HCTS Test Developers Guide

1.0 Introduction

The Hardware Certification Test Suite (HCTS) has been designed to accommodate individual Test Cases provided by third-party developers. The goal of the Test Developers Guide is to provide a comprehensive specification and step-by-step walkthrough of the Test Case integration process. Two primary components are described: How to design a Test Case descriptor file and how to integrate Test Cases into the Controller Test User Interface (UI).

1.1 Glossary

XTF	Extensible Test Framework: The XTF is responsible for managing the scheduling and execution of Test Cases as well as providing a flexible API, capable of supporting multiple UIs.
HCTS	Hardware Certification Test Suite: HCTS was designed to certify hardware compatibility for usage with the Solaris Operating System. As a secondary consideration, HCTS incorporates a number of Test Cases useful for general hardware testing.
Scheduler	An element of the XTF responsible for controlling the timing and execution of all aspects of individual and grouped Test Cases.
Test Case Program	The core program responsible for conducting the testing operations of an individual Test Case. All other associated descriptors and subprograms are designed to facilitate the successful execution of a Test Case Program.
Test Case	The set of components used to define and control the complete execution cycle of a Test Case Program. Components are wrapped in descriptor files to provide a standard interface such that the XTF can define an appropriate UI and manage the execution of the Test Case when initialized.
Test Case Group	A collection of one or more Test Cases selected to be initialized simultaneously and executed in parallel.
Test Category	A collection of one or more Test Case Groups selected to be initialized and executed one after the other. Typically a Test Category specifies a complete layout of the tests required to Certify or test a type of hardware.
UI	User Interface: The interface through which all selections and interactions exposed to a user may be manipulated.

XML	Extensible Markup Language: XML descriptor files are the primary focus of the Test Developer's Guide. These files present a standardized interface which can be examined and utilized by the XTF.
Hook	A subprogram executed as part of a Test Case or Test Category running cycle for the purpose of assisting the proper execution of the complete testing progress.

1.2 Overview

Utilizing the various interface options provided by the XML Test Descriptor standard provides the opportunity for a Test Case to interact with the user, maintain a controlled test environment and abstract the process of inserting a Test Case into any HCTS UIs. In order to successfully integrate a Test Case into HCTS, a developer must complete the following procedures:

Create a Test Case Descriptor File	This file is used to specify all user interactions, environmental control procedures and descriptive elements.
Create a Test Category Descriptor File	This file defines a test category. One or more Test Cases may be associated with the test category. Through the creation of a Test Category Descriptor File a developer defines a testing procedure which will be accessible in the “Controller Test” portion of any HCTS UI.

2.0 Message Passing System

Any message designated to be displayed by a UI must be passed to the XTF using hcts_msg. Every program specified by a Test Category Descriptor File, as well as the run Hook program detailed by the Test Case Descriptor File, utilize a version of hcts_msg which references error messages defined in an XML file. Any message can be passed to the XTF, which in turn will be displayed in a UI, by outputting a new line to standard output conforming to the following format:

hcts_msg[messageNumber, messageType, messageArguments]

Note: As denoted by the format displayed above, each of the three message arguments are separated by a "," (Comma) character. If either the "," (Comma) or "\" (Backslash) characters are used as part of any of the argument values the characters must be escaped using a "\" (Backslash) character. For example, the statement, 'type=<component\device>,risk=high' would be properly escaped as, 'type=<component\\device>\,risk=high'.

The messageNumber argument must correspond to one of the entry keys defined in a message file. The following example details the expected format of an XML message file:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "http://java.sun.com/dtd/properties.dtd">

<properties>
    <entry key="user_stop">User @arg1 stop the test.</entry>
    <entry key="terminate">Test is terminated.</entry>
    <entry key="timeout">Test timeout!</entry>
    <entry key="error_font">Unable to resolve font name.</entry>
    <entry key="running">@arg1 is running the Hello World test.</entry>
</properties>
```

The messageType argument must be one of three accepted values:

Value: 1	Indicates a message of type ERROR
Value: 2	Indicates a message of type WARNING
Value: 3	Indicates a message of type INFO

The following is an example message output based on the message file defined above:

hcts_msg[running,3,Jimmy]

When the preceding line is output by a Hook program, the UI will display the message: 'INFO: Jimmy is running the Hello World test.'

3.0 How to Create a Test Case Descriptor File for a Test Case Program

An example will be used to illustrate the format and usage of the various components of a Test Case Descriptor File. The Test Case labelled 'hello_world' will be generated piece by piece and then assembled into to a complete Descriptor File example at the end of this section. The core of the 'hello_world' Test Case is the program named 'hello_world.sh' which simply sleeps for several seconds and then outputs “Hello World!”.

3.1 Test Case Program

The Test Case Program is the set of instructions responsible for enacting a testing procedure. All other components described in this document are utilized to describe and control the interactions of the Test Case Program. The HCTS XTF imposes the requirements listed below on all Test Case Programs. If a developer wishes to insert a Test Case Program which has already been written they may need to modify the program to conform to these requirements.

- *Return a '0' value to indicate the testing procedure has passed and a non-zero value to indicated a testing failure. If the Test Case has been generated for non-testing purposes (ex. Assistant Test Case, System Stress Generator), simply return '0' in all*

cases.

- *Trap signal 17 to clean up.*
- *A Test Case Program is not permitted to request user interaction while the program is executing.*

Any output messaging is expected to be directed to test case log files and Test Case report files (Detailed in sections 2.0, 3.2.4 and 4.1.5).

The following simple program, saved as 'hello_world.sh', implements the Test Case functionality which is to added to HCTS. Both a Test Case Descriptor File and Test Category Descriptor File must be created to incorporate our example Test Case:

```
#! /bin/sh
clean_up()
{
    echo hcts_msg[terminate,1]
    echo "Hello Test Ending!"
}
# set trap
trap 'clean_up; exit 1' 17
sleep 30
echo 'Hello World!'
```

3.2 Creating a Test Case Descriptor File

The name of the Test Case Descriptor File should be the same as the name attribute of the Test Case. This file specifies which Test Case Program to execute and description information. The postfix of the filename must be the same as the value of “xtf.test.postfix” variable defined in the configuration file. The “xtf.test.postfix” variable is defined in the file 'xtf.conf', located in the '/opt/SUNWhcts/etc/' directory. Typically this postfix value will be ".xml".

3.2.1 Test Case Element (Required Element)

The Test Case element is the root element of the Test Case Descriptor File.

name (Required Attribute)

The name of the Test Case. The name must be unique in the scope of all installed Test Cases and the same as the Test Case Descriptor File (not include the postfix).

basedir (Optional Attribute)

The path specified by this attribute is relative to the Test Case installation directory (i.e. /opt/SUNWhcts/testcases/). All relative paths specified by executable elements in the Test Case Descriptor File will utilize this path as a base directory.

For our Hello World example, the 'hello_world.xml' Test Case Descriptor File will be placed in the /opt/SUNWhcts/testcases/ directory and the hook programs will be placed in the /opt/SUNWhcts/testcases/example/ directory. We open a new file and start the Test Case Descriptor File as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase name="hello_world" basedir="example">
```

3.2.2 Attribute Elements

Attribute Elements describe a set of informational data to be passed to the HCTS XTF. Each attribute must specify a name and value element.

name

Specified the name of the attribute.

value

Specifies the value of the attribute.

The following attributes are reserved:

name: version (Optional Attribute)	**value**: The version of the Test Case.
name: short-description (Optional Attribute)	**value**: Provide a brief description of the Test Case.
name: long-description (Optional Attribute)	**value:** Provide a detailed description of the Test Case. Typically, this description will describe the runtime, usage considerations, detailed purpose and any other instructive statements.
name: time (Required Attribute)	**value:**Indicate the maximum runtime permitted by the Test Case. If the Test Case does not complete execution within the specified time the XTF will kill the Test Case Program execution. The specification of this value ensures an error will not cause the Test Case to hang. Specify the value using the following format: hours:minutes:seconds. For example, 1:5:30 indicates a timeout value of 1 hour, 5 minutes and 30 seconds.
name : assist (Optional Attribute)	**value**: Set the value of this attribute to 'true' to indicate the associated Test Case is being used to facilitate a purpose other than testing (i.e. The Test Case is an assistant). If the value is set to 'true' the execution of the Test Case will be stopped if all other Test Cases executing in parallel with this Test Case have failed.
name : instance (Optional Attribute)	**value**: Indicates the number of Test Case program instances to be executed by the XTF. If this element is not specified the default value will be set to '1' (one) instance.

Now we append the following attribute elements to describe our Test Case Descriptor File. Note that only the attributes relevant to this Test Case are specified.

```
<attribute>
   <name>version</name>
   <value>1.0</value>
</attribute>
<attribute>
   <name>short-description</name>
   <value>Hello World!</value>
</attribute>
<attribute>
   <name>long-description</name>
   <value>This is the basic Hello World introductory example.</value>
</attribute>
<attribute>
   <name>time</name>
   <value>0:2:0</value>
</attribute>
<attribute>
   <name>author</name>
   <value>James</value>
</attribute>
```

3.2.3 Argument Elements

Argument Elements are used to augment the operation of the Test Case program as well as any associated assistant programs (i.e. Hooks). Utilize the various sub-elements to specify the operation of the arguments. Any number of arguments are permitted.

Important Note: The order of Argument Elements in the Test Case Descriptor File directly correlates the order in which the arguments will passed to the specified programs.

name (Required Element)

The name of the argument.

type (Optional Element)

The type is used to define arguments which will be passed directly to a program when the program is executed. Any argument element which does not have a defined type element will be set to env (Environment variable) by default.

Command Line Parameter Arguments

name: env	**value**: This argument will be set as a environment variable.
name: fixed	**value**: The value of this argument will be added after the command.
name: nsv	**value**: The "name value" will be added after the command.
name: nev	**value**: The "name=value" will be added after the command.

scope (Optional Element)

The scope of an argument defines which program or programs will receive the specified argument. If the scope element is not defined, the argument will be passed to all associated programs (i.e. The Test Case Program and other Hook Programs). The only permissible value of a scope element is 'all' or the name of a hook program defined in the Test Case Descriptor File. Specifying a specific hook program means only that hook program will receive the argument.

In the following example of an nsv type argument the scope has been restricted to the run Hook program:

```
<argument>
        <name>User</name>
        <type>nsv</type>
        <value>root</value>
        <scope>run</scope>
</argument>
```

Due to the restriction imposed by the scope element in this example only the Test Case program will receive this argument as part of the command line execution of the hello_world.sh program. The Test Case program would be executed using the following command:

hello_world.sh User root

default-value (Optional Element)

The default-value argument provides the flexibility to specify standard default values for an argument thus providing a central location to modify all such values. Default values have the lowest priority of any value assignment. A value modification to the argument made at any level will be used instead of the default value.

The following example demonstrates the specification of a default-value for an argument of type nev:

```
<argument>
        <name>User</name>
        <type>nev</type>
        <default-value>newUser</default-value>
</argument>
```

Every Test Case hook program will receive this argument data as part of the command line execution. For example, the setTestArgs Hook program would be executed as follows:

hello_adjust.sh User=newUser

3.2.4 Test Case Hook Elements

The Test Case Hook Elements defines the Test Case Program and other associated assistant programs to be executed by the XTF.

name (Required Element)

Each Hook specified must include a name element. Currently, the valid value of the name element is 'run'.

savelog (Optional Element)

The savelog element may be optionally included, with a value of “false”, to indicate that any output resulting from these execution of the Test Case program should not be saved to a log file. If this element is not specified, a default value of “true” will be applied and any output from the Test Case program will be directed to a log file. The log file is automatically generated by the XTF when execution of a Hook program begins. The name of the file will include the name of the the associated Test Case name and a time stamp. All log files will be stored in the '/opt/SUNWhcts/logs/' directory.

executable (Required Element)

The executable element is a required element used to specify the execution command required to execute the Hook program. Two sub-elements are utilized to completely specify the execution command:

basedir (Optional Sub-Element)

The basedir sub-element may be used to specify a base directory other than the one specified in the Test Case Element. If this element is not specified the base directory specified in the Test Case Element will be automatically applied. Both relative and absolute paths may be specified. For example, '/opt/SUNWhcts/testcases/example' and 'example' are equivalent and are both accepted by the XTF.

command (Required Sub-Element)

The name of the Test Case program. The specific details of the run Hook are listed below.

run (Required Hook)

This Hook is used to indicate which program is to be executed as the actual Test Case program.

The following Hook definition must be added to our example Test Case Descriptor File to specify the run Hook:

```
<hook>
   <name>run</name>
   <executable>
      <command>hello_world.sh</command>
   </executable>
</hook>
```

3.3 Completed hello_world Example

Now we will combine all of the example components together to form a complete Test Case Descriptor File for our hello_world example:

[File: hello_world.xml]

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase name="hello_world" basedir="example">
    <attribute>
        <name>version</name>
        <value>1.0</value>
    </attribute>
    <attribute>
        <name>short-description</name>
        <value>Hello World!</value>
    </attribute>
    <attribute>
        <name>long-description</name>
        <value>This is the basic Hello World introductory example.</value>
    </attribute>

    <argument>
        <name>sleeptime</name>
        <type>env</type>
        <default-value>1000</default-value>
    </argument>

    <hook>
        <name>run</name>
        <executable>
            <command>hello_world.sh</command>
        </executable>
    </hook>
</testcase>
```

After placing the Test Case Program in the appropriate directory (In this case: /opt/SUNWhcts/testcases/example) the Test Case has been wrapped into HCTS. Next we will take a look at specifying how the Test Case will appear in the HCTS UIs through the creation of a Test Category.

4.0 How to Create a Test Category Descriptor File

A Test Category Descriptor File is used to define the Test Cases which will be combined to facilitate category level testing. Typically this means Test Cases will be combined to form a testing procedure for a particular hardware category such as Network, Storage or Video. One or more Test Cases must be associated with the Test Category. Through the creation of a Test Category Descriptor File a developer defines a testing procedure which will be accessible in the “Controller Test” portion of any HCTS UI.

4.1 Creating a Test Category Descriptor File

This file will specify which Test Cases are included in the Test Case Category, as well as all other supporting programs and description information. The name of the Test Category Descriptor File should be the same as that of the Test Category. The postfix of the filename should be the same as the value of “xtf.test.postfix” variable defined in the configuration file. The “xtf.test.postfix” variable is defined in the file 'xtf.conf', located in the '/opt/SUNWhcts/etc/' directory. Typically this postfix value will be ".xml".

4.1.1 Test (Required Element)

The Test element is the root of the Test Category Descriptor File.

name (Required Attribute)

The name of the Test Category. The name must be unique in the scope of all installed Test Categories.

basedir (Optional Attribute)

All relative paths specified by executable elements in the Test Category Descriptor File will utilize this path as a base directory. Both relative and absolute paths may be specified. For example, '/opt/SUNWhcts/tests/example' and 'example' are equivalent and are both accepted by the XTF. Note: A base directory must be specified in at least one location in the Test Category Descriptor File, either in the Test element or as an attribute of each Test Category Hook.

autoGroupId (Optional Attribute)

The autoGroupId attribute specifies a boolean value. If the autoGroupId value is “true” then the id attribute of each testcasegroup element will be automatically assigned. If the value is “false” then each testcasegroup id attribute must be explicitly defined. The default value for autoGroupId is “false”.

Now we will construct a Test Category Descriptor File for our hello_world example and a few other simple Test Cases. The hello_category.xml Descriptor File will be placed in the /opt/SUNWhcts/tests/ directory and all supporting programs will be placed in the /opt/SUNWhcts/tests/example/ directory. We open a new file and start the Test Category Descriptor File as follows:

```
<?xml version="1.0" encoding="UTF-8"? >
<test name="hello_category” basedir="example" autoGroupId="false">
```

4.1.2 Attribute Elements

Attribute Elements describe a set of informational data to be passed to the HCTS XTF. Each attribute must specify a name and value element.

name

Specified the name of the attribute.

value

Specifies the value of the attribute.

The following attributes are reserved.

name: version (Optional Attribute)	**value**: The version of the Test Category
name: short-description (Optional Attribute)	**value**: Provide a brief description of the Test Category.
name: long-description (Optional Attribute)	**value**: Provide a detailed description of the Test Category. Typically, this description will describe the included Test Cases, runtime, usage considerations, detailed purpose and any other instructive statements.
name: message-file (Required Attribute)	**value**: Provide the name of the XML file containing all of the predefined messages that any Test Case program associated with the Test Category will reference during operation. Each element of the message file must specify a unique message number and a corresponding message. This file should be located at /opt/SUNWhcts/messages directory.
name: includes (Required Attribute)	**value**: A list of Test Cases which will be included as part of the Test Category Descriptor File. Separate the names of the Test Cases using a ":" (Colon) character.

Now we append the following attribute elements to describe our Test Category Descriptor File.

```
<attribute>
    <name>version</name>
    <value>1.0</value>
</attribute>
<attribute>
    <name>short-description</name>
    <value>Hello Category</value>
</attribute>
<attribute>
    <name>long-description</name>
    <value>If you execute this category the following Test Cases will run:
            hello_world, font_changer, color_changer</value>
</attribute>
<attribute>
    <name>message-file</name>
    <value>helloCategory-message.xml</value>
</attribute>
<attribute>
    <name>includes</name>
    <value>hello_world:font_changer:color_changer</value>
</attribute>
```

4.1.3 Argument Elements

Arguments at the Test Category level permit the passing of one or more environment variables to each Test Case programs and Hook programs associated with a Test Category. Specifying arguments is optional. Each Hook program at the Test Case level and Test Category level will receive arguments with one exception. The probe Hook (described in Section 4.1.5) does not receive arguments.

name (Required Element)

value: The name of the argument.

default-value (Optional Element)

value: The default value of this argument.

Description (Optional Element)

value: The text should describe the argument purpose, expected range of input and an example of input when applicable.

The following example implements a Test Category argument which defines a global "runtime" argument which can be referenced by all Hook programs associated with the Test Category Descriptor File:

```
<argument>
    <name>runtime</name>
    <default-value>1000</default-value>
</argument>
```

4.1.4 testcategory (Required Element)

Every Test Category Descriptor File must specify exactly one testcategory element. This element describes the execution order and grouping of the Test Cases associated with the Test Catergory. There are two execution rules to consider when designing the testcategory element: (A) All Test Groups are executed serially and (B) All Test Cases belonging to a Test Group are executed in parallel.

Attribute: name (Required)

The name of the Test Category.

testcasegroup (Required Element)

A testcategory element must specify one or more testcasegroup elements. Reminder: All Test Groups specified in a testcategory will be executed one after the other (i.e. In serial).

Attribute: id

Each testcasegroup element requires a unique identifier. If the autoGroupId

attribute of test element was set to “false” in the test element then the developer is responsible for assigning an appropriate group identifier to every testcasegroup element. The value of this attribute can be any character; however, numerical sequences are typically used.

Attribute: autoCaseId

Much like the autoGroupId attribute assigned in the test element, the autoCaseId attribute indicates whether the Test Cases belonging to a group will be automatically assigned. If the value of this attribute is set to “false” then the developer will be responsible for assigning unique identifiers to each Test Case belonging to the testcasegroup element.

testcase (Required Element)

Reminder: All Test Cases specified in a testcasegroup will be executed at the same time (i.e. In parallel).

Required Attribute: name

The name of one of the Test Cases to be assigned to the associated testcasegroup.

Optional Attribute: argument_auto_bind

The argument_auto_bind attribute is used to create a dependency between arguments of the same name specified in both the Test Category Descriptor File and Test Case Descriptor File. If the of this attribute is set to “true” then the dependency will be enabled and the value of any Test Case argument will be replaced with the value of the corresponding Test Category argument. Essentially this feature allows a Test Category to apply global group argument values to coordinate Test Case execution and management.

Optional Element: attribute

A testcase Element may optionally specify one or more attribute elements. The name of any attribute element must match one or the names defined by the associated Test Case. Attributes defined for a given Test Case in the Test Category Descriptor File will overwrite the corresponding attribute value defined in the Test Case Descriptor File.

name (Required Sub-element)
The name of a Test Case attribute matching the name of an attribute in defined in the corresponding Test Case Descriptor File.

value (Required Sub-element)
The new value which will replace the corresponding Test Case value.

Optional Element: bind

The bind element permits developers to optionally bind Test Category arguments to Test Case arguments. Note that the bind element differs from the argument_auto_bind attribute in that a bind element can associate arguments which do not share the same name. In the example below, the Test Category argument “runtime” is bound to the Test Case “sleeptime” argument. As a result, when the Test Category is executed, the value of the argument “sleeptime” in the Test Case “hello_world” will be assigned the value of the “runtime” argument.

Required Attribute: test_argument
The name of a Test Category argument which will replacing a Test Case argument.

Required Attribute: testcase_argument
The name of Test Case argument which will be replaced by a Test Category argument.

Let's take a look at a specification which implements two groups. Group 0 specifies the Test Case hello_world should be executed first. Group 1 specifies the Test Cases font_changer and color_changer should be executed at the same time after the Test Case hello_world has completed.

```
<testcategory name="hello_category">
        <testcasegroup id="0" autoCaseId="false">
                <testcase id ="0" name="hello_world” argument_auto_bind="false">
                        <bind test_argument="runtime" testcase_argument="sleeptime"/>
                </testcase>
        </testcasegroup>
        <testcasegroup id="1" autoCaseId="true">
                <testcase name="font_changer” argument_auto_bind="true">
                        <attribute>
                                <name>long-description</name>
                                <value>The font used to display “Hello World!” will change every 2
                                seconds.</value>
                        </attribute>
                </testcase>
                <testcase name="color_changer” argument_auto_bind="false">
                        <attribute>
                                <name>long-description</name>
                                <value>The color used to display “Hello World!” will change every 1
                                seconds.</value>
                        </attribute>
                </testcase>
        </testcasegroup>
</testcategory>
```

4.1.5 Hook Elements

A Test Category Hook is a subprogram executed as part of a Test Category running cycle for the purpose of assisting the proper execution of the complete testing progress.

name (Required Element)

Each Hook specified must include a name element. The possible values for this name

element are as follows: probe, getDevices, setTestArgs, setup, check and clean.

savelog (Optional Element)

The savelog element may be optionally included, with a value of “false”, to indicate that any output resulting from these execution of the Hook program should not be saved to a log file. If this element is not specified, a default value of “true” will be applied and any output from the Hook program will be directed to a log file. The log file is automatically generated by the XTF when execution of a Hook program begins. The name of the file will include the name of the associated Hook program, the associated Test Category name and a time stamp. All log files will be stored in the '/opt/SUNWhcts/logs/' directory.

executable (Required Element)

The executable element is a required element used to specify the execution command required to execute the Hook program. Two sub-elements are utilized to completely specify the execution command:

basedir (Optional Sub-Element)

The basedir sub-element may be used to specify a base directory other than the one specified in the test Element. If this element is not specified, the base directory specified in the test Element will be automatically applied. Note: A base directory must be specified in at least one location in the Test Category Descriptor File, either in the Test element or as an attribute in each of the Test Category Hook executable elements.

command (Required Sub-Element)

The name of the Test Category Hook program.

The specific details of each type of Hook are listed below:

probe

The probe Hook program may be deployed at the developer's option. This Hook program is the first program to be executed. Typically the probe Hook is used to scan one or more testing targets to assess their readiness for testing. If the probe Hook program returns any non-zero value the Test Category will not be executed.

As an example, suppose a Test Category employs a series of network related Test Cases. The developer would design the associated probe Hook program to ensure the target network devices have been properly configured. If a condition exists which would invalidate the testing procedure the probe Hook program would output a non-zero value and one or more appropriate messages (Using hcts_msg).

The following Hook definition must be added to the Test Category Descriptor File to specify the probe Hook:

```
<hook>
        <name>probe</name>
        <executable>
                <command>hello_probe.ksh</command>
        </executable>
</hook>
```

getDevices

The Test Category getDevices Hook program enables developers to provide hardware selection options to HCTS users. The hardware specified by this Hook applies only to the Test Category UI.

Each line of output generated by a Test Category getDevices Hook program may represent only a single device and must conform to the following format:

parentId: argName=argValue;devName;devDriver;devData;devStatus :nodeId

parentId

The nodeId corresponding to the parent device of the associated device.

argName

The name of the argument. An argument name need not be unique. Developers may use this item to categorize the type of device being referenced (Ex. device, controller, peripheral, etc.).

argValue

The value of argument. Typically this value will be used by a Developer's Hook programs to identify a specific device.

devName

The name of the device. This name will typically be displayed to the user when device selection is required. A developer should ensure each device name is unique. If two devices share the same name some differentiating indicator should be attached to the device name.

devDriver

The name of the device driver utilized by the associated device. Driver information will typically be displayed along with the device name in the UI.

devData

Any other data can be passed to the XTF and UI using devData. Typically, this data will be displayed exactly as specified through various informational display implementations. The display implementation utilized by a UI will be dependent on the status indicated in the devStatus field.

devStatus

There are three accepted status indicators for this field:
0 - No errors or warning generated. Any data existing in the devData field will be displayed as informational data associated with the device.
1 - Error has been generated. Indicates a critical failure has occurred or been detected for the associated component. Testing or Certification of this component may be restricted.
2 - Warning has been generated. Indicates a general warning is to be issued. Such warnings should be used to indicate required actions needed to avoid potential testing failures.

Note: The devData field should be utilized to specify any messages to be displayed as a result of the devStatus indicator. Error and warning status indicators should never be issued without qualifying message statements.

nodeId

The unique identifier of a device specified by a Test Category getDevices Hook program.

The following is a possible output sample from a Test Category getDevices Hook program:

```
0: controller=nvidiaA; nVidia Corporation MCP55 Ethernet; nge; Manufacturer=NVIDIA ; 0 :1
0: controller=intelA; Intel Corporation 82546EB; e1000g; Manufacturer=Intel ;0 :2
1: device=port1; Port 1; ; ;0 :3
1: device=port2; Port 2; ; ;0 :4
2: device=port3; Port 3; ; ;0 :5
2: device=port4; Port 4; ; ;0 :6
```

This output will be interpreted by the XTF as: Device 1 (Broadcom) has two children devices, port1 and port2. Device 2 (Intel) has two children devices, port3 and port4.

After user interaction the selected hardware will be made available to each Hook program as environment variables corresponding to argName. If more than one selection has been made for a corresponding argName the values will be separated by the XTF_ARGUMENT_VALUE_SEPARATOR (Typically set to “*]”). If a user had selected both BroadCom and Intel in our previous example then the environment variable “controller” would have the value “BroadCom*]Intel”.

The following Hook definition must be added to the Test Category Descriptor File to specify the getDevices Hook:

```
<hook>
        <name>getDevices</name>
        <executable>
                <command>hello_fetch.ksh</command>
        </executable>
</hook>
```

setTestArgs

The Test Category setTestArgs Hook program is responsible for dynamically modifying a Test Category.

Each line of output generated by a Test Category setTestArgs Hook program must conform to the following format:

categoryname groupid testcasename testcaseid [argumentname=argumentvalue | @attributename=attributevalue...]

categoryname

Corresponds to the name of the category element in which the argument or attribute to be modified belongs.

groupid

Corresponds to the group ID element in which the argument or attribute to be modified belongs. If the group ID has been auto generated then a sequence of numbers, beginning with 0, will be applied to each group in the order in which they are listed.

testcasename

Corresponds to the Test Case element in which the argument or attribute to be modified belongs.

testcaseid

Corresponds to the Test Case element in which the argument or attribute to be modified belongs.

argumentname

Indicate the name of the argument to be modified using the following format: argumentname=argumentvalue

@attributename

Indicate the name of the argument to be modified using the following format: @attributename=attributevalue

The Test Category setTestArgs Hook is commonly used to modify the number Test Case instances to match the number of hardware components selected by a user or to modify arguments based on the detected hardware configurations.

The Test Category setTestArgs Hook program may be used to achieve the following results:

- Add a new Test Case group

- Add a new Test Case for an existing Test Case group
- Skip the execution of a Test Case
- Modify the argument value belonging to a Test Case
- Modify the attribute value or add a new attribute to a Test Case

Here are some examples of Test Case execution modifiers based on the example generated so far:

Add a new Test Case named fontSize_changer to a new Test Case Group:

```
hello_category 2 fontSize_changer 0
```

Remove the Test Case named font_changer from Group 1:

```
hello_category 1 font_changer 0 @instance=0
```

The following Hook definition must be added to the Test Category Descriptor File to specify the setTestArgs Hook:

```
<hook>
        <name>setTestArgs</name>
        <executable>
                <command>hello_adjust.ksh</command>
        </executable>
</hook>
```

setup

The setup Hook program may be optionally utilized by developers to prepare the environment before the execution of a Test Category. To continue with our network example, the setup Hook program would be configured to establish connections with remote systems. If the Hook program returns a non-zero value then the testing procedure will not be executed and the user will be informed testing has failed.

The following Hook definition must be added to the Test Category Descriptor File to specify the setup Hook:

```
<hook>
        <name>setup</name>
        <executable>
                <command>hello_setup.ksh</command>
        </executable>
</hook>
```

check

The check Hook program may be optionally utilized by developers to verify proper environment configuration has be completed. Typically the check and setup Hook programs are tightly coupled with each other. If the Hook program returns a non-zero value then the testing procedure will not be executed and the user will be informed testing has failed.

The following Hook definition must be added to the Test Category Descriptor File to

specify the setTestArgs Hook:

```
<hook>
        <name>check</name>
        <executable>
                <command>hello_check.ksh</command>
        </executable>
</hook>
```

clean

The clean Hook program may be optionally utilized by developers to clean and restore the system environment after completing execution of a Test Category. Typically temporary files are deleted and modified environment components are restored.

The following Hook definition must be added to the Test Category Descriptor File to specify the setTestArgs Hook:

```
<hook>
        <name>clean</name>
        <executable>
                <command>hello_clean.ksh</command>
        </executable>
</hook>
```

4.2 Completed hello_category Example

Now we will combine all of the example components together to form a complete Test Category Descriptor File for our hello_category example:

[File: hello_category.xml]

```
<?xml version="1.0" encoding="UTF-8"? >
<test name="hello_category" basedir="example" autoGroupId="false">
        <attribute>
                <name>version</name>
                <value>1.0</value>
        </attribute>
        <attribute>
                <name>short-description</name>
                <value>Hello Category</value>
        </attribute>
        <attribute>
                <name>long-description</name>
                <value>If you execute this category the following Test Cases will run: hello_world,
                font_changer, color_changer</value>
        </attribute>
        <attribute>
                <name>message-file</name>
                <value>helloCategory-message.xml</value>
        </attribute>
        <attribute>
                <name>includes</name>
                <value>hello_world:font_changer:color_changer</value>
        </attribute>
```

```
<argument>
        <name>runtime</name>
        <default-value>1000</default-value>
</argument>

<testcategory name="hello_category">
        <testcasegroup id="0" autoCaseId="false">
                <testcase id ="0" name="hello_world” argument_auto_bind="false">
                        <bind test_argument="runtime" testcase_argument="sleeptime"/>
                </testcase>
        </testcasegroup>
        <testcasegroup id="1" autoCaseId="true">
                <testcase name="font_changer” argument_auto_bind="true">
                        <attribute>
                                <name>long-description</name>
                                <value>The font used to display “Hello World!” will change every 2
                                seconds.</value>
                        </attribute>
                </testcase>
                <testcase name="color_changer” argument_auto_bind="false">
                        <attribute>
                                <name>long-description</name>
                                <value>The color used to display “Hello World!” will change every 1
                                seconds.</value>
                        </attribute>
                </testcase>
        </testcasegroup>
</testcategory>

<hook>
        <name>probe</name>
        <executable>
                <command>hello_probe.ksh</command>
        </executable>
</hook>
<hook>
        <name>getDevices</name>
        <executable>
                <command>hello_fetch.ksh</command>
        </executable>
</hook>
<hook>
        <name>setTestArgs</name>
        <executable>
                <command>hello_adjust.ksh</command>
        </executable>
</hook>
<hook>
        <name>setup</name>
        <executable>
                <command>hello_setup.ksh</command>
        </executable>
</hook>
<hook>
        <name>check</name>
        <executable>
                <command>hello_check.ksh</command>
        </executable>
</hook>
<hook>
```

```
                <name>clean</name>
                <executable>
                        <command>hello_clean.ksh</command>
                </executable>
        </hook>
</test>
```