

---

# **XTT Function Modules Guide**



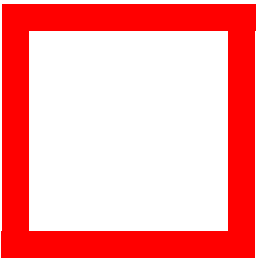
**X-treme Testing Tool version 0.1**

Created on 20 March 2009.

Copyright © 1997 – 2009, 724 Solutions Inc. All rights reserved.

724 Solutions is a trademark of 724 Solutions Inc. All other brands and names used herein are or may be trademarks of their respective owners. This documentation is protected under international copyright laws. No part of this documentation may be reproduced without the prior written consent of 724 Solutions Inc.

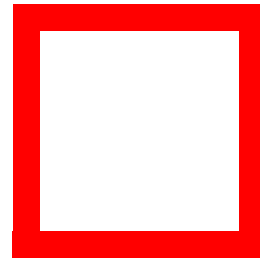




# Contents

- Contents** 3
- Introduction** 4
  - Purpose and Scope 4
  - Target Audience 4
  - Conventions 4
  - Related Documents 5
  - Summary of Changes 5
- Understanding Function Modules** 6
  - Overview 6
  - Where to Find the Documentation for Function Modules 6
  - Documentation of Function Modules 7
- Writing Function Modules** 8
  - Overview 8
  - Points to keep in mind when writing a Function Module 8
  - Source Code of a Function Module 10
  - A Sample Test Script to test the Function Module 12
  - Description of the Function Modules 14
- Index** 17





# Introduction

X-treme Testing Tool (XTT) is a Java-based testing tool that helps test X-treme FrameWork (XF) products. XTT is a framework that uses Function Modules to build and test the different functionalities of XF products. Function Modules are Java classes that are written or extended by developers. Testers then use these Function Modules to write test scripts in XML that are used to test XF products. Further details about XTT are available in the *Installation Guide*, the *User Interface Guide* and the *Utility Guide*.

This manual describes the Function Modules written by developers who extend XTT, which are then used by testers who use XTT to test XF products.

---

## Purpose and Scope

This manual is designed to help you understand and use Function Modules.

It includes the following parts:

- Introduction (this chapter): provides an Overview of the current manual.
- Understanding Function Modules: provides an Overview of Function Modules.
- Writing Function Modules: provides additional information about writing Function Modules that is relevant to developers extending XTT and to testers using test scripts to test the functionalities of XF products.

---

## Target Audience

This document is intended for developers who extend XTT, and for software testers who use XTT to test XF products.

---

## Conventions

This guide uses the following typographic conventions:

- **Enter** -- menu options, page and dialog titles, section and field names, and other visual elements of the GUI.
  - `cmd` -- text that you enter using the keyboard, as well as other code.
-

---

### Related Documents

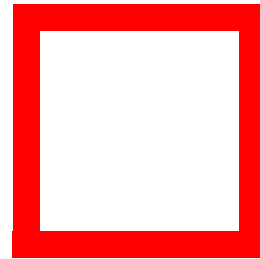
This manual is part of the XTT v0.1 documentation set that includes the following documents:

- Installation Guide
- User Interface Guide
- Utility Guide

---

### Summary of Changes

Issue	Date	Release Changes
v0.1	March 13, 2009	For XTT Release (Draft version)



# Understanding Function Modules

---

## Overview

Function Modules are Java classes written by developers. Software testers then use these modules to test certain functionalities of the X-treme FrameWork (XF) products.

Function Modules are the basic building blocks of the test scripts written by the testers. It is essential that developers write Function Modules, as without these, testers cannot test the product(s).

Function Modules are written in Java and are named for the functionality that they test. For example, `FunctionModule_HTTP` can help the tester test the HTTP and HTTPS GET functions. Developers write new function modules or enhance existing ones, as and when the need arises to test certain functionalities in the product.

Each function module consists of several methods, also called functions. For example, the `sendGetRequest` function is part of the `FunctionModule_HTTP` function module. Once a developer writes a certain function module, testers use this function module to call the function in their test scripts by passing certain parameters. These test scripts are written in XML.

---

## Where to Find the Documentation for Function Modules

These function modules are documented in the Javadocs that accompany the XTT product. The Javadocs can be referred from the doc folder that accompanies the XTT package. To open the Javadocs:

1. Go to the folder where you have extracted XTT. You will find a doc folder inside it.
2. Click on the doc folder and you will see an .html file called `index.html`. This file contains all the function modules.

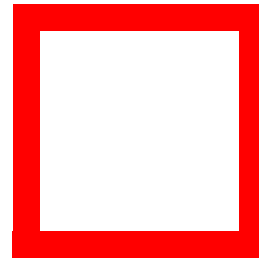
All the function modules extend the `FunctionModule` class. So, there are multiple classes extending the `FunctionModule` class.

---

## Documentation of Function Modules

The Function Module Javadocs consists of a left frame that lists the different classes, including the FunctionModule classes. When you click on any FunctionModule class, the details of that class are displayed in the right frame. The Function Module Javadocs consists of the following parts:

- Name of the class
- Hierarchy of the class
- Package Details
- Description of the class
- Field Summary
- Constructor Summary
- Method Summary



# Writing Function Modules

---

## Overview

A function module extends the `FunctionModule` class. Developers write new function modules or extend existing ones to enable testers to test the different functionalities of XF products. This chapter outlines how to write a Function Module, and also gives examples of the source code for a Function Module and a test script written in XML used to test a Function Module.

---

## Points to keep in mind when writing a Function Module

You should perform the following mandatory checks before you start programming business logic when writing a Function Module:

- Check if the argument passed to the Function Module is not null.
- Check the length of the array. This check is done to ensure that the method has all the necessary parameters for executing the method successfully.

The following are some points to keep in mind when writing a Function Module:

1. Function Modules can be written only using the Java programming language.
2. All Function Modules must extend the `FunctionModule` class.

For example, `public class FunctionModule_HTTP extends FunctionModule {...}`

3. All the functions in the Function Module have to be public.

For example, `public void checkHeader(String[] parameters){...}`

4. Each (public) method in the Function Module always takes a single argument, which is an array of `String`.

For example, `public void checkPostData(String[] parameters) {...}`

---



5. The first thing you need to do in the method is to check the argument for null.

For example,

```
if(parameters==null)
{
    XTTProperties.printFail(this.getClass().getName()+" :
        enableProxy: proxyAddress proxyPort");
    XTTProperties.printFail(this.getClass().getName()+" :
        enableProxy: connection proxyAddress proxyPort");
    return;
}
```

6. If the argument is null, print a fail statement containing the method name and the list of allowed parameters and return.
7. If the argument is not null then check for the length of the array. This check is to confirm that the method has all the necessary parameters for executing the method successfully. The check is dependent on the method's business logic and can vary from method-to-method.

For example,

```
if(parameters.length<3||parameters.length>4)
{
    XTTProperties.printFail(parameters[0] +
        ":"+"MISSING_ARGUMENTS+": proxyAddress proxyPort");
    XTTProperties.printFail(parameters[0] +
        ":"+"MISSING_ARGUMENTS+": connection proxyAddress
        proxyPort");
    XTTProperties.setTestStatus
        (XTTProperties.FAILED_WITH_MISSING_ARGUMENTS);
}
```

8. In the example given below, the method checks that the length of the array should at least be three. It implies that the calling test script should at least pass two parameters. Note that the zeroth element of the array is always the method name and the parameters passed by the test script start from the first element of the array (in the same order).
9. After checking the length of the array, proceed with the business logic of the method. For example, the business logic for FunctionModule\_HTTP is different from that for FunctionModule\_SNMP.
10. Build the code and run the build script.
11. Write a test script to test the code.
12. Check the code into a repository after getting the necessary permissions.

---

## Source Code of a Function Module

The following is the source code of a Function Module:

```
package com.mobilgw.xtt;

//all the imports

public class FunctionModule_Basic extends FunctionModule
{
    /**
     * Add the integer values of the parameters.
     *
     * @param parameters    array of String containing the parameters.
     * <br><code>parameters[0]</code> argument is always the method name,
     * <br><code>parameters[1]</code> argument is the variable to store
     * the result to,
     * <br><code>parameters[2]</code> and following are the integer values
     * to add up
     * <br>If null is used as <code>parameters</code> it sends the
     * allowed parameters list
     * to the {@link XTTProperties#printFail(java.lang.String)}
     * XTTProperties.printFail(java.lang.String)} method and returns.
     */
    public void addVariable(String parameters[])
    {
        if(parameters==null)
        {
            XTTProperties.printFail(this.getClass().getName()+" :
            addVariable: variableName valueX");
            return;
        }
        if(parameters.length<3)
        {
            XTTProperties.printFail(parameters[0]+": "+MISSING_ARGUMENTS+" :
            variableName valueX");
            XTTProperties.setTestStatus
            (XTTProperties.FAILED_WITH_MISSING_ARGUMENTS);
        } else
        {

```


```
int i=2;
try
{
    StringBuffer add=new StringBuffer("");
    String plus="";
    long val=0;
    for(i=2;i<parameters.length;i++)
    {
        val=val+(Long.decode(parameters[i])).longValue();
        add.append(plus+" "+parameters[i]+"");
        plus=" + ";
    }
    XTTProperties.printInfo(parameters[0]+":
    "+parameters[1]+"= "+add+" = "+val);
    XTTProperties.setVariable(parameters[1],val+"");
} catch (NumberFormatException nfe)
{
    XTTProperties.printFail(parameters[0]+": parameter"+i+" is
    not a number: '"+parameters[i]+"");
    XTTProperties.setTestStatus
        (XTTProperties.FAILED_WITH_INVALID_ARGUMENTS);
}
}
}
```

---

## A Sample Test Script to test the Function Module

The following is a sample test script to test the Function Module given above:

```
<test>
<name>xtt.TC.001</name>
<description>Validate function module Basic</description>
<!-- Add the integers 5 and 7 and store the result in a variable named
as 'sum' -->
<function name="addVariable" module="Basic">
<parameter>sum</parameter>
<parameter>5</parameter>
<parameter>7</parameter>
</function>
<!-- Validate the variable 'sum' to be 12 -->
<function name="compareString" module="Basic">
<variable>sum</variable>
<parameter>12</parameter>
</function>
<!-- Add 3 to the variable named 'sum' and store the result in the
same variable -->
<function name="addVariable" module="Basic">
<parameter>sum</parameter>
<variable>sum</variable>
<parameter>3</parameter>
</function>
<!-- Validate the variable 'sum' to be 15 -->
<function name="compareString" module="Basic">
<variable>sum</variable>
<parameter>15</parameter>
</function>
<!-- Remove the variable 'sum' from memory -->
<function name="removeVariables" module="Basic">
<parameter>sum</parameter>
</function>
<!-- Creates a new variable named 'x' with value 25 -->
<!-- Note that we aren't adding anything to 'x' -->
<function name="addVariable" module="Basic">
<parameter>x</parameter>
<parameter>25</parameter>
```



```
</function>
<!-- Validate the variable 'x' to be 25 -->
<function name="compareString" module="Basic">
<variable>x</variable>
<parameter>25</parameter>
</function>
<!-- Remove the variable 'x' from memory -->
<function name="removeVariables" module="Basic">
<parameter>x</parameter>
</function>
</test>
```

---

## Description of the Function Modules

Name of Function Module	Description
FunctionModule	Is the FunctionModule base class for all Function Modules.
FunctionModule_Basic	Provides functions for basic operations that can be done in test scripts.
FunctionModule_Content	Provides functions to check if content is of various image types. It also has utility functions to extract and store various image properties like its dimensions.
FunctionModule_Diameter	Provides Diameter and DiameterS GET functions.
FunctionModule_DNS	Returns an IP for A or AAAA requests, first checking the internal XTT DNS name store, then doing a lookup through the Java interface to the local machine's DNS servers.
FunctionModule_EWS	Waiting for information from Roger.
FunctionModule_HTTP	Provides HTTP and HTTPS GET functions.
FunctionModule_ICAP	Provides functions to interact with the ICAP server.
FunctionModule_LDAP	Provides functions for Lightweight Directory Access Protocol (LDAP). It has methods for starting secure and nonsecure LDAP server instances that can process LDAP requests.
FunctionModule_MMSC	Provides functions for starting and stopping secure and nonsecure MMSC instances. MMSC stands for Multimedia message centers and the running instances of MMSC are essentially used to receive and send an appropriate response to a MM7 submit request. The response can be customized by using the various functions mentioned in the Javadoc of this FunctionModule.
FunctionModule_Multipart	Provides functions for encoding and decoding a MIME/Multipart message.
FunctionModule_Proxy	Provides simple HTTP proxy.
FunctionModule_Push	Provides functions for sending WAP Push messages. It also provides functions to query if the body of a PAP response (that is, response to a WAP PUSH request) contains a specified regular expression value.

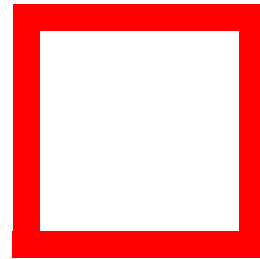
Name of Function Module	Description
FunctionModule_Radius	Provides functions for Radius Logon and Radius Logoff.
FunctionModule_Remote	Lets tests interact with RemoteXTT.
FunctionModule_RTSP	Provides RTSP and RTSPS GET functions.
FunctionModule_Semaphore	Provides functions for semaphoring.
FunctionModule_SIP	Provides SIP and SIPS GET functions.
FunctionModule_SIS	Provides functions for SISServer. SIS stands for Subscriber Information Server. The function module encompasses various methods to receive SIS calls for fetching subscriber information based on the specified Subscriber Information levels.
FunctionModule_SMPP	Provides functions for sending SMPP packets.
FunctionModule_SMS	Provides SMSCenter and SMS send functions.
FunctionModule_SMTP	Provides functions for the SMTP server. It also provides functions for sending mails using SMTP.
FunctionModule_SNMP	Provides functions for testing XTT consistency.
FunctionModule_SOAP_API	Provides the SOAPClient function.
FunctionModule_SQL	Waiting for information from Roger.
FunctionModule_STI	Provides functions for a simple, multi-threaded instance of STI server. STI stands for Standard Transcoding Interface.
FunctionModule_TCP	Provides functions for sending TCP packets.
FunctionModule_Throttle	Provides functions for throttling.
FunctionModule_UCP	Provides functions for sending UCP packets.
FunctionModule_UDP	Provides functions for sending UDP packets.
FunctionModule_VASP	Provides MM7 through HTTP and SOAP functions.
FunctionModule_WAP	Provides connection-oriented unencrypted wap1 GET functions.
FunctionModule_XML	Provides functions for testing XML document consistency.
FunctionModule_XTT	Provides functions for testing XTT consistency.

---

*Tip:* To know more about the various methods of a particular Function Module, refer to the Javadoc of that Function Module.

---





# Index

## D

Description of the Function Modules 14

Documentation of Function Modules 7

## F

Function Modules 6

## S

Sample Test Script 12

Source Code 10

## W

Where to Find the Documentation 6

Writing a Function Module 8

## X

X-treme Testing Tool 4