Native TCT Guideline for Tizen3.0

Content Lists

Tizen Compliance Tests

Tizen Compliance Tests

Overview

- Tizen Compliance Tests (TCT) verify conformance to the Tizen Compliance Specification (TCS).
- These tests are intended to be used by Tizen device implementers to enable the Tizen-compliant development environment for Tizen application developers.

Native TCT

- Native TCT is a set of tools and test cases to test Native requirements defined in the Tizen Compliance Specification (TCS).
- It includes : Native TCT covers Signature, Native API, App Control, Privilege, Resource, Device Capability Features
- Native TCT consists of :
  - UTC (Unit Test Case)
  - ITC (Integration Test Case)
  - CTC (Compatibility Test Case)
  - TBT (Tizen Behavior Test)
  - EFL-UTC Packages
- TCT manager is a GUI tool to manage whole tests, from planning to results.

Common TC Guide

Prerequisite

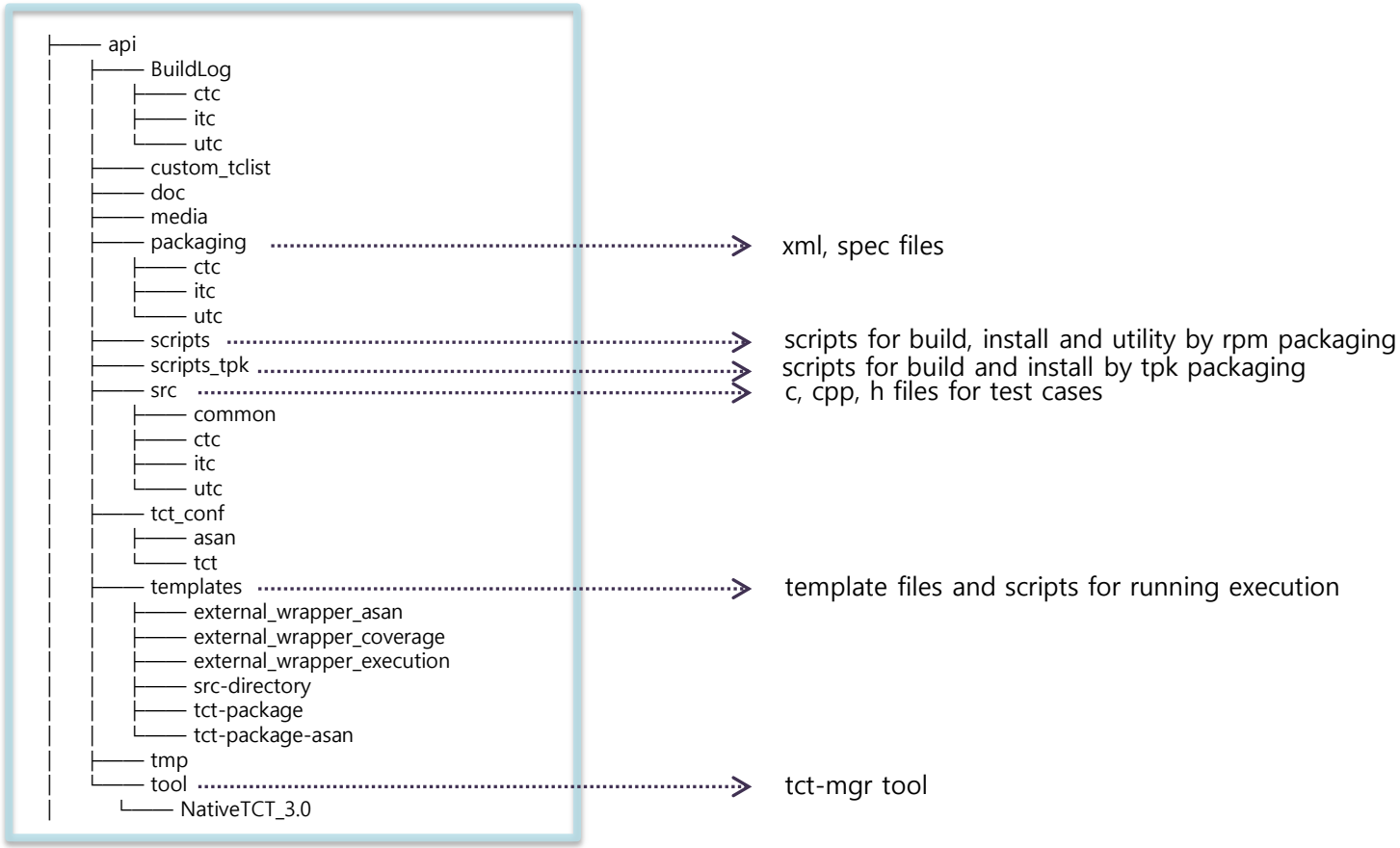
■ TC Information

1) TCT

A. Public Git Path (review.tizen.org) : test/tct/native/api

B. Branch : tizen_3.0

■ Directory Structure



Install TCT-MGR Tool

Prerequisite

- Python
- Java (1.6 and above version)

Download Tool

- Public Git Path (review.tizen.org) : test/tct/native/api
- Branch : tizen_3.0
- api/tool/NativeTCT_3.0.tar.gz

Install tct-mgr on host device

- Extract NativeTCT_3.0.tar.gz
- sudo python tct-setup.py

Install TCT backend runner on target/emulator

- sudo python /opt/tct/tizen_native_3.0/scripts/tct-config-device.py

Adding/Removing TCT Module

Adding TCT Module

[ ] : mandatory field
< > : optional field

1) Go to project root directory : <api> directory
2) Run scripts/init.sh file : This will generate tcbuild and tcbuildsdk binary at root directory.

   $ *./scripts/init.sh*
3) Run addmod/addmodcpp command at project root location:

   $ *sudo ./tcbuild addmod [build_type] [module_name]* (for adding 'C' module)

   $ *sudo ./tcbuild addmodcpp [build_type] [module_name]* (for adding 'CPP' module)

   [build_type] = "itc/ctc/utc"

The new module will be generated with default spec, xml and source files for given 'build_type' (itc / ctc / utc).

tct native header files will also get generated for mobile, wearable, tv and common_iot profile each inside src/'module_name' folder.

If the module is not supported for any of the profile (mobile/wearable/tv/common_iot), then user should mention it in *tct_unsupported.txt* file for that profile.

Note : The tct framework will refer tct_unsupported.txt file to check for unsupported packages, and will automatically exclude such packages from tct build/install process.

Removing TCT Module

1) Go to project root directory : <api> directory
2) Run scripts/init.sh file : This will generate tcbuild and tcbuildsdk binary at root directory.

   $ *./scripts/init.sh*
3) Run rmmod command at project root location:

   $ *sudo ./tcbuild rmmod [build_type] [module_name]* (this command is applicable for 'C' as well as 'CPP' module)

   [build_type] = "itc/ctc/utc"

The spec, xml and source files of the mentioned 'module_name' for given 'build_type' (itc / ctc / utc) will be removed from the <api> directory.

Merging mobile/wearable/tv/common_iot code to single branch code (1/8)

A. Developing tct header files, each for mobile, wearable, tv and common_iot separately.

a.) Now there will be no tct-<module-name>-<native/core>.h inside the module source directory.
Instead of this, there will be 3 tct header files (one each for mobile, wearable, tv and common_iot), in the following nomenclature form:
tct-<module-name>-<native/core>_<device-type>.h.

b.) These tct header files for mobile, wearable, tv and common_iot will be maintained and managed manually by the tct developer.
These tct header files will not be generated by tct build command. So, put the list of required test cases in their respective tct header files accordingly.
Installation file (retriever.sh) will also use these tct header files, and not source code file, to generate tests.xml and
<mobile/wearable/tv/common_iot>_pkg_info.xml for tct-manager execution.

tct header file (src/itc/efl-ext /tct-efl-ext-native_mobile.h)

```
#ifndef __TCT_EFL_EXT_NATIVE_H__
#define __TCT_EFL_EXT_NATIVE_H__

#include "testcase.h"
#include "tct_common.h"

extern void ITs_efl_ext_startup(void);
extern void ITs_efl_ext_cleanup(void);

extern int ITc_eext_floatingbutton_movement_block_set_get_p(void);
//extern int ITc_eext_win_keygrab_set_unset_p(void);
extern int ITc_eext_floatingbutton_mode_set_get_p(void);
extern int ITc_eext_floatingbutton_pos_bring_in_p(void);

testcase tc_array[] = {
    {"ITc_eext_floatingbutton_movement_block_set_get_p", ITc_eext_floatingbutton_movement_block_set_get_p, ITs_efl_ext_startup, ITs_efl_ext_cleanup},
    //{"ITc_eext_win_keygrab_set_unset_p", ITc_eext_win_keygrab_set_unset_p, ITs_efl_ext_startup, ITs_efl_ext_cleanup},
    {"ITc_eext_floatingbutton_mode_set_get_p", ITc_eext_floatingbutton_mode_set_get_p, ITs_efl_ext_startup, ITs_efl_ext_cleanup},
    {"ITc_eext_floatingbutton_pos_bring_in_p", ITc_eext_floatingbutton_pos_bring_in_p, ITs_efl_ext_startup, ITs_efl_ext_cleanup},
    {NULL, NULL}
};

#endif // __TCT_EFL_EXT_NATIVE_H__
```

Merging mobile/wearable/tv/common_iot code to single branch code (2/8)

B. Merging tct-<module-name>-<native/core>.c file for mobile/wearable/tv/common_iot branch to single file.

a.) At the starting lines of the file, include only device specific tct header file, as mention below.

```
#include <stdio.h>
#include <string.h>
#include "tct_common.h"

#ifdef MOBILE
#include "tct-wifi-direct-native_mobile.h"
#endif  //MOBILE

#ifdef WEARABLE
#include "tct-wifi-direct-native_wearable.h"
#endif  //WEARABLE

#ifdef TV
#include "tct-wifi-direct-native_tv.h"
#endif  //TV

#ifdef COMMON_IOT
#include "tct-wifi-direct-native_common_iot.h"
#endif  //COMMON_IOT
```

Use this code snippet at the start of the file.
This will include only device specific tct header file.

b.) If there happens to be any differences in code for mobile/wearable/tv/common_iot, then write that part of code separately for that specific device type using

'#ifdef MOBILE' -> Mobile specific code

'#ifdef WEARABLE' -> Wearable specific code

'#ifdef TV' -> TV specific code

'#ifdef COMMON_IOT' -> common_iot specific code

Merging mobile/wearable/tv/common_iot code to single branch code (3/8)

C. Merging source code files for mobile/wearable/tv/common_iot branch to single file.

a.) If there happens to be any difference in source code file for mobile, wearable and tv, then write that part of code separately for that device type, like as mention in below example, this definition of 'DeleteEvasWindow' function will be built for only TV.

```
#ifdef TV
/**
* @function        DeleteEvasWindow
* @description          Delete Evas window object
* @parameter[IN]    NA
* @return               NA
*/
void DeleteEvasWindow(void)
{
    if(g_pEvasNaviframe)
    {
        evas_object_del(g_pEvasNaviframe);
        g_pEvasNaviframe = NULL;
    }
    if(g_pEvasLayout)
    {
        evas_object_del(g_pEvasLayout);
        g_pEvasLayout = NULL;
    }
    if(g_pEvasConformant)
    {
        evas_object_del(g_pEvasConformant);
        g_pEvasConformant = NULL;
    }
    if(g_pEvasWindow)
    {
        evas_object_del(g_pEvasWindow);
        g_pEvasWindow = NULL;
    }
} ? end DeleteEvasWindow ?
#endif  //TV
```

This definition of 'DeleteEvasWindow' will be compiled for TV only.

b.) In short, for any differences in source code file (.c and .h) for mobile/wearable/tv/common_iot, then write that part of code separately for that device type using

'#ifdef MOBILE' -> Mobile specific code

'#ifdef WEARABLE' -> Wearable specific code

'#ifdef TV' -> TV specific code

'#ifdef COMMON_IOT' -> common_iot specific code

Merging mobile/wearable/tv/common_iot code to single branch code (4/8)

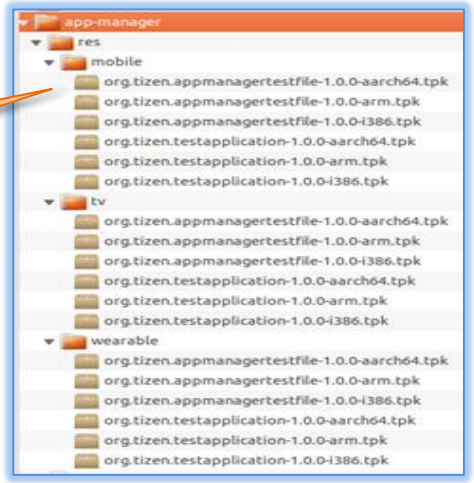
[device_target_type]
"mobile/wearable/tv/common_iot"

D. Copying/Handling tpk and resource files

■ Copying tpk files

If the module needs specific tpk to be installed before execution, the tpk should be placed under 'res/[device_target_type]/' at its source folder.
These tpk will be installed automatically during the time of module execution.

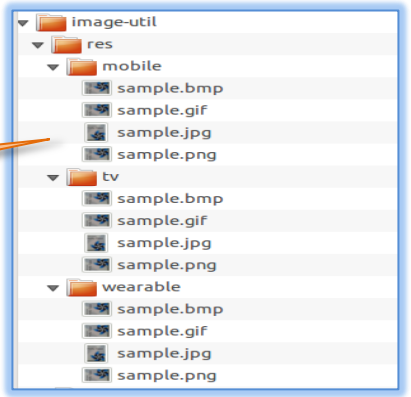
Putting 'tpk' files at 'res/[device_target_type]'. At execution, tpk files will automatically be installed to the device.



■ Copying/Handling the resource files

i. If the resource files need to be copied to 'res' folder at "DEVICE_SUITE_TARGET_30" device target location, then put these files under 'res/[device_target_type]/' location at its source folder.

Putting resource files at 'res/[device_target_type]'. Before execution, these files will automatically be copied to '{DEVICE_SUITE_TARGET_30}/res' device location, and will have "User::App::Shared" smack access label.



ii. post-inst.sh : Developer should use 'post-inst.sh' file to copy the resource files to any target device location or change any permission for the device/resource files. The detailed description about using the 'post-inst.sh' file is mentioned in next slide.

Merging mobile/wearable/tv/common_iot code to single branch code (5/8)

E. Using post-inst.sh to do module specific operation

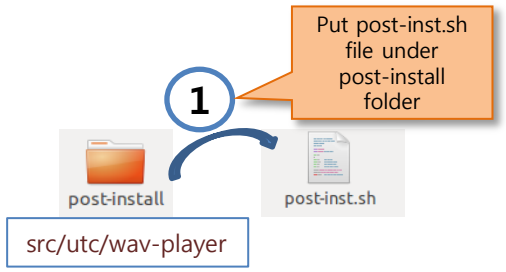
The tool executes post-inst.sh file at the device during execution.

This is module specific file, and it should be put under 'post-install' folder at its respective source folder location.

During tct install command, post-inst.sh automatically copied to tct zip package directory.

Below steps demonstrates the process to copy resource files from host to Internal Storage Directory location of the target device

i. Create 'post-install' folder at module tct source location and then create post-inst.sh file inside it.



ii. Copy necessary resource files to rpm data package in the spec file.

Please refer adjacent diagram to understand this.

module spec file

```
%install
rm -rf %{buildroot}
%make_install
mkdir -p %{buildroot}/usr/share/license
cp LICENSE %{buildroot}/usr/share/license/%{name}
mkdir -p %{buildroot}/usr/share/packages/
cp packaging/utc/%{name}.xml %{buildroot}/usr/share/packages/
mkdir -p %{buildroot}/usr/apps/%{name}/bin

mkdir -p %{buildroot}%{APP_PATH}%{name}/data/res
cp src/utc/wav-player/res/sound_5.wav %{buildroot}%{APP_PATH}%{name}/data/res/sound_5.wav
```

2

Copying resource files to rpm 'data' package

iii. Edit post-inst.sh file.

a. Do copy and other permission related tasks in 'inst' block part.

This block will be executed on the target before test case execution starts.

b. Remove files and do other related tasks in 'else' block part.

This block will be executed on the target after test case execution ends.

module post-inst.sh

```
if [ $MODE == "inst" ]; then
    echo "Installing pre-requisites for the package $PKG_NAME"
    mkdir -p $DEVICE_PHYSICAL_STORAGE_30
    cp -R $APP_DIR/$PKG_NAME/data/* $DEVICE_PHYSICAL_STORAGE_30/
    echo "Installing the pre-requisites for the package $PKG_NAME ======"
else
    echo "Un-installing the pre-requisites for the package $PKG_NAME"
    rm -rf $DEVICE_PHYSICAL_STORAGE_30/res
    echo "Un-installing the pre-requisites for the package $PKG_NAME ======"
fi
```

3

Resource files copied to internal storage directory location with change in files permission

Resource files deleted

Merging mobile/wearable/tv/common_iot code to single branch code (6/8)

F. Add unsupported module information in 'tct_unsupported.txt' to avoid build and install in the unsupported profile.

a.) If there is an unsupported package in specific profile, Add package information into 'tct'_unsupported.txt' like below.

Format : *<device_type>:<architecture>:<build-type>:<module-name>;*

```
###### mobile ######

###### wearable ######

###### tv ######
tv:armv7l:utc:messages;
tv:aarch64:utc:messages;
tv:i586:utc:messages;
tv:x86_64:utc:messages;
```

tct/tct_unsupported.txt

G. "tct_common.h" should be added above "#ifdef [device type]" in tct-[module_name]-core.c/native.c

```
#include "tct_common.h"

#ifdef MOBILE
#include "tct-%{MODULE_NAME}-native_mobile.h"
#endif  //MOBILE

#ifdef WEARABLE
#include "tct-%{MODULE_NAME}-native_wearable.h"
#endif  //WEARABLE

#ifdef TV
#include "tct-%{MODULE_NAME}-native_tv.h"
#endif  //TV
```

Merging mobile/wearable/tv/common_iot code to single branch code (7/8)

H. Merging spec files for mobile/wearable/tv/common_iot branch to single spec file.

a.) In %build section, use below mentioned cmake command for mobile/wearable/tv/common_iot. Edit '-DBUILDTCTYPE' to 'itc or ctc or utc' according to each spec file.

```
%build

%define PREFIX "%{_libdir}/%{name}"

export LDFLAGS+="-Wl,--rpath=%{PREFIX} -Wl,--as-needed"

 %if %{?DEVICE_BUILD_TYPE_MOBILE:1}0
cmake . -DMODULE="%{MODULE_NAME}" -DBUILDTCTYPE="itc/ctc/utc" -DDEVICE_BUILD_TYPE="mobile" -DCMAKE_INSTALL_PREFIX=%{_prefix}
%endif

 %if %{?DEVICE_BUILD_TYPE_WEARABLE:1}0
cmake . -DMODULE="%{MODULE_NAME}" -DBUILDTCTYPE="itc/ctc/utc" -DDEVICE_BUILD_TYPE="wearable" -DCMAKE_INSTALL_PREFIX=%{_prefix}
%endif

%if %{?DEVICE_BUILD_TYPE_TV:1}0
cmake . -DMODULE="%{MODULE_NAME}" -DBUILDTCTYPE="itc/ctc/utc" -DDEVICE_BUILD_TYPE="tv" -DCMAKE_INSTALL_PREFIX=%{_prefix}
%endif

make %{?jobs:-j%jobs}
```

Edit '-DBUILDTCTYPE' to 'itc or ctc or utc' accordingly.

b.) If there happens to be any difference in other sections for mobile/wearable/tv/common_iot, then write that part of scripts separately for that specific device type, like as mention below:

```
BuildRequires:  pkgconfig(ecore)
BuildRequires:  pkgconfig(elementary)
BuildRequires:  pkgconfig(capi-system-info)

 %if %{?DEVICE_BUILD_TYPE_WEARABLE:1}0
BuildRequires:  pkgconfig(libsystemd-journal)
#BuildRequires:  pkgconfig(ecore-x)
#BuildRequires:  pkgconfig(x11)
#BuildRequires:  pkgconfig(xext)
#BuildRequires:  pkgconfig(xi)
#BuildRequires:  pkgconfig(inputproto)
BuildRequires:  pkgconfig(fontconfig)
BuildRequires:  pkgconfig(cairo)
BuildRequires:  pkgconfig(capi-system-info)
BuildRequires:  pkgconfig(appcore-efl)
BuildRequires:  pkgconfig(capi-appfw-application)
%endif
```

This segment of code will be executed for wearable device only.

Merging mobile/wearable/tv/common_iot code to single branch code (8/8)

I. Merging CMakeLists.txt file of specific module for mobile/wearable/tv/common_iot branch to single CMakeLists.txt file.

a.) If there happens to be any difference in CMakeLists.txt file for mobile, wearable, tv and common_iot, then write that part of code separately for that device type, like as mention in below examples the packages requirement are different in mobile, wearable, tv and common_iot.

```
IF( DEFINED MOBILE )
PKG_CHECK_MODULES(${CAPI_LIB} REQUIRED
        ${CAPI_LIB}
    capi-appfw-application
        bundle
    glib-2.0
    capi-system-info
    ecore
    elementary
    dlog
    capi-system-info
)
ENDIF()

IF( DEFINED TV )
PKG_CHECK_MODULES(${CAPI_LIB} REQUIRED
        ${CAPI_LIB}
    capi-appfw-application
        bundle
    glib-2.0
    capi-system-info
    ecore
    elementary
    dlog
    capi-system-info
)
ENDIF()

IF( DEFINED WEARABLE )
PKG_CHECK_MODULES(${CAPI_LIB} REQUIRED
    ${CAPI_LIB}
    capi-appfw-application
    bundle
    glib-2.0
    capi-system-info
    ecore
    elementary
    dlog
    libsystemd-journal
    #ecore-x
    #x11
    #xext
    #xi
    #inputproto
    fontconfig
    cairo
    capi-system-info
    appcore-efl
    capi-appfw-application
    capi-system-info
)
ENDIF()
```

This segment of code will be executed for mobile device only.

This segment of code will be executed for tv device only.

This segment of code will be executed for wearable device only.

Git Commit

Usage

- *git add/rm <files>*
- *git commit –s –m [commit message]*
- For raising patch on existing one, *git commit –-amend*
- *git push origin HEAD:refs/for/tizen_3.0*

Commit Message Rule

- [TC Type][Module][ACR-xxx or Non-ACR][description]
- Example

  [UTC][application][ACR-519][Add TCs for set/unset_defapp]

  [CTC][platform-permission][Non-ACR][Delete TC which need not privilege]

RPM TC Guide

TCT Source Code Build Process

Build TCT Module by RPM approach

[ ] : mandatory field
< > : optional field

1) Go to project root directory : <api> directory
2) Run scripts/init.sh file : This will generate tcbuild and tcbuildsdk binary at root directory.

   $ *./scripts/init.sh*

3) Run build command at project root location:

   $ *sudo ./tcbuild build <build_type> <module_name> [arch_type] [device_type]* (for target build)

   $ *sudo ./tcbuildsdk build <build_type> <module_name> [arch_type] [device_type]* (for emulator build)

   <build_type> = "itc/ctc/utc"

   [arch_type] = "armv7l/aarch64" (for tcbuild) and "i586/x86_64" (for tcbuildsdk)

   [device_type] = "mobile/wearable/tv/common_iot"

At "api/tct_conf/tct" directory, there are tct<32/64>_<device_type>.conf files to support different build process. (32 means armv7l/i586 while 64 means aarch64/x86_64)

To change the build repo url location, please edit corresponding file according to its build configuration type (i.e. to build for wearable target 64 bits, edit tct64_wearable.conf)

The generated RPMS location will be unique for each device_type and architecture type.

RPM_DIR="$HOME/GBS-ROOT-TCT-[**device_type**]/local/repos/[**target/sdk**]/**[armv7l/i586/aarch64/x86_64]**/RPMS"

This will be handled by the script framework internally and user does not need to bother for this.

NOTE:

a.) To build all the packages (itc+ctc+utc for all modules),

$ *sudo ./tcbuild build [arch_type] [device_type]*

b.) To build all the packages for specific build_type, (itc or ctc or utc),

$ *sudo ./tcbuild build <build_type> [arch_type] [device_type]*

c.) [arch_type] and [device_type] position can be interchange without any effect.

$ *sudo ./tcbuild build <build_type> <module_name> [device_type] [arch_type]*

$ *sudo ./tcbuild build <build_type> <module_name> [arch_type] [device_type]*

Both commands are same

TCT Source Code Install and TCT-Manager Execution

Install TCT Module by RPM approach

[ ] : mandatory field
< > : optional field

1) Go to project root directory : <api> directory
2) Run installation command at project root location:

   $ *sudo ./tcbuild install <build_type> <module_name> [arch_type] [device_type]* (for target install)

   $ *sudo ./tcbuildsdk install <build_type> <module_name> [arch_type] [device_type]* (for emulator install)

   [build_type] = "itc/ctc/utc"

   [arch_type] = "armv7l/aarch64" (for tcbuild) and "i586/x86_64" (for tcbuildsdk)

   [device_type] = "mobile/wearable/tv/common_iot"

3) Execution of installed rpm modules on tizen_3.0 tct-mgr : TC Execution Guide

NOTE:

a.) To install all the packages (itc+ctc+utc for all modules),

$ *sudo ./tcbuild install [arch_type] [device_type]*

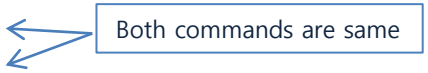
b.) To install all the packages for specific build_type, (itc or ctc or utc),

$ *sudo ./tcbuild install <build_type> [arch_type] [device_type]*

c.) [arch_type] and [device_type] position can be interchange without any effect.

$ *sudo ./tcbuild install <build_type> <module_name> [device_type] [arch_type]*

$ *sudo ./tcbuild install <build_type> <module_name> [arch_type] [device_type]*



TPK TC Guide

Prerequisite

■ Install SDK and Build sample tpk

1) SDK

A. Install Tizen-3.0 SDK (Update Tizen SDK to Tizen Studio) in Linux PC (having tizen-studio folder at path "/home/<username>/tizen-studio")

B. Create security profile by the name "**test**" and Generate author certificate

- Tools > Certificate Manager

C. Create a Tizen empty project through Tizen IDE (at path "/home/<username>/workspace")

- New > Tizen Native Project > Select one in Template > Finish

D. Build tpk of <C.>

- Project Right Click > Build Signed Package

※ FAQ

1) scripts_tpk/tpk_create.sh: line 26: tizen: command not found

- Check whether tizen studio directory path of your linux PC is matching with below command in scripts_tpk/init.sh

sudo ln -sf $HOME/tizen-studio/

2) Default compiler is llvm. But if you want to change to gcc compiler, change below codes.

i) script_tpk/tpk_create.sh

ii) Find "COMPILER_TYPE=" and change to gcc

iii) sh scripts_tpk/init.sh

3) Signing... Exception in thread "main" java.lang.NoClassDefFoundError: org/eclipse/ui/plugin/AbstractUIPlugin

i) Check whether you create 'test' security file and build template app

ii) If i) step already done,

a) Remove workspace and workspace_<profile> directory

b) Do i) step again

Build/Install TCT Module by TPK Approach

Build & Install TCT Module by TPK approach

[ ] : mandatory field
< > : optional field

1) Go to project root directory : <api> directory
2) Run scripts_tpk/init.sh file : This will generate tpkbuild binary at root directory. This is common binary for device as well as emulator build.

   $ *./scripts_tpk/init.sh*
3) If SDK installation path is different than "/home/<username>/tizen-studio" then script_tpk/init.sh file is to be modified as per installed path.
4) This solution utilizes the source code present in "src" folder of tizen_3.0 branch and creates the corresponding tpk at path : "/home/<username>/workspace_<profile>/<module-name>/Debug"
5) Run build command at project root location: (Do not use 'sudo')

   $ *./tpkbuild build <build_type> <module-name> [arch_type] [device_type]* builds specified module

   $ *./tpkbuild build <build_type > [arch_type] [device_type]* builds all either ITC or CTC or UTC modules present in src folder

   $ *./tpkbuild build [arch_type] [device_type]* builds all (ITC and CTC and UTC) modules present in src folder

   <build_type> = "itc/ctc/utc"

   [arch_type] = "armv7l/aarch64" (for device) and "i586/x86_64" (for emulator)

   [device_type] = "mobile/wearable"
6) Run install command at project root location:

   $ *sudo ./tpkbuild install <build_type> <module-name> [arch_type] [device_type]* installs specified module

   $ *sudo ./tpkbuild install <build_type > [arch_type] [device_type]* installs all either ITC or CTC or UTC modules present in src folder

   $ *sudo ./tpkbuild install [arch_type] [device_type]* installs all (ITC and CTC and UTC) modules present in src folder

   <build_type> = "itc/ctc/utc"

   [arch_type] = "armv7l/aarch64" (for device) and "i586/x86_64" (for emulator)

   [device_type] = "mobile/wearable"

   TCT zip file is located in "opt/tct/tizen_native_3.0/packages/[device_type]"

7) Execution of installed tpk modules on tizen_3.0 tct-mgr : TC Execution Guide

Make spec.sh file for tpk build (1/2)

Contents of spec.sh file

- The section from **spec** file ((in rpm code, /packaging/itc/<module>.spec file)) in **TCT** Directory of the module which copies some files to 'data' Or some tpk installation Or smack label set etc., is added in **spec.sh** file (which resides in "scripts_tpk" folder) accordingly as required for specific modules.

- E.g "media-content" UTC

```
mkdir -p %{buildroot}%{APP_PATH}%{name}/data
cp src/itc/media-content/TestImage.jpg %{buildroot}%{APP_PATH}%{name}/data/TestImage.jpg
cp src/itc/media-content/TestImage.jpg %{buildroot}%{APP_PATH}%{name}/data/Remove.jpg
cp src/itc/media-content/TestAudio.mp3 %{buildroot}%{APP_PATH}%{name}/data/TestAudio.mp3
mkdir -p %{buildroot}%{APP_PATH}%{name}/data/Images
cp src/itc/media-content/TestImage.jpg %{buildroot}%{APP_PATH}%{name}/data/Images/TestImage.jpg
cp src/itc/media-content/TestImage.jpg %{buildroot}%{APP_PATH}%{name}/data/Images/BookMarkImage.jpg
```

Spec file in rpm code /packaging/itc/<module>/native-media-content-itc.spec file

```
"org.tizen.media-content-native-itc")
        echo "Installing pre-requisites for the package $1"
        chsmack -a "User::App::Shared" ../../data
        chsmack -e "User::App::Shared" ../../data
        mkdir -p ../../data
        mkdir -p ../../data/Images
        mkdir -p ../../data/TestFolder
        mkdir -p ../../data/Videos
        mkdir -p ../../data/Music
        cp TestImage.jpg ../../data/TestImage.jpg
        cp TestImage.jpg ../../data/Remove.jpg
        cp TestAudio.mp3 ../../data/TestAudio.mp3
        cp TestImage.jpg ../../data/Images/TestImage.jpg
        cp TestImage.jpg ../../data/Images/BookMarkImage.jpg
```

Spec.sh file in scripts_tpk folder for media-contet module

Make spec.sh file for tpk build (2/2)

Contents of spec.sh file

- If "post-inst.sh" file exists in rpm code for any module (e.g in rpm code, /src/itc/<module>/post-install/ post-inst.sh) then contents of this file should be added in 'spec.sh' file

- E.g "media-content" UTC

```
if [ $MODE == "inst" ]; then
    echo "Installing pre-requisites for the package $PKG_NAME"
    mkdir -p $DEVICE_PHYSICAL_STORAGE_30
    mkdir -p $DEVICE_PHYSICAL_STORAGE_30/TestFolder
    mkdir -p $DEVICE_PHYSICAL_STORAGE_30/Images
    mkdir -p $DEVICE_PHYSICAL_STORAGE_30/Videos
    mkdir -p $DEVICE_PHYSICAL_STORAGE_30/Music
    cp $APP_DIR/$PKG_NAME/data/TestImage.jpg $DEVICE_PHYSICAL_STORAGE_30/TestFolder/TestImage.jpg
    cp $APP_DIR/$PKG_NAME/data/TestImage.jpg $DEVICE_PHYSICAL_STORAGE_30/TestImage.jpg
    cp $APP_DIR/$PKG_NAME/data/TestImage.jpg $DEVICE_PHYSICAL_STORAGE_30/Remove.jpg
    cp $APP_DIR/$PKG_NAME/data/TestAudio.mp3 $DEVICE_PHYSICAL_STORAGE_30/TestAudio.mp3
```

Post-inst.sh file in rpm code src/itc/media-content/post-install/post-inst.sh file

```
mkdir -p $DEVICE_PHYSICAL_STORAGE_30
mkdir -p $DEVICE_PHYSICAL_STORAGE_30/TestFolder
mkdir -p $DEVICE_PHYSICAL_STORAGE_30/Images
mkdir -p $DEVICE_PHYSICAL_STORAGE_30/Videos
mkdir -p $DEVICE_PHYSICAL_STORAGE_30/Music
cp $APP_DATA_DIR/TestImage.jpg $DEVICE_PHYSICAL_STORAGE_30/TestFolder/TestImage.jpg
cp $APP_DATA_DIR/TestImage.jpg $DEVICE_PHYSICAL_STORAGE_30/TestImage.jpg
cp $APP_DATA_DIR/TestImage.jpg $DEVICE_PHYSICAL_STORAGE_30/Remove.jpg
cp $APP_DATA_DIR/TestAudio.mp3 $DEVICE_PHYSICAL_STORAGE_30/TestAudio.mp3
```

Spec.sh file in scripts_tpk folder for media-contet module

RPM code hard-code handling in tpk_create.sh file (1/3)

If there is any hardcoded APP_ID or Path used in rpm code, then it is required to change this value in case of TPK approach as in case of TPK, APP_ID and Path is different than rpm. E.g.

Case 1: APP_ID in source code

#define BADGE_PACKAGE "native.badge-itc" (File: api/src/itc/badge/ITs-badge-common.h)

In case of TPK, package_id/app_id is: org.tizen.badge-native-itc

So, in tpk_create.sh file, it is handled as:

```
if [ $MODULE_NAME == "badge" ]; then
    if [ "$5" == "utc" ]; then
        sed -i -e 's/core.badge-tests/org.tizen.badge-native-utc/g' $1/src/utc-badge.c
    elif [ "$5" == "itc" ]; then
        sed -i -e 's/native.badge-itc/org.tizen.badge-native-itc/g' $1/inc/ITs-badge-common.h
    fi
fi
```

So, when "ITs-badge-common.h" file will be copied to /workspace/module/inc folder, package id will be :

#define BADGE_PACKAGE "org.tizen.badge-native-itc"

RPM code hard-code handling in tpk_create.sh file (2/3)

If there is any hardcoded APP_ID or Path used in rpm code, then it is required to change this value in case of TPK approach as in case of TPK, APP_ID and Path is different than rpm. E.g.

Case 2: Path changes

#define ICON_PATH "/usr/apps/core-accounts-svc-tests/shared/res/account.png" (File: utc-accounts-svc.c)

In case of TPK, path should be: /opt/home/owner/apps_rw/org.tizen.accounts-svc-native-utc/shared/res/account.png"

So, in tpk_create.sh file, it is handled as:

```
if [ $MODULE_NAME == "accounts-svc" ]; then
    if [ "$5" == "utc" ]; then
        sed -i -e 's:/usr/apps/core-accounts-svc-tests/:/opt/home/owner/apps_rw/org.tizen.accounts-svc-native-utc/:g' $1/src/utc-accounts-svc.c
        sed -i -e 's/core.accounts-svc-tests/org.tizen.accounts-svc-native-utc/g' $1/src/utc-accounts-svc.c
    elif [ "$5" == "itc" ]; then
        sed -i -e 's:/usr/apps/native-accounts-svc-itc/:/opt/home/owner/apps_rw/org.tizen.accounts-svc-native-itc/:g' $1/inc/ITs-accounts-svc-common.h
        sed -i -e 's/native.accounts-svc-itc/org.tizen.accounts-svc-native-itc/g' $1/inc/ITs-accounts-svc-common.h
    fi
fi
```

So, when "utc-accounts-svc.c" file will be copied to /workspace/module/src folder, package id will be :

#define ICON_PATH "/opt/home/owner/apps_rw/org.tizen.accounts-svc-native-utc/shared/res/account.png"

RPM code hard-code handling in tpk_create.sh file (1/3)

If there is any hardcoded APP_ID or Path used in rpm code, then it is required to change this value in case of TPK approach as in case of TPK, APP_ID and Path is different than rpm. E.g.

Case 3: APP_ID change in tizen-manifest file

In UTC "accounts-svc" module, app_id is core.accounts-svc-tests in core-accounts-svc-tests.xml file in rpm code.

But in case of TPK, app_id should be "org.tizen.accounts-svc-native-utc"

```
#Change in xml file for specific modules
if [ $MODULE_NAME == "accounts-svc" ]; then
    if [ "$5" == "utc" ]; then
        sed -i -e 's/core.accounts-svc-tests/org.tizen.accounts-svc-native-utc/g' tizen-manifest.xml
    elif [ "$5" == "itc" ]; then
        sed -i -e 's/native.accounts-svc-itc/org.tizen.accounts-svc-native-itc/g' tizen-manifest.xml
    fi
fi
```

Tpk_create.sh file handling for app_id in xml file

```
<privileges>
        <privilege>http://tizen.org/privilege/account.read</privilege>
        <privilege>http://tizen.org/privilege/account.write</privilege>
</privileges>
<account>
    <account-provider appid="core.accounts-svc-tests" providerid="http:/
        <icon section="account">account.png</icon>
        <icon section="account-small">account_small.png</icon>
```

App_id in Xml file in rpm code

```
<privileges>
<privilege>http://tizen.org/privilege/account.read</privilege>
<privilege>http://tizen.org/privilege/account.write</privilege>
</privileges>
<account>
<account-provider appid="org.tizen.accounts-svc-native-utc" provi
<icon section="account">account.png</icon>
```

App_id in Tizen-manifest xml file in workspace

Test Case Guide

Test Case Method – Black Box Testing

Black Box Testing

- Method of software testing that examines the functionality of an application without peering into its internal structures or workings
- Specific knowledge of the application's code/internal structure and programming knowledge in general is not required. The tester is aware of *what* the software is supposed to do but is not aware of *how* it does it.

Test Cases

- Test cases are built around specifications and requirements. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters.
- The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

TCT

- TCT MUST be developed by Black Box Testing method.
- TCT MUST be used only Public API and Public Enumeration and include Public Header.

Add Privileges

Add Privileges in manifest file

- Define the privileges required by your test app in xml file
- xml Location
  - packaging/utc/core-<MODULE_NAME>-tests.xml
  - packaging/itc/native<MODULE_NAME>-itc.xml
  - packaging/ctc/native<MODULE_NAME>-ctc.xml
- Note
  - Must add only public privilege in TCT
  - Avoid typo mistake (this can cause installation issue of the tct binary over device)
  - Remove unused privilege

Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
    .......
  <privileges>
        <privilege>http://tizen.org/privilege/externalstorage</privilege>
        <privilege>http://tizen.org/privilege/mediastorage</privilege>
        <privilege>http://tizen.org/privilege/content.write</privilege>
  </privileges>
</manifest>
```

Startup() / Cleanup()

Annotation Rule & Naming Convention

startup()

```
/**
 * @function      <TC_TYPE>_<MODULE_NAME>_startup
 * @description   Called before each test
 * @parameter     NA
 * @return        NA
 */
void <TC_TYPE>_<MODULE_NAME>_startup(void)
{
    /* pre-condition of each TC */
}
```

cleanup()

```
/**
 * @function      <TC_TYPE>_<MODULE_NAME>_cleanup
 * @description   Called after each test
 * @parameter     NA
 * @return        NA
 */
void <TC_TYPE>_<MODULE_NAME>_cleanup(void)
{
    /* post-condition of each TC */
}
```

Note

- Startup/Cleanup function Must return void type
- These routines run before/after each TC execution
- Should hold common routines of each test cases

example

wrong

```
int <TC_TYPE>_<MODULE_NAME>_startup(void)
{
    /* pre-condition of each TC */
    return 1;
}
```

wrong

Test Case

Annotation Rule

UTC

```
/**
 * @testcase utc_<MODULE_NAME>_<API_NAME>_p
 * @since_tizen
 * @description
*/
int utc_<MODULE_NAME>_<API_NAME>_p(void)
{
    /* Positive TC*/
    return 0;
}
```

ITC

```
/**
* @testcase  ITc_<Mixed_API_NAME>_p
* @since_tizen
* @type
* @description
* @scenario
* @apicovered
* @passcase
* @failcase
* @precondition
* @postcondition
*/
int ITc_<Mixed_API_NAME>_p(void)
{
    /* Positive TC*/
    return 0;
}
```

Naming Convention

- Add at least one positive TC and one negative TC for each API

TC Type	Positive TC	Negative TC
UTC	utc_<MODULE_NAME>_<API_NAME>_p utc_<MODULE_NAME>_<API_NAME>_p2, p3, p4 ...	utc_<MODULE_NAME>_<API_NAME>_n utc_<MODULE_NAME>_<API_NAME>_n2, n3, n4 ...
ITC/CTC	ITc/CTc_<Mixed_API_NAME>_p ITc/CTc_<Mixed_API_NAME>_p2, p3, p4 ...	-

Verdict Check

- **You SHOULD use assert macros to evaluate the result of each API**
- **Assert macros (Ensure that test case performs all cleanup if assert fails)**
  - Defined in src/common/assert.h for UTC

assert_eq(var, ref)	- Check if var == ref, print both values otherwise
assert_neq(var, ref)	- Check if var != ref, print both values otherwise
assert_gt(var, ref)	- Check if var > ref, print both values otherwise
assert_geq(var, ref)	- Check if var >= ref, print both values otherwise
assert_lt(var, ref)	- Check if var < ref, print both values otherwise
assert_leq(var, ref)	- Check if var <= ref, print both values otherwise
assert(exp)	- Check if exp != NULL, print exp otherwise

  - Defined in src/common/tct_common.h for ITC/CTC

PRINT_RESULT(eCompare, eRetVal, API, Error)	- Check if eCompare == eRetval, print otherwise
PRINT_RESULT_NORETURN(eCompare, eRetVal, API, Error)	- Check if eCompare == eRetval, print but not exit otherwise
PRINT_RESULT_CLEANUP(eCompare, eRetVal, API, Error, FreeResource)	- Check if eCompare == eRetval, print and free otherwise
CHECK_VALUE_STRING(StringVariable, API)	- Check if StringVariable != NULL, print otherwise
CHECK_VALUE_INT(Variable, API)	- Check if Variable > 0, print otherwise
CHECK_HANDLE(Handle, API)	- Check if Handle != NULL, print otherwise

Note: Please Ensure that test case performs all cleanup if asserts fails.

Feature Check

- If there is any related feature required by api then feature checking routine must be added before calling API

Example

```
void utc_iotcon_startup(void)
{
    ret = system_info_get_platform_bool("http://tizen.org/feature/iot.oic",
&g_feature);

    if (SYSTEM_INFO_ERROR_NONE != ret) {
        fprintf(stderr, "system_info_get_platform_bool() Fail(%d)", ret);
        return;
    }

    if (true == g_feature) {
        ret = iotcon_connect();
        if (IOTCON_ERROR_NONE != ret){
            fprintf(stderr, " iotcon_connect() Fail(%d)", ret);
            return;
        }
    }
}
```

If some feature is required by all TCs in <module_name>.c then feature check routine should be added in startup() function.
If not, you should add it into each TC related with feature

Just in case of supported feature, we can expect that API is working well

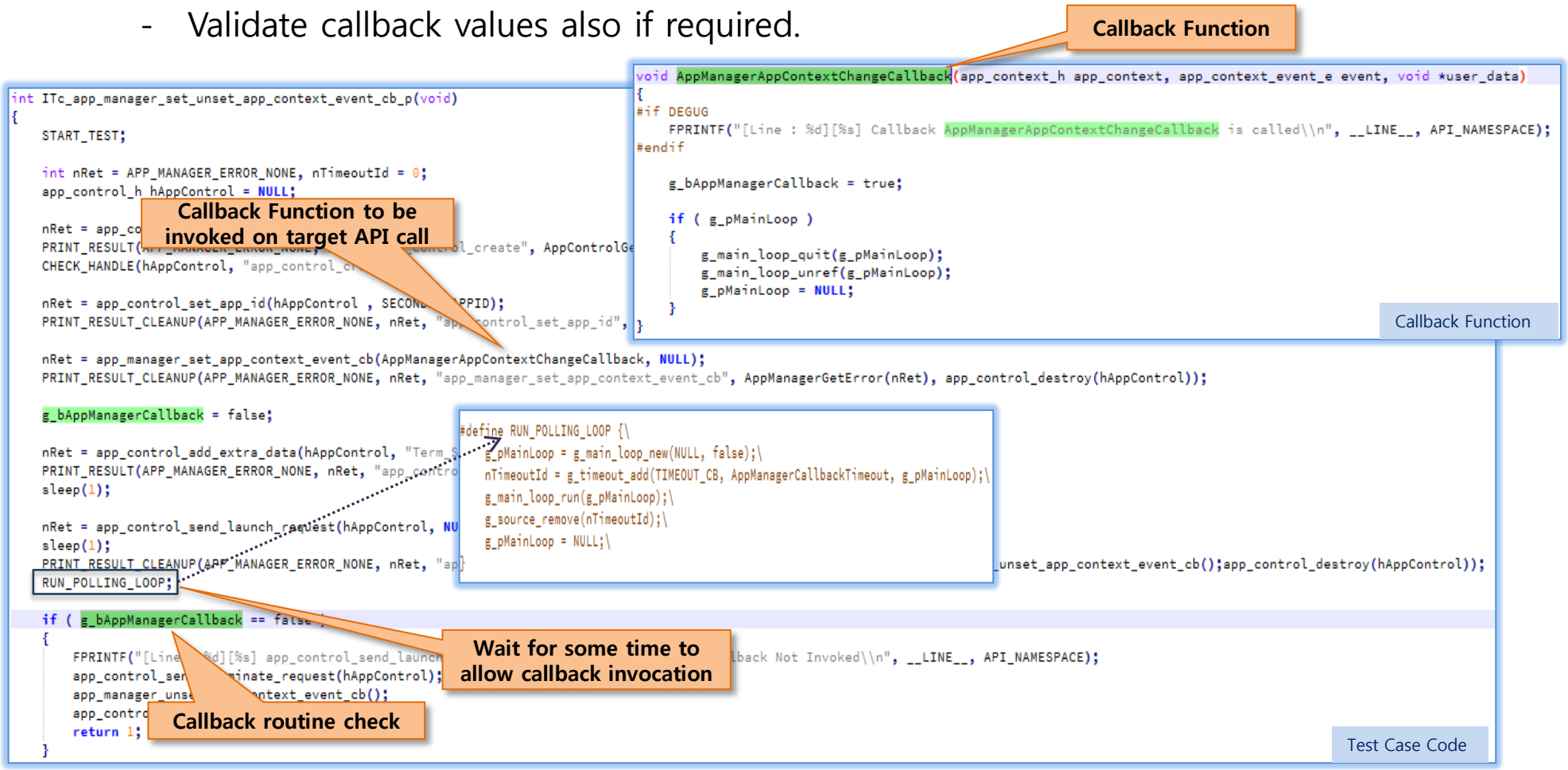
```
int utc_iotcon_get_timeout_p(void)
{
   if (false == g_feature) {
       ret = iotcon_get_timeout(NULL);
       assert_eq(ret, IOTCON_ERROR_NOT_SUPPORTED);
       return 0;
   }

   ret = iotcon_get_timeout(timeout);
```

If not supported, we should check whether return value from each API is ERROR_NOT_SUPPRTED

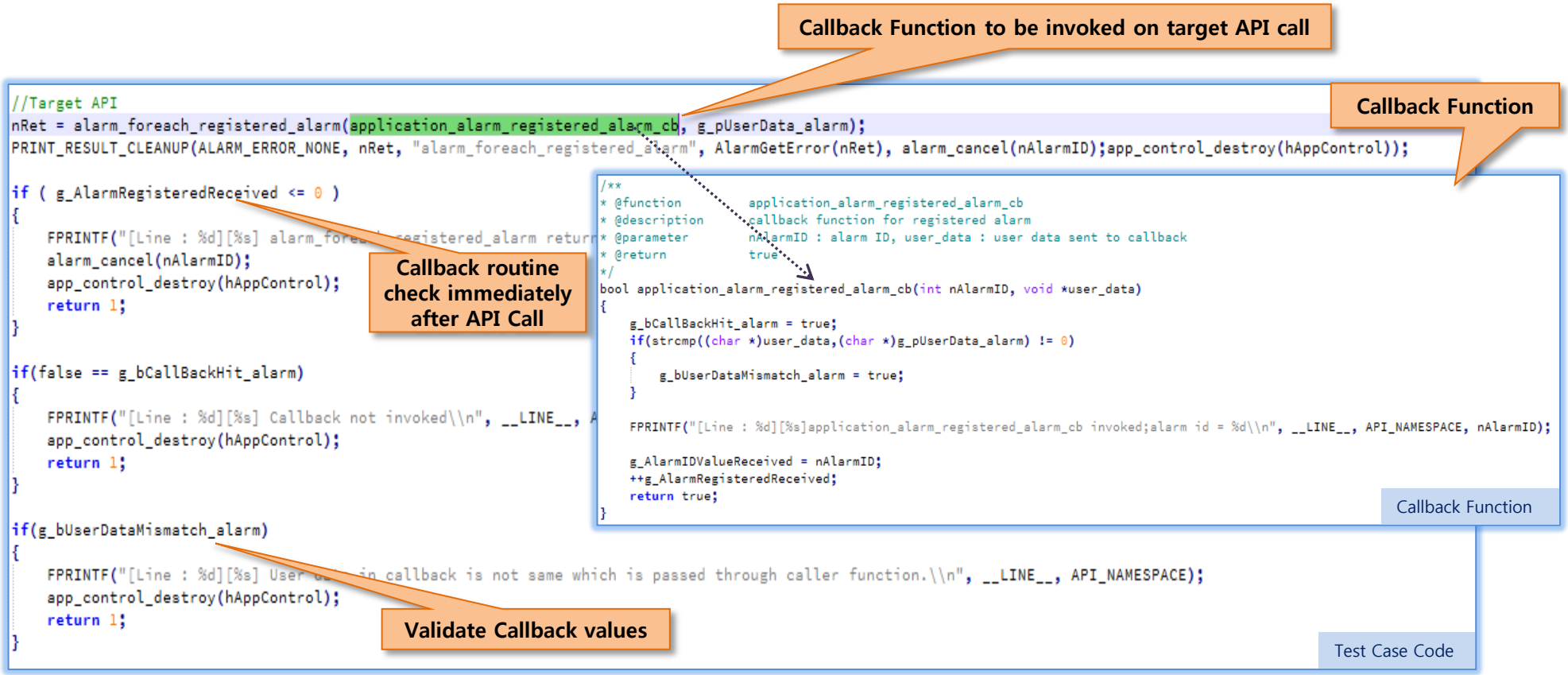
Callback Routine Check (1/2)

- **Asynchronous Callback : If API invokes Asynchronous callback, Must add Callback checking routine.**
  - Callback is invoked after API call returns.
  - Need to wait for some proper time values after API call for callback hit.
  - Validate callback values also if required.



Callback Routine Check (2/2)

- **Synchronous Callback : If API invokes Synchronous callback, Must add Callback checking routine.**
  - Callback is invoked before API call returns.
  - No delay is required to wait for the callback after API has been called.
  - Validate callback values also if required.




```
//Target API
nRet = alarm_foreach_registered_alarm(application_alarm_registered_alarm_cb, g_pUserData_alarm);
PRINT_RESULT_CLEANUP(ALARM_ERROR_NONE, nRet, "alarm_foreach_registered_alarm", AlarmGetError(nRet), alarm_cancel(nAlarmID);app_control_destroy(hAppControl));

if ( g_AlarmRegisteredReceived <= 0 )
{
    FPRINTF("[Line : %d][%s] alarm_foreach_registered_alarm return
    alarm_cancel(nAlarmID);
    app_control_destroy(hAppControl);
    return 1;
}

if(false == g_bCallBackHit_alarm)
{
    FPRINTF("[Line : %d][%s] Callback not invoked\\n", __LINE__, A
    app_control_destroy(hAppControl);
    return 1;
}

if(g_bUserDataMismatch_alarm)
{
    FPRINTF("[Line : %d][%s] User da in callback is not same which is passed through caller function.\\n", __LINE__, API_NAMESPACE);
    app_control_destroy(hAppControl);
    return 1;
}
```

```
/**
* @function         application_alarm_registered_alarm_cb
* @description      callback function for registered alarm
* @parameter        nAlarmID : alarm ID, user_data : user data sent to callback
* @return           true
*/
bool application_alarm_registered_alarm_cb(int nAlarmID, void *user_data)
{
    g_bCallBackHit_alarm = true;
    if(strcmp((char *)user_data,(char *)g_pUserData_alarm) != 0)
    {
        g_bUserDataMismatch_alarm = true;
    }

    FPRINTF("[Line : %d][%s]application_alarm_registered_alarm_cb invoked;alarm id = %d\\n", __LINE__, API_NAMESPACE, nAlarmID);

    g_AlarmIDValueReceived = nAlarmID;
    ++g_AlarmRegisteredReceived;
    return true;
}
```

Proper Clean Up Process for Each TC

Use of Pair APIs :

- Pair APIs like Create/Destroy, Connect/Disconnect Must Not be used in single.
- Test Case must call pairing APIs at its proper locations so that Platform should maintain its original state after each Test Case execution.

```
int ITc_application_ui_app_add
{
    START_TEST;
    app_event_handler_h event_
    //Target API
    int nRet = ui_app_add_event_
    PRINT_RESULT(APP_ERROR_NONE,
    // Give some sleep between cre
    usleep(2000);
    //Target API
    nRet = ui_app_remove_event_handler(event_handler);
    PRINT_RESULT(APP_ERROR_NONE, nRet, "ui_app_remove_event_handler", AppGetError(nRet));
    return 0;
}
```

API call to Add event handler

API call to Remove event handler

Remove Testing Files/Data After TC Execution

- If Test Case needs any testing files/data, then it Must be created and then removed after Test Case Execution gets over.

```
%post
mkdir -p /home/owner/content/TestFolder
cp %{APP_PATH}%{name}/data/TestImage.jpg /home/owner/content/TestFolder/TestImage.jpg
```

Test case File Added for TC Execution

Entry in Spec Files

```
%postun
rm -rf /home/owner/content/TestFolder
```

Test case File Removed during Uninstallation

API call in its Valid State

Call API in its Valid State when developing Positive Test Cases:

- If API needs to be called in certain state, that state Must be ensured before making API call.

```
int ITc_player_capture_video_p(void)
{
    START_TEST;

    player_state_e state;

    char pPath[PATH_LEN] = {0,};
    if ( false == PlayerAppendToAppDataPath(MEDIA_PATH_VIDEO2, pPath))
    {
        FPRINTF("[Line : %d][%s] unable to get the app data path\\n", __LINE__, API_NAMESPACE);
        return 1;
    }

    int nRet = player_set_display(g_player, PLAYER_DISPLAY_TYPE_EVAS, g_pEvasObject);
    PRINT_RESULT(PLAYER_ERROR_NONE, nRet, "player_set_display", PlayerGetError(nRet));

    sleep(2);

    nRet = player_set_uri(g_player, pPath);
    PRINT_RESULT(PLAYER_ERROR_NONE, nRet, "player_set_uri", PlayerGetError(nRet));

    nRet = player_prepare(g_player);
    PRINT_RESULT(PLAYER_ERROR_NONE, nRet, "player_prepare", PlayerGetError(nRet));

    nRet = player_start(g_player);
    PRINT_RESULT_CLEANUP(PLAYER_ERROR_NONE, nRet, "player_start", PlayerGetError(nRet), player_unprepare(g_player));

    nRet = player_get_state(g_player,&state);
    PRINT_RESULT_CLEANUP(PLAYER_ERROR_NONE, nRet, "player_get_state", PlayerGetError(nRet), player_stop(g_player);player_unprepare(g_player));

    PlayerGetState(state);
    if ( state != PLAYER_STATE_PLAYING )
    {
        FPRINTF("[Line : %d][%s] Player_start() call", __LINE__, API_NAMESPACE);
        player_stop(g_player);
        player_unprepare(g_player);
        return 1;
    }

    sleep(4);

    nRet = player_pause(g_player);
    PRINT_RESULT_CLEANUP(PLAYER_ERROR_NONE, nRet, "player_pause", PlayerGetError(nRet), player_stop(g_player);player_unprepare(g_player));
```

Check Player for 'Playing' State before calling Pause

Pause should be called only when Player State is 'Playing'

Avoid TC Crash and Memory Leak

Pointer/Handle Null Value Check:

- Before Using Handle Value, it Must be Null Check
- Before accessing Pointer/String values, it Must be Null Check

```
if ( hInputBuff != NULL )
{
    media_packet_destroy(hInputBuff);
    hInputBuff = NULL;
}
if ( hFormat != NULL)
{
    media_format_unref(hFormat);
    hFormat = NULL;
}
```

Null Check before Calling APIs over Handle

Free all Allocated memory:

- Any memory allocated by Test Case Must be freed after using it.
- If any API call specifies to do free operation by caller (through free API or any release API call), then it Must get freed by test case.

```
nRet = storage_get_directory(nStorageID, eStorageDirectory, &pszFilePath);
if ( nRet != STORAGE_ERROR_NONE && nRet != STORAGE_ERROR_NOT_SUPPORTED )
{
    FPRINTF("[Line : %d][%s] storage_get_direct
    return false;
}
if ( nRet == STORAGE_ERROR_NOT_SUPPORTED )
{
    FPRINTF("[Line : %d][%s] Storage Not Supported\\n", __LINE__, API_NAMESPACE);
    FREE_MEMORY(pszFilePath);
    return true;
}
```

```
#define FREE_MEMORY(buffer) {\
    if ( buffer != NULL )\
{\
    free(buffer);\
    buffer = NULL;\
}\
}
```

storage_get_directory API call provide allocated string value. This need to be free by test case

Free Allocated string

No Specific Device Binary or Reference App Dependency

No Specific Device Binary Dependency:

- Test Case Must Not be specific to any particular device binary.
- Test Case Must be developed in the way that it should remain valid for all the device binaries of specific device category (mobile / wearable / tv / common_iot).
- Test Case Must be developed by considering all cases since test results might be different by HW dependency.

No Reference App Dependency:

- There is no conformity that reference application (like gallery) would be existing in the target device, so there Must Not be any test case in the TCT which should be dependent over the Reference Application..
- If test case needs dependency over any sample application (like to test API which launch application), then it Must use its own sample tpk package. The sample tpk package Must be uninstalled and removed after test case execution.

TC Execution Guide

Install TCT-MGR Tool

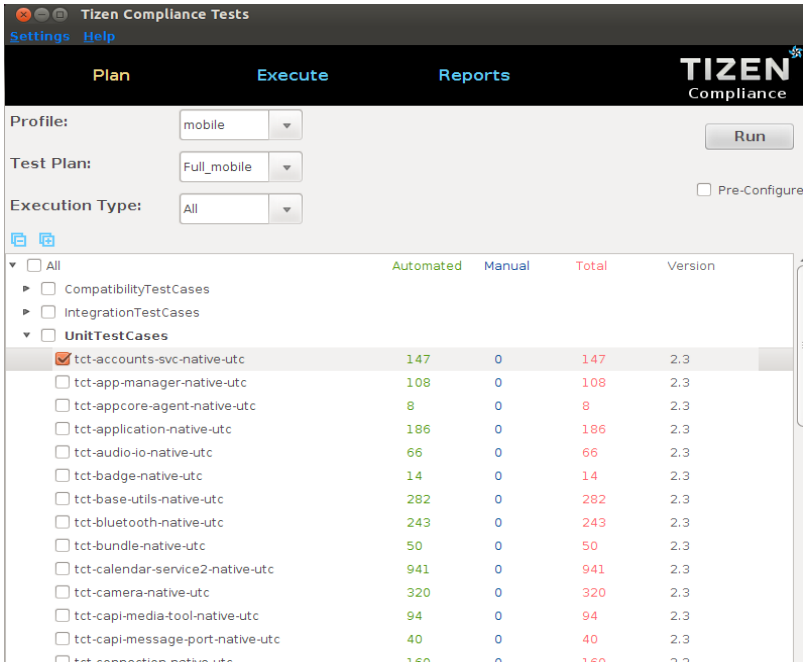
- **Prerequisite**
  - Python
  - Java (1.6 and above version)
- **Download Tool**
  - Public Git Path (review.tizen.org) : test/tct/native/api
  - Branch : tizen_3.0
  - api/tool/NativeTCT_3.0.tar.gz
- **Install tct-mgr on host device**
  - Extract NativeTCT_3.0.tar.gz
  - sudo python tct-setup.py
- **Install TCT backend runner on target/emulator**
  - sudo python /opt/tct/tizen_native_3.0/scripts/tct-config-device.py

Run TCs

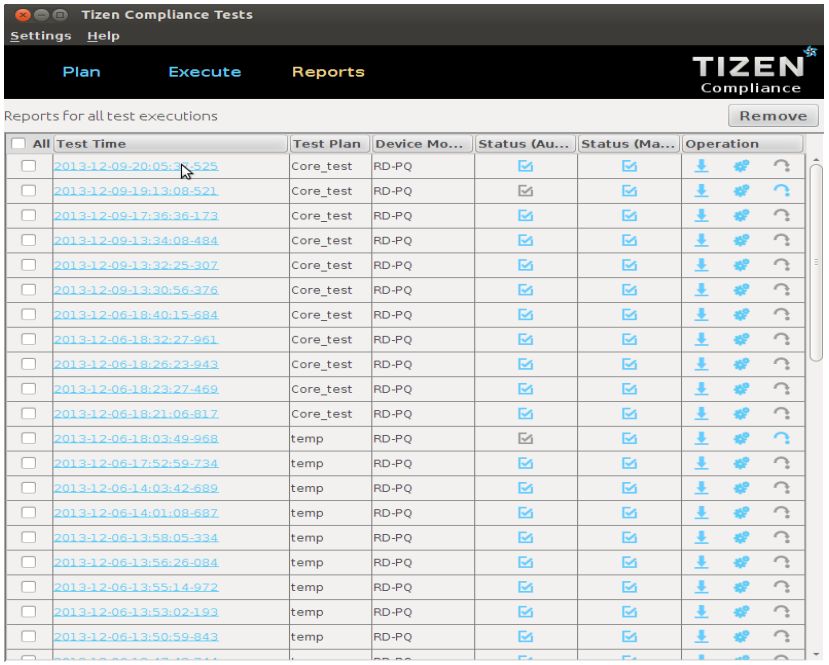
- **You can run TCs using Core-TCT**

```
$ tct-mgr
```

- You will find your module has been added under “UnitTestCases” category
- To run a test-suite, select a module and click “Run” button



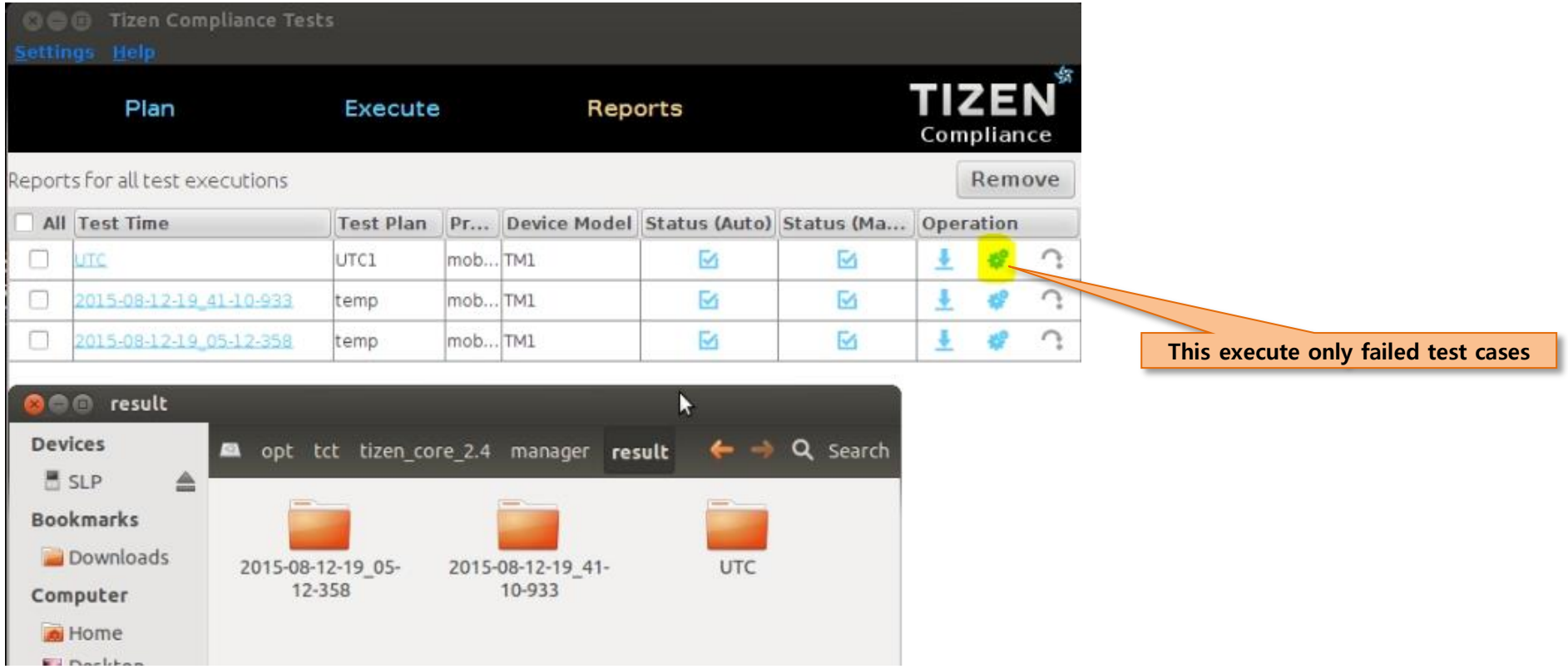
Plan Tab



Reports Tab

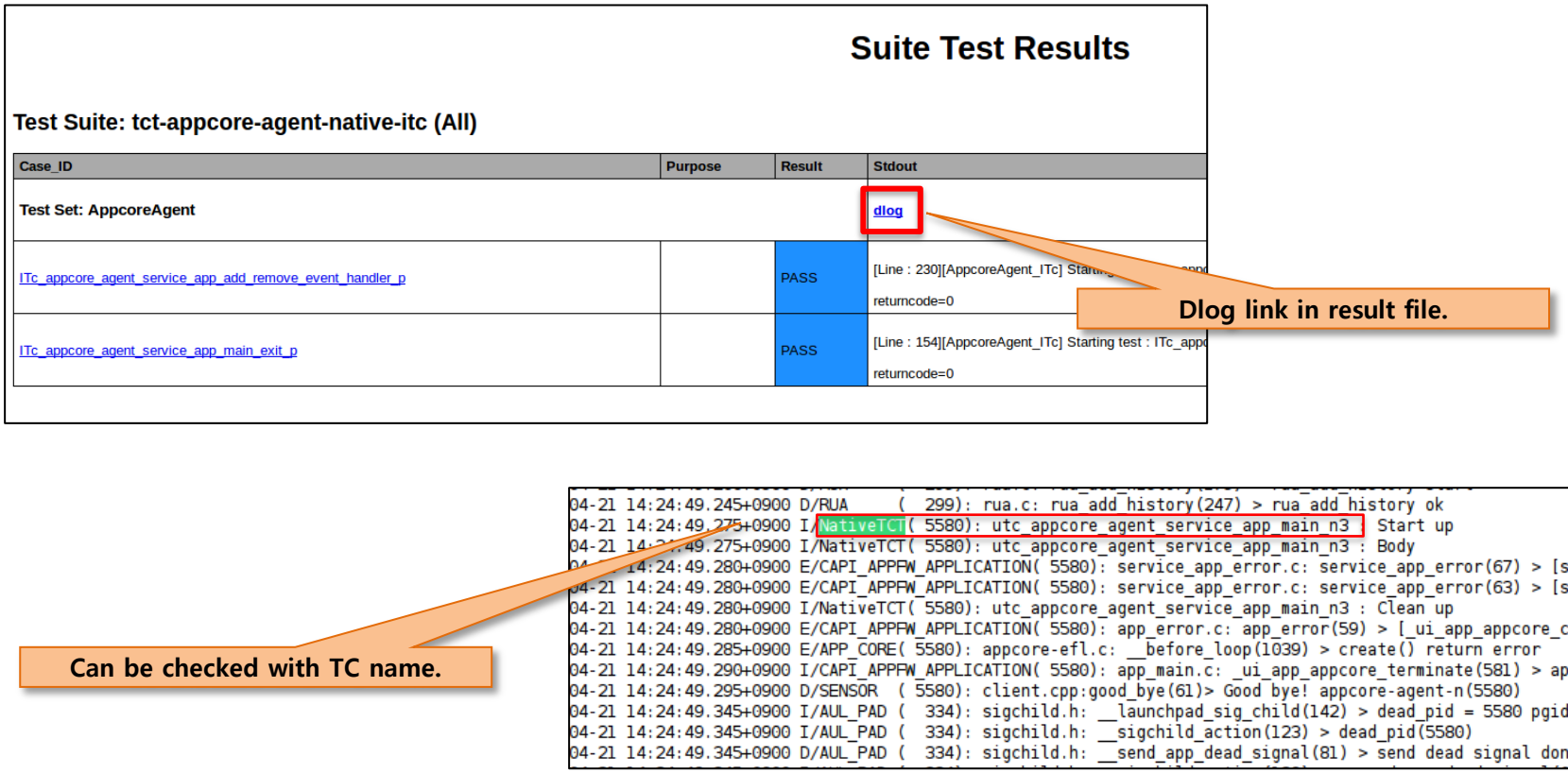
Simple guide to execute only failed TCs

1. Install TCT Binary (using steps as mentioned in earlier slide 'Build & Install TCs').
2. Copy TCT result over host folder location: /opt/tct/tizen_native_3.0/manager/result
3. Run TCT-Manager tool (select tizen_3.0) and go to Reports.
4. Press failed TC execution button of the TCT result which you had copied in previous step. This will execute only failed TCs which is reported in TCT result.



Get dlog information during execution

1. Dlog is printed in result file during execution.



Suite Test Results

Test Suite: tct-appcore-agent-native-itc (All)

Case_ID	Purpose	Result	Stdout
Test Set: AppcoreAgent			dlog
ITc_appcore_agent_service_app_add_remove_event_handler_p		PASS	[Line : 230][AppcoreAgent_ITc] Starting test : ITc_appc returncode=0
ITc_appcore_agent_service_app_main_exit_p		PASS	[Line : 154][AppcoreAgent_ITc] Starting test : ITc_appc returncode=0

Dlog link in result file.

Can be checked with TC name.

```
04-21 14:24:49.245+0900 D/RUA     (  299): rua.c: rua_add_history(247) > rua_add_history ok
04-21 14:24:49.275+0900 I/NativeTCT( 5580): utc_appcore_agent_service_app_main_n3 : Start up
04-21 14:24:49.275+0900 I/NativeTCT( 5580): utc_appcore_agent_service_app_main_n3 : Body
04-21 14:24:49.280+0900 E/CAPI_APPFW_APPLICATION( 5580): service_app_error.c: service_app_error(67) > [ser
04-21 14:24:49.280+0900 E/CAPI_APPFW_APPLICATION( 5580): service_app_error.c: service_app_error(63) > [ser
04-21 14:24:49.280+0900 I/NativeTCT( 5580): utc_appcore_agent_service_app_main_n3 : Clean up
04-21 14:24:49.280+0900 E/CAPI_APPFW_APPLICATION( 5580): app_error.c: app_error(59) > [_ui_app_appcore_cre
04-21 14:24:49.285+0900 E/APP_CORE( 5580): appcore-efl.c: __before_loop(1039) > create() return error
04-21 14:24:49.290+0900 I/CAPI_APPFW_APPLICATION( 5580): app_main.c: _ui_app_appcore_terminate(581) > app_
04-21 14:24:49.295+0900 D/SENSOR  ( 5580): client.cpp:good_bye(61)> Good bye! appcore-agent-n(5580)
04-21 14:24:49.345+0900 I/AUL_PAD (  334): sigchild.h: __launchpad_sig_child(142) > dead_pid = 5580 pgid =
04-21 14:24:49.345+0900 I/AUL_PAD (  334): sigchild.h: __sigchild_action(123) > dead_pid(5580)
04-21 14:24:49.345+0900 D/AUL_PAD (  334): sigchild.h: __send_app_dead_signal(81) > send dead signal done
```

Switching between multi-users for TCT execution

Edit TCT_CONFIG file (/opt/tools/TCT_CONFIG) at host system to switch among multi-users for TCT execution.

```
[DEVICE]
DEVICE_SUITE_TARGET_24=/opt/usr/media
DEVICE_SUITE_TARGET_30=/home/owner/share
DEVICE_USER_30=/home/owner
DEVICE_STORAGE_30=/home/owner/content
DEVICE_EXECUTION_MODE_30=owner
```

TCT_CONFIG for execution using 'owner' as user.

To switch to different user, replace 'owner' with another 'user' name in this file.

For example, for TCT execution with user name 'tct', TCT_CONFIG should be changed as mention below.

```
[DEVICE]
DEVICE_SUITE_TARGET_24=/opt/usr/media
DEVICE_SUITE_TARGET_30=/home/tct/share
DEVICE_USER_30=/home/tct
DEVICE_STORAGE_30=/home/tct/content
DEVICE_EXECUTION_MODE_30=tct
```

TCT_CONFIG for execution using 'tct' as user name.

■ Description for TCT_CONFIG file

1) During Execution, the TCT-Manager tool copies '/opt/tools/TCT_CONFIG' file at host system to target device location ('/tmp/TCT_CONFIG').
2) The test cases are executed based on the key value pair mentioned in TCT_CONFIG file.

```
[DEVICE]
DEVICE_SUITE_TARGET_24=/opt/usr/media
DEVICE_SUITE_TARGET_30=/home/owner/share
DEVICE_USER_30=/home/owner
DEVICE_STORAGE_30=/home/owner/content
DEVICE_EXECUTION_MODE_30=owner
```

DEVICE_SUITE_TARGET_30 : 'tct' install directory. If tct zip package has 'res' folder then it gets copied at this location. If 'res' contains tpk files then it gets installed.

DEVICE_USER_30 : User directory name. Needed by specific modules only.

DEVICE_STORAGE_30 : Internal storage directory location. This location is same which "storage_foreach_device_supported" API provides for "STORAGE_TYPE_INTERNAL".

DEVICE_EXECUTION_MODE_30 : pkgcmd (to install tpk) and app_launcher (to launch tct binary) commands execution mode.

※ Appendix – known issue 01

- **You should install below Python packages.**

```
$ sudo apt-get install rpm2cpio

$ sudo apt-get install tree

$ sudo apt-get install timeout

$ sudo apt-get install python-pip

$ sudo apt-get install python-support

$ sudo apt-get install python-requests

$ sudo apt-get install python-setuptools
```

※ Appendix – Multi-Target Execution

1. Connect Several devices to 1 Host PC.

- All devices should be flashed with same tizen binary.

2. Generate Plan (Only at first time)

- Run tct-mgr and select the packages what you want to run. Click 'Run' button and create new plan.

3. Set preconditions

- $ sudo /opt/tct/tizen_native_3.0/scripts/tct-config-device.py --deviceid={devid} (Need to be done for each device)
- Set TC_Config.txt (Need to be done for each device) : $ sdb pull /tmp/TC_Config.txt
- Set precondition in TC_Config.txt and Push to target : $ sdb push /tmp/TC_Config.txt

4. Run tct-shell command.

- $ sudo tct-shell -p {plan file} --tizen-version tizen_native_3.0 --distribute --disable --log DEBUG (Only one time needed)

※ --output {path} : If you want to give result path, set with this command.

Default result path : /opt/tct/tizen_native_3.0/shell/result

Ex) sudo tct-shell –p Full_mobile_plan.xml --tizen-version tizen_native_3.0 --distribute --disable --log DEBUG --output ~/Desktop

Coverage Measurement Guide

Prerequisite

TCT Information

1) UTC in TCT

A. Public Git Path : test/tct/native/api
B. Branch : tizen_3.0
C. Directory : tct/src/utc/[package_name]

TBT Information

1) TBT

A. Public Git Path : test/tct/native/behavior
B. Branch : tizen_3.0_mobile
C. Directory : behavior/tbtcoreapp/

Coverage Measurement using Gcov (1/4)

■ Building Framework Packages (for gcno data) and Configuring Target

1) Modify CMakefile.txt or packaging/*.spec or Makefile.am of CAPI Pkg to enable gcov instrumentation

A. Modification in CMakefile.txt file

i. SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${EXTRA_CFLAGS} -fPIC -Wall -Werror -fprofile-arcs -ftest-coverage")

B. Modification in spec file

i. export CXXFLAGS = "-fprofile-arcs -ftest-coverage"

ii. export LDFLAGS = "-lgcov"

C. Modification in Makefile.am file (example of cairo pkg)

i. AM_CPPFLAGS = -I$(srcdir) $(CAIRO_CFLAGS) ₩

-fprofile-arcs ₩

-ftest-coverage

ii. AM_LDFLAGS = $(CAIRO_LDFLAGS) ₩

-lgcov

2) Build CAPI Pkg

A. home] gbs build -A armv7l --include-all

3) Install Pkg to Target and Find gcov data file(source_file_name.gcno) in local build root

A. Install gcov enabled pkg to Target

i. Location : /GBS-ROOT/local/repos/armv7l/RPMS/[pkg_name].rpm

B. Find gcov data file in local build root

i. /GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0/home/abuild/rpmbuild/BUILD/[capi-pkg-name-version]/CMakeFiles/[capi-pkg-name].dir/src/[source_file_name].gcno

Coverage Measurement using Gcov (2/4)

■ Executing Test Cases (to generate gcda data)

1) Code Coverage For TCT (using TCT-Manager)

i) Build and install TCT Package

A. Build UTC

i. home] sudo ./tcbuild build utc [pkg-name] <arch_type> <device_type>

RPM Build location : GBS-ROOT-TCT-<device_type>/local/repos/device/< arch_type >/RPMS/

You can find core-[pkg-name]-tests-0.1-0.armv7l.rpm

B. Install UTC RPMs for Coverage

i. home] sudo ./tcbuild install_coverage utc [pkg-name] <arch_type> <device_type>

This will generate tct binary packages for Coverage

ii) Run TC using TCT-Manager

A. Run TCT-Manager tool.

B. Select 'Tizen Ver' field as "tizen_native_3.0" in TCT-Manager. This will display the package in category under "UnitTestCases".

C. Select the package and Execute test cases.

Note:

On executing the test cases inside the package for multiple times, the coverage data will keep appending to the gcda file on each run.
So, its better to remove "/tmp/home/abuild/rpmbuild/BUILD/[pkg-name]" folder location inside target before starting fresh Execution.

Coverage Measurement using Gcov (3/4)

■ Executing Test Cases (to generate gcda data) continued…

2) Code Coverage For TBT (If you need TBT coverage then do this else skip this)

i) Modify TBT source code to add support for coverage

A. Set gcda file location using 'setenv' function at the start of the main function

```
int main(void)
{
    int ret;

    //setting gcda file location for coverage
    setenv("GCOV_PREFIX","/tmp",1);
```

B. Add 'ui_app_exit()' API call inside '_app_destroy_cb' function of your module's view file. This will cause application to exit when you come out of that module on pressing 'back button'. This is important because application should exit gracefully to create gcda data. You can use 'ui_app_exit' at any other suitable location also as per your need.

```
static void _app_destroy_cb(void* this)
{
    RETM_IF(NULL == this, "data is NULL");

    runtimeinfo_view *view = NULL;
    view = (runtimeinfo_view*)this;
    RETM_IF(NULL == view, "view is NULL");

    SAFE_DELETE(view->view);
    SAFE_DELETE(view);

    ui_app_exit();
}
```

'ui_app_exit' inside '_app_destroy_cb' of tbt-runtimeinfo-view.c

ii) Build and Run TBT (tbtcoreapp) using tizen sdk as Tizen Native Application.

iii) Do Manual Test cases of your module and then Exit the application (using point B as mention above). gcda will be generated on application exit.

3) After TCT/TBT execution, check gcda data file on target

gcda files Location : /tmp/home/abuild/rpmbuild/BUILD/[pkg-name]/xxx/xxx/src/[source_file_name].gcda
(gcda file location can vary slightly inside "/tmp/home/abuild/rpmbuild/BUILD", depending on the platform source code folder hierarchy

Coverage Measurement using Gcov (4/4)

■ Extracting gcov line coverage data

1) Pull gcov data to local build root

A. In target : sh-4.1#] cp /tmp/home/abuild/rpmbuild/BUILD/pkg-name/xxx/xxx/src/*.gcda /tmp

B. In GBS-ROOT directory

i. /GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0/home/abuild/rpmbuild/BUILD/[capi-pkg-name-version]/CMakeFiles/[capi-pkg-name].dir/src/] sdb pull /tmp/*.gcda

ii. Matching .gcda file location according to .gcno file location

2) Extracting Coverage Data

A. Install lcov in scratch box

i. Go to ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0 directory

ii. xxx] Copy attached 'lcov-1.11-1.noarch.rpm' to this location

or sudo wget http://downloads.sourceforge.net/ltp/lcov-1.11-1.noarch.rpm

iii. xxx] sudo chroot ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0

iv. xxx] rpm -ivh --force --nodeps lcov-1.11-1.noarch.rpm

v. xxx] cd /home/abuild/xxx/xxx/xxx/CmakeFiles/

vi. xxx] lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx.info

vii. xxx] genhtml capi-xxx-xxx.info -o out

viii. open index.html in out directory

Coverage Measurement For Daemon Process (1/4)

■ Configuring Target (1/2)

1) Modify CMakefile.txt or packaging/*.spec or Makefile.am of Daemon Pkg to enable gcov instrumentation

A. Modification in CMakefile.txt file

i. SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${EXTRA_CFLAGS} -fPIC -Wall -Werror -fprofile-arcs -ftest-coverage")

B. Modification in spec file

i. export CXXFLAGS = "-fprofile-arcs -ftest-coverage"

ii. export LDFLAGS = "-lgcov"

C. Modification in Makefile.am file (example of cairo pkg)

i. AM_CPPFLAGS = -I$(srcdir) $(CAIRO_CFLAGS) ₩

-fprofile-arcs ₩

-ftest-coverage

ii. AM_LDFLAGS = $(CAIRO_LDFLAGS) ₩

-lgcov

2) Modify daemon source files

A. Modification in **main function** of daemon process: Set the gcda file path location to '/tmp' directory inside target.

```
int main(void)
{
    int ret;

    //setting gcda file location for coverage
    setenv("GCOV_PREFIX","/tmp",1);
```

B. Modification in daemon source files **API functions** to dump the coverage to gcda file : Use "__gcov_flush();" API call.

```
__gcov_flush();
```

Note:

'__gcov_flush' will dump past coverage data accumulated till this call. So, its good idea to use '__gcov_flush' at common or multiple hit locations so that coverage data will get dumping regularly to gcda file.

Coverage Measurement For Daemon Process (2/4)

■ Configuring Target (2/2)

3) Build Daemon Pkg

A. home] gbs build -A armv7l --include-all

4) Install Pkg to Target and Find gcov data file (source_file_name.gcno) in local build root

A. Install gcov enabled pkg to Target

i. Location : /GBS-ROOT/local/repos/armv7l/RPMS/[pkg_name].rpm

B. Find gcov data file in local build root

i. /GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0/home/abuild/rpmbuild/BUILD/[daemon-pkg-name-version]/daemon/CMakeFiles/[daemon-pkg-name].dir/src/[source_file_name].gcno

Note:

gcno file locations can vary slightly depending on the platform folder hierarchy.

It's good idea to use '*find . –name "*.gcno*"' command to track the correct location for all the gcno files.

Coverage Measurement For Daemon Process (3/4)

■ Executing TC in Target using TCT-Manager

1) Build and install TC Package

A. Build UTC

i. home] sudo ./tcbuild build utc [pkg-name] <arch_type> <device_type>

RPM Build location : GBS-ROOT-TCT-<device_type>/local/repos/device/< arch_type >/RPMS/

You can find core-[pkg-name]-tests-0.1-0.armv7l.rpm

B. Install UTC RPMs for Coverage

home] sudo ./tcbuild install_coverage utc [pkg-name] <arch_type> <device_type> (if tct coverage also needed)

Or,

home] sudo ./tcbuild install utc [pkg-name] <arch_type> <device_type> (if tct coverage not needed)

2) Run TC using TCT-Manager

A. Run TCT-Manager tool.

B. Select 'Tizen Ver' field as "tizen_native_3.0" in TCT-Manager. This will display the package in category under "UnitTestCases".

C. Select the package and Execute test cases.

3) Check gcov data file on target (after the package gets executed in TCT-Manager)

A. gcda files Location : /tmp/home/abuild/rpmbuild/BUILD/[pkg-name]/daemon/xxx/xxx/src/[source_file_name].gcda

Note:

1. To find the daemon coverage by test case execution, its good to first forcefully terminate the daemon process (use kill command) and then deletes the gcda files for daemon (inside "/tmp/home/abuild/rpmbuild/BUILD/"). This will remove past accumulated daemon coverage data.
2. gcda file location can vary slightly inside "/tmp/home/abuild/rpmbuild/BUILD", depending on the platform folder hierarchy.

Coverage Measurement For Daemon Process (4/4)

■ Extracting gcov line coverage data

1) Pull gcov data to local build root

A. In target : sh-4.1#] cp /tmp/home/abuild/rpmbuild/BUILD/pkg-name/daemon/xxx/xxx/src/*.gcda /tmp

B. In GBS-ROOT directory

i. /GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0/home/abuild/rpmbuild/BUILD/[daemon-pkg-name-version]/daemon/CMakeFiles/[daemon-pkg-name].dir/src/] sdb pull /tmp/*.gcda

ii. Matching .gcda file location according to .gcno file location

2) Extracting Coverage Data

A. Install lcov in scratch box

i. Go to ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0 directory

ii. xxx] sudo wget http://downloads.sourceforge.net/ltp/lcov-1.11-1.noarch.rpm

iii. xxx] sudo chroot ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0

iv. xxx] rpm -ivh --force --nodeps lcov-1.11.-1.noarch.rpm

v. xxx] cd /home/abuild/xxx/xxx/xxx/CmakeFiles/

vi. xxx] lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx.info

vii. xxx] genhtml capi-xxx-xxx.info -o out

viii. open index.html in out directory

Excluding Coverage from HTML (1/2)

■ Which cases?

Excluding Data	
Type	**Description**
Logs	If you want to remove log lines from coverage data, you can remove it
Not used function	If you want to remove not used functions that is not included APIs scope, you can remove it
Not supported feature	If target don't have feature, So, APIs can not running in target, you can remove related code
Not called Callback	In some cases, if you can't make H/W callback or system callback, you can remove it
System Error	Codes for environmental errors such as SMACK, Memory Leak, CPU, Low battery can be removed

■ How can I do?

- Please refer next page

Excluding Coverage from HTML (2/2)

■ Excluding Coverage from HTML report

- **Excluding file from HTML**
  - If you want to remove some files in coverage report data, please try the following OR please try as below
    - remove file.gcno, file.gcda, file.o files before runing lcov command
    - "lcov --remove capi-media-audio-io.info audio_io.c" command also remove file from coverage data but not updated on html
- **Excluding some lines from HTML**
  - If you want to remove some lines from file, please try as follow
    - add "//LCOV_EXCL_LINE"comment at the end of lines (ex. LOGE("test ₩n"); //LCOV_EXCL_LINE)
- **Excluding some block of codes from HTML**
  - If you want to remove some block of codes from file, please try as follow
    - add "//LCOV_EXCL_START"comment at the start or block and add "//LCOV_EXCL_STOP" comment at the end of block
- **Excluding LOG related line from C/CPP file**
  - Add following codes in C/CPP file

Excluding LOG

```
#ifdef LOGI
#undef LOGI
#endif

#ifdef LOGE
#undef LOGE
#endif

#define LOGI(format, arg...) \
        printf(format, ##arg); //LCOV_EXCL_LINE


#define LOGE(format, arg...) \
        printf(format, ##arg); //LCOV_EXCL_LINE
```

Excluding Block

```
int audio_in_prepare(audio_in_h input)
{
        //LCOV_EXCL_START
        AUDIO_IO_NULL_ARG_CHECK(input);
        audio_in_s *handle = (audio_in_s *) input;
        int ret = MM_ERROR_NONE;

        if (handle->is_async) {
                ret = mm_sound_pcm_capture_start_async(handle->mm_handle);
        } else {
                ret = mm_sound_pcm_capture_start(handle->mm_handle);
        }

        if (ret != MM_ERROR_NONE) {
                return __convert_audio_io_error_code(ret, (char*)__FUNCTION__);
        }

        LOGI("[%s] mm_sound_pcm_capture_start() success",__FUNCTION__);
        return AUDIO_IO_ERROR_NONE;
        //LCOV_EXCL_STOP
}
```

Analysis of gcov raw data (1/2)

■ General report of gcov data

- **index.html in out directory (result of 5 page)**
  - Summarize the function and line coverage for all source files in tizen package
    - Generates coverage data according to source folder hierarchy of tizen package
    - Click "src" in html then, you can find all coverage data of source code (c/cpp)

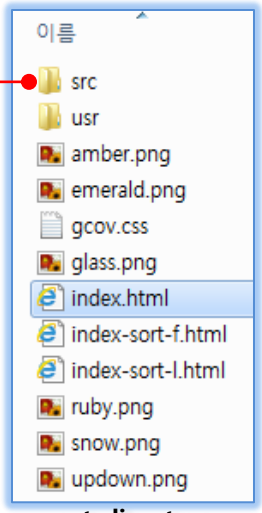
LCOV - code coverage report

Current view: top level
Test: device.info
Date: 2015-05-29 10:11:02

	Hit	Total	Coverage
Lines:	319	676	47.2 %
Functions:	36	68	52.9 %

Directory	Line Coverage		Functions	
/usr/include	50.0 %	1 / 2	-	0 / 0
/usr/include/bits	66.7 %	2 / 3	-	0 / 0
src	47.1 %	316 / 671	52.9 %	36 / 68

< index.html >



< out directory >

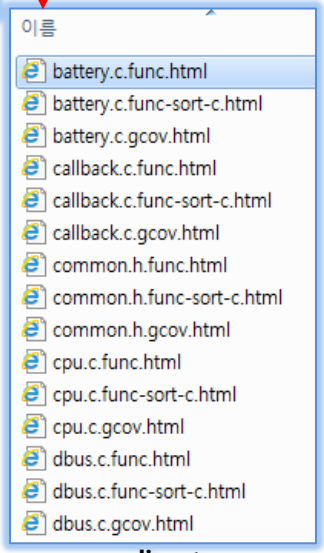
- **src directory**
  - Function coverage information for each files in src directory of tizen package
    - Click file named html then, you can find all function coverage data of file(c/cpp)

Current view: top level - src - battery.c (source / functions)
Test: device.info
Date: 2015-05-29 10:11:02

	Hit	Total
Lines:	24	29
Functions:	3	3

Function Name	Hit count
device_battery_get_level_status	3
device_battery_get_percent	3
device_battery_is_charging	3

< file_name_func.html >

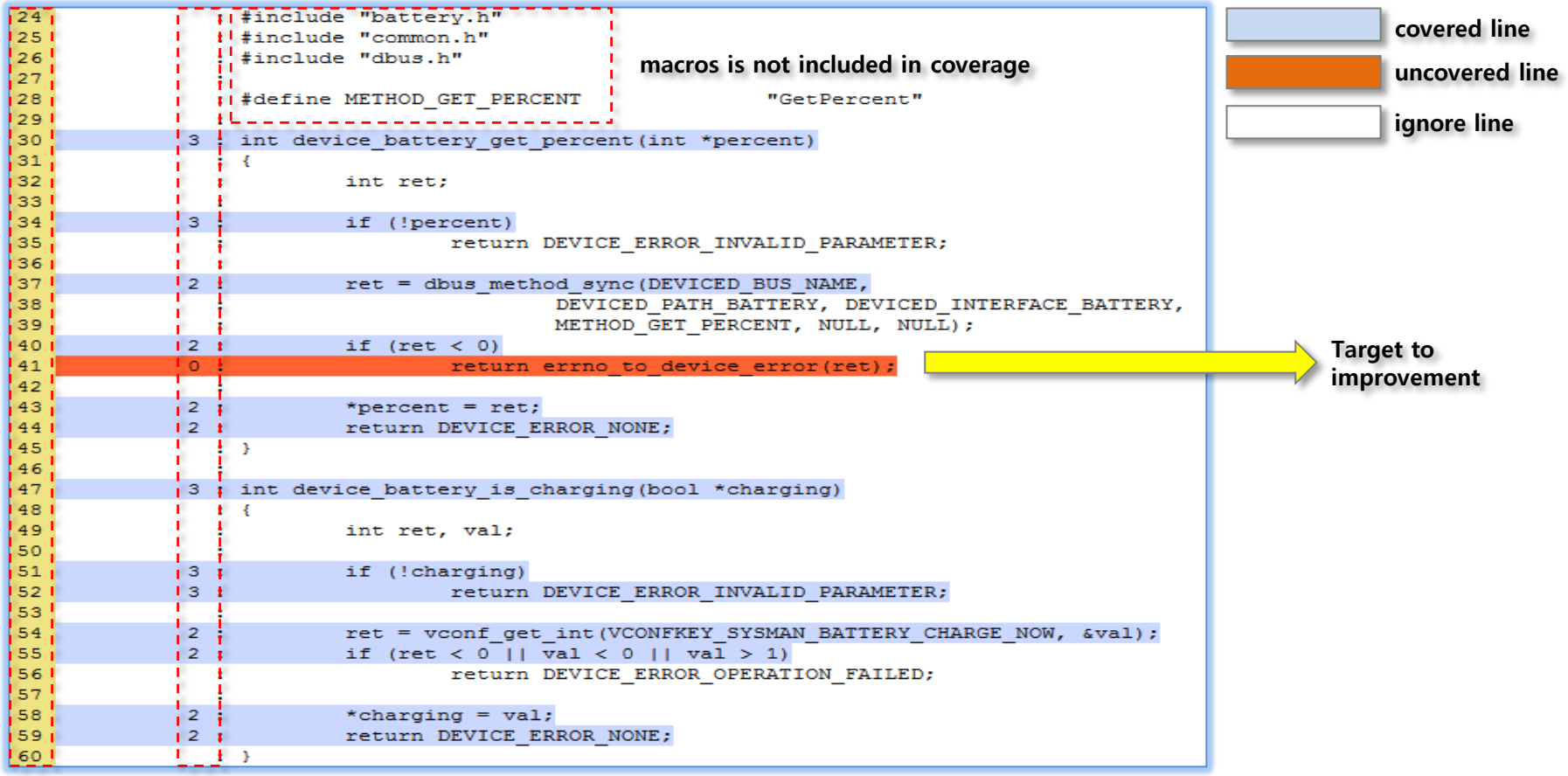


< src directory >

Analysis of gcov raw data (2/2)

■ Coverage data

- **If you click some file in html, you can find following coverage information**
  - index.html → "source directory" → "file_name.c/cpp"
  - To improve coverage rate, you should consider orange color line only



Merging Coverage : Framework And Daemon Process

General Approach to Extract Coverage Report for single module

i. Go to ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0 directory

ii. xxx] sudo wget http://downloads.sourceforge.net/ltp/lcov-1.11-1.noarch.rpm

iii. xxx] sudo chroot ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0

iv. xxx] rpm -ivh --force --nodeps lcov-1.11.-1.noarch.rpm

v. xxx] cd /home/abuild/xxx/xxx/xxx/CmakeFiles/

vi. xxx] lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx.info (this step will be different for merged approach)

vii. xxx] genhtml capi-xxx-xxx.info -o out (this step will be different for merged approach)

viii. open index.html in out directory

Approach to Merge Framework and Daemon Process Coverage Data

Follow steps (i) to (v) as mentioned above and then do followings:

1) Generate different coverage.info file for framework and daemon process (this step differs from step (vi) mentioned above)

A. lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx.info (generate info file for framework)

B. lcov -c -d daemon-xxx-xxx.dir/ -o daemon-xxx-xxx.info (generate info file for daemon process)

2) Merge .info files to generate common Coverage Report (this step differs from step (vii) mentioned above)

A. genhtml **capi-xxx-xxx.info daemon-xxx-xxx.info** -o out

B. open index.html in out directory and find merged coverage report.

Note:

'genhtml' command can read multiple info files one by one and generate common coverage report after processing all the .info files.

If you want to generate common coverage report for multiple modules, then you can use this merging approach to get single coverage report.

Merging Coverage : Mobile, Wearable and TV Coverage

If the source files for different device type (mobile wearable and tv, which you intend to merge) are same then you can merge the coverage from different device types to generate merged coverage report.

General Approach to Generate Coverage Report for single module

i. Go to ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0 directory

ii. xxx] sudo wget http://downloads.sourceforge.net/ltp/lcov-1.11-1.noarch.rpm

iii. xxx] sudo chroot ~/GBS-ROOT/local/BUILD-ROOTS/scratch.armv7l.0

iv. xxx] rpm -ivh --force --nodeps lcov-1.11.-1.noarch.rpm

v. xxx] cd /home/abuild/xxx/xxx/xxx/CmakeFiles/

vi. xxx] lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx.info

vii. xxx] genhtml capi-xxx-xxx.info -o out (this step will be different for merged approach)

viii. open index.html in out directory

Approach to Merge Coverage Data from different device types.

1) **Follow steps (i) to (vi) as mentioned above to generate different coverage.info files using their respective (gcda+gcno) data**

   A. lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx_mobile.info (mobile info file)

   B. lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx_wearable.info (wearable info file)

   C. lcov -c -d capi-xxx-xxx.dir/ -o capi-xxx-xxx_tv.info (tv info file)

2) **Merge .info files to generate merged coverage report**

   A. genhtml **capi-xxx-xxx_mobile.info capi-xxx-xxx_wearable.info capi-xxx-xxx_tv.info** -o out

   B. open index.html in out directory and find merged coverage report.

Note:

'genhtml' command can read multiple info files one by one and generate merged coverage report after processing all the .info files.

The coverage data keeps appending to the coverage report during each .info file processing, which generates merged coverage report finally.

GDB Debugging for TCT Guide

TCT Test Case Execution Under GDB Debugging (1/3)

■ Install gdb and dependent rpm binaries over target device

1) Download following rpm packages from *http://download.tizen.org/snapshots/tizen/mobile/latest/repos/<target>/packages/armv7l/*:

A. gdb-<version>.armv7l
B. gdb-devel-<version>.armv7l
C. gdb-docs-<version>.armv7l
D. gdb-server-<version>.armv7l
E. libgthread-<version>.armv7l
F. libpthread-stubs-<version>.armv7l
G. libpython-<version>.armv7l
H. python-<version>.armv7l
I. python-gobject-<version>.armv7l

2) Copy and Install the rpm packages to target device

A. Copy the rpm packages inside device at location : /home/owner/content/
B. Install the rpm binaries (rpm -ivh --force --nodeps <rpm path location>)

Note:

This is one time process to install gdb over the target device.

Once gdb gets installed, then don't follow this pre-requisite process.

TCT Test Case Execution Under GDB Debugging (2/3)

■ Build and Install TCT RPM Packages over Target Device

1) Build TCT package

home] sudo ./tcbuild build <itc/ctc/utc> [pkg-name] <arch_type> <device_type>

RPM Build location : GBS-ROOT-TCT-<device_type>/local/repos/device/< arch_type >/RPMS/

For example, *sudo ./tcbuild build itc runtime-info armv7l mobile* (if debugging for itc/runtime-info)

This will generate 3 rpm build binaries, native-runtime-info-itc-0.1-0.armv7l.rpm, native-runtime-info-itc-debuginfo-0.1-0.armv7l.rpm and native-runtime-info-itc-debugsource-0.1-0.armv7l.rpm

2) Install TCT package over target device

Copy all the three tct rpm packages inside device at location : /home/owner/content/

Install these rpm binaries over device (rpm -ivh --force --nodeps <rpm path location>)

3) Provide smack access and execute label permission to tct binary (to be done in root account mode)

tpk-backend --preload -y <pkg-name> (for example, tpk-backend --preload -y native-runtime-info-itc)

Change execute smack label permission of tct binary to "User::App::<app_id>"

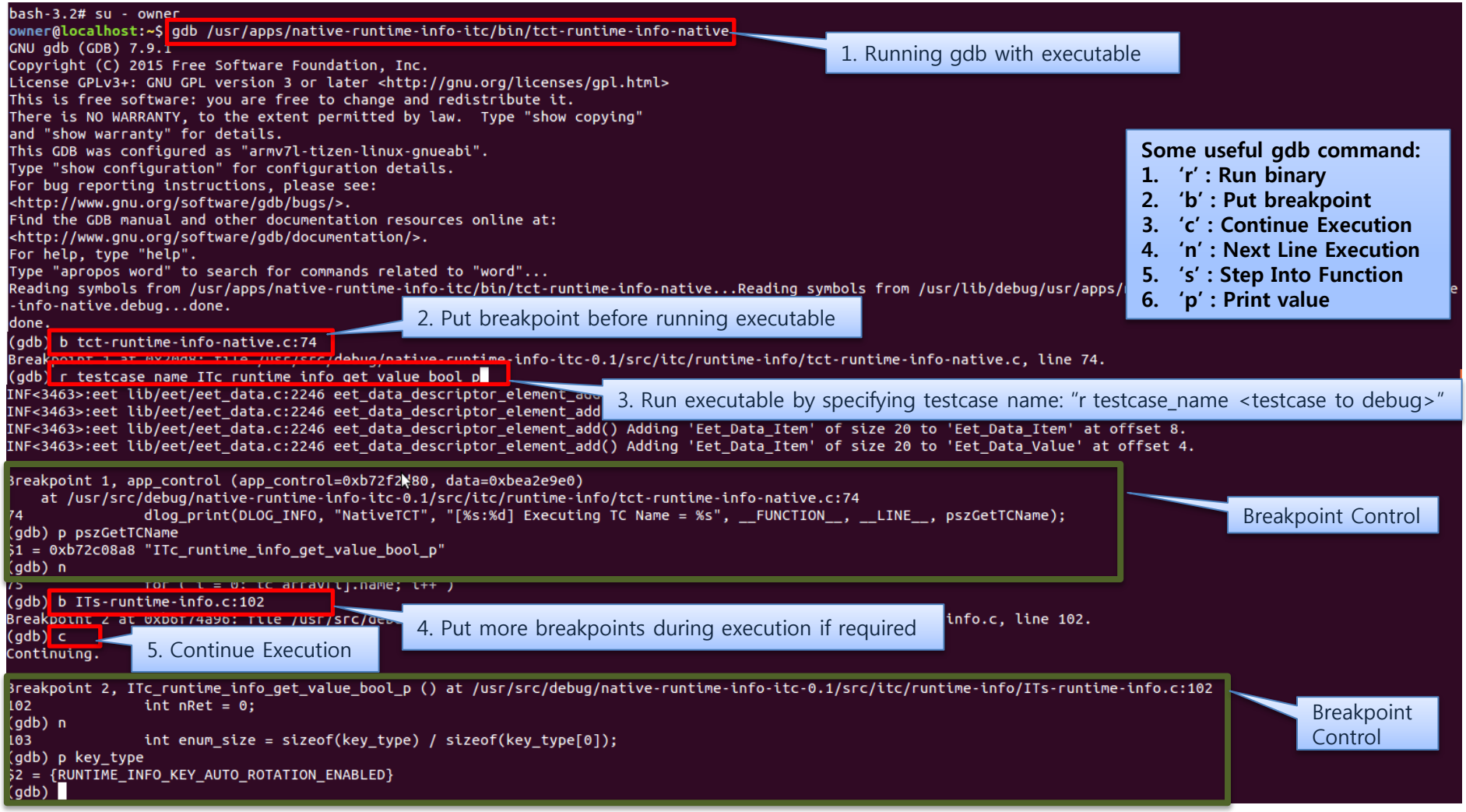
(for example chsmack -e "User::App::native.runtime-info-itc" /usr/apps/native-runtime-info-itc/bin/tct-runtime-info-native)

TCT Test Case Execution Under GDB Debugging (3/3)

1) **Login to target device under 'owner' mode.**

   sdb shell -> su - owner

2) **Run gdb with tct binary package (/usr/apps/<package-name>/bin/<executable name>)**



GDB Debugging With TCT Core Dump File (1/3)

■ Install gdb and dependent rpm binaries over target device

1) Download following rpm packages from *http://download.tizen.org/snapshots/tizen/mobile/latest/repos/<target>/packages/armv7l/*:

A. gdb-<version>.armv7l
B. gdb-devel-<version>.armv7l
C. gdb-docs-<version>.armv7l
D. gdb-server-<version>.armv7l
E. libgthread-<version>.armv7l
F. libpthread-stubs-<version>.armv7l
G. libpython-<version>.armv7l
H. python-<version>.armv7l
I. python-gobject-<version>.armv7l

2) Copy and Install the rpm packages to target device

A. Copy the rpm packages inside device at location : /home/owner/content/
B. Install the rpm binaries (rpm -ivh --force --nodeps <rpm path location>)

Note:

This is one time process to install gdb over the target device.

If gdb is already installed on the device, then don't follow this pre-requisite process.

GDB Debugging With TCT Core Dump File (2/3)

■ Build and Install TCT RPM Packages over Target Device

1) Build TCT package

home] sudo ./tcbuild build <itc/ctc/utc> [pkg-name] <arch_type> <device_type>

RPM Build location : GBS-ROOT-TCT-<device_type>/local/repos/device/< arch_type >/RPMS/

For example, *sudo ./tcbuild build ctc audio-io armv7l mobile* (for ctc/audio-io)

This will generate 3 rpm build binaries, native-audio-io-ctc-0.1-0.armv7l.rpm, native-audio-io-ctc-debuginfo-0.1-0.armv7l.rpm and native-audio-io-ctc-debugsource-0.1-0.armv7l.rpm

2) Install TCT package over target device

Copy all the three tct rpm packages inside device at location : /home/owner/content/

Install these rpm binaries over device (rpm -ivh --force --nodeps <rpm path location>)

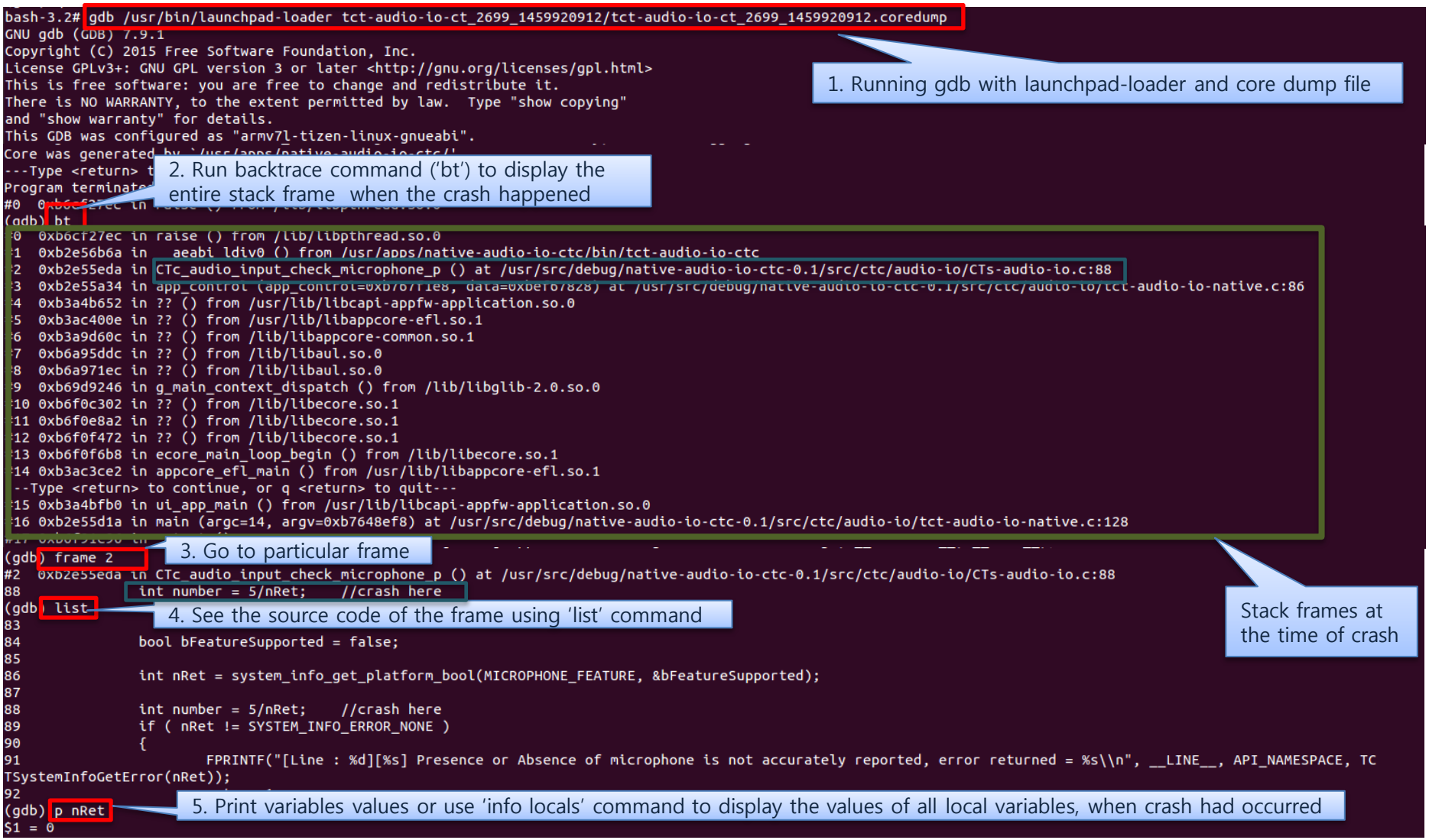
Note:

Build and Installation should be done only for the tct package which had generated the Core Dump file.

GDB Debugging With TCT Core Dump File (3/3)

1) **Login to target device in 'root' mode.**

2) **Run gdb with launchpad-loader (/usr/bin/launchpad-loader) and core dump file**



Thank you