BBjDocsGenerator User's Guide

BBjDocsGenerator Overview

In BBj 21.00 and higher, BBj includes a BBjDocsGenerator utility which replaces the older BBjToJavadoc utility. The BBjDocsGenerator parses documentation blocks in object-oriented BBj program text, and creates API documentation from it. The generated documentation is in HTML format. It supports a number of command-line parameters that permit each run to be configured differently, permitting customized output from processing one or more BBj source files.

Documentation Blocks in BBj Source Code

BBjDocsGenerator will process either single- or multi-line documentation blocks in a BBj program file.

- A single-line block contains a single line of BBj code that begins with "`REM /**`" that ends with "`*/`". The documentation block includes all of the text between the opening "`/**`" and the closing "`*/`".
- A multi-line block begins with a single line of BBj code that begins with "`REM /**`". It ends with the next line of the form "`REM <optional text> */`". The documentation block includes all of the text between the "`/**`" on the opening line and the "`*/`" on the closing line.

  Note: since BBj is case insensitive, the REM verb can be in any case (upper, lower, or mixed).

The BBjDocsGenerator identifies and processes description documentation blocks associated with:

- a package (at the BBj program file level)
- BBj classes (at the BBj Custom Object class level)
- BBj class fields (at the BBj Custom Object field level)
- BBj class methods (at the BBj Custom Object method level)

Tags in Description Documentation Blocks

BBjDocsGenerator will recognize and process some Javadoc-style tags that are present in documentation blocks. The following tags are supported:

- `@author` - defines one or more authors who have contributed to the item being described
- `@version` - defines the version identifier for the item being described
- `@param` - describes a parameter for a class method
- `@return` - describes the value being returned by a class method
- `@see` - points to additional documentation that might be helpful in understanding the item being described
- `@since` - defines the version or date in which the item being described was added
- `@deprecated` - marks the item being described as deprecated

In general, the text following one of these tags is "free form". The individuals writing the code and adding the tags can put whatever information they like there, so it is important that they understand where their text will appear, who will see it, and when.

BBjDocsGenerator also supports markdown features in the documentation blocks, and applies those markdown instructions when it generates the output documentation.

Examples of Documentation Blocks

Some examples of documentation blocks can be seen in this BBj program code:

```
rem /**
rem * <code>Class MyFactory</code> - My Factory Class
rem * This class works as well as can be expected...
rem * @author J. X. LastName
rem * @version 1.0
rem */
class public MyFactory
...
  REM /** BBjNumber A - Our first field */
  field private static BBjNumber A
...
  Rem /**
  rEm * Constructor MyFactory
  REM * @see <a href =
"https://documentation.basis.com/BASISHelp/WebHelp/bbjwindow.htm" target =
"_blank">BBjWindow</a> for more details
  reM * @author A. B. SomeOtherLastName
  REM */
  method public MyFactory()
...
  rem /**
  rem * Method createNewFrame:
  rem * Instantiate a wizard frame
  rem * @param BBjTopLevelWindow Wizard window object
  rem * @param HashMap Common data map
  rem * @return BBjNumber Frame number
  rem */
...
```

Figure 1. Examples of Documentation Blocks

Specifying a Package in BBj Source Code

Besides recognizing documentation blocks as the source of descriptions and details for the output documentation, BBjDocsGenerator also supports one special instruction: a package specifier.

Defining a package at the top of a BBj Source File causes the documentation generated from that file to be organized into a package (or "namespace") with that name. For example, placing a single line of BBj code that begins with "`REM PACKAGE <NAME>`" at the top of your BBj source file tells the BBjDocsGenerator to group all of the documentation entries from this file into a single package named "`<NAME>`". The "`<NAME>`" is white-space delimited (it is the first whole word encountered following "`REM PACKAGE`").

BBjDocsGenerator User's Guide

Note: since BBj is case insensitive, the "`REM PACKAGE`" text can be in any case (upper, lower, or mixed).

Examples of Package Tags

Some examples of package tags can be seen below (where each block of text represents the text at the top of a different BBj source file):

File #1:

```
rem /**
rem * Optional package comments
rem */
rem package main
```

File #2:

```
rem /**
rem * Optional package comments
rem */
rem package main.sub1
```

File #3:

```
rem /**
rem * Optional package comments
rem */
rem package main.sub2
```

Figure 2. Examples of Package Tags

Defining these packages at the tops of three different BBj Source Files will cause BBjDocsGenerator to generate its documentation in a similar folder structure. In this example, the "main" folder would be in the output folder, with "sub1" and "sub2" folders under it like this:

<output folder>

+-- main

+------ sub1

+------ sub2

Using this model, you can organize your documentation output in a variety of ways.

Executing BBjDocsGenerator from a BBj Program

To generate documentation files for each of a list of BBj source files, call one of the `BBjDocsGenerator.generateBBjdoc()` methods in a BBj program such as the example program shown in Figure 3.

```
use java.util.ArrayList
use com.basis.bbjutilities.bbjdocsgenerator.BBjDocsGenerator
use com.basis.bbjutilities.bbjdocsgenerator.BBjDocsParameters

REM Tell BBjDocsGenerator where to put the output files
outputDir! = "C:\BBjDocsGenerator\MyFileDocs\"

REM Create a list of the file (or files) to process
prog! = "C:\temp\MyFile.bbj"
declare ArrayList list!
list! = new ArrayList()
list!.add(prog!)
REM To process more than one file in a single execution, simply add more
REM   files to list! here

REM Instantiate the BBjDocsGenerator classes needed
declare BBjDocsGenerator generator!
declare BBjDocsParameters params!
generator! = new BBjDocsGenerator()
params! = new BBjDocsParameters()

REM Define optional parameters telling BBjDocsParameters how to run
params!.enableVerbose()

REM Tell BBjDocsGenerator to generate documentation for each file in
REM   list!, putting the output files in outputDir!, using the parameters
REM   in params! to control the output
generator!.generateBBjdoc(list!, outputDir!, params!)

end
```

Figure 3. A Simple BBj Program to Run the BBjDocsGenerator

In order to instantiate a `BBjDocsGenerator` and a `BBjDocsParameters`, BBj must be able to find definitions for those classes. These classes are defined in the BBjDocsGenerator.jar file that is installed in the `<bbj_home>/lib` folder beginning with BBj 20.30. For BBj to access that BBjDocsGenerator.jar file, you must first define a classpath that tells BBj where to find it. For this example, we defined a classpath named docs_generator in the Enterprise Manager as shown in Figure 4 .

BBjDocsGenerator User's Guide

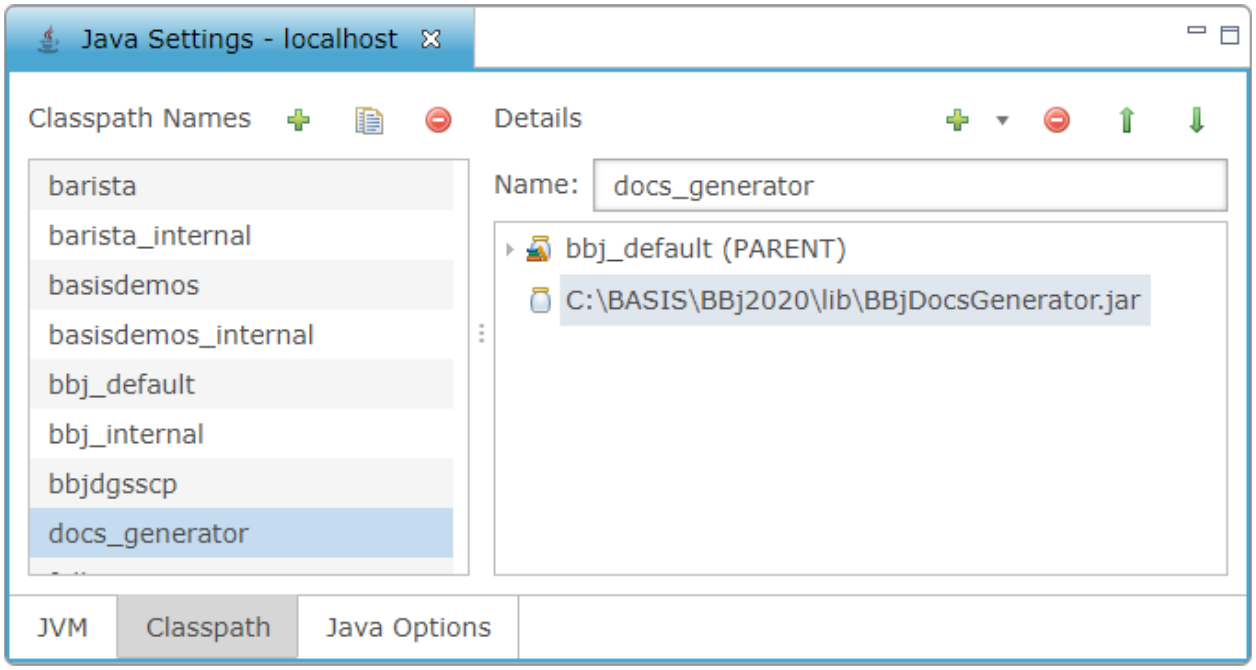

Figure 4. The docs_generator Classpath for the BBjDocsGenerator

When including the BBjDocsGenerator in a BBj program, you will need to use the "-CPdocs_generator" (classpath) argument to specify the classpath, as well as USE statements similar to those in Figure 3.

Parameters for BBjDocsGenerator.generateBBjDoc()

There are two forms of `BBjDocsGenerator.generateBBjdoc()` available, as shown in the table below. Both take an ArrayList of strings as the first parameter (the list of one or more BBj source files to process), and a string providing the full path to the folder to hold the output documentation files as the second parameter. The third parameter is optional: a `BBjDocsParameters` object with one or more parameters set to control how the file(s) are to be processed. The simple BBj program in Figure 3 uses a `BBjDocsParameters` variable, `params!`, for that purpose.

Return Value	Syntax
`void`	`generateBBjDoc(ArrayList<String> list!, String outputDir!)`
`void`	`generateBBjDoc(ArrayList<String> list!, String outputDir!, BBjDocsParameters params!)`

Parameter	Description
`ArrayList<String> list!`	The list of one or more BBj Source files to process
`String outputDir!`	The full path to the directory to hold the output documentation files
`BBjDocsParameters params!`	An optional parameter specifying a BBjDocsParameters object with one or more parameters set to control how the files are to be processed.

BBjDocsGenerator User's Guide

Standard Parameters of BBjDocsParameters

Table 1 below lists the standard `BBjDocsParameters` methods that are available, and the result on the generated documentation files of setting each one.

Parameter	Methods	Description

Author	`boolean isEnabledAuthorTag()`
	`void enableAuthorTag()`
	`void disableAuthorTag()`
		Controls whether the text following an @author tag is included in the generated documentation. By default, the text following an @author tag is NOT included.

Bottom Text	`boolean hasBottomText()`
	`void setBottomText(String)`
	`void clearBottomText()`
	`String getBottomText()`
		Controls writing text as the bottom line of each generated document. By default, there is no bottom text.

Document Title	`boolean hasDocumentTitle()`
	`void setDocumentTitle(String)`
	`void clearDocumentTitle()`
	`String getDocumentTitle()`
		Controls writing text as the title of each overview-summary file. By default, there is no title text.

Document Visibility	`boolean isIncludedPrivate Visibility()`
	`boolean isIncludedProtected Visibility()`
	`boolean isIncludedPublic Visibility()`
	`String getVisibility()`
	`void includePublicProtectedPrivate Visibility()`
	`void includePublicProtected Visibility()`
	`void includePublicVisibility()`
		Controls the highest level of visibility of items (classes and methods) that will appear in the output documentation. By default, only items with public visibility will appear in the output documentation.
		● Each `isIncluded…()` method returns true or false to report whether that specific visibility level will appear in the output or not.
		● `getVisibility()` returns one of three text strings representing the highest visibility level that will appear in the output. Items with lower visibility levels will also appear:
		○ "private" (the highest level)
		○ "protected"
		○ "public" (the lowest level)
		● Each `includePublic…()` method sets the visibility level(s) that will appear in the output. For clarity's sake, each method specifies all of the levels that will appear (be included).

Log Path	`boolean hasLogFilePath()`
	`void setLogFilePath(String)`
	`void clearLogFilePath()`
	`String getLogFilePath()`
		Controls the full path to the file where the log entries from document generation should be saved *if* the Verbose parameter is enabled. By default, no log file is created.

BBjDocsGenerator User's Guide

New Lines	`boolean isEnabledConverting Newlines()`
	`void enableConverting Newlines()`
	`void disableConverting Newlines()`
		Controls the conversion of newline characters ("\n") to "
		" tags. By default, there is no conversion.
		NOTE: Even if conversion is enabled, only newline characters ("\n") that are not followed by an HTML tag are converted.

Deprecated List	`boolean isEnabledCreating DeprecatedList()`
	`void enableCreating DeprecatedList()`
	`void disableCreating DeprecatedList()`
		Controls the creation of a list of deprecated items when generating the documentation. By default, creating a list of deprecated items is enabled (a list will be created).

Index Page	`boolean isEnabledCreating IndexTab()`
	`void enableCreatingIndexTab()`
	`void disableCreatingIndexTab()`
		Controls the creation of an Index tab in the generated output. By default, creating an index tab is enabled (an index tab will be created).

Since Tag	`boolean isEnabledSinceTag()`
	`void enableSinceTag()`
	`void disableSinceTag()`
		Controls whether the text following a @since tag is included in the generated documentation. By default, the text following a @since tag is included.

Timestamp	`boolean isEnabledTimestamp()`
	`void enableTimestamp()`
	`void disableTimestamp()`
		Controls the inclusion of a timestamp in the generated output files to indicate when they were generated. By default a timestamp is generated and written into a hidden HTML comment at the beginning of each page.

Tree	`boolean isEnabledCreating TreeOverview()`
	`void enableCreating TreeOverview()`
	`void disableCreating TreeOverview()`
		Controls the creation of a tree overview when generating the documentation. By default, creating a tree overview is enabled (a tree overview will be created).

Stylesheet Path	`boolean hasStylesheetPath()`
	`void setStylesheetPath(String)`
	`void clearStylesheetPath()`
	`String getStylesheetPath()`
		Controls the full path to a file containing the stylesheet for document generation. By default, no stylesheet file will be used, and the default stylesheet is used instead.

Top Text	`boolean hasTopText()`
	`void setTopText(String)`
	`void clearTopText()`
	`String getTopText()`
		Controls writing text as the top line of each generated document. By default, there is no top text.

Verbose	`boolean isVerbose()`
	`void enableVerbose()`
	`void disableVerbose()`
		Controls whether entries will be written to the log file during documentation generation. By default, verbose is disabled (no log entries are written to the log file). NOTE: even if verbose is enabled, log entries will only be written if a log file is also specified using the Log Path parameter.

Version	`boolean isEnabledVersionTag()`
	`void enableVersionTag()`
	`void disableVersionTag()`
		Controls whether the text following a @version tag is included in the generated documentation. By default, the text following a @version tag is NOT included.

Table 1. The Standard BBjDocsParameters Methods to Customize the BBjDocsGenerator

Advanced Parameters of BBjDocsParameters

Table 2 below lists the advanced `BBjDocsParameters` methods that are available, and provides a high-level description of the result on the generated documentation files of setting each one.

Parameter	Method	Description

Class Placeholder List	`HashMap getStandardClass PlaceHoldersList()`
	`void addClassPlaceHolder( String, String, String)`
	`void setClassPlaceHoldersList( String, HashMap)`
		Manage a list of standard placeholders (substitution tags) to apply when processing classes.

Class Template	`boolean hasClassTemplate()`
	`String getClassTemplate()`
	`void setClassTemplate(String)`
	`void clearClassTemplate()`
		Manage a custom template to be used to display items in the class section. The string passed in is a full path to the file containing the new template. The file is read and its content is stored in this attribute (the path is not stored).

Deprecated Placeholder List	`HashMap getDeprecated PlaceHoldersList()` `void addDeprecatedPlaceHolder( String, String)`	Manage a list of placeholders (substitution tags) to apply when processing the entries in the deprecated section.

Deprecated Template	`boolean hasDeprecatedTemplate()`
	`String getDeprecatedTemplate()`
	`void setDeprecatedTemplate( String)`
	`void clearDeprecatedTemplate()`
		Manage a custom template to be used to display items in the deprecated section. The string passed in is a full path to the file containing the new template. The file is read and its content is stored in this attribute (the path is not stored).

Index Placeholder List	`HashMap getIndex PlaceHoldersList()` `void addIndexPlaceHolder( String, String)`	Manage a list of placeholders (substitution tags) to apply when processing the entries in the index section.

Index Template	`boolean hasIndexTemplate()`
	`String getIndexTemplate()`
	`void setIndexTemplate(String)`
	`void clearIndexTemplate()`
		Manage a custom template to be used to display items in the index section. The string passed in is a full path to the file containing the new template. The file is read and its content is stored in this attribute (the path is not stored).

Overview Placeholder List	`HashMap getOverview PlaceHoldersList()` `void addOverviewPlaceHolder( String, String)`	Manage a list of placeholders (substitution tags) to apply when processing the entries in the overview section.

Overview Template	`boolean hasOverviewTemplate()`
	`String getOverviewTemplate()`
	`void setOverviewTemplate( String)`
	`void clearOverviewTemplate()`
		Manage a custom template to be used to display items in the overview section. The string passed in is a full path to the file containing the new template. The file is read and its content is stored in this attribute (the path is not stored).

Tree Placeholder List	`HashMap getTree PlaceHoldersList()` `void addTreePlaceHolder( String, String)`	Manage a list of placeholders (substitution tags) to apply when processing the entries in the tree section.

Tree Template	`boolean hasTreeTemplate()`
	`String getTreeTemplate()`
	`void setTreeTemplate(String)`
	`void clearTreeTemplate()`
		Manage a custom template to be used to display items in the tree section. The string passed in is a full path to the file containing the new template. The file is read and its content is stored in this attribute (the path is not stored).

Use Placeholder List	`HashMap getUse PlaceHoldersList()`
	`void addUsePlaceHolder( String, String, String)`
	`void setUsePlaceHoldersList( String, HashMap, String)`
		Manage a list of placeholders (substitution tags) to apply when processing the entries in the use section.

Use Template	`boolean hasUseTemplate()`
	`String getUseTemplate()`
	`void setUseTemplate(String)`
	`void clearUseTemplate()`
		Manage a custom template to be used to display items in the use section. The string passed in is a full path to the file containing the new template. The file is read and its content is stored in this attribute (the path is not stored).

Table 2. The Advanced BBjDocsParameters Methods to Customize the BBjDocsGenerator

You should not need to use these advanced methods, and thus a detailed explanation of how to use each method is not included here.

Output Files

After successfully running the BBjDocsGenerator, you will find that it created a collection of files in or under the folder you specified as the `outputDir!` argument to `generateBBjdoc(list!, outputDir!, params!)`. For example, if you processed exactly one BBj program file, `MyClass.bbj`, that contained a comment setting the package as `MyPackage` and exactly one class named `MyClass`, you would see the following subfolders and files in `outputDir!`:

- allclasses.html
- allclasses-frame.html

- deprecated-list.html
- help.html
- Index.html
- index-all.html
- overview-frame.html
- overview-summary.html
- overview-tree.html
- script.js
- stylesheet.css
- class-use (folder)
    - MyClass.html
- MyPackage (folder)
    - MyClass.html
    - package-frame.html
    - package-summary.html

Let's take a look at those folders and files, and see what is in each of them.

Folder and/or File	**Description**
allclasses.html	This file contains only a hyperlink-list of each of the classes in this documentation set, without a class description and without using any CSS template or formatting.
allclasses-frame.html	This file contains a hyperlink-list of each of the classes in this documentation set, without a class description. This page applies the class frame template, so its layout and formatting are based on the CSS and class template files.
deprecated-list.html	This file contains a list of the classes and methods in this documentation set that have the `@deprecated` tag. If there are none, this file merely states that. A deprecated class or method is not recommended for use, and if possible a suggested alternative is given. Deprecated items may be removed in future implementations.
help.html	This file offers explanatory text and other helpful information about BBjDocsGenerator and its logic.
index.html	This file lists all of the packages and class files resulting from BBjDocsGenerator processing this documentation set. It includes the output from overview-frame.html and overview-summary.html.
index-all.html	This file lists all of the class files resulting from BBjDocsGenerator processing this documentation set, grouped by first letter in increasing alphabetic order.
overview-frame.html	This file contains a list of all of the packages in this documentation set, and a link to all-classes.html.

BBjDocsGenerator User's Guide

overview-summary.html	This file contains a hyperlink-list of each of the packages in this documentation set, without any description or other details.
overview-tree.html	This file contains a hyperlink-list of each of the classes in this documentation set, organized hierarchically by package.
script.js	This file contains a small number of javascript functions used by the other output pages.
stylesheet.css	This file defines a cascading style sheet whose components are used by a number of the other output pages.
class-use/ (folder)	This folder holds one file per class, each listing "which other classes in this documentation set use this class". In this context, using a class refers to using the class type for a method argument, field, or return type.
class-use/MyClass.html	This file contains all of the information available about which other classes in this documentation set use the class "MyClass". In this case, there are no other classes, so this file merely states that. If there had been multiple classes, and one or more of those other classes had used the class MyClass as a type, this file would have identified those classes and provided a hyperlink to each.
MyPackage/ (folder)	This folder was created because of the package comment that identified the package as "MyPackage". It should contain one file per class in that package, each named after a class. **NOTE:** If there had been no package named MyPackage, there would be no MyPackage folder.
MyPackage/MyClass.html	Because class MyClass is defined in the BBj program file that also has the MyPackage package comment, we get a MyClass.html file that documents the class MyClass. Had there been other classes in MyPackage, each would have its own file named after that class. **NOTE:** If there had been no package comment, then all of these <class>.html files would instead be directly in the requested `outputDir!`.
MyPackage/package-frame.html	This file contains all of the information available about the package MyPackage, including a hyperlink and a description for each class in MyPackage. This page applies the class frame template, so its layout and formatting are based on the CSS and class template files.
MyPackage/package-summary.html	This file contains only a hyperlink-list of each of the classes in MyPackage, without a class description and without the CSS template or formatting.

BBjDocsGenerator User's Guide

Summary

BBjDocsGenerator creates an array of output files that together provide documentation that is suitable for your BBj Application Programming Interface (API). It relies upon the BBj program files containing a number of tags with accompanying description text to define the documentation it generates. The BBjDocsGenerator was created based on the Java concept of Javadoc API documentation; in the near future the BBjDocsGenerator should be tailored more for BBj programs and details than for Java. There will also be an Eclipse plug-in for the BBjDocsGenerator in the near future, which will provide a more user-friendly interface than the command-line UI outlined in this document.