CHAPTER 4

Test Class and Case Comment Blocks

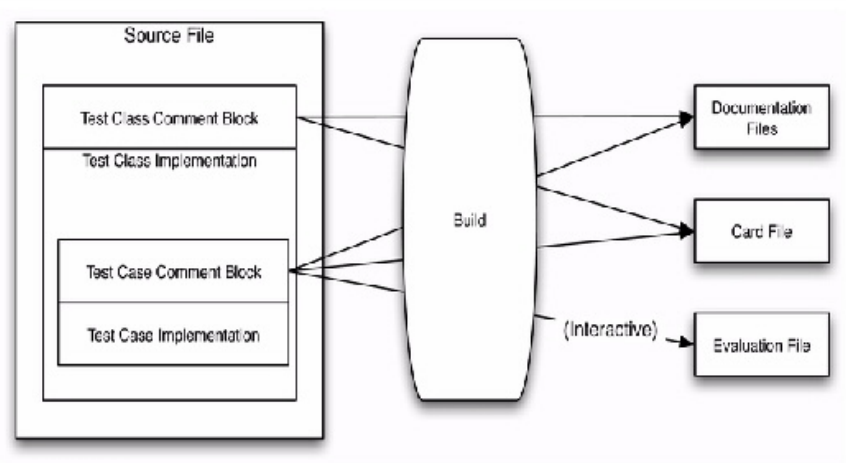
You must precede each test class and each test case with a descriptive comment block. These blocks identify the classes and methods that represent test classes and cases. They also give the build system and the harness information required to build and bundle tests and to display configurable properties to testers. The comment blocks are similar to Javadoc tool "doc comments" described at http://java.sun.com/j2se/javadoc/writingdoccomments/

This chapter has the following sections:

Comment Block Overview

When you build a test pack, the build system generates the files shown in FIGURE 4-1 from comment blocks you code for each test class and test case.

FIGURE 4-1 Files Generated from Comment Blocks



The default build generates the following files:

- Documentation files - HTML files created by the Javadoc tool. When a tester clicks the Documentation tab, the graphical harness displays these files. There are additional test pack and package documentation files which you create separately as described in Chapter 5.
- Card file - Test class and case information used by the harness, such as property definitions and a list of related files that test cases require to run. The word "card" has no significance.

  You can alternatively create a card file manually, as described in Chapter 25. You can turn off the autogeneration of card files with a build option.

  You can verify the contents of a card file using the tool described in Chapter 12.
- Evaluation File - For interactive tests only, an HTML file that contains tester instructions for running and evaluating the test. You can alternatively create an evaluation file manually, as described in Chapter 27.

CODE EXAMPLE 4-1 shows a typical test case comment block.

CODE EXAMPLE 4-1 Typical Test Case Block Comment

```
/**
   * Interactive test that asks user to install more than one test MIDlet
   * suite associated with this test case. In order to do that the developer needs
   * to define the <code>JADPath&lt;N&gt;</code> and
   * <code>JARPath&lt;N&gt;</code> properties, where
   * <code>N</code> is decimal integer starting from 1 to infinity (without gaps).
   * For instance, if you need a testcase to install 3 MIDlet suites, use N=1,2, and 3.
   * Each property contains the pointer to the appropriate test MIDlet.
   *
   * @testcase
   *
   * @precondition none
   *
   * @userInteraction
   * <ul>
   * <li>Install <code>syspropviewer.jad</code> using the
   *    provided URL</li>
   * <li>Select installed MIDlet and launch it</li>
   * <li>Install <code>choicegroup.jad</code> using the
   *    provided URL</li>
   * <li>Select installed MIDlet and launch it.</li>
   * </ul>
   * @postcondition The 2 MIDlet suites are installed
   * @passCriteria User is able to install both of the MIDlet suites and
   *    launch them. The "SysPropertiesViewer" MIDlet should show a list of
   *    system properties describing your device. "ChoiceGroupDemo" application
   *    should display different types of choice group UI.
   *
   * @card.property JADPath1=syspropviewer/syspropviewer.jad
   * @card.property JARPath1=syspropviewer/syspropviewer.jar
   * @card.property JADPath2=choicegroup/choicegroup.jad
   * @card.property JARPath2=choicegroup/choicegroup.jar
   */
void t02InstallMoreThanOneMIDletSuite();
```

FIGURE 4-2 and FIGURE 4-3 show the (rendered) test case documentation and evaluation file generated from the block comment shown in CODE EXAMPLE 4-1.

FIGURE 4-2 Typical Generated Test Case Documentation

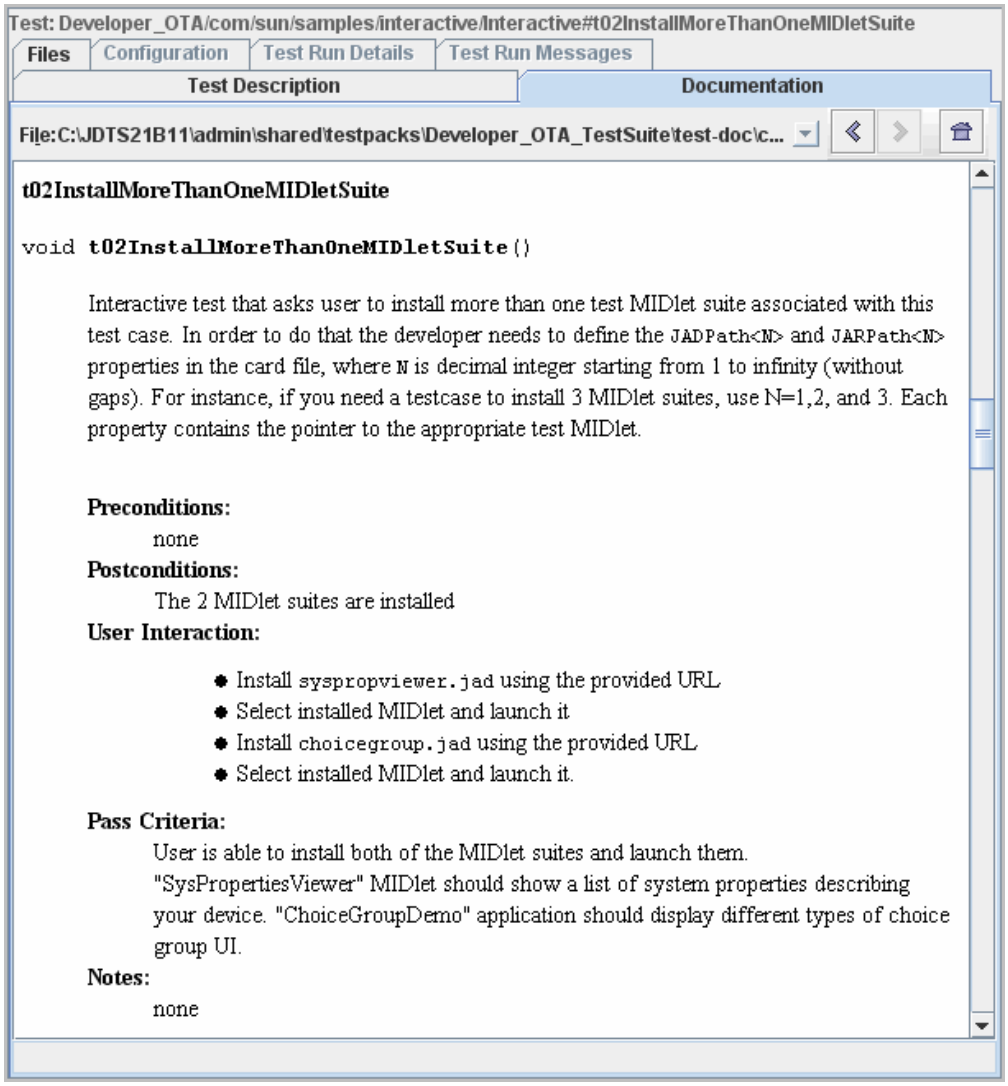
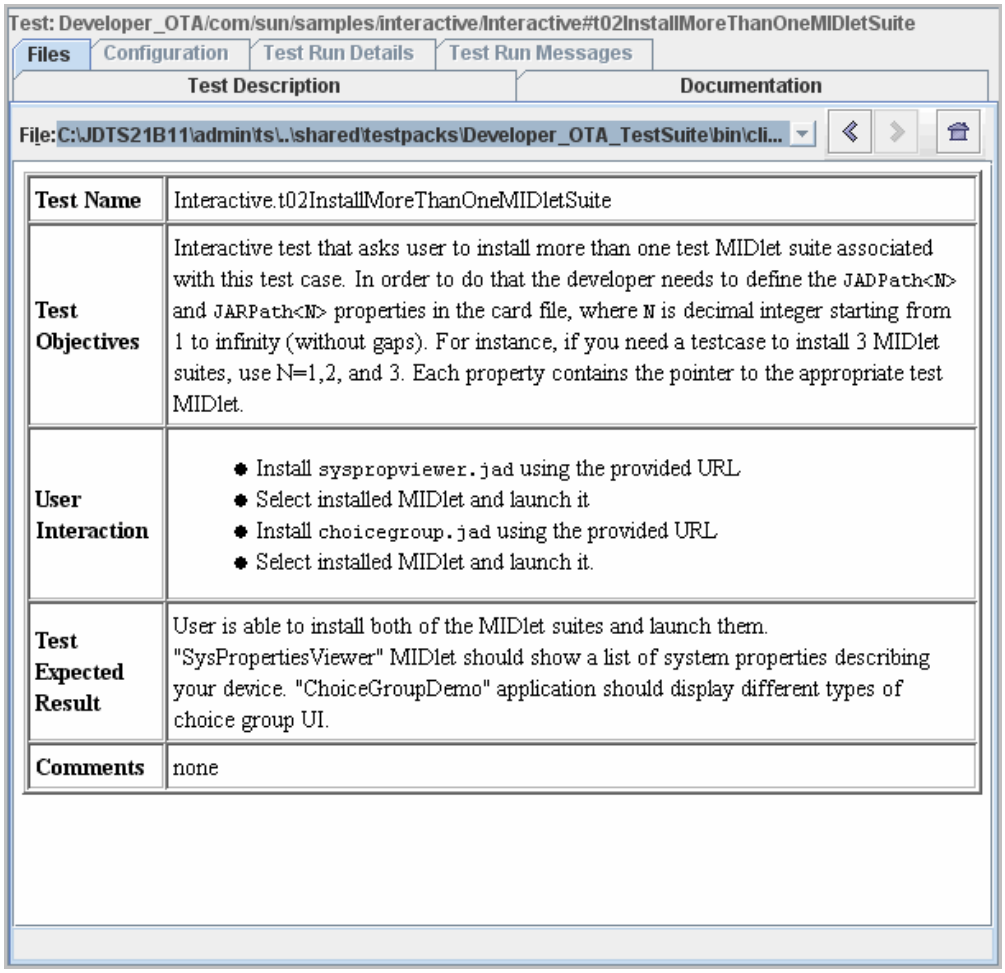


FIGURE 4-3 Typical Generated Interactive Test Evaluation Instructions



Although there is some flexibility in formatting a comment block, observe the following rules:

- Each block begins with a line containing only `/**` and ends with a line containing only `*/`. The class or test case declaration associated with the comment block follows the `*/`.
- Subsections of a comment block are introduced by tags, which have the form `@`*tagName*. The supported tags are defined in TABLE 4-1 and TABLE 4-2. You can also include standard Javadoc tool tags, such as `@since, @throws`. You can use `{@inheritDoc}` in JavaDoc tool tags. Case is significant in tag names. Write `@userInteraction` rather than `@userinteraction`.
- The untagged lines preceding the first tag introduce the class or case. In Javadoc tool terms, these lines are the “main description”
- Each block must have an `@testcase` or an `@testclass` tag.
- If the same tag appears in both the test class comment block and a test case comment block, the precedence rules described in Chapter 16 apply.
- All tags except `@card.*`, `@testcase,` and `@testclass` can be followed by lines of simple (version 4.0.1) HTML to create formatted output such as bullet lists. The lines following `@userInteraction` in CODE EXAMPLE 4-1 give an example.
- `@card.*` tags have these rules:
  - They cannot contain HTML.
  - Lines terminated by a `\` (backslash) character continue on the next line. On the continuation line, the `*` (asterisk) character is ignored. For example:

    ```
    * @card.property x=one \
    *two
    ```

    The value of `x` is `one two`.

- You can use Java programming language properties file escape sequences. For example: `\\` (backslash), `\u`*HHHH* (Unicode character with hexadecimal value *HHHH*). However, do not use `\n` (line break) in the value of a property. The line break character might cause unexpected behavior. For property file details, see http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html#load(java.io.InputStream)

Test Class Comment Block Tags

TABLE 4-1 describes the Java Device Test Suite tags that can or must appear in a test class comment block. The Instances column specifies how many instances of a tag are permitted.

TABLE 4-1 Test Class Block Comment Entries

Entry	Test Types	Required?	Instances	Description
`/**` `*` *htmlLines*	All	Yes	1	*htmlLines* is the class description in the generated test documentation. In Javadoc tool terms, this is the "main description".
`* @testclass`	All	For test classes	1	Identifies next class definition as a test class.
`* @assumption` *htmlLines*	All	No	0, 1	A condition that is assumed to be true, but cannot be changed, and, in some cases, even checked by the tester. A typical assumption is an interpretation of a related specification statement. If an assumption does not appear to be true, the test should be considered not applicable to the device, which typically means that the test fails if it is run. *htmlLines* is the content of the corresponding section of the generated test documentation and applies to all cases in the class.
`* @precondition` *htmlLines*	All	No	0, 1	A condition that must be true just prior to the execution of the test. If the precondition is not met, the test may give a wrong result. The precondition statement is checkable and achievable: There should be a way to change the environment and guarantee the precondition to be fulfilled. This is typically the tester's responsibility to ensure that the test precondition is fulfilled. Typical precondition is a reminder about some necessary configuration settings. *htmlLines* is the content of the corresponding section of the generated test documentation and applies to all cases in the class.
`* @postcondition` *htmlLines*	All	No	0, 1	A condition that must be true immediately after successful execution of a test case. "Successful" means no error, unexpected exception, or VM_EXIT. *htmlLines* is the content of the corresponding section of generated test documentation and applies to all cases in class.

Entry	Test Types	Required?	Instances	Description
`* @keyword` *keywordList*	All	No	0, *n*	Keyword(s) that testers can use to filter (subset for execution) this test case or test class. *keywordList* is a space-separated list of keywords. Interactive tests ***must*** have this entry: `@keyword interactive`. To see the current list of keywords and their definitions, launch the harness and create or open a work directory. Choose Configure > Edit Configuration or Configure > New Configuration. In the Test Selection section of the interview, answer Yes to Specify Keywords? The More Info pane displays the current list of keywords.
`* @notes` *htmlLines*	All	No	0, 1	Notes that provide test class information to the tester. Quoting from a specification is one use of this tag. If a note is not necessary, do not use the tag.
`* @card.property` *value*	All	No	0, *n*	Defines a property for all test cases in the class. See @card.property for details.
`* @card.specialproperty` *value*	All except OTA	No	0, *n*	Adds a line to the JAD file in a test bundle that contains this test. See @card.specialproperty for details.
`* @card.requires [`*value*`] [*]`	All	No	0, *n*	Ensures that a test bundle that contains this test also contains all files the test needs to run. See @card.requires for details.
`* @card.attribute` *value*	All	No	0, 2	Defines the factors that determine the severity of a test case failure. See @card.attribute for details. Two of these comments are required (at the case or class level) if you want to give testers the ability to select tests and report results by failure severity, as they can with Sun tests. The *Java Device Test Suite Tester's Guide* describes the severity concept.

Test Case Comment Block Tags

TABLE 4-2 describes the Java Device Test Suite tags that can or must appear in a test case comment block. The Instances column specifies how many instances of a tag are permitted.

TABLE 4-2 Test Case Block Comment Entries

Entry	Test Types	Required?	Instances	Description
/** * *htmlLines*	All	Yes	1	*htmlLines* is the class description in the generated test documentation. For interactive tests, *value* is the content of Test Objectives in the evaluation file. In Javadoc tool terms, this is the “main description”.
* `@testcase`	All	For test cases	1	Identifies next method definition as a test case.
* `@assumption` *htmlLines*	All	No	0, 1	A condition that is assumed to be true, but cannot be changed and, in some cases, even checked by the tester. A typical assumption is an interpretation of a related specification statement. If an assumption does not appear to be true, the test should be considered not applicable to the device, which typically means that the test fails if it is run. *htmlLines* is the content of the corresponding section of the generated test documentation.
* `@precondition` *htmlLines*	All	No	0, 1	A condition that must be true just prior to the execution of the test. If the precondition is not met, the test may give a wrong result. The precondition statement is checkable and achievable: there should be a way to change the environment and guarantee the precondition to be fulfilled. It is typically the tester's responsibility to ensure that the test precondition is fulfilled. A typical precondition is a reminder about some necessary configuration settings. *htmlLines* is the content of the corresponding section of the generated test case documentation.
* `@postcondition` *htmlLines*	All	No	0, 1	A condition that must be true immediately after successful execution of a test case. “Successful” means no error, no unexpected exception, and no VM_EXIT. *htmlLines* is the content of the corresponding section of generated test documentation.
* `@passCriteria` *htmlLines*	Runtime, OTA	For interactive	0, 1	Condition or conditions that must be met for the test to pass. For interactive tests, *htmlLines* is the content of Test Expected Result section of the test evaluation instructions.
* `@userInteraction` *htmlLines*	Runtime and OTA interactive	For interactive	0, 1	Instructions for tester to execute an interactive test. *htmlLines* is the content of the User Interaction section of the test evaluation instructions.

Entry	Test Types	Required?	Instances	Description
`* @referenceImages` *URLs*	Runtime and OTA interactive	No	0, 1	*URLs* is URL(s) of image(s) displayed in evaluation instructions.
`* @performanceMetric` *value*	Benchmark	For benchmark	0, 1	Use instead of `@passCriteria`. *value* must be one of: `UnitRate`, `SystemLoad`.
`* @keyword` *keywordList*	All	No	0, *n*	Keywords testers can use to filter (subset for execution) this test case or test class. *keywordList* is a space-separated list of keywords. Interactive tests ***must*** have `@keyword interactive`. To see the current list of keywords and their definitions, launch the harness and create or open a work directory. Choose Configure > Edit Configuration or Configure > New Configuration. In the Test Selection section of the interview, answer Yes to Specify Keywords? The More Info pane displays the current list of keywords.
`* @notes` *htmlLines*	All	No	0, 1	Notes that provide test case information to the tester. For interactive tests, *htmlLines* is the content of the Comments section of the evaluation file. Quoting a specification or describing unusual test behavior ("On some devices, the screen blinks") are typical uses of notes. If a note is not necessary, do not use the tag.
`* @card.property` *value*	All	No	0, *n*	Defines a test case property. See @card.property for details.
`* @card.requires [`*value*`] [*]`	All	No	0, *n*	Ensures that a test bundle that contains this test also contains all files the test needs to run. See @card.requires for details.
`* @card.attribute` *value*	All	No	0, 2	Defines the factors that determine the severity of a test case failure. See @card.attribute for details. Two of these comments are required (at the case or class level) if you want to give testers the ability to select tests and report results by failure severity, as they can with Sun tests. The *Java Device Test Suite Tester's Guide* describes the severity concept.

For tags that can be used with both test classes and test cases, use the class tag to specify values that apply to all cases in a class. Use the case tag to specify values that apply to one case only. For example, suppose you specify the following:

```
* @testclass
* @keyword interactive
* ...
* @testcase
* @keyword onePartnerMIDlet
```

This means that all test cases in the class can be filtered with the `interactive` keyword but only the specified test case can also be filtered with `onePartnerMIDlet`.

Tag Details

Tags with more complex syntax or semantics are described in this section.

`@card.property`

If a test class or case needs a user-visible property, name the property and specify its default value with the following syntax:

```
* @card.property PropertyName=DefaultValue
```

For example:

```
* @card.property MaxDistance=12
```

Properties that apply to multiple classes can be defined globally at the test pack level. Such properties are defined in the `testsuite.info` file at the top of the test pack directory in the file system. A property defined at the test pack level can be overridden by defining a property of the same name within the test class or case comment blocks. See Properties and Parameter Expansion for details.

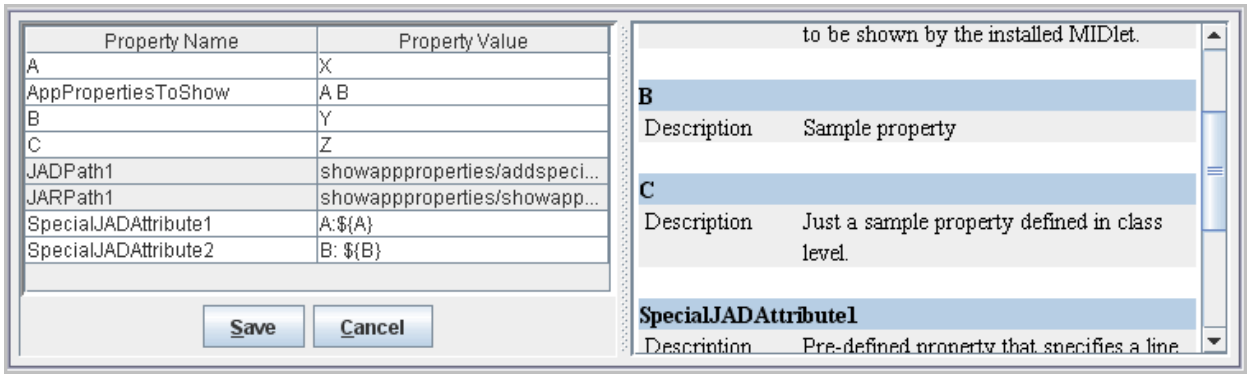
The following test pack property attributes are also supported for test class and case properties.

- `scope`: To hide a property from users, specify the value `hidden`. The `advanced` value is not supported for classes and cases.
- `readonly`: To prevent user modification of the property's default value, specify the value `true`.
- `doc`: Give a short description of the property, for example:

```
* @card.property PixelHeightNoLessThan.doc=minimum number of pixels allowed for object height
```

FIGURE 4-4 shows an example of `doc` values that appear when a tester right-clicks a test case in the harness test tree and chooses Configure Test. Read-only properties are displayed in gray (for example, JADPath1 in FIGURE 4-4).

FIGURE 4-4 `doc` **Properties in Configure Test Window**



For the exact syntax of property attributes, refer to Scope, Read-only Properties, and Online Documentation for a Property

`@card.specialproperty`

Use a special property definition to direct the harness to add a line to the JAD file that it creates for a test bundle containing this test. In effect, a special property definition is a way to pass a parameter to an application management system (AMS), sometimes called a Java

application manager. The static parameter applies only to the test bundle associated with the JAD file.

Use the following syntax to specify a line to be added to the JAD file:

`* @card.specialproperty <jad>.`*n*`=`*LineToAdd*

n is a number that distinguishes multiple `@card.specialproperty` entries in the same source file. The harness interprets *LineToAdd* in this line as a name:value pair taking the first colon symbol as the separator between the name and the value. The harness copies the name:value pair to the JAD file. The test device AMS interprets the name:value pair when it downloads the JAD file. It is the test developer's responsibility to ensure that name:value pair is meaningful to the AMS and follows the JAD file syntax. The JAD file is defined in the MIDP specification.

For example, to add a line to a JAD file associated with a test bundle containing `MIDlet2:AlarmMidlet`, include a line like this in the test class block comment:

```
* @card.specialproperty <jad>.1=MIDlet2:AlarmMidlet
```

Parameter expansion, described in Properties and Parameter Expansion, also applies to special property definitions. The samples in *devKitRoot*`/tests/runtime/src/client/com/sun/samples/network/client/push/` illustrate its use.

`@card.requires`

This tag specifies files that must be in the test bundle for one or more test cases in a class to execute. Examples of required files include media files to play on the test device and helper classes. CODE EXAMPLE 4-2 gives examples for a class and a case:

CODE EXAMPLE 4-2 Example Required File Values

```
* @testclass
* ...
* @card.requires Pic1.jpg
* @card.requires com/some/apackage/img/Pic2.jpg
* @card.requires *
* @card.requires com/some/apackage/Aclass.class *

* @testcase
* ...
* @card.requires Pic3.jpg
```

Observe the following when writing path names for this tag:

- The path name separator must be a forward slash (`/`) character.
- If the path name contains a separator (for example, `com/some/Foo.class` or `img/Pic.png`), the path is relative to the directory specified in the test pack property `TestClassesDir`. For example, for runtime test packs, this directory is typically `bin/client/verified` (relative to the test pack root).
- If the path name does not contain a separator (as in the `Pic1.jpg` line in CODE EXAMPLE 4-2), the file is in the same directory as the class of the file containing the `@card.requires` tag.
- The special file name of `*` directs the build to automatically find all class files required by the class containing the `card.requires` comment.
- Following the name of a required class, the `*` character means "include this file and find the files required by the required file".
- You can specify class name explicitly as when a class is loaded using `Class#forName(String)`, for example.
- If a test class refers to a constant field defined in another class, the class that defines the constant may not be found by the card file generator. The reason is "inlining" constant values by `javac` compiler. The test case `testCase1` in *devKitHome*`/tests/runtime/src/com/sun/samples/Automated/SampleAutomatedTest.java` demonstrates

this effect. See the comments in the classes `IInlinedConstants.java` and `IReferencedConstants.java` for more detail.

- Automatic searching applies to Java programming language `.class` files only. Do not specify an asterisk (`*`) with non-class resources, such as images. For example, the following is an error:

```
* @card.requires Pic1.png *
```

The path in an `@card.requires` entry can specify parameter expansion notationas described in Properties and Parameter Expansion. CODE EXAMPLE 4-3 shows an example.

CODE EXAMPLE 4-3 Parameter Expansion in `@card.requires` Example

```
* @card.property req=a
* ...
* @card.requires com/some/samples/automated/${req}.html
```

In this example, changing the value of `req` to `b` will include the file `com/some/samples/automated/b.html` in test bundles.

Note - Properties are not inherited. In particular, required file properties declared in class `X` do not apply to `X`'s superclass or subclasses. Be sure to declare the files that each class requires.

`@card.attribute`

Each test class and case can have a failure severity indicator, which is computed from two factors, ***functionality*** and ***impact***. TABLE 4-3 shows the three functionality and impact codes and the failure severity values that the Java Device Test Suite computes from them. The *Java Device Test Suite Tester's Guide* gives more information on failure severity.

TABLE 4-3 Severity Calculation from Functionality and Impact

	Impact		
Functionality	**1 - Critical**	**2 - Significant**	**3 - Limited**
1 - Primary	1 - Very High	2 - High	3 - Medium
2 - Secondary	2 - High	3 - Medium	4 - Low
3 - Nonessential	3 - Medium	4 - Low	5 - Very Low

Use the following guidelines to select a functionality value:

- 1 - Primary: The associated Test Objectives are a mandatory requirement of the implementation function tested by the test.

- 2 - Secondary: Indicates a recommended practice of the implementation function in accordance with Test Objectives. There may be valid reasons in particular circumstances to ignore this recommendation of the implementation function.
- 3 - Nonessential: Indicates that the Test Objectives are an optional requirement of the implementation function tested by the test.

Use the following guidelines to select an impact value:

- 1 - Critical: The tested device is effectively unusable as a result of the test failure. The test failure causes critical impact on the operation of the device/implementation. Critical impact should cover the case when implementation does not work. The test failure renders the implementation ineffective.
- 2 - Significant: Device failure causes significant impact. A test failure identifies a serious but predictable and manageable device failure.
- 3 - Limited: Device failure causes only limited or insignificant impact on the behavior and performance of the device/implementation.

You express functionality and impact in `@card.attribute` tags of the following form:

```
* @card.attribute functionality=value
* @card.attribute impact=value
```

In both cases, *value* must be 1, 2, or 3. CODE EXAMPLE 4-4 shows an example. TABLE 4-3 shows that this case's computed failure severity is 4 - Very Low.

CODE EXAMPLE 4-4 Example Severity Attributes

```
* @testcase
* ...
* @card.attribute functionality=2
* @card.attribute impact=3
```