The following presentation contains examples of advanced 4Test coding techniques I developed while contracting at Merrill Lynch in New York City. The presentation focuses on real-world examples of how these advanced techniques can be used to simplify 4Test code and to improve test script readability, maintainability and reliability.

Segue Software Annual Users Conference
Orlando, Florida

Advanced 4Test Coding Techniques: Recursion, Threads and Critical Sections

Gerard Sczepura
President
Advanced Desktop Solutions Inc.

Objectives

- Become familiar with Recursion, Threads, Semaphores, Critical Resources and Critical Sections
- Learn how to apply these advanced techniques to enhance your test scripts
  - Simplify Code
  - Improve Readability and Maintainability
  - Improve Script Reliability

Recursion

- Mathematics
  - 5! = 5 x 4!
    5 x 4 x 3!
    5 x 4 x 3 x 2!
    5 x 4 x 3 x 2 x 1!

FACT(N) = N * FACT(N-1)

Recursion

- Acronyms
  - GNU: GNU's Not UNIX
- Algorithms
  - Binary Trees
- Puzzles
  - "Tower of Brahma"

Recursive Procedures

- Definition
  - A recursive procedure is one that calls itself from within the procedure body
  - A recursive call creates a second activation of the procedure, *during the lifetime of the first activation*

Recursive Procedures

- Advantages
  - Permits more lucid and concise descriptions of algorithms
  - Recursive procedures are generally easier to understand

Example 1: Problem

- Test data maintained in an Excel spreadsheet
- Data is saved as CSV file for test execution
- Spreadsheet contains field identifiers, 4Test object names, and one or more rows of input values

Example 1: Problem

- 4Test object names are atomic
  - For example, HtmlTextField5, HtmlPopupList1, etc.
- Object hierarchy is determined at runtime

Example 1: Solution

- GetObjectHierarchy function
  - Calls WindowChildren function
  - Loops through list of window children
  - Calls itself (recursion) when window class is BrowserChild or ExtendedBrowserChild

Example 1: Code


```
[+] void GetObjectHierarchy (WINDOW wMain, STRING sWin, out WINDOW wWinFound)
    [ ] INTEGER i
    [ ] WINDOW w
    [ ] LIST OF WINDOW lwChildren
    [ ]
    [ ] ResOpenList (">> Function::dtcc_GetObjectHierarchy ({wMain}, {sWin}, {wWinFound})")
    [ ]
    [ ] lwChildren = WindowChildren (wMain)
    [-] for each w in lwChildren
        [-] if (WindowIsDefined (w, sWin))
            [ ] wWinFound = w.@sWin
            [ ] Print ("FOUND IT!")
            [ ] break
        [-] else
            [-] if (ClassOf (w) == @"BrowserChild" || ClassOf (w) ==
                @"ExtendedBrowserChild")
                [ ] GetObjectHierarchy (w, sWin, wWinFound)
```

Example 1: Window Declaration

[+] window Product_Generic MunicipalTrade
- [ ] parent Browser
- [+] BrowserChild Top
  - [+] BrowserChild Branding
    - [+] HtmlImage HtmlImage1
    - [+] HtmlImage HtmlImage2
  - [+] BrowserChild ProdImage
    - [+] HtmlImage HtmlImage1
    - [+] HtmlHeading MunicipalTrade
- [-] BrowserChild Main
  - [+] BrowserChild Top
  - [+] BrowserChild TRANSFERINFORMATION
  - [-] BrowserChild Bottom
    - [+] HtmlPushButton BACK
    - [+] HtmlPushButton Submit
    - [+] HtmlPushButton Reset

Example 1: Results Log


```
>> Function::GetObjectHierarchy (MunicipalTrade, Reset, MunicipalTrade)
    >> Function::GetObjectHierarchy (MunicipalTrade.Top, Reset, MunicipalTrade)
        >> Function::GetObjectHierarchy (MunicipalTrade.Top.Branding, Reset, MunicipalTrade)
            << Function::GetObjectHierarchy
        >> Function::GetObjectHierarchy (MunicipalTrade.Top.ProdImage, Reset, MunicipalTrade)
            << Function::GetObjectHierarchy
        << Function::GetObjectHierarchy
    >> Function::GetObjectHierarchy (MunicipalTrade.Main, Reset, MunicipalTrade)
        >> Function::GetObjectHierarchy (MunicipalTrade.Main.Top, Reset, MunicipalTrade)
            << Function::GetObjectHierarchy
        >> Function::GetObjectHierarchy (MunicipalTrade.Main.TRANSFERINFORMATION, Reset,
                MunicipalTrade)
            << Function::GetObjectHierarchy
        FOUND IT!
        << Function::GetObjectHierarchy
```

Example 1: Results Log


```
>> Function::GetObjectHierarchy (MunicipalTrade.Main_frame, Reset,
    MunicipalTrade.Main.Bottom.Reset)
        << Function::GetObjectHierarchy
    >> Function::GetObjectHierarchy (MunicipalTrade.Top_frame, Reset,
    MunicipalTrade.Main.Bottom.Reset)
        >> Function::GetObjectHierarchy (MunicipalTrade.Top_frame.Branding, Reset,
            MunicipalTrade.Main.Bottom.Reset)
            << Function::GetObjectHierarchy
        >> Function::GetObjectHierarchy (MunicipalTrade.Top_frame.ProdImage, Reset,
            MunicipalTrade.Main.Bottom.Reset)
            << Function::GetObjectHierarchy
        >> Function::GetObjectHierarchy (MunicipalTrade.Top_frame.Menubar, Reset,
            MunicipalTrade.Main.Bottom.Reset)
            << Function::GetObjectHierarchy
        << Function::GetObjectHierarchy
    << Function::GetObjectHierarchy
Window is 'MunicipalTrade.Main.Bottom.Reset'
```

Example 2: Code


```
[-] WINDOW GetObjectHierarchy (WINDOW wMain, STRING sWin)
    [ ] INTEGER i
    [ ] WINDOW w
    [ ] WINDOW wWinFound = NULL
    [ ] LIST OF WINDOW lwChildren
    [ ]
    [ ] ResOpenList (">> Function::dtcc_GetObjectHierarchy ({wMain}, {sWin})")
    [ ]
    [ ] lwChildren = WindowChildren (wMain)
    [ ]
    [-] for each w in lwChildren
        [-] if (WindowIsDefined (w, sWin))
            [ ] wWinFound = w.@sWin
            [ ] Print ("FOUND IT!")
            [ ] break
```

Example 2: Code


```
        [-] else
                [-] if (ClassOf (w) == @"BrowserChild" || ClassOf (w) ==
                    @"ExtendedBrowserChild")
                    [ ] if (wWinFound == NULL)
                            [ ] GetObjectHierarchy (w, sWin, wWinFound)
[ ]
[ ] Print ("<< Function::dtcc_GetObjectHierarchy")
[ ] ResCloseList ()
[ ]
[ ] return (wWinFound)
```

Example 2: Results Log


```
[-] >> Function::GetObjectHierarchy (MunicipalTrade, Reset)
    [-] >> Function::GetObjectHierarchy (MunicipalTrade.Top, Reset)
        [-] >> Function::GetObjectHierarchy (MunicipalTrade.Top.Branding, Reset)
            [ ] << Function::GetObjectHierarchy
        [-] >> Function::GetObjectHierarchy (MunicipalTrade.Top.ProdImage, Reset)
            [ ] << Function::GetObjectHierarchy
        [ ] << Function::GetObjectHierarchy
    [-] >> Function::GetObjectHierarchy (MunicipalTrade.Main, Reset)
        [-] >> Function::GetObjectHierarchy (MunicipalTrade.Main.Top, Reset)
            [ ] << Function::GetObjectHierarchy
        [-] >> Function::GetObjectHierarchy (MunicipalTrade.Main.TRANSFERINFORMATION, Reset)
            [ ] << Function::GetObjectHierarchy
        [ ] FOUND IT!
        [ ] << Function::GetObjectHierarchy
    [ ] << Function::GetObjectHierarchy
[ ] MunicipalTrade.Main.Bottom.Reset
```

Example 3: Problem

- Browser application prompts for Certificate selection within a DialogBox
- DialogBox is modal, SysMenu is disabled
- Must dismiss using Cancel or Finish PushButtons

Example 3: Problem

- PushButtons are NOT first-level descendents of the parent window
  - (Default recovery system will not close these windows)

Example 3: Solution

- CloseObstinateWindow function
  - Calls GetChildren function
  - Loops through list of window children
  - Issue recursive call when window class is BrowserChild

Example 3: Solution

- Appends each PushButton found in an lwPushButton variable
- Clicks the appropriate PushButton in lwPushButton

Example 3: Code


```
[-] void CloseObstinateWindow (WINDOW wOW)
    [ ] LIST OF WINDOW lwChildren
    [ ] LIST OF WINDOW lwGrandChildren
    [ ] LIST OF WINDOW lwPushButton
    [ ] LIST OF WINDOW lwBrowserChild
    [ ] WINDOW w
    [ ]
    [ ] ResOpenList (">> Function::CloseObstinateWindow ({wOW})")
    [ ]
    [ ] lwChildren = wOW.GetChildren ()
    [ ] wOW.SetActive ()
    [-] for each w in lwChildren
        [-] if (w.Exists ())
            [-] if (w.GetClass () == BrowserChild)
                [ ] CloseObstinateWindow (w)
            [-] else
                [-] // call another function
```

Example 3: Code


```
[-] if (ListCount (lwPushButton) > 0)
        [-] for each w in lwPushButton
            [-] if (w.Exists ())
                [-] if (w.GetCaption () == "Continue" || w.GetCaption () == "Finish")
                    [ ] Print ("*--> Continuing window...")
                [-] else
                    [-] if (w.GetCaption () == "Cancel" || w.GetCaption () ==
                "No" || w.GetCaption () == "OK")
                        [ ] Print ("*--> Dismissing window...")
                [ ] w.Click ()
                [ ] break
    [ ]
    [ ] Print ("<< Function::CloseObstinateWindow")
    [ ] ResCloseList ()
```

Recursion: Summary

- Recursion is a way of expressing a solution to a problem
- Use out or inout function arguments to return values
- Be careful to terminate recursive procedures when using the return statement

Concurrent Coding Techniques

- Show how to spawn threads in 4Test
  - parallel or spawn/rendezvous statements
- Demonstrate use of semaphores
- Examine situations where mutual exclusion can be used
  - critical statement

Threads

- "Basic unit of computation," or "a unit of execution"
- A.K.A. process, program, or "task"

Threads

- Windows NT context
  - "Process is the dynamic invocation of a program along with the system resources required for the program to run"
    - Code, data, address space, system resources, at least one *thread of execution*

Threads

- Concepts
  - A thread is an entity within a process that is scheduled for execution
  - Time slicing is used to share the processor among all active threads
  - Thread's context consists of processor state, stacks, and private storage area for use by subsystems, RT libraries and DLLs

Threads

- Shared Resources
  - Critical Resource
    - A resource that can only be accessed by one user (thread) at a time
  - Critical Section
    - A Code region that accesses a critical resource

Threads

- Synchronization Techniques
  - Semaphore
    - The semaphore is a facility that allows threads to block and unblock each other
    - Integer variable whose value can only be altered by operations P and V
      - P and V implemented as Acquire and Release in 4Test

Threads

- Binary Semaphore is a semaphore whose maximum value is 1
  - Defined in 4Test using the SEMAPHORE data type
    - » SEMAPHORE sem1 = 1

Threads

- Mutual exclusion
  - A facility which when invoked, prevents any other operation (thread) from executing between the initiation and termination of the invocation
    - Implemented in 4Test by critical statement

Example 4: Problem

- Parallel, distributed environment
- A mach_state.txt file, resides on the host machine
  - machine name, application name, server, entity, database, current user, current trade info, clearcase view info

Example 4: Problem

- All test scripts running in parallel modify this file at runtime
- File is a critical resource
- Use semaphore to block and unblock threads which execute the block of code which accesses the critical resource (file)

Example 4: Code


```
[+] void MachineConfig_Update (STRING sMachine, MachineAttribute Attr, STRING sNewVal)
    [ ] //
    [ ] // Variables and initialization code
    [ ] //
    [ ] Acquire (sem1)
    [+] do
        [ ] HFILE hFile = FileOpen (sTxtFile, FM_READ)
        [ ] HFILE hNewFile = FileOpen (sNewFile, FM_WRITE)
        [ ]
        [+] while (FileReadLine (hFile, sLine))
            [+] if (GetField (sLine, ",", [INTEGER] MACH) == sMachine)
                [ ] switch Attr
                [ ]
                [ ] // case statements
                [ ]
```

Example 4: Code


```
                [ ] sNewLine = Stuff (sLine, StrPos (GetField (sLine, ",", iPos), sLine),
                    Len (GetField (sLine, ",", iPos)), sNewVal)
            [+] else
                [ ] sNewLine = sLine
                [ ]
            [ ] FileWriteLine (hNewFile, sNewLine)
            [ ]
        [ ] FileClose (hNewFile)
        [ ] FileClose (hFile)
        [ ]
        [ ] LocalMachine->SYS_RemoveFile (sTxtFile)
        [ ] LocalMachine->SYS_CopyFile (sNewFile, sTxtFile)
        [ ] LocalMachine->SYS_RemoveFile (sNewFile)
    [+] except
        [ ] ExceptPrint ()
        [ ] LogError ("")
    [ ] Release (sem1)
```

Example 5: Problem

- SUT allows users to login based on a geographical region or entity
- SUT may prompt user with an Entity selection, warning message, password expiration DialogBox or some combination of these

Example 5: Solution

- Spawn two threads
  - Handle Entity Selection
  - Handle Warning and Password Expiration
- Use critical blocks to prevent interleaving
- Timer is used to limit each thread's execution time

Example 5: Code


```
[+] BOOLEAN LoginWithEnt (STRING sUserName, STRING sPassword, STRING sServerName, STRING
                          sEntity, BOOLEAN bShadowEntity optional)
    [ ] Login.Click ()
    [ ] HTIMER hWaitTime = TimerCreate ()
    [ ] TimerStart (hWaitTime)
    [ ]
    [+] parallel
        [-] while (TRUE)
            [+] if (EntityDialog_popup.Exists ())
                [+] critical
                    [ ] EntityDialog_popup.Text.TypeKeys (sEntity)
                    [ ] EntityDialog_popup.OK.Click ()
                [ ] break
            [+] else
                [+] if (TimerValue (hWaitTime) > 30.0)
                    [ ] break
```

Example 5: Code


```
[+] while (TRUE)
        [-] if (Launcher_Question.Exists ())
                [-] if (Launcher_Question.Warning.Exists ())
                        [+] critical
                                [ ] HTML_LogWarning ("…")
                                [ ] Launcher_Question.No.Click ()
                        [ ] break
                [-] else
                        [-] if (Launcher_Question.YourPassword.Exists ())
                                [-] critical
                                        [ ] Launcher_Question.Yes.Click()
                                                                        [ ] break
        [+] else
                [+] if (TimerValue (hWaitTime) > 30.0)
                        [ ] break
```

Example 6: Problem

- Need to synchronize two events prior to continuing an application's End of Day processing
  - Detect and dismiss an error message
  - Detect when End of Day DialogBox exists

Example 6: Code


```
[+] parallel
    [+] if (InfoMessage.Message.Exists (90))
        [ ] HTML_LogError (InfoMessage.Message.GetText ())
        [ ] InfoMessage.Dismiss.Click ()
        [ ]
    [+] while (! this.DialogBox ("Reconciliation").Exists (90))
        [ ] sleep (1)
```

SetTrap Alternative

- Design
  - Use multitestcase to spawn two threads: a function and testcase
  - Define a global variable to trigger function
  - Set the variable in an appstate
  - Reset the variable in TestcaseExit

SetTrap Alternative

- Observations
  - Extend capability without replacing method
  - Must execute continuously
  - Script runs noticeably slower
  - Code belongs in a critical block
  - Can't spawn a thread from a critical block

Concurrency: Summary

- Threads can be spawned as statements, functions or testcases
- Semaphores are used to control access to critical resources
- Mutual exclusion is used to protect "sensitive" areas of code or to minimize interleaving

References

- Recursion
  - Lorentz, Richard, *Recursive Algorithms*, Ablex, Norwood, New Jersey, 1994.
  - Roberts, Eric S., *Thinking Recursively*, John Wiley, New York, 1986.

References

- Threads
  - Custer, Helen, *Inside Windows NT*, Microsoft Press, Redmond, Washington, 1993.
  - Tsichritzis, Dionysios, C. and Bernstein, Philip A., *Operating Systems*, Academic Press, New York, 1974.

Contact Information

Gerard Sczepura

610-294-9678

610-972-6879 (cell)

gsczep@epix.net