EDKII Build Lab and EFI Shell

Intel Corporation

Software and

Services Group

Disclaimer

THIS INFORMATION COTNAINED IN THIS DOCUMENT, INCLUDING ANY TEST RESULTS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT OR BY THE SALE OF INTEL PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel retains the right to make changes to its specifications at any time, without notice.

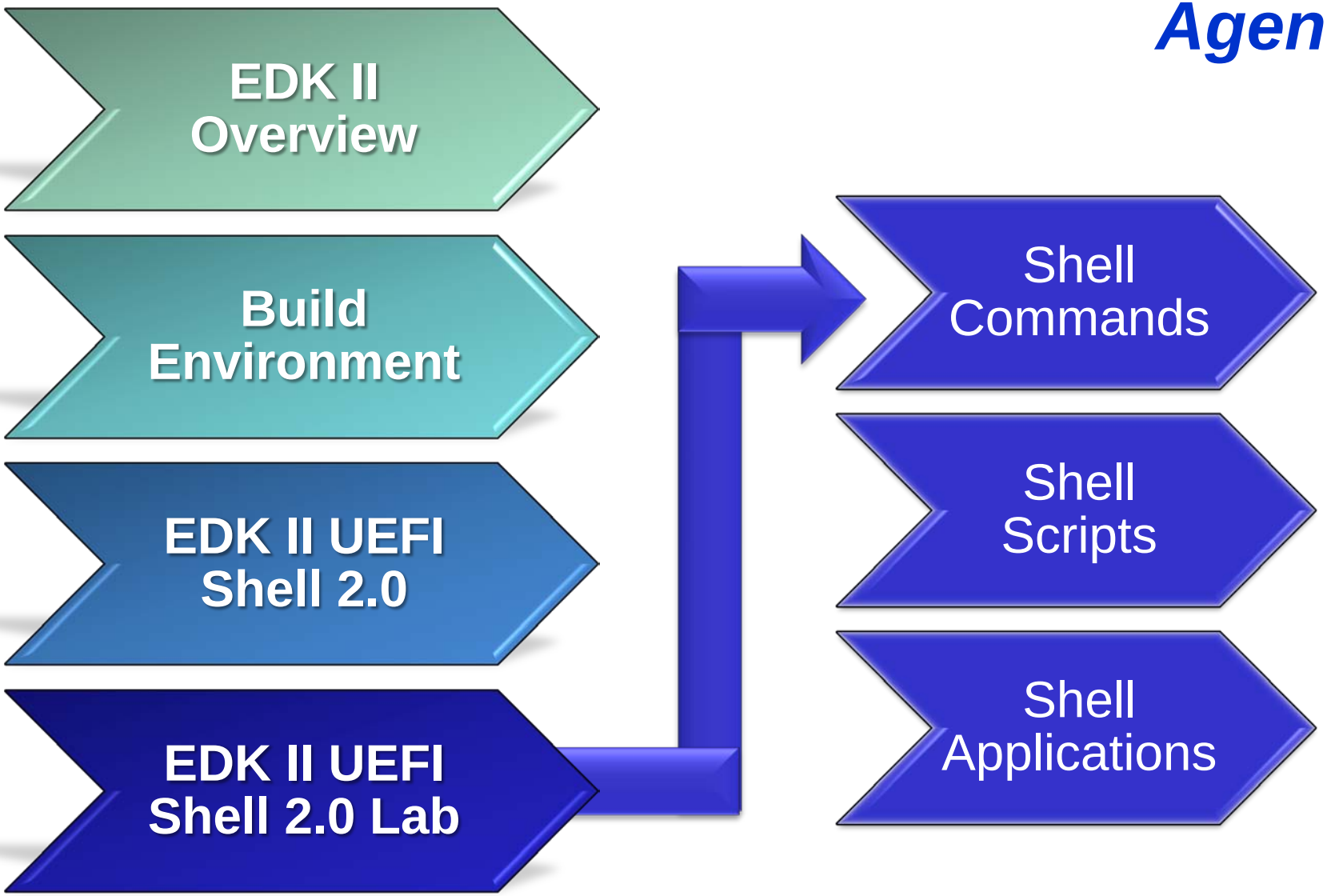
Recipients of this information remain solely responsible for the design, sale and functionality of their products, including any liability arising from product infringement or product warranty.

Intel may make changes to specifications, product roadmaps and product descriptions at any time, without notice.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Agenda



Philosophy of EDK II

File extensions

EDK II Infrastructure

- Directory structure
- Packages
- Libraries
- Platform Configuration Database (PCD)

What is EFI Development Kit 2 (EDK II)

Philosophy of EDK II

- Support all UEFI and PI development needs
- Separate tool code from source code
- Build existing EDK modules
  - Via EDK Compatibility Package (ECP)
- Package Definition File: DEC
  - DEC defines package of modules
- FLASH Mapping Tool
- Move as much as possible to C code
- Open source home on http://tianocore.org (Sourcforge)

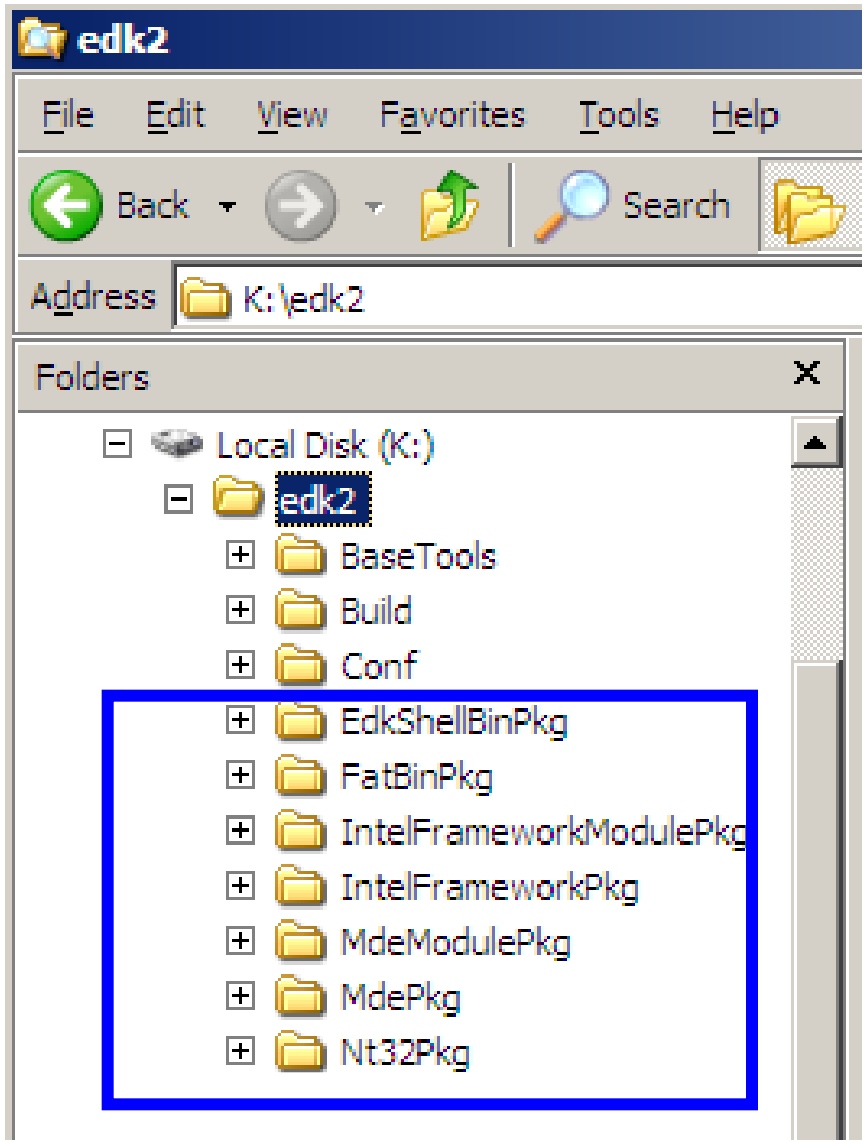
File Extensions

- **.DSC** file - Platform Description
  - describes the build rules, libraries and components
- **.DEC** file - Package Declaration
  - declares the interfaces
- **.INF** file - Module Definition
  - defines one component
- **.FDF** file - Flash Description File
- **.DXS** file - Dependency expression file – now [DEPEX]
- **.FV** file - Firmware Volume image file

See EDK II Build Specification Documentation:

http://sourceforge.net/projects/edk2/files/

EDK II Directory Structure - NT32 Emulation



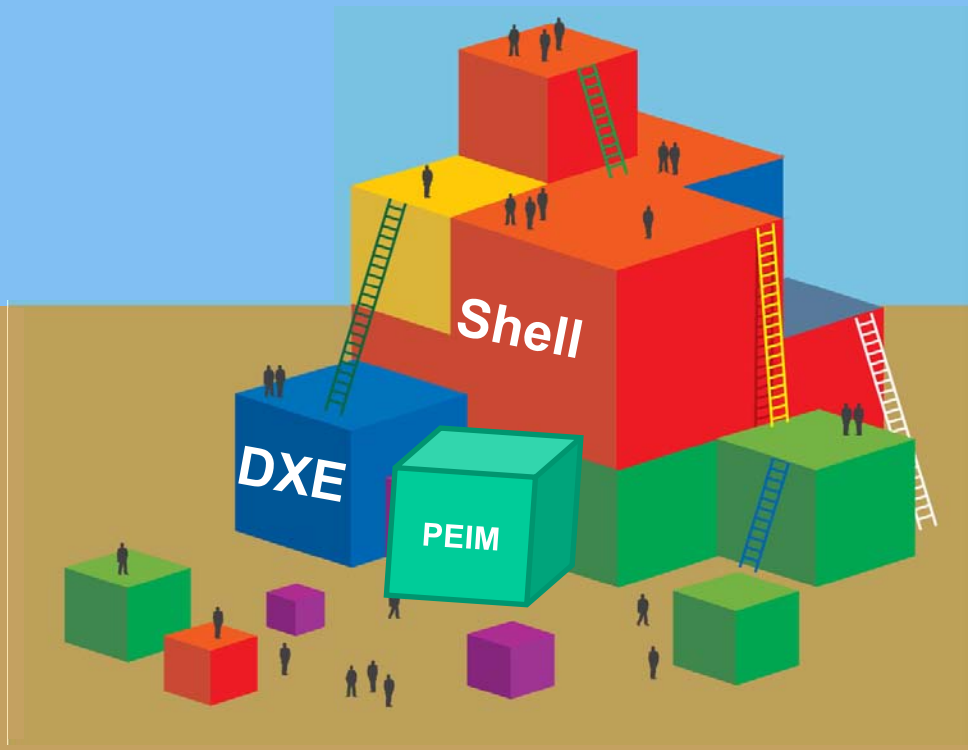
- Package Concept for each Directory
- Platforms contained in a Package

2 Steps to Build

```
>EdkSetup --nt32
>Build -p
Nt32Pkg\Nt32Pkg.dsc
-a IA32
```

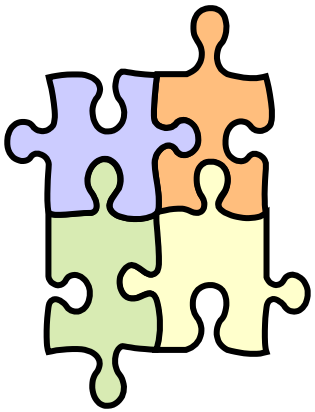
Modules

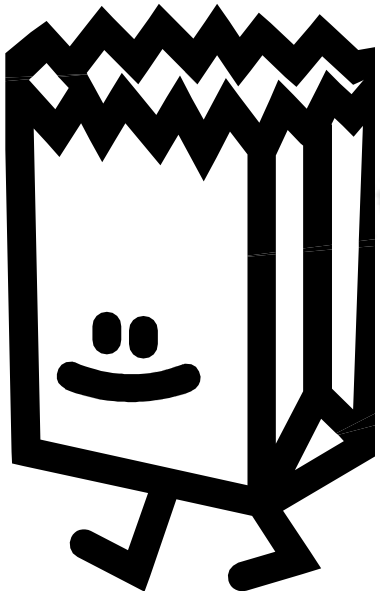
- A module is the smallest separate object compiled in the EDKII. A module is a single resultant .EFI file.
- Module examples:
  - A UEFI/DXE driver
  - A PEIM
  - A UEFI application
  - A Library



Package Philosophy

- As standards evolve there is a need to target your development on the set of standards you care about
- Solution: break the EDK II up into “packages” and enable customers to make their own packages.
- Only package together what is needed





Package Examples - Specs

- MdePkg
  - Include files and libraries for Industry Standard Specifications (i.e. UEFI, PI, PCI, USB, SMBIOS, ACPI, SMBIOS, etc)
- MdeModulePkg
  - Modules (PEIMs + DXE Drivers + UEFI Drivers + UEFI Applications) that only definitions from the Industry Standard Specification defined in the MdePkg.
- IntelFrameworkPkg
  - Include files and libraries for those parts of the Intel Platform Innovation Framework for EFI specifications that were not adopted “as is” by the UEFI or PI specifications
- IntelFrameworkModulePkg
  - Contains modules (PEIMs + DXE Drivers + UEFI Drivers) that make reference to one or more definitions in the IntelFrameworkPkg.

Package Examples

Platforms

- Nt32Pkg – contains drivers and applications required for running over Windows
- OVMF – Virtual Machine Firmware

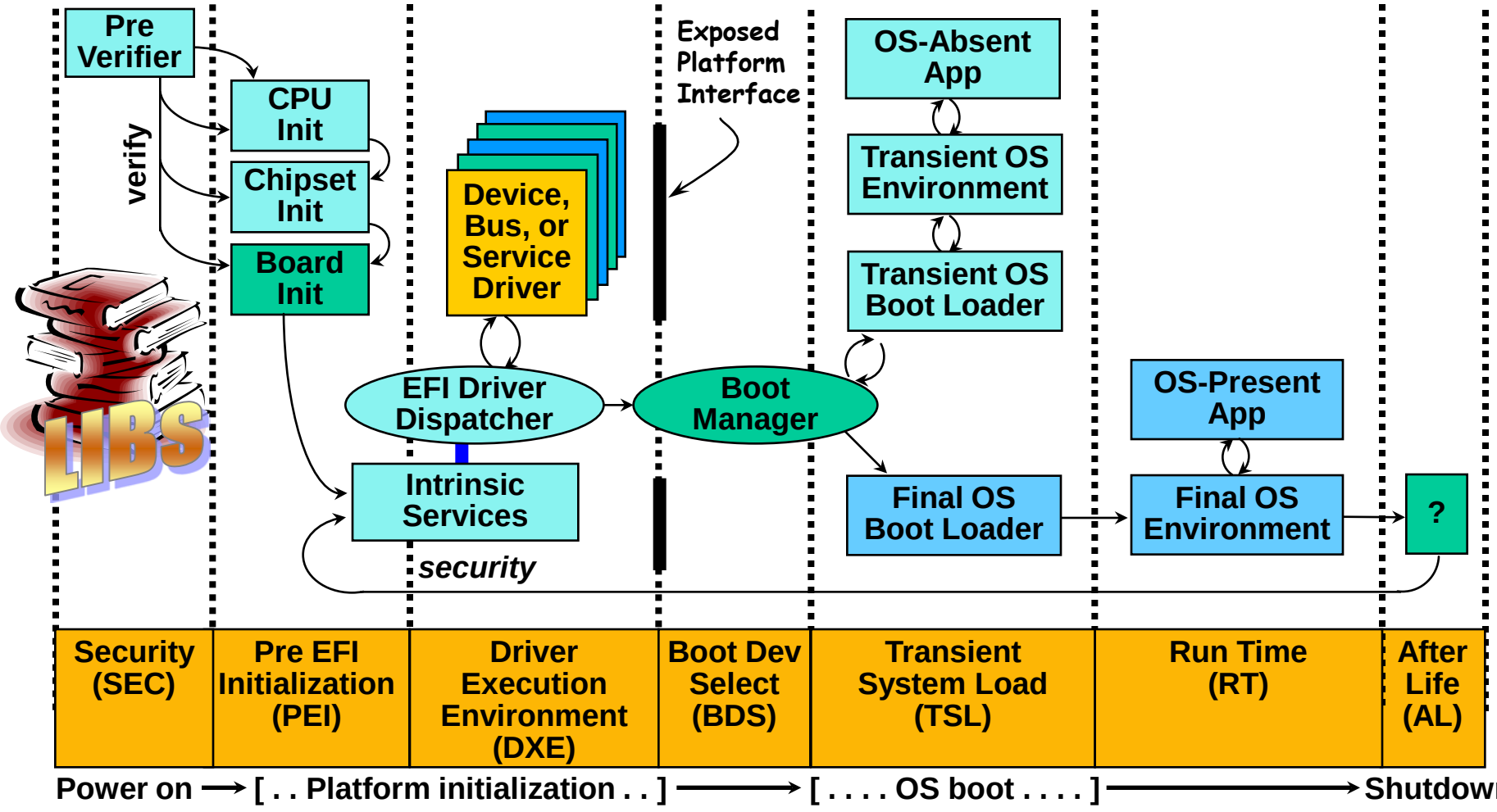
Chipset/Processor

- IA32FamilyCpuPkg.- Intel Architecture
- Ich9Pkg – Intel I/O controller hub 9

Functionality

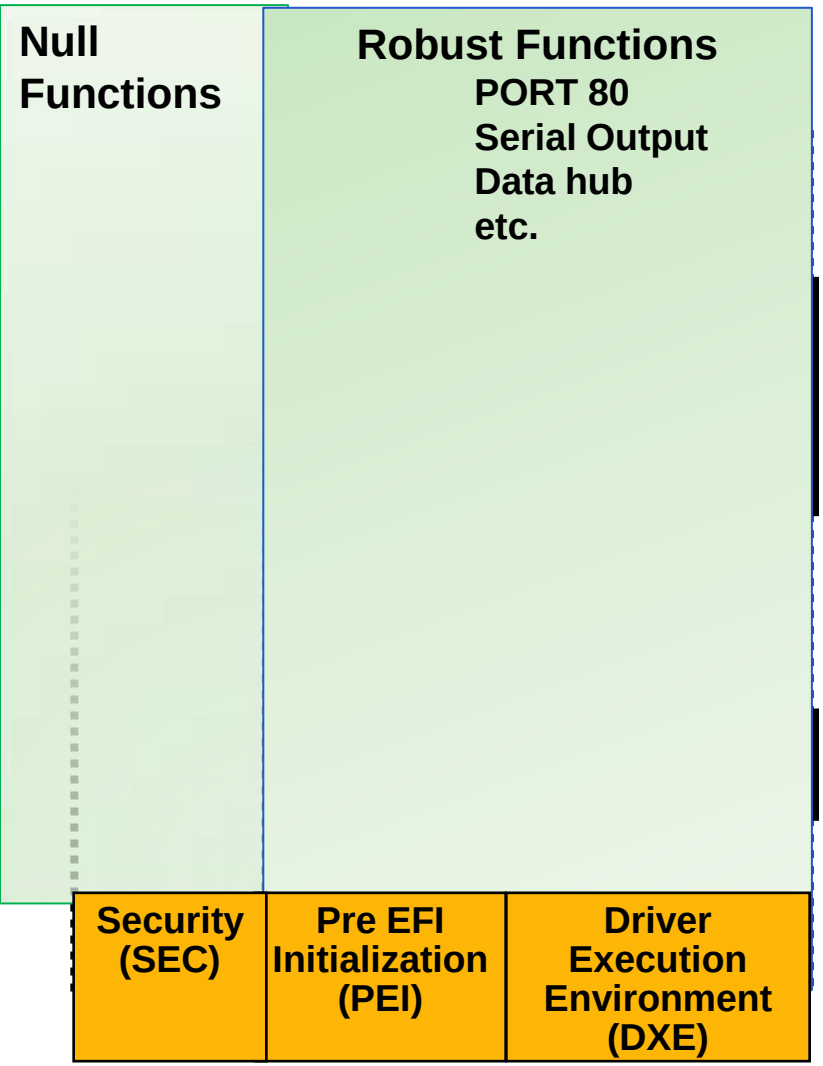
- ShellPkg
- NetworkPkg

Libraries - UEFI /PI Execution Phases



Same lib classes exist across multiple phases

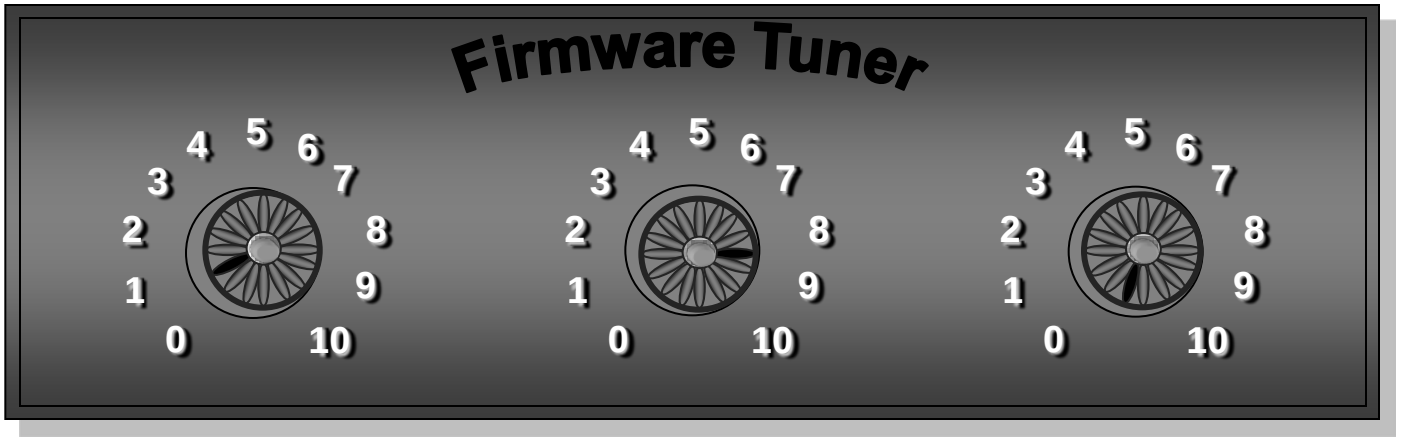
Example: Using EDK II Libs



- **DebugLib** library class name
  - The debugging characteristics might be different during the different boot up phases.
    - SEC – Null or no code
    - PEI-DXE-BDS – Code to output to serial port/ port 80 codes
- Library Class names can be linked with different Processor resolutions (IA32/ x64) & different boot phases
- Syntax covered in 1 & 3 day training

Platform Configuration Database

- Knobs to fine tune your firmware



Platform Configuration Database Goal

- PCD entries are used for module “parameterization”.
- Benefits:
  - Reduce the need to edit source code
  - No searching for “magic” #define statements
  - Maximize module reuse across platforms
  - APIs for access to PCD entries
- PCDs can store platform information
  - Vital Product Data (VPD)
  - Serial Number, etc…
  - Setup options

Maximizes the re-use of modules - Minimize Source code editing

EDK II Infrastructure Overview summary

- Directory structure
  - Logical buckets for functionality
- Packages
  - Build can build single package by itself
- Libraries
  - Way to map implementation to a library name
- Platform Configuration Database (PCD)
  - Way to define and assign values without changing actual code

Agenda

Development Environment

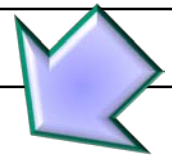
- Compiler tool chains:
  - Microsoft C compilers and WDK
  - Intel C/C++ compiler
  - Intel C UEFI Byte code compiler
  - GCC V4.x or later
- Operating Systems:
  - Microsoft Windows (XP, VS2008) [DEFAULT]
    - Windows 7 in progress
  - Apple Mac OS X
  - RedHat Enterprise Linux
  - Novell SuSE Linux

Script Files

- edksetup.bat or edksetup.sh
  - used to setup/verify a developer's workspace,
  - ensure that environment variables like WORKSPACE and EDK_TOOLS_PATH are set correctly
  - Sets up configuration files - target.txt, tools_def.txt, build_rule.txt

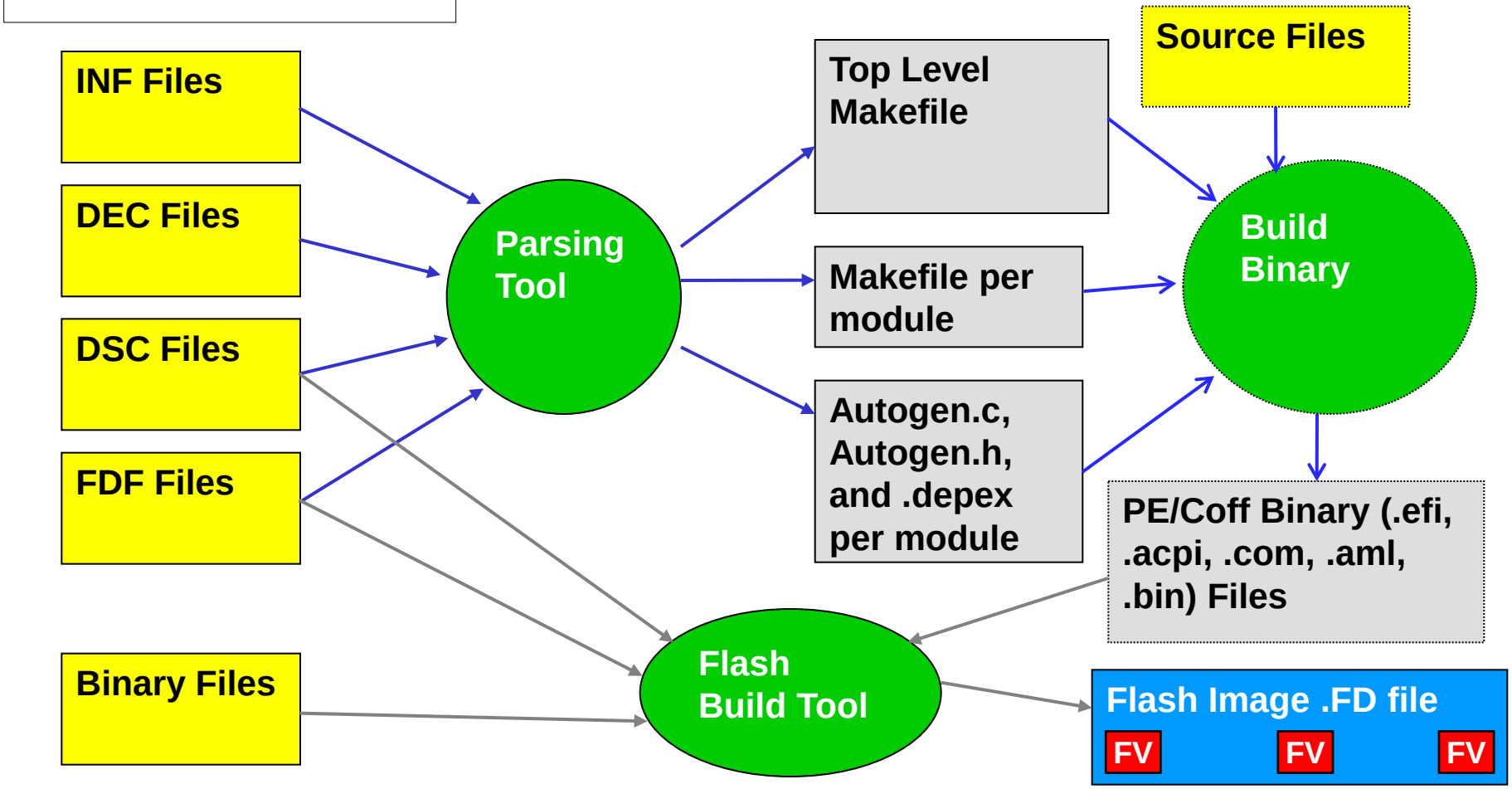
Target.txt

Tag	Meaning
Active Platform	What DSC file represents the platform that you are building?
Target	Debug Build or Release Build? DEBUG or RELEASE
TARGET_ARCH	What is the Architecture that you want to build? IA32, IPF, X64, EBC, or ARM
TOOL_CHAIN_CONF	Path to the tools_def file
TOOL_CHAIN_TAG	Section of the tools_def file to use
MAX_CONCURRENT_THREAD_NUMBER	How many threads to use to build



Platform Build Process Flow

KEY
Green – Process
Yellow – Source or Inputs
Grey – Output or Intermediate
Blue – Final ROM image



Basic Platform Build

- Open command prompt
  - If on Windows use the Visual Studio* 2008 Command Prompt
- Navigate to root of EDK2 workspace
- Run “edksetup”
- Run “build”
- Output is finished FD(s)

Basic Module Build

- Open command prompt
  - If on Windows use the Visual Studio* 2008 Command Prompt
- Navigate to root of EDK2 workspace
- Run “edksetup” script file
- Change to a directory with an INF
- Run “build”
- Output is .EFI files in build directory

How EDK II Build Works – Parsing Stage

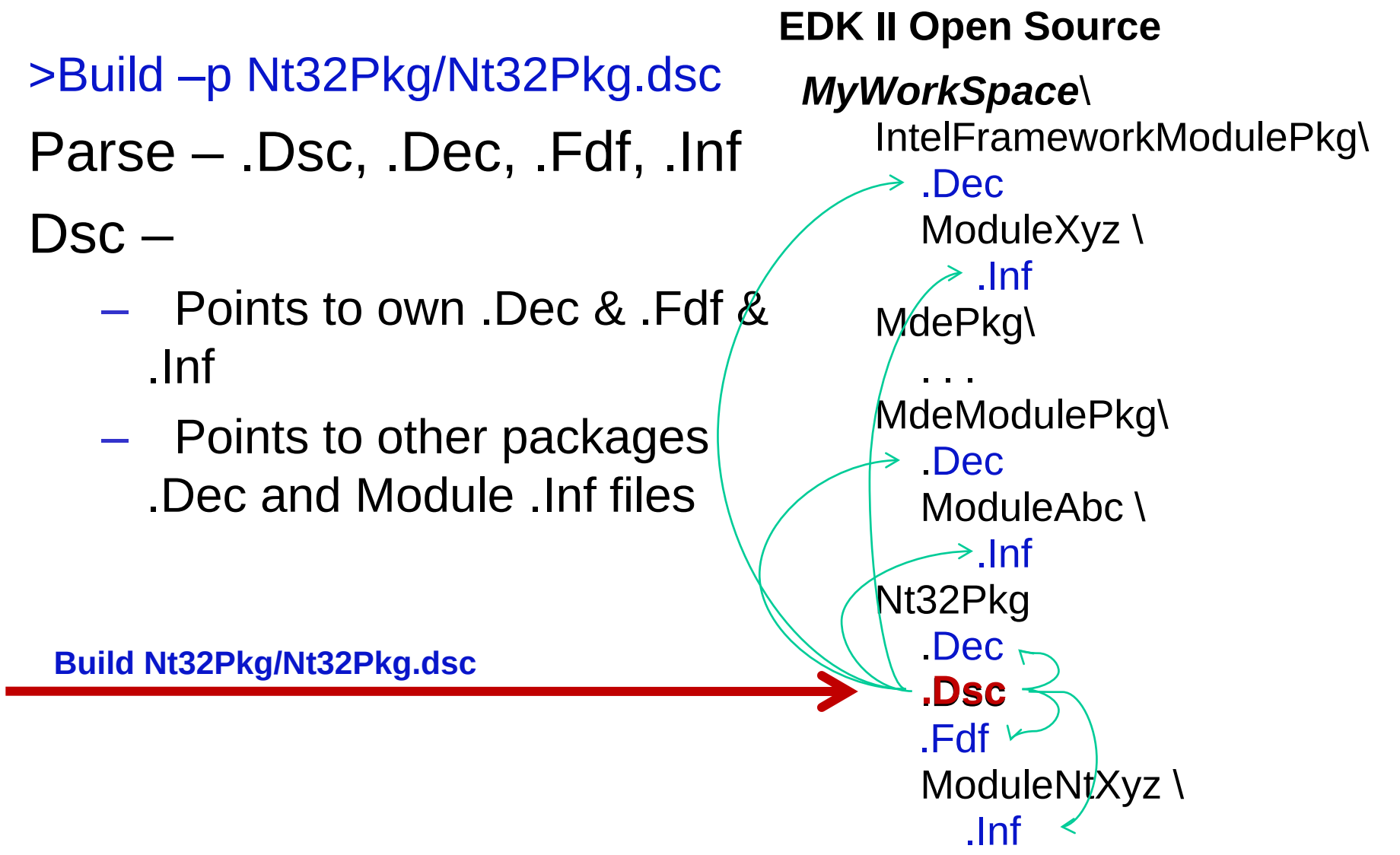
>Build –p Nt32Pkg/Nt32Pkg.dsc

Parse – .Dsc, .Dec, .Fdf, .Inf

Dsc –

- Points to own .Dec & .Fdf & .Inf
- Points to other packages .Dec and Module .Inf files

Build Nt32Pkg/Nt32Pkg.dsc



Build Command Line Usage

```
>BUILD -help
```

Usage: build.exe [options] [all|fds|genc|genmake|clean|cleanall|cleanlib|modules|libraries|run]

--version	show program's version number and exit
-h, --help	show this help message and exit
-a TARGETARCH, or --arch=TARGETARCH	ARCHS is one of list: IA32, X64, IPF or EBC, which overrides target.txt's TARGET_ARCH definition. To specify more archs, please repeat this option.
-p PLATFORMFILE, or --platform=PLATFORMFILE	Build the platform specified by the DSC file name argument, overriding target.txt's ACTIVE_PLATFORM definition.
-m MODULEFILE, or --module=MODULEFILE	Build the module specified by the INF file name argument.
-b BUILDTARGET, or --buildtarget=BUILDTARGET	BuildTarget is one of list: DEBUG, RELEASE, which overrides target.txt's TARGET definition. To specify more TARGET, please repeat this option.
-t TOOLCHAIN, or --tagname=TOOLCHAIN	Using the Tool Chain Tagname to build the platform, overriding target.txt's TOOL_CHAIN_TAG definition.
-Y REPORT-TYPE or --report-type=REPORT-TYPE	Generate Reports for the PCD, LIBRARY, FLASH, DEPEX, BUILD_FLAGS, FIXED_ADDRESS

See Backup for complete list

Where is the output

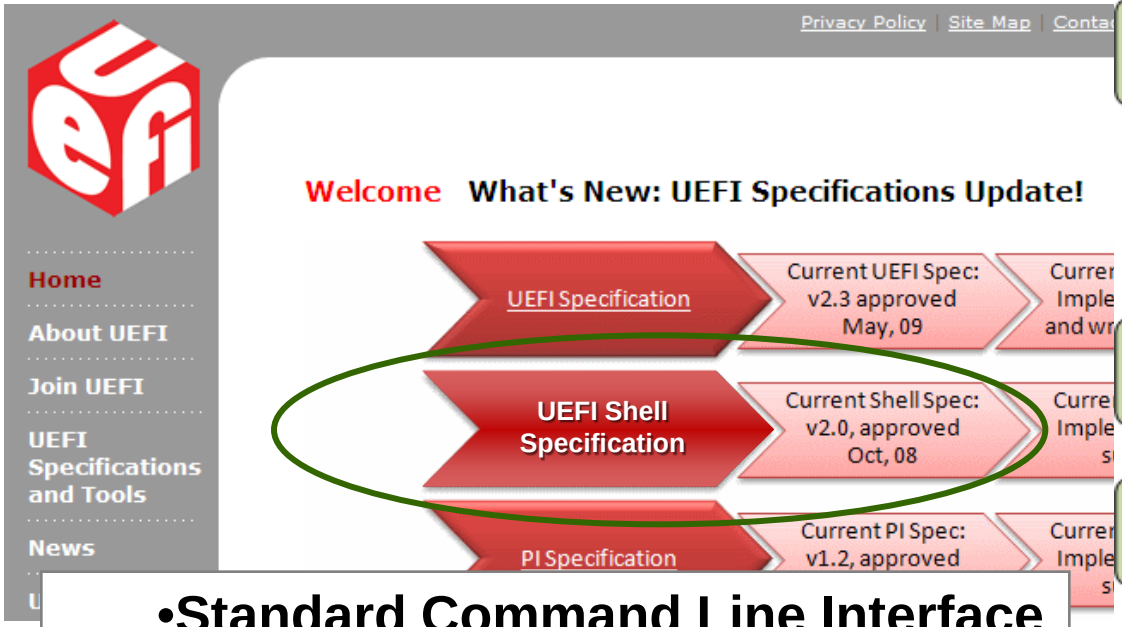
```
Build\NT32\DEBUG_MYTOOLS\FV
Build\NT32\DEBUG_MYTOOLS\IA32\Pkg\ModuleName\SA3\OUTPUT
Build\NT32\DEBUG_MYTOOLS\IA32\Pkg\ModuleName\SA3\DEBUG
```

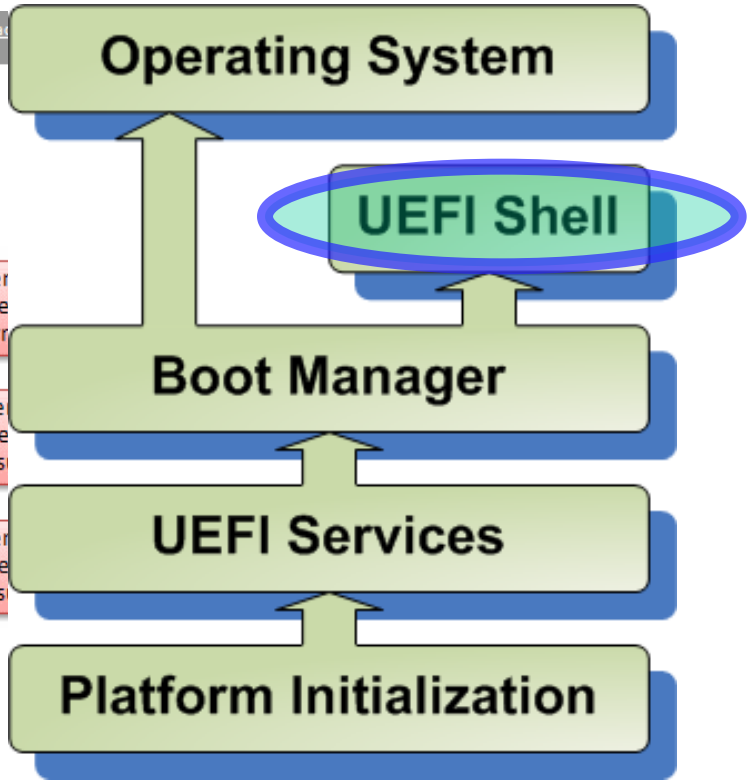
Path Element	Description	Notes
Build	Build directory	This is default.
NT32	platform being used	
DEBUG_MYTOOLS	build mode and tool chain	From target.txt
IA32	processor architecture	Contains platform makefile
FV	Contains final images	Both FV and FD images
Pkg\ModuleName	path to INF file	One for each INF
SA3	name of INF file	Contains module makefile
OUTPUT	.EFI file location	
DEBUG	Autogen files	

Agenda

UEFI Shell Specification

- Spec Version 2.0 Released October 2008
- Implementation in progress





- **Standard Command Line Interface**
- **Sits directly on UEFI firmware**
- **Optimized for small footprint**
- **Supports rich applications**
- **Standard scripting & API**

UEFI Shell vs. EFI Shell

Small Size Profiles	0: Shell API only 1: Basic scripting support 2: File support cmds (cd,cp,mv) 3: Adds interactive CLI + profiles
New Shell API	Smaller executable size Expose previously hidden shell capabilities Execution break support
Enhanced Scripting	Compatible with existing scripts Added input redirection & piping Enhanced if command
Shell Commands	Standardized existing usage Updated for UEFI 2.1+ Standardized argument usage and output

Different Levels Of Shell Support

- Different levels of support for different usage scenarios and space constraints:
  - Level 0: No Command-line Interface (CLI). No shell commands. Only shell API.
  - Level 1: Adds basic scripting support
  - Level 2: Adds basic commands (cd, cp, mv)
  - Level 3: Adds interactive CLI
- Shell support level can be detected using an environment variable.
- Beyond level 3, additional command “profiles” are defined for debug, networking and driver support.

Support levels save space in low-resource environments

UEFI Shell Standard Commands

Level/Profile	Commands
Level 0	None
Level 1	else, endfor, endif, exit, for, goto, if, shift
Level 2	attrib, cd, cp, date, del, load, map, mkdir, mv, reset, rm, set, time, timezone, touch
Level 3	alias, cls, dir, help, ls, mode, pause, type, ver
UEFI Debug Profile	bcfg, comp, dblk, dmem, dmpstore, echo, edit, eficompress, efidecompress, hexedit, loadpcirom, mem, memmap, mm, pci, sermode, setsize, smbiosview
UEFI Network Profile	ipconfig, ping
UEFI Driver Profile	connect, devices, devtree, dh, disconnect, drivers, drvcfg, drvdiag, openinfo, reconnect, unload

Choose the amount of shell capability for market segment / platform type

UEFI Shell API Overview

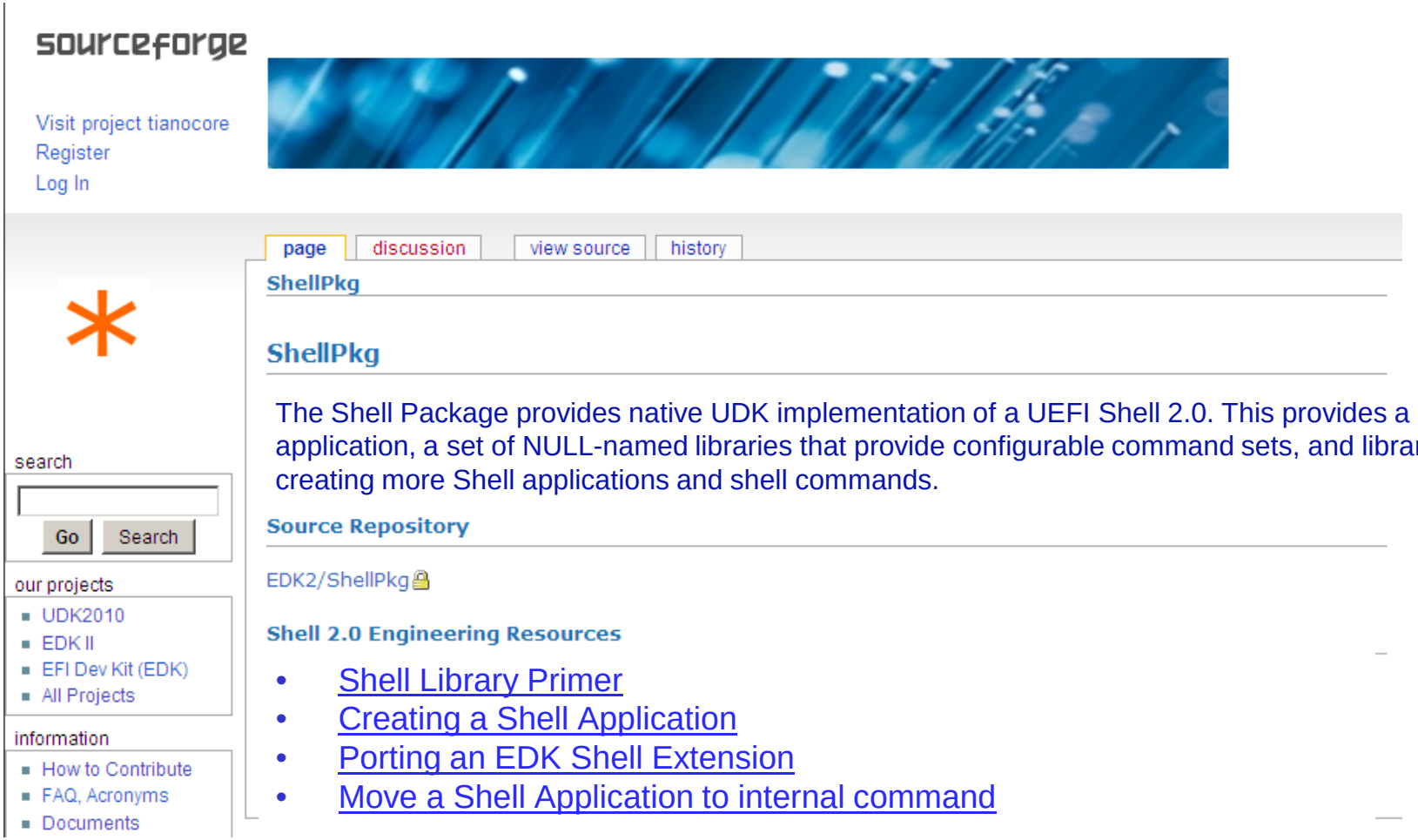
Group	Functions
File Manipulation	CreateFile, DeleteFile, ReadFile, WriteFile, DeleteFileByName, CloseFile, FindFiles, FindFilesInDir, GetFilePosition, SetFilePosition, GetFileInfo, SetFileInfo, FreeFileList, OpenFileByName, OpenFileList, OpenRoot, OpenRootByHandle, GetFileSize, RemoveDupInFileList
Mapping, Alias & Environment Variables	GetMapFromDevicePath, GetFilePathFromDevicePath, GetDevicePathFromFilePath, GetDevicePathFromMap, SetMap, SetAlias, GetEnv, SetEnv, GetCurDir, SetCurDir
Launch Application Or Script	Execute, BatchIsActive, IsRootShell
Miscellaneous	GetPageBreak, EnablePageBreak, DisablePageBreak, GetHelpText, GetDeviceName

EFI_SHELL_PROTOCOL is installed on each application image handle

EDK II Shell Library

- EDK II native library for shell applications
  - ShellPkg on Open Source, Repository: https://edk2.svn.sourceforge.net/svnroot/edk2/trunk/edk2
- Supports binary portability between EFI shell 1.0 and UEFI 2.0 shell
- Shell protocols
  - Calls into UEFI (default) or EFI for functionality
- Shell parameters
  - Handles parameter parsing for flag, value, and position command-line parameters
  - EDK II UEFI Shell 2.0 Library globals in Shell applications
    - `#Include <Library/ShellLib.h>`
    - `mEfiShellParametersProtocol`
    - `mEfiShellProtocol`

Documentation for EDK II ShellPkg

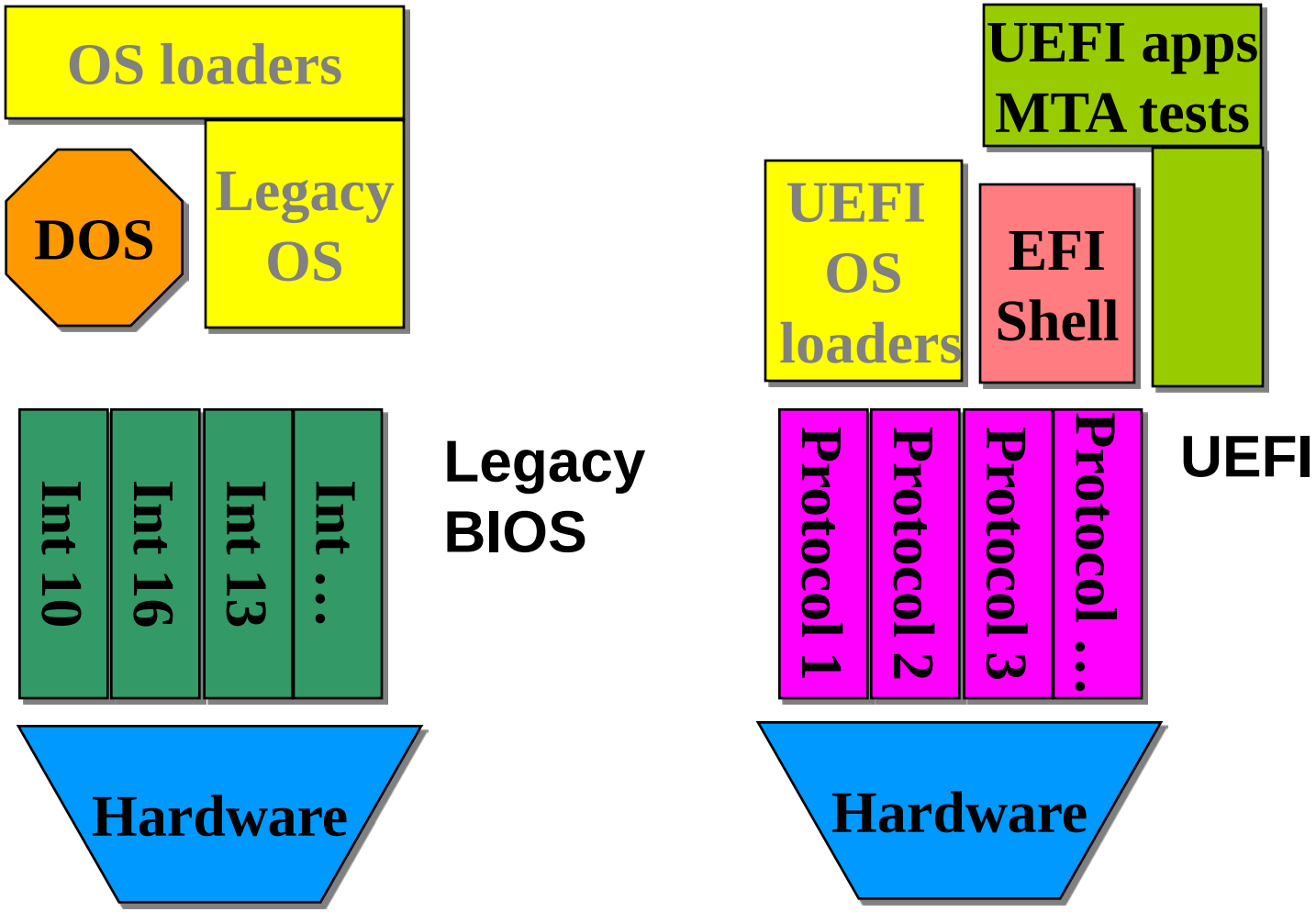


http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=ShellPkg

UEFI Shell Scripts

- Text files with **`.nsh`** extension are shell scripts
- Supports command-line arguments using positional parameters **`%0`** - **`%9`** and **`shift`**.
- Supports standard script commands
  - **`if/else/endif`**
  - **`goto`**
  - **`for/endfor`**
  - **`echo`**
  - **`exit`**
- Supports input & output redirection & pipes.
  - Can also redirect to/from environment variables!

Analogy to Old DOS: BIOS



Shell Usage

- Execute preboot programs
  - Operating system install
  - Testing
  - Disk utilities
  - Driver Diagnostics
- Move files around between the hard disk, floppy disk, CD-ROM, USB flash devices, and so on
- Load a preboot EFI driver in the system (has an .efi suffix), examples:
  - LAN stack tcpip drivers
  - Update old drivers in flash
  - New drivers for plugin cards

EFI & UEFI File System

Can manipulate only UEFI system fatxx partitions where boot loader and UEFI applications are located

```
Shell> map
Device mapping table
fs0  : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
 Scsi(Pun0,Lun0)/HD(Part1,Sig8983DFE0-F474-01C2-507B-
 9E5F8078F531)
blk0 : Acpi(PNP0A03,0)/Pci(1F|1)/Ata(Primary,Slave)
blk1 : Acpi(PNP0A03,0)/Pci(1F|1)/Ata(Primary,Master)
blk2 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)
blk3 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
  Scsi(Pun0,Lun0)/HD(Part1,Sig8983DFE0-F474-01C2-507B-
9E5F8078F531)
blk4 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
  Scsi(Pun0,Lun0)/HD(Part2,Sig898D07A0-F474-01C2-F1B3-
12714F758821)
blk5 : Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/
  Scsi(Pun0,Lun0)/HD(Part3,Sig89919B80-F474-01C2-D931-
F8428177D974)
```

Device Path

fs0 :
Acpi(PNP0A03,1)/Pci(1F|0)/Pci(2|0)/Scsi(Pun0,Lun0)/HD(Part1
,
Sig8983DFE0-F474-01C2-507B-9E5F8078F531)

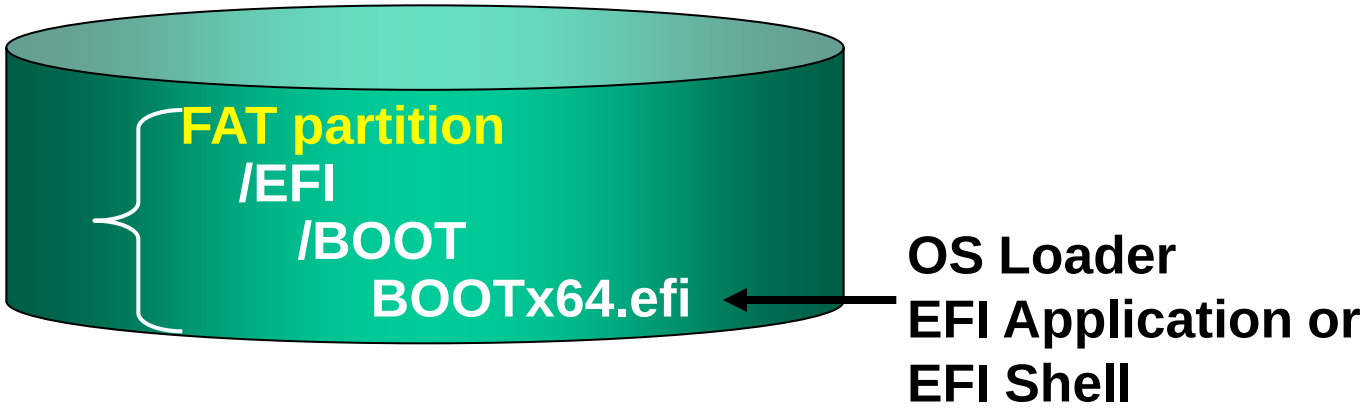
Fs0:

Acpi(PNP0A03,1)
Pci(1F|0)/Pci(2|0)
Scsi(Pun0,Lun0)

HD(Part1,Sig8983DFE0-F474-01C2-507B-9E5F8078F531)

External Shell Boot

- To boot to the Shell one partition must be formatted FAT.
- On the Fat partition under /EFI/boot/* special file called:
  - BOOTx64.efi or BOOTIA32.efi

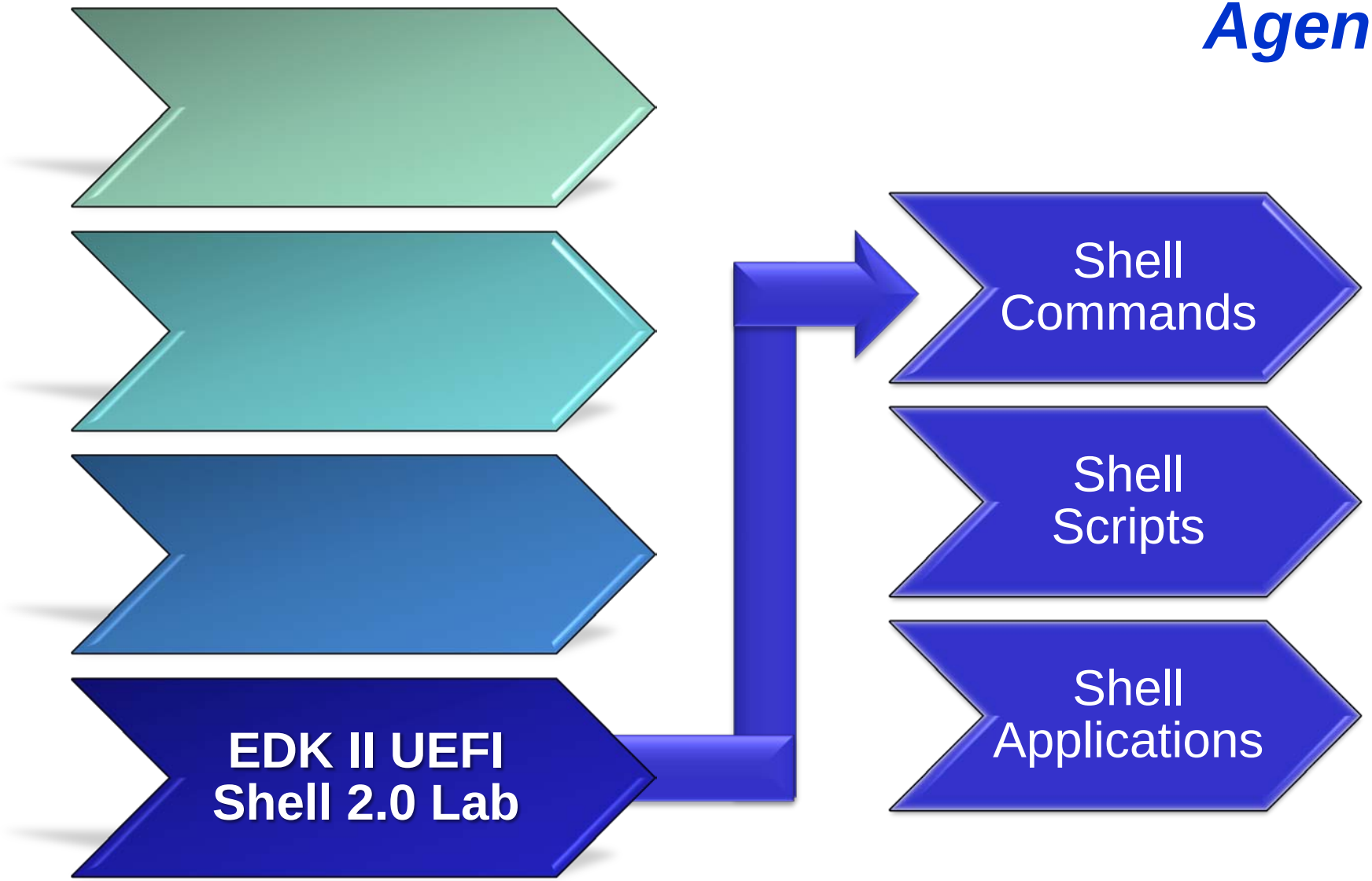


- Boot to Shell on USB
  - Binary image of shell named BOOTx64.efi can be used

EFI & UEFI Shell Commands Help ?

alias
attrib
bcfg
cd
cls
comp
connect
cp
date
dblk
del
devices
devtree
dh
dir
disconnect
dmem
dmpstore
drivers
drvcfg
drvdiag
echo
edit
eficompress
efidecompress
else
endfor
endif
exit
for
getmtc
goto
guid
help
hexedit
if
ifconfig
ipconfig
load
loadpcirom
ls
map
mem
memmap
mkdir
mm
mode
mv
openinfo
parse
pause
pci
ping
reconnect
reset
rm
sermode
set
setsize
setvar
shift
smbiosview
time
touch
type
unload
ver

Agenda



NT32 Build Lab

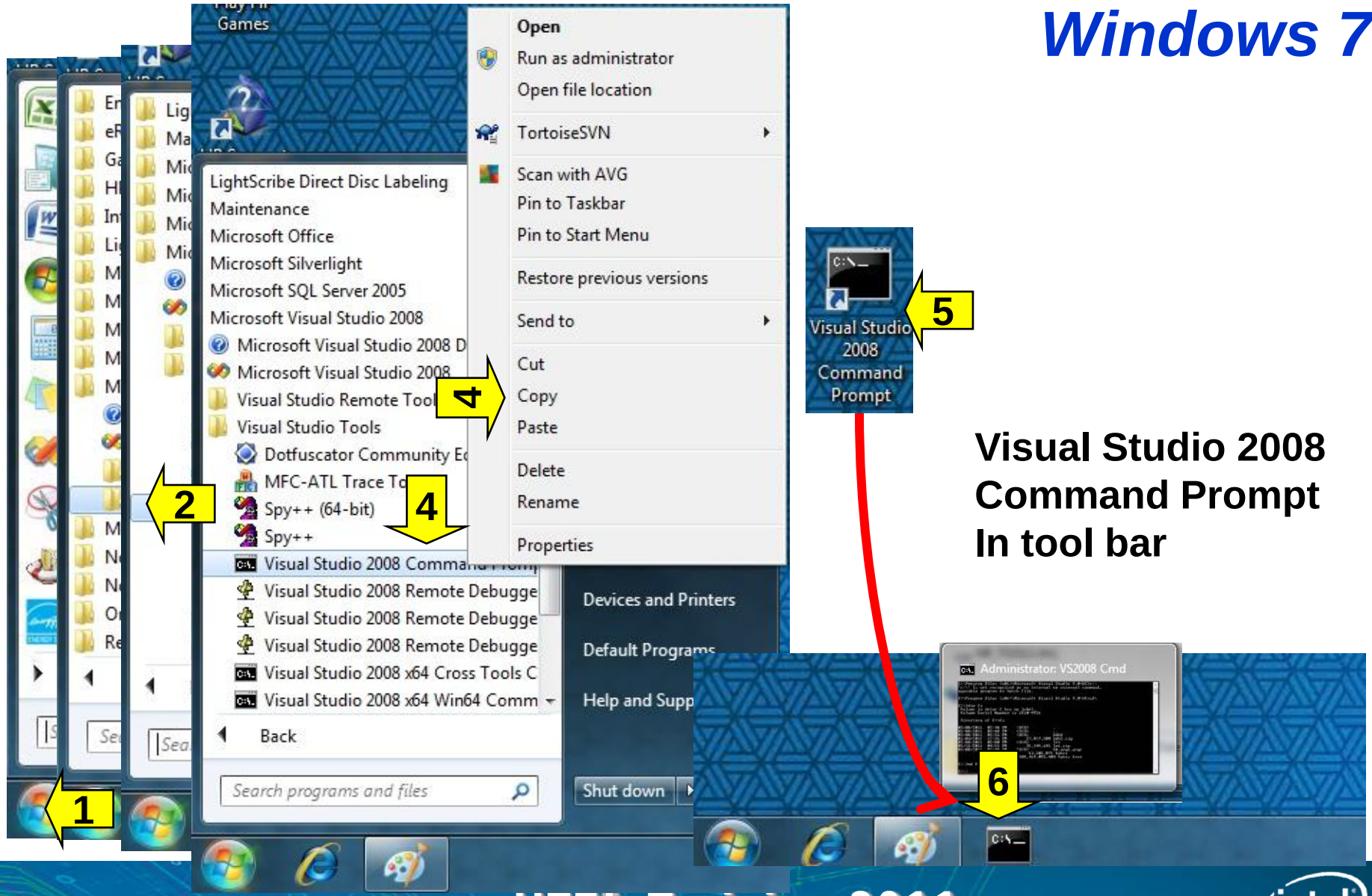
- System requirements:
  - Microsoft Windows* XP or 32 Windows 7
  - 500MB+ System Memory
  - 1GB+ Free Space on Hard Drive
  - Visual Studio 2008 (Ver 9.0) Professional installed

- Copy Edk2 Source from Share to C:\Fw

OR

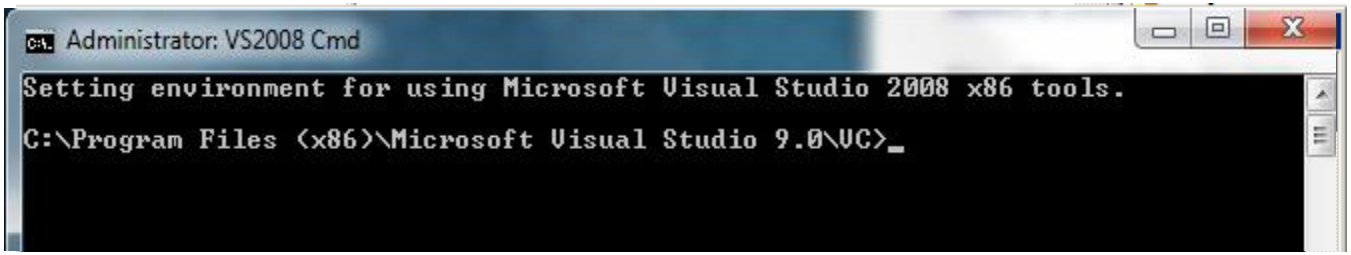
- CD Directory NT32BuildLab/EDKII has Edk2.zip
  - Unzip to C:\Fw

Compiler Environment on Windows 7



Visual Studio 2008 Command Prompt In tool bar

Building the NT32 Environment



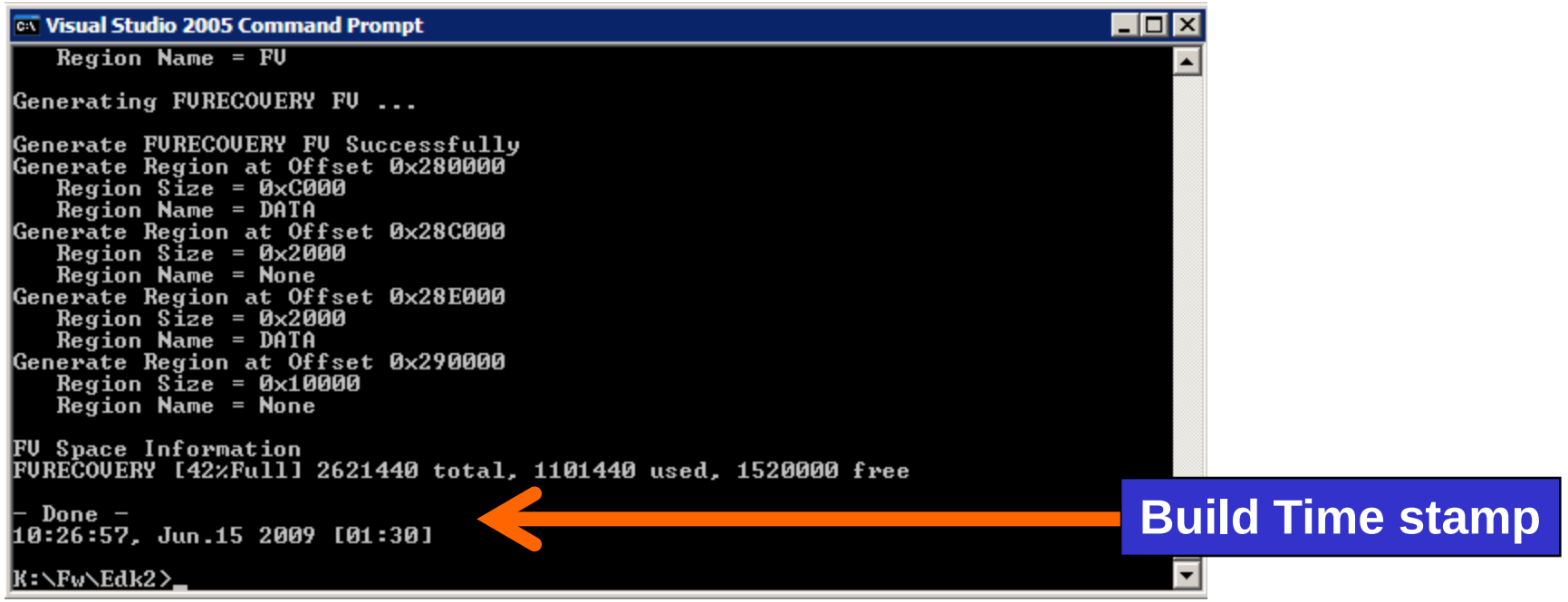
1. `CD C:\FW\Edk2`
2. `EdkSetup`
3. `Notepad Conf\Target.txt`

Change TOOL_CHAIN_TAG = # → ***replace MYTOOLS with VS2008x86***

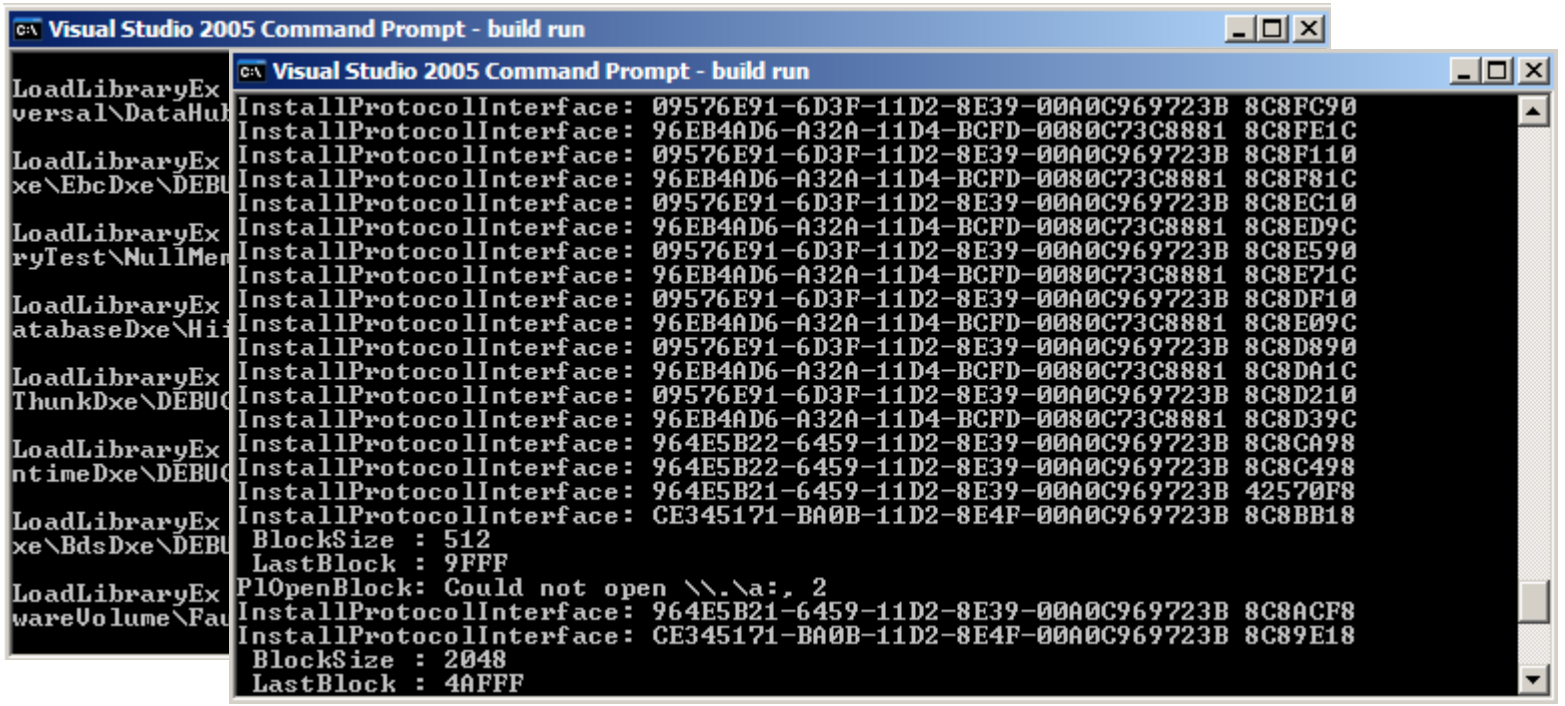
4. `Build`
5. Wait for Build to finish ☺

NT32 Build Lab

- After BUILD is complete type:
  - **`Build Run`**



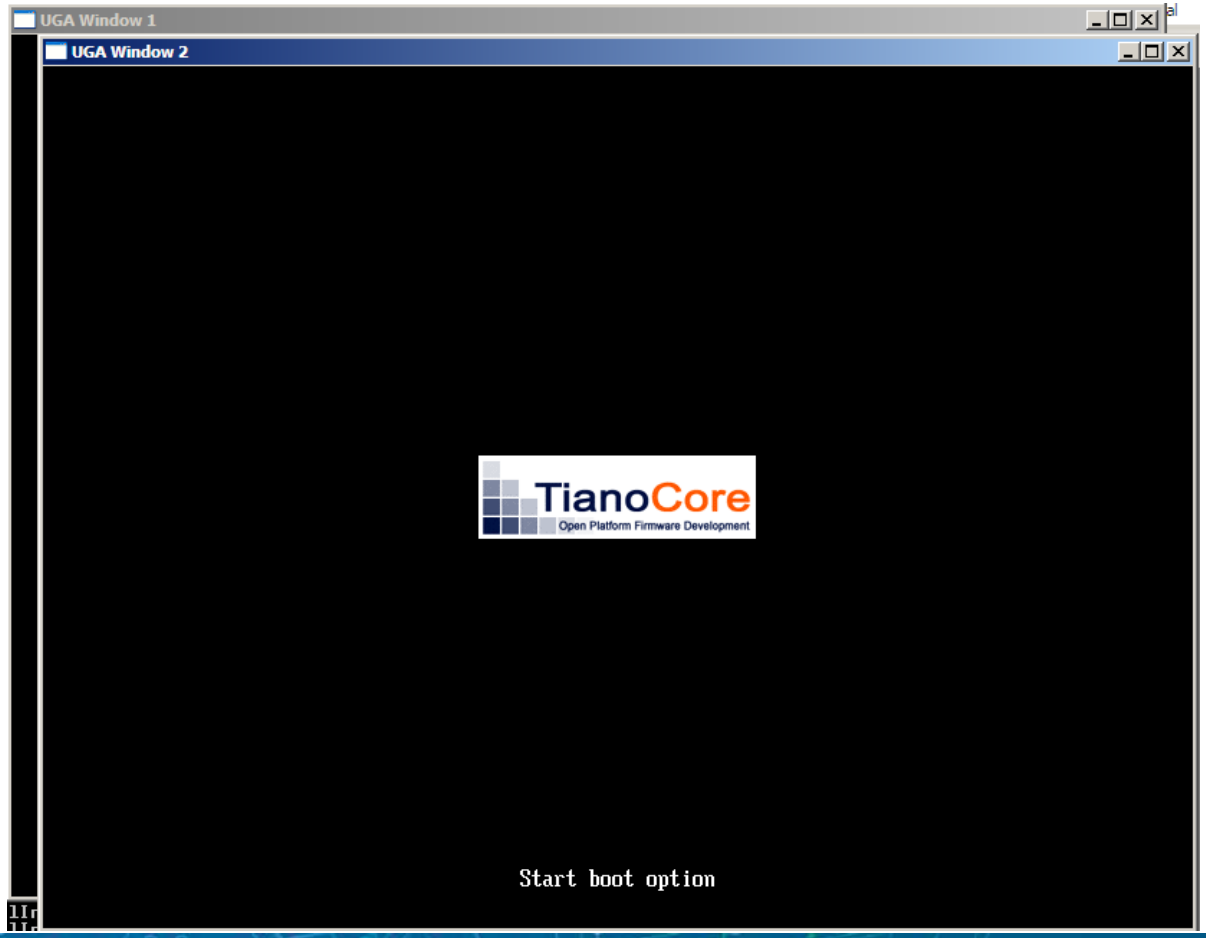
Build Run



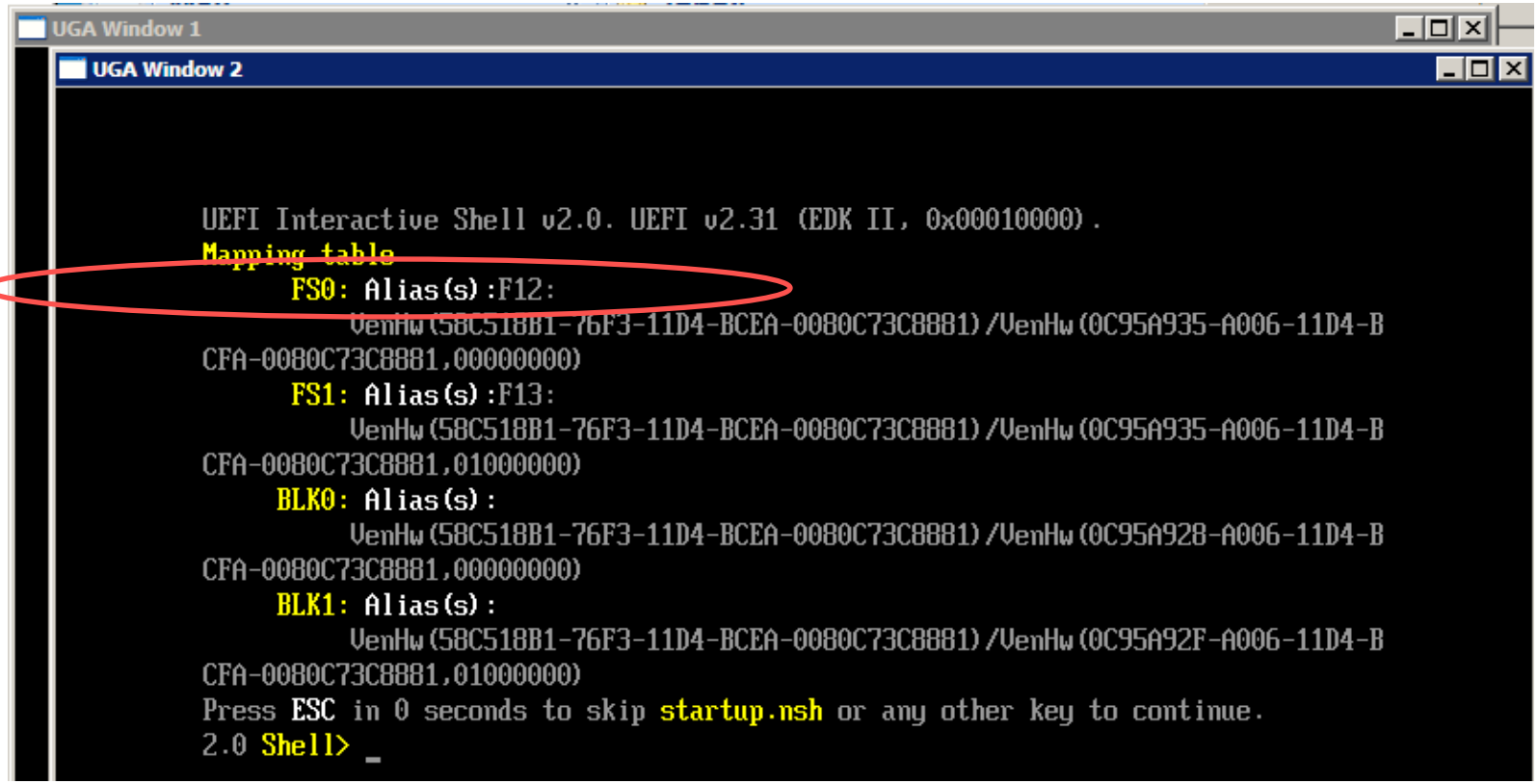
- Start Emulation
  - Emulation runs as a Windows* application
  - Reset vector code for desktop/server replaced with Windows* application startup code
  - Startup code initializes the UEFI firmware and calls dispatcher to start loading drivers

NT32 Emulated Boot

- Splash screen displayed
  - Windows drivers simulate output to screen, input from keyboard and disk operations to Hard Drive

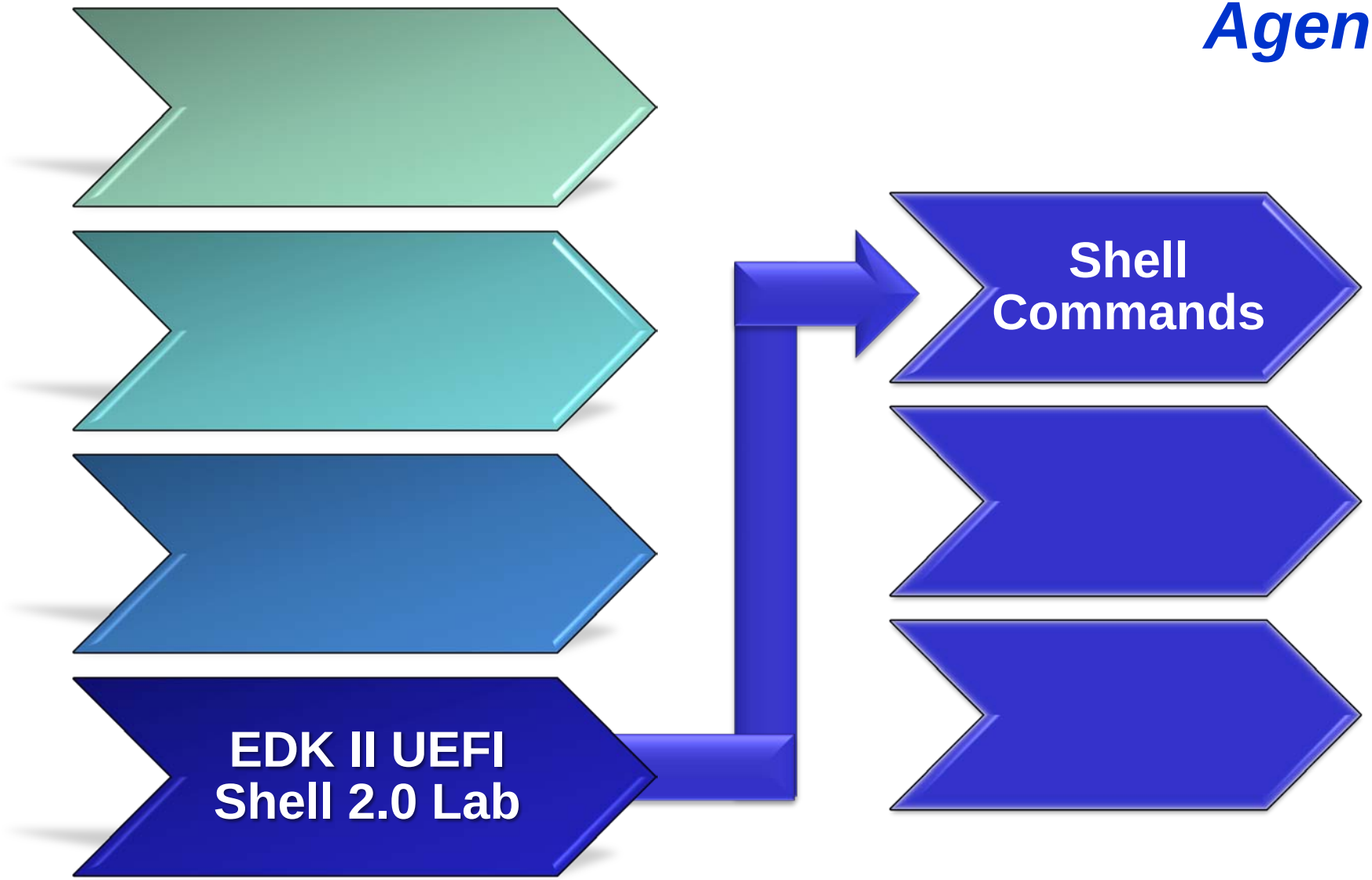


Boot to the EFI Shell



- "f12:" or "FS0:" – moves you onto a mapped hard disk.
- **"gEfiNt32PkgTokenSpaceGuid.PcdWinNtFileSystem|L".!..\\..\\..\\..\\EdkShellBinPkg\\Bin\\Ia32\\Apps"|VOID*|106" – PCD in DSC that controls mapping.**

Agenda



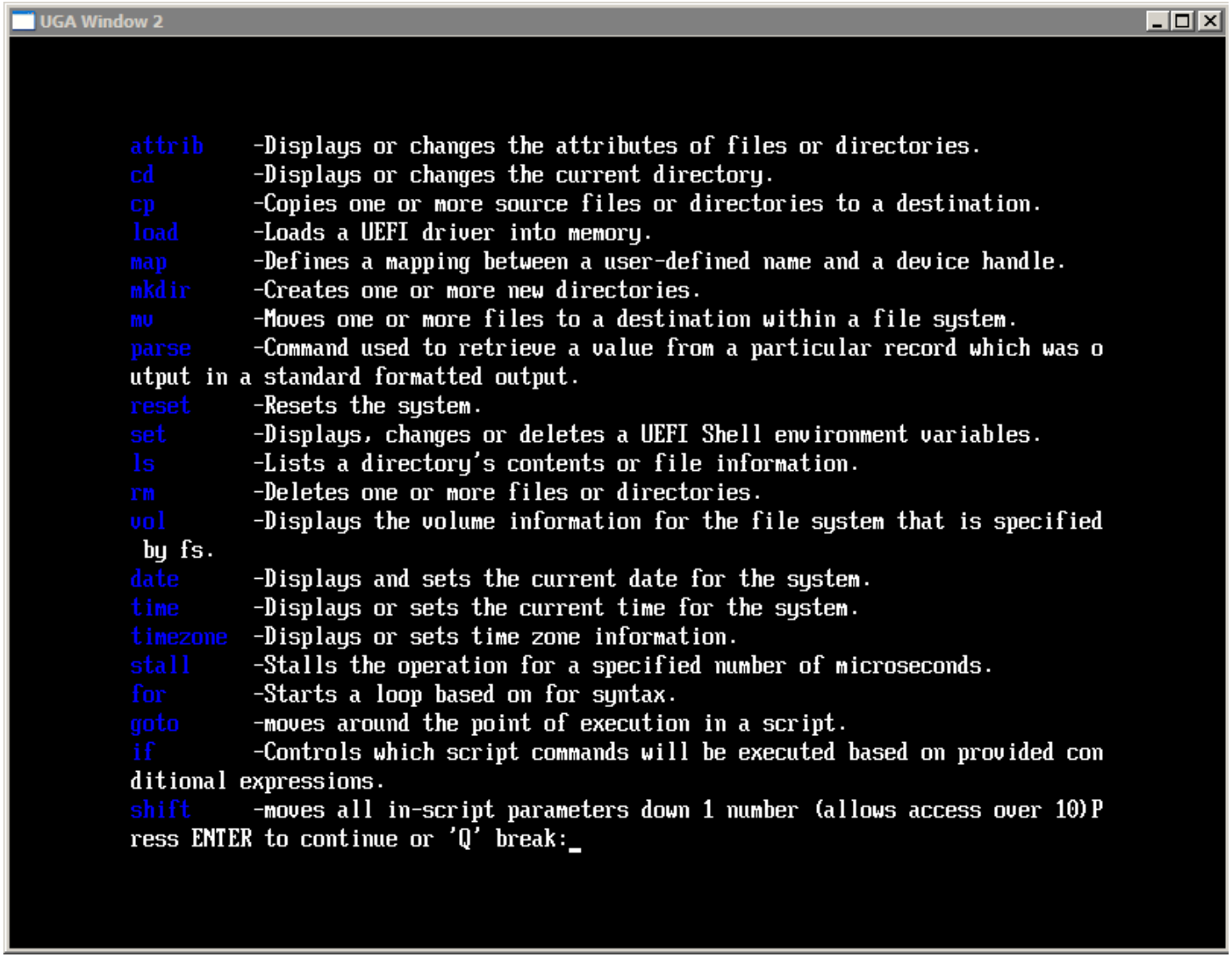
Common Shell Commands For Debugging

- Help
- MM
- Mem
- Memmap
- Drivers
- Devices
- Devtree
- Dh
- Load

-b is the command line parameter for halting output after each page.

EFI Shell Help

- Help –b
- Reset



mm – Displays or modifies memory, I/O or PCI resources

Usage:

MM Address [Width 1|2|4|8] [-MMIO | -MEM | -IO | -PCI] :Value –n

Address	- Starting address for MMIO, MEM, IO or PCI
Width	- Size accessed in bytes (1, 2, 4 or 8)
-MMIO	- MMIO Range (0 – 0xFFFFFFFF_FFFFFFFF) ;
-MEM	- Memory Range (0 – 0xFFFFFFFF_FFFFFFFF)
-IO	- IO Address Range (0 – 0xFFFF)
-PCI	- PCI Config Address (0x000000ssbbddffrr)
	ss = SEG, bb = BUS, dd = DEV, ff = FUNC, rr = REGISTER
Value	- Value to write
-n	- Non-interactive mode

Try:

>MM 0004BE6000 **Windows XP**

>MM 0005FA4000 **Windows 7**

mm – Displays or modifies memory, I/O or PCI resources *(cont)*

1. MEM type is the default
2. In interactive mode, type a hex value to modify. Enter ‘q’ or ‘.’ to exit.
3. Use the PCI command to discover the PCI device before using MM to modify PCI configuration space.
4. Use ‘-n’ mode inside of shell script files (*.nsh)
5. Not all PCI registers are writable. The PCI option will not do read-modify write. MM will only write the value posted.

mem – Displays contents of system memory or device memory

Usage:

MEM [-b] [address] [size] [-MMIO]

-b - Display one screen at a time

address - Starting Address (hex) to display. This needs to be an even address boundary for the processor the command is run on.

size - Number of bytes to display (hex)

-MMIO - Memory mapped I/O. Turns on any bits required to force memory access across out to the PCI bus

Note: **Run w/o args to see the system table entry pointer and all other system table pointer addresses.**

memmap – Displays memory map maintained by firmware

Usage:

MEMMAP [-b]

-b - Display one screen at a time

Note:

Use the *UEFI Specification** to lookup the memory type.

drivers – Displays list of UEFI drivers

Usage:

DRIVERS [-b] [-lXXX]

-b - Displays one screen at a time

-lXXX - Displays drivers using the ISO 639-2 language specified by XXX

Note:

Run DRIVERS -? to see the display format

devices – Display list of devices managed by UEFI drivers

Usage:

DEVICES [-b] [-lXXX]

-b - Displays one screen at a time

-lXXX - Displays devices using the ISO 639-2 language specified by XXX

Note:

Run DEVICES /? to see the display format

devtree – displays Tree of Driver Model Drivers

Usage:

DEVTREE [-b] [-d] [lXXX] [DeviceHandle]

-b	- Displays one screen at a time
-d	- Displays device tree using device paths
-lXXX	- Displays drivers using the ISO 639-2 language specified by XXX
DeviceHandle	- Displays device handle below a certain handle

dh – Displays handles

Usage:

DH [-b] [-d] [-lXXX] [-v] [handle] [-p prot_id]

-b	- Displays one screen at a time
-d	- Displays device tree using device paths
-lXXX	- Displays drivers using the ISO 639-2 language specified by XXX
-v	- Dumps information on all handles
handle	- Dumps information on a specific handle
-p prot_id	- Dumps all handles of a certain protocol

load – Load drivers

Usage:

LOAD [-nc] file [file…]

-nc	- Load the driver, but do not connect the driver
file	- File that contains the UEFI driver (.efi extension)

Note:

- LOAD can handle multiple files & supports wildcards
- Use the ‘UNLOAD’ command to unload an UEFI driver

stall – Stall the Processor

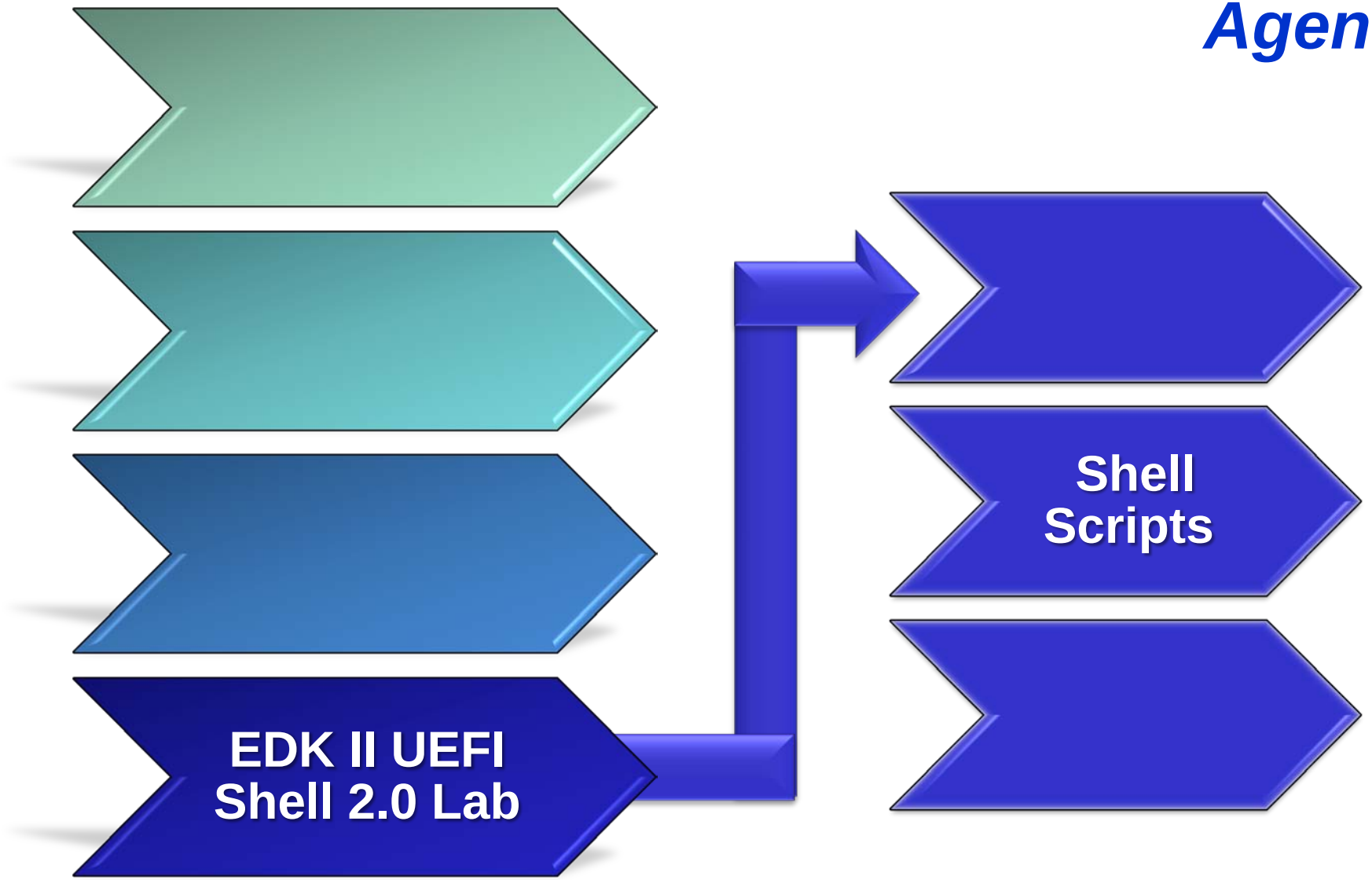
Usage:

STALL microseconds

Note:

- The ‘microseconds’ value is in decimal.
- STALL will cause the emulation environment to ‘sleep’ for the specified period.

Agenda



EFI Shell scripts

- The EFI Shell can execute commands from a file, which is called a *batch script file* (.nsh files).

- Benefits: These files allow users to simplify routine or repetitive tasks.
  - Perform basic flow control.
  - Allow branching and looping in a script.
  - Allow users to control input and output and call other batch programs (known as *script nesting*).

Lab - using EFI Shell Scripts

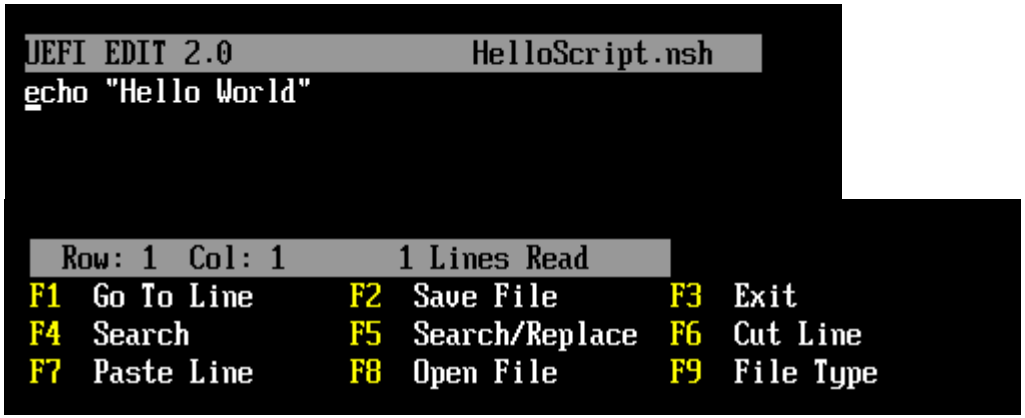
- Use a shell script to Print “Hello World”

1) Create text file in directory c:\Fw\Edk2\Build\NT32\DEBUG_MYTOOLS\IA32: (hint: use Shell “Edit”)

2.0 F12:\>edit HelloScript.nsh

2) HelloScript.nsh content:

echo “Hello World“



3) Use “F2” and then “F3” to save and EXIT

4)2.0 F12:\> HelloScript

Lab - using EFI Shell Scripts

- Use a shell script to run shell commands “Date” & “Time”

Create two text files in directory
c:\Fw\Edk2\Build\NT32\DEBUG_MYTOOLS\IA32: (use Shell “Edit”)

```
2.0 F12\:>edit script1.nsh
```

1) **Script1.nsh** content:

```
script2.nsh
if exist %cwd%Mytime.log then
      Type Mytime.log
endif
echo "%HThank you. %NByeBye : )%N“
```

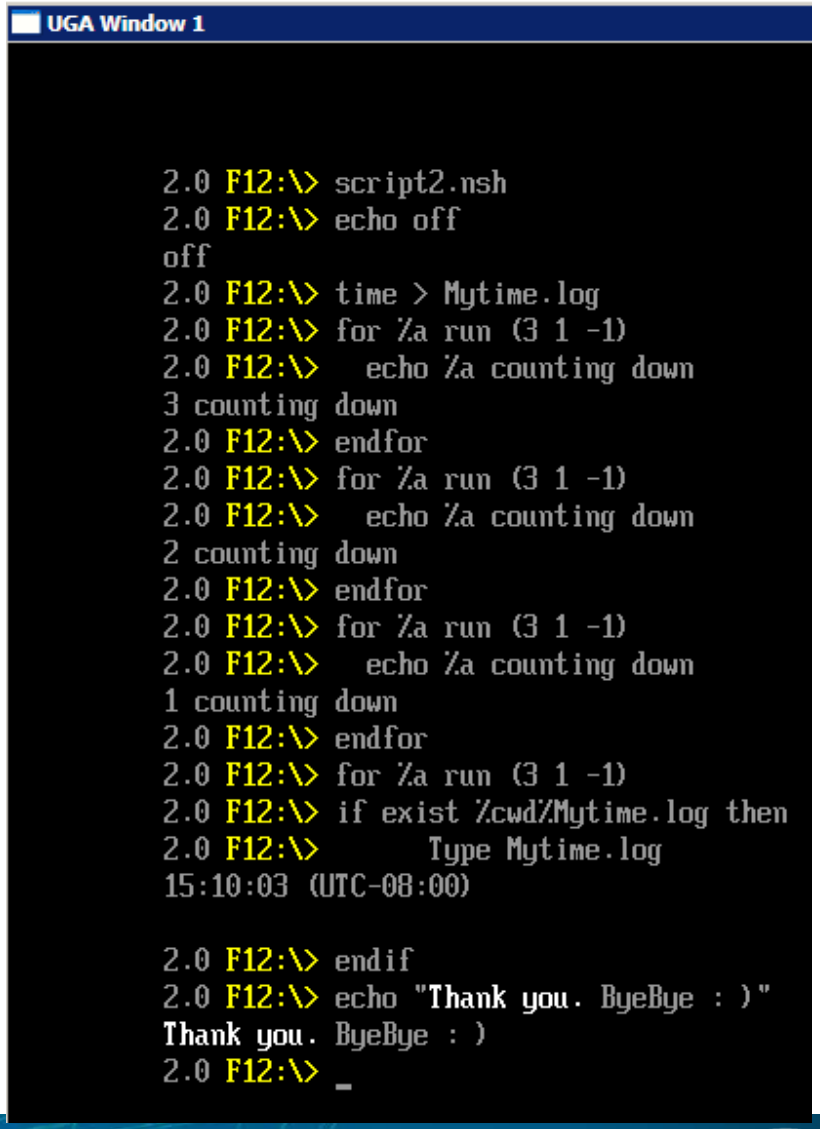
NOTE: “%H” and “%N” usage here

2) **Script2.nsh** content:

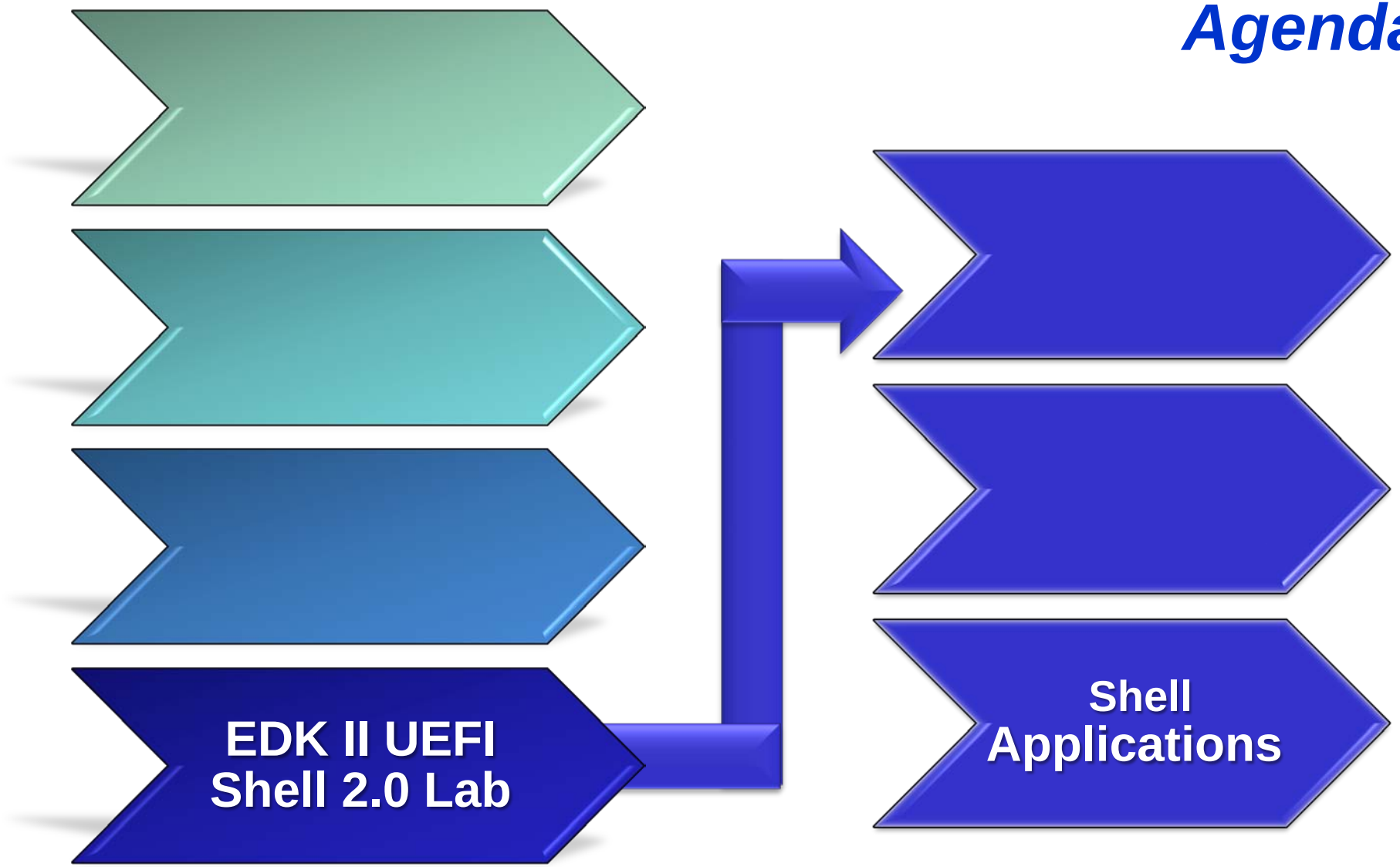
```
time > Mytime.log
for %a run (3 1 -1)
  echo %a counting down
  #Date
endfor
```

Run the Shell Scripts

- Build Run
- Boot to shell
- `2.0 Shell> F12:`
- Excute script1.nsh
- `2.0 F12:\>script1`



Agenda



Writing Shell Applications

- Simple UEFI Application

1. Hello World UEFI App.
2. Add Shell and HII
3. Shell with Script -GetChar

- Shell With Argc / Argv

4. Both UEFI Shell 2.0 &1.0
5. Using UEFI Shell 2.0 Library

- Shell with Files

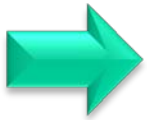
6. Static file name
7. File name from Argv

Simple UEFI Applications

UEFI Applications

Minimum files:

- Myfile.inf
- Myfile.c



```
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
  return EFI_SUCCESS;
}
```

```
[Defines]
  INF_VERSION
  BASE_NAME
  FILE_GUID
  MODULE_TYPE
  VERSION_STRING
  ENTRY_POINT
[Sources]
  MyFile.c

[Packages]
  MdePkg/MdePkg.dec

[LibraryClasses]
  UefiApplicationEntryPoint

[Guids]

[Ppis]

[Protocols]
```

Lab 1 – UEFI Application Hello World

Write a UEFI Application to Print the string "Hello World"

1. Create a directory \MyShellApps\Application\HelloWorld in the Workspace directory for Edk2
2. Copy, paste, and rename HW1.c.txt and HW1.inf.txt from the training material CD/Zip to that directory (Note: Lab-SampleCode/EdkIIShell/HW on CD/Zip)
   - Rename HW1.c.txt and HW1.inf.txt to HelloWorld.c and HelloWorld.inf
   - Note: The Build is case sensitive
3. Edit the HelloWorld.inf
   - Look in the INF for "XXXXXXXXXXX" sections that will need information
   - Create a Name and GUID, and then fill in the MODULE_TYPE
     - Hint: Guidgen or http://www.guidgenerator.com/

Application INF file

```
[Defines]
  INF_VERSION          = 0x00010005
  BASE_NAME            = XXXXXXXXXXX  <-- HelloWorld
  FILE_GUID            = XXXXXXXXXXX  <-- 1234ABCD ...
  MODULE_TYPE          = XXXXXXXXXXX  <-- UEFI_APPLICATION
  VERSION_STRING       = 1.0
  ENTRY_POINT          = UefiMain

[Sources]
  XXXXXXXXX    <-- HelloWorld.c
[Packages]
  #XXXXXXXX

[LibraryClasses]
  #XXXXXXXXXXXX

[Guids]

[Ppis]

[Protocols]
```

Application 'C' file

```
/** @file
  This is a simple shell application
**/
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
  return EFI_SUCCESS;
}
```

Does not do anything but return Success

Will it compile now?

- Not yet.
  - Need to add headers to the .C
  - Need to add a reference to INF from the platform DSC
  - Need to add a few Packages dependencies and Libraries to the .INF

Application Lab

- Updating the .INF .DSC and .C files
  1. .INF File
     - Packages
       - Basic: MdePkg
     - [LibraryClasses]
       - All applications: UefiApplicationEntryPoint
  2. .C file
     - Header references
       - <Uefi.h>
       - <Library/UefiApplicationEntryPoint.h>
  3. .DSC (Nt32Pkg\Nt32Pkg.dsc)
     - [Components . . .]

       . . .
       - Add Inf reference in components section and before the build options
     - [BuildOptions]

Solution

HelloWorld.inf - Notepad

File Edit Format View Help

```
[Packages]
  MdePkg/MdePkg.dec

[LibraryClasses]
  UefiApplicationEntryPoint
  UefiLib

[Guids]
```

HelloWorld.inf

HelloWorld.c.txt - Notepad

File Edit Format View Help

```
#include <Uefi.h>
#include <Library/UefiApplicationEntryPoint.h>

/**
  as the real entry point for the application.

  @param[in] ImageHandle    The firmware allocated handle for the EFI image.
  @param[in] SystemTable    A pointer to the EFI System Table.
```

HelloWorld.c

Nt32Pkg.dsc - Notepad

File Edit Format View Help

```
  MyShellApps/Applications/HelloWorld/HelloWorld.inf

##############################################################################################
##
#
# BuildOptions Section - Define the module specific tool chain flags that should be used as
#                        the default flags for a module. These flags are appended to any
```

Nt32Pkg/Nt32Pkg.dsc

Will it compile now?

Yes,

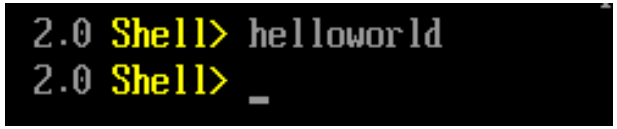
- Invoke the Build (see next slide if build errors)

```
C:\fw\Edk2> Build
```

- To test - use BUILD Run in the EDK2 directory
  - Run your application from the shell

```
C:\fw\Edk2> Build Run
2.0 Shell> HelloWorld
```

- When run, it will immediately unload because the main function is empty



If Build Errors

Solution

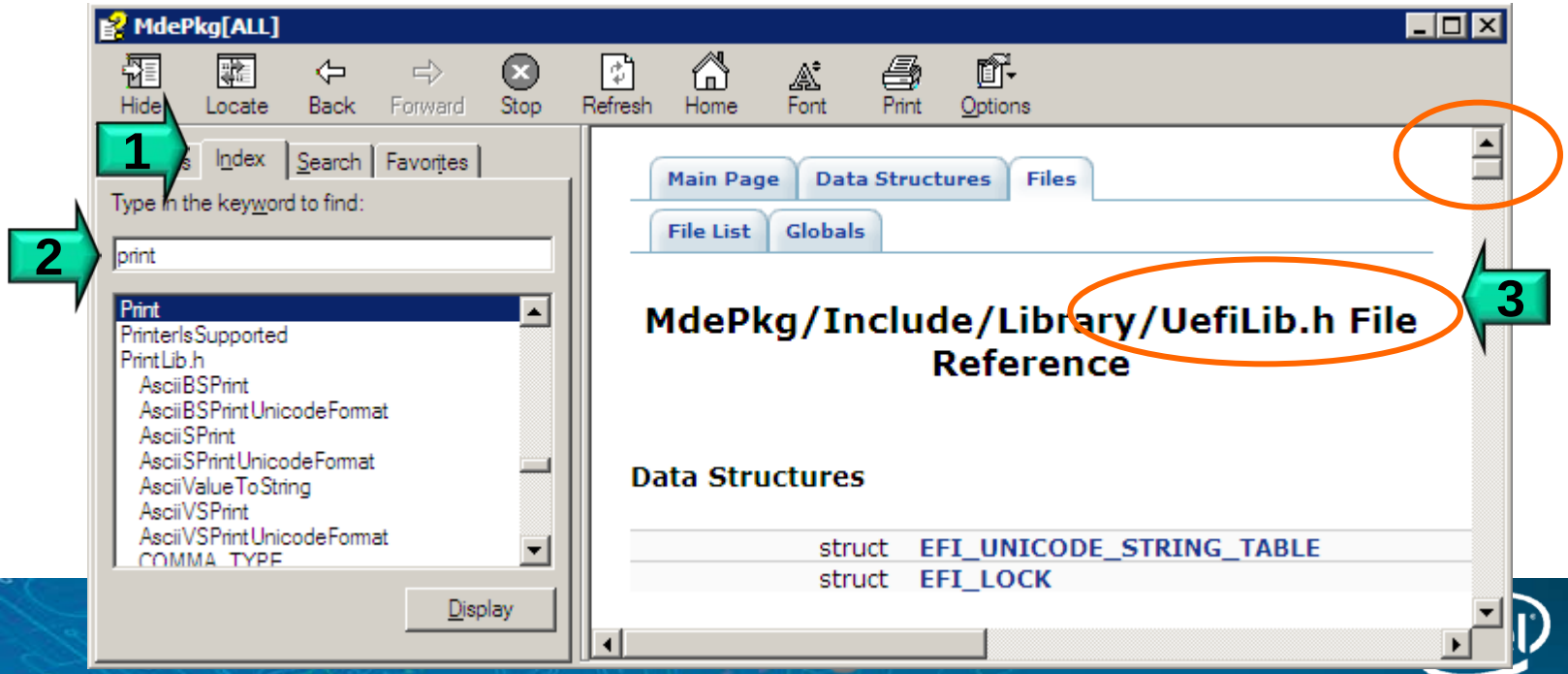
- See CD or ZIP for files: HW2.c.txt and HW2.inf.txt in directory
  - UEFI_Training…\Lab-SampleCode\EdkIIShell\HW
  - Copy to c:\Fw\Edk2\MyShellApps\Application\HelloWorld
  - Rename to HW2.c.txt and HW2.inf.txt to HelloWorld.c and HelloWorld.inf
- See Nt32Pkgxxxxx.dsc in
  - UEFI_Training…\Lab-SampleCode\EdkIIShell\HW
  - Search for "MyShellApps"
  - Add this line to c:\Fw\Edk2\Nt32Pkg\Nt32Pkg.dsc in your workspace
- Invoke Build again: `C:\fw\Edk2> Build`

Add "Print" Application

- Add code to make your application print to the console the string "Hello World"

Hint: use the MdePkg.chm to find where the "Print" function is located

1. Search the MdePkg.chm and find that the Print function by clicking on the "Index" tab
2. Type "Print" and double click
3. Scroll to the top in the right window to see that the print function is in the UefiLib.h file



Add "Print" to your Application

Solution - .C and .inf files

HelloWorld.c

```
#include <Uefi.h>
#include
   <Library/UefiApplicationEntryPoint
   .h>
#include <Library/UefiLib.h>

EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{

Print(L"Hello World\n\n");
  return EFI_SUCCESS;
}
```

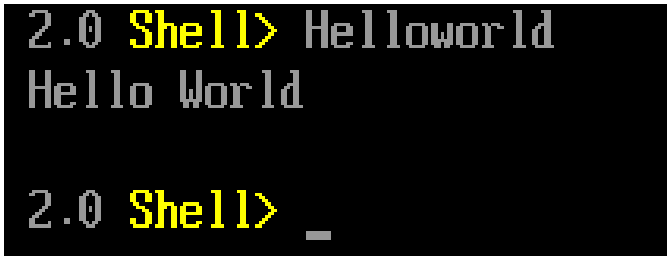
HelloWorld.inf

```
[LibraryClasses]
  UefiApplicationEntryPoint
  UefiLib
```

Test:

Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> HelloWorld
```




```
2.0 Shell> Reset
```

Note: Solution files on CD

Writing Shell Applications

- Simple UEFI Application
- Shell With Argc / Argv
- Shell with Files

✓ Hello World UEFI App.

2. Add Shell and HII
3. Shell with Script -GetChar
4. Both UEFI Shell 2.0 &1.0
5. Using UEFI Shell 2.0 Library
6. Static file name
7. File name from Argv

Simple UEFI Applications with UEFI Shell 2.0 and HII

UEFI Applications

Minimum files:

- Myfile.inf
- Myfile.c
- Myfile.h
- Myfile.uni

```
#langdef en-US  "English"
#langdef fr-FR  "Francais"

#string STR_MY_STRING
    #language  en-US
       “My String for Hello World!\n“
```

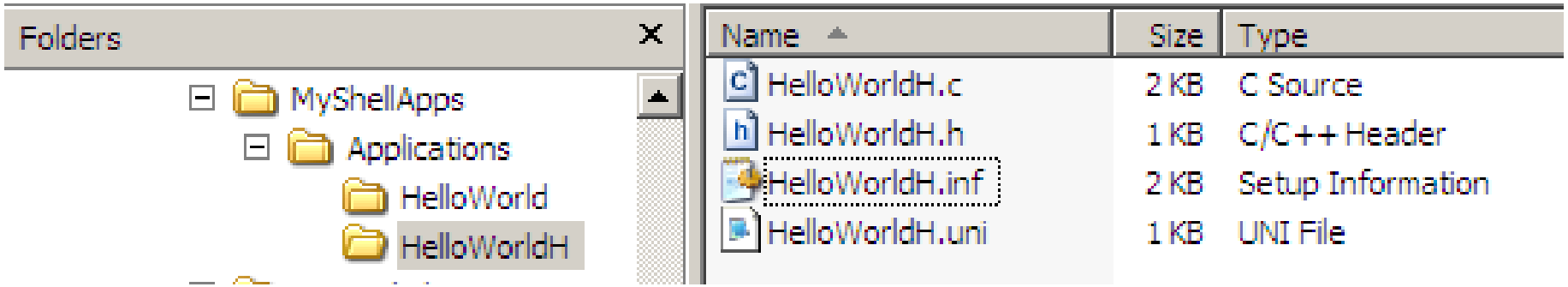
```
#ifndef _MYFILE_H
#define _MYFILE_H
#include <Uefi.h>

#include
<Library/UefiApplicationEntryPoint.h
   >
#include <Library/UefiLib.h>
#include <Library/HiiLib.h>
#include <Library/DebugLib.h>
#include <Library/ShellLib.h>

#define MYFILE_STRING_PACK_GUID \
   {0x6435f523, 0x5cd3,
    0x444f, 0xa4, 0x23, 0x83, 0x4f,
    0xbb, 0x71, 0x29, 0xc7};
#endif
```

Lab 2 –Hello World

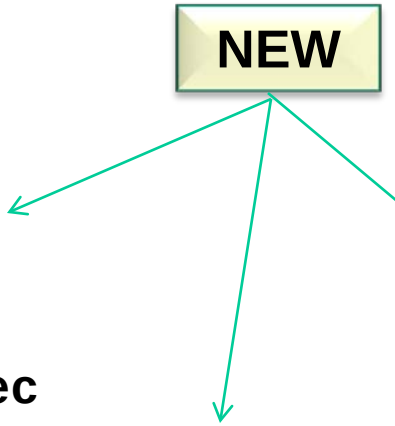
1. Copy the *HelloWorld* Directory form Lab 1 to *HelloWorldH* under the *MyShellApps/Applications* Direcotory
2. Rename HelloWorld.c and HelloWorld.inf to HelloWorldH.c and HelloWorldH.inf and update
3. Create HelloWorldH.h
   - Next slides will add content
4. Create HelloWorldH.uni
   - Next slides will add content



Lab 2 –. INF Hello World

- Update the .INF file

```
[Sources]
  HelloWorldH.c
  HelloWorldH.h
  HelloWorldH.uni
[Packages]
  MdePkg/MdePkg.dec
  MdeModulePkg/MdeModulePkg.dec
  ShellPkg/ShellPkg.dec
```




```
[LibraryClasses]
  UefiApplicationEntryPoint
  UefiLib
  HiiLib
  ShellLib
```

Lab 2 –. C for Hello World

- Update the .c file

```
#include "HelloWorldH.h"
extern UINT8
    HelloWorldHStrings[];

EFI_STATUS
UefiMain (
  IN EFI_HANDLE
     ImageHandle,
  IN EFI_SYSTEM_TABLE
     *SystemTable
  )
{
  EFI_HII_HANDLE       HiiHandle;
```

```
  // Register to HII
HiiHandle = HiiAddPackages (
     &gEfiCallerIdGuid,
     ImageHandle,
     HelloWorldHStrings, NULL);
ASSERT (HiiHandle != NULL);

ShellPrintHiiEx(-1,-1,NULL,
     STRING_TOKEN (STR_HELLOWORLD),
     HiiHandle);
ShellPrintHiiEx(-1,-1,NULL,
     STRING_TOKEN (STR_GOODBYE),
     HiiHandle);

return EFI_SUCCESS;
}
```

Lab 2 –. h for Hello World

- Create the .h file

```
#ifndef _HELLOWORLDH_H
#define _HELLOWORLDH_H


#include <Uefi.h>
#include
   <Library/UefiApplicationEntry
   Point.h>
#include <Library/UefiLib.h>

#include <Library/HiiLib.h>
#include <Library/DebugLib.h>
#include <Library/ShellLib.h>
```

GUID is Needed for HII Packaging

```
#define
   HELLOWORLD_STRING_PACK_GUID \
        {0x6435f523, 0x5cd3,
   0x444f, 0xa4, 0x23, 0x83, 0x4f,
   0xbb, 0x71, 0x29, 0xc7};

#endif
```

Lab 2 –. uni for Hello World

- Create the .uni file

Note “%B” “%N” “%V”

```
#langdef en-US  "English"
#langdef fr-FR  "Francais"

#string STR_HELLOWORLD #language en-US "%BHello, %Nworld!\n"
                       #language fr-FR "%BHello, %Nworld!\n"
#string STR_GOODBYE    #language en-US "%VGoodbye, world!%N\n"
                       #language fr-FR "%VGoodbye, world!%N\n"
```

Lab 2 –Nt32 .dsc Hello World

- Update the Nt32Pkg/Nt32Pkg.dsc file

```
[LibraryClasses.common.UEFI_APPLICATION]
  PcdLib|MdePkg/Library/BasePcdLibNull/BasePcdLibNull.inf
  PrintLib|MdeModulePkg/Library/DxePrintLibPrint2Protocol/\
     DxePrintLibPrint2Protocol.inf
  ShellLib|ShellPkg/Library/UefiShellLib/UefiShellLib.inf
  FileHandleLib|ShellPkg/Library/UefiFileHandleLib/\
     UefiFileHandleLib.inf
  SortLib|ShellPkg/Library/UefiSortLib/UefiSortLib.inf
```

Shell Libraries

```
[Components.IA32]

  .   .   .

    MyShellApps/Applications/HelloWorld/HelloWorld.inf
    MyShellApps/Applications/HelloWorldH/HelloWorldH.inf
```

Helloworld w/ HII

Lab 2 – Compile and run

- Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> HelloWorldH
```

- Notice the different text colors

IF Build Errors Solution files are on UEFI_Training

…\Lab-SampleCode\EdkIIShell\HWH

HWH2.* copy to HelloWorldH.* respectively and repeat

Writing Shell Applications

- Simple UEFI Application
- Shell With Argc / Argv
- Shell with Files

✓ Hello World UEFI App.
✓ Add Shell and HII
3. Shell with Script -GetChar
4. Both UEFI Shell 2.0 &1.0
5. Using UEFI Shell 2.0 Library
6. Static file name
7. File name from Argv

Shell Applications with Shell Scripts

UEFI Applications

Write an Appl. to Demonstrate the use both UEFI Protocols and UEFI Shell 2.0 Appl.

Minimum files:

- Myfile.inf
- Myfile.c
- Myfile.nsh

GetKey.c

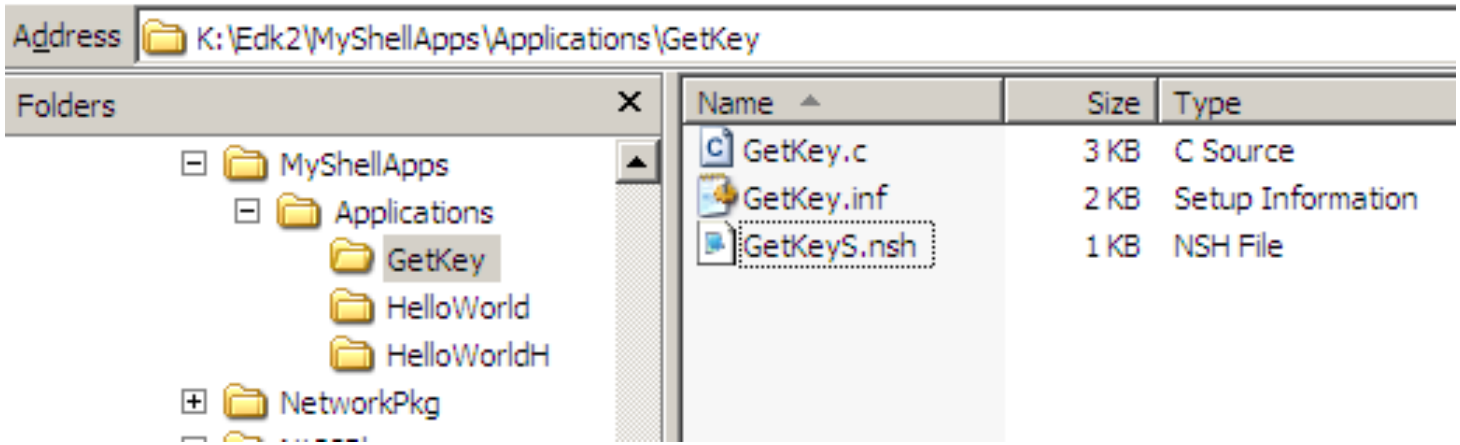
```
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
   EFI_STATUS    Status;
   // UEFI Calls and UEFI Shell
   // calls to get char from console
   // and store char in Environment
   // Variable
   return ( Status);
}
```

GetKeyS.nsh

```
echo "Select key for" _key
getkey
echo %H%_key%%N is the key in environment now
```

Lab 3 –Shell Script w/ Application

1. Copy the *HelloWorld* Directory form Lab 1 to *GetKey* under the *MyShellApps/Applications* Directory
2. Rename HelloWorld.c and HelloWorld.inf to GetKey.c and GetKey.inf and update
3. Create GetKeyS.nsh
   - Next slides will add content



Lab 3 –. INF Shell Script w/ Application

- Update the .INF file

```
[Defines]
  INF_VERSION    = 0x00010005
  BASE_NAME      = GetKey
  FILE_GUID      = 10C75C00-. . .
  MODULE_TYPE    = UEFI_APPLICATION
  VERSION_STRING = 1.0
  ENTRY_POINT    = UefiMain

[Sources]
  GetKey.c
[Packages]
  MdePkg/MdePkg.dec
  ShellPkg/ShellPkg.dec
```

```
[LibraryClasses]
  UefiApplicationEntryPoint
  UefiLib
  ShellLib
```

Lab 3 –. c Shell Script w/ Application

- Add code to make your application wait for an event (WaitForEvent) and use the (WaitForKey) as the event
- Use Shell Library to break on an escape from the WaitForKey event
  - Hint: use the MdePkg.chm to find where the “WaitForEvent” and the “WaitForKey” functions are located
  - Another Hint: The system table is passed in as a parameter to your sample application
  - Search the EDK II code for “WaitForEvent” ☺

Lab 3 – GetKey.c update

Solution – Search the MdePkg.chm

1. Search the MdePkg.chm and find where the “WaitForEvent” is located. It is part of the “Boot Services”.
2. Search the MdePkg.chm and find where the “WaitForKey” is located. It is part of the “EFI_SIMPLE_TEXT_INPUT_PROTOCOL as part of ConIn.
3. The WaitForEvent can be referenced through the Boot Services pointer which can be referenced through the System Table
4. The WaitForKey can be referenced through the System Table passed into our Sample application
5. Check the UEFI Spec for the parameters needed
6. An example can be found in MdePkg\Library\UefiLib\Console.c :

```
gBS->WaitForEvent (1, &gST->ConIn->WaitForKey,
   &EventIndex);
```

NOTE: *gST* is for Global System Table and
gBS is for Global Boot Services

Lab 3 –. GetKey.c

- Update the .c file

```
#include <Uefi.h>
#include <Library/
  UefiApplicationEntryPoint.h>
#include <Library/UefiLib.h>
#include <Library/
   UefiBootServicesTableLib.h>

#include <Library/ShellLib.h>
// uses Shell globals gSP & gSPP
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE     ImageHandle,
  IN EFI_SYSTEM_TABLE *SystemTable
  )
{
   EFI_STATUS  Status;
```

```
 UINTN         EventIndex;
 EFI_INPUT_KEY Key;
 EFI_EVENT     Events[2];
 Events[0] = gST->ConIn->WaitForKey;
 Events[1] = gSP->ExecutionBreak;
 Status = gBS->WaitForEvent (2,
   Events,&EventIndex);

 if (EventIndex == 1 || Status !=
   SHELL_SUCCESS) {
        return SHELL_ABORTED;
 }
 gST->ConIn->ReadKeyStroke
  (gST->ConIn, &Key);
 gSP->SetEnv(L"_key", (CHAR16 *)
   &Key.UnicodeChar, TRUE);

 return (EFI_SUCCESS);
}
```

Lab 3 –. GetKeyS.nsh and Nt32pkg.dsc

- Create the GetKeyS.nsh

```
echo "Select key for" _key
getkey
echo %H%_key%%N is the key in environment now
```

- Save File GetKeyS.nsh
  C:\Fw\Edk2\Build\NT32\DEBUG_MYTOOLS\IA32
  (MYTOOLS – VS2008x86)

- Update the Nt32Pkg/Nt32Pkg.dsc file add line

```
MyShellApps/Applications/GetKey/GetKey.inf
```

Lab 3 – Compile and run

- Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> GetKeyS
```

```
2.0 Shell> getkeys
2.0 Shell> echo "Select key for" _key
Select key for _key
2.0 Shell> getkey
2.0 Shell> echo %_key% is the key in environment now
Y is the key in environment now
2.0 Shell> set
     path = .\;FS0:\efi\tools\;FS0:\efi\boot\;FS0:\;FS
\;FS1:\
Lasterror = 0x00000000
profiles = ;Driver1;Install;Debug1;network1;
uefishellsupport = 3
uefishellversion = 2.0
uefiversion = 2.31
     _key = Y
DebugLasterror = 0x00000000
2.0 Shell> _
```

IF Build Errors Solution files are on UEFI_Training

…\Lab-SampleCode\EdkIIShell\GK

GK2.* copy to GetKey.* respectively and repeat

Writing Shell Applications

- Simple UEFI Application
- Shell With Argc / Argv
- Shell with Files

✓ Hello World UEFI App.
✓ Add Shell and HII
✓ Shell with Script -GetChar

4. Both UEFI Shell 2.0 &1.0
5. Using UEFI Shell 2.0 Library

6. Static file name
7. File name from Argv

Lab 4 –Both Shell 2.0 and 1.0 Args -Application

- Write a Shell Application to evaluate command line parameters to work in both UEFI Shell 2.0 and EDK Shell 1.0
- Hint: by using the GUID from the both Shell implementations we can determine which Shell appl. we are running inside of.

Start from our HelloWorld Appl.

1. Copy the *HelloWorld* Directory form Lab 1 to SampleArgs under the *MyShellApps/Applications* Directory
2. Rename HelloWorld.c and HelloWorld.inf to SampleArgs.c and SampleArgs.inf and update

Lab 4 –. INF Shell w/ Argc & Argv

- Update the .INF file

```
[Defines]
  INF_VERSION    = 0x00010005
  BASE_NAME      = SampleArgs
  FILE_GUID      = 10C75C00-. . .
  MODULE_TYPE    = UEFI_APPLICATION
  VERSION_STRING = 1.0
  ENTRY_POINT    = UefiMain

[Sources]
  SampleArgs.c
[Packages]
  MdePkg/MdePkg.dec
  ShellPkg/ShellPkg.dec
```

```
[LibraryClasses]
  UefiApplicationEntryPoint
  UefiLib
  ShellLib
```

Lab 4 –. c Shell w/ Argc & Argv

- Update the SampleArgs.c file

```
#include <Uefi.h>
#include <Library/
  UefiApplicationEntryPoint.h>
#include <Library/UefiLib.h>
#include <Library/
   UefiBootServicesTableLib.h>


#include <Protocol/
  LoadedImage.h>   // to get parms
#include <Protocol/
  EfiShellInterface.h> //Shell 1.0
#include <Protocol/
  EfiShellParameters.h>//Shell 2.0
```

```
EFI_STATUS  // Main using Argc &
EFIAPI      //   Argv
SampleMain (
  IN UINTN Argc,
  IN CHAR16 **Argv
  )
{
 UINTN  Index;
 Print(L"SampleArgs : called
     with %d parameters\n", Argc);
 for (Index = 0; Index < Argc;
                          Index++) {
    Print(L"Argv[%d]: %s\n", Index,
        Argv[Index]);
 }
 Print(L"\nDone getting
    parameters\n\n");
 return EFI_SUCCESS;
}
```

Lab 4 –SampleArgs. c Continued

```
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE     ImageHandle,
  IN EFI_SYSTEM_TABLE *SystemTable
  )
{ // Local vars
 EFI_STATUS      Status;
 EFI_SHELL_PARAMETERS_PROTOCOL    *EfiShellParametersProtocol;
 EFI_SHELL_INTERFACE              *EfiShellInterface;
 UINTN     Argc;
 CHAR16    **Argv;
 EFI_GUID  mEfiShellParametersProtocolGuid =
                      EFI_SHELL_PARAMETERS_PROTOCOL_GUID;
 EFI_GUID  mEfiShellInterfaceGuid =
                      SHELL_INTERFACE_PROTOCOL_GUID;
//Initialize local protocol pointers
EfiShellParametersProtocol = NULL;
EfiShellInterface = NULL;
```

Lab 4 –SampleArgs. c Continued

```
// check input parameters from command line using UEFI Shell 2.0
 Status = gBS->OpenProtocol(ImageHandle,
            &mEfiShellParametersProtocolGuid,
            (VOID **)&EfiShellParametersProtocol,
            ImageHandle,
            NULL,
            EFI_OPEN_PROTOCOL_GET_PROTOCOL
            );
 if (!EFI_ERROR(Status)) {
  // use shell 2.0 interface
   Print(L“Using UEFI Shell 2.0 Parameter Protocol\n");
   Argc = EfiShellParametersProtocol->Argc;
   Argv = EfiShellParametersProtocol->Argv;
    // Call our main with Argc / Argv parameters
   SampleMain ( Argc, Argv);
 }else
       //check if EFI Shell 1.0
```

Lab 4 –SampleArgs. c Continued

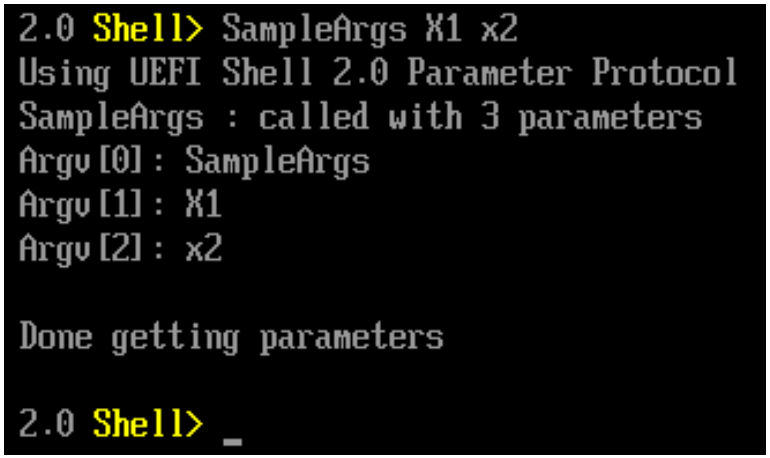
```
    {// else check if EFI Shell 1.0
       Status = gBS->OpenProtocol(ImageHandle,
                    &mEfiShellInterfaceGuid,
                    (VOID **)&EfiShellInterface,
                    ImageHandle,
                    NULL,
                    EFI_OPEN_PROTOCOL_GET_PROTOCOL
                    );
         if (!EFI_ERROR(Status))
         {
          Print(L“Using EFI Shell 1.0 Interface Protocol\n");
          Argc = EfiShellInterface->Argc;
          Argv = EfiShellInterface->Argv;
          SampleMain ( Argc, Argv);
          }else {
                Print(L"\nGetting Shell params did NOT work: \n");
          }
     } //End else for UEFI Shell 2.0

    return EFI_SUCCESS;
}
```

Lab 4 – Compile and run

- Update Nt32Pkg.dsc with the SampleArgs.inf
- Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> SampleArgs X1 X2
```



IF Build Errors Solution files are on UEFI Training

…\Lab-SampleCode\EdkIIShell\SA

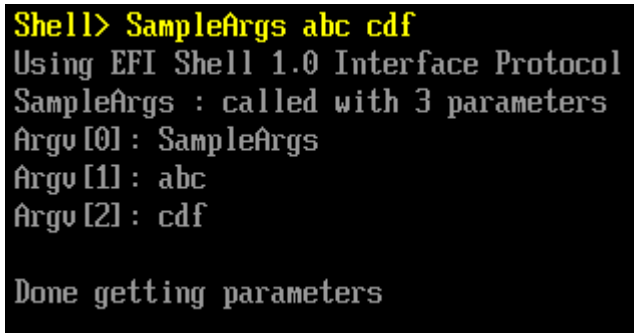
SA2.* copy to SampleArgs* respectively and repeat build

Lab 4 – Run in EFI Shell 1.0

- Invoke the EFI Shell 1.0 inside of the UEFI Shell 2.0

```
2.0 Shell> F13:
2.0 Shell> Shell_full.efi
```


```
Shell> SampleArgs abc cdf
```




```
Shell> Exit
```

- Exit Returns to UEFI Shell 2.0 interface

Writing Shell Applications

- Simple UEFI Application
- Shell With Argc / Argv
- Shell with Files

✓ Hello World UEFI App.
✓ Add Shell and HII
✓ Shell with Script -GetChar

✓ Both UEFI Shell 2.0 &1.0
5. Using UEFI Shell 2.0 Library

6. Static file name
7. File name from Argv

Lab 5 –Shell 2.0 Args w/ ShellLib

- Write a Shell Application to evaluate command line parameters to work in UEFI Shell 2.0 using the UEFI Shell 2.0 Library and not use EDK Shell 1.0

1. Create Directory SampleArgs2 under the *MyShellApps/Applications* Directory
2. Copy the Solution files on UEFI Training CD

…/Lab-SampleCode/EdkIIShell/SAS2 to
C:/fw/Edk2/MyShellApps/Applications/SampleArgs2

Rename SAS2.c.txt SAS2.inf.txt to SampleArgs2.c and SampleArgs2.inf

Lab 5 – Compile and run

- Edit SampleArgs2.c
  - Notice the differences from the SampleArg.c
    - No call to locate protocol for Shell 1.0
    - Using the Shell 2.0 global gSPP for the Shell parameters protocol
- Update Nt32Pkg.dsc with the SampleArgs2.inf
- Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> SampleArgs2 X1 X2
```

- NEXT: Simplify the SampleArgs2.c with using only the Shell Global gSPP variable and removing calls to locate the shell parameter protocol

Lab 5–. c Simplify using Shell Global

- Update the SampleArgs2.c file using gSPP

```
#include <Uefi.h>
#include <Library/
  UefiApplicationEntryPoint.h>
#include <Library/UefiLib.h>
#include <Library/ShellLib.h>

//No updates to SampleMain
EFI_STATUS  // Main using Argc &
EFIAPI      //   Argv
SampleMain (
//   . . .

EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
```

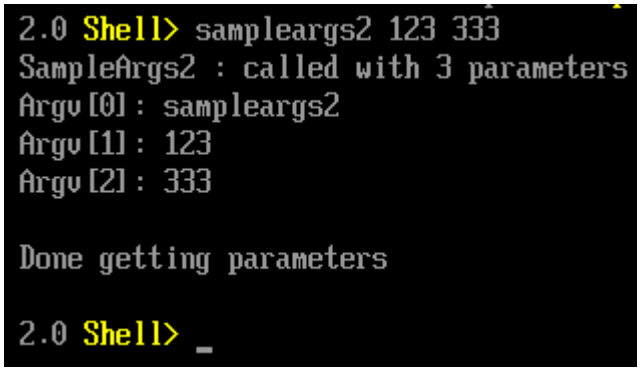
```
 UINTN          Argc;
 CHAR16         **Argv;

 if (gSPP != NULL){
    Argc = gSPP->Argc;
    Argv = gSPP->Argv;
 }else {
   Print(L"\nSorry, Getting Shell
         params did NOT work \n");
   return EFI_UNSUPPORTED;
 }
  // use shell 2.0 interface
   SampleMain ( Argc, Argv);
   return EFI_SUCCESS;
}
```

Lab 5 – Simplify using Shell Global

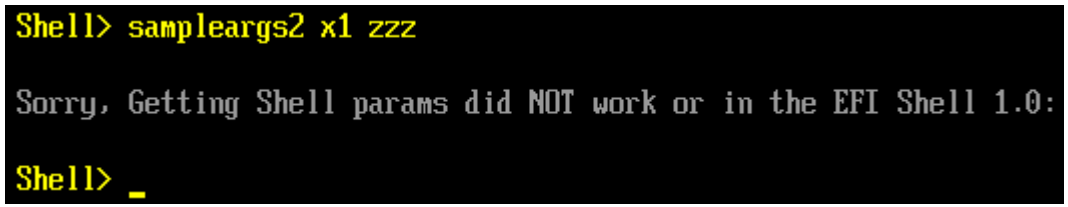
- Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> SampleArgs2 123
```



- Check EFI Shell 1.0

```
2.0 Shell>F13:
2.0 Shell>Shell_full.efi
Shell> SampleArgs x1 zzz
```



Writing Shell Applications

- Simple UEFI Application
  - ✓ Hello World UEFI App.
  - ✓ Add Shell and HII
  - ✓ Shell with Script -GetChar
- Shell With Argc / Argv
  - ✓ Both UEFI Shell 2.0 &1.0
  - ✓ Using UEFI Shell 2.0 Library
- Shell with Files
  6. Static file name
  7. File name from Argv

Lab 6 –Shell 2.0 Files

- Write a Shell Application to Open a file with a static file name as part of the .c file

1. Create Directory SampleFile under the *MyShellApps/Applications* Directory
2. Copy the Solution files on UEFI Training CD

…/Lab-SampleCode/EdkIIShell/SF to

C:/fw/Edk2/MyShellApps/Applications/SampleFile

Rename SF2.c.txt SF2.inf.txt to SampleFile.c and SampleFile.inf

3. Update Nt32Pkg.dsc with the SampleFile.inf
4. Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> F12:
2.0 Shell> SampleFile
2.0 Shell> Dir test.txt
```

You will need to be in a file system WHY?

See the result file is 0 length

Writing Shell Applications

- Simple UEFI Application
  - ✓ Hello World UEFI App.
  - ✓ Add Shell and HII
  - ✓ Shell with Script -GetChar
- Shell With Argc / Argv
  - ✓ Both UEFI Shell 2.0 &1.0
  - ✓ Using UEFI Shell 2.0 Library
- Shell with Files
  - ✓ Static file name
  - 7. File name from Argv

Lab 7 –Shell 2.0 Files w/ Args

- Write a Shell Application to Open a file with the name passed through on the parameter line.

1. Create Directory SampleFileA under the *MyShellApps/Applications*
2. Merge the changes from Lab 6 SampleArgs2.c and Lab 7 SampleFile.c as well as the changes from Lab 6 SampleArgs2.inf and Lab 7 SampleFile.inf
3. Also consider user error conditions like if no name is given on the command line
4. Update Nt32Pkg.dsc with the SampleFileA.inf
5. Invoke the Build

```
C:\fw\Edk2> Build
C:\fw\Edk2> Build Run
2.0 Shell> F12:
2.0 Shell> SampleFileA
2.0 Shell> Dir
```

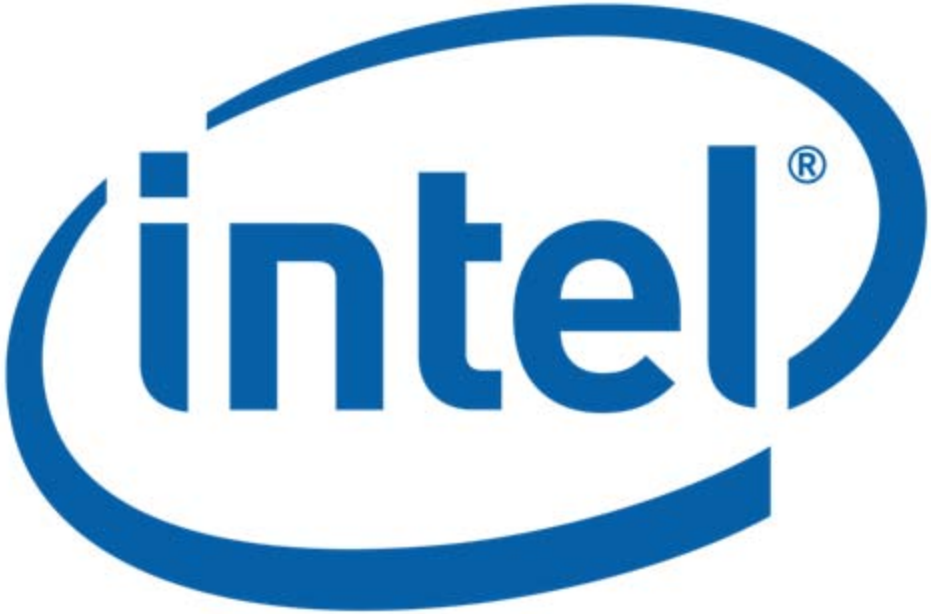
Summary

- The EDK II emulation platform can be used to debug individual modules, prior to creating a real platform
- The EFI Shell have a rich set of commands that can help with debugging UEFI applications and UEFI drivers.

Additional Resources

- UEFI Specification Web
  - http://www.uefi.org
- UEFI open source Code http://www.tianocore.org
  - EFI Developer Kit (EDK II)
    - http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=EDK2
      - Nt32, DUET, OVMF and Unix environments
      - Shell binaries
    - EDK II Documentation: http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=Documents
    - EDK II Library Documentation http://sourceforge.net/projects/edk2/files/EDKII_Libraries/
  - UEFI Shell 2.0 – Part of the EDK II packages
    - http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=EDKII_Packages
    - UEFI shell Documentation: http://sourceforge.net/apps/mediawiki/tianocore/index.php?title=ShellPkg

- Questions?



Backup