UCF Template Module Development Guide v013023
Richard Todd
Test Engineering Services
Requires UniConsole(.exe) v012623 or later.

Table of Contents

1. Description

This document defines the structure of a UCF template module and how to customize the basic functionality. This document provides ‘best practices’ for customization.

This document is best reviewed in parallel with a UCF template module (TplModule.cpp) source file.

2. Terminology

Term	Definition
UCF	The UniConsole Framework
Management Console	The tab/text based console that is used to monitor and control all application functionality
Test Executive	The user interface used to automatically sequence all tests and save measurement data to a local database
./.	Path seperator syntax that is equivelant to .\. on Windows systems
UI	Collectively the Management Console and Test Executive user interfaces

3. The Template Module

The TplModule.h/cpp files define a template module that is used as the starting point for all UCF plug-in modules.

3.1. Module filetype

Modules are compiled from a C++ template file and are linked into a shared library (*.so for linux, *.dll for windows)

3.2. Module naming convention

Modules can be renamed to anything that is appropriate. The UCF loads libraries based on location in a predefined directories and not names.

3.3. Directory location for modules common to profile families

Modules placed in **./ucfroot/opt/<Profile Family Name>/CommonLibs/** will be loaded for each profile in the family.

3.4. Directory location for modules unique to a specified profile

Modules placed in **./ucfroot/opt/<Profile Family Name>/<Profile Name>/** will be loaded for the specified profile.

3.5. Directory location for modules that will always load

Modules placed in **./ucfroot/bin/<Release Name>/StaticModules/** will be loaded each time the UCF is started.

4. UCF Messaging

The template module contains a message handler with specific sections used to implement common test automation tasks.

4.1. Message handler function

The message handler is a function contained in the template module which is named as follows:

*TplModule::Initialize(iSharedLibFramework<iIXmgr>::ModuleStates level,iIXmgr *global_ix) { … }*

Where: ‘level’ is the message

4.2. Message index

The UCF initializes a module by sending a sequence of messages in a predefined order to the message handler in the template module.

Messages are sent sequentially from position #1 to #7.
For each message, its task and position in the initialization sequence is shown.

UCF Message	Position	Task	Reference
`GetInterfaces`	1	Import interfaces from the UCF framework	5.1
`StartModule`	2	Initialize custom code within the module	5.2
`MakeExternalConnections`	3	Create connections to external instrumentation (RS232, Ethernet, etc)	5.3
`Start Processes`	4	Initialize external test instrumentation	5.4
`Activate`	5	Activate / wake-up the test system	5.5
`SetToIdle`	6	Set the test system to an idle state	5.6
`Shutdown`	7**	Shutting down the test system	5.7

** This message is sent when the UCF application is closing

5. Message Definitions

The UCF sends messages to the template module on startup and shutdown. This section contains the definitions and purpose of each message.

5.1. GetInterfaces

This message notifies the module to import any interfaces from the UCF that are necessary for correct functionality.

a) Message Handler Code

```
case(state::GetInterfaces):
   /* Get the interfaces previously registered by the framework and other modules */

    SetBaseInterfaces(global_ix,ModuleId);
   /*  Configure standard/required interfaces */

    TheSystem=global_ix->iGet<iStdIOChannelSystem>(ModuleId, "Unable to find System interface");
    TheTestExec=global_ix->iGet<iTestUtility> (ModuleId);
     /*  Get interfaces to the Standard IO Module and Test Executive */
    break;
```

b) Purpose

The UCF operates using any number of modules that have no knowledge of each other other than virtual interfaces that are exported to the UCF.

This message is a notification that all other modules have exported their interfaces which can now be imported as needed.

c) An example

A template module may need resources contained in a system or test executive module and uses those resources by importing critical virtual interfaces.

5.2. StartModule

This message is a notification that all modules have exported all interfaces.

a) Message Handler Code

```
case(state::StartModule):
/* Initializations and/or configurations
    1.  configure stdouts out/sys/trace
    2.  register keyboard commands
    3.  register test steps
    */
    Start();
    * call a sub-function to do initializations */

    break;
```

b) Purpose

A module that imports interfaces can only be initialized after all modules have both exported / imported interfaces as necessary. This message is a notification that all imported interfaces can be used as needed.

c) Example

```
int TplModule::Start() {

    StdOutConsole.Configure(StdOut,StdOut->GetRegisteredConsoleHandle("StdOut"));
    StdSysConsole.Configure(StdOut,StdOut->GetRegisteredConsoleHandle("StdSys"));
    StdTraceConsole.Configure(StdOut,StdOut->GetRegisteredConsoleHandle("StdTrace"));
    /*  Configure the text output channels */

    Cmds->RegisterApplicationCmd    (
        "Command1",
        new Ucfx(&TplModule::ACommandLineDirective, this),
        "(param1,param2)\tA demo command line directive"
        );
        /*  Register a command line directive */

    return(1);
}
```

5.3. MakeExternalConnections

This message is a notification to create connections/handles to external instrumentation that will be controlled from this module.

a) Message Handler Code

```
case(state::MakeExternalConnections):
/*  Create connections to external instrumentation */

    SerialPort1=TheSystem->RegisterIOChannelDevice(1,"A generic RS232 channel")

    break;
```

b) Purpose

All Instrumentation connections are made here, before any instrument initializations are made to force and ‘early’ system failure to save debug and verification time

5.4. StartProcesses

This message is sent to notify the module that all internal configurations and connections have been made, and test system initialization can proceed.

a) Message Handler Code

```
case(state::StartProcesses):
/*  autostart any runt-ime processes */

    SetStaticConfiguration();
    /* Initiate short and simple configurations */

    startup_thread=std::thread(&TplModule::ThreadedStartup,this,global_ix);
    startup_thread.detach();
    /*  start lengthy external initializations or configurations in the ‘ThreadedStartup’ function */

    break;
```

b) Purpose

In cases where it is a lengthy process to initialize test instrumentation, the initialization is done in a seperate thread in order to maintain UI responsiveness (progress bar and text messages).

c) Example

Any and all exceptions that are thrown **MUST** be caught within this function.

```
void TplModule::ThreadedStartup(iIXmgr *ix) {
/*  Perform any system initializations or configurations on startup that might tie up the main thread of execution
    */
     try {
          TheAppEngine->IncrementActiveThreadCount();
          InitializeAndConfigureTheSystem(...);
          /*  An example call */
     }
     catch(IOString msg) {
          TheAppEngine->iErrorManager()->SetHardError(msg);
     }
     catch(...) {
          TheAppEngine->iErrorManager()->SetHardError("Unknown exception while initializing the system");
     }
     TheAppEngine->DecrementActiveThreadCount();
}
```

5.5. Activate

This optional message is sent at the end of the test system initialization state to bring it out of a ‘sleep’ state if/when necessary.

a) Message Handler Code

```
case(state::Activate):
/*  wake up an idle module
     1. bring any external systems out of a sleep state as required
     */
     ...
     break;
```

5.6. SetToIdle

This optional message is sent to set a test system to an idle state to begin testing.

a) Message Handler Code

```
case(state::SetToIdle):
/* set the module and external systems to an idle state */
     SetToIdle();
     break;
```

b) Example

Prior to starting a test sequence, the outputs of all system power supplies are disabled.

5.7. Shutdown

This message is sent when the UCF application is closed.

a) Message Handler Code

```
case(state::Shutdown):
/* set the module and external systems to an idle state */
     ...
     break;
```

b) Example

When the application is shutting down, do a software cleanup of any allocated system handles, etc.

6. Misc Template Module Configurations

6.1. Defining the module name and description

The module is named and described in the following declaration and function calls. This is the information that will be displayed on startup, and when invoking the 'ls-mod' / 'status' command.

```
TplModule       TheModule ("This is the name that will be displayed to identify this module");

const char* TplModule::GetDescription() {
/*  return a module description and build date to the framework */

    static IOString desc("This module implements a generic UniConsole module.\nV100122\nBuild date = ");
    desc+=__DATE__;

    return(desc);
}
```

6.2. Exporting interfaces to the UCF framework

Interfaces are exported to the UCF at the shared library entry point. A reference to the module is returned to the UCF framework.

Exporting virtual interfaces is beyond the scope of this guide.

```
iSharedLibFramework<iIXmgr>* GetEntryPoint(iIXmgr *local_ix,unsigned module_id) {
/*  Export any interfaces that need to be visible to the UCF framework */

    TheModule.ModuleId=module_id;
    local_ix
        ->iSet<iTestExec_1u>(&TheModule)
        ->iSet<iTestUtility>(&TheModule);

  return(&TheModule);
}
```