Guía para Añadir Nuevos Argumentos al Proyecto DynaRange

Introducción

Este documento describe el procedimiento estándar para añadir un nuevo argumento de configuración al proyecto. Gracias a la refactorización del sistema de argumentos en una clase centralizada (ArgumentManager), el proceso es ahora modular y seguro, garantizando que tanto el ejecutable de línea de comandos (**rango**) como la interfaz gráfica (**dynaRangeGui**) se mantengan sincronizados.

El objetivo de esta arquitectura es cumplir con los principios **DRY** (No Repetirte) y **SRP** (Principio de Responsabilidad Única), asegurando que la definición de un argumento (su nombre, tipo, valor por defecto, etc.) exista en **un único lugar**.

Procedimiento para Añadir un Nuevo Argumento

El proceso se divide en 3 pasos obligatorios para que el argumento funcione en el núcleo del programa y en el CLI, y un cuarto paso opcional para integrarlo en la interfaz gráfica.

Usaremos como ejemplo la creación de un argumento hipotético:

- **Nombre largo**: --nuevo-arg
- **Nombre corto**: -na
- **Tipo de dato**: float (lo manejaremos como double)
- **Valor por defecto**: 0.5
- **Rango válido**: de 0.0 a 1.0

Paso 1: Declarar el Argumento en el Registro Central

Este es el paso más importante. Aquí es donde "damos de alta" el nuevo argumento en el sistema.

1. Abre el fichero src/core/arguments/ArgumentManager.cpp.
2. Localiza el método ArgumentManager::RegisterAllArguments().
3. Añade una nueva entrada al mapa m_descriptors con la definición de tu argumento.

Ejemplo de código:

C++

```
// Dentro de ArgumentManager::RegisterAllArguments()

    // ... (otras definiciones de argumentos) ...
    m_descriptors["sensor-resolution-mpx"] = {"sensor-resolution-mpx", "", _("Sensor
```

```
resolution..."), ArgType::Double, 0.0};

    // --- AÑADIR LA NUEVA LÍNEA AQUÍ ---
    m_descriptors["nuevo-arg"] = {"nuevo-arg", "na", _("Help text for the new
argument"), ArgType::Double, 0.5, false, 0.0, 1.0};
    // -----------------------------------

    m_descriptors["snr-threshold-is-default"] = {"snr-threshold-is-default", "", "",
ArgType::Flag, true};
```

Desglose de la línea añadida:

- "nuevo-arg": El nombre largo (y la clave del mapa).
- "na": El nombre corto para el CLI.
- _("..."): El texto de ayuda que se mostrará en la línea de comandos.
- ArgType::Double: El tipo de dato.
- 0.5: El valor por defecto.
- false: Indica que el argumento no es obligatorio.
- 0.0, 1.0: Los valores mínimo y máximo para la validación automática.

Paso 2: Integrar el Argumento en el Parser del CLI

Ahora que el sistema conoce el argumento, debemos indicarle a la librería CLI11 cómo leerlo desde la línea de comandos.

1. En el mismo fichero (ArgumentManager.cpp), localiza el método ArgumentManager::ParseCli().
2. Añade una nueva línea para enlazar el argumento a una variable temporal dentro de la estructura temp_opts.
3. Añade otra línea al final del método para copiar el valor parseado desde temp_opts a nuestro mapa de valores centralizado.

Ejemplo de código:

C++

```
// Dentro de ArgumentManager::ParseCli()

    ProgramOptions temp_opts;
    // ...
    app.add_option("--sensor-resolution-mpx", temp_opts.sensor_resolution_mpx, _("Sensor
resolution..."));

    // --- 1. AÑADIR ENLACE A CLI11 ---
    app.add_option("--nuevo-arg", temp_opts.nuevo_arg, _("Help
text..."))->check(CLI::Range(0.0, 1.0));

    try {
        app.parse(argc, argv);
    } // ...

    // ... (código para copiar otros valores) ...
```

```
    m_values["sensor-resolution-mpx"] = temp_opts.sensor_resolution_mpx;
    m_values["snr-threshold-is-default"] = (snr_opt->count() == 0);

    // --- 2. COPIAR VALOR AL MAPA CENTRAL ---
    m_values["nuevo-arg"] = temp_opts.nuevo_arg;
```

Paso 3: Exponer el Argumento al Motor de Análisis

Para que el resto del programa (el motor de procesado, la generación de plots, etc.) pueda utilizar el valor del nuevo argumento de forma segura, debemos añadirlo a la estructura ProgramOptions.

1. Abre el fichero src/core/arguments/ProgramOptions.hpp.
2. Añade un nuevo miembro a la struct ProgramOptions con su tipo y valor por defecto.
3. C++

```
// Dentro de struct ProgramOptions
struct ProgramOptions {
    // ... (miembros existentes) ...
    double sensor_resolution_mpx = 0.0;
    double nuevo_arg = 0.5; // <-- AÑADIR AQUÍ
};
```

4.
5. Abre src/core/arguments/ArgumentManager.cpp y localiza el método ArgumentManager::ToProgramOptions().
6. Añade la línea que transfiere el valor desde el mapa del gestor a la instancia de la estructura opts.
7. C++

```
// Dentro de ArgumentManager::ToProgramOptions()
ProgramOptions ArgumentManager::ToProgramOptions() {
    ProgramOptions opts;
    // ... (otras asignaciones) ...
    opts.sensor_resolution_mpx = Get<double>("sensor-resolution-mpx");

    // --- AÑADIR LA LÍNEA DE TRANSFERENCIA AQUÍ ---
    opts.nuevo_arg = Get<double>("nuevo-arg");

    // ... (resto del método) ...
    return opts;
}
```

8. ¡Listo! Con estos tres pasos, el argumento --nuevo-arg ya es una parte funcional del ejecutable rango, con su valor por defecto, validación y ayuda automática.

Paso 4 (Opcional): Conectar el Argumento a la Interfaz Gráfica (GUI)

Si el nuevo argumento debe ser configurable desde la GUI:

1. **En wxFormBuilder**: Añade el control visual necesario (un wxSlider, wxTextCtrl, etc.) al panel correspondiente.
2. **En DynaRangeFrame.hpp y .cpp**: Crea un nuevo método "getter" que lea y devuelva el valor del nuevo control. Por ejemplo: double GetNuevoArg() const;.
3. **En GuiPresenter.cpp**: Localiza el método UpdateManagerFromView().
4. Añade una línea que use el nuevo "getter" de la vista para actualizar el valor en el ArgumentManager.
5. C++

```
// Dentro de GuiPresenter::UpdateManagerFromView()
void GuiPresenter::UpdateManagerFromView() {
    auto& mgr = ArgumentManager::Instance();
    // ... (otras llamadas a mgr.Set) ...
    mgr.Set("patch-ratio", m_view->GetPatchRatio());

    // --- AÑADIR LA NUEVA CONEXIÓN AQUÍ ---
    mgr.Set("nuevo-arg", m_view->GetNuevoArg());
}
```

6. Con este último paso, el valor del control de la GUI se propagará automáticamente a todo el sistema cada vez que se genere una previsualización del comando o se inicie un análisis.