Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas
Organización de Lenguajes y Compiladores 2
Primer semestre 2019
Catedráticos: Ing. Bayron López, Ing. Erick Navarro y Ing. Edgar Saban
Tutores académicos: Mike Gutiérrez, Javier Navarro, Julio Arango

Creator XML

Primer proyecto de laboratorio

Contenido

1 Objetivos

1.1 Objetivo general

Aplicar los conocimientos del curso de Organización de Lenguajes y Compiladores 2 en el desarrollo de una aplicación.

1.2 Objetivos específicos

- Utilizar herramientas para la generación de analizadores léxicos y sintácticos.
- Realizar análisis semántico a los diferentes lenguajes a implementar.
- Introducir al estudiante a nuevas tecnologías que utilizan los conceptos del desarrollo de compiladores.
- Que el estudiante aplique los conocimientos adquiridos en la carrera para el desarrollo de un intérprete de máquina de pila.

2 Descripción

2.1 Antecedentes

En la actualidad se ha llegado a la conclusión de que existe la necesidad de crear interfaces gráficas de forma abstracta, es decir, se desea crear interfaces de usuario (UI) independientes del lenguaje de programación que se esté usando para crear la aplicación, con esto se logrará crear independencia total entre UI con la lógica de programación.

2.2 Descripción de la solución

Creator XML permitirá definir interfaces de usuario (UI) utilizando plantillas escritas en un lenguaje basado en **XML**[1]. **Creator XML** se integrará con el lenguaje de programación funcional **FuncionScript**, para generar las interfaces de usuario, recolectar y almacenar los datos que se reciban por medio de las interfaces de usuario.

2.3 Componentes de la aplicación

- **FuncionScript (FS):** FuncionScript es un lenguaje de programación basado en el paradigma **fuctional programing,** este es un paradigma de programación declarativo, es decir, los algoritmos son desarrollados únicamente usando expresiones (lógicas, aritméticas y relaciones).
  Este lenguaje permitirá utilizar el motor **Creator XML** para poder generar interfaces de usuario (UI) a partir de plantillas XML escritas en **GenericXM**L. **FuncionScript** podrá transformar los datos ingresados en las interfaces de usuario (UI) generadas a partir de las plantillas XML, los datos procesados con FuncionScript serán convertidos de nuevo en un archivo XML. FuncionScript será capaz de leer los datos que han sido almacenados en archivos XML y a partir de estos permitirá la generación de reportes.

[1] XML es un meta lenguaje que se utiliza para almacenar datos de forma legible.

- **GenericXML (GX):** GenericXML será un lenguaje basado en XML, este se utilizará para definir las interfaces de usuario (UI). Cada interfaz de usuario tendrá también un archivo GenericXML que servirá para almacenar los datos que fueran capturados por medio de la interfaz gráfica.

- **ExtremeEditor (EE):** Este será un IDE que permitirá la creación y edición de archivos XML y archivos **FS.**.

2.4 Flujo general de la aplicación

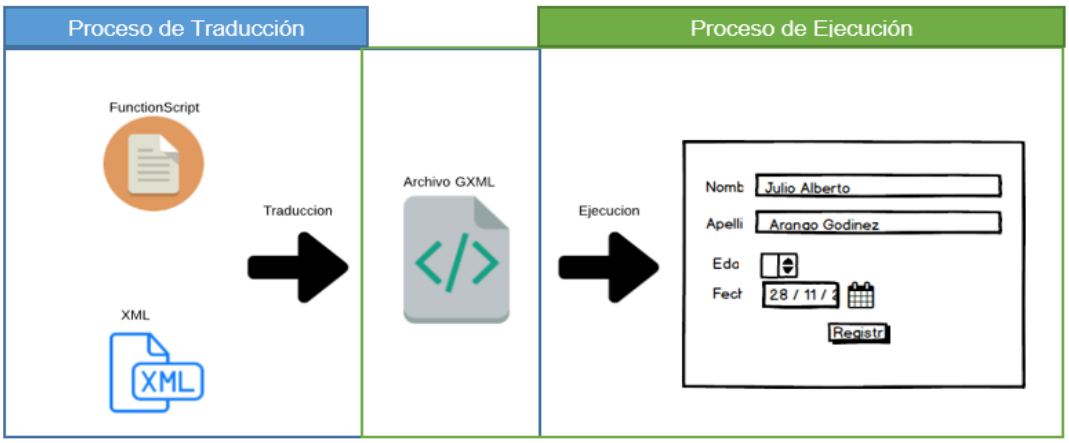
La imagen en esta página muestra como es el flujo de los componentes de la aplicación, comenzando con una entrada conformada por al menos 2 archivos, un archivo XML y uno o varios FuncionScript los cuales son implementados de tal forma que puedan declararle funcionalidad a la interfaz que se desea crear.

Proceso de Traducción

La aplicación al tener todas las entradas pasa por un proceso de traducción, donde uniendo todas las entradas genera un archivo GXML el cual contendrá todos los datos de la interfaz de forma más compacta.

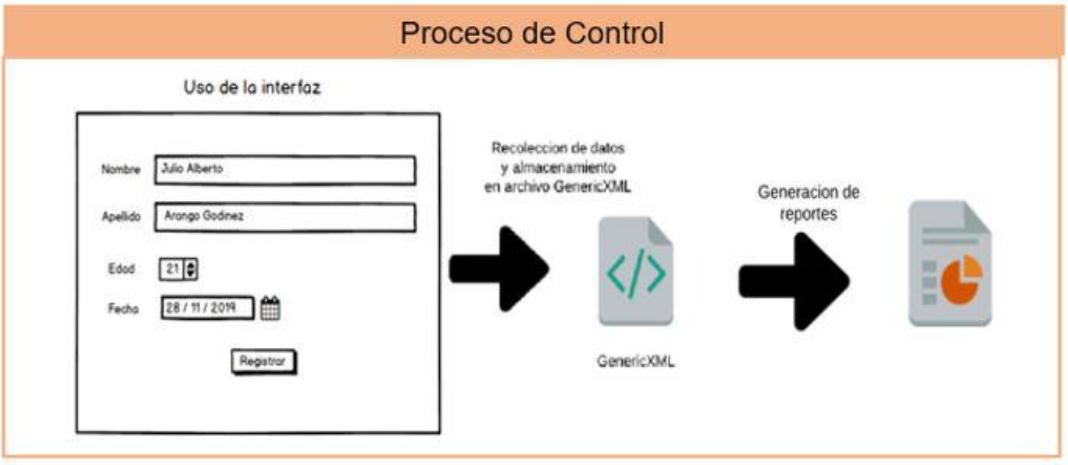
Proceso de Ejecución

La aplicación contará con un proceso de ejecución en la cual tomará un archivo GXML y con toda la información contenida en ella generará una salida que será la interfaz para el usuario.



Proceso de control

Una vez creada la interfaz, esta llevará un proceso de control, donde guardará en un archivo toda la información necesaria cada vez que esta se utilice, esta información contendrá fecha, hora, tiempo de uso entre otra información y permitiendo así poder generar un reporte con esta información.



Es importante aclarar que el archivo que guardará la información capturada utilizará el mismo formato que GXML, pero este será otro archivo independiente al archivo con el cual se genera la Interfaz, utilizará la misma sintaxis y extensión, pero este archivo únicamente guardará los datos de cada uso de la interfaz.

2.5 Flujo específico de la aplicación

2.5.1 Definición de la interfaz de usuario por medio GenericXML

La definición de cada componente de la interfaz se encontrará en archivos GenericXML, estos archivos servirán de plantillas para que el motor de Creator XML junto con FuncionScript puedan generar la interfaz descrita.

A continuación, se describen los pasos a seguir para esta etapa:

Generación de archivos FuncionScript

La generación de archivos FS se realizará sobre el IDE de la aplicación, en estos archivos se definirán las funciones por las cuales va a interactuar el usuario de la interfaz.

Generación de archivo XML

La generación del archivo XML se realizará sobre el IDE de la aplicación, esta contendrá la sintaxis propia del lenguaje con la cual se generará la interfaz que el usuario desea.

Traducción de los archivos XML a GXML

Luego de la implementación de los archivos FS con XML y terminada la definición de la interfaz en archivos XML la aplicación realizará una traducción sobre el archivo a lenguaje GXML, este lenguaje será el que contendrá de manera genérica la interfaz, para después poder ser usada por cualquier usuario.

2.5.2 Transformación de la plantilla Genérica XML con FuncionScript

FuncionScript se encargará de transformar los datos ingresados en las interfaces de usuario, por medio de funciones nativas, en archivos GenericXML.

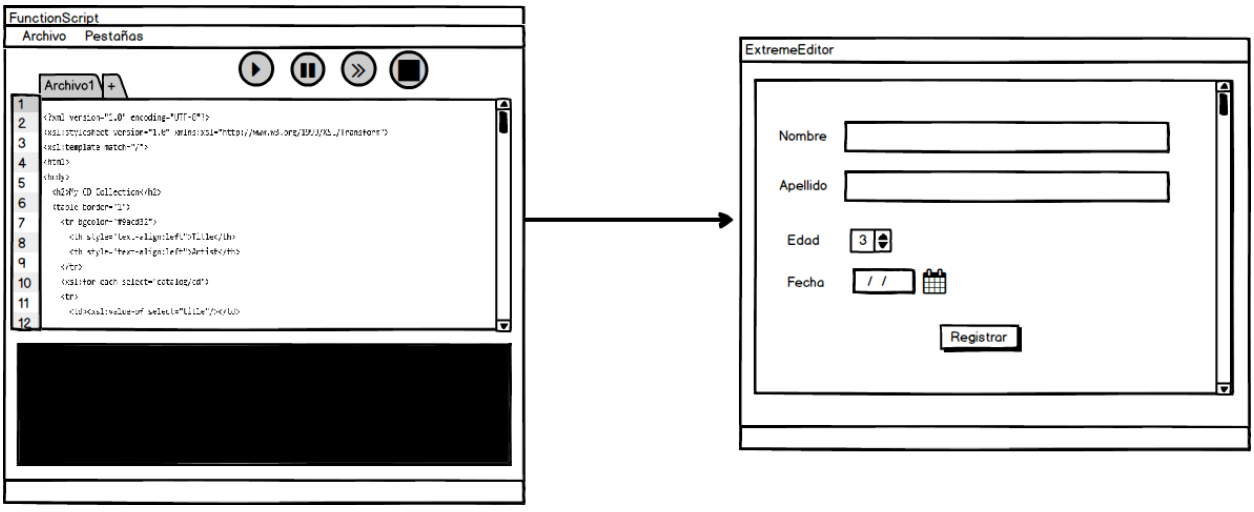
A continuación, se describen los pasos a seguir para esta etapa:

Generación de interfaz

Al tener la interfaz ya traducida a lenguaje GXML, esta podrá generarse obteniendo los componentes gráficos y podrá usarse en cualquier medio deseado.

Uso de la funcionalidad implementada de FS

Con la interfaz ya generada los usuarios serán capaces de utilizarla, con cada interacción realizada, si se implementó una función de FS, esta se activará y ejecutará.



2.5.3 Recolección de datos por medio de la interfaz de usuario

La recolección de datos se realizará por medio de la interfaz sobre un archivo de GenericXML, guardando todos los datos ingresados para que FuncionScript sea capaz de interpretarlos y generar reportes.

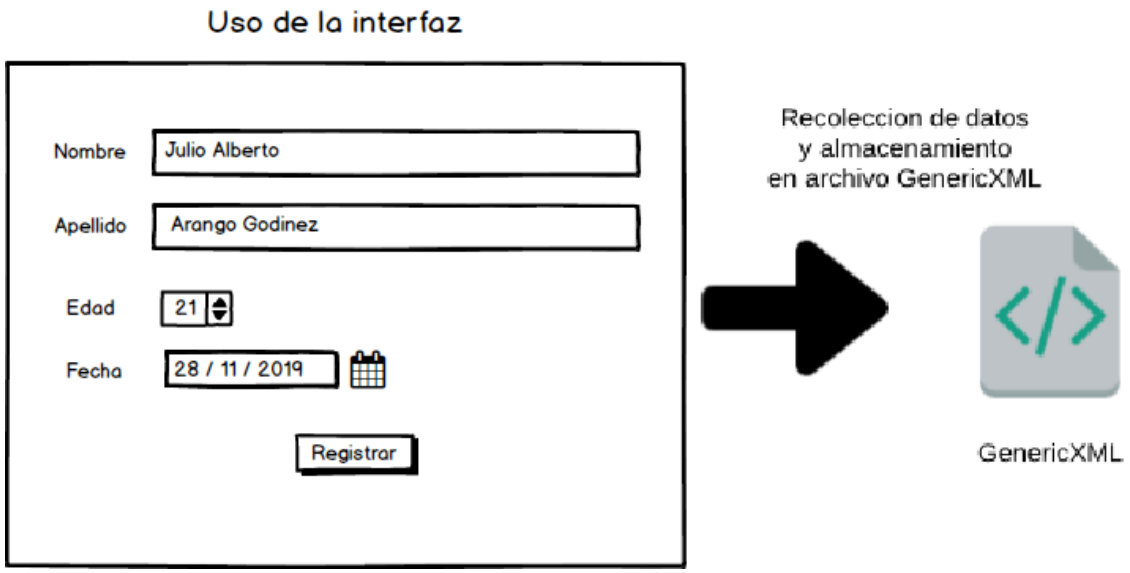
A continuación, se describen los pasos a seguir para esta etapa:

Uso de la interfaz

Todo usuario será capaz de interactuar con la interfaz generada y esta podrá recolectar toda la información ingresada por este usuario.

Recolección de datos sobre archivo GenericXML

Toda la información generada en cada uso de la interfaz se guardará en un archivo GenericXML, este archivo será único para cada interfaz y se utilizará más adelante para análisis e interpretación de datos.



2.5.4 Tratamiento de los datos recolectados por medio de FuncionScript

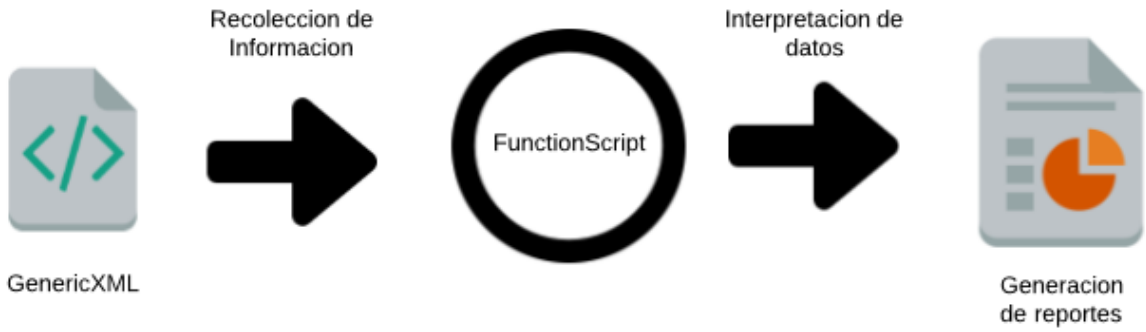
FuncionScript contará con la funcionalidad de leer todos los datos y a su vez podrá interpretarlos de tal forma que pueda crear un reporte actualizado con toda la información recolectada.

A continuación, se describen los pasos a seguir para esta etapa:

Extracción y análisis de datos guardados

En cada interfaz existe un archivo GenericXML con toda la información generada de cada uso de la interfaz, toda esta información se recolectará guardándola y analizándola.

Luego de ser analizada toda la información FuncionScript tendrá funciones nativas capaces de generar un informe general de dichos datos.



2.5.5 Ejemplo de ejecución

En esta se sección se mostrará un ejemplo de cómo será el funcionamiento de **CreatorXml**, la herramienta CreatorXml tiene como fin definir interfaces de usuario de forma abstracta por medio de archivos gxml y convirtiendo estos archivos en interfaces de usuario (UI), haciendo uso del lenguaje **FuncionScript.** El paso intermedio será capturar datos por medio de las interfaces creadas y almacenarlos en archivos gxml, por último, se usará **FuncionScript** para generar reportes a partir de los datos que se hayan capturado.

El proceso de utilización de CreatorXml se resume en los siguientes pasos:

1. Definición del archivo **Gxml**
2. Transformación del archivo **Gxml** en una interfaz de usuario por medio de **FuncionScript**
3. Captura de datos por medio de la interfaz de usuario creada.
4. Creación de reportes usando **FuncionScript** a partir de los datos capturados en el paso 3.

2.5.5.1Definición del ejemplo

Para este ejemplo buscamos definir un proyecto el cual será capaz de recopilar datos de los usuarios por medio de una interfaz amigable e intuitiva, estos datos serian

- **Nombre:** dando al usuario un campo de tipo texto donde pueda ingresar su nombre.
- **Apellido:** dando al usuario un campo de tipo texto donde pueda ingresar su apellido.
- **Teléfono:** dando al usuario un campo de tipo texto donde pueda ingresar su teléfono, esta tendrá una particularidad, la cual mostrará el número “4545-4040” por defecto, dando así al usuario un ejemplo de cómo debe llenar dicho campo.

- **Selección de País:** dando al usuario un campo desplegable con varias opciones de países válidos, estos países serán: Guatemala, El Salvador, Honduras y Costa Rica.

También la interfaz contará con un botón con el texto “Enviar”, este botón tendrá la tarea de recolectar toda la información que el usuario a ingresado, finalizando así el rol del usuario con el proyecto.

Una vez ya generada la interfaz del proyecto, esto será compartido a diferentes usuarios interesados, de todos los usuarios que introduzcan su información, nos interesa conocer mediante un reporte los nombres únicamente de los usuarios de “Guatemala”.

Para lograr dicho proyecto se deben de seguir la siguiente secuencia de pasos en la aplicación:

Paso 1: Definición del archivo Gxml

El primer paso será definir una interfaz gráfica por medio de un archivo **gxml,** la imagen 2.1 muestra un archivo gxml en el que se define un formulario que tiene los siguientes componentes:

1. **Un campo de tipo texto que servirá para capturar el nombre de un usuario**
2. **Un campo de tipo texto que servirá para capturar el apellido de un usuario**
3. **Un campo de tipo texto para capturar un número de teléfono**
4. **un campo de tipo dropdown (menú desplegable) que servirá para escoger un país de origen.**

La imagen 2.1 muestra cómo serán definidas las interfaces de usuario por medio de Gxml.

```
<ventana id="principal" tipo="principal">
  <contenedor id="Uno">
    <control tipo="texto" alto="25" ancho="40" nombre="primer nombre">
      <Texto>
        Nombre
      </Texto>
    </Control>
    <Control type="texto" alto="25" ancho="40" nombre="Segundo nombre">
      <Texto>
        Apellido
      </Texto>
    </Control>
    <Control type="texto" alto="25" ancho="40" nombre="teléfono">
      <Texto>
        Teléfono
      </Texto>
      <Defecto>
        4545-4040
      </Defecto>
    </Control>
```

Definición de nombre

Definición de apellido

Definición de teléfono

Teléfono por defecto

```
<Control tipo="desplegable" alto="25" ancho="40" nombre="país">
    <Texto>
        Ciudad
    </Texto>
    <ListaDatos>
        <Dato>Guatemala</ Dato >
        < Dato >El Salvador</ Dato >
        < Dato >Honduras</ Dato >
        < Dato >Costa Rica</ Dato >
    </ListaDatos>
</Control>
</Contenedor>
<Contenedor id="dos">
    <Enviar alto="25" ancho="40">
        <texto>
            Enviar
        </texto>
    </Enviar>
</contenedor>
</ventana>
```

Definición de País

Opciones de País

Definición del botón Enviar

Imagen 2.1

Paso 2: Transformación del archivo gxml por medio de FuncionScript

El IDE de la aplicación contará con un editor de texto en el cual se pueden crear archivos y también contará con la opción de abrir archivo, cualquiera de las 2 formas al haber terminado de definir la interfaz en un archivo GXML, este a través del IDE de la aplicación será analizado y ejecutado desde el momento en el que el usuario realice la compilación del archivo, esto llamará funciones nativas de FuncionScript para transformar el archivo Gxml en un UI.

El archivo **gxml** que se muestra en la imagen 2.1 se transformará con la ayuda de FuncionScript en el formulario que se ve en la imagen 2.2

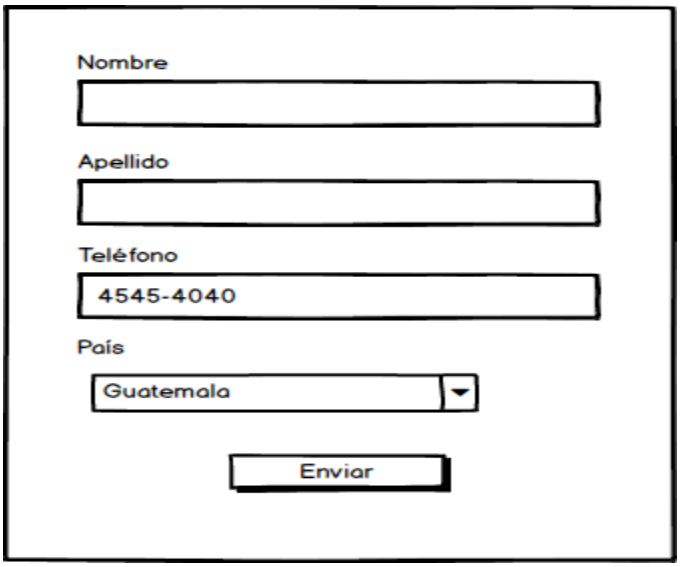


Imagen 2.2

Paso 3: Recopilación de datos utilizando la interfaz de usuario creada

Una vez definida la interfaz gráfica por medio de **gxml y** luego de haber transformado este **gxml** con **FuncionScript,** los usuarios entrarán en contacto con la UI creada, y los datos que se capturen serán almacenados en otro archivo **gxml.**

El formulario que se muestra en la imagen 2.2 servirá para colectar datos (Nombre, apellido, teléfono, y país, para este ejemplo). Cada vez que un usuario interactúe con la interfaz y de clic sobre el botón enviar, los datos recolectados se guardarán en un archivo gxml como el que se muestra en la imagen 2.3.

```
<lista tipo="principal">
   <principal>
      <nombre>nombre intento 1</nombre>
      <apellido>apellido intento 1</apellido>
      <teléfono>teléfono intento 1</teléfono>
      <país>Guatemala</país>
   </principal>
   <principal>
      <nombre>nombre intento 2</nombre>
      <apellido>apellido intento 2</apellido>
      <teléfono>teléfono intento 2</teléfono>
      <país>Guatemala</país>
   </principal>
   <principal>
      <nombre>nombre intento 3</nombre>
      <apellido>apellido intento 3</apellido>
      <teléfono>teléfono intento 3</teléfono>
      <país>Honduras</país>
   </principal>
</lista>
```

Imagen 2.3

Cada interfaz de usuario creada tendrá un archivo **gxml** que servirá para almacenar todos los datos que hayan sido capturados a través de la UI generada.

La imagen 2.3 muestra el archivo Gxml relacionado a la interfaz gráfica de la **Imagen 2.2,** esta imagen muestra el resultado de haber usado el formulario por tres diferentes usuarios.

Paso 4: Creación de reportes usando funcionScript

Una vez se disponga de datos, se utilizará el lenguaje **funcionScript** para poder hacer reportes a partir de la información recolectada, vea la **imagen 2.4**.

```
/**/
var data = crearArrayDesdeArchivo('../../principalData.gxml');
/**
La variable data contiene la información como un array de la siguiente forma
data = [
{
   nombre: "nombre intento 1",
   apellido: "apellido intento 1",
   teléfono: "teléfono intento 1",
   ciudad: “Guatemala",
},

{
   nombre: “nombre intento 2",
   apellido: “apellido intento 2",
   teléfono: “teléfono intento 2",
   ciudad: “Guatemala",
},

{
   nombre: “nombre intento 3",
   apellido: “apellido intento 3",
   teléfono: “teléfono intento 3",
   ciudad: “Honduras",
},

]
 **/

/**Filtrar los datos por el país, solo selecciona los que tenga el país == Guatemala**/
data = data.filtrar((item) => {
   retornar item.pais == "Guatemala"
});

/**imprimir el nuevo array item por item.
Resultado:
nombre intento 1
```

```
nombre intento 2
**/
data.map((item) => {
    imprimir(item.nombre);
});
```

Imagen 2.4

La imagen 2.4 muestra fragmentos de código que se utilizarán para poder ver el nombre de los usuarios que han seleccionado a Guatemala como país de origen en el formulario de la imagen 2.2.

El resultado de la ejecución del fragmento de código de la imagen 2.4 se muestra en la imagen 2.5.

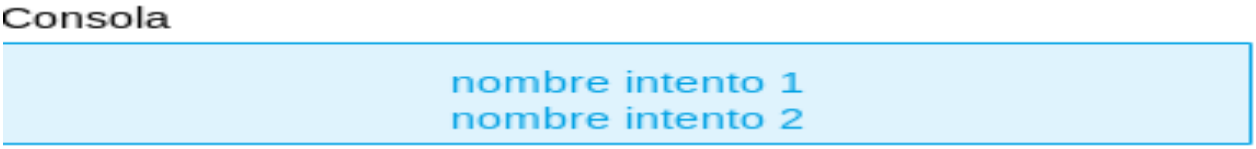


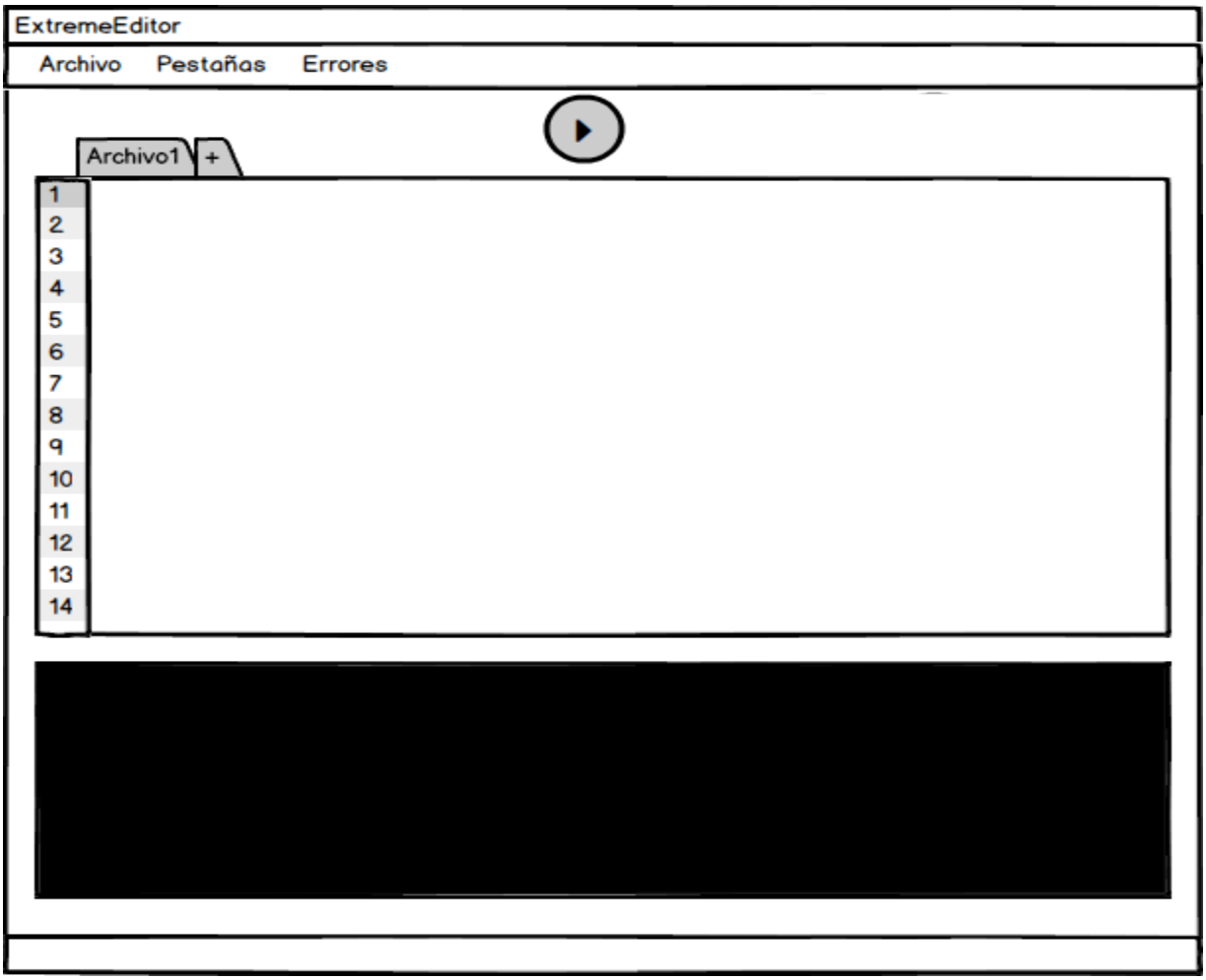
Imagen 2.5

3 ExtremeEditor

A continuación, se describen los componentes que conformaran la aplicación.

3.1 Componentes de la aplicación

El primer componente para los usuarios estará conformado por un IDE que permitirá la creación y edición de archivos tanto XML como archivos de FunctionScript.



Barra de Herramientas

Archivo

- **Abrir:** Funcionalidad que permitirá abrir un archivo principal del lenguaje FS o XML en una nueva pestaña.

- **Guardar:** Permitirá guardar los cambios de la pestaña actual en el archivo.

- **Guardar Como:** Permitirá crear un nuevo archivo con formato FS o XML, con la información que se encuentre en la pestaña actual.

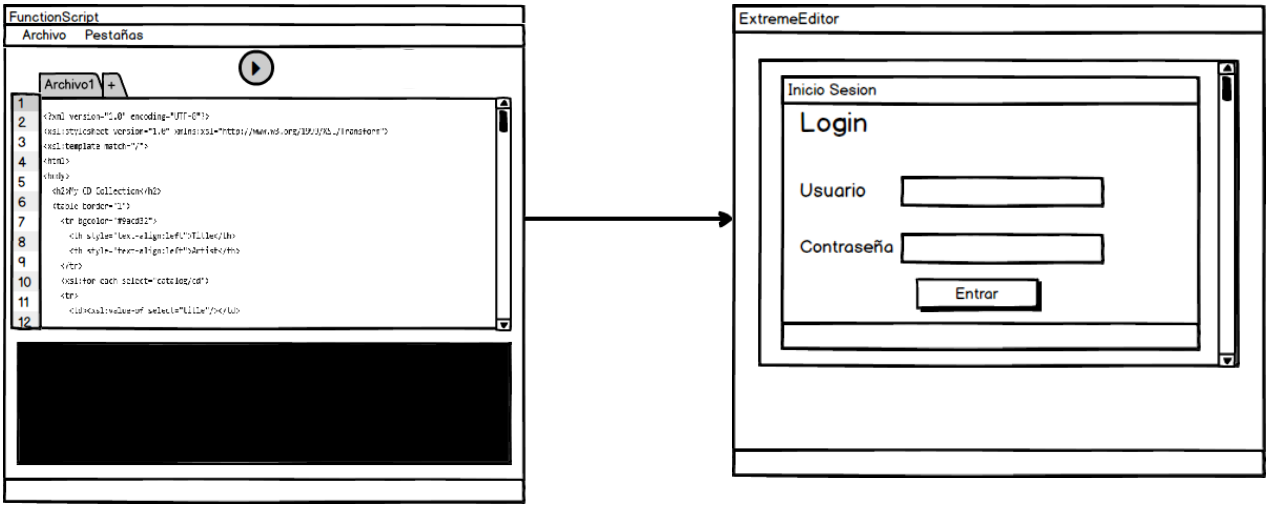
Pestañas

- **Nueva pestaña:** Funcionalidad que permite a la aplicación generar una nueva pestaña.
- **Cerrar pestaña:** Funcionalidad que permite a la aplicación cerrar la pestaña actual.

Errores: Permitirá generar el reporte de errores detectados y deberá mostrarlos en formato HTML o deberá ser mostrado desde la aplicación en forma de tablas.

3.2 Generador de Interfaz

Luego de la generación de archivos FunctionScript y XML estos se incorporarán y el usuario será capaz de visualizar la interfaz generada en un apartado distinto, esto con la finalidad de poder comprobar de forma conjunta la funcionalidad con la interfaz generada.



4 GenericXML

Luego de ser creadas, cada interfaz se verá reflejado con un archivo GenericXML que está basado en el lenguaje XML, este archivo servirá para la recolección de datos de cada interfaz, el tipo de dato y la forma en que estará organizada se determinará por la interfaz generada.

4.1 Lenguaje

GenericXML es un lenguaje basado en XML, todas las instrucciones del gxml deben de venir dentro de los símbolos '<' y '>'. Estas instrucciones tendrán el nombre de **etiquetas.** Cada etiqueta tiene un conjunto específico de elementos opcionales para configurar la apariencia y/o acciones de las etiquetas y toda etiqueta tendrá su inicio y fin.

4.2 Notación dentro del enunciado

Dentro del enunciado, se utilizarán las siguientes notaciones cuando se haga referencial GenericXML.

Para definir la sintaxis y ejemplos de sentencias de código de alto nivel se utilizará un rectángulo gris.

Asignación de colores

Dentro del enunciado y en el editor de texto de la aplicación, para el código de alto nivel, seguirá el formato de colores establecido en la siguiente tabla. Estos colores deberán de ser implementados en el editor de texto.

Color	Token
Azul	Palabras Reservadas
Naranja	Cadenas, caracteres
Morado	Números
Gris	Comentarios
Negro	Otro

Palabras reservadas

Son palabras que no podrán ser utilizadas como identificadores ya que representan una instrucción específica y necesaria dentro del enunciado.

4.3 Características del lenguaje

Esta sección describirá las características que contiene el lenguaje GenericXML.

4.3.1 Case Insensitive

El lenguaje es case insensitive, esto significa que no existirá diferencia entre mayúsculas y minúsculas, esto aplicará tanto para palabras reservadas propias de gxml como para identificadores. Por ejemplo, el id de una ventana de nombre “Formulario” no será distinto a definir otra ventana con el id “contador”.

4.3.2 Comentarios

El lenguaje gxml tendrá la opción de poder posicionar comentarios de una línea o de varias líneas en cualquier parte del documento.

Los comentarios de una línea comienzan con “##” y finalizan con un salto de línea.

```
## Este es un comentario de una línea
```

Los comentarios de varias líneas comenzarán con “#$” y finalizán con “$#”.

```
#$ este es
Un comentario
De varias lineas $#

<Contenedor> #$ este es un Segundo ejemplo $# </Contenedor>
```

4.3.3 Elementos de etiquetas

Cada etiqueta puede contener o no, una o varios elementos, estos elementos servirán para definirle valores o propiedades a las etiquetas las cuales serán descritos por cada etiqueta, cada vez que se haga referencia a:

```
(Elementos)*
```

Se entenderá que en esa área puede venir desde cero a varios elementos válidos para la etiqueta, cada elemento será válido que venga una única vez y pueden venir en cualquier orden.

4.4 Estructura del lenguaje

El lenguaje gxml es un lenguaje de etiquetado basado en xml, en otras palabras, su sintaxis es basada en etiquetas, para agregar componentes a la interfaz, se deberán utilizar las etiquetas que se describen a continuación:

Etiqueta Importar

Etiqueta Importar, esta etiqueta puede venir más de una vez en el archivo, pero únicamente al inicio del archivo, esta etiqueta no contiene elementos y el contenido de la etiqueta se encontrará la dirección relativa o completa del archivo gxml o del archivo FuncionScript a importar, se identificara por medio de su extensión, su funcionalidad consiste en que obtiene todas las ventanas del archivo importado, en caso que existiera una ventana importada con el mismo id de una ventana del archivo que se está analizando, únicamente quedará la del archivo actual.

Sintaxis

```
<Importar>
## Direccion de archivo a importar
</Importar>
```

Ejemplo

```
<Importar>
/Archivo.gxml
</Importar>

<Importar>
C:\Users\Usuario\Documentos\Proyecto\Archivo.gxml
</Importar>
```

Etiqueta Ventana

Etiqueta raíz de cada interfaz, puede venir más de una ventana por archivo gxml, esta etiqueta contiene 2 elementos obligatorios y una opcional, los cuales son:

Obligatorias

- **Id:** Nombre único de identificación, este id no podrá repetirse en ninguna otra ventana.
- **Tipo:** Cada ventana tendrá un tipo de rol que identificara su funcionamiento, los tipos validos son:

- **Principal:** Tipo único, solo una ventana tendrá el tipo principal, esto significa que con dicha ventana comenzará la ejecución de la interfaz.
- **Secundaria:** Segundo tipo de ventana, estas ventanas pueden ser llamadas únicamente por referencias en acciones de otros controladores.

Opcional

- **Color:** Elemento que definirá el color de toda la ventana, este valor debe venir en código hexadecimal, en caso este elemento no viniera se tomará un valor por defecto de blanco (“#ffffff”);
- **AcciónInicial:** Esta etiqueta contendrá una llamada a una función, esta función puede contener parámetros, pero únicamente de datos nativos propios de funcionScript, incluyendo cálculos aritméticos, lógicos o relacionales. Este valor ira entre llaves “{ }”, para evitar conflictos con los parámetros a llamar.
- **AcciónFinal:** Esta etiqueta contendrá una llamada a una función, esta función puede contener parámetros, pero únicamente de datos nativos propios de funcionScript, incluyendo cálculos aritméticos, lógicos o relacionales. Este valor ira entre llaves “{ }”, para evitar conflictos con los parámetros a llamar.

Sintaxis

```
<Ventana (Elementos)*>
## Contenedores
</Ventana>
```

Ejemplo

```
<Ventana id=” primera” tipo=”principal”>
## Contenedores
</Ventana>

<Ventana id=” segunda” tipo=”Secundaria”>
## Contenedores
</Ventana>

<Ventana id=” tercera” tipo=”Secundaria”>
## Contenedores
</Ventana>
```

Etiqueta Contenedor

Etiqueta contenedora, esta etiqueta será la encargada de contener todos los controladores de la interfaz, únicamente puede venir dentro de la etiqueta <Ventana> y no pueden existir dentro de ningún otro contenedor, esta etiqueta contiene 3 elemento obligatorio y 4 etiquetas opcionales, las cuales son:

Obligatoria

- **Id:** Nombre único de identificación, este id no podrá repetirse en ningún otro contenedor dentro de la misma ventana.
- **X:** Elemento que contendrá la posición en X donde se posicionará.
- **Y:** Elemento que contendrá la posición en Y donde se posicionará.

Opcionales

- **Alto:** Elemento que definirá el alto con que contará dicho contenedor, este valor se tomará en pixeles, en caso no viniera se tomará un alto por defecto de 500px.
- **Ancho:** Elemento que definirá el alto con que contará dicho contenedor, este valor se tomará en pixeles, en caso no viniera se tomará un ancho por defecto de 500px.
- **Color:** Elemento que definirá el color de toda la ventana, este valor debe venir en código hexadecimal, en caso este elemento no viniera se tomará el mismo color de la ventana.
- **Borde:** Elemento que definirá un borde alrededor de todo el contenedor, este elemento puede tener únicamente 2 valores
  - **Verdadero:** Valor que definirá que si tendrá borde el contenedor.
  - **Falso:** Valor que definirá que no tendrá borde el contenedor.

Sintaxis

```
<Contenedor (Elementos)* >
## Instrucciones
</Contenedor>
```

Ejemplo

```
<Contenedor id = “Contenedor1” Alto = 100 Ancho = 200>
  ## Etiquetas de Instrucciones
</Contenedor>
<Contenedor id = “Contenedor2”>
  ## Etiquetas de Instrucciones
</Contenedor>
<Contenedor id = “Contenedor3” (Elementos)* >
  ## Etiquetas de Instrucciones
</Contenedor>
```

Etiqueta Texto

Etiqueta texto, esta etiqueta permitirá escribir al usuario un texto el cual se imprimirá en la interfaz, solo puede venir dentro de una etiqueta <Contenedor>, el contenido que este dentro de la etiqueta será el texto que se estará imprimiendo, esta etiqueta contiene 3 elementos obligatorio y contiene 5 elementos opcionales, los cuales son:

Obligatorios

- **Nombre:** Este definirá el identificador propio de la etiqueta texto, este identificador es único y no pueden existir otro con el mismo nombre en la misma ventana.
- **X:** Elemento que contendrá la posición en X donde se posicionará.
- **Y:** Elemento que contendrá la posición en Y donde se posicionará.

Opcionales

- **Fuente:** Elemento que definirá el tipo de fuente con que se escribirá el texto.
- **Tam:** Elemento que definirá el tamaño de fuente con que se escribirá el texto.
- **Color:** Elemento que definirá el color con que se escribirá el texto, este valor debe venir en código hexadecimal, en caso este elemento no viniera, se tomará el color negro (#000000).
- **Negrita:** Elemento que definirá si el texto vendrá en Negrilla, este elemento puede tener únicamente 2 valores
  - **Verdadero:** Valor que definirá que si vendrá en negrilla el texto.
  - **Falso:** Valor que definirá que no vendrá en negrilla el texto.
- **Cursiva:** Elemento que definirá si el texto vendrá en Cursiva, este elemento puede tener únicamente 2 valores
  - **Verdadero:** Valor que definirá que si vendrá en cursiva el texto.
  - **Falso:** Valor que definirá que no vendrá en cursiva el texto.

Sintaxis

```
<Texto (Elementos*) > Texto que se imprimirá  </Texto>
```

Ejemplo

```
<Contenedor id = "Ejemplo">
        ## Etiquetas de Instrucciones
        <Texto> Texto que se imprimirá </Texto>
        ## Etiquetas de Instrucciones
        <Texto Tam = 15 Color= "#000000">
            Texto que se imprimirá con tamaño 15 px y color negro
        </Texto>
        ## Etiquetas de Instrucciones
</Contenedor>
```

Controladores

Los controladores son todos aquellos componentes que servirán en la interfaz para la recolección de datos, toda información ingresada por el controlador se deberá guardar en el archivo de datos gxml definido para la interfaz, existen diversos tipos de controladores, los cuales son:

- **Texto:** Este dará un campo de texto de una línea en el cual el usuario podrá ingresar texto.
- **Numérico:** Este dará un campo numérico seguido de dos flechas, una hacia arriba y otra hacia abajo, los cuales permitirá incrementar o decrementar el valor actual respectivamente.
- **TextoArea:** Este dará un campo de texto de varias líneas en el cual el usuario podrá ingresar texto.
- **Desplegable:** Este dará un campo el cual tendrá respuestas predefinidas, dando la opción de escoger una.

Estos controladores vendrán por la etiqueta <Control>, la cual tendrá 4 elementos obligatorios y dependiendo del tipo del control serán válidos distintos elementos opcionales, los cuales son:

Obligatorios

- **Tipo:** Este definirá el tipo del control, los tipos validos son los descritos anteriormente en controladores.
- **Nombre:** Este definirá el identificador propio del controlador, este identificador es único y no pueden existir otro con el mismo nombre en la misma ventana.
- **X:** Elemento que contendrá la posición en X donde se posicionará.
- **Y:** Elemento que contendrá la posición en Y donde se posicionará.

Opcionales

- **Alto:** Elemento que definirá el alto con que contará dicho contenedor, este valor se tomará en pixeles, en caso no viniera se tomará un alto por defecto de 50px, este elemento es opcional para cualquier tipo de control.
- **Ancho:** Elemento que definirá el alto con que contará dicho contenedor, este valor se tomará en pixeles, en caso no viniera se tomará un ancho por defecto de 50px, este elemento es opcional para cualquier tipo de control.
- **Fuente:** Elemento que definirá el tipo de fuente con que se escribirá el texto, este elemento es opcional para los controles de texto y textoarea.
- **Tam:** Elemento que definirá el tamaño de fuente con que se escribirá el texto, este elemento es opcional para los controles de texto y textoarea.
- **Color:** Elemento que definirá el color con que se escribirá el texto, este valor debe venir en código hexadecimal, en caso este elemento no viniera, se tomará el color negro (#000000), este elemento es opcional para los controles de texto y textoarea.

- **Negrita:** Elemento que definirá si el texto vendrá en Negrilla, este elemento es opcional para los controles de texto y textoarea, este elemento puede tener únicamente 2 valores
    - **Verdadero:** Valor que definirá que si vendrá en negrilla el texto.
    - **Falso:** Valor que definirá que no vendrá en negrilla el texto.
- **Cursiva:** Elemento que definirá si el texto vendrá en Cursiva, este elemento es opcional para los controles de texto y textoarea, este elemento puede tener únicamente 2 valores
    - **Verdadero:** Valor que definirá que si vendrá en cursiva el texto.
    - **Falso:** Valor que definirá que no vendrá en cursiva el texto.
- **Máximo:** Elemento que definirá un número máximo el cual es válido, este elemento es opcional para el control numérico, en caso no viniera el control no tiene valor máximo.
- **Mínimo:** Elemento que definirá un número mínimo el cual es válido, este elemento es opcional para el control numérico, en caso no viniera el control no tiene valor mínimo.
- **Acción:** Esta etiqueta contendrá una llamada a una función, esta función puede contener parámetros, pero únicamente de datos nativos propios de funcionScript, incluyendo cálculos aritméticos, lógicos o relacionales. Este valor ira entre llaves "{ }", para evitar conflictos con los parámetros a llamar, este elemento es opcional para los controles de tipo texto.

Para todos los demás tipos controles los cuales no se menciona como opción en el elemento, serán invalidas si viniera entre sus opciones.

Etiquetas Especiales

Los controles serán capaces de tener etiquetas especiales que definirán aún más sus funciones, estas etiquetas únicamente son válidas dentro de la etiqueta <control> y se validara que según el tipo pueda venir tal etiqueta, estas etiquetas son.

Etiqueta dato

Etiqueta dato, esta etiqueta definirá una de las opciones del control desplegable, puede venir varias veces dentro de la etiqueta <ListaDatos> y todos los datos ingresados deberán darse como opción en el desplegable, esta etiqueta únicamente es válida si el control es de tipo "desplegable".

Esta etiqueta contiene un elemento opcional y el contenido que se escriba entre la etiqueta será el valor del dato que se insertara en el desplegable.

Opcionales

- **Referencia:** Esta etiqueta contendrá un id, al momento de que el control desplegable escoja este dato, la ventana actual se cerrará y abrirá una ventana con el id que contiene dicho elemento.

Etiqueta Lista Datos

Etiqueta Lista Datos, esta etiqueta en su contenido contendrá varias etiquetas dato anteriormente mencionada, únicamente es válida si el control es de tipo “desplegable”, contiene un elemento opcional.

- **Acción:** Esta etiqueta contendrá una llamada a una función, esta función puede contener parámetros, pero únicamente de datos nativos propios de funcionScript, incluyendo cálculos aritméticos, lógicos o relacionales y objetos. Este valor irá entre llaves “{ }”, para evitar conflictos con los parámetros a llamar.

Etiqueta Defecto

Etiqueta defecto, esta etiqueta va a dar un valor por defecto al campo del control que lo contiene, esta etiqueta es válida para todos los tipos de control, se tiene que validar que sea del mismo tipo de dato que contiene el control, si es un control numérico que acepte solo números, si es de tipo texto o textoarea que acepte texto y si está en un desplegable que su dato por defecto se encuentre entre las opciones del desplegable.

Esta etiqueta no contiene ningún elemento y el contenido que se escriba entre la etiqueta será el valor por defecto del control.

Si las etiquetas especiales vienen fuera de un control o que el control no fuera de un tipo valido, se tiene que marcar como error.

Sintaxis control tipo texto

```
<Control tipo = “Numérico” Nombre = “ControlTexto” (Elementos)*>
</Control>

<Control tipo = “Numérico” Nombre = “ControlTexto2” (Elementos)*>
      <Defecto>
            Esta es un área de texto
      </Defecto>
</Control>
```

Sintaxis control tipo textoArea

```
<Control tipo = “TextoArea” Nombre = “ControlTexto” (Elementos)*>
</Control>

<Control tipo = “TextoArea” Nombre = “ControlTexto2” (Elementos)*>
```

```
    <Defecto>
        Esta es un área de texto
        Que acepta textos con
        Múltiples líneas
    </Defecto>
</Control>
```

Sintaxis control tipo numérico

```
<Control tipo = "Numérico" Nombre = "ControlNum" (Elementos)*>
</Control>

<Control tipo = "Numérico" Nombre = "ControlNum2" (Elementos)*>
    <Defecto>
        201504481
    </Defecto>
</Control>
```

Sintaxis control tipo desplegable

```
<Control tipo = "Desplegable" Nombre = "ControlDesp" (Elementos)*>
   <ListaDatos>
        <Dato> Opción 1</Dato>
        <Dato> Opción 2</Dato>
        <Dato> Opción 3</Dato>
        <Dato> Opción n</Dato>
   </ListaDatos>
</Control>

<Control tipo = "Desplegable" Nombre = "ControlNum2" (Elementos)*>
    <ListaDatos>
        <Dato> Opción 1</Dato>
        <Dato> Opción 2</Dato>
        <Dato> Opción 3</Dato>
        <Dato> Opción n</Dato>
   </ListaDatos>
    <Defecto>
        Opción 1
    </Defecto>
</Control>
```

Multimedia

La interfaz generada será capaz de soportar archivos multimedia, para esta opción se permiten únicamente 3 tipos de archivos, los cuales son:

- **Música:** Tipo de archivo de música, la interfaz mostrará un control de música donde el usuario podrá pausar, iniciar y manejar la canción a su gusto.
- **Video:** Tipo de archivo de video, la interfaz mostrará un control de video donde el usuario podrá pausar, iniciar y manejar el video a su gusto.
- **Imagen:** Tipo de archivo con imagen, la interfaz mostrará la imagen en un marco.

Estos elementos multimedia vendrán por la etiqueta <Multimedia> y contendrán 5 elementos obligatorios y 5 opcionales, los cuales son:

Obligatorios

- **Path:** contendrá la dirección relativa del archivo multimedia a reproducir.
- **Tipo:** definirá el tipo de multimedia, los tipos validos son los descritos anteriormente.
- **Nombre:** Este definirá el identificador propio del archivo multimedia, este identificador es único y no pueden existir otro con el mismo nombre en la misma ventana.
- **X:** Elemento que contendrá la posición en X donde se posicionará.
- **Y:** Elemento que contendrá la posición en Y donde se posicionará.

Opcionales

- **Alto:** Elemento que definirá el alto con que contará dicha multimedia, este valor se tomará en pixeles, en caso no viniera se tomará un alto por defecto de 200px, este elemento es opcional para cualquier tipo de multimedia.
- **Ancho:** Elemento que definirá el alto con que contará dicha multimedia, este valor se tomará en pixeles, en caso no viniera se tomará un ancho por defecto de 200px, este elemento es opcional para cualquier tipo de multimedia.
- **Auto-Reproducción:** Elemento que definirá si la multimedia se reproducirá al instante de ser mostrada, este elemento es opcional para cualquier tipo de multimedia
    - **Verdadero:** Valor que definirá que si se reproducirá el archivo multimedia.
    - **Falso:** Valor que definirá que no se reproducirá el archivo multimedia.

Etiqueta Botón

Etiqueta botón, los botones en gxml son los accionadores del lenguaje, estos podrán llamar funciones de los archivos FuncionScript importados, esta etiqueta tendrá 3 elementos obligatorios y 4 opcionales, el contenido dentro del botón, será el texto del botón, este podría ser una etiqueta Texto o bien solo texto escrito por el usuario, los elementos mencionados son:

Obligatorios

- **Nombre:** Este definirá el identificador propio del botón, este identificador es único y no pueden existir otro con el mismo nombre en la misma ventana.
- **X:** Elemento que contendrá la posición en X donde se posicionará.
- **Y:** Elemento que contendrá la posición en Y donde se posicionará.

Opcionales

- **Alto:** Elemento que definirá el alto con que contara el botón, este valor se tomará en pixeles, en caso no viniera se tomará un alto por defecto de 100px, este elemento es opcional para cualquier tipo de multimedia.
- **Ancho:** Elemento que definirá el alto con que contará el botón, este valor se tomará en pixeles, en caso no viniera se tomará un ancho por defecto de 100px, este elemento es opcional para cualquier tipo de multimedia.
- **Referencia:** Esta etiqueta contendrá un id, al momento de que el botón sea presionado, la ventana actual
- **Acción:** Esta etiqueta contendrá una llamada a una función, esta función puede contener parámetros, pero únicamente de datos nativos propios de funcionScript, incluyendo cálculos aritméticos, lógicos o relacionales. Este valor ira entre llaves “{ }”, para evitar conflictos con los parámetros a llamar.

Etiqueta Enviar

Etiqueta especial y única del lenguaje gxml, este tendrá la apariencia de un botón, sin embargo, tendrá la acción específica de recolectar toda la información de los controladores y almacenarla en el archivo gxml definido por la interfaz, esta etiqueta tendrá 3 elementos obligatorios y 4 opcionales, el contenido dentro de la etiqueta, será el texto del botón, este podría ser una etiqueta Texto o bien solo texto escrito por el usuario, los elementos mencionados son:

Obligatorios

- **Nombre:** Este definirá el identificador propio del botón, este identificador es único y no pueden existir otro con el mismo nombre en la misma ventana.
- **X:** Elemento que contendrá la posición en X donde se posicionará.
- **Y:** Elemento que contendrá la posición en Y donde se posicionará

Opcionales

- **Alto:** Elemento que definirá el alto con que contara el botón enviar, este valor se tomará en pixeles, en caso no viniera se tomará un alto por defecto de 100px, este elemento es opcional para cualquier tipo de multimedia.
- **Ancho:** Elemento que definirá el alto con que contará el botón enviar, este valor se tomará en pixeles, en caso no viniera se tomará un ancho por defecto de 100px, este elemento es opcional para cualquier tipo de multimedia.
- **Referencia:** Esta etiqueta contendrá un id, al momento de que el botón enviar sea presionado, la ventana actual se cerrará y abrirá una ventana con el id que contiene dicho elemento.
- **Acción:** Esta etiqueta contendrá una llamada a una función, esta función puede contener parámetros, pero únicamente de datos nativos propios de funcionScript, incluyendo cálculos aritméticos, lógicos o relacionales. Este valor ira entre llaves “{ }”, para evitar conflictos con los parámetros a llamar.

5 FuncionScript

El lenguaje FuncionScript como cualquier lenguaje de programación, tiene su propia estructura, reglas de sintaxis y paradigma de programación, es un derivado del lenguaje de programación javascript, por lo que sus reglas de sintaxis son parecidas a dichos lenguajes.

5.1 Notación dentro del enunciado

Dentro del enunciado, se utilizarán las siguientes notaciones cuando se haga referencia al código de alto nivel.

Sintaxis y ejemplos

Para definir la sintaxis y ejemplos de sentencias de código de alto nivel se utilizará un rectángulo gris.

```
```

Asignación de colores

Dentro del enunciado y en el editor de texto de la aplicación, para el código de alto nivel, seguirá el formato de colores establecido en la Tabla 1. Estos colores también deberán de ser implementados en el editor de texto.

Tabla 1: código de colores.

Color	Token
Azul	Palabras reservadas
Naranja	Cadenas, caracteres
Morado	Números
Gris	Comentario
Negro	Otro

Palabras reservadas

Son palabras que no podrán ser utilizadas como identificadores ya que representan una instrucción específica y necesaria dentro del lenguaje.

Expresiones

Cuando se haga referencia a una ‘expresión’, se hará referencia a cualquier sentencia que devuelve un valor, ya sea una operación aritmética, una operación relacional, una operación lógica, un atributo, variable, parámetro, función, objeto o matriz.

Cerradura positiva

Esta será usada cuando se desee definir que un elemento del lenguaje FuncionScript podrá venir una o más veces: (elemento)+.

Cerradura de Kleene

Esta será usada cuando se desee definir que un elemento del lenguaje FuncionScript podrá venir cero o más veces: (elemento)*.

Opcionalidad

Esta será usada cuando se desee definir que un elemento del lenguaje FuncionScript podrá o no venir (cero o una vez): (elemento)?.

5.2 Características generales

Esta sección describirá las características generales que definen el lenguaje de alto nivel FuncionScript.

5.2.1 Paradigma de programación

El paradigma usado en FuncionScript es **fuctional programing,** este es un paradigma de programación declarativo, es decir, los algoritmos son desarrollados únicamente usando expresiones (lógicas, aritméticas y relaciones), usando funciones de orden superior y recursividad.

FuncionScript es un lenguaje de programación case insensitive y no soportará la sobrecarga de métodos.

5.2.2 Identificadores

Un identificador será utilizado para dar un nombre a variables, métodos o estructuras. Un identificador es una secuencia de caracteres alfabéticos [A-Z a-z] incluyendo el guion bajo [_] o dígitos [0-9] que comienzan con un carácter alfabético o guion bajo.

Ejemplos de identificadores válidos

```
Este_es_un_id_valido_1
_es_un_id_valido_1
Este_un_id_valido_1
```

Ejemplos de identificadores no válidos

```
0id
Is-15
```

5.2.3 Valor nulo

Se reservará la palabra nulo en el lenguaje FuncionScript y está se utilizará para hacer referencia a la nada, indicará la ausencia de valor.

5.2.4 Variables

Las variables serán unidades básicas de almacenamiento en FuncionScript. Una variable se definirá por la combinación de un identificador, un tipo y un inicializador opcional. Además, la variable tiene un entorno que define su usabilidad.

5.2.5 Tipos de datos

Se utilizarán los siguientes tipos de dato:

- **Cadena**: este tipo de dato aceptará cadenas de caracteres, que deberán ser encerradas dentro de comillas dobles o comillas simples.
- **Número**: este tipo de dato, aceptará valores numéricos enteros y decimales.
- **Booleano**: este tipo de dato aceptará valores de verdadero y falso que serán representados con palabras reservadas que serán sus respectivos nombres (“verdadero” y “falso”).
- **Objetos**: serán un conjunto de claves valor.

El tipo de una variable será determinado en tiempo de ejecución. Por lo que FuncionScript será un lenguaje con tipado dinámico.

Tipo de Dato	Ejemplo
Cadenas	“Cadena de ejemplo”
Numero	4.5 o 4
Booleano	Verdadero, Falso
Carácter	‘a’ ‘1’
Objetos	{nombre: “objeto”}

Las variables de tipo de numérico serán capaces de soportar valores en formato **entero como 100** y valores en formato **decimal como 95.9999.**

5.3 Sintaxis

A continuación, se detalla la sintaxis que manejaran todas las sentencias del lenguaje **FuncionScript**.

5.3.1 Bloque de Sentencias

Sera un conjunto de sentencias delimitado por llaves “{ }”, cuando se haga referencia a esto querrá decir que se está definiendo un entorno local con todo y sus posibles instrucciones, es decir las variables declaradas en dicho entorno, únicamente podrán ser utilizadas en dicho entorno o bien en entornos hijos.

Sintaxis

```
{LISTA_SENTENCIAS}
```

```

```

5.3.2 Signos de Agrupación

Para dar orden y jerarquía a ciertas operaciones aritméticas se utilizarán los signos de agrupación. Para los signos de agrupación se utilizarán los paréntesis.

Sintaxis

```
55 * ((432- 212) / 123)
```

5.3.3 Comentarios

Los comentarios permiten añadir notas de texto que son ignorados por el compilador, los comentarios pueden ser de una línea o de múltiples líneas.

Comentarios de una línea

Iniciarán con los símbolos “//” y finalizarán con salto de línea

Sintaxis

```
// Nota de texto \n
```

Ejemplo

```
// Esto es un comentario de una línea
```

Comentarios de múltiples líneas

Iniciarán con los símbolos “/*” y finalizarán con “*/”

Sintaxis

```
/* Nota de texto */
```

Ejemplo

```
/* Esto es un comentario
de múltiples líneas */
```

5.3.4 Operaciones aritméticas

Una operación aritmética es un conjunto de reglas que permiten obtener otras cantidades o expresiones a partir de datos específicos. A continuación, se definen las operaciones aritméticas soportadas por el lenguaje.

Suma

Operación aritmética que consiste en reunir varias cantidades (sumandos) en una sola (la suma). El operador de la suma es el signo más (+).

Especificaciones sobre la suma:

- Al sumar dos datos numérico o carácter el resultado será numérico.
- Al sumar datos de tipo cadena el resultado será la concatenación de ambos datos.
- Al sumar un dato numérico con un dato de tipo carácter el resultado será la suma del código ascii del carácter y el número.
- No será posible sumar datos de tipo booleano con datos numéricos, caracteres o booleanos.
- Será posible concatenar cadenas con booleanos.

En la tabla 3 se presentan ejemplos de la suma.

Tabla 2: sistema de tipos para la suma.

Operando	Tipo resultante	Ejemplos
entero + decimal decimal + entero decimal+ caracter caracter+ decimal decimal + decimal	decimal	5 + 4.5 = 9.5 7.8 + 3 = 10.8 15.3 + 'a' = 112.3 'b'+ 2.7 = 100.7 3.56 + 2.3 = 5.86
Entero + caracter Caracter + Entero Entero+ Entero	entero	7 + 'c' = 106 'C' + 7 = 74 4 + 5 = 9
Entero + cadena cadena + Entero booleano + cadena cadena + booleano caracter + cadena cadena + caracter cadena + cadena	cadena	7 + "hola" = "7hola" "hola" + 7 = "hola7" falso + "." = "falso." "es " + falso = "es falso" '7' + "hola" = "7hola" "hola" + '7' = "hola7" "hola "+"mundo"="hola mundo"

Cualquier otra combinación será inválida y deberá arrojar un error de semántica.

Resta

Operación aritmética que consiste en quitar una cantidad (sustraendo) de otra (minuendo) para averiguar la diferencia entre las dos. El operador de la resta es el signo menos (-).

Especificaciones sobre la resta:

- Al restar dos datos numéricos (`entero, decimal, caracter`) el resultado será numérico.
- No será posible restar datos numéricos con tipos de datos de tipo cadena.
- No será posible restar datos de tipo booleano con datos numéricos, caracteres o booleanos.
- Al restar un dato de tipo caracter el resultado será la resta del código ascii del carácter.

En la tabla 4 se presentan ejemplos de la resta.

Tabla 3: Sistema de tipos para la resta

Operadores	Tipo resultante	Ejemplos
Entero- decimal decimal - Entero decimal - caracter carácter - decimal decimal - decimal	decimal	5 - 4.5 = 0.5 7.8 - 3 = 4.8 15.3 - 'a' = 81.7 'b' - 2.7 = 95.3 3.56 - 2.3 = 1.26
Entero- caracter caracter – Entero Entero - Entero	entero	7 - 'c' = -92 'C' - 7 = 60 4 - 5 = 1

Cualquier otra combinación será inválida y deberá arrojar un error de semántica.

Multiplicación

Operación aritmética que consiste en sumar un número (multiplicando) tantas veces como indica otro número (multiplicador). El operador de la multiplicación es el asterisco (*).

Especificaciones sobre la multiplicación:

- Al multiplicar dos datos de tipo entero, decimal, carácter el resultado será numérico.
- No será posible multiplicar datos numéricos con tipos de datos cadena o booleanos.
- Al multiplicar un dato de tipo carácter el resultado será la multiplicación del código ascii del carácter.

En la tabla 5 se presentan ejemplos de la multiplicación.

Tabla 4: Sistema de tipos para la multiplicación

Operandos	Tipo resultante	Ejemplos
Entero* decimal decimal * Entero decimal * caracter caracter * decimal decimal * decimal	decimal	5 * 4.5 = 22.5 7.8 * 3 = 23.4 15.3 * 'a' = 1484.1 'b' * 2.7 = 264.6 3.56 * 2.3 = 8.188
Entero* caracter caracter * Entero Entero* Entero	entero	7 * 'c' = 693 'C'* 7 = 469 4 * 5= 20

Cualquier otra combinación será inválida y deberá arrojar un error de semántica.

División

Operación aritmética que consiste en partir un todo en varias partes, al todo se le conoce como dividendo, al total de partes se le llama divisor y el resultado recibe el nombre de cociente. El operador de la división es la diagonal (/).

Especificaciones sobre la división:

- Al dividir dos datos de tipo entero, decimal o carácter el resultado será numérico.
- No será posible dividir datos numéricos con tipos de datos cadena o booleanos.
- Al dividir un dato de tipo carácter el resultado será la división del código ascii del carácter.
- Al dividir un dato numérico entre 0 deberá arrojarse un error de ejecución.

En la tabla 6 se presentan ejemplos de la división.

Tabla 5: Sistema de tipos para la división

Operadores	Tipo de dato resultant e	Ejemplos
entero/ decimal decimal / entero decimal / caracter caracter / decimal decimal / decimal Entero/ caracter caracter / entero entero/ Entero	decimal	5 / 4.5 = 1.11111 7.8 / 3 = 2.6 15.3 / 'a' =0.1577 'b' / 2.7 = 28.8889 3.56 / 2.3 = 1.5478 7 / 'c' = 0.7070 'C' / 7 = 9.5714 4 / 5 = 0.8

Cualquier otra combinación será inválida y deberá arrojar un error de semántica.

Potencia

Operación aritmética que consiste en multiplicar varias veces un mismo factor.

Especificaciones sobre la potencia:

- Al potenciar dos datos de tipo entero, decimal, caracter, booleano el resultado será numérico.

En la tabla 7 se presentan ejemplos de la potencia.

Tabla 6: Sistema de tipos para la potencia

Operadores	Tipo de dato resultante	Ejemplos

`entero^ decimal`
`decimal ^ entero`
`decimal ^ caracter`
`caracter ^ decimal`
`booleano ^ decimal`
`decimal ^ booleano`
`decimal ^ decimal`
	`decimal`	`5 ^ 4.5 = 1397.54`
		`7.8 ^ 3 = 474.55`
		`'b' ^ 2.7 = 237853.96`
		`verdadero ^ 1.2 = 1.0`
		`4.5 ^ verdadero = 1.0`
		`3.56 ^ 2.3 = 0.0539`

`entero^ caracter`
`caracter ^ entero`
`booleano ^ entero`
`entero^ booleano`
`entero^ entero`
	`entero`	`'C' ^ 7 = 6060711605323`
		`4 ^ verdadero = 4`
		`4 ^ verdadero = 1`
		`4 ^ 5 = 1024`

Cualquier otra combinación será inválida y deberá arrojar un error de semántica.

Aumento

Operación aritmética que consiste en añadir una unidad a un dato numérico. El aumento será una operación de un solo operando. El aumento sólo podrá venir del lado derecho de un dato. El operador del aumento es el doble signo más (++).

Especificaciones sobre el aumento:

- Al aumentar un tipo de dato numérico (`entero, decimal, caracter`) el resultado será numérico.
- El aumento podrá realizarse sobre números o sobre identificadores de tipo numérico (variables, parámetros o atributos).

Sintaxis

```
variable++;
```

Ejemplo

```
contador++;
```

Decremento

Operación aritmética que consiste en quitar una unidad a un dato numérico.
El decremento será una operación de un solo operando. El decremento sólo podrá venir del lado derecho de un dato. El operador del decremento es el doble signo menos (--).

Especificaciones sobre el decremento:

- Al decrementar un tipo de dato entero, decimal o caracter el resultado será numérico.
- El decremento podrá realizarse sobre números o sobre identificadores de tipo entero, decimal o carácter, ya sean variables, parámetros o atributos.

Sintaxis

```
variable--;
```

Ejemplo

```
contador--;
```

Asignación y operación

Esta funcionalidad realizará la asignación y operación hacia la variable con la que se está operando. Los tipos permitidos serán los descritos en la tabla 8.

Tabla 7: tipos de asignación y operación.

Operador	Ejemplo	Descripción
+=	variable += 10;	Realizará la suma de 10 con la variable y el resultado será asignado a la variable.
*=	Variable *= 20;	Realizará la multiplicación de 20 con la variable y el resultado será asignado a la variable.
-=	Variable -= 15;	Realizará la resta de 15 a la variable y el resultado será asignado a la variable.

/=	Variable /= 35;	Realizará la división de la variable entre 35 y el resultado será asignado a la variable.

Precedencia de operadores

Para saber el orden jerárquico de las operaciones aritméticas se define la siguiente precedencia de operadores. Todas las operaciones aritméticas tendrán asociatividad por la izquierda. La precedencia de los operadores irá de menor a mayor según su aparición en la tabla 9.

Tabla 8: tabla de precedencia de operadores aritméticos.

Nivel	Operador
0	+ -
1	* /
2	^
3	++ --
4	-(Unario)

Los operadores unarios son aquellos que le dan un signo a la expresión, dándole un valor positivo o negativo.

Ejemplos:

```
-5
-(43*3)
-4*3
-variable
```

5.3.5 Operadores relacionales

Una operación relacional es una operación de comparación entre dos valores, siempre devuelve un valor de tipo lógico (`booleano`) según el resultado de la comparación. Una operación relacional es binaria, es decir tiene dos operadores siempre.

Especificaciones sobre las operaciones relacionales:

- Será válido comparar datos numéricos (`entero`, `decimal`) entre sí, la comparación se realizará utilizando el valor numérico con signo de cada dato.

- Será válido comparar cadenas de caracteres entre sí, la comparación se realizará sobre el resultado de sumar el código ascii de cada uno de los caracteres que forman la cadena.
- Será válido comparar datos numéricos con datos de caracter en este caso se hará la comparación del valor numérico con signo del primero con el valor del código ascii del segundo.
- Será válido comparar valores lógicos (`booleano`) entre sí.

En la tabla 10 se muestran los ejemplos de los operadores relacionales.

Tabla 9: operadores relacionales

Operador relacional	Ejemplos de uso
>	5 > 4 = verdadero 4.5 > 6 = falso “abc” > “abc” = falso ‘a’ > “a” = falso 97 > ‘a’ = falso
<	5 < 4 = falso 4.5 < 6 = verdadero “abc” < “abc” = falso ‘a’ < “a” = falso 97 < ‘a’ = falso
>=	5 >= 4 = verdadero 4.5 >= 6 = falso “abc” >= “abc” = verdadero ‘a’ >= “a” = verdadero 97 >= ‘a’ = verdadero
<=	5 <= 4 = falso 4.5 <= 6 = verdadero “abc” <= “abc” = verdadero ‘a’ <= “a” = verdadero 97 <= ‘a’ = verdadero
==	5 == 4 = falso 4.5 == 6 = falso “abc” == “abc” = verdadero ‘a’ == “a” = verdadero 97 == ‘a’ = verdadero
!=	5 != 4 = verdadero 4.5 != 6 = verdadero

	"abc" != "abc" = falso 'a' != "a" = falso 97 != 'a' = falso

5.3.6 Operaciones lógicas

Las operaciones lógicas son expresiones matemáticas cuyo resultado será valor lógico (booleano). Las operaciones lógicas se basan en el álgebra de Boole.

Tabla 10: tabla de verdad para operadores booleanos

OPERADOR A	OPERADOR B	A || B	A && B	!A
Falso	falso	falso	falso	verdadero
Falso	verdadero	verdadero	falso	verdadero
Verdadero	falso	verdadero	falso	falso
Verdadero	verdadero	verdadero	verdadero	falso

A continuación, se presenta la jerarquía de operadores lógicos.

Tabla 11: precedencia y asociatividad de operadores lógicos

NIVEL	OPERADOR	ASOCIATIVIDAD
0	Or "||"	Izquierda
1	And "&&"	Izquierda
2	Not "!"	Derecha

En la tabla 11 se muestran los ejemplos de los operadores lógicos.

Tabla11: operadores lógicos

Operador lógico	Ejemplos de uso
&&	Verdadero && verdadero = verdadero 4 > 4 && verdadero = falso 4 < 3 && verdadero = verdadero Verdadero && falso = falso
||	Verdadero || verdadero = verdadero 4 > 4 || verdadero = verdadero 4 < 3 || verdadero = verdadero

	Verdadero || falso = verdadero Falso || 4*4 < 1 = falso
!	!verdadero = falso !falso = verdadero !(4 > 3) = falso

5.3.7 Declaración de variables

Una variable deberá ser declarada antes de poder ser utilizada. Para la declaración de variables será necesario empezar con la palabra reservada “var” seguida del identificador que esta tendrá. La variable podrá o no tener un valor de inicialización. Es posible declarar dos o más variables a la vez, en tal caso los identificadores estarán separados por comas.

Sintaxis

```
“var” LISTA_ID (= EXPRESION)?
```

Ejemplo

```
var  uno;
var uno, dos, tres;
var uno, dos, tres = 0;
Var cad = “hola”;
```

5.3.8 Arreglos

Se denominará arreglo al conjunto de datos que pueden llegar a tener todos el mismo tipo o bien pueden variar entre sus tipos de datos, los tipos de los arreglos son dinámicos como las variables, esto quiere decir que una posición del arreglo puede tener un tipo en un momento dado y en otro momento puede tener otro tipo de dato.

Arreglos Homogéneos

Un arreglo homogéneo será todo aquel arreglo en el cual todos sus datos son del mismo tipo, ya sea un tipo primitivo (Entero, cadena, etc) o un tipo de objeto. **No existen arreglos de arreglos.**

Arreglos Heterogéneos

Un arreglo heterogéneo será todo aquel arreglo en el cual más de alguno de sus datos son de distinto tipo que con los demás datos, ya sea un tipo primitivo (Entero, cadena, etc) o un tipo de objeto.

Sintaxis

```
Var id = [];
Var id = [(item)+];
```

Ejemplo

```
Var ar = []; //arreglo vacío
Var ar = [1, 2, 3]; //arreglo homogéneo, con tres elementos del mismo tipo
Var ar = [1, verdadero, “cadena”]; //arreglo heterogéneo con tres elementos de tipos diferentes.
```

5.3.9 Acceso a posiciones dentro de un arreglo

Se podrá acceder a posiciones dentro de un arreglo, especificando la posición a la cual se desea acceder, estas posiciones tienen un índice inicial de 0, esto quiere decir que el primer elemento del arreglo se accede con un índice 0 y el último elemento se accede con un índice de un numero menos al tamaño del arreglo.

Sintaxis

```
Identificador [expresión];
```

Ejemplo

```
Var tres = {nombre: “compi2”};
Var arr = [1, “dos”, tres];  // arr[0] = 1, arr[1] = “dos”, arr[2] = tres
Var nombre = arr[2].nombre;
```

5.3.10 Objetos

Los objetos se estructuras de datos que estarán compuestas por conjuntos de claves-valores separados por comas.

```
Var miobjeto = {
    curso: “compi2”,
    Oportunidad:”7”,
    Aprobado: “falso”
};
```

5.3.11 Imprimir

Esta sentencia recibirá un valor de tipo cadena o carácter y lo mostrará en la consola de salida.

Sintaxis

```
imprimir( “cadena” );
```

Ejemplo

```
imprimir(‘a’);
```

5.3.12 Importar

La sentencia importar permitirá hacer referencia a otras clases del mismo lenguaje. Esta sentencia deberá de ser utilizada fuera de la clase. La sentencia recibirá una cadena que contenga la ruta del archivo. La ruta del archivo deberá ser la dirección física del archivo. El formato del archivo deberá ser “*.fs”, se deberá de validar que el formato del archivo sea correcto al momento de importar.

Sintaxis

```
importar (“cadena”);
```

Ejemplo

```
importar (“/ruta/archivo_importado.fs”);
```

5.3.13 Sentencias de transferencia

FuncionScript soportará dos sentencias de salto o transferencia: **detener**, y **retornar**. Estas sentencias transferirán el control a otras partes del programa.

Detener

La sentencia **detener** tendrá dos usos:

- Terminar un bloque de sentencias en una sentencia **selecciona**

Terminar la ejecución de un ciclo.

SINTAXIS

```
detener;
```

EJEMPLO

```
selecciona (y /50) {
```

```
      caso 1: {
      // sentencias que se ejecutarán si y/50 = 1.
      } caso 2: {
        Detener;
      // sentencias que se ejecutarán si y/50 = 2.
      } defecto: {
      // sentencias que se ejecutarán si y/50 no coincide ningún caso.
      }
}
```

NOTAS

- Deberá verificarse que la sentencia detener aparezca únicamente dentro de un ciclo o dentro de una sentencia seleccionar.
- En el caso de sentencias cíclicas, la sentencia **detener** únicamente detendrá la ejecución del bloque de sentencias asociado a la sentencia cíclica que la contiene.
- En el caso de sentencia seleccionar, la sentencia **detener** únicamente detendrá la ejecución del bloque de sentencias asociado a la sentencia case que la contiene.

Retornar

La sentencia **retornar** se empleará para salir de la ejecución de sentencias de un método y, opcionalmente, devolver un valor. Tras la salida del método se vuelve a la secuencia de ejecución del programa al lugar de llamada de dicho método**.**

SINTAXIS

```
retornar [EXPRESION];
```

EJEMPLO

```
retornar 0;
```

NOTAS

- Deberá verificarse que la sentencia **retornar** aparezca únicamente dentro de un método o función.
- Debe verificarse que en funciones la sentencia **retornar**, retorne una expresión del mismo tipo del que fue declarado en dicha función.
- Debe verificarse que la sentencia **retornar**, sin una expresión asociada este contenida únicamente en métodos (funciones de tipo vacío).

5.3.14 Sentencias de selección

Operador ternario

Esta sentencia **retornará** una expresión en función de la veracidad de una condición lógica.

SINTAXIS

```
CONDICION? EXPRESION1: EXPRESION2 ;
```

Si la condición es verdadera entonces el valor resultante será EXPRESION1, de lo contrario si la condición fuera falsa el valor retornado será EXPRESION2.

EJEMPLO

```
var var1 = 1;
var enterito = var1==0? 12: 69;
// Ya que var1 no es igual a 0 entonces cadenita = “var1 es 0”
```

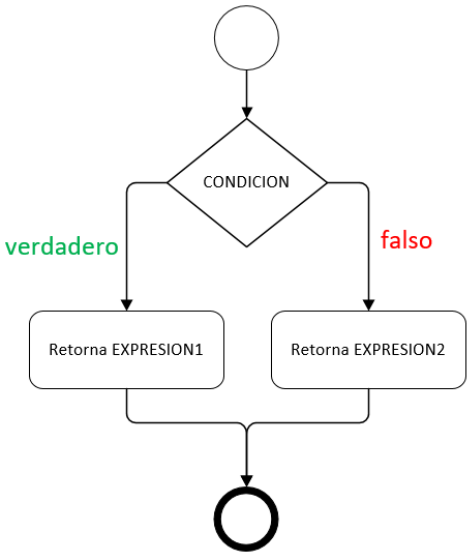


*Ilustración 1: diagrama de flujo para un **operador ternario***

Si

Esta sentencia de selección bifurcará el flujo de ejecución ya que proporciona control sobre dos alternativas basadas en el valor lógico de una expresión (condición).

SINTAXIS

```
si(CONDICION) BLOQUE_SENTENCIAS
(sino si (CONDICION) BLOQUE_SENTENCIAS) *
```

```
[sino BLOQUE_SENTENCIAS]
```

EJEMPLO

```
si (variable > 0) {
      var1 = 0;
} sino si (variable < 0) {
      var1 = 1;
} sino {
      var1 = 2;
}
```

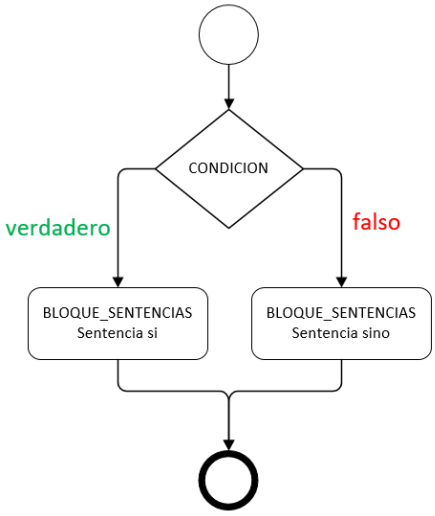


*Ilustración 2: diagrama de flujo para la sentencia de selección **si***

Selecciona

Esta sentencia de selección será la bifurcación múltiple, es decir, proporcionará control sobre múltiples alternativas basadas en el valor de una expresión de control.

La sentencia **selecciona** compara con el valor seguido de cada **caso**, y si coincide, se ejecuta el bloque de sentencias asociadas a dicho **caso**, hasta encontrar una instrucción **detener**.

SINTAXIS

```
selecciona(EXPRESION) {
caso EXPRESION: BLOQUE_SENTENCIAS [detener;]
(caso EXPRESION: BLOQUE_SENTENCIAS [detener;])*
[defecto: BLOQUE_SENTENCIAS [detener;]]
}
```

EJEMPLO

```
selecciona (y /50) {
      caso 1: {
      // sentencias que se ejecutarán si y/50 = 1.
      } caso 2: {
      // sentencias que se ejecutarán si y/50 = 2.
      } defecto: {
      // sentencias que se ejecutarán si y/50 no coincide ningún caso.
      }
}
```

NOTAS

- Si i es el caso actual y n el número de casos, si la expresión del caso i llegara a coincidir con la expresión de control (para $i < n$), se ejecutarán todos los bloques de sentencias entre los casos i y n mientras no se encuentre la instrucción **detener**.
- Tanto la expresión de control como las expresiones asociadas a cada uno de los casos deberá ser de tipo primitivo, es decir, `carácter, entero, decimal` o `booleano`.
- Si ninguno de los casos coincide con la expresión de control se ejecutará el bloque de sentencias asociadas a **defecto.**

5.3.15 Funciones y procedimientos

Las funciones serán conjuntos de sentencias que podrán ser llamados desde otras funciones o procedimientos. Este conjunto de instrucciones deberá de retornar un valor del tipo de dato especificado. A excepción del tipo “vacío” que este tipo no retornara ningún valor. Para retornar el valor se utilizará la palabra reservada “retornar” seguida del valor.

SINTAXIS

```
funcion nombre_procedimiento(parámetros){
  //varias sentencias
}
```

EJEMPLO

```
funcion suma(var operador1, entero operador2) {
   retornar operador1 + operador2;
}

Var suma = (x,y)=>x+y;
```

5.3.16 Llamada a procedimientos y funciones

Para llamar un procedimiento o función en el lenguaje FuncionScript no será necesario que se defina el procedimiento o función previamente, y se realizará de la siguiente forma.

Sintaxis

```
nombre_método( lista_parametros );
```

Ejemplo

```
Procedimiento1();
Variable = Funcion1();
```

5.4 Funciones con arreglos propias de FuncionScript

5.4.1 Ordenamiento descendente

Esta función propia de los arrays tendrá dos modalidades:

- Alfabético, para cadenas.
- Numérico para datos numéricos.

SINTAXIS

```
Arreglo.Descendente();
```

EJEMPLO

```
var frutas = ["Banana", "Orange", "Apple", "Mango"];
frutas.descendente();       // ordena frutas, Apple, Banana, Mango, Orange

var cantidades = [10,213,45,1,234];
cantidades.descendente();       // ordena cantidades, 1, 10, 45, 213, 234
```

5.4.2 Ordenamiento ascendente

Esta función propia de los arrays tendrá dos modalidades:

- Alfabético, para cadenas.
- Numérico para datos numéricos.

SINTAXIS

```
Arreglo.ascendente();
```

EJEMPLO

```
var frutas = ["Banana", "Orange", "Apple", "Mango"];
frutas.ascendente();        // ordena frutas, Orange, Mango, Banana,  Apple

var cantidades = [10,213,45,1,234];
cantidades.ascendente();        // ordena cantidades, 234, 213, 45,10, 1
```

5.4.3 Crear array desde archivo

Esta función será utilizada para crear un vector de objetos a partir de un archivo gxml en donde se encontrarán los datos recolectados por las interfaces de usuario.

SINTAXIS

```
var data = crearArrayDesdeArchivo('../../principalData.gxml');
```

5.4.4 Invertir

Esta función propia de los arrays acomodara el arreglo de forma inversa a su estado actual, esto quiere decir que el primer elemento pasa a ser el último, el segundo como penúltimo y así respectivamente.

SINTAXIS

```
Arreglo.invertir();
```

EJEMPLO

```
var frutas = ["Banana", "Orange", "Apple", "Mango"];
frutas.invertir();        // ordena frutas, Mango, Apple, Orange, Banana
```

5.4.5 Máximo

Esta función propia de los arrays retornara el valor máximo del arreglo.

SINTAXIS

```
Arreglo.maximo();
```

EJEMPLO

```
var cantidades = [10,213,45,1,234];
var numMax = cantidades.maximo();        // retorna 234
```

5.4.6 Mínimo

Esta función propia de los arrays retornara el valor mínimo del arreglo.

SINTAXIS

```
Arreglo.minimo();
```

EJEMPLO

```
var cantidades = [10,213,-45,1,234];
var numMin = cantidades.minimo();        // retorna -45
```

5.4.7 Función filtrar

El método filtrar() crea una matriz con todos los elementos de matriz que aprobar un examen (siempre en función).

Nota: filtrar() no se ejecuta la función para los elementos de la matriz sin valores.

Nota: filtrar() no cambia la matriz original.

EJEMPLO

Devuelve un array de todos los valores de la matriz de las edades que tienen 18 años o más

```
var ages = [32, 33, 16, 40];

function checkAdult(age) {
  retornar age >= 18;
}

ar1= ages.filter(checkAdult);

ar1.map((item) => {
    imprimir("mayores de edad:" + item);
});

/*
valor: 32
valor: 33
valor: 40
*/
```

5.4.8 Función buscar

El método buscar() devuelve el valor del primer elemento en una matriz que aprobar un examen (siempre en función).

El método buscar() ejecuta la función una vez para cada elemento de la matriz:

Si se encuentra un elemento de la matriz donde la función devuelve un valor verdadero, buscar() devuelve el valor de ese elemento de la matriz (y no comprueba los valores restantes)

De lo contrario devuelve undefined

Nota: buscar() no se ejecuta la función para las matrices vacías.

Nota: buscar() no cambia la matriz original.

EJEMPLO

Obtener el valor del primer elemento de la matriz que tiene un valor de 18 o más:

```
var ages = [3, 10, 18, 20];

funcion checkAdult(age) {
  retornar age >= 18;
}

funcion myFuncion() {
  var buscarAge = ages.buscar(checkAdult);
  imprimir(“El primer element adulto es: ”+buscarAge);
}
```

5.4.9 Función map

El método de map() crea una nueva matriz con los resultados de llamar a una función para cada elemento de la matriz.

El método de map() llama a la función proporcionada una vez para cada elemento de una matriz en orden.

EJEMPLO

Se le suma 2 a cada elemento de ar y luego se imprime:

```
var ar = [1, 2, 3];

funcion sumar(item){
    retornar item + 2;
}
var ar1 = ar.map(sumar);

ar1.map((item) => {
    imprimir("valor:" + item);
});

/*
valor: 3
valor: 4
valor: 5
*/
```

5.4.10 Función reduce

El método reduce() la matriz a un solo valor.

El método reduce() ejecuta una función proporcionada para cada valor de la matriz (de izquierda a derecha).

El valor devuelto de la función se almacena en un acumulador (resultado/total).

Nota: reducir () no se ejecuta la función para los elementos de la matriz sin valores.

EJEMPLO

Obtener la suma de los números en la matriz:

```
var numbers = [65, 44, 12, 4];

funcion getSum(total, num) {
  retornar total + num;
}

funcion myFuncion() {
  x = numbers.reduce(getSum);
}
```

5.4.11 Función todos

El método todos() comprueba si todos los elementos de una matriz aprobar un examen (siempre en función).

El método todos() ejecuta la función una vez para cada elemento de la matriz:

Si se encuentra un elemento de la matriz donde la función devuelve un valor falso, todos() devuelve falso (y no comprueba los valores restantes)

Producirse ningún falso, todos() devuelve verdadero.

Nota: todos() no se ejecuta la función para los elementos de la matriz sin valores.

Nota: todos() no cambia la matriz original

EJEMPLO

Compruebe si todos los valores de la matriz de edad están de 18 años o más:

```
var ages = [32, 33, 16, 40];

funcion checkAdult(age) {
  retornar age >= 18;
```

```
}

funcion myFuncion() {
          si (ages.todos(checkAdult)){
                imprimir(“todos tienen mas de 18”);
        }
}
```

5.4.12 Función alguno

El método alguno() comprueba si alguno de los elementos de una matriz aprobar un examen (siempre en función).

El método alguno() ejecuta la función una vez para cada elemento de la matriz:

Si se encuentra un elemento de la matriz donde la función devuelve un valor verdadero, alguno() devuelve verdadero (y no comprueba los valores restantes)

De lo contrario devuelve falso

Nota: alguno() no se ejecuta la función para los elementos de la matriz sin valores.

Nota: alguno() no cambia la matriz original.

EJEMPLO

Compruebe si los valores de la matriz de edad están de 18 años o más:

```
var ages = [3, 10, 18, 20];

funcion checkAdult(age) {
  retornar age >= 18;
}

funcion myFuncion() {
  x = ages.alguno(checkAdult);
}
```

5.5 Funciones Nativas de Interfaz

Las funciones nativas son aquellas predefinidas para el lenguaje, FuncionScript contiene una serie de funciones nativas que ayudan a la aplicación a crear la Interfaz de Usuario deseada, estas funciones son:

5.5.1 LeerGxml

Esta es la función principal para la creación de la Interfaz, esta función lee el archivo de lenguaje gxml, recibe como parámetro la dirección relativa o completa del archivo, en caso no existiera el archivo nos retornará un nulo y nos retorna todos los elementos del lenguaje en forma de arreglo, con ello podremos ingresar y crear los elementos deseados.

Sintaxis

```
Var archivo = LeerGxml("direccion");
```

Ejemplo

```
Var archivo = LeerGxml("Interfaz3.gxml") // retorna un arreglo los elementos Interfaz3
```

5.5.2 ObtenerPorEtiqueta

Esta función nativa puede usarse unicamente en un arreglo que contenga elementos Gxml, recibe de parámetro el tipo de etiqueta deseado y nos retorna en forma de arreglo todos los elementos que cumplan con el tipo de etiqueta que recibe de parámetro, una vez obtenidos se podrá acceder a cada elemento de los objetos gxml.

Sintaxis

```
Var ventanas = archivo.ObtenerPorEtiqueta("etiqueta");
```

Ejemplo

```
Var ventanas = archivo.ObtenerPorEtiqueta("ventana"); // retorna todas las ventanas de archivo
```

5.5.3 ObtenerPorId

Esta función nativa puede usarse unicamente en un arreglo que contenga elementos Gxml, recibe de parámetro el id del nodo deseado y nos retorna en forma de objeto un único elemento que tiene el id del parámetro, en caso no exista el id nos retornará un nulo, una vez obtenido se podrá acceder a cada elemento de los objetos gxml.

Sintaxis

```
Var nodo = archivo.ObtenerPorId("id");
```

Ejemplo

```
Var nodo = archivo.ObtenerPorNombre(“contenedor1”); // retorna el nodo del objeto gxml que tenga de id contenedor1
```

5.5.4 ObtenerPorNombre

Esta función nativa puede usarse unicamente en un arreglo que contenga elementos Gxml, recibe de parámetro el nombre del nodo deseado y nos retorna en forma de objeto un único elemento que tiene el nombre del parámetro, en caso no exista el id nos retornará un nulo, una vez obtenido se podrá acceder a cada elemento de los objetos gxml.

Sintaxis

```
Var nodo = archivo.ObtenerPorNombre(“nombre”);
```

Ejemplo

```
Var nodo = archivo.ObtenerPorNombre(“contenedor1”); // retorna el nodo del objeto gxml que tenga de nombre contenedor1
```

5.5.5 Crear Ventana

Esta función Nativa, como su nombre nos dice, nos retornará una nueva ventana lista para crearle todos los elementos de la interfaz en su interior, todos sus parámetros podrán ser obtenidos por medio de la variable ventana que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta ventana recibe 1 parámetro, el cual es:

- **Color**: Parámetro de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.

Sintaxis

```
Var ventana1 = CrearVentana(“Color hexadecimal”)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);
Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);
```

5.5.6 CrearContenedor

Esta función Nativa, como su nombre nos dice, nos retornará un nuevo contenedor listo para crearle todos los elementos de la interfaz en su interior, unicamente podrá ser llamada sobre una variable ventana, todos sus parámetros podrán ser obtenidos por medio de la variable contenedor que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, este contenedor recibe 6 parámetros, los cuales son:

- **Alto**: Parámetro de tipo entero que definirá el alto del contenedor.
- **Ancho**: Parámetro de tipo entero que definirá el ancho del contenedor.
- **Color**: Parámetro de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.
- **Borde**: Parámetro Booleano que definirá si tiene o no borde.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el contenedor.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el contenedor.

Sintaxis

```
Var cont1 = CrearContenedor(Alto, Ancho, Color, Borde, X, Y)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);
```

5.5.7 Crear Texto

Esta función Nativa, como su nombre nos dice, nos creará un nuevo texto en la interfaz, esta función no nos retorna nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable texto que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 7 parámetros, los cuales son:

- **Fuente**: Parámetro de tipo cadena que definirá que en su texto tendrá el nombre de la fuente con que se desea crear el texto.
- **Tamaño**: Parámetro de tipo entero que definirá el tamaño de fuente con que se creará el texto.
- **Color**: Parámetro de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.
- **Negrita**: Parámetro Booleano que definirá si se creará con formato de negrilla o no.
- **Cursiva**: Parámetro Booleano que definirá si se creará con formato de cursiva o no
- **Referencia**: Parámetro de tipo cadena que contendrá el nombre de la ventana a la cual partirá.

Sintaxis

```
Contenedor.CrearTexto(Fuente, Tamaño, Color, X, Y, Negrilla, Cursiva, Referencia)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);
```

```
Var elementos = archivo.ObtenerPorEtiqueta(“texto”);

Var creacion1 = elementos[0];

Cont1.CrearTexto(creacion1.fuente, creacion1.tam, creacion1.color, creacion1.x, creacion1.y, creacion1.negrita, creacion1.cursiva, creacion.referencia);
```

5.5.8 Crear Caja texto

Esta función Nativa, como su nombre nos dice, nos creará una nueva caja de texto en la interfaz, esta función no nos retorna nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable controlador de tipo texto que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 9 parámetros, los cuales son:

- **Alto**: Parámetro de tipo entero que definirá el alto de la caja texto.
- **Ancho**: Parámetro de tipo entero que definirá el ancho de la caja texto.
- **Fuente**: Parámetro de tipo cadena que definirá que en su texto tendrá el nombre de la fuente con que se desea crear el texto.
- **Tamaño**: Parámetro de tipo entero que definirá el tamaño de fuente con que se creará el texto.
- **Color**: Parámetro de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.
- **Negrita**: Parámetro Booleano que definirá si se creará con formato de negrilla o no.
- **Cursiva**: Parámetro Booleano que definirá si se creará con formato de cursiva o no.

Sintaxis

```
Contenedor.CrearCajaTexto(Alto, Ancho, Fuente, Tamaño, Color, X, Y, Negrilla, Cursiva)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);
```

```
Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);

Var creacion1 = archivo.ObtenerPorId(“caja1”);

Cont1.CrearCajaTexto(creacion1.alto,creacion1,ancho, creacion1.fuente, creacion1.tam, creacion1.color, creacion1.x, creacion1.y, creacion1.negrita, creacion1.cursiva);
```

5.5.9 Crear Área Texto

Esta función Nativa, como su nombre nos dice, nos creará una nueva área de texto en la interfaz, esta función no nos retorna nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable controlador de tipo TextoArea que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 9 parámetros, los cuales son:

- **Alto**: Parámetro de tipo entero que definirá el alto del área de texto.
- **Ancho**: Parámetro de tipo entero que definirá el ancho del área de texto.
- **Fuente**: Parámetro de tipo cadena que definirá que en su texto tendrá el nombre de la fuente con que se desea crear el texto.
- **Tamaño**: Parámetro de tipo entero que definirá el tamaño de fuente con que se creará el texto.
- **Color**: Parámetro de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.
- **Negrita**: Parámetro Booleano que definirá si se creará con formato de negrilla o no.
- **Cursiva**: Parámetro Booleano que definirá si se creará con formato de cursiva o no

Sintaxis

```
Contenedor.CrearAreaTexto(Alto, ancho, Fuente, Tamaño, Color, X, Y, Negrilla, Cursiva)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId("id");

Var ventanas = archivo.ObtenerPorEtiqueta("ventana");

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta("contenedor");

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);

Var creacion1 = archivo.ObtenerPorId("area1");

Cont1.CrearAreaTexto(creacion1.alto,creacion1,ancho, creacion1.fuente, creacion1.tam, creacion1.color, creacion1.x, creacion1.y, creacion1.negrita, creacion1.cursiva);
```

5.5.10 Crear Control Numérico

Esta función Nativa, como su nombre nos dice, nos creará un nuevo control numérico en la interfaz, esta función no nos retorna nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable controlador de tipo numérico que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 6 parámetros, los cuales son:

- **Alto**: Parámetro de tipo entero que definirá el alto del control numérico.
- **Ancho**: Parámetro de tipo entero que definirá el ancho del control numérico.
- **Máximo**: Parámetro de tipo entero que definirá el número máximo del control numérico.
- **Mínimo**: Parámetro de tipo entero que definirá el número mínimo del control numérico.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el control numérico.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el control numérico.

Sintaxis

```
Contenedor.CrearControlNumerico(Alto, Ancho, Maximo, Minimo, X, Y)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);

Var creacion1 = archivo.ObtenerPorId(“numero1”);

Cont1. CrearControlNumerico (creacion1.alto, creacion1,ancho, creacion1.maximo, creacion1.minimo, creacion1.x, creacion1.y);
```

5.5.11 Crear Desplegable

Esta función Nativa, como su nombre nos dice, nos creará un nuevo control desplegable en la interfaz, esta función no nos retorna nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable controlador de tipo desplegable que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 6 parámetros, los cuales son:

- **Alto**: Parámetro de tipo entero que definirá el alto del control desplegable.
- **Ancho**: Parámetro de tipo entero que definirá el ancho del control desplegable.
- **ListaDatos**: Parámetro que será un arreglo de tipo dato, esta será la lista que mostrará el control desplegable.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el control desplegable.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el control desplegable.
- **Defecto:** Parámetro que marcará el valor por defecto del desplegable.

Sintaxis

Sintaxis

Contenedor.CrearDesplegable(Alto, Ancho, lista, X, Y, Defecto)

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);

Var creacion1 = archivo.ObtenerPorId(“desplegable1”);

Cont1. CrearDesplegable (creacion1.alto, creacion1,ancho, creacion1.lista, creacion1.x, creacion1.y, creacion1.lista[0]);
```

El control desplegable de gxml tiene la propiedad especial ante los demás controles ya que tiene una lista de datos, esta lista de datos se puede obtener por medio del atributo listadatos, este atributo nos retornará un arreglo de datos a los cuales se les puede personalizar los atributos, los atributos de cada dato son:

- **Fuente**: Atributo de tipo cadena que definirá que en su texto tendrá el nombre de la fuente con que se desea crear el texto.
- **Tamaño**: Atributo de tipo entero que definirá el tamaño de fuente con que se creará el texto.
- **Color**: Atributo de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.
- **InicioX**: Atributo de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Atributo de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.

- **Negrita**: Atributo Booleano que definirá si se creará con formato de negrilla o no.
- **Cursiva**: Atributo Booleano que definirá si se creará con formato de cursiva o no
- **Referencia**: Atributo de tipo cadena que contendrá el nombre de la ventana a la cual partirá.

Sintaxis

```
Desplegable.lista[0].atributo = ##;
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color, verdadero, cont1.x, cont1.y);

Var creacion1 = archivo.ObtenerPorId(“desplegable1”);

Creacion1.Fuente = “Arial”;
Creacion1.Tam =12;
```

5.5.12 Crear Botón

Esta función Nativa, como su nombre nos dice, nos creará un nuevo botón en la interfaz, esta función nos retorna el objeto botón y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable texto que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 7 parámetros, los cuales son:

- **Fuente**: Parámetro de tipo cadena que definirá que en su texto tendrá el nombre de la fuente con que se desea crear el texto.
- **Tamaño**: Parámetro de tipo entero que definirá el tamaño de fuente con que se creará el texto.
- **Color**: Parámetro de tipo cadena, el cual en su texto tendrá el código hexadecimal del color deseado.

- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el botón.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el botón.
- **Referencia**: Parámetro de tipo cadena que contendrá el nombre de la ventana a la cual partirá.

Para los botones que contienen acciones, se le da el valor al evento AlClic para ejecutar dicha acción.

Sintaxis

```
Var bot = Contenedor.CrearBoton(Fuente, Tamaño, Color, X, Y,Referencia)
Bot.AlClic(Metodo(12));
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId("id");

Var ventanas = archivo.ObtenerPorEtiqueta("ventana");

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta("contenedor");

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color,
      verdadero, cont1.x, cont1.y);

Var elementos = archivo.ObtenerPorEtiqueta("texto");

Var creacion1 = elementos[0];

Cont1.CrearBoton(creacion1.fuente, creacion1.tam, creacion1.color, creacion1.x,
      creacion1.y, creacion.referencia);
```

5.5.13 Crear Imagen

Esta función Nativa, como su nombre nos dice, nos creará una nueva imagen en la interfaz, esta función no nos retornará nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable multimedia que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 3 parámetros, los cuales son:

- **Ruta:** Parámetro de tipo cadena que contendrá la dirección relativa de la imagen que se desea.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.

Sintaxis

```
Contenedor.CrearImagen(Ruta, X, Y)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color,
      verdadero, cont1.x, cont1.y);

Var elementos = archivo.ObtenerPorEtiqueta(“multimedia”);

Var creacion1 = elementos[0];

Cont1.CrearImagen(creacion1.path, creacion1.x, creacion1.y);
```

5.5.14 Crear Reproductor

Esta función Nativa, como su nombre nos dice, nos creará un nuevo reproductor de música en la interfaz, esta función no nos retornará nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable multimedia que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 3 parámetros, los cuales son:

- **Ruta:** Parámetro de tipo cadena que contendrá la dirección relativa de la imagen que se desea.
- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.

Sintaxis

```
Contenedor.CrearReproductor(Ruta, X, Y)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color,
 verdadero, cont1.x, cont1.y);

Var elementos = archivo.ObtenerPorEtiqueta(“multimedia”);

Var creacion1 = elementos[0];

Cont1.CrearReproductor(creacion1.path, creacion1.x, creacion1.y);
```

5.5.15 Crear Video

Esta función Nativa, como su nombre nos dice, nos creará un nuevo reproductor de video en la interfaz, esta función no nos retornará nada y podrá únicamente podrá ser llamada sobre una variable contenedor, todos sus parámetros podrán ser obtenidos por medio de la variable multimedia que deseamos crear, la variable la podemos obtener por medio de las funciones anteriormente descritas, esta función recibe 3 parámetros, los cuales son:

- **Ruta:** Parámetro de tipo cadena que contendrá la dirección relativa de la imagen que se desea.

- **InicioX**: Parámetro de tipo entero que definirá la posición en X de la ventana donde iniciará el texto.
- **InicioY**: Parámetro de tipo entero que definirá la posición en Y de la ventana donde iniciará el texto.

Sintaxis

```
Contenedor.CrearVideo(Ruta, X, Y)
```

Ejemplo

```
Var nodo = archivo.ObtenerPorId(“id”);

Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Var contenedores = archivo.ObtenerPorEtiqueta(“contenedor”);

Var contenedor1 = contenedores[0];

Var cont1 = ventana1.CrearContenedor(cont1.alto, cont1.ancho, cont1.color,
    verdadero, cont1.x, cont1.y);

Var elementos = archivo.ObtenerPorEtiqueta(“multimedia”);

Var creacion1 = elementos[0];

Cont1.CrearVideo(creacion1.path, creacion1.x, creacion1.y);
```

5.6 Eventos

Los eventos son accionadores propios de FuncionScript los cuales se activan al momento en que en la interfaz sucede cierta acción, estos son:

5.6.1 Al Clic

Evento que se dispara al momento de hacer clic en algún elemento de la interfaz, esta propiedad está disponible para los botones, el botón enviar y las etiquetas Texto y reciben de parámetro una llamada a método.

Sintaxis

```
boton.AlClic(Metodo(12))
```

Ejemplo

```
Var elementos = archivo.ObtenerPorEtiqueta("texto");

Var creacion1 = elementos[0];

Creacion1. AlClic(Metodo(12))
```

5.6.2 Al Cargar

Evento que se dispara al momento de terminar de cargar la ventana actual, esta propiedad está disponible únicamente para las ventanas y reciben de parámetro una llamada a método.

Sintaxis

```
ventana.AlCargar(Metodo(12))
```

Ejemplo

```
Var ventanas = archivo.ObtenerPorEtiqueta("ventana");

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Ventana1.AlCargar(Metodo(12));
```

5.6.3 Al Cerrar

Evento que se dispara al momento de cerrar la ventana actual, esta propiedad está disponible únicamente para las ventanas y reciben de parámetro una llamada a método.

Sintaxis

```
ventana.AlCerrar(Metodo(12))
```

Ejemplo

```
Var ventanas = archivo.ObtenerPorEtiqueta(“ventana”);

Var ven1 = ventanas[0];

Var ventana1 = CrearVentana(ven1.color);

Ventana1.AlCerrar(Metodo(12));
```

6 Manejo de errores

La herramienta deberá de ser capaz de identificar todo tipo de errores (léxicos, sintácticos y semánticos) al momento de interpretar el lenguaje de entrada (FuncionScript, Gxml). Estos errores se almacenarán y se mostrarán en la aplicación. Este reporte deberá contener el día y hora de ejecución, seguido de una tabla con la descripción detallada de cada uno de los errores.

Los tipos de errores se deberán de manejar son los siguientes:

- Errores léxicos.
- Errores sintácticos.
- Errores semánticos

Unos errores léxico es aquel en el que se encuentra un carácter que no está permitido como parte de un lexema.

Ejemplo 1

```
V$ar nombre = “mike”;
```

En la cadena anterior se introdujo arbitrariamente el componente léxico “$”, esto deberá de provocar un error léxico. La forma en que se recuperará de un error léxico es con la estrategia del modo de pánico, por lo que se ignorarán los siguientes caracteres hasta encontrar un token bien formado. En el ejemplo anterior se ignorarán los siguientes caracteres “**V$ar**” hasta terminar de formar el token “numero”.

Los errores sintácticos son aquellos que darán como resultado de ingresar una cadena de entrada que no corresponde con la sintaxis definida para el lenguaje.

Ejemplo 2

```
Nombre var = “mike”
```

En la entrada del ejemplo anterior se invirtió el orden de los tokens que se usan para formar una declaración de variable (ver ejemplo 3). Al encontrarse con un error sintáctico la estrategia que se tomará será la de terminar el proceso de análisis y reportar el error en ese mismo instante.

Ejemplo 3

```
Var numero = 10;
```

En el ejemplo anterior se muestra el formato correcto en el que se declarará una variable.

Un error semántico se dará por ejemplo al intentar usar una variable que no ha sido declarada.

```
...
entero numero1 = numero2 + 10;
…
```

Como se ve en ejemplo 4 la variable "numero2" no ha sido declarada por lo que se mostrará un error semántico, la forma en que se manejaran los errores semánticos consistirá en parar el análisis y mostrar un mensaje de error.

Consideraciones

- Si se existiesen varios errores léxicos se podrá continuar con el análisis de la cadena de entrada y los errores detectados se mostrarán al final del análisis.
- Si se encontrara algún un error sintáctico en la cadena de entrada se deberá de reportar en ese instante y pararse el proceso de análisis.
- Si se hallará un error de tipo semántico se procederá a parar el proceso de análisis y se reportará el error de inmediato.

La tabla de errores debe contener como mínimo la siguiente información:

- Línea: Número de línea donde se encuentra el error.
- Columna: Número de columna donde se encuentra el error.
- Tipo de error: Identifica el tipo de error encontrado. Este puede ser léxico, sintáctico o semántico.
- Descripción del error: Dar una explicación concisa de por qué se generó el error.

Ejemplo

Tipo	Descripción	Fila	Columna
Léxico	El carácter “°” no pertenece al alfabeto del lenguaje	**3**	**1**
Sintáctico	Se esperaba ID y se encontró “/”	**74**	**15**
Semántico	La variable “nombre” no ha sido declarada	**89**	**12**

7 Entregables y Restricciones

7.1 Entregables

1. Código fuente
2. Aplicación funcional
3. Gramáticas
4. Manual técnico
5. Manual de usuario

Deberán entregarse todos los archivos necesarios para la ejecución de la aplicación, así como el código fuente, la gramática y la documentación. La calificación del proyecto se realizará con los archivos entregados en la fecha establecida. El usuario es completa y únicamente responsable de verificar el contenido de los entregables. La calificación se realizará sobre archivos ejecutables. Se proporcionarán archivos de entrada al momento de calificación.

7.2 Restricciones

1. La aplicación deberá ser desarrollada utilizando el lenguaje java con IDE libre.
2. Para la generación de analizadores se utilizarán las herramientas Jlex y Cup.
3. Copias de proyectos tendrán de manera automática una nota de 0 puntos y serán reportados a la Escuela de Ingeniería en Ciencias y Sistemas los involucrados.
4. Todos los lenguajes implementados no hacen distinción entre mayúsculas y minúsculas, es decir no son case sensitive.

7.3 Requisitos mínimos

Los requerimientos mínimos del proyecto son funcionalidades del sistema que permitirán un ciclo de ejecución básica, para tener derecho a calificación se deben cumplir con lo siguiente:

- Extreme Editor (Todos los componentes del IDE)
- GenericXML
    - Etiqueta Texto
    - Etiqueta dato
    - Etiqueta Lista de datos
    - Etiqueta enviar
    - Etiqueta Botón
    - Etiqueta Ventana
- FuncionScript
    - Declaración de variables
    - Asignación de variables
    - Operaciones aritméticas
    - Operaciones relacionales
    - Operaciones lógicas
    - Arreglos

- Imprimir
- Sentencias de selección
- Sentencias de transferencia
- Llamada a procedimientos y funciones
- Funciones con arreglos propias de funcionScript
    - Ordenamiento descendente
    - Ordenamiento ascendente
    - Máximo
    - Mínimo
    - Función filtrar
    - Función buscar
    - Función map
    - Función reduce
    - Función todos
    - Función algunos
- Funciones nativas de la interfaz
    - LeerGxml
    - Obtenerporetiqueta
    - ObtenerporID
    - ObtenerPorNombre
    - CrearTexto
    - CrearVentana
    - CrearContenedor
    - CrearCajaTexto
    - CrearAreaTexto
- Eventos
    - Al Clic

7.4 Entrega del proyecto

1. La entrega será virtual y por medio de la plataforma Classroom.
2. La entrega de cada uno de los proyectos es individual.
3. Para la entrega del proyecto se deberá cumplir con todos los requerimientos mínimos.
4. No se recibirán proyectos después de la fecha ni hora de la entrega estipulada.
5. La entrega del proyecto será mediante un archivo comprimido de extensión rar o zip.
6. Entrega del proyecto:

Miércoles 20 de marzo de 2019 a las 22:00 horas