Testing Macro Function Through Cantata++

Version No.: 1.0

Date: 24-Apr-2011

Project Name: DMAU ARINC 615A

Project Code: C102176

Document Revision History

Version No.	Date	Prepared by / Modified by	Significant Changes
1.0	24-Apr-2011	Kiran Devalla, Saurabh Rahul	Initial Version

Table of Contents

1 Problem Statement

During preprocessing Cantata++ replaces all macros, including macro functions, with its defined content. Because function macros also get replaced with its content Cantata++ does not create stub for these functions which is required to perform testing successfully. This causes macro functions to remain un-tested through Cantata++.

2 Goals and Objectives

The objective is to create a work-around compatible with Cantata++ environment such that macro functions are testable through Cantata++

3 Strategy

The strategy for finding solution to this problem involves the following:

1. Good understanding of Cantata++
2. Brainstorming
3. Research

4 Technical Comparison and Cost Benefit Analysis

Technical comparison of Cantata++ with other relevant testing tools is not applicable here as tool selection process is already completed by client itself and use of Cantata++ for Unit Testing is mandated by client.

5 Success Criteria

Success Criteria:

The approach decided should be

1. Compatible to DO-178B guidelines for airborne software
2. Compatible to embedded software testing fundamentals
3. Compatible to client expectations and technical requirements

6 Resource Requirement

In-project resources may be used to complete this activity. Nevertheless, a generic list of resources is presented here for the purpose of providing a clear insight into resource requirements in case such an activity is carried out independently:

1. Hardware Resource:

    a. Desktop Personal Computer with minimum 512 KB RAM

2. Software/Tool Resource:

    a. Cantata++/ V 5.3.2

    b. GNU-X Tools/ V 4.2

    c. Microsoft Windows/ XP

    d. Microsoft Office/ 2003

3. Network Resource:

    Network resource is not a constraint for carrying out this activity. However, default network resources are available as part of organization's and department's IT policy.

4. Human Resource:

    Human resource may again be a part of the project team itself. Nevertheless, basic requirement may be of only one person with good knowledge of Cantata++. However, other project members/ resources may team-up to provide better insight and expertise. Knowledge of Cantata++ may not be mandatory for support team but for the core team working on the solution a good knowledge of Cantata++ is mandatory.

7 Evaluation Requirement

Success of the approach is evaluated based on the successful evaluation of following documents containing the implementation details:

1. Test Procedure

2. Test Case/ Script

3. Test Result

8 Results Validation

If the test procedure, test case/ scripts and test result are prepared correctly as per Cantata++ features and test result successfully shows that macro function has been successfully tested, as expected, through Cantata++, then it will be considered validated.

9 Piloting Requirements

There is no separate piloting requirement for this. However, once successful the finalized approach will be implemented in DMAU ARINC 615A Project during its Unit Testing Phase in consultation with client.

10 Market Survey

Not Applicable

11 Project Management Plan for In-house Development

Not Applicable

12 Implementation Plan

The solution found and implemented successfully is listed below in steps:

1. Create the test script file, say test.c, as per testing requirements
2. Create a .c file, say replace.c
3. Copy all the contents of the Include Section of file test.c to the Include Section of file replace.c
4. Include an undefined command in replace.c for macro function, say m_macro(), which is there in the testable source code.

   #undef m_macro
5. In dummy.c define alternate name for macro function by replacing 'm_macro()' with 'stub_m_macro()'

   #define m_macro() stub_m_macro()
6. Include the source code file (say, source.c), where function under test (say, m_func()) is defined, in replace.c

   #include "source.c"

   Note: m_func() invokes the macro function, m_macro().
7. Create a function, say pp_test_macro(), in replace.c to invoke the function under test, m_func().

   a. Case I: If m_func() returns some value (not void), say of data-type int

```
int pp_test_macro()
{
        int ret;
        ret = m_func();
        return ret;
}
```

b. Case II: If return type of m_func() is void

```
void pp_test_macro()
{
    m_func();
    return;
}
```

8. In the stub section of test script, test.c, define a stub for stub_m_macro() as shown below:

```
int stub_m_macro()
{
        int returnValue;
        REGISTER_CALL("CPU_CRITICAL_ENTER");

        IF_INSTANCE("INSTANCE1")
        {
        returnValue = 12;
        return returnValue;
        }

        LOG_SCRIPT_ERROR("Call instance not defined.");
        Return returnValue;
}
```

9. In test script, test.c, invoke pp_test_macro() in each associated test case
10. In expected call sequence of function under test, m_func(), invoke original macro function name, m_macro, with proper instance, i.e.,

    `m_macro#INSTANCE1;`

Through this step-by-step approach function m_macro() becomes testable.

Both structural and functional coverage is achieved through this approach.

This approach is compatible with embedded software testing fundamentals, DO-178B guidelines and also with client's requirements.