Block Tags

@param	Documents a value parameter of a function or a type parameter of a class, property or function.
@param[*name*]	To better separate the parameter name from the description, if you prefer, you can enclose the name of the parameter in brackets.
@return	Documents the return value of a function.
@constructor	Documents the primary constructor of a class.
@receiver	Documents the receiver of an extension function.
@property	Documents the property of a class which has the specified name. This tag can be used for documenting properties declared in the primary constructor, where putting a doc comment directly before the property definition would be awkward.
@throws class, @exception class	Documents an exception which can be thrown by a method. Since Kotlin does not have checked exceptions, there is also no expectation that all possible exceptions are documented, but you can still use this tag when it provides useful information for users of the class.
@sample *identifier*	Embeds the body of the function with the specified qualified name into the documentation for the current element, in order to show an example of how the element could be used.
@see *identifier*	Adds a link to the specified class or method to the See also block of the documentation.
@author	Specifies the author of the element being documented.
@since	Specifies the version of the software in which the element being documented was introduced
@suppress	Excludes the element from the generated documentation. Can be used for elements which are not part of the official API of a module but still have to be visible externally.

Example

```
/**
 * A group of *members*.
 *
 * This class has no useful logic; it's just a docume ntation example.
 *
 * @param T the type of a member in this group.
 * @property name the name of this group.
 * @const ructor Creates an empty group.
 */
class Group< T>(val name: String) {
       /**
         * Adds a [member] to this group.
         * @return the new size of the group.
         */
       fun add(me mber: T): Int { ... }
}
```