "exit", "help", "alias", "unalias",
    NULL,
};

static const char *nonRT_command_table[] = {
    "-h",
    NULL,
};

static const char *alias_table[] = {
    "param", "pin",
    NULL,
};
```

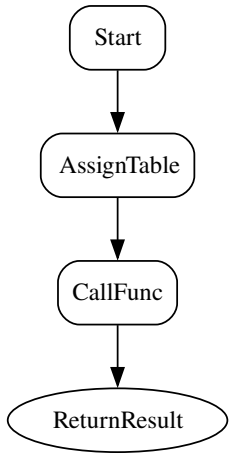
```
static const char *show_table[] = {
    "all", "alias", "comp", "pin", "sig", "param", "funct", "thread",
    NULL,
};

static const char *save_table[] = {
    "all", "alias", "comp", "sig", "link", "linka", "net", "neta", "param",
"thread",
    NULL,
};

static const char *list_table[] = {
    "comp", "alias", "pin", "sig", "param", "funct", "thread",
    NULL
};

static const char *status_table[] = {
    "alias", "lock", "mem", "all",
    NULL
};

static const char *pintype_table[] = {
    "bit", "float", "u32", "s32",
    NULL
};

static const char *lock_table[] = { "none", "tune", "all", NULL };
static const char *unlock_table[] = { "tune", "all", NULL };

static const char **string_table = NULL;

static hal_type_t match_type = -1;
static int match_writers = -1;
static hal_pin_dir_t match_direction = HAL_DIR_UNSPECIFIED;
const char *loadusr_table[] = {"-W", "-Wn", "-w", "-iw", NULL};
```

FUNCIONES

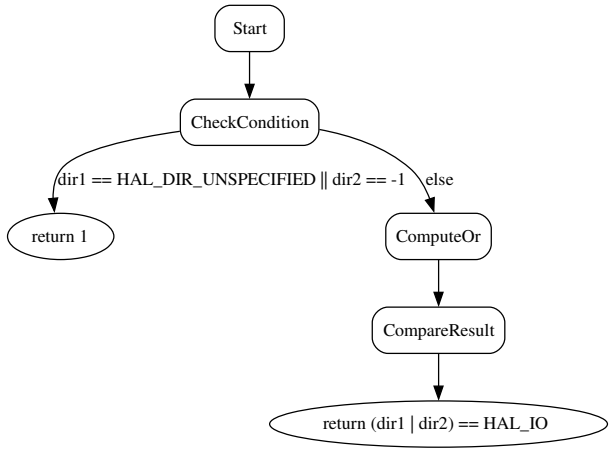
1. `static char *table_generator(const char *text, int state){}`
   Descripción: generador genérico que sirve para autocompletar desde tablas estáticas de cadenas (como command_table, alias_table, list_table, etc.), devolviendo la primera coincidencia que encaje con el prefijo.



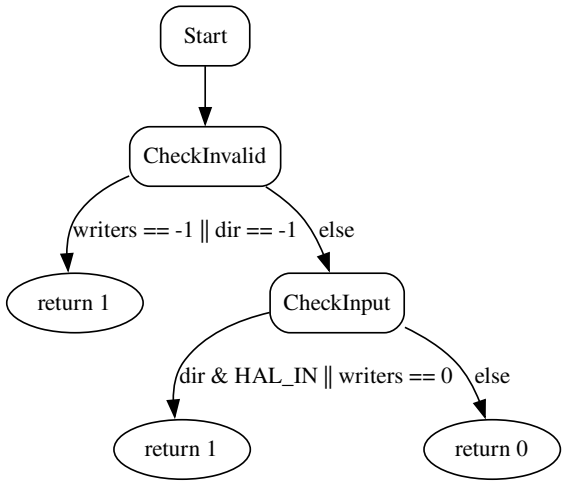
2. `static char **completion_matches_table(const char *text, const char **table, hal_completer_func func) {}`
   Descripción: configura la tabla de cadenas y delega la lógica de autocompletado en la función func.



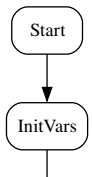
3. `static int direction_match(hal_pin_dir_t dir1, hal_pin_dir_t dir2) {}`
   Descripción: `devuelve directamente 1, o el valor booleano de la comparación (dir1 | dir2)`

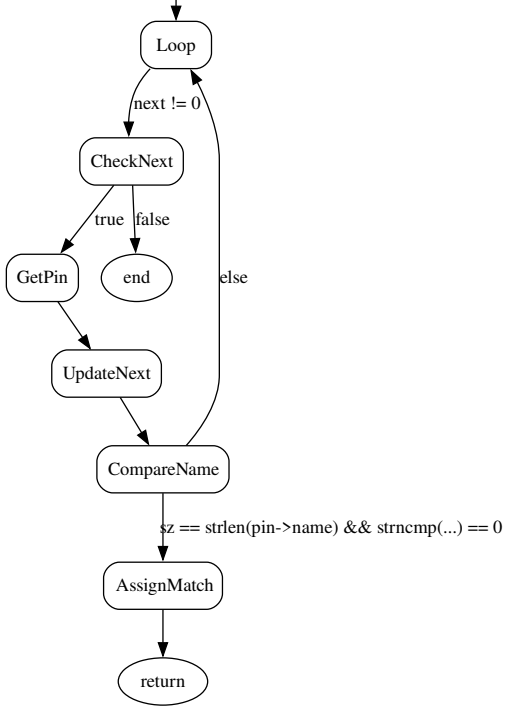


4. `static int writer_match(hal_pin_dir_t dir, int writers) {}`
   Descripción: `Filtro lógico que decide si un pin coincide con las condiciones de escritura.`

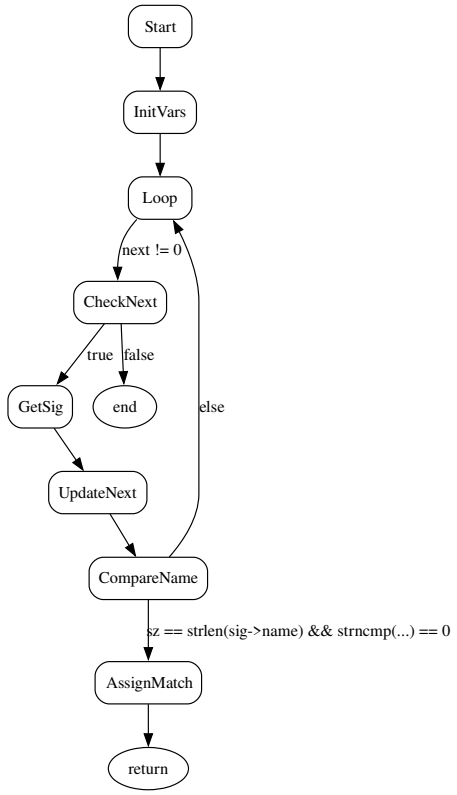


5. `static void check_match_type_pin(const char *name) {}`
   Descripción: `Recorre la lista de pines HAL buscando uno cuyo nombre coincida exactamente con el prefijo de name. Si lo encuentra, guarda su tipo y dirección en variables globales y termina.`

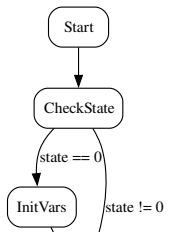


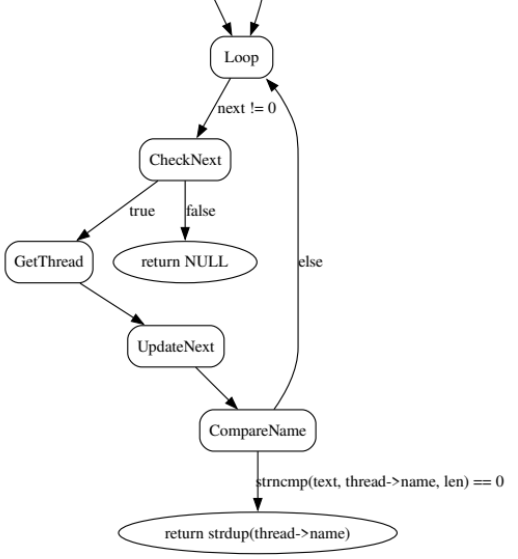


6. `static void check_match_type_signal(const char *name) {}`
   Descripción: Recorre la lista de **señales HAL** y guarda su tipo y número de escritores si encuentra coincidencia.

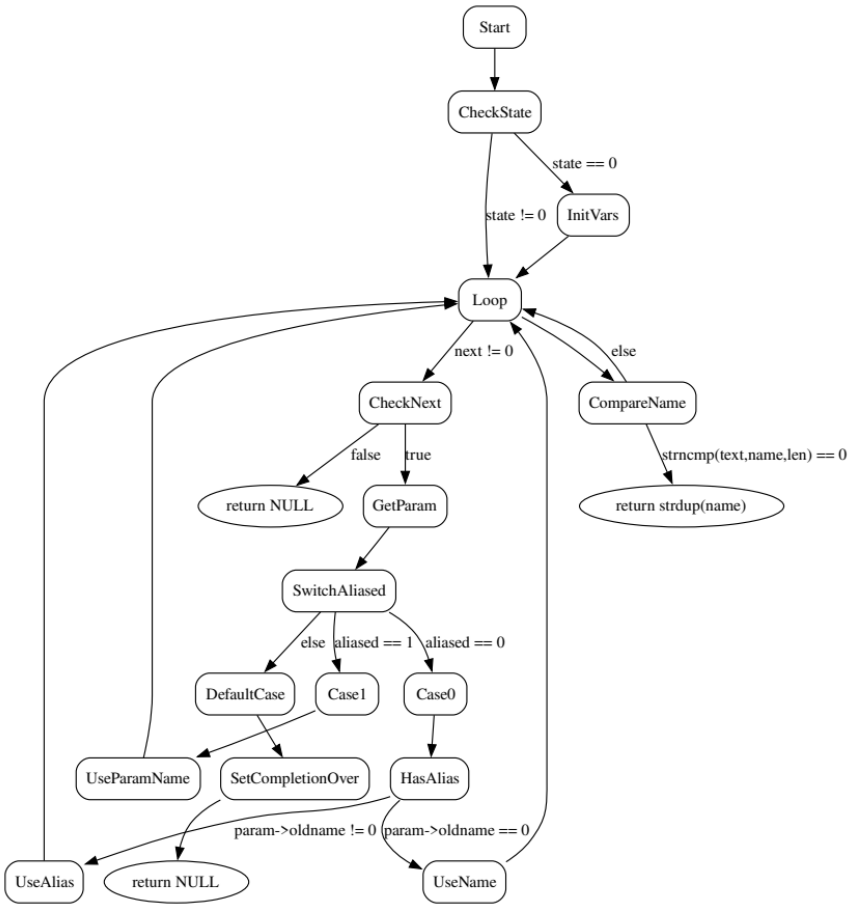


7. `static char *thread_generator(const char *text, int state) {}`
   Descripción: Similar en estructura a los generadores de pines y señales, pero aplicado a la lista de **hilos HAL**.

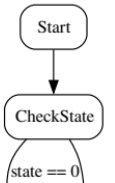


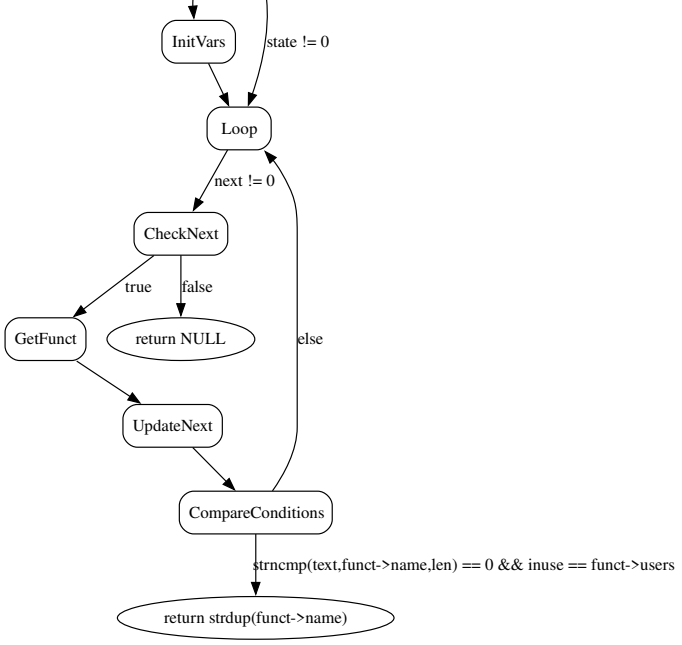


8. `static char *parameter_generator(const char *text, int state) {}`
   Descripción: gestiona tanto nombres actuales como **alias antiguos** de parámetros HAL, asegurándose de que se procesen en orden correcto.

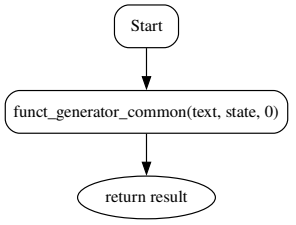


9. `static char *funct_generator_common(const char *text, int state, int inuse) {}`
   Descripción: Muy similar a los generadores anteriores, pero añade una condición extra: además de comparar el nombre, también valida que el número de usuarios (inuse) coincida con el campo funct→users.



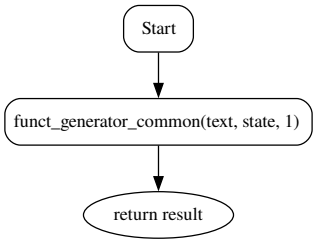


10. `static char *funct_generator(const char *text, int state) {}`
    Descripción: `Delega todo el trabajo en funct_generator_common, con inuse = 0`



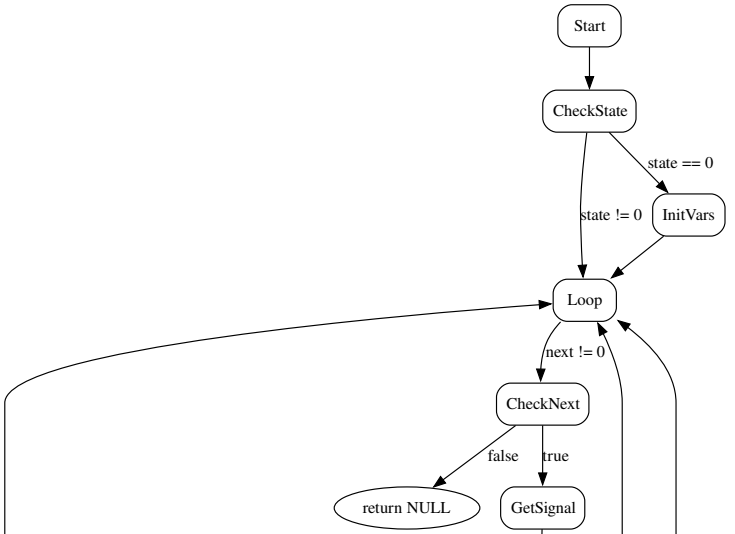
11. `static char *attached_funct_generator(const char *text, int state) {}`

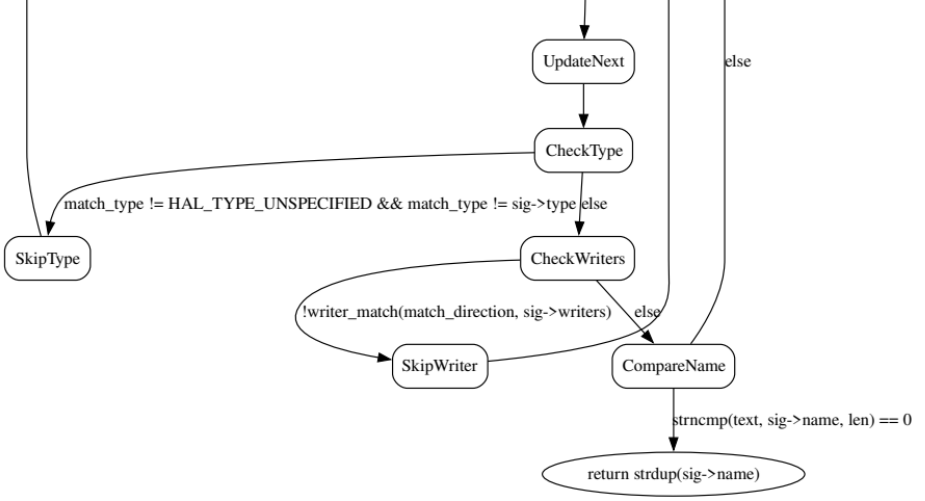
    Descripción: `Función` **`wrapper`**`: delega todo el trabajo en funct_generator_common, pero con inuse = 1`



12. `static char *signal_generator(const char *text, int state) {}`

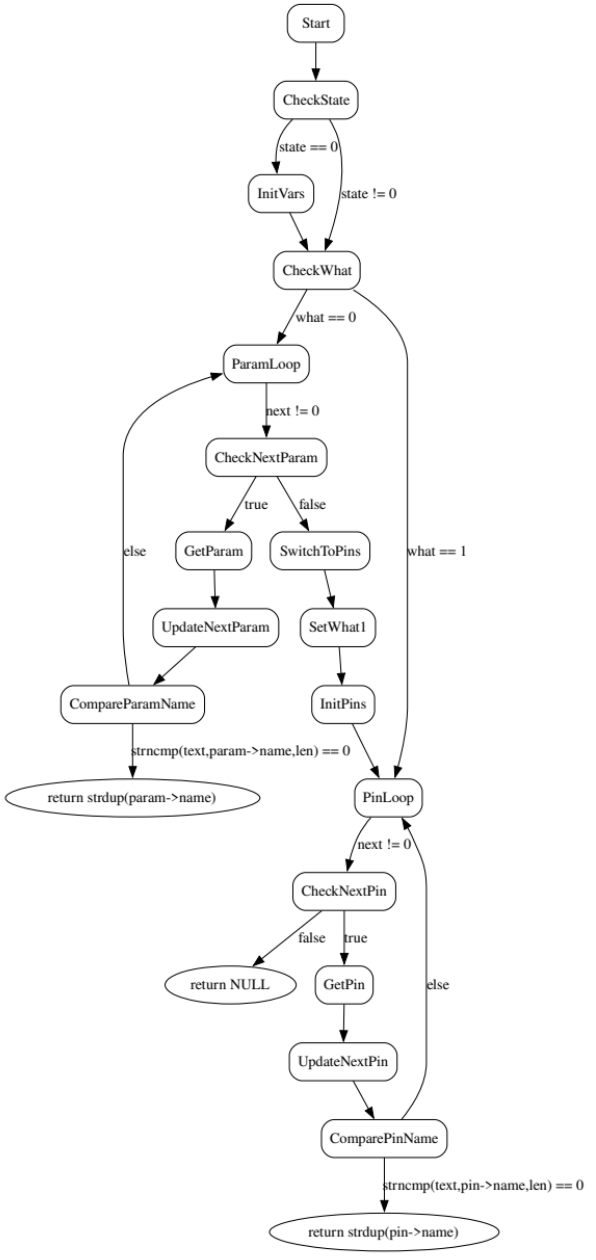
    Descripción: `No solo compara nombres, sino que también filtra por` **`tipo de señal`** `y por` **`compatibilidad de escritores`** `antes de aceptar una coincidencia.`





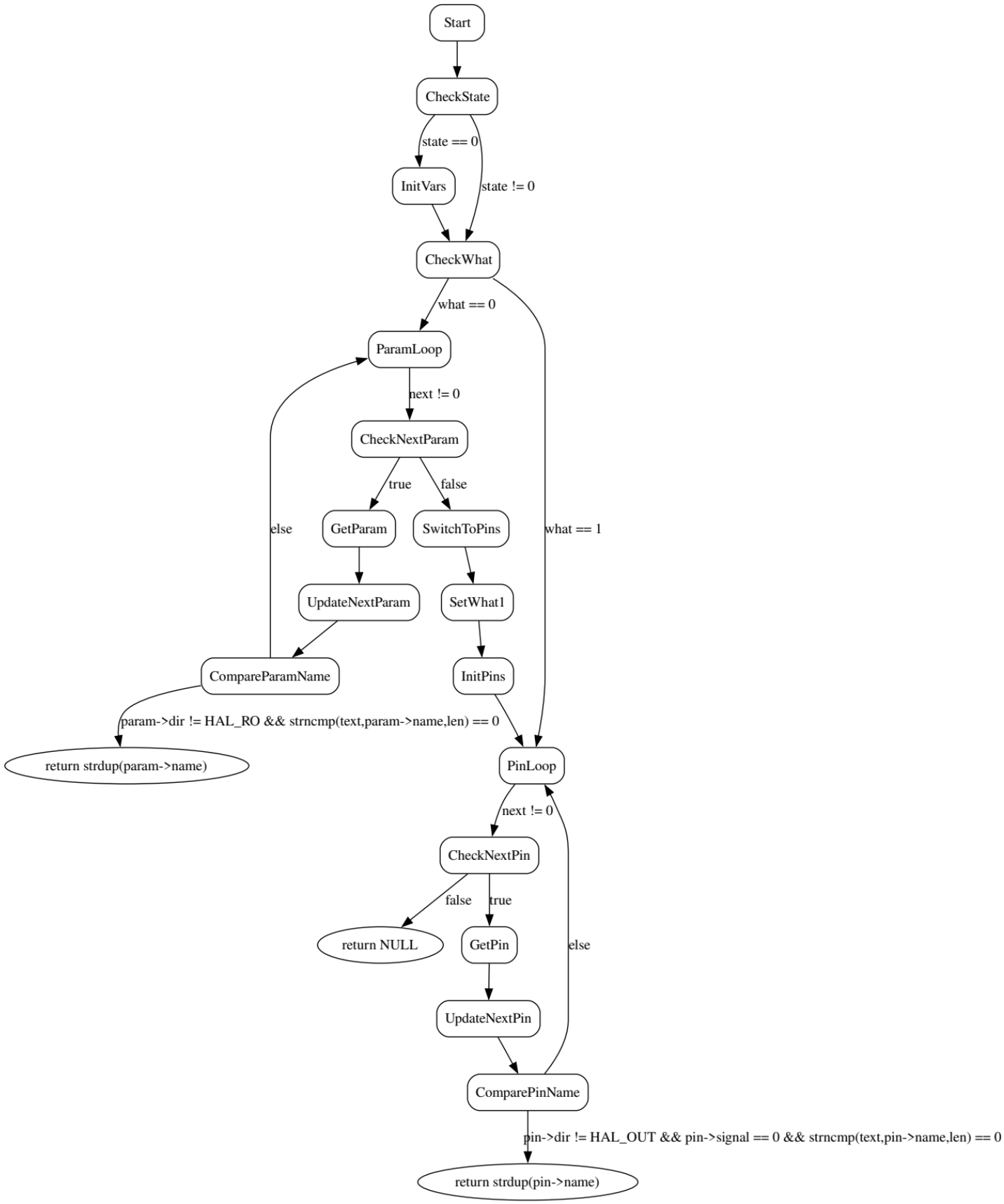
13. `static char *getp_generator(const char *text, int state) {}`

Descripción: Busca primero en parámetros y luego en pines. Si no encuentra coincidencia en la primera lista, automáticamente cambia a la segunda.



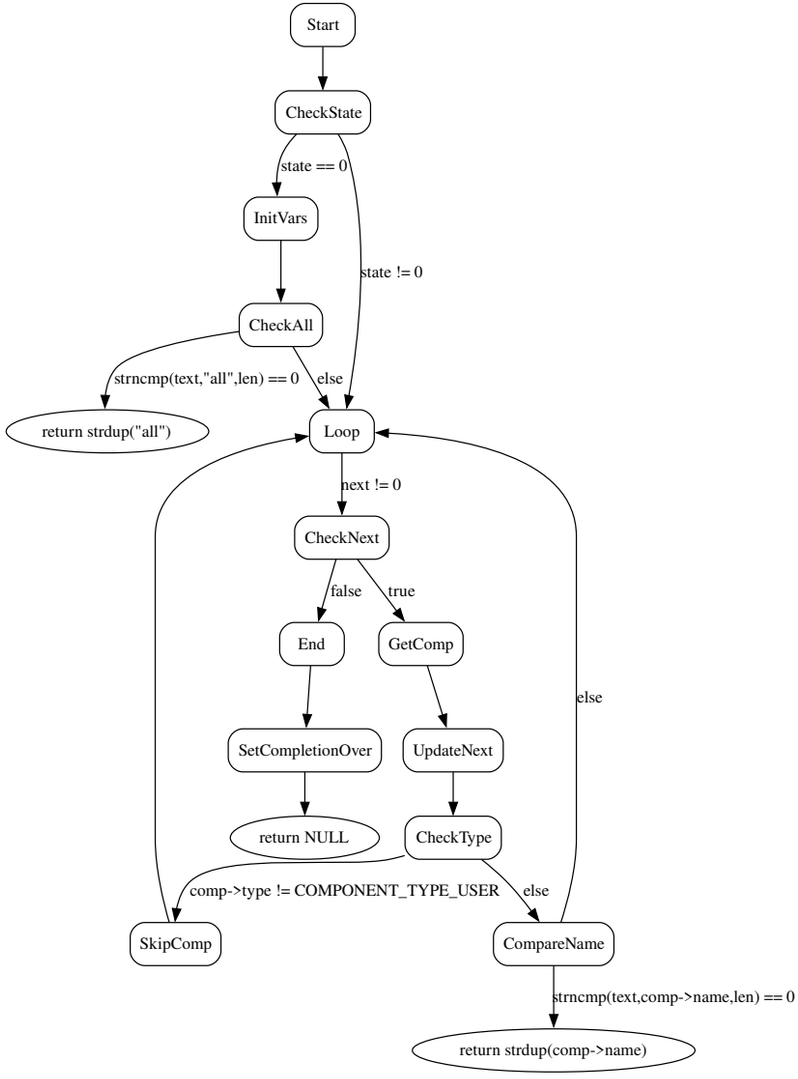
14. `static char *setp_generator(const char *text, int state) {}`

Descripción: Muy parecida a getp_generator, pero con **restricciones adicionales**;los parámetros no deben ser HAL_RO,los pines no deben ser HAL_OUT, y no deben estar conectados (signal == 0).



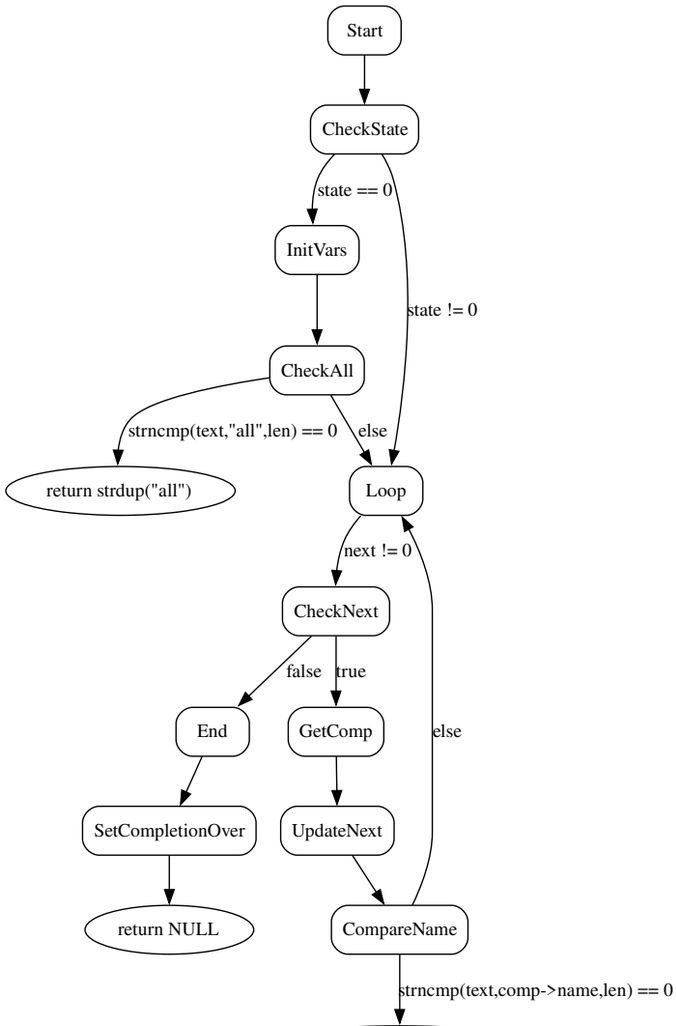
15. `static char *usrcomp_generator(const char *text, int state) {}`

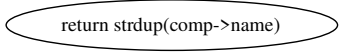
Descripción: generador de autocompletado para **componentes de usuario** en HAL. Tiene un caso especial para la palabra "all", y si no, recorre la lista de componentes filtrando por tipo y nombre.



16. `static char *comp_generator(const char *text, int state) {}`

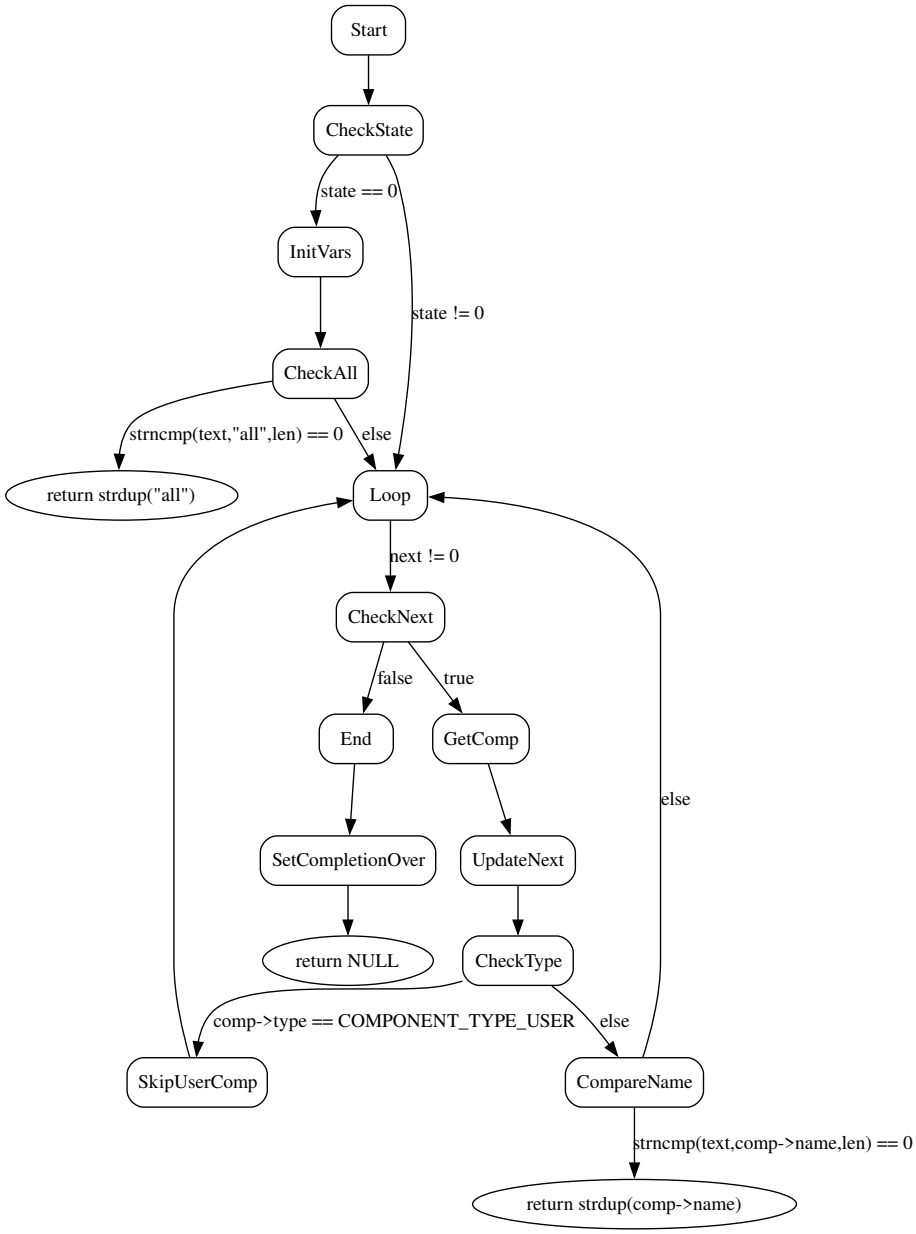
Descripción: Generador de autocompletado para **componentes HAL**. Tiene un caso especial para la palabra "all", y si no, recorre la lista de componentes comparando nombres.





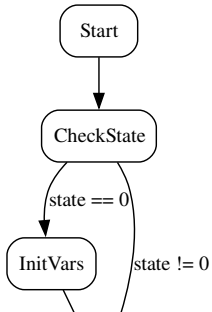
17. `static char *rtcomp_generator(const char *text, int state) {}`

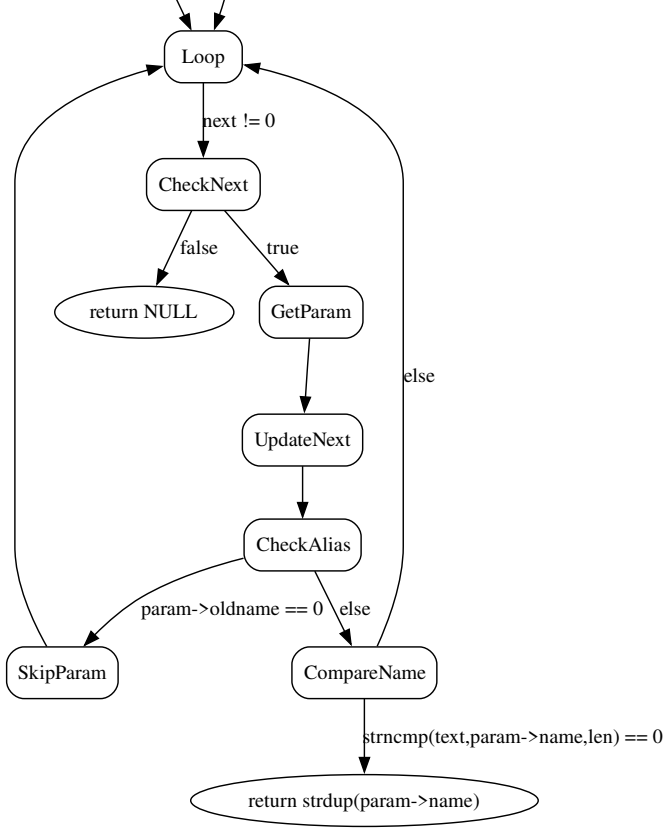
Descripción: Generador de autocompletado para **componentes en tiempo real (RT)** dentro de HAL, similar a comp_generator, pero añade un filtro que **excluye los componentes de tipo usuario** (COMPONENT_TYPE_USER).



18. `static char *parameter_alias_generator(const char *text, int state) {}`

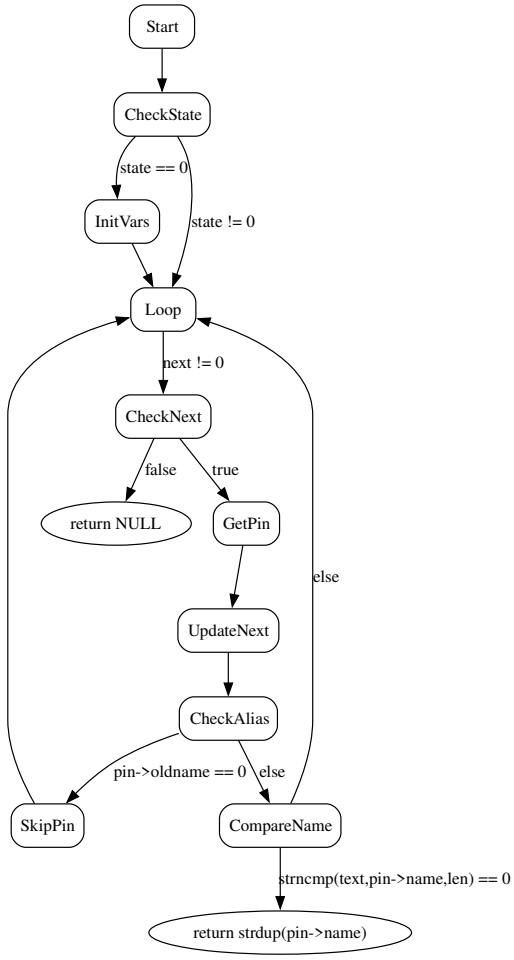
Descripción: Generador de autocompletado que **solo considera parámetros con alias**. Si el parámetro no tiene alias (oldname == 0), lo ignora y sigue buscando.





19. `static char *pin_alias_generator(const char *text, int state) {}`

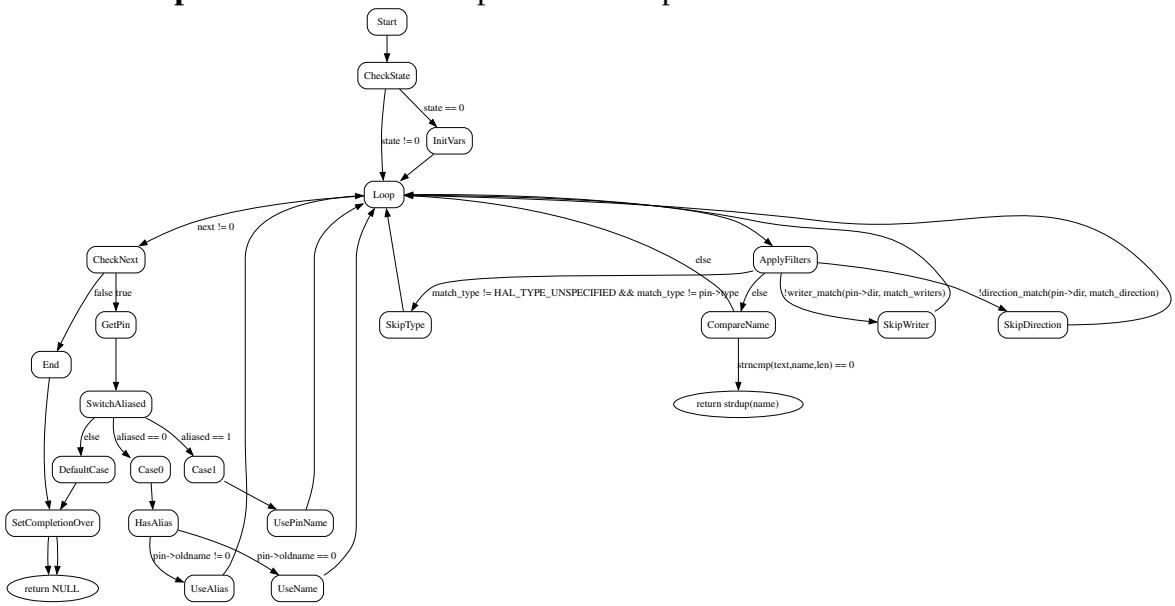
Descripción: `Generador de autocompletado que` **`solo considera pines con alias`**`. Si el pin no tiene alias (oldname == 0), lo ignora y sigue buscando.`



20. `static char *pin_generator(const char *text, int state) {}`

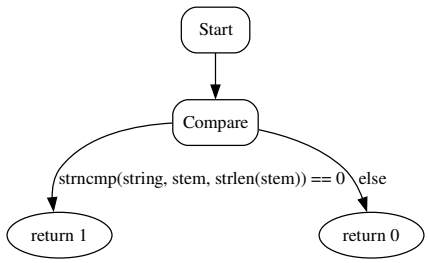
Descripción: `Esta función combina` **Gestión de alias** (primero devuelve alias si

existe, luego el nombre real), **Filtros de coincidencia** (tipo, dirección y escritores) y **Comparación de nombre** para autocompletado.



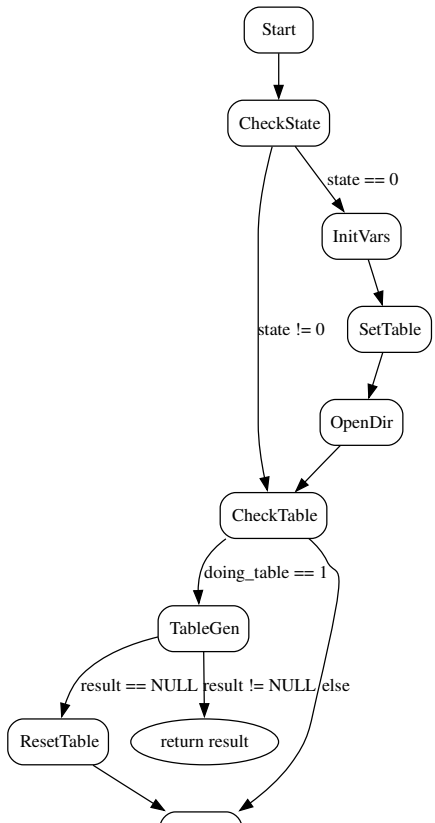
21. `static int startswith(const char *string, const char *stem) {}`

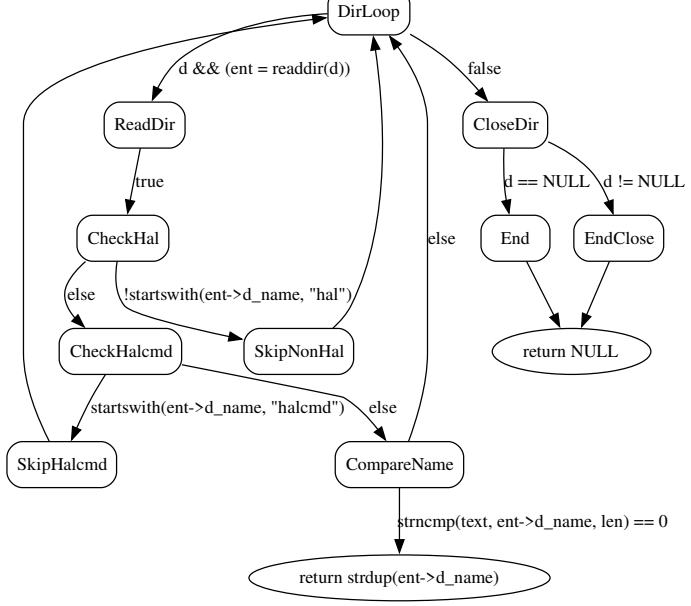
Descripción: **`Wrapper minimalista`** sobre `strncmp` que encapsula la lógica de "¿empieza con…?". Muy útil para simplificar código en generadores de autocompletado.



22. `static char *loadusr_generator(const char *text, int state) {}`

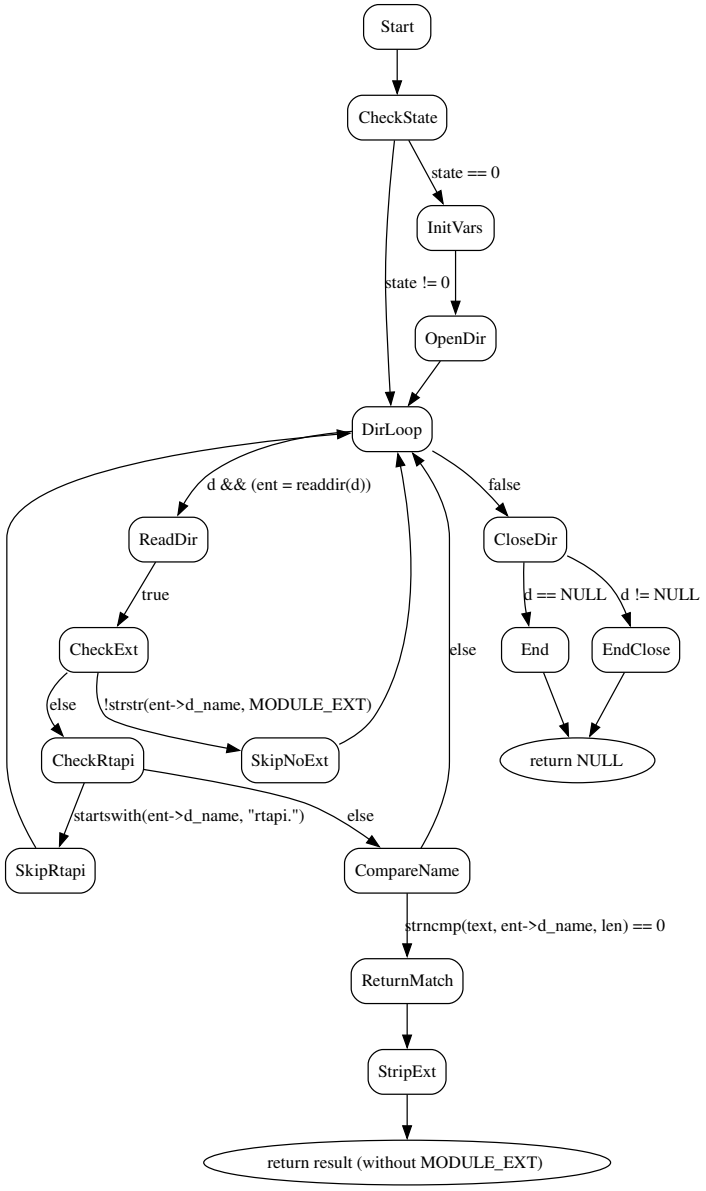
Descripción: Esta función combina dos fuentes de autocompletado; **Tabla de comandos (**`loadusr_table`**)** y **Ejecutables HAL en el directorio de binarios (**`EMC2_BIN_DIR`**)**, filtrando solo los que empiezan por "hal" pero no "halcmd".





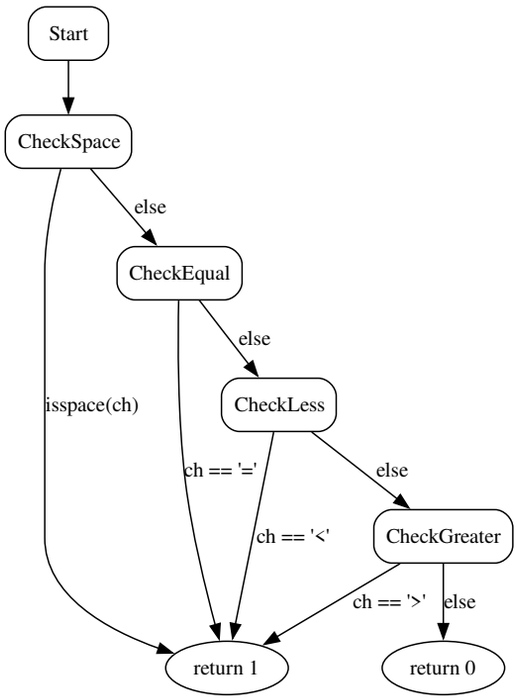
23. `static char *loadrt_generator(const char *text, int state) {}`

Descripción: Función útil para autocompletar nombres de **módulos HAL en tiempo real**, filtrando solo los válidos (con extensión de módulo, que no sean rtapi.*) y devolviendo el nombre sin extensión.



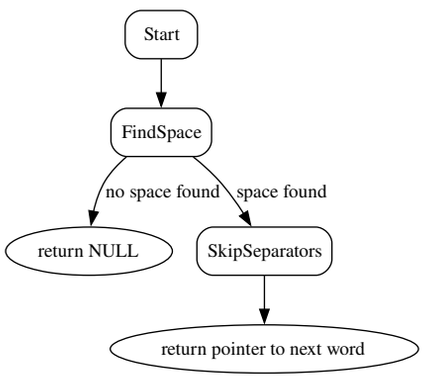
24. `static inline int isskip(int ch) {}`

Descripción: Devuelve verdadero (1) si el carácter es un espacio en blanco o uno de los símbolos =, <, >. Es una función auxiliar muy útil para parsers o analizadores de texto que necesitan ignorar separadores.



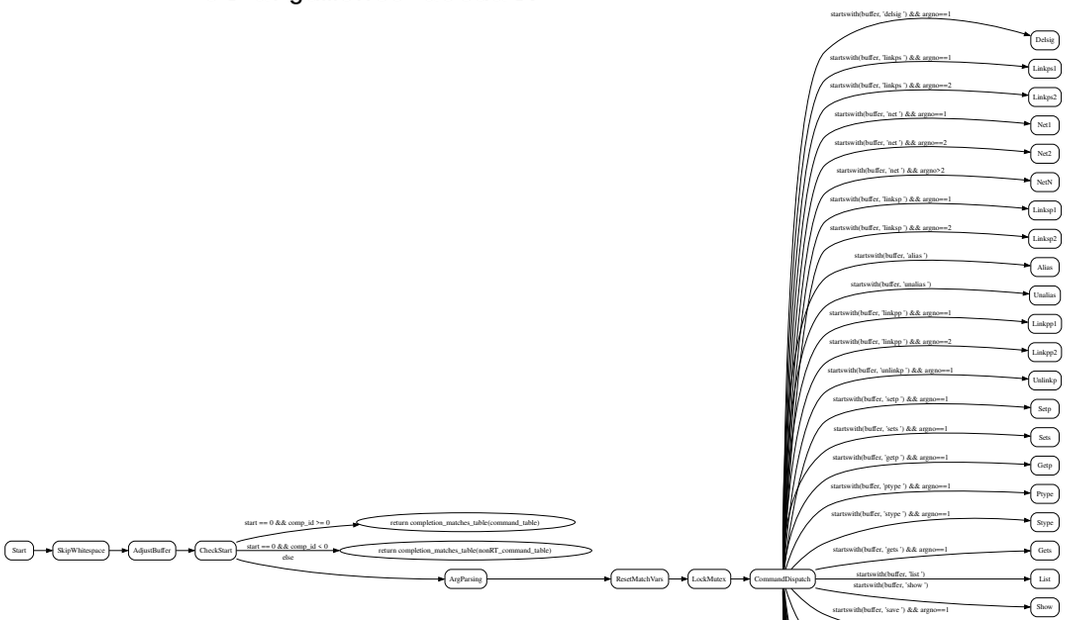
25. `static char *nextword(char *s) {}`

Descripción: Pequeño parser que permite recorrer una cadena palabra por palabra, ignorando separadores comunes en expresiones HAL (, \t, =, <, >).



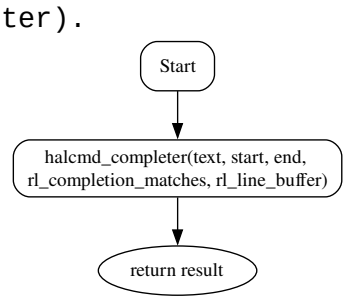
26. `char **halcmd_completer(const char *text, int start, int end, hal_completer_func func, char *buffer) {}`

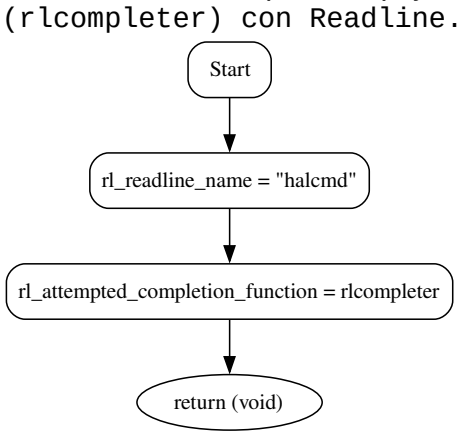
Descripción: Pieza central del sistema de autocompletado de HAL. Integra todos los generadores y decide cual usar según el comando y el argumento actual.



27. `static char **rlcompleter(const char *text, int start, int end) {}`

Descripción: **`Wrapper minimalista`** que conecta el sistema de autocompletado de GNU Readline con el autocompletador específico de HAL (halcmd_completer).



28. `void halcmd_init_readline() {}`

Descripción: **`Punto de arranque del sistema de autocompletado HAL`**; define el nombre de la sesión (halcmd) y conecta la función de autocompletado (rlcompleter) con Readline.