

# Randy Marques Consultancy

## Embedded Software Development

---



## GBS User

Generic Build Support  
for Users  
6.00

# Introduction - Who am I

---

- Randy Marques - CASE Consultant
  - Retired
  - CEO / Owner Randy Marques Consultancy
  - Nederlands Normalisatie Instituut (NEN)
    - Nederlandse Programmeertalen Commissie (NC 381 22)
      - WG14 (International ANSI-C Committee)
  - Teach at various Universities and Colleges
- “Consultancy by Walking Around”
  - Software Engineering since 1971
  - Coding Standards since 1978
  - Build Automation since 1980
  - C Programming since 1983
  - Static Analysis since 1993
  - Les Hatton’s Safer C<sup>TM</sup> trainer since 2001



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Concept

---

- GBS is a concept
  - Understand the concept and GBS will help you
  - Refuse to understand GBS: it will work against you
  - Main purpose is to support the project
    - Individual needs are second to the project needs
- Basics:
  - Simplicity
  - Straightforward
  - Consistency
  - No Tricks
  - No Exceptions
  - No 'clever' solutions
  - No 'private' scripts



# Features

- Fully portable and relocatable directory structure
- Multiple platform support (Win10/WSL/Linux)
- Same physical directory structure used for all platforms (on shared network-drives)
- Generated, full compliant 'make' files
  - 100% reliable builds
  - Cross reference
- Allows subdivision into SubSystems and Components
- Any number of SubSystems and/or Components
- Any number of libraries and/or executables per Component
- Strict applicable scoping rules
- Support for generation of 3rd party software
- Integrated support for any compiler
- Integrated support for Auditing tools like QAC, QAC++, PCLint and ++Test
- Integrated support for Documentation tools like Doxygen
- No user-written scripts
- Support for multi-site environments
- Command-line oriented
  - GUI available
- Support for GUI integration (e.g. Visual Studio, SlickEdit, Eclipse, Notepad++)
- Automated directory creation and structure setup
- Independent from Configuration Management System (CMS)
  - CMSs supported (for automated structure creation): Git and SubVersion
- Parallel generation (also in 'grid')
- Background generation ('at' jobs) with extensive logfile
- Prepared for tools like 'Softfab', 'BuildForge', 'Hudson' and 'CruiseControl'
- Uniform way of working
- Simple in use. Easy to learn. **Powerful** due to simplicity and consistency
- Suitable for small, medium and large systems
- Only dependent on Perl (Version 5.10 or later)



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Build Automation Basics

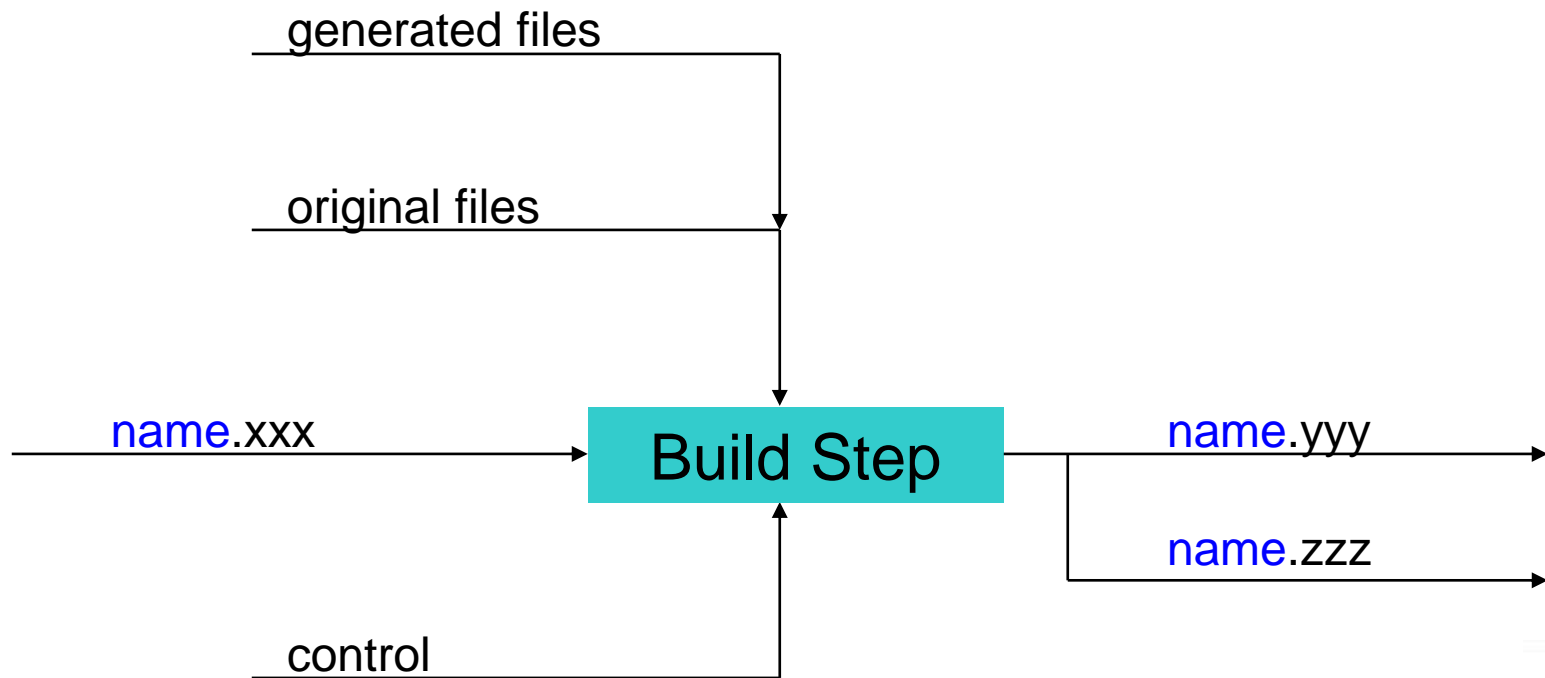
---

- Building of Software:
  - Sequence of build-steps
  - Some steps use results of previous steps
    - Pre-compile, Compile, Archive (lib), Link, Locate



# Build Automation Basics

- Anatomy of a Build Step





# Build Automation Basics

---

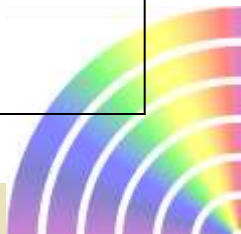
- name.xxx: Main input
  - Source
- Generate concurrent for more than 1 platform: generated files must be placed in different directories for various platforms
- Most Archivers and/or Linkers do not have a 'main-input' file. So we need to do something special here.



# Build Automation Basics

- Generating an executable (linking)
  - Traditionally done in 'make' file
  - Link-file
    - Works the same way as 'compile file'
    - **name.glk => name.exe**
    - Contains:
      - <component>:<objectfile-name>
    - Also:
      - .include ...

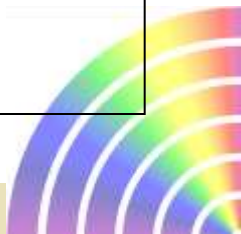
- `Name.glk => Name.exe`
- Contents:
  - A:a.o
  - A:a1.o
  - B:b.o
  - B:b1.o
  - C:c.lib



# Build Automation Basics

- Generating a library (archiving)
  - Also traditionally done in 'make' file
  - Library-file
    - Works the same way as 'compile file'
    - **name.glb => name.lib**
    - Contains:
      - <component>:<objectfile-name>
    - Also:
      - .include ...

- `Name.glb => Name.lib`
- Contents:
  - A:a.o
  - A:a1.o
  - B:b.o
  - B:b1.o



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Directory Structure

---

- Purpose: to support the Build Process
- Fully relocatable
  - No Absolute Directory Paths
- Environment Variables
  - Set inside (part-of) the directory structure
- Levels:
  - System
  - SubSystem
  - Component
  - Sub\_directory
- Directory Scoping is used to support the Build Process, not the software architecture



# Directory Structure

---

- SubSystem
  - An independent generation-unit within GBS
    - A directory-structure with files that, during generation, produce software that can be delivered (released) to other SubSystems and/or end-customers.
    - Not per se an architectural SubSystem
    - Contains one or more Components
    - The number of SubSystems should be limited
      - Most Systems will have only one SubSystem!



# Directory Structure

---

- Component
  - A files-container within a GBS SubSystem
    - Lowest level directory-tree in GBS  
Here the source and object files reside.
    - Not per se an architectural component
    - May very well contain more than one architectural component and/or parts of architectural components.
    - Files in Components cannot refer to files located inside Components of other SubSystems



# Directory Structure

---

- Deliverable
  - Set of files produced by a SubSystem for use
    - in another SubSystem and/or
    - as final product(s).
  - One or more libraries with one or more header-files.
  - A whole directory structure with executables, start-up scripts, icons, data, etc.
- Build
  - Sequence of generation steps for a specific build, with a specific compiler using the same set of compile-options, possibly followed by archiving, linking, etc., resulting in a deliverable.





# Directory Structure

---

- System:
  - EXT (externals) Directory
    - 3rd party SW Directories
  - DEV (Development) Directory
    - SubSystem Directories
  - RES (Results) Directory
    - SubSystems Transfer Directories
  - SYS
  - SYSBUILD
    - Generation scripts per Build
  - SYSAUDIT
  - SYSTOOL
  - SYS
  - SILO
  - TMP



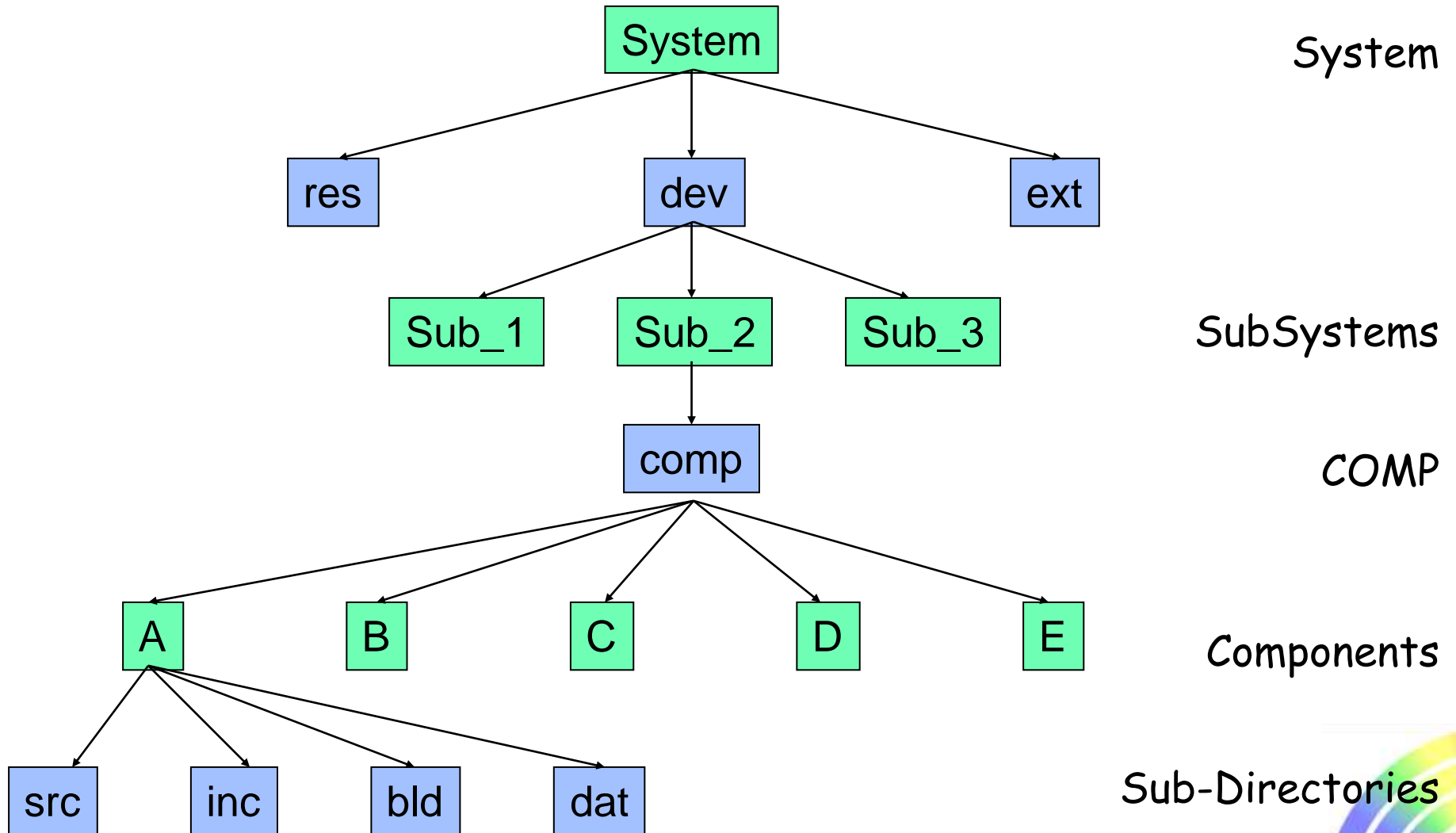
# Directory Structure

---

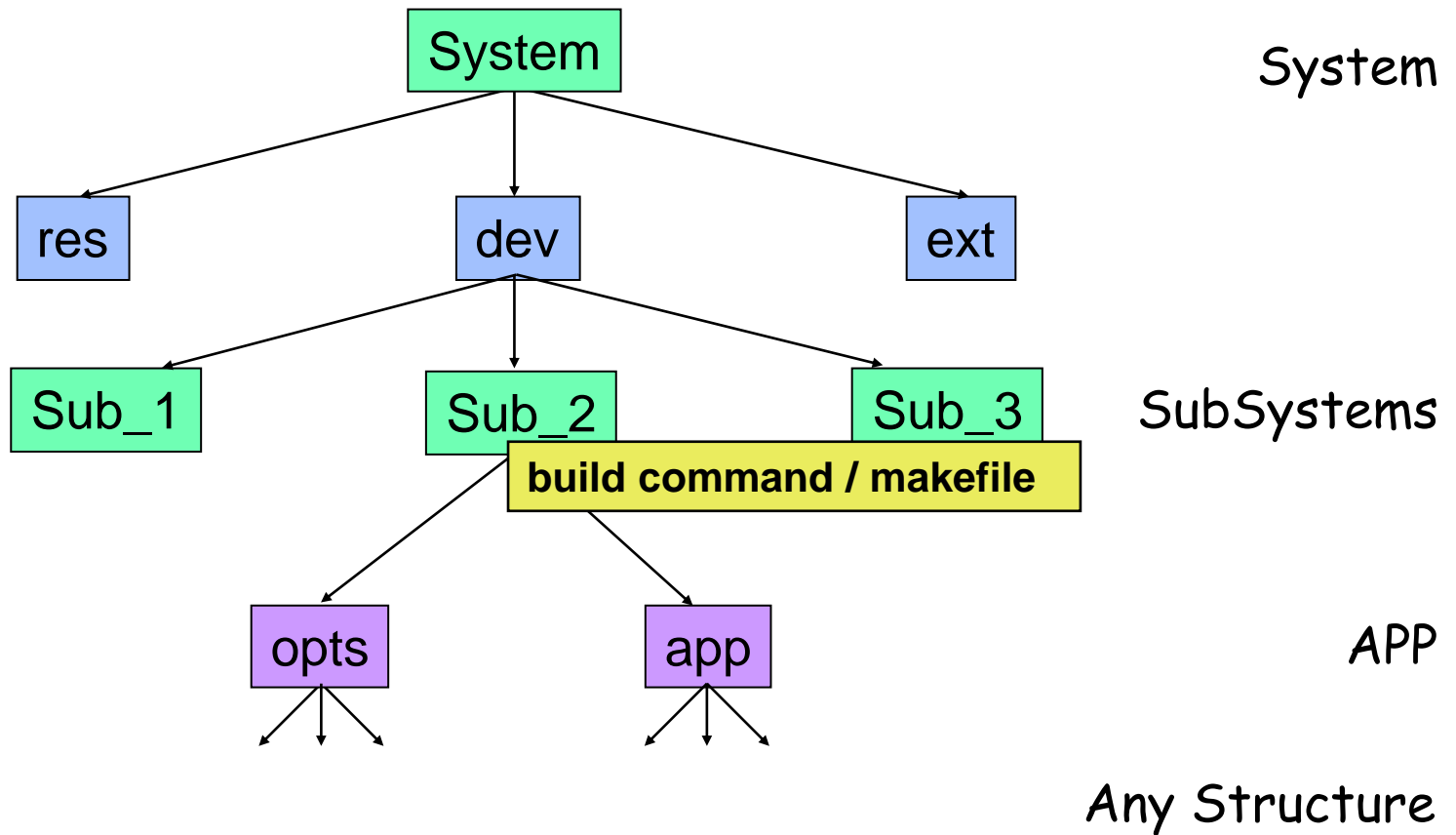
- SubSystem Directory: All
  - BUILD-directory
  - AUDIT-directory
  - TOOL-directory
  - EXPORT-directory (optional)
  - IMPORT-directory (optional)
- SubSystem Directory: Full GBS
  - COMP-Directory
    - Component Directories
- SubSystem Directory: Non GBS
  - For 'make', Visual Studio and Other types of SubSystems
  - APP-directory
  - OPTS-directory
  - generation scripts



# Directory Structure: Full GBS



# Directory Structure: Non Full GBS



# Directory Structure

- Component Sub-directories
  - SRC
    - Sources
  - INC
    - Global (exported) Header-files
  - LOC
    - Local Header-files
  - BLD
    - Contains <build>-Directories
      - Results of building (compilations, archiving, linking)
  - AUD
    - Contains <audit>/<build>-Directories
  - DAT
  - SAV
  - OPT



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Diversity

---

- Creating programs with varying functionality:
  - Platform Diversity
  - Hardware Diversity
  - Functionality Diversity
- Types:
  - Archive Diversity (SCMS diversity)
  - Compile time Diversity
  - Link time Diversity
  - Run time Diversity



# Diversity

---

- Compile-time Diversity

```
MAKE-FILE:
```

```
    -D RECORDER
```

```
FILE.C:
```

```
    #ifdef RECORDER
```

```
        ...
```

```
        ...
```

```
    #else
```

```
        ...
```

```
        ...
```

```
    #endif
```





# Diversity

- **CFG1.C:**

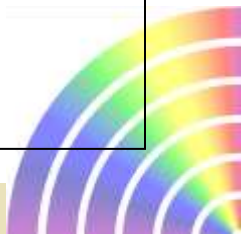
```
bool recorder( void)
    { return TRUE; }
```

- **CFG2.C:**

```
bool recorder( void)
    { return FALSE; }
```

- **FILE.C:**

```
if (recorder())
{
    ...
    do_recorder();
} else
{
    ...
    ...
}
```



# Diversity

---

- Link-time and Run-time diversity combined
  - Link FILE.O either with CFG1.O or CFG2.O
  - Need not be static
    - Read a file (.ini)
    - Read Hardware Memory (jumpers)
    - Ask user



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Scope Control I

---

- Focusing on Essentials & Structuring
  - Organising things
  - Keeping the same things together
  - Postponing decisions / Stepwise refinement
- Daily examples
  - Library
  - Warehouse
  - Dictionary
  - Nails, Screws and Bolts
  - Our Eyes
- Military
  - Defense against frontal assault
  - Target distribution



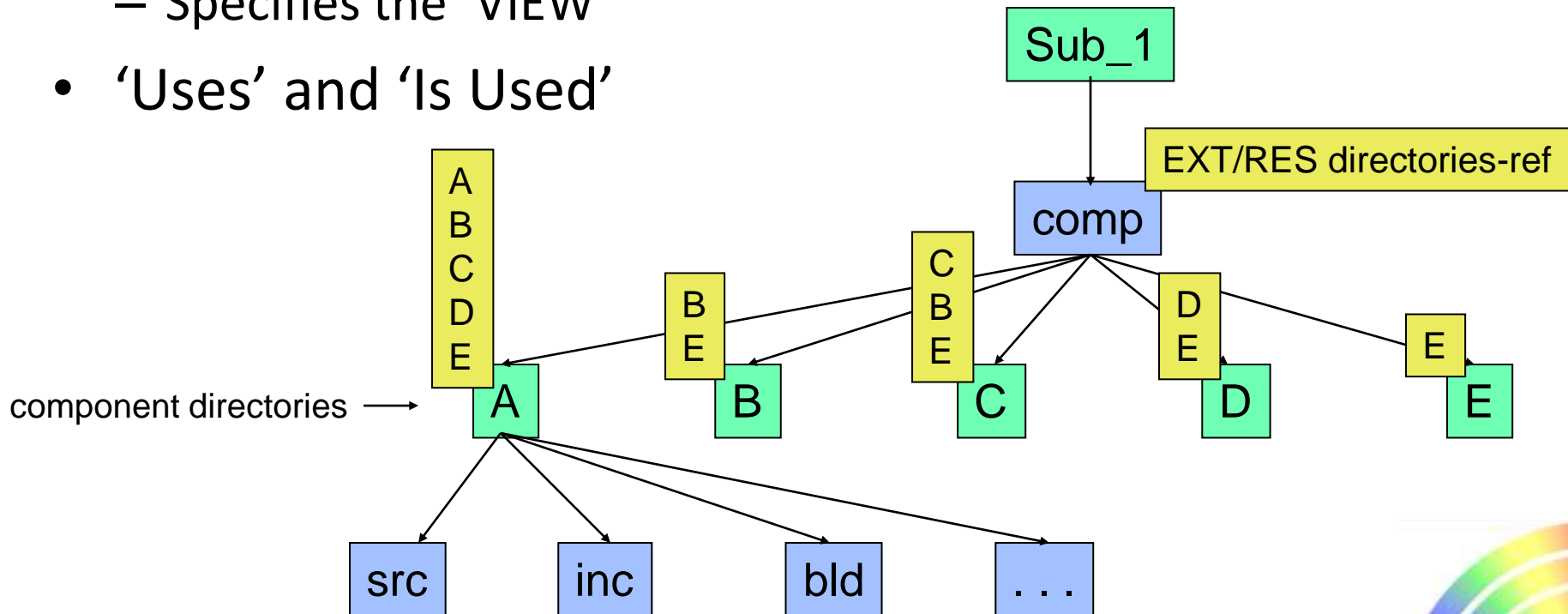
# Scope Control II

- Electronic Hardware
  - Chip on Board (Yellow wire?)
  - Board on Backplane (Yellow wire?)
  - Backplane in Cabinet (Yellow wire?)
  - Group of Cabinets (Yellow wire?)
- Software
  - Block - Inside a block
  - Block - Inside a function
  - Function – Inside a function / block
  - Function – Inside a file
  - File – In a Component
  - Component – In a SubSystem
  - SubSystem – In a System
- GBS supports strict scope control



# Scoping

- Scope-files contain component-names, no directory specs.
- SCOPE.GBS in Component Directory
  - Specifies the 'VIEW'
- 'Uses' and 'Is Used'



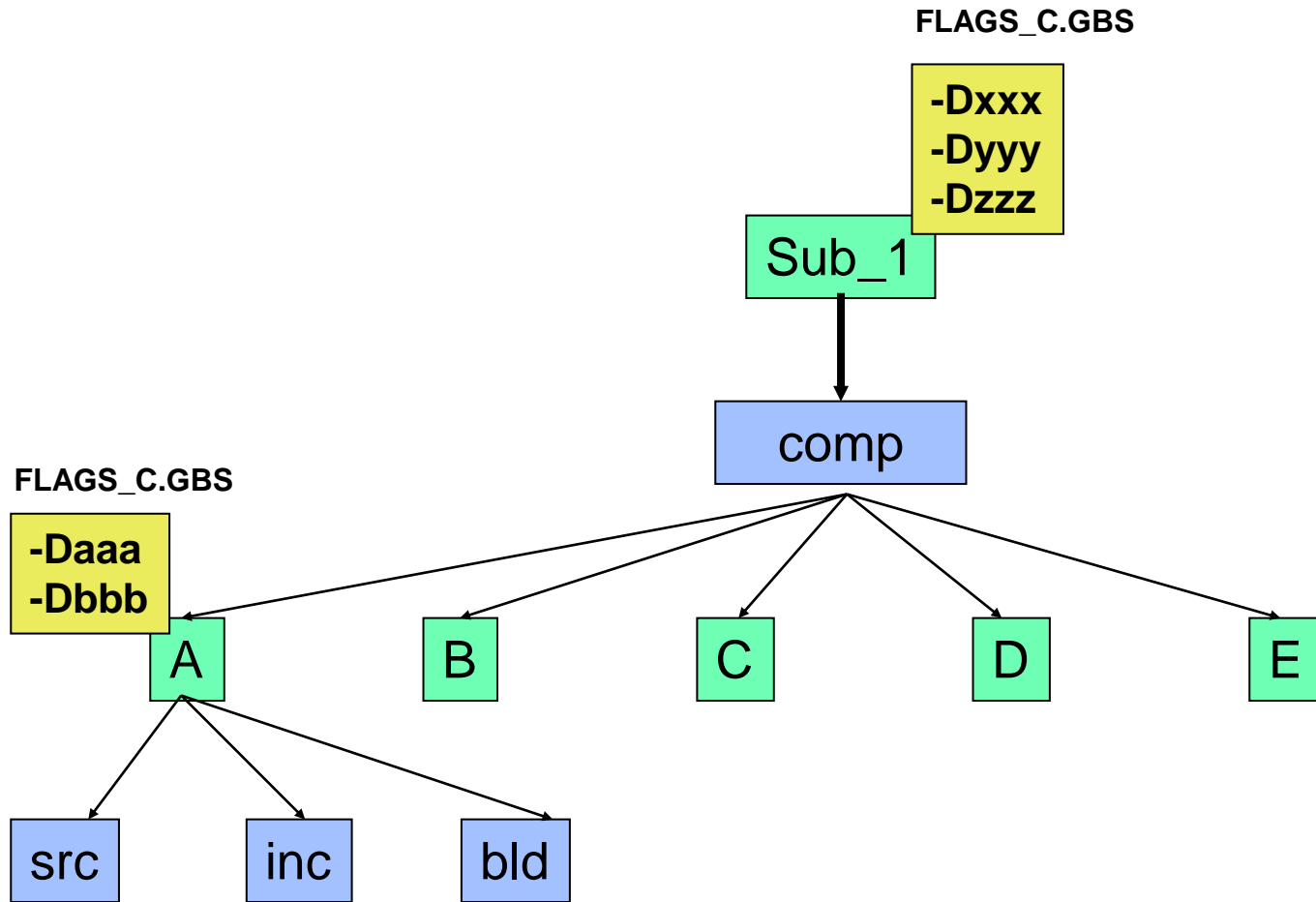
# Compile-time Options - 1

---

- Options are placed separately in option-files
  - Options for all C-files in project:
    - FLAGS\_C.GBS In System SYSBUILD directory
  - Additional Options for all C-files in a SubSystem:
    - FLAGS\_C.GBS In SubSystem BUILD directory
  - Additional Options for all C-files in a Component:
    - In FLAGS\_C.GBS in Component OPT directory
- For compilation, options are placed in the order specified above.
  - Last option wins...



# Compile-Time Options - 2





# Generating a Compilation

- Given a source file of the current Component of the current SubSystem of the current System with a current Build, we have:
  - Source File name (file.c)
  - Compiler to be used
  - Extension of object-file name (.o)
  - Object-file name (file.o)
  - Header-File Directory information
  - Compile Options Information
  - Input & Output Directory
- So we can have a generic script that generates a dedicated compile command.



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Beginning with GBS

---

- Perl
- Install GBS
- Setup GBS
- Startup GBS
- Preset Environment Variables
- Global Personal Environment Variables
- Global GBS Environment Variables
- Global Project/System Environment Variables



# Beginning with GBS - Perl

- Perl
  - GBS uses Perl-scripts
  - Perl must be installed and the *perl* command must be executable either via
    - The PATH
    - or
      - Addressed by Environment Variable GBS\_PERL\_PATH
  - Do not install Perl in a directory that starts with '5'
    - Use v5
  - You need at least Perl 5.16.3 preferably with **PerlTkx**
    - On Linux PerlTkx must be installed separately with ppm (Perl Package Manager)



# Beginning with GBS – Install GBS

- Multiple versions of GBS can be installed:
  - <anyroot>/GBS/<GBS\_SCRIPTS\_REL>/
  - <anyroot>/GBS
    - <anyroot> : GBS\_SCRIPTS\_ROOT
    - May not contain whitespace
    - Location
      - Central Network Drive (slower – always up-to-date)
      - On each machine
  - GBS\_SCRIPTS\_REL
    - <version>
      - Latest version - overwritten
    - <version>\_<build>
      - Specific version
  - GBS\_SCRIPTS\_PATH
    - <GBS\_SCRIPTS\_ROOT>/<GBS\_SCRIPTS\_REL>



# Beginning with GBS – Install GBS

---

- Unzip to a new temp directory
- 'cd' to that directory
- Win32:
  - Run Install.bat
- Linux
  - chmod ugo+x Install.sh
  - ./Install.sh
- Answer questions
- Delete the unzip directory



# Beginning with GBS – Setup GBS

- This part may be skipped if it was already done during Install
- Initial setup of GBS (once only)
  - ‘cd’ to your GBS\_SCRIPTS\_PATH
  - Run:
    - *\_setup.bat* (Win32)
    - *.\_setup.sh* (Unix/Linux) (Mind the dot!)
- **.gbs** directory
  - ‘Starting-point’ for GBS
  - Created in:
    - %APPDATA% (Win32)
    - ~/ (Linux)



# Beginning with GBS – Setup GBS

---

- During setup:
  - Windows
    - A GBS startup-shortcut is created on your desktop
  - Linux:
    - Your `~/.bashrc` file(s) are updated to contain a *gbs* command
    - If you have a GUI:
      - A GBS startup-shortcut is created on your desktop





# Beginning with GBS – The Command Line

---

- Answering questions:
  - Possible values are between ()
  - Default value is between []
    - Enter a single space if empty value (not the default) is wanted
  - Enter ! to quit processing safely
  - Enter ? to get help (usually not available ☹)



# Beginning with GBS – Startup GBS

- Windows
  - Double-click on the GBS Startup shortcut
  - Enter:  
gbs
  - Note:
    - GBS runs on Win10 (and probably WinXP, Vista, Win7 & Win 8)
- Linux
  - With GUI
    - Double-click on the GBS Startup shortcut
  - No GUI
    - Open an X-term window
  - Enter:  
gbs
  - Note:
    - GBS runs ONLY on the Bourne-Again-shell (bash)
- Answer the questions



# Beginning with GBS – First use

- GBS maintains a list of Systems (work-areas) per user
  - Yes! In the .gbs directory!
    - No! Do not try to be clever!
- To add an existing System (checked out work-area):
  - `cd` to the GBS\_SYSTEM\_PATH directory (containing dev, etc)
  - `swr --add`
- List added Systems:
  - `swr`
- Help: `gbsman` and/or `gbshelp`



# Preset Environment Variables

---

- Manually
- Define in:
  - Windows: Registry (Advanced System Settings: Environment Variables)
  - Linux: ~/.profile (~/.bash\_profile, ~/.bash\_login)
- Names:
  - GBS\_PERL\_PATH



# Global Personal Environment Variables

---

- Defined and changed by:
  - gbssettings => Profile
- Items:
  - GBS\_SITE
  - GBS\_LOG\_ROOT
  - GBS\_BEEPS
  - GBS\_HOME



# Configuration Settings

---

- Defined and changed by:
  - gbssettings => Config
- Items:
  - Company Name
  - Ignores
  - Editor /Browser / Viewer
  - Terminal
  - Navigator
  - Administrator / Integrator



# Global GBS Environment Variables

- You can put global definitions of Plugin ROOT paths in a separate file (site.gbs.sh or site.gbs.bat) assuming that all your Plugins on your PC have the same ROOT path.
  - You can remove/comment the `_ROOT` definitions from your `switch.gbs` files
- To create or modify:
  - `gbsedit site.gbs`
  - Note:
    - Do not try to be 'clever' with this file
    - Placing any 'clever' stuff in this file may cause GBS to malfunction. If not today: definitely in the future.



# Global Project/System Environment Variables

---

- Where:
  - GBS\_SYSTEM\_PATH/switch.gbs (.bat/.sh)
- Items:
  - GBS\_TEMPLATE\_PATH
  - GBSEXT\_scm\_PATH
  - GBSEXT\_compiler\_etc\_locations
  - More later...





# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# The GBS Commands In General

- All commands have the format:
  - `command [args | gbs-options]... | [gbs-environment]...`
    - `gbs-options` always start with `--` (minus minus)
- General options (always available):
  - `command --h` will give you short help
  - `command --h option...` will give you short help on the specified option(s)
  - `command --help` will give you more extensive help
  - `command --help option...` will give you long help on the specified option(s)
- `gbs-environment`:
  - `<name>=<value>` (GBS\_ may be omitted)



# The GBS Commands in general

- Messages are always preceded by the name of the command in uppercase
- All commands return a status
  - 0 == OK (Linux, Windows)
- Prompts:
  - Possible values are between ( )
  - Default value is between [ ]
  - To abort
    - ^C (bad way!)
    - !<enter> (good way!)
  - Example
    - Choice (1-3) [3] :
  - '?' Gives help - if available
  - Sorry, no command-history in Unix Perl (maybe in a later release)



# GBS Navigation Commands

- GBS works with currencies:
  - Current System
    - GBS\_SYSTEM\_PATH
  - Current SubSystem (remembered per System)
    - GBS\_SUBSYS
  - Current Component (remembered per SubSystem)
    - GBS\_COMPONENT
  - Current Build (remembered per System)
    - GBS\_BUILD
- Setting GBS currencies
  - Set Current System: swr
  - Set Current SubSystem: sws
  - Set Current Component: swc
  - Set Current Build: swb
  - Show currencies: gbs



# GBS Navigation Commands

- Navigating GBS Directories:
  - `cdsystem`
  - `cddev, cdres, cdext, cdsysbuild`
  - `cdsub`
  - `cdcomp, cdbuild, cdaudit`
  - `cdcomponent`
  - `cdsrc, cdinc, cdloc, cdbld, cdsav, cddat`
  - `cdbuild`
- Caution: Never change a GBS Environment Variable!  
Use the GBS commands to do that
- Caution: Never create GBS files and/or directories!  
Let GBS commands do that (`swc, swb, sws, swr`)



# Building

- Generate one or more items from a file in the 'src' directory to one or more files in the 'bld/<build>' directories.
  - E.g: compilation, creating a library, linking
  - File-extension specifies the type of build that is required (e.g.: \*.c → C-compile)
- Specific, generic rules:
  - The source file is an argument in the command-line and is taken from the current 'src' directory
    - More than one file from the 'src' directory may be specified
    - Wildcards are honored.
  - The object-files will be written to the current 'bld/<build>' directories.
    - The name of the object-file will be equal to the name of the source-file.
    - Filename extensions may differ and will be specific for various platforms. i.e.: name.c → name.obj or name.o
    - If the build fails, name.\* will be deleted from the 'bld' sub-directories



# Building: Include Paths

- Include path (-I/-L) will be assembled in the following order:
  - The current 'loc' directory
  - The current 'inc' directory.
  - The 'inc' directories of the other components within the same SubSystem:
    - In the order and as specified in the 'scope.gbs' file
  - As specified in the external reference file in the 'build/<build>' directory to be able to include stuff from the 'import', 'res' and 'ext' areas.
- Use GBS command 'gbswhich' to show the path



# Building : -D options

- The build-time options (-D) will be assembled in the following order:
  - Fixed Build settings for the whole project as defined in the option-file in the 'sysbuild/<build>' directory.
  - If present: Build-time options explicit for a specific SubSystem in the option-file in the current 'build/<build>' directory.
  - If present: Build-time options explicit for a specific component in the option-file in the current 'opt' directory per Build.
  - If present: Build-time options as specified in the command-line  
This means that options specified on the command-line will win.
- Use GBS command `'gbswhich'` to show all -D options





# Building: The comp-file-spec

- `gbsbuild`, `gbsmake` (and `gbsaudit`) have the same type of syntax: the comp-file-spec
- `<comp_file-spec>`:
  - `[<component>:]<file-comma-list>`
  - wild-cards allowed
  - e.g:
    - `file1.c`            `file1.c` in current Component
    - `A:*.c`             All `*.c` files in Component A
    - `*:*.*`             All files in all Components of current SubSystem
  - If `<component>` is omitted then current component is taken
  - If `<component>` is specified, this component becomes the current component for the duration of the execution.
  - More than one `<comp-file-spec>` can be specified
  - `-D` options are passed to preceding files in list



# Building: Make

- Makefiles:
  - Generated
    - Implicitly with `gbssysbuild`
    - Explicitly with `gbsmakemake`
  - Flavours:
    - `gbsmake ALL`
    - `gbsmake <component-list>`
    - `gbsmake <comp-file-list>`
  - make-files are generated per Build and per SubSystem
  - **Never** check-in a GBS generated make-file!
  - `gbsxref` uses `gbsmakemake` information to generate a GUI-controlled cross-reference



# Building

- Specifying options
  - `gbsbuild` and `gbsaudit` also accept `-D` options
  - Environment variables in the format `GBS_FLAGS_<type>` define options to be used for a specific compiler/linker/etc
  - Specifying `GBS_FLAGS_<type>="-D..."` defines the environment variable for the duration of the execution



# Building

- GBS recognizes a few built-in options:
  - **Compilation:**
    - DEBUGGER
      - YES, NO
    - MODE
      - DEBUG, ASSERT, FINAL, PROFILING
    - OPT
      - YES, NO, SIZE, SPEED, DEBUG
  - **Linking**
    - DEBUGGER
      - YES, NO
    - MAP
      - YES, NO
- **How to specify:**
  - `gbsbuild *.c MODE=DEBUG`
  - **Or via SetEnv:** `GBS_MODE=DEBUG`



# Building

---

- The difference between 'build' and 'make'
  - build '
    - you specify the source (e.g. file.c)
    - only the specified file(s) will be built
    - all the specified files will be built
  - 'make'
    - you specify the resulting file (e.g. file.o)
    - other files (even in other components) may be built
    - specified files may or may not be (re-)built

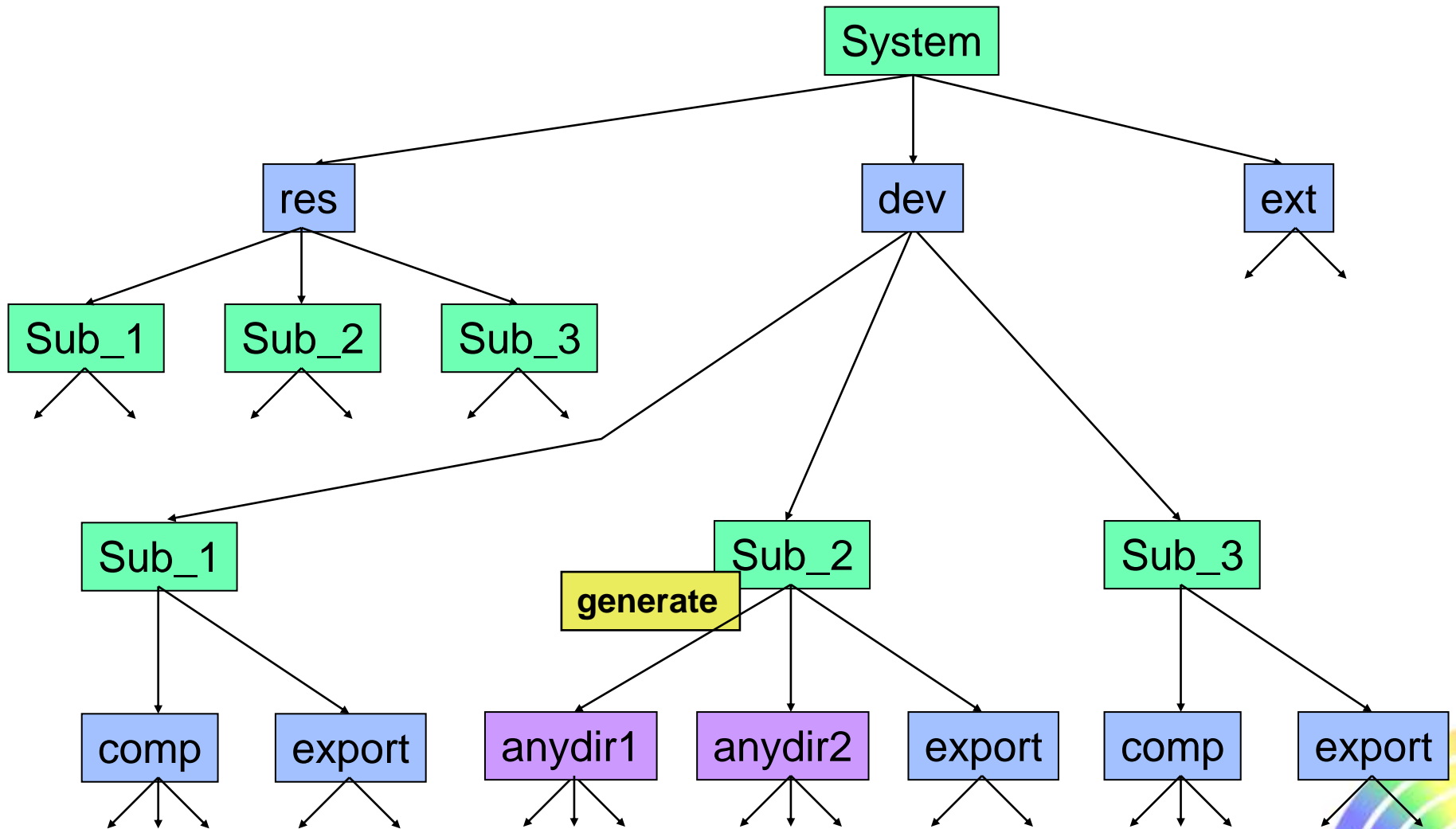


# Exporting: gbsexport

- Creates the 'deliverables' of a SubSystem
- Copies the various elements from within the SubSystem to the `export` and/or `res/<subsys>` directory
- Existence of `export` and/or `res/<subsys>` directory specify actions to be taken.
- Build-sensitive
- A whole directory-tree can be created in `export`
- Every Component can have an 'export.gbs' file
  - Specifies which files of that component are to be exported to a specific sub-directory in `export`
- 'smart'-copy: file-attributes (date-time) remain unchanged
- Syntax later...



# Exporting



# Generating on a higher level

- `gbssysbuild`, `gbssysmake`, `gbssysaudit`
  - Concept of 'steps':
    - Subsystem
    - Script
  - Specify step or range of steps
  - `gbsexport` included
  - Runs in batch-mode
    - Results to log-file
    - Can be started with a delay
    - On Unix you can shutdown your terminal ('at')





# General Commands

- `gbs`
  - You can always enter the `gbs` command
  - It will display your currencies
- `gbsman`
  - The GBS manual pages
- `gbsmaint`
  - An assortment of maintenance and cleanup functions
- `gbswhich`
  - Show compile-options, include-paths, location of header-files etc
- `gbsedit`
  - Allows you to Create/Edit GBS specific files
- `gbsstats`
  - Gives statistics on nr. of files, components, etc
- `gbssilo`
  - Generate the silo HTML pages and start the browser
- `gbssettings`
  - (Re-)define GBS EnvVars in an controlled way



# Building Commands

---

- `gbsbuild`, `gbsmake`, `gbsaudit`
  - For Files and Components
- `gbsmakemake`
  - Creates a make-file
- `gbssysbuild`, `gbssysmake`, `gbssysaudit`
  - For SubSystems /scripts and total System



# General Commands – Non GBS Specific

---

- wordrep
  - Batch Replace words in file(s)
- filerep
  - Batch Rename files
- proto
  - Create C, C++ & Perl function-prototypes
- bgpids
  - Shows PIDs of Background jobs (Linux only)
- pgrep
  - 'grep' based on Perl regular expressions
- On Windows:
  - grep, tail, which



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Guarantee Portability and Relocatability

- Never specify an Absolute Path
- Use GBS environment variables
  - GBS\_SYSTEM\_PATH
  - GBS\_EXT\_PATH
  - GBS\_RES\_PATH
  - etc...
- And/or environment variables defined in the switch.gbs file
  - All must be prefixed GBSEXT\_
  - use 'entry' part to set the variables
  - use 'exit' part to unset
    - GBSEXT\_ EnvVars will be unset automatically
  - use Env. Variable GBS\_SITE to distinguish between sites



# Customizing GBS

---

- LOG directory:
  - GBS\_LOG\_PATH
- Browser, Viewer, Editor
  - GBS\_BROWSER, GBS\_EDITOR, GBS\_VIEWER
- Beeps (Alarm/Bell)
  - GBS\_BEEPS
- Make (careful!!)
  - GBS\_MAKE
- Batch, Foreground and Background processing:
  - GBS\_BATCH, GBS\_SUBWIN, GBS\_SUBMIT
- Background processing:
  - GBS\_BG\_NOTIFIER



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping and Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# GBS Files

---

- General
  - All \*.gbs files:
    - Ignore blank lines
    - Ignore lines starting with '#'





# GBS Files

---

- Creating GBS directories and files
  - Never create GBS directories and/or GBS files by yourself
  - GBS will do that for you, ensuring that all directories and files are created properly and are added to the SCM System, only if needed.
  - If you need a new component enter
    - `swc --new`
  - If you need a new gbs-file, use
    - `gbsedit`



# switch.gbs

- System switch.gbs:
  - Executed when an Switch System (swr) is executed
    - When entering a System with parameter 'entry'
    - When leaving a System with parameter 'exit'
  - There is always a switch.gbs file
- Used to setup (and cleanup) the environment for a specific System
- All EnvVars must be prefixed with GBSEXT\_
  - Note: No '\_' between GBS and EXT
- Do not rely on settings in .profile / .kshrc and/or Registry!!



# GLK/GLB files

- General:
  - A line that starts with '#' is ignored
  - A line that starts with '.include glkb-file' performs an include of the specified glkb-file.
  - Included glkb-files are searched according to the general include-path mechanism and must be placed in inc, loc, ext or res directories.
  - Empty lines are ignored
- Specific
  - Absolute file-specifications must not be used!
  - GBS\_BLD\_<in\_file\_type> environment variables for the current Build are set.  
Generic glkb-files suitable for various Builds.  
i.e.:
    - `file1$GBS_BLD_C` -> file1.o or file1.obj
    - `file2$GBS_BLD_ASM` -> file2.o or file2.obj



# GLKB files

- Lines contain specifications for the linker. eg.:
  - object-files, libraries and flags
- The following types of lines are input to the linker:
  - 'Absolute' file/library reference:
    - A line that starts with a '\$' or '%' is presented to the linker as-is
  - Files/Libraries from the current Build directory:
    - *name*  
Are prefixed with `../bld/<build>/` before presented to the linker
  - Files/Libraries from a specific component:
    - *component:name1*  
Are prefixed with `component-dir/bld/<build>/`
  - Files/Libraries from external directories (specified with -L option)
    - *+name1 name2*  
Are presented to the linker as-is (without the '+')
  - ... more in `gbs help`
- **DO NOT PLACE `-L` and/or `-I` options in GLKB files!**



# export.gbs

- Output-directory specification:  
A line that starts at column 1 specifies a directory relatively to the export directory where file(s) specified in the following Input-files specifications will be copied to.
- Input-files specification  
Lines not starting at column 1 specify the files that have to be copied.  
They are taken relatively to the component-directory.
- Environment Variables of the type `$GBS_BLD_src-type` can be used to specify Build-specific file-types
- Wildcards are not allowed!



# export.gbs

- Example:

```
#=====
# [component] export.gbs
#=====
$GBS_BUILD/inc
  inc/country.h
$GBS_BUILD/lib
  bld/$GBS_BUILD/gps$GBS_BLD_GLB
  bld/$GBS_BUILD/foo$GBS_BLD_C

###EOF###
```



# Program

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping & Building
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals
- Final Remarks & Questions



# Final Remarks & Questions

---

- GBS is built for speed
- GBS is built to help you
  - Throughout consistency
  - Reliability
- Do not write your own scripts
- If you have a good idea:
  - Tell me!
  - If it fits in the generic concept I will add it to GBS
- Use GBS as intended
  - Do not try to be ‘smart’
    - ‘Clever’ is even worse!





# Final Remarks & Questions

---

- If you encounter problems:
  - Probably there is already a solution
  - Do not try to ‘fix’ it without proper knowledge
- So:
  1. Read the Help (gbshelp)
  2. Ask your local GBS intermediate (GBS Administrator)
  3. Contact me

Read the Help

Read the Help

Did I mention to Read the Help?



# Final Remarks & Questions

---

- Introduction
- Build Automation Basics
- The Directory Structure
- Diversity
- Scoping
- Beginning with GBS
- The GBS commands
- The GBS environment
- GBS Internals



# Final Remarks & Questions

---

Smart people find complex solutions

Intelligent people find simple solutions

