Buffer Object Oriented

|  |
| --- |
| **Título Proyecto** |

|  |  |
| --- | --- |
| **Solicitante:** | **Tekniker** |
| **Código Doc. / Versión:** | / 1.0 |
| **Elaborado por:** | A. Izpizua |
| **Aprobado por:** |  |
| **Fecha:** | 27/02/2018 |

| INDICE |

[1. Introducción 2](#_Toc507496031)

[2. Descripción de las clases 2](#_Toc507496032)

[2.1 Clase genérica 2](#_Toc507496033)

[2.2 Clases hijas 4](#_Toc507496034)

[2.2.1 TimeDataBuffer 4](#_Toc507496035)

[2.2.2 WaveformBuffer 5](#_Toc507496036)

[3. Como realizar una clase nueva 5](#_Toc507496037)

| HISTORIA DEL DOCUMENTO |

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha** | **Autor** | **Comentarios** |
| **1.0** | 27/02/2018 | A. Izpizua | *Primera versión del tool de waveform* |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introducción

En este documento se recoge la estructura de las clases utilizadas para crear las funciones del buffer equivalente al que ya existía, pero con ciertas mejoras a tener en cuenta

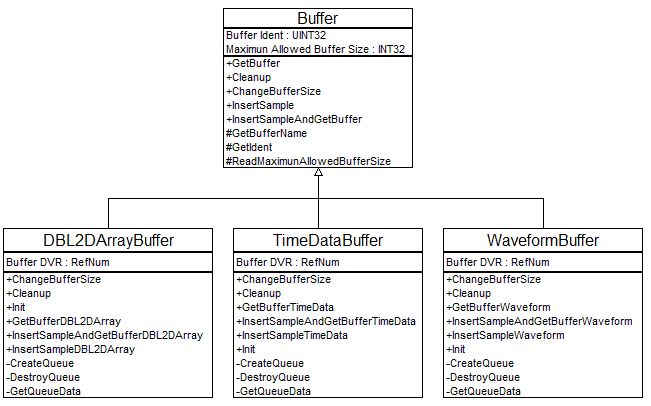
* Los datos se pueden insertar en el buffer en un bucle diferente al que se recogen
* Se puede modificar el tamaño máximo del buffer al vuelo. Esta operación tienes sus implicaciones y no debe realizarse continuamente ya que se está realizando una reserva de memoria cada vez y si la nueva memoria no cabe en el hueco anterior se queda un trozo de memoria que el gestor de memoria no va a utilizar.

Para esta versión inicial se han codificado buffer para 2 tipos de datos.

También se han incluido los test para los métodos públicos de los dos tipos de datos. Se usa VI tester de JKI para realizar los test.

# Descripción de las clases

Para cada tipo de datos se crea una clase hija de una clase genérica.



## Clase genérica

Se utiliza esta clase como base y es la que se va a llamar. Los métodos se pueden llamar desde diferentes bucles. Una vez inicializado el objeto, este hace referencia siempre al mismo buffer. Diferentes objetos son diferentes buffers

**Propiedades**

Esta clase tiene dos propiedades

* Buffer Ident. Sirve para identificar el buffer. Esta propiedad será usada por los métodos protegidos de esta clase
* Maximum allowed buffer size. Es una constant que se ha codificado como una propiedad de esta clase padre. Limita el tamaño del buffer y por tanto de la cola más grande que se puede crear.

**Métodos protegidos**

Tiene dos métodos protegidos. Estos métodos se utilizan por los diferentes métodos de las clases hijas para crear e identificar las colas.

* GetBufferName. Genera el nombre de la cola de manera que no existan dos colas con el mismo nombre
* GetIdent. Genera o recoge el número que identifica al buffer. Este número es único para cada ejecución

Un acceso de lectura a la propiedad “Maximum allowed buffer size”. Este acceso está protegido de tal manera que sólo puede esta accesible para las clases hijas y no se puede acceder desde el exterior del árbol de clases.

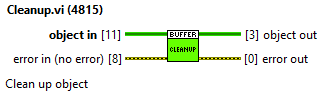
**Métodos públicos**

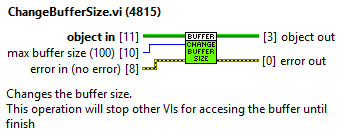
Los métodos públicos de esta clase son los métodos de la paleta y por tanto son los métodos que se van a usar junto con el método init de cada una de las clases hijas.

Existen 2 tipos de métodos en función de la necesidad.

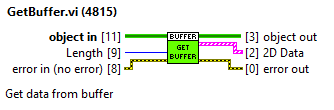
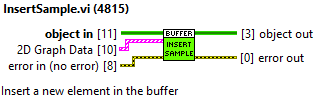
* El primer tipo son métodos que requieren de que las clases hijas tengo un método del mismo nombre que haga el override
* El segundo método son VIs polimórficos que llaman a métodos estáticos de las clases hijas. Estos métodos no pueden ser de tipo override, porque el conector de entrada/salida no es igual, ya que se adapta al tipo de datos en cada clase. Los métodos de las clases no pueden tener el mismo nombre así que como convenio se llamarán igual que el padre y se le añade el tipo de dato al final.

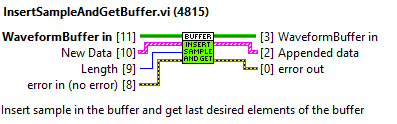
Los métodos de override son

* Cleanup. Limpia el objeto, eliminando datos y referencias  
  
* ChangeBufferSize. Cambia el tamaño del buffer. Para ello destruye la cola y vuelve a encolar los datos que tenía en la cola que crea. Mientras se ejecuta este método el resto de métodos están bloqueados, hay que tenerlo en cuenta en buffers de gran tamaño.



Los métodos implementados como polimórficos son:

* GetBuffer. Al llamar a este método se obtienen los últimos valores insertados en el buffer. La entrada Length permite manejar el número de elementos devueltos. Un número negativo en esta entrada devolverá siempre el buffer entero, sino devolverá el número deseado o los datos que haya en el buffer  
  
* InsertSample. Inserta un dato en el buffer. Si el buffer esta completo eliminará el más antiguo  
  
* InsertSampleAndGetBuffer. Realiza las dos operaciones anteriores en un mismo paso. Este VI está pensado para sustituir al buffer antiguo.



## Clases hijas

En este apartado se describen las propiedades de cada una de las clases implementadas, pero no se describen los métodos, ya que estos son los mismos que para la clase padre.

### TimeDataBuffer

Este buffer es del tipo array de estructuras de los campos array de tiempos y array de doubles. Esta pensado para capturas en las que se tiene el tiempo y el dato por separado.

### WaveformBuffer

Este buffer es de tipo array de waveforms. Esta pensado para la captura típica de array de seañales en formato waveform.

### DBL2DArray

Este buffer es de tipo array de 2D de doubles. Esta pensado para señales sin tiempo. Cada array es una señal. El buffer concatena los arrays a la salida.

# Como realizar una clase nueva

Para realizar una clase nueva hay que seguir los siguientes pasos

1. Realizar un branch en el repositorio git o al menos cambiar el Branch “BufferObjectOriented”
2. Abrir el proyecto BufferObjectOriented situado en ..\ LabVIEW 2016\user.lib\TeknikerLib2\BufferTek\BufferObjectOriented.
3. Clonar una de las clases hijas
4. Modificar el control púbico TipoDatoTypeDef, para que se ajuste al tipo de datos deseado
5. Renombrar el control cambiando el TipoDato por el nuevo tipo de dato
6. Renombrar los métodos GetBufferTipoDato, InsertSampleTipoDato e InsertSampleAndGetBufferTipoDato, sustituyento TipoDato por el nuevo tipo de dato
7. Corregir los errores de los métodos
8. Añadir los nuevos métodos en la clase padre a los métodos polimórficos
9. Crear los test de VI tester para la nueva clase. Se puede clonar una de las clases de test y modificar los test. Hay que testear al menos los métodos públicos mencionados (ver el test de GetSample)
10. Una vez superados todos los test, para lo cual habrá que modificar los test, añadir la clase de test al test suite. Para ello abrir el New.vi de la clase TestAllBufferTypes y añadirlo al array de las demás clases. Esto es un Test suite
11. Correr el VI tester para este test suite y verificar que pasan nuevamente todos los test.
12. Guardar los resultados junto con los anteriores
13. Documentar lo que hace la clase en este documento y añadir datos al control de cambios del documento en la página 1.
14. Modificar la paleta de los buffers añadiendo el TypeDatoBuffer\_Init.
15. Hacer un commit de los cambios
16. Cambiar al Branch master
17. Hacer un merge del Branch utilizado para el desarrollo de la clase.
18. Hacer push del master y preferiblemente del Branch utilizado. Si se usa el Branch “BufferObjectOriented” es obligatorio realizar un push de este Branch.