Documentación de proyectos con Doxygen

José Tomás Tocino García
josetomas.tocino@uca.es

16 de marzo de 2011

¿Qué es Doxygen?

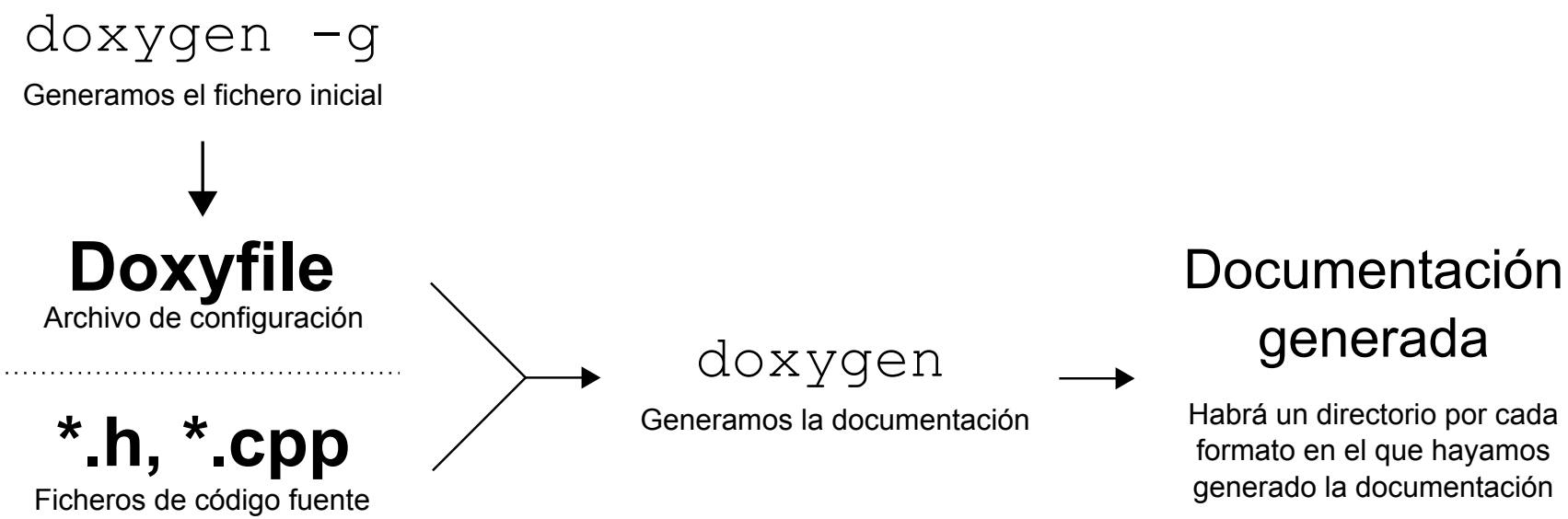
Doxygen es una herramienta de generación automática de documentación.

Es capaz de leer el código y documentarlo desde cero o ayudado de una sintaxis especial, muy sencilla, que aprenderemos aquí.

Funciona con muchos lenguajes: C, C++, Java, PHP, Python, VHDL...

El resultado también puede mostrarse en muchos formatos: en forma de web HTML, como un documento PDF, etcétera.

Elementos que conforman Doxygen



- Doxygen viene en los repositorios de la mayoría de distribuciones:
  `sudo apt-get install doxygen`
- El fichero `doxyfile` guarda las opciones de configuración. Por defecto se genera en formato HTML y LaTeX.

Principales opciones del fichero de configuración

- Nombre del proyecto:
  `PROJECT_NAME = SecretProject`
- Idioma de la interfaz:
  `OUTPUT_LANGUAGE = English (*)`
- Archivos a leer:
  `INPUT =`
  `FILE_PATTERNS =`
- Extraer todo:
  `EXTRACT_ALL = YES`
- Formatos a generar:
  `GENERATE_HTML = YES`
- Usar Graphviz para generar los diagramas:
  `HAVE_DOT = YES`
- Descripción breve, termina con un punto:
  `JAVADOC_AUTOBRIEF = YES`

Sintaxis básica para documentar el código

Los comentarios que Doxygen interpreta como documentación tienen una sintaxis especial. Siempre se colocan justo antes del código que queramos documentar.

Las sintaxis más habituales son los comentarios con tres barras:

```
1  /// Esto es un comentario de Doxygen
```

O los comentarios en bloque con **dos asteriscos** de inicio:

```
1  /**
2    * Esto es un comentario de bloque
3    */
```

Partes de la documentación de un elemento

Doxygen divide la documentación de los elementos en tres partes:

1. **Descripción corta**: en una línea, explicar brevemente el código.
2. **Descripción larga**: párrafo con varias líneas con una explicación más extensa. Hay que dejar un salto de línea para indicar el comienzo de la descripción larga.
   Si tenemos activado `JAVADOC_AUTOBRIEF = YES`, la descripción larga comienza tras el primer punto en los comentarios tipo `/** ... */`
3. **Elementos adicionales**: podemos documentar, por ejemplo, para qué sirve cada parámetro de entrada, o qué valores devuelve la función, etc. Utilizaremos unos comandos del estilo @brief, @param, @return...

Un ejemplo

```
1  /// Representa una ventana.
2  /// Esto es la descripcion detallada
3  class Ventana{
4  public:
5      /// Abre la ventana.
6      /// Las bisagras deben estar bien engrasadas.
7      void Abrir();
8
9      /// Color de la ventana.
10     int color;
11 };
```

Un ejemplo

Mismo ejemplo, pero sintaxis alternativa

```
1  /**
2   * @brief Representa una ventana.
3   *
4   * Esto es la descripcion detallada.
5   */
6  class Ventana{
7  public:
8      /**
9       * @brief Abre la ventana.
10      *
11      * Las bisagras deben estar bien engrasadas.
12      */
13     void Abrir();
14
15     /** Color de la ventana */
16     int color;
17 };
```

Elementos adicionales

Los elementos adicionales aportan mucha información:

```
1       /**
2        * @brief Cierra la ventana.
3        *
4        * Permite indicar la velocidad a la que
5        * cerrar la ventana.
6        *
7        * @param v Velocidad a la que se cierra.
8        *
9        * @return true si la ventana se ha roto
10       * del portazo.
11       *
12       */
13      bool Cerrar(float v);
```

Elementos adicionales

Hay una lista inmensa de comandos que podemos usar además de los ya vistos:

- `@author`: autor del código.
- `@date`: fecha.
- `@exception`: posibles excepciones que lanza el método.
- `@todo`: elementos pendientes en la lista TO-DO.
- Podemos agrupar elementos:

```
1        /// @{
2        /// @name Coordenadas de las ventanas
3        float x;
4        float y;
5        /// @}
```

¿Preguntas?