# Como documentar um programa typescript com a sintaxe tsdoc ↵ ℘

## 1. INDEX



## 2. CONTEÚDO

1. **Introdução**
    1. **Objetivo:**
        1. O TSDoc é uma proposta para padronizar os comentários de documentos usados no código TypeScript , para que diferentes ferramentas possam extrair conteúdo sem se confundir com a marcação umas das outras. Mais informações sobre a especificação tsdoc veja aqui.
        2. Com os comentários padronizados baseado em TSDoc o programa TypeDoc gera um site inteiro com a documentação de todas as classes e funções.
        3. [🔙]

2. **Pre-requisitos:**

    1. Conhecimento das linguagem javascript e typescript.

    2. Os pacotes abaixo devem estar instalados e configurados:

        1. nodejs - Um tempo de execução de JavaScript que nos permite executar javascript fora de um navegador. Também nos permite executar Javascript no lado do servidor.
        2. npm - Significa **Node Package Manager** e é uma ferramenta que nos permite instalar e gerenciar pacotes de nodejs como dependências.
        3. npx - **NPX** é um executor de pacote **NPM** que torna realmente fácil instalar qualquer tipo de executável de nodejs que normalmente teria sido instalado usando NPM.

    3. [🔙]

3. **Benefícios:**

    1. Padronizar a documentação do código typescript para que qualquer programador possa dar continuidade na programação do projeto.

    2. Para que uma aplicação cliente possa acessar um banco de dados ou qualquer serviço do servidor, é preciso saber o nome de cada método, parâmetros de entrada do método e tipo de resposta do método. Sem uma documentação precisa e clara, esses serviços não poderão ser consumidos pelos clientes.

    3. Existe um programa de nome typeDoc que gera a documentação html do código baseado na especificação TSDoc.

    4. [🔙]

2. **Resumo**

    1. O padrão **tsdoc** criado pela microsoft é mais simples que o padrão jsdoc da linguagem javascript, porque o mesmo, é aplicado a linguagem typescript. A linguagem typescript é tipada e por isso muitas tags do jsdoc relacionadas a tipos de dados, não serão necessárias documentar, visto que a própria sintaxe typescript descreve os tipos de dados necessários.

    2. Existe um programa gerador de documentos de nome typedoc que gera um site inteiro usando os comentários de todos os códigos typescripts contidas em uma pasta.

    3. Outro recurso muito útil da especificação tsdoc é permitir que programas possam criticar se a versão da api a ser publicada é compatível com a versão anterior publicada. Existe um programa open source chamado api_extractor responsável por essa tarefa.

    4. A melhor maneira de aprender é fazendo, e neste documento criamos um capítulo de exemplos de uso das tags tsdoc.

    5. Recomendo ler o item que descreve um exemplo completo neste documento.

    6. [🔙]

3. **Descrição**

    1. O pacote **@microsoft/tsdoc** é a implementação de referência de um analisador para a sintaxe TSDoc. Você não pode usá-lo diretamente. É um componente do mecanismo que deve ser incorporado a outras ferramentas de documentação.

    2. Se você estiver implementando uma ferramenta que precisa extrair informações dos comentários do código TypeScript, **@microsoft/tsdoc** fornece uma solução fácil que implementará corretamente o TSDoc.

    3. A anatomia geral de um comentário TSDoc tem estes componentes:

        1. **A seção de resumo:** O conteúdo da documentação até a primeira tag de bloco é chamado de “resumo”. A seção de resumo deve ser breve. Em um site de documentação, ele será mostrado em uma página que lista resumos de vários itens de API diferentes. Em uma página de detalhes para um único item, o resumo será mostrado seguido pela seção de comentários (se houver).

        2. **O bloco de comentários:** O bloco de “comentários” começa com a tag **@remarks** do bloco. Ao contrário do resumo, as observações podem conter um conteúdo de documentação extenso. A seção de comentários não deve reformular as informações do resumo, uma vez que a seção de resumo sempre será exibida onde quer que a seção de comentários apareça.

        3. **Blocos adicionais:** Outros blocos TSDoc normalmente seguem o **@remarks** bloco. Cada bloco é introduzido por uma etiqueta de bloco, tais como **@param**, **@returns**, **@example**, etc.

        4. **Tags modificadoras:** as tags modificadoras geralmente vêm por último. As tags modificadoras não têm um bloco de conteúdo associado; em vez disso, sua presença indica um aspecto da declaração. Alguns exemplos de marcas modificadoras são: **@public**, **@beta**, e **@virtual**.

        5. **Tags embutidas:** as tags embutidas podem aparecer dentro de qualquer seção e são sempre delimitadas pelos caracteres **{** e **}**. O conteúdo adicional pode aparecer após o nome da tag e antes do **}** delimitador. Seu formato é específico da tag. Exemplos de tags embutidas são **{@link alvo | título do alvo}** e **{@inheritDoc}**.

    4. Tags de liberação:

        1. Os quatro tags de lançamento são: @internal, @alpha, @beta, @public. Eles são aplicados a itens de API, como classes, funções de membro, enums, etc. O API Extractor classifica cada item de API exportado individualmente, de acordo com seu nível de suporte pretendido:

            1. **internal:** indica que um item de API deve ser usado apenas por outros pacotes NPM do mesmo mantenedor. Terceiros nunca devem usar APIs “internas”. Para enfatizar isso, os prefixos de sublinhado devem ser usados para itens com uma tag @internal (explícita).

    2. **alpha:** indica que um item de API eventualmente se destina a ser público, mas atualmente está em um estágio inicial de desenvolvimento. Terceiros não devem usar APIs “alfa”.

    3. **beta:** indica que um item da API foi lançado como uma visualização ou para fins experimentais. Os terceiros são incentivados a experimentá-lo e fornecer feedback. No entanto, uma API “beta” NÃO deve ser usada na produção, pois pode ser alterada ou removida em uma versão futura.

    4. **public:** indica que um item de API foi lançado oficialmente e agora faz parte do contrato com suporte de um pacote. Se o esquema de versão SemVer for usado, a assinatura da API não poderá ser alterada sem um incremento de versão MAJOR.

2. Quando uma API é introduzida pela primeira vez, geralmente começa como **@alpha**. Conforme o design amadurece, ele se gradua em **@alpha** -> **@beta** -> **@public** . A designação **@internal** é usada principalmente para resolver problemas de encanamento e geralmente não está em nenhum roteiro para se tornar pública. (Também existe uma tag **@deprecated**, mas é uma opção que pode ser combinada com qualquer uma das tags acima.)

3. A tag de liberação se aplica recursivamente aos membros de um contêiner (por exemplo, **class** ou **interface**). Por exemplo, se uma classe for marcada como **@beta**, todos os seus membros terão automaticamente esse status; você NÃO precisa adicionar a tag **@beta** a cada função de membro. No entanto, você pode adicionar **@internal** a uma função de membro para dar a ela um status de liberação diferente.

4. Por último, observe que certas regras lógicas se aplicam. Por exemplo, uma **@public** função não deve retornar um tipo **@beta**. Uma classe **@beta** não deve herdar de uma classe base **@internal**, etc. Atualmente, o API Extractor não valida essas regras, mas o fará em breve.

5. [🔙]

4. As tags TSDoc abaixo são usadas dentro de um comentário (/** */) precedida do caractere @. Quando precisar que @tag seja visualizada no texto então coloca-la entre o caractere ( ` crase). Exemplo: `@tag`.

    1.

| TSDOC | JSDOC | DESCRIÇÃO |
|---|---|---|
| @alpha | x | Designa que o estágio de lançamento de um item de API é “alfa”. Destina-se a ser usado por desenvolvedores de terceiros eventualmente, mas ainda não foi lançado. O ferramental pode cortar a declaração de uma liberação pública. Exemplo... |

| TSDOC | JSDOC | DESCRIÇÃO |
|---|---|---|
| @beta | x | Designa que o estágio de lançamento de um item de API é “beta”. Ele foi lançado experimentalmente para desenvolvedores terceirizados com o objetivo de coletar feedback. O API não deve ser usado na produção, pois seu contrato pode ser alterado sem aviso prévio. A ferramenta pode cortar a declaração de uma versão pública, mas pode incluí-la em uma versão de visualização do desenvolvedor. Exemplo... |
| @decorator | x | Os decoradores ECMAScript às vezes são uma parte importante de um contrato de API. No entanto, hoje o compilador TypeScript não representa decoradores nos arquivos de saída .d.ts usados por consumidores de API. A @decorator tag fornece uma solução alternativa, permitindo que uma expressão do decorador seja citada em um comentário de documento. Exemplo... |
| @default Value | @default | Esta tag de bloco é usada para documentar o valor padrão para um campo ou propriedade, se um valor não for atribuído explicitamente. Essa tag só deve ser usada com campos ou propriedades que são membros de um TypeScript class ou interface. Exemplo... |
| @deprecated | @deprecated | Esta tag de bloco comunica que um item de API não é mais compatível e pode ser removido em uma versão futura. A @deprecated tag é seguida por uma frase que descreve a alternativa recomendada. Ele se aplica recursivamente aos membros do contêiner. Por exemplo, se uma eventproperty classe for descontinuada, todos os seus membros também serão. Exemplo... |
| @event Property | @event | Quando aplicada a uma classe ou propriedade de interface, indica que a propriedade retorna um objeto de evento ao qual os manipuladores de eventos podem ser anexados. A API de manipulação de eventos é definida pela implementação, mas normalmente o tipo de retorno de propriedade seria uma classe com membros como addHandler()e removeHandler(). Uma ferramenta de documentação pode exibir essas propriedades sob um título “Eventos” em vez do título “Propriedades” usual. Exemplo... |

| TSDOC | JSDOC | DESCRIÇÃO |
|---|---|---|
| @example | @example | No **Tsdoc** indica uma seção de documentação que deve ser apresentada como um exemplo ilustrando como usar a API. Pode incluir um exemplo de código. Qualquer texto subsequente que apareça na mesma linha que a @example tag deve ser interpretado como um título para o exemplo. Caso contrário, a ferramenta de documentação pode indexar os exemplos numericamente.<br>**ATENÇÃO**: **Jsdoc** tem sintaxe é diferente. Exemplo... |
| @experimental | x | No **tsdoc** tem mesma semântica **@beta**, mas usada por ferramentas que não suportam um **@alpha** no estágio de lançamento. Exemplo... |
| @inherit Doc | @inheritDoc | Esta tag inline é usada para gerar automaticamente a documentação de um item de API, copiando-o de outro item de API. O parâmetro de tag inline contém uma referência ao outro item, que pode ser uma classe não relacionada ou até mesmo uma importação de um pacote NPM separado.<br>**ATENÇÃO**: A sintaxe de **jsdoc** é diferente da **tsdoc**. Exemplo... |
| @internal | x | Designa que um item de API não deve ser usado por desenvolvedores terceirizados. O ferramental pode cortar a declaração de uma liberação pública. Em algumas implementações, certos pacotes designados podem consumir itens internos da API, por exemplo, porque os pacotes são componentes do mesmo produto. Exemplo... |
| @label | x | A tag {@label} inline é usada para rotular uma declaração, de forma que ela possa ser referenciada usando um seletor na notação de referência da declaração TSDoc. Exemplo... |
| @link | @link | A tag {@link} inline é usada para criar hiperlinks para outras páginas em um sistema de documentação ou URLs gerais da Internet. Em particular, ele oferece suporte a expressões para fazer referência a itens de API. Exemplo... |

| TSDOC | JSDOC | DESCRIÇÃO |
|---|---|---|
| @override | @override | Este modificador tem semântica semelhante à **override** palavra - chave em C# ou Java. Para uma função de membro ou propriedade, indica explicitamente que esta definição está substituindo (ou seja, redefinindo) a definição herdada da classe base. A definição da classe base normalmente seria marcada como **virtual**. Um instrumento de documentação pode impor que as **@virtual**, **@override**, e / ou **@sealed** modificadores são aplicados de forma consistente, mas isto não é exigido pela norma TSDoc. Exemplo... |
| @package Documentation | x | Usado para indicar um comentário de documento que descreve um pacote NPM inteiro (em oposição a um item de API individual pertencente a esse pacote). O comentário @packageDocumentation é encontrado no arquivo*.d.ts que atua como o ponto de entrada para o pacote e deve ser o primeiro /**comentário encontrado nesse arquivo. Um comentário contendo uma tag @packageDocumentation nunca deve ser usado para descrever um item individual da API. Exemplo... |
| @param | @param | Usado para documentar um parâmetro de função. A tag @param é seguida por um nome de parâmetro, seguido por um hífen, seguido por uma descrição.<br>ATENÇÃO: O jsdoc tem descrição mais detalhada porque javascript não tem tipo. Exemplo... |
| @private Remarks | @private | Inicia uma seção de conteúdo de documentação adicional que não se destina ao público. Uma ferramenta deve omitir esta seção inteira do site de referência da**API**, o arquivo *.d.ts gerado e quaisquer outras saídas que incorporem o conteúdo. Exemplo... |
| @public | @public | Designa que o estágio de lançamento de um item de API é “público”. Ele foi oficialmente lançado para desenvolvedores terceirizados e sua assinatura é garantida como estável (por exemplo, seguindo as regras de controle de versão semântica). Exemplo... |

| TSDOC | JSDOC | DESCRIÇÃO |
|---|---|---|
| @readonly | @readonly | Esta tag modificadora indica que um item de API deve ser documentado como sendo somente leitura, mesmo se o sistema de tipo TypeScript puder indicar o contrário. Por exemplo, suponha que uma propriedade de classe tenha uma função setter que sempre lança uma exceção explicando que a propriedade não pode ser atribuída; nesta situação, a tag @readonly pode ser adicionado para que a propriedade seja mostrada como somente leitura na documentação. Exemplo... |
| @remarks | x | A documentação principal de um item de API é separada em uma breve seção de “resumo”, opcionalmente seguida por uma seção de “comentários” mais detalhada. Em um site de documentação, as páginas de índice (por exemplo, mostrando os membros de uma classe) mostrarão apenas os breves resumos, enquanto as páginas de detalhes (por exemplo, descrevendo um único membro) mostrarão o resumo seguido pelos comentários. A tag @remarks de bloco termina a seção de resumo e começa a seção de comentários para um comentário de documento. Exemplo... |
| @returns | @returns | Usado para documentar o valor de retorno de uma função. Exemplo... |
| @sealed | x | Este modificador tem semântica semelhante à sealed palavra - chave em C # ou Java. Para uma classe, indica que as subclasses não devem herdar da classe. Para uma função ou propriedade de membro, indica que as subclasses não devem substituir (ou seja, redefinir) o membro. Um instrumento de documentação pode impor que as @virtual, @override, e / ou @sealed modificadores são aplicados de forma consistente, mas isto não é exigido pela norma TSDoc. Exemplo... |
| @see | @see | Usado para construir uma lista de referências a um item de API ou outro recurso que pode estar relacionado ao item atual.<br>**Observação:** o **JSDoc** tenta criar um hiperlink automático para o texto imediatamente após @see. Como isso é ambíguo com texto sem formatação, o TSDoc exige uma {@link}marca explícita para fazer hiperlinks. Exemplo... |

| TSDOC | JSDOC | DESCRIÇÃO |
|---|---|---|
| @throws | @throws | Usado para documentar um tipo de exceção que pode ser lançado por uma função ou propriedade.<br>Um bloco @throws separado deve ser usado para documentar cada tipo de exceção. Esta tag é apenas para fins informativos e não impede que outros tipos sejam lançados. É sugerido, mas não obrigatório, que o bloco @throws comece com uma linha contendo apenas o nome da exceção. Exemplo... |
| @type Param | x | Usado para documentar um parâmetro genérico. A tag @typeParam é seguida por um nome de parâmetro, seguido por um hífen, seguido por uma descrição. O analisador TSDoc reconhece essa sintaxe e a extrai em um nó DocParamBlock. Exemplo... |
| @virtual | @abstract | Este modificador tem semântica semelhante à palavra-chave **virtual** em C# ou Java. Para uma função ou propriedade de membro, indica explicitamente que as subclasses podem substituir (ou seja, redefinir) o membro.<br>Um instrumento de documentação pode impor que as @virtual, @override, e / ou @sealed são modificadores aplicados de forma consistente, mas isto não é exigido pela norma TSDoc. Exemplo... |
| @ignore | @ignore | No aplicativo TypeDoc essa tag impede que o código subsequente seja documentado. Exemplo... |
| x | x | A tag @category do gerador de documento TypeDoc reconhece essa tag. Não encontrei um exemplo prático para esta tag. Exemplo... |
| x | @module | O programa TypeDoc usa para especificar um comentário na parte superior de um arquivo de origem para documentar esse arquivo de origem e, opcionalmente, substituir o nome de um ponto de entrada. Consulte a **seção Arquivos** para obter mais detalhes. Exemplo... |
| x | x | ... |
| x | x | ... |
| x | x | ... |

2. [🔙]

5. **Exemplos.**

1. Exemplo da tag @alpha:

    1. Código TypeScript:

```
/**
* Represents a book in the catalog.
* @public
*/
export class Book {
   /**
      * The title of the book.
      * @alpha
      */
   public get title(): string;{}

   /**
      * The author of the book.
      */
   public get author(): string;{}
};

Nota: Neste exemplo, Book.author herda sua @public
designação
da classe que o contém, enquanto Book.title é marcado como
“alfa”.
```

    2. [⬅️]

2. Exemplo da tag @beta:

    1. Código TypeScript:

```
/**
* Represents a book in the catalog.
* @public
*/
export class Book {
   /**
      * The title of the book.
      * @beta
      */
      public get title(): string;{}

      /**
         * The author of the book.
         */
      public get author(): string;{}
};

Nota: Neste exemplo, Book.author herda sua @public
designação
```

```
da classe que o contém, enquanto Book.title é marcado como
“beta”.
```

2. [⬅️]

3. Exemplo da tag @decorator

    1. Código TypeScript

```
class Book {
   /**
      * The title of the book.
      * @decorator `@jsonSerialized`
      * @decorator `@jsonFormat(JsonFormats.Url)`
      */
   @jsonSerialized
   @jsonFormat(JsonFormats.Url)
   public website: string;{}
}
Nota: RFC #271: @decorator tag for documenting ECMAScript
decorators
```

    2. [⬅️]

4. Exemplo da tag @defaultValue

    1. Código TypeScript

```


enum WarningStyle { DialogBox, StatusMessage, LogOnly}

interface IWarningOptions {
   /**
   * Determines how the warning will be displayed.
   *
   * @remarks
   * See {@link WarningStyle| the WarningStyle enum} for
more details.
   *
   * @defaultValue `WarningStyle.DialogBox`
   */
   warningStyle: WarningStyle;

   /**
   * Whether the warning can interrupt a user's current
activity.
   * @defaultValue
   * The default is `true` unless
   *  `WarningStyle.StatusMessage` was requested.
   */
```

```
    cancellable?: boolean;

    /**
    * The warning message
    */
    message: string;
}
```

2. [⬅BACK]

5. Exemplo da tag @deprecated

    1. Código TypeScript

```
/**
* The base class for controls that can be rendered.
*
* @deprecated Use the new {@link Control} base class
instead.
*/
export class VisualControl {
. . .
}
```

    2. [⬅BACK]

6. Exemplo da tag @eventProperty

    1. Código TypeScript

```
class MyClass {
/**
    * This event is fired whenever the application navigates
to a new page.
    * @eventProperty
    */
public readonly navigatedEvent:
FrameworkEvent<NavigatedEventArgs>;
}
```

    2. [⬅BACK]

7. Exemplo da tag @example

    1. Código TypeScript

```
Nota: Para este exemplo de código, os títulos gerados podem
ser “Exemplo” e “Exemplo 2” :
```

```
/**
* Adds two numbers together.
* @example
* Here's a simple example:
* ```
* // Prints "2":
* console.log(add(1,1));
* ```
* @example
* Here's an example with negative numbers:
* ```
* // Prints "0":
* console.log(add(1,-1));
* ```
*/
export function add(x: number, y: number): number {
}
```

2. [⬅BACK]

8. Exemplo da tag @experimental

1. Código TypeScript

```
/**
* Represents a book in the catalog.
* @public
*/
export class Book {
   /**
      * The title of the book.
      * @experimental
      */
   public get title(): string;{}

   /**
      * The author of the book.
      */
   public get author(): string;{}
};

Nota: Neste exemplo, Book.author herda sua @public
designação
da classe que o contém, enquanto Book.title é marcado como
“experimental”.
```

2. [⬅BACK]

9. Exemplo da tag @inheritDoc

1. Código TypeScript

```
import { Serializer } from 'example-library';

/**
 * An interface describing a widget.
 * @public
 */
export interface IWidget {
   /**
    * Draws the widget on the display surface.
    * @param x - the X position of the widget
    * @param y - the Y position of the widget
    */
   public draw(x: number, y: number): void;
}

/** @public */
export class Button implements IWidget {

   /** {@inheritDoc IWidget.draw} */
   public draw(x: number, y: number): void {
      . . .
   }

   /**
      * {@inheritDoc example-library#Serializer.writeFile}
      * @deprecated Use {@link example-
library#Serializer.writeFile} instead.
      */
   public save(): void {
      . . .
   }
}
```

2. [⬅️BACK]

10. Exemplo da tag @internal

1. Código TypeScript

```
/**
 * Represents a book in the catalog.
 * @public
 */
export class Book {
    /**
    * The title of the book.
    * @internal
    */
```

```
    public get _title(): string;{}

    /**
    * The author of the book.
    */
    public get author(): string;{}
};
```

2. [🔙]

11. Exemplo da tag @label

    1. Código TypeScript

```
export interface Interface {
  /**
  * Shortest name:  {@link InterfaceL1.(:STRING_INDEXER)}
  * Full name:      {@link (InterfaceL1:interface).
(:STRING_INDEXER)}
  *
  * {@label STRING_INDEXER}
  */
  [key: string]: number;

  /**
  * Shortest name:  {@link InterfaceL1.(:NUMBER_INDEXER)}
  * Full name:      {@link (InterfaceL1:interface).
(:NUMBER_INDEXER)}
  *
  * {@label NUMBER_INDEXER}
  */
  [key: number]: number;

  /**
  * Shortest name:  {@link InterfaceL1.(:FUNCTOR)}
  * Full name:      {@link (InterfaceL1:interface).
(:FUNCTOR)}
  *
  * {@label FUNCTOR}
  */
  (source: string, subString: string): boolean;

  /**
  * Shortest name:  {@link InterfaceL1.(:CONSTRUCTOR)}
  * Full name:      {@link (InterfaceL1:interface).
(:CONSTRUCTOR)}
  *
  * {@label CONSTRUCTOR}
  */
  new (hour: number, minute: number);
}
```

2. Nota: a {@label} notação não foi finalizada. Consulte a edição nº 9 do GitHub.

3. [⬅BACK]

12. Exemplo da tag @link

    1. Código TypeScript

```
 /**
  * Let's learn about the `{@link}` tag.
  *
  * @remarks
  *
  * Links can point to a URL: {@link
https://github.com/microsoft/tsdoc}
  *
  * Links can point to an API item: {@link Button}
  *
  * You can optionally include custom link text: {@link
Button | the Button class}
  *
  * Suppose the `Button` class is part of an external
package.
  * In that case, we can include the package name when
referring to it:
  *
  * {@link my-control-library#Button | the Button class}
  *
  * The package name can include an NPM scope and import
path:
  *
  * {@link @microsoft/my-control-library/lib/Button#Button |
the Button class}
  *
  * The TSDoc standard calls this notation a "declaration
reference".
  * The notation supports references to many different kinds
  * of TypeScript declarations.  This notation was
originally
  * designed for use in `{@link}` and `{@inheritDoc}` tags,
  * but you can also use it in your own custom tags.
  *
  * For example, the `Button` can be part of a TypeScript
namespace:
  *
  * {@link my-control-library#controls.Button | the Button
class}
  *
  * We can refer to a member of the class:
  *
  * {@link controls.Button.render | the render() method}
```

```
  *
  * If a static and instance member have the same name, we
can use a selector to distinguish them:
  *
  * {@link controls.Button.(render:instance) | the render()
method}
  *
  * {@link controls.Button.(render:static) | the render()
static member}
  *
  * This is also how we refer to the class's constructor:
  *
  * {@link controls.(Button:constructor) | the class
constructor}
  *
  * Sometimes a name has special characters that are not a
legal TypeScript identifier:
  *
  * {@link restProtocol.IServerResponse."first-name" | the
first name property}
  *
  * Here is a fairly elaborate example where the function
name is an ECMAScript 6 symbol,
  * and it's an overloaded function that uses a label
selector (defined using the `{@label}`
  * TSDoc tag):
  *
  * {@link my-control-library#Button.
([UISymbols.toNumberPrimitive]:OVERLOAD_1)
  * | the toNumberPrimitive() static member}
  *
  * See the TSDoc spec for more details about the
"declaration reference" notation.
  */
```

2. Nota: A notação para referências de declaração não foi finalizada. Consulte a edição nº 9 do GitHub.

3. [⬅️]

13. Exemplo da tag @override

    1. Código TypeScript

```
class Base {
  /** @virtual */
  public render(): void {
  }

  /** @sealed */
  public initialize(): void {
  }
```

```
}

class Child extends Base {
  /** @override */
  public render(): void;
}
```

2. [🔙]

14. Exemplo da tag @packageDocumentation

    1. Código TypeScript

```
 // Copyright (c) Example Company. All rights reserved.
Licensed under the MIT license.

 /**
  * A library for building widgets.
  *
  * @remarks
  * The `widget-lib` defines the {@link IWidget} interface
and {@link Widget} class,
  * which are used to build widgets.
  *
  * @packageDocumentation
  */

  /**
  * Interface implemented by all widgets.
  * @public
  */
  export interface IWidget {
     /**
        * Draws the widget on the screen.
        */
     render(): void;
  }
```

    2. [🔙]

15. Exemplo da tag @param

    1. Código TypeScript

```
/**
  * Returns the average of two numbers.
  *
  * @remarks
```

```
  * This method is part of the {@link core-
library#Statistics | Statistics subsystem}.
  *
  * @param x - The first input number
  * @param y - The second input number
  * @returns The arithmetic mean of `x` and `y`
  *
  * @beta
*/
function getAverage(x: number, y: number): number {
  return (x + y) / 2.0;
}
```

2. [⬅️]

16. Exemplo da tag @privateRemarks

    1. Código TypeScript

```
/**
  * The summary section should be brief. On a documentation
web site,
  * it will be shown on a page that lists summaries for many
different
  * API items.  On a detail page for a single item, the
summary will be
  * shown followed by the remarks section (if any).
  *
  * @remarks
  *
  * The main documentation for an API item is separated into
a brief
  * "summary" section optionally followed by an `@remarks`
block containing
  * additional details.
  *
  * @privateRemarks
  *
  * The `@privateRemarks` tag starts a block of additional
commentary that is not meant
  * for an external audience.  A documentation tool must
omit this content from an
  * API reference web site.  It should also be omitted when
generating a normalized
  * *.d.ts file.
*/
```

    2. [⬅️]

17. Exemplo da tag @public

1. Código TypeScript

```
/**
  * Represents a book in the catalog.
  * @public
*/
export class Book {
  /**
     * The title of the book.
     * @internal
     */
  public get _title(): string;

  /**
     * The author of the book.
     */
  public get author(): string;
};
```

2. [⬅BACK]

18. Exemplo da tag @readonly

1. Código TypeScript

```
export class Book {
  /**
   * Technically property has a setter, but for
documentation purposes it should
   * be presented as readonly.
   * @readonly
  */
  public get title(): string {
     return this._title;
  }

  public set title(value: string) {
     throw new Error('This property is read-only!');
  }
}
```

2. [⬅BACK]

19. Exemplo da tag @remarks

1. Código TypeScript

```
/**
  * The summary section should be brief. On a documentation
```

```
web site,
  * it will be shown on a page that lists summaries for many
different
  * API items.  On a detail page for a single item, the
summary will be
  * shown followed by the remarks section (if any).
  *
  * @remarks
  *
  * The main documentation for an API item is separated into
a brief
  * "summary" section optionally followed by an `@remarks`
block containing
  * additional details.
  *
  * @privateRemarks
  *
  * The `@privateRemarks` tag starts a block of additional
commentary that is not meant
  * for an external audience.  A documentation tool must
omit this content from an
  * API reference web site.  It should also be omitted when
generating a normalized
  * *.d.ts file.
*/
```

2. [🔙]

20. Exemplo da tag @returns

1. Código TypeScript

```
/**
  * Returns the average of two numbers.
  *
  * @remarks
  * This method is part of the {@link core-
library#Statistics | Statistics subsystem}.
  *
  * @param x - The first input number
  * @param y - The second input number
  * @returns The arithmetic mean of `x` and `y`
  *
  * @beta
*/
function getAverage(x: number, y: number): number {
  return (x + y) / 2.0;
}
```

2. [🔙]

21. Exemplo da tag @sealed

    1. No exemplo de código abaixo, Child.render() substitui o membro virtual Base.render(), mas Base.initialize() não deve ser substituído porque está marcado como “sealed”.

    2. Código TypeScript

```
class Base {
  /** @virtual */
  public render(): void { }

/** @sealed */
  public initialize(): void {  }
}

class Child extends Base {
  /** @override */
  public render(): void;
}
```

    3. [🔙]

22. Exemplo da tag @see

    1. Código TypeScript

```
/**
  * Parses a string containing a Uniform Resource Locator
(URL).
  * @see {@link ParsedUrl} for the returned data structure
  * @see {@link https://tools.ietf.org/html/rfc1738|RFC
1738}
  * for syntax
  * @see your developer SDK for code samples
  * @param url - the string to be parsed
  * @returns the parsed result
*/
function parseURL(url: string): ParsedUrl;
```

    2. [🔙]

23. Exemplo da tag @throws

    1. Código TypeScript

```
/**
  * Retrieves metadata about a book from the catalog.
  *
  * @param isbnCode - the ISBN number for the book
```

```
  * @returns the retrieved book object
  *
  * @throws {@link IsbnSyntaxError}
  * This exception is thrown if the input is not a valid
ISBN number.
  *
  * @throws {@link book-lib#BookNotFoundError}
  * Thrown if the ISBN number is valid, but no such book
exists in the catalog.
  *
  * @public
*/
function fetchBookByIsbn(isbnCode: string): Book; {}
```

2. [🔙]

24. Exemplo da tag @typeParam

    1. Código TypeScript

```
/**
  * Alias for array
  *
  * @typeParam T - Type of objects the list contains
*/
type List<T> = Array<T>;

/**
  * Wrapper for an HTTP Response
  * @typeParam B - Response body
  * @param <H> - Headers
*/
interface HttpResponse<B, H> {
   body: B;
   headers: H;
   statusCode: number;
}
```

    2. [🔙]

25. Exemplo da tag @virtual

    1. No exemplo de código abaixo, Child.render() substitui o membro virtual Base.render():

    2. Código TypeScript

```
class Base {
  /** @virtual */
  public render(): void {}
```

```
  /** @sealed */
  public initialize(): void {}
}

class Child extends Base {
  /** @override */
  public render(): void; {}
}
```

3. [⬅️]

26. @ignore

    1. No gerar de documento TypeDoc essa tag impede que o código subsequente seja documentado.

        1. Código TypeScript

```
/** @ignore */
function doSomething(target: any, value: number): number;
```

    2. [⬅️]

27. @module

    1. Usado para especificar um comentário na parte superior de um arquivo de origem para documentar esse arquivo de origem e, opcionalmente, substituir o nome de um ponto de entrada. Consulte a **seção Arquivos** para obter mais detalhes.

        1. Código TypeScript

```
// file1.ts
/**
  * Documento do arquivo file1.ts
  *
  * Este é o programa principal do aplicativo e está
definido o menu principal.
  * @module - Menu Principal do aplicativo
  *
  * @module
  */
```

    2. [⬅️]

28. @xxx

    1. ...

        1. Código TypeScript

```

```

    2. [⬅️]

29. Aqui está um exemplo de código que ilustra todos os vários componentes da sintaxe do comentário do documento:

    1. Código TypeScript

```
  // Módulo baseWidget.ts
  /**
   * Documento do módulo baseWidget.ts
   *
   * @remark
   * Conjunto de classes abstratas usada para visualização
de telas.
    * - Link para módulo: {@link nscomponent | módulo
nscomponent}
    * - Link para classe: {@link nscomponent.TNSComponent |
Class TNSComponent}
   *
   *
   * @module - Classes abstratas para ferramentas visuais.
   *
   * @alpha
   */


  /**
   * A classe base para todos os widgets.
   *
   * @remarks
   * Para detalhes, veja {@link https://example.com/my-
protocol | as especificações do protocolo}.
   *
   * @internal
   */
  export abstract class BaseWidget {
      /**
  *     _title é a variável da propriedade pública
        título e só pode ser acessado pelos métodos get e
set.
       */
      private _title  : string;
```

```
      /**
       *
       * @param a_title - String com o título do widget
       */
      constructor (a_title:string){
          this._title = a_title;
      }

    /**
     * Desenha o widget.
     *
     * @remarks
     *
     * O membro `draw` implementa a renderização principal
para um widget.
     *
     * @param force
     *   - true - se deve forçar o redesenho;
     *   - false - se não deve forçar o redesenho.
     *
     * @returns
     *   - true, se a renderização ocorreu;
     *   - falso, se a visualização já estava atualizada
     *
     * @beta @virtual
     */
    public draw(force: boolean): boolean {
      return force;
    }

    /**
     * Se o widget está visível no momento.
     *
     * @example
     * Aqui está um exemplo de código para ocultar um
widget:
     *
     * ```typescripts
     * let myWidget = new MyWidget();
     * myWidget.visible = false;
     * ```
     *
     * @defaultValue `true`
     */
    public visible: boolean = true;


    /**
     *
     * Os métodos **Get** ou **set** ler ou grava o atributo
**_title** da propriedade **title**.
     * Veja mais sobre propriedade {@link
https://javascript.info/property-
```

```
accessors#:~:text=Accessor%20descriptors&text=get%20%E2%80%93%20a%20function%20without%20arguments,same%20as%20for%20data%20properties. | aqui}.
     */
    public get title(): string {
      return this._title;
    }

    // NOTA: API Extractor e typedoc considera suas funções getter e setter
    // de propriedade como um único item de API.
    // Não escreva nenhuma documentação para o setter.
    public set title(value: string) {
      this._title = value;
    }
  }
```

2. [⬅BACK]

30. [⬅BACK]

6. **Instalar**

    1. NPM packages.

        1. O plugin eslint-plugin-tsdoc fornece uma regra para validar se os comentários de documentos do TypeScript estão em conformidade com a especificação **TSDoc**. Veja mais...

            1. Código typescript

```
    /**Seleciona a pasta do seu projeto*/
    cd my-project

    //** Instala o pacote eslint-plugin-tsdoc com a opção --save-dev */
    npm install --save-dev eslint-plugin-tsdoc
```

            2. [⬅BACK]

        2. O TypeDoc é um gerador de documentação para projetos TypeScript, é executado em Node.js e está disponível como um pacote NPM. Veja mais sobre typeDoc aqui.

            1. instalar TypeDoc:

                1. Código shellscript

```
   /** Selecione a pasta do seu projeto e execute o
comando*/
   npm install typedoc --save-dev
```

2. Uso

    1. Para gerar a documentação, o TypeDoc precisa saber o ponto de entrada (**entry point**) do projeto e as **opções do compilador** TypeScript. Ele tentará encontrar seu arquivo **tsconfig.json** automaticamente então você pode apenas especificar o ponto de entrada de sua biblioteca:

        1. Código shellscript

```
/** Selecione a pasta do seu projeto e
execute o comando*/
typedoc src/index.ts
```

    2. Se você tiver vários pontos de entrada, especifique cada um deles. Se você **especificar um diretório**, TypeDoc tratará cada arquivo contido nele como um ponto de entrada.

        1. Código shellscript

```
/** Selecione a pasta do seu projeto e
execute o comando*/
typedoc package1/index.ts
package2/index.ts

//** O comando abaixo é gerado documento
de dos arquivos da pasta ./ts */
typedoc ./ts
```

    3. Ao executar o typedoc a partir da CLI, você pode definir qualquer opção, exceto os arquivos de entrada no arquivo json denominado **typedoc.json**.

        1. Código do arquivo typedoc.json:

```

```

```
{
  "entryPoints": ["src/index.ts"],
  "out": "docs"
}
```

4. [⬅]

2. O pacote API Extractor ajuda a construir melhores pacotes de biblioteca TypeScript, fazendo crítica do código publicado em relação a versão anterior. Caso existe incompatibilidade entre versões, o programa avisa que as aplicações clientes não serão mais compatíveis com a versão atual do servidor bck-end.

    1. Para instalar o pacote NPM em seu ambiente global, use um comando como este:

        1. Código shellscript

```
sudo npm install -g @microsoft/api-extractor
```

        2. Supondo que sua variável **PATH** de ambiente esteja configurada corretamente, agora você deve ser capaz de invocar a ferramenta **api-extractor** de seu shell.

    2. Crie um arquivo de configuração de modelo:

        1. Em seguida, precisamos criar um arquivo de configuração **api-extractor.json** para o seu projeto. O comando a seguir criará um arquivo de modelo que mostra todas as configurações e seus valores padrão:

            1. Código shellscript

```
  /** Selecione a pasta do seu projeto e execute o
comando*/
  api-extractor init
```

        2. .

    3. [⬅]

7. **REFERÊNCIAS**

    1. O que é TSDoc?

    2. Como posso usar o TSDoc?

    3. packages depending on @microsoft/tsdoc

4. Checa se tem erros nos comentários tsdoc
5. @use JSDoc
6. OpenAPI Initiative Registry
7. TSDoc Comment
8. TSDoc Playground
9. typedoc-plugin-markdown
10. TypeDoc - Documente seu Código
11. API Extractor
12. extension vscode - Better Comments
13. [⬅️]

8. **HISTÓRICO**

1. 26/02/2021
    - ☑ Criar este documento baseado no modelo02.md;
    - ☑ Escrever tópico Objetivos;
    - ☑ Escrever tópico Pre-requisitos
    - ☑ Escrever tópico Benefício
    - [⬅️]
2. 02/02/2021
    - ☑ Escrever tópico descrição
    - ☑ Escrever tópico instalar
    - [⬅️]
3. 04/02/2021
    - ☑ Escrever tópico Referências
    - ☑ Atualizar o histórico deste documento.
    - ☑ Adicionar nos tópicos descrição e exemplos as tags sugeridas por TypeDoc:
        - ☑ @ignore
        - ☑ @category
        - ☑ @module
4. 05/02/2021
    - ☑ Registrar este documento em ./index.html
    - ☑ Melhorar a documentação dos tópicos:

        - [x] Objetivo
        - [x] Pre-requisitos
        - [x] Beneficios
    - [x] Escrever o resumo deste documento
    - [x] Atualizar o histórico deste documento.

5. 08/02/2021

    - [ ] Revisão deste documento para checar os erros de português.

    - []