BCL

Sistema de Extensiones

Referencia para Desarrolladores

Versión 2.0.0

BCL Development Team

Noviembre 2025

Índice

1. Introducción

BCL v2.0 introduce un sistema de extensiones dinámicas que permite añadir nuevos comandos al lenguaje sin modificar el intérprete. Las extensiones son bibliotecas compartidas (`.so`) que se cargan en tiempo de ejecución mediante el comando `LOAD`.

1.1. Características

- **Carga dinámica**: Módulos se cargan con `dlopen()`
- **API estable**: Interfaz versionada para compatibilidad
- **Sin recompilación**: Añadir comandos sin recompilar BCL
- **Acceso completo**: API para variables, valores, y errores
- **Portabilidad**: Funciona en Linux, macOS, *BSD

1.2. Requisitos

- BCL 2.0.0 o superior
- Compilador C (gcc, clang)
- Soporte para bibliotecas compartidas (`-ldl`)

2. Comando LOAD

2.1. Sintaxis

```
LOAD ruta_extension
```

2.2. Descripción

Carga una extensión dinámica desde la ruta especificada. La extensión debe ser un archivo `.so` que exporte la función `bcl_extension_init`.

2.3. Ejemplo

```
#!/usr/bin/env bcl

# Cargar extensión SOCKET
LOAD "extensions/socket.so"

# Ahora el comando SOCKET está disponible
SET server [SOCKET SERVER 8080]
PUTS "Servidor escuchando en puerto 8080"
```

2.4. Manejo de Errores

- Si la extensión no existe, retorna error
- Si no exporta `bcl_extension_init`, retorna error
- Si la inicialización falla, retorna error
- Si ya está cargada, retorna error

3. API de Extensiones

3.1. Estructura bcl_extension_api_t

Cada extensión recibe un puntero a esta estructura en su función de inicialización:

```c
typedef struct {
    int version;  /* Debe ser BCL_EXTENSION_API_VERSION */

    /* Registrar nuevo comando */
    int (*register_command)(bcl_interp_t *interp,
                            const char *name,
                            bcl_extension_cmd_func_t func);

    /* Establecer mensaje de error */
    void (*set_error)(bcl_interp_t *interp,
                      const char *fmt, ...);

    /* Crear/destruir valores */
    bcl_value_t *(*value_create)(const char *str);
    void (*value_destroy)(bcl_value_t *val);
    const char *(*value_get)(bcl_value_t *val);

    /* Manipular variables */
    bcl_result_t (*var_set)(bcl_interp_t *interp,
                            const char *name,
                            const char *value);
    bcl_value_t *(*var_get)(bcl_interp_t *interp,
                            const char *name);
} bcl_extension_api_t;
```

3.2. Función de Inicialización

Toda extensión debe exportar:

```c
int bcl_extension_init(bcl_extension_api_t *api);
```

Esta función debe:

1. Verificar `api->version == BCL_EXTENSION_API_VERSION`
2. Guardar punteros a funciones de la API
3. Registrar comandos usando `api->register_command()`
4. Retornar 0 si éxito, -1 si error

3.3. Función de Comando

Los comandos registrados tienen esta firma:

```
bcl_result_t mi_comando(bcl_interp_t *interp,
                        int argc,
                        char **argv,
                        bcl_value_t **result);
```

Donde:

- `interp`: Intérprete BCL
- `argc`: Número de argumentos (sin contar el nombre del comando)
- `argv`: Array de argumentos como strings
- `result`: Puntero donde guardar el resultado

Debe retornar `BCL_OK` (0) si éxito, `BCL_ERROR` (1) si error.

4. Extensión SOCKET

4.1. Descripción

La extensión SOCKET implementa comunicación TCP similar a TCL. Permite crear servidores y clientes TCP, aceptar conexiones, enviar y recibir datos.

4.2. Compilación

```
cd extensions
make socket.so
```

4.3. Subcomandos

4.3.1. SOCKET SERVER

Sintaxis:

```
SOCKET SERVER puerto
```

Descripción: Crea un servidor TCP escuchando en el puerto especificado.
Retorna: Handle del socket (ej: `sock0`)
Ejemplo:

```
SET server [SOCKET SERVER 8080]
PUTS "Servidor: $server"
```

4.3.2. SOCKET CLIENT

Sintaxis:

```
SOCKET CLIENT host puerto
```

Descripción: Conecta a un servidor TCP.
Retorna: Handle del socket
Ejemplo:

```
SET client [SOCKET CLIENT "localhost" 8080]
PUTS "Conectado: $client"
```

4.3.3. SOCKET ACCEPT

Sintaxis:

```
SOCKET ACCEPT handle_servidor
```

Descripción: Acepta una conexión entrante (bloquea hasta que llegue un cliente).
Retorna: Handle del cliente conectado
Ejemplo:

```
SET client [SOCKET ACCEPT $server]
PUTS "Cliente conectado: $client"
```

4.3.4. SOCKET SEND

Sintaxis:

```
SOCKET SEND handle datos
```

Descripción: Envía datos a través del socket.
Retorna: Número de bytes enviados
Ejemplo:

```
SET bytes [SOCKET SEND $client "Hello, World!\n"]
PUTS "Enviados $bytes bytes"
```

4.3.5. SOCKET RECV

Sintaxis:

```
SOCKET RECV handle [max_bytes]
```

Descripción: Recibe datos del socket.
Retorna: Datos recibidos como string
Ejemplo:

```
SET data [SOCKET RECV $client 1024]
PUTS "Recibido: $data"
```

4.3.6. SOCKET CLOSE

Sintaxis:

```
SOCKET CLOSE handle
```

Descripción: Cierra el socket.
Ejemplo:

```
SOCKET CLOSE $client
SOCKET CLOSE $server
```

5. Ejemplos Completos

5.1. Servidor Echo Simple

```
#!/usr/bin/env bcl

LOAD "extensions/socket.so"

# Crear servidor en puerto 9999
SET server [SOCKET SERVER 9999]
PUTS "Servidor echo escuchando en puerto 9999"

# Aceptar cliente (bloquea)
PUTS "Esperando cliente..."
SET client [SOCKET ACCEPT $server]
PUTS "Cliente conectado!"

# Echo loop
WHILE 1 DO
    SET data [SOCKET RECV $client 1024]

    IF [EXPR [STRING LENGTH $data] == 0] THEN
        PUTS "Cliente desconectado"
        BREAK
    END

    PUTS "Recibido: $data"
    SOCKET SEND $client $data
END

# Cleanup
SOCKET CLOSE $client
SOCKET CLOSE $server
```

5.2. Cliente TCP

```
#!/usr/bin/env bcl

LOAD "extensions/socket.so"

# Conectar al servidor
SET client [SOCKET CLIENT "localhost" 9999]
PUTS "Conectado al servidor"

# Enviar mensaje
SOCKET SEND $client "Hola desde BCL!\n"

# Recibir respuesta
SET response [SOCKET RECV $client 1024]
PUTS "Respuesta: $response"

# Cerrar
SOCKET CLOSE $client
```

5.3. Servidor Multi-Cliente (Simple)

```
#!/usr/bin/env bcl

LOAD "extensions/socket.so"

PROC HANDLE_CLIENT WITH client DO
    PUTS "Nuevo cliente conectado"

    WHILE 1 DO
        SET data [SOCKET RECV $client 1024]

        IF [EXPR [STRING LENGTH $data] == 0] THEN
            BREAK
        END

        PUTS "Cliente dice: $data"
        SOCKET SEND $client "Echo: $data"
    END

    SOCKET CLOSE $client
    PUTS "Cliente desconectado"
END

# Servidor principal
SET server [SOCKET SERVER 8080]
PUTS "Servidor multi-cliente en puerto 8080"

WHILE 1 DO
    PUTS "Esperando cliente..."
    SET client [SOCKET ACCEPT $server]

    # En producción, usar fork o hilos
    # Aquí manejamos uno a la vez
    HANDLE_CLIENT $client
END
```

6. Crear Tu Propia Extensión

6.1. Plantilla Básica

```
#include <stdio.h>
#include <string.h>

/* API de BCL */
static struct {
    int version;
    int (*register_command)(void *interp,
                            const char *name,
                            void *func);
    void (*set_error)(void *interp,
                      const char *fmt, ...);
    void *(*value_create)(const char *str);
    void (*value_destroy)(void *val);
```

```c
    const char *(*value_get)(void *val);
} bcl_api;

/* Tu comando */
static int mi_comando(void *interp, int argc,
                      char **argv, void **result) {
    if (argc != 1) {
        bcl_api.set_error(interp,
            "wrong # args: should be \"MICMD arg\"");
        return 1; /* BCL_ERROR */
    }

    /* Procesar... */
    *result = bcl_api.value_create("OK");
    return 0; /* BCL_OK */
}

/* Inicialización */
int bcl_extension_init(void *api_ptr) {
    /* Copiar API */
    memcpy(&bcl_api, api_ptr, sizeof(bcl_api));

    /* Verificar versión */
    if (bcl_api.version != 1) return -1;

    /* Registrar comando */
    if (bcl_api.register_command(NULL, "MICMD",
                                 mi_comando) != 0) {
        return -1;
    }

    return 0;
}
```

6.2. Compilación

```
gcc -Wall -Wextra -fPIC -shared \
    -o mi_extension.so mi_extension.c
```

6.3. Uso

```
LOAD "mi_extension.so"
MICMD "argumento"
```

7. Referencia Rápida

7.1. Comandos del Sistema

Comando	Descripción
`LOAD path`	Carga extensión dinámica

7.2. Extensión SOCKET

Comando	Descripción
`SOCKET SERVER port`	Crea servidor TCP
`SOCKET CLIENT host port`	Conecta a servidor
`SOCKET ACCEPT handle`	Acepta conexión
`SOCKET SEND handle data`	Envía datos
`SOCKET RECV handle [max]`	Recibe datos
`SOCKET CLOSE handle`	Cierra socket

BCL Extensions System v2.0

Para más información, visite:
https://github.com/yourusername/bcl