# MoReUse / SDE2 2.3

*Getting Started with SDE2*

*Version 2.6*



Getting Started with SDE2 Version 2.6
Publication Date: Sep 29, 2006

Abstract

This document contains an introductory description of SDE2 for novice users.

Keywords

MoReUse, DVP, SDE2, Novice Users

References

[MoReUse]MoReUse 3.1 Standards Book

LIPP/2005/055

Dec 22, 2005, Con Holzscherer

[SDE2]SDE2 2.3 User Manual

STA/SDM/SDE2_2.3/0007

Sep 29, 2006, Bhaskar G

Revision History

| | | | | |
|---|---|---|---|---|
| 2002-04-16 | 1.1 | SDE2 1.7 Alpha | Proposed | Shivaraj P |
| 2002-05-02 | 1.2 | SDE2 1.7 Beta | Accepted | Shivaraj P |
| 2003-06-16 | 1.3 | SDE2 1.7 | Approved | Shivaraj P |
| 2004-05-18 | 1.4 | SDE2 2.0 Alpha | Proposed | Shivaraj P |
| 2004-06-18 | 1.5 | SDE2 2.0 Beta | Accepted | Shivaraj P |
| 2004-07-16 | 1.6 | SDE2 2.0 | Approved | Shivaraj P, Roopa M |
| 2005-03-18 | 1.7 | SDE2 2.1 Alpha | Proposed | Shivaraj P |
| 2005-04-08 | 1.8 | SDE2 2.1 Beta | Accepted | Roopa M |
| 2005-05-20 | 1.9 | SDE2 2.1 | Approved | Bhaskar G |
| 2005-09-02 | 2.0 | SDE2 2.1_SP1 | Approved | Bhaskar G |
| 2005-09-30 | 2.1 | SDE2 2.2 Beta | Accepted | Bhaskar G |
| 2005-10-28 | 2.2 | SDE2 2.2 | Approved | Bhaskar G |
| 2006-06-02 | 2.3 | SDE2 2.2_SP1 | Approved | Bhaskar G |
| 2006-07-28 | 2.4 | SDE2 2.3_Alpha | Proposed | Bhaskar G |
| 2006-08-11 | 2.5 | SDE2 2.3_Beta | Accepted | Bhaskar G |
| 2006-09-29 | 2.6 | SDE2 2.3 | Approved | Bhaskar G |

# Chapter 1
# Introduction



## What is covered in this chapter?

This chapter provides the reader with an introduction to the Software Development Environment 2 (SDE2). This chapter contains:

- List of prerequisites
- Overview of SDE2

## 1.1 Prerequisites

This manual assumes you are familiar with:

- C programming language
- Perl programming language
- Makefiles
- gmake

  For more information about **gmake** see:

  **http://www.gnu.org/manual/make/html_mono/make.html**

Please familiarize yourself with the terms and abbreviations in the *Glossary* before reading this manual.

### 1.1.1 Tools and Softwares Required

SDE2 requires the following tools and softwares installed on the system, other than the compiler toolsets

- Perl - The latest version can be downloaded from:

  **http://www.perl.com/download.csp**

- Cygwin (for windows hosts only)- SDE2 delivers cygwin for windows users along with the release. However, the latest version of cygwin can be downloaded from:

  **http://www.cygwin.com/**

- GNU make (gmake) - SDE2 delivers gmake for windows users along with the release. Other users can download the latest version of gmake from:

  **http://directory.fsf.org/GNU/make.html**

- Doxygen and Graphviz - SDE2 users who need to generate Auto Documentation can download Doxygen and Graphviz from:

  **Doxygen - http://www.doxygen.org/ or http://www.stack.nl/~dimitri/doxygen/**

  **Graphviz- http://www.graphviz.org/**

## 1.2 Purpose and Scope

This manual provides you with a concise overview of the SDE2 package, to help you become familiar with the basic principles of SDE2, and start building components with the help of SDE2. As this document describes only the basic SDE2 features, see SDE2 User Manual for more details.

## 1.3 What is SDE2?

SDE2 is an associated product of LIPP's MoReUse Programme ( Refer [MoReUse] ).

SDE2 is a generic, integrated and MoReUse compliant software build environment for the production of reusable software IPs and systems.

The term generic in the context of build implies the support for multiple configurations.
The term integrated in the context of build implies the support for integrating 3rd party tools.
Ther term production means the support for producing binary releases.
The term reusable means the support for reusable software development.

SDE2 makefiles are invoked using gmake. They take care of dependency checking (source and header file dependencies), compilation and linking of software program components for a wide range of target platforms. SDE2 can be run on the following host platforms: PC (Windows NT or Windows XP or Windows 2000) and Linux. SDE2 supports software components that are implemented using C, C++, assembly level language, System C and JAVA.

The main function of SDE2 is to create and deliver re-usable software components, which can be used across different target platforms without any changes. A source component can be built for different platforms, by setting the corresponding set of platform specific environment variables into the host environment. This process of setting different platforms will be explained in detail in later part of this manual.

# Chapter 2
# Package Overview



## What is covered in this chapter?

This chapter provides information about obtaining the archive, installing it and background on some of the variables. The topics include:

- Downloading the archive
- Unpacking the archive
- Setting up the SDE2 environment
- Setting up the SDE2 directory structure
- Descriptions of environment and makefile variables

## 2.1 Installation

The SDE2 installation procedure consists of:

1. Downloading the archive
2. Extracting the archive into a directory
3. Setting up the environment to run SDE2

### 2.1.1 Downloading the archive

Authorized users may download the zipped SDE2 package from the Central Online Distribution Server (CODS) web site at:

**http://pww-dtc.soton.sc.philips.com/CODS/**

If you are authorized and cannot download the file, contact your local CODS administrator or **cto.helpdesk@philips.com**.

A list of all authorized users can be found in that location.

### 2.1.2 Extracting the archive into a directory

#### 2.1.2.1 Windows

Complete the following steps to unpack the archive on a Windows platform.

1. Open the gzipped file using WinZip.
2. Extract the contents into the directory where you want to install the SDE2 software. For example,

   **c:\sde2_23**

**NOTE:** If you intend to use a directory with its name containing space. Then, the SDE2 environment variables like _TMROOT and _TMTGTBUILDROOT needs to be set in DOS 8.3 format.

For eg: **c:/Program Files/SDE2** as **c:/progra~1/sde2**

#### 2.1.2.2 Linux

Complete the following steps to unpack the archive on a Unix platform.

1. Open the gzipped file. For example,

   **$ cd <location of the gz file>**
   **$ gunzip sde2_2_3.tar.gz**

   The file **sde2_2_3.tar** is now in the **<location of the gz file>** directory.

2. Select an SDE2 installation directory and copy the tar'd file to this directory. For example, if the installation directory is **~/sde2**, copy **sde2_2_3.tar** file to the this directory.

   **$ cp sde2_2_3.tar ~/sde2**

3. Change the current directory to the installation directory.

   **$ cd ~/sde2**

4. Run the following commands to extract the files in **sde2_2_3.tar**. For example,

   **$ tar xvf sde2_2_3.tar**

   Now you will see **sde_template/** in your current directory and directories such as **sde**, **comps**, **intfs**, **inc**, etc., in the **sde_template** directory after the untar.

### 2.1.3 Setting up the environment to run SDE2

After installing SDE2, you need to set certain environment variables in order to use SDE2 for building components.These environment variables determine the configuration class, root location information etc.

SDE2 simplifies the above process by providing you with certain predefined batch files and a simple graphical user interface for various configuration classes.The user can set the environment variables using predefined batch files or by using graphical user interface as described below:

#### 2.1.3.1 Using Batch Files

SDE2 provides various configuration specific batch files at the following location:

**<SDE2 installation Directory>\<SDE2 Root Directory>\project\sites\<site name>**.

For example:

**c:\sde2_23\sde_template\project\sites\blrsdm**

Create a new folder under **<SDE2 installation Directory>\<SDE2 Root Directory>\project\sites** with the proper site name and copy all the batch files present in the original site (blrsdm) folder to the newly created site folder. For example,

```
cd  c:\sde2_23\sde_template\sde\sites
mkdir  ehvcmd
cd  ehvcmd
cp c:\sde2_23\sde_template\sde\sites\blrsdm\*.*
```

Initialize all the essential variables, listed in *Table 2-1*, in the respective batch files and run those batch files in order to set the environment to use SDE2, as explained in section *Section 3.2, Setting up the host environment for SDE2* on page 10l

#### 2.1.3.2 Using GUI

SDE2 provides the user interface for setting environment variables at location:

**<SDE2 installation Directory>\<SDE2 Root Directory>\sde\tools**.

For example:

**c:\sde2_23\sde_template\sde\tools**

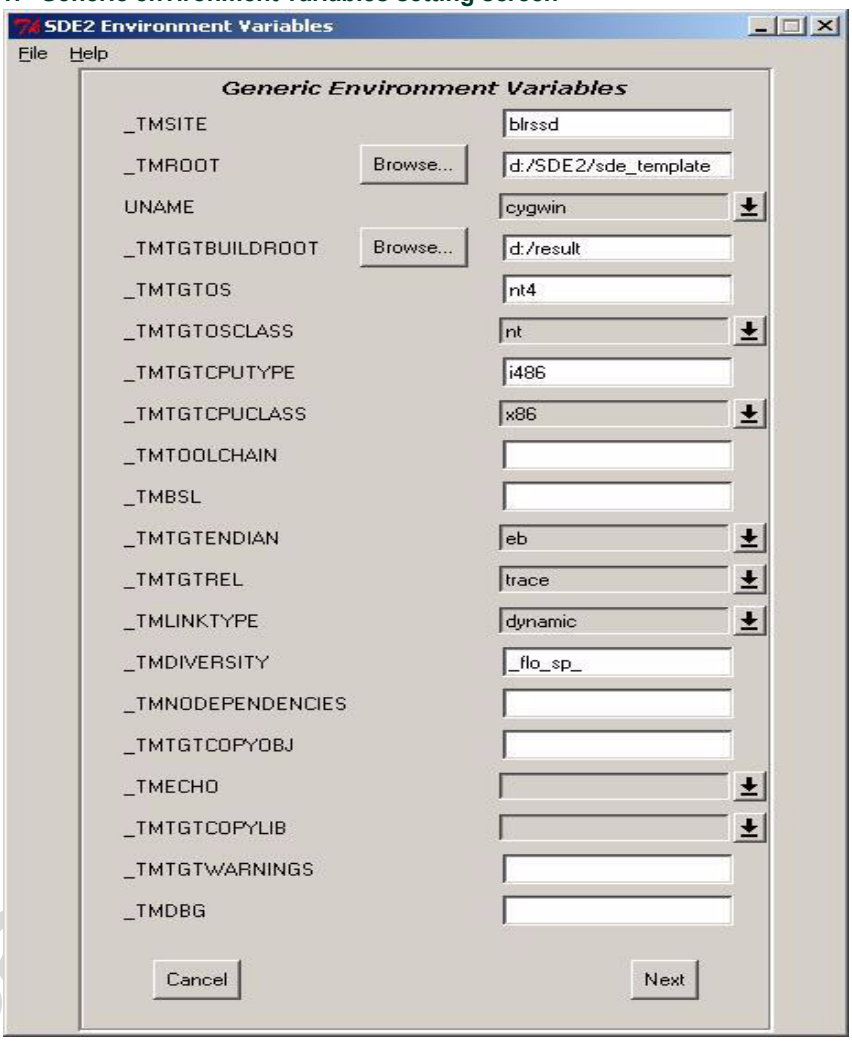
Execute batch file **setenv.bat** present in **<SDE2 installation Directory>\<SDE2 Root Directory>\sde\tools**.

Initialize all the essential variables listed in *Table 2-1*, through the graphical user interface. Environment variables specific to each configuration class are also set by the GUI.

Following are the steps to create a batch file using GUI.

1. Call batch file **setenv.bat** present in the above mentioned location.
2. You will see the graphical user interface as mentioned in *Figure 2-1*

**Figure 2-1: Generic environment variables setting screen**



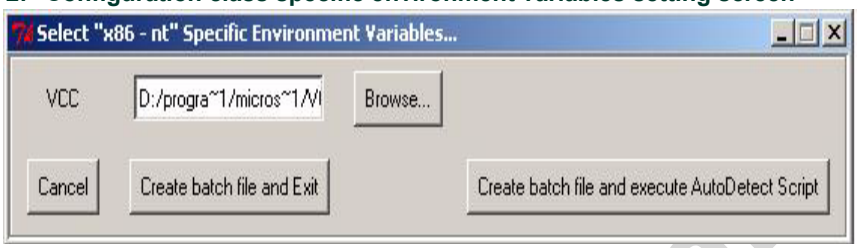

3. Set the values of environment variables. This can be done in two ways:

   a. Set the values by manually entering the value for each environment variable.

   b. Select **File → Open. Choose an existing batch file. The value for each environment variable is populated with that present in the selected batch file. Modify any value of environment variable, if required.**

4. Click on **Next.** It leads you to the dialogbox as shown in *Figure 2-2*. If there is no valid configuration class supported by the current release of SDE2, this dialogbox does not appear. It goes to Batch file is created as follows..

**Figure 2-2: Configuration class specific environment variables setting screen**




5. Set/modify the environment variables specific to configuration class. Configuration class is mentioned in the first screen(*Figure 2-1*).
6. Click on "**Create batch file and Exit**" to finish setting of environment variables for the configuration class selected. Here, only the batch file is created and GUI exits.
7. Batch file is created as follows.

<_TMTGTCPUCLASS><_TMTOOLCHAIN}>_<_TMTGTOSCLASS>_<_TMTGTREL>_<_TMLINKTYPE>_<_TMDIVERSITY>_<_TMTGTENDIAN>.bat

The location of the batch file is under **`<SDE2 installation Directory>\<SDE2 Root Directory>\project\sites\<site name>.`**

8. The user interface also verifies settings of the environment variables, by executing auto-detection (auto_det.pl) perl script. Click on **"Create batch file and execute AutoDetect Script"** as shown in *Figure 2-2* to execute auto_det.pl. Here batch file is created at the location mentioned above and "**auto_det.pl**" is executed for the configuration class selected.

### 2.1.4 SDE2 directory structure

SDE2 and MoReUse implement a flat directory structure that is easily reused, because:

- A flat directory structure is independent of owner, project and architecture.
- All visible components are separately released (or releasable).
- SDE2 can easily find the component interfaces and can thus enforce certain rules.

The SDE2 directory structure contains the following directories:

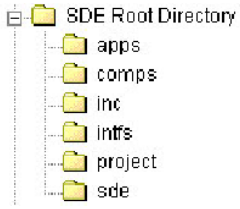



**Figure 2-3: The SDE directory structure**

For more details on SDE2 directory structure, please refer to section 3.1of SDE2 User manual .

**Note:** The directories **comps**, **intfs** and **apps** can be placed in another location, in a multiproject context. See *Section 3.9, Multiproject SDE2* of the SDE2 User manual for more details.

## 2.2 General principle and variables in SDE2

SDE2 is a generic build environment. The general principle behind SDE2 is to provide a environment where you can edit and run configuration-specific batch files, which enables you to build a particular component for different configurations without making changes to the component. For this purpose, SDE2 uses various environment variables to determine the location of the root directory, the name of the site, the type of configuration, etc.

The following table lists environment variables and their respective descriptions.

**Table 2-1: Environment variables and descriptions**

| Environment variable | Description |
|---|---|
| _TMROOT | Location of the SDE2 root directory. |
| _TMSITE | Name of the site. Example: **ehvpid**, **blrsdm** |
| _TMTGTBUILDROOT | SDE2 release directory. |
| _TMTGTCPUTYPE | CPU type. Example: **i486**, **r3940**, **r4300** |
| _TMTGTCPUCLASS | CPU class. Example: **x86**, **arm**, **mips** |
| _TMTGTOS | OS type. Example: **nt4**, **psos250** |
| _TMTGTOSCLASS | OS class. Example: **VxWorks**, **pSOS** |
| _TMTGTREL | Release type of the library. Example: **debug**, **assert**, **retail** |

SDE2 also uses makefile variables. The following table lists makefile variables and their descriptions.

**Table 2-2: Makefile variables and descriptions**

| Makefile variable | Description |
|---|---|
| DIR_LOCAL | Local directory for the component to be built. |
| CXX_SOURCES | Space-separated list of the names of all C++ source files. |
| C_SOURCES | Space-separated list of the names of all C source files. |
| REQUIRES | Space-separated list of the names of the other required components |
| LIBS | Space-separated list of libraries and DLLs required for the component to be built. |
| DIR_INCLUDE | Space-separated list of the include directories. |

The **REQUIRES** section contains the name of the other components whose interface is used by the component to be built and the **LIBS** section contains the names of the other components whose functions have been called by the component to be built. For a detailed explanation on **REQUIRES** and **LIBS**, see *Section 3.4.1, Dependencies* on page 17.

# Chapter 3
# Example Components



## What is covered in this chapter?

The SDE2 package contains example components in its directory structure. These components provide an easy and first-hand method to test various features of SDE2. This chapter:

- Explains the SDE2 directory structure
- Provides examples for building components
- Provides examples for building executables

## 3.1 SDE2 component directory structure

In the SDE2 directory structure, components are found in the **comps** directory. The main component directory name starts with **tm**, followed by an optional layer name andWhat is then by the component name. Component names always start with a capital letter and consist of only letters and digits. Any special characters (non-alphanumeric) are not allowed. For example, **tmComp1**, **tmnlReal** and **tmDbg** are legal component directory names and **Comp1**, **tmcomp1** and **tmDbg_dvp** are illegal component directory names.

Let us take an example component **tmComp1**.

This component can be seen in the SDE2 directory structure as below: .


**Figure 3-1: SDE2 directory structure**

The main component directory **tmComp1** consists of the following subdirectories:

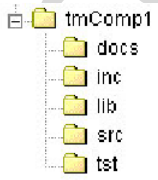



**Figure 3-2: tmComp1 directory structure**

The **docs** directory contains all the documents about the component.

The **inc** directory contains all public header files. In this example, **tmComp1**.h.

The `lib` directory is present if a binary build is to be released. For details on binary releases see

*Section 3.3.3*

The `src` directory contains source files and private header files. In this example, **tmComp1.c**.

The `tst` directory contains example applications to test the component.

For more details on the component directory structure, see Chapter 3 in MoReUse Standards book *[MoReUse]*.

## 3.2 Setting up the host environment for SDE2

For building any component in SDE2 for any specific configuration, the user should initialize a set of environment variables, corresponding to that particular configuration. (For more information see *Section 2.2, General principle and variables in SDE2* on page 8.)

SDE2 provides the user with an easy way to set default values for these environment variables for various configurations. SDE2 comprises several configuration-specific batch files, which contain default values that can be modified for building components for various configurations. Each configuration may consist of more than one batch file for different build flavors (such as release type, link type, etc.). For example, **x86_nt_debug_static_default.bat** and **mips_ce_retail_dynamic_default.bat**

Because the default values provided in these batch files depend on site-specific (local) settings, they are located in the SDE2 directory structure as below:


**Figure 3-3: SDE2 directory structure**

where **<SiteName>** is the name of the location/site, where the user is located; for example **blrsdm**, and **ehvcmd**. SDE2 contains the default **<SiteName> blrsdm**.

Complete the following steps to create site-specific batch files.

1. Create a new directory under **`<SDE2 Root Directory>\project\sites\`** with the local site name.
2. Copy the contents of **`<SDE2 Root Directory>\project\sites\blrsdm\`** into the new site-specific folder.
3. Open any required batch file and modify the values of variables as per the local settings. (See *Section 2.2, General principle and variables in SDE2* on page 8 for more information.)_**TMSITE** should be initialized to the name of the new site-specific directory created.

This is an example batch file **x86_nt_debug_static_default.bat**.

```
# example for building on a pc under a Windows operating system.
#*********************************************
# host platform specific settings
#*********************************************
# Name of the site, user can decide one name for their site
set _TMSITE=blrsdm
# Name of the directory where SDE2 is installed
set _TMROOT=c:/work/sde_template

# Utility for Unix simulation on Windows
set UNAME=cygwin

# Path where you want to put the libraries and other generated files
# from build process
set _TMTGTBUILDROOT=c:/work/result_nt

PATH=%_TMROOT%\sde\cygwin;%PATH%

# To echo messages
# Set it to 1 if you want to see detailed messages from
# SDE2 else leave it blank
set _TMECHO

# VCC stands for Microsoft Visual C++
# Do not use spaces in VCC path, only 8.3 format is allowed
# This variable tells the path of the complier
set VCC=C:/progra~1/micros~3/VC98
#*********************************************
# Target platform specific settings
#*********************************************
# Below are the processor and operating system specific settings
# Name of the OS type
set _TMTGTOS=nt4

# Name of the OS class
set _TMTGTOSCLASS=nt

# Name of the CPU type
set _TMTGTCPUTYPE=i486

# Name of the CPU class
set _TMTGTCPUCLASS=x86

#*************************************************
# flavour settings
#*************************************************
# It tells endianness to be set for each configuration class
#possible values can be eb or el. For x86_nt it is always el.
```

```
#eb stands for big endian and el stands for little endian.

set _TMTGTENDIAN=el
# type of release debug, assert or retail
set _TMTGTREL=debug

# Link type: this value is not currently used but should be set to
# static
set _TMLINKTYPE=static

# optional flavour setting for diversity to be explained later
set _TMDIVERSITY=_flo_mp_
# call a batch file to set few parameters for VC++
call %VCC%\bin\vcvars32.bat
```

4. Open a DOS shell and run the batch file.

Now, the host environment is set to a specific configuration, for which the batch file is run and you can start building components for that configuration using SDE2.

The above mentioned steps can be ignored, if the host environment for SDE2 is set using GUI. Refer *Section 2.1.3.2, Using GUI* on page 5 .The graphical user interface does the above steps for you.

To view the setting, at the DOS command line type:

```
set
```

You can also modify the value of an existing variable by typing:

**set <EnvironmentVariableName>=<NewValue>**

If the host platform is Unix, the value of an existing variable can be modified by typing the following commands:

**export <EnviromentVariableName>=<NewValue>** – Korn shell

**setenv <EnvironmentVariableName>=<NewValue>** – C shell

**<EnvironmentVariableName>=<NewValue>;export var** – Bourne shell

## 3.3 Building components: example 1

This section provides an example for building components. The first example component is **tmComp1**. This component is located in **<SDE2 Root Directory>\comps**. The contents of the component directory **tmComp1** are below:

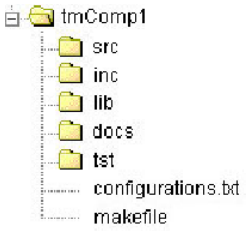



**Figure 3-4: tmComp1 directory structure**

The subdirectories **src**, **inc**, **lib**, **docs** and **tst** are already explained in *Section 3.1, SDE2 component directory structure* on page 9. The component directory **tmComp1** also contains two files: **Configuration.txt** and **makefile**.

The **configuration.txt** file contains information about all possible configurations, for which the component is buildable. It is not necessary to understand the details about this file to use SDE2.

The **makefile** is the entry point to the build component. Details about the **makefile** are described in the following section.

### 3.3.1 Component makefile

Every component must have a makefile in its main directory. The name of this file is **makefile**. This is the entry point for **gmake** to process and build libraries for the component.

The **makefile** contains all the required SDE2-specific definitions and rules to build several targets.

The first line must contain the location of the component.

```
DIR_LOCAL = comps\tmComp1
```

The second line includes a file **environment.mk** which initializes some important makefile variables.

All the source files corresponding to the component are also specified, along with their relative location information.

```
C_SOURCES = src\tmComp1.c
```

The makefile contains a **REQUIRES** section to indicate the interface dependency and a **LIBS** section, to indicate the functional dependency of the component on other components. These sections are explained more in detail, see *Section 3.4.1, Dependencies* on page 17.

The **makefile** specifies any local flags, that need to be used while compiling the component source files (**LOCAL_CFLAGS**, **LOCAL_CXXFLAGS**).

The **makefile** defines a target **all**, which further contains the targets **configuration** and **lib**. The **configuration** target, which is mandatory, checks configurations and sets locations to store the output libraries created for different configurations. The **lib** target builds the library by including a configuration-specific **makelib.mk** file.

The contents of makefile for the component **tmComp1** are listed below:

```
#Component makefile for tmComp1
#relative path of the component directory
DIR_LOCAL = comps/tmComp1

# Do not change the following include. It sets some makefile
# variables It should always be there
include $(_TMROOT)/sde/environment.mk

#-------------------------------------------------------
# Source environment variables
# Relative path of C source files
C_SOURCES=\
src/tmComp1.c
#*******************************************************
# Do not change this
#*******************************************************
# Following are the 2 targets to build The configuration target is
# mandatory; it checks the current configuration. The lib target
# builds the library.
all: configuration lib
#*************************************************************
# Do not change the following include
#*************************************************************
# This .mk file is included to build library specific to the
# configuration
ifneq ($(DIR_CONFIG),_)
include $(DIR_SDE)/$(DIR_CONFIG)/makelib.mk
endif
```

Complete the following steps to build the component under Windows NT:

1. Open a DOS shell, and run the site-specific batch file of the required configuration.
2. Change the present working directory to the component directory. For example,

   `<SDE2 Root Directory>\comps\tmComp1`
3. Type the following command:

   **gmake**

The command **gmake** creates libraries corresponding to **tmComp1**. Libraries are generated in the **<Build directory>**, as specified by the environment variable **_TMTGTBUILDROOT** (library locations are explained in detail in the following section, *Section 3.3.2, Build location for libraries* on page 15).

You can also delete/clean the libraries by running the **gmake clean** command, which deletes the generated libraries. Then you can recreate the libraries by running the **gmake** command again.

### 3.3.2 Build location for libraries

Libraries are located in the **<Build directory>**. The **<Build directory>** is specified, using the environment variable **_TMTGTBUILDROOT**. (Please verify your site-specific batch file for **_TMTGTBUILDROOT** to ensure it is set for the exact build location). If the path specified for this variable does not exist, SDE2 will create this directory.

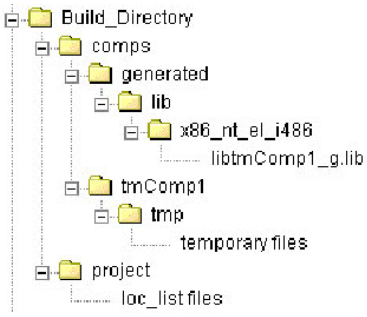



**Figure 3-5: Build Directory structure**

The **<Build_directory>** consists of **comps** and **project** subdirectories.The **project** subdirectory contains the location information of all the components present in the SDE2 **comps** directory.

The **comps** subdirectory further consists of a generated subdirectory and several component subdirectories with the respective component names.

The **generated** directory contains a **lib** subdirectory, which further contains several directories for different configuration classes, endianness and CPU types. For example; **x86_nt_eb_i486** and **tm_psos_el_tm32**. All the generated libraries are stored in these directories, as their respective configurations, CPU type and endianness settings.

The generated libraries follow the naming convention:

**lib<CompName><RELType>.<Extension>**, where:

**<CompName>** – name of the component

**<RELType>** – release type set in the environment

(For debug **<RELType>=_g**, for assert, **<RELType>=_a** and for retail, **<RELType>=<empty>**.)

**<Extension>** – library file extension depending on the configuration class

(For example, for **x86_nt**, **<Extension> = lib** and for **tm_psos**, **<Extension> = a**.)

In the example component of **tmComp1** for **x86_nt** configuration, if the environment variables are set:

**_TMTGTBUILDROOT = c:\build_out**
**_TMTGTCPUTYPE= i486**

**_TMTGTENDIAN = el**
**_TMTGTREL = debug,**

the location of the generated library is:

**C:\build_out\comps\generated\lib\x86_nt_el_i486\libtmComp1_g.lib**

As stated, **<Build_directory>\comps** also contains component subdirectories. Each component directory contains a subdirectory **tmp**, to hold the intermediate files generated while building the components, such as object files, dependency files, option files, etc.

**Note:** If **_TMTGTBUILDROOT** is not defined in your batch file, the libraries are generated in **<SDE2 Root Directory>\comps\generated\lib\x86nt_el_i486**

### 3.3.3 Binary release

SDE2 is capable of producing binary releases for the components. The binary release of a component consists of the libraries built for the component in the component main directory for certain release modes and diversities. If you want to create a binary release, set the environment variable **_TMTGTCOPYLIB** to 1. In this case, the libraries are first generated in the same way as explained above and then the **lib** directory in the generated build tree is copied into the component directory.

In our example component **tmComp1**, the directory structure for a binary release is:

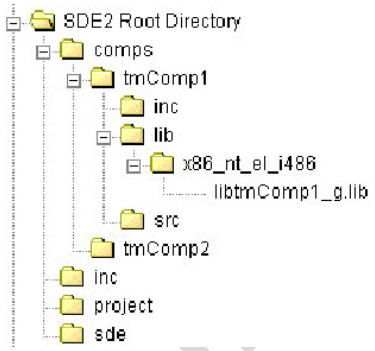



**Figure 3-6: Binary release directory structure**

## 3.4 Building components: example 2

Now build the second example component, **tmComp8**, which has:

- Dependencies on other components
- A component diversity

A component can be dependent on other components in two ways:

1. Interface dependency
2. Functional dependency

### 3.4.1 Dependencies

In case of interface dependency, a component requires the interface (header files) of other components. This situation can be specified in the component makefile by means of a **REQUIRES** section.

For example, in the makefile of **tmComp8**,

**REQUIRES = tmComp1**

means **tmComp8** requires the header files of **tmComp1** (**tmComp1.h**).

In case of functional dependency, the component can call functions, which are implemented as parts of other components. This situation is specified in the component makefile by means of a **LIBS** section. For example, in the makefile of **tmComp8**,

**LIBS = tmComp1**

means **tmComp8** calls a function, which is implemented as part of **tmComp1**.

### 3.4.2 Component diversity

Component diversities are nothing but different flavors for which different libraries are built. A component may behave differently with different diversities. A best reusable component is one which does not posses any diversity. However, practically, these diversities are indispensable.

Diversity flavors are defined by an environment variable **_TMDIVERSITY**. The component makefile reads the value of this variable and builds a separate library for each value of **_TMDIVERSITY**. For example,

**_TMDIVERSITY = _flo_mp_**

The naming convention for the libraries generated with diversities is as follows:

**libtm<CompName><Diversity>_<RELType>.<Extension>**

For example: **libtmComp8_mp_g.lib**

**Note:** **_TMDIVERSITY** can be set in the environment with all valid diversity values, each separated by an underscore (_). A component can generate different libraries for only those diversity values, which are supported by the component. Thus, the generated library name will contain only those diversity values, for which it is generated. For example, if **_TMDIVERSITY** is set to **_flo_mp_**, The library name for **tmComp8** will not contain **_flo** in **<Diversity>** field.

### 3.4.3 Contents of component source file

The contents of the **tmComp8** component source file (**tmComp8.c**) are listed below, to show how this component calls a function implemented in another component. This source file calls the function **tmComp1Print()**, which is implemented in component **tmComp1**. Also note that this file requires the header file (**tmComp1.h**) of **tmComp1**. When you include any interface (header file), it is not required to include it with its absolute path.

```
#include "tmComp8.h"
#include "tmComp1.h"
```

```
void tmComp8Print(int i){
        int j=8+i*10;
        printf("value %d:\n Calling:\n",j);
/*  call to a function implemented in tmComp1  */
        tmComp1Print( 2 );
};
```

### 3.4.4 Contents of component makefile

Now look into the component makefile of **tmComp8**. As we saw before, this component has dependencies and diversities. In order to take care of dependency, you need to add **REQUIRES** and **LIBS** sections in the component makefile as shown below:

```
REQUIRES = tmComp1
LIBS = tmComp1
```

In order to take care of diversity in **tmComp8**, a separate **diversity.mk** file is created and included in the component makefile. This **diversity.mk** file is also located in the component directory. (The contents of **diversity.mk** file are discussed in the following *Section 3.4.5, Contents of diversity.mk file* on page 19).

If a component has diversity, separate libraries are generated for each diversity flavor. This is accomplished by defining a makefile variable **LIB_SUFFIX** in the component makefile.

```
LIB_SUFFIX = $(_tmComp8_DIVERSITY)
```

The value of the variable **_tmComp8_DIVERSITY** is derived from the value of the environment variable **_TMDIVERSITY**. The target **diversity**, defined in **diversity.mk** file, determines the value of **_TMDIVERSITY** (see *Section 3.4.5, Contents of diversity.mk file* on page 19). So, it is important to include this target (**diversity**) with the target **all**, as shown below:

```
all: configuration diversity lib
```

The contents of the component makefile are listed below:

```
#Component makefile for tmComp8
#relative path of the component directory
DIR_LOCAL = comps/tmComp8
# Do not change the following include it sets some makefile
# variables. It should always be there
include $(_TMROOT)/sde/environment.mk
#-------------------------------------------------------
# Source environment variables
#-------------------------------------------------------
C_SOURCES = src/tmComp8.c
#-------------------------------------------------------
# Required components
#-------------------------------------------------------
# name of the component where called function is implemented. In this
# case we have called a function in tmcomp8/src/tmcomp8.c which is
# actually implemented in tmComp1
REQUIRES       = tmComp1
LIBS           = tmComp1
```

```
# This is to generate DLLs
EXPORTS        = tmComp8Print
#check diversity, define _<component>_DIVERSITY
#to check the type of diversities component is sensitive to
#it will include diersity.mk see next section for contents of
#diversity.mk
include diversity.mk
#to generate the library name as per the diversity. This will be
#added to the name of the library.
LIB_SUFFIX = $(_tmComp8_DIVERSITY)
#*************************************************************
# Do not change this
#*************************************************************
# Following are the 3 targets to build. The configuration target is
# mandatory; it sets the configuration. The lib target builds
# the library. Diversity target is to check if _TMDIVERSITY is set
# with proper values or not
all: configuration diversity lib
# Include this file is to build library specific to the configuration
ifneq ($(DIR_CONFIG),_)
include $(DIR_SDE)/$(DIR_CONFIG)/makelib.mk
endif
```

### 3.4.5 Contents of diversity.mk file

We have seen that if a component has diversities, a separate library is built for each diversity flavor.

The **diversity.mk** file, located in the component directory, reads the diversity settings from the environment variable **_TMDIVERSITY**, and passes in the component makefile. The diversity setting is read to a variable **_<CompName>_DIVERSITY** in **diversity.mk** file and this value is used to initialize **LIB_SUFFIX** in the component makefile.

Consider the example component **tmComp8**. This component has two diversity flavors: single processor and multiprocessor (**_sp_** and **_mp_**). The **diversity.mk** file, located in **<SDE2 Root Directory>\comps\tmComp8**, checks the value of **_TMDIVERSITY** for **_mp_** or **_sp_** and assigns the value read to a variable, **_tmComp8_DIVERSITY**. If the variable **_TMDIVERSITY** is not defined or defined with a value other than either **_sp_** or **_mp_**, it displays an error, stating that **_TMDIVERSITY should contain either _mp_ or _sp_**.

The contents of **diversity.mk** file, for the component **tmComp8**, are listed below:

```
# to check diversity if defined by _TMDIVERSITY variable
# and if yes whether it is _mp or _sp
ifeq ($(findstring _mp_,$(_TMDIVERSITY)),_mp_)
_tmComp8_DIVERSITY=_mp
else
ifeq ($(findstring _sp_,$(_TMDIVERSITY)),_sp_)
_tmComp8_DIVERSITY=_sp
endif
endif
```

```
# target diversity
#If _TMDIVERSITY is not defined it flashes the appropriate message
diversity::
ifeq   (,$(filter _mp_ _sp_, $(findstring _mp_, $(_TMDIVERSITY))$(findstring _sp_, $(_TMDIVERSITY))))
            @$(ECHO) "_TMDIVERSITY ($(_TMDIVERSITY)) must contain one of _mp_ or _sp_"
            @exit 1
endif  (,$(filter _mp_ _sp_, $(findstring _mp_, $(_TMDIVERSITY))$(findstring _sp_, $(_TMDIVERSITY))))
```

## 3.5 Building executables/example applications

Now we are familiar with building reusable components. In a real-life scenario, we need to build applications by making use of several components. All the required components for building an application are available as libraries. An application should also have access to the component interfaces.

After you build a component, it is important to test it with some example applications. SDE2 provides you with some example applications along with the example components. These applications are available in the **tst** directory of main component directory. The directory **tst** consists of several subdirectories, each one for implementing different test cases (**Tst1, Tst2 etc.**). Each of these subdirectories is further comprised of **docs**, **src** directories and a **makefile**. The **docs** directory will have documents related to the application and the **src** directory will contain the application source files.

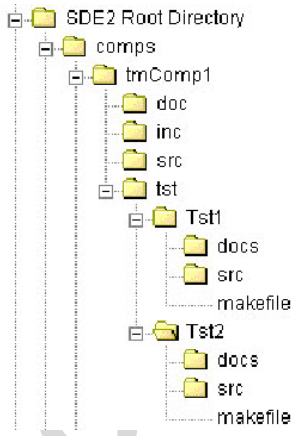



**Figure 3-7: Locations of test directories**

Now, let us build an example application to test components. Let us take the example of component **tmComp8**. SDE2 provides you with a test application in **<SDE2 Root Directory>\comps\tmComp8\tst\Tst1.** The directory **Tst1** contains a **src** directory, which has an application source file **test.c**. The following section discusses the contents of this source file (**test.c**).

### 3.5.1 Contents of test application source file (test.c)

An example application source file for the component **tmComp8** is located in the directory **<SDE2 Root Directory>\comps\tmComp8\tst\Tst1\src**. This file calls a function, which is implemented as part of **tmComp8**. Also, the interface of **tmComp8** (**tmComp8.h**) is included in the source file.

The contents of this source file are listed below.

```
#include "tmComp8.h"
#include <stdio.h>
static void MyMain();
/* specific section for psos and VxWorks */
#if TMFL_OS_IS_PSOS || TMFL_OS_IS_VXWORKS
extern void GtmTick()
{
}

void root(void)
{
        MyMain();
}
#else
int main (void)
{
        MyMain();
        return 0;
}
#endif

void MyMain()
{
        printf("This is test executable for tmComp8a component.\n");
        printf("It prints ");
/* function call whose implementation is in tmcomp8 library */
        tmComp8Print( 4 );
}
```

### 3.5.2 Contents of application makefile

Applications are built in the same way as components are built. Hence, each application will have its own makefile. In our example of **tmComp8,** the makefile is located in the directory**<SDE2 Root Directory>\comps\tmComp8\tst\Tst1**.

Any application, which makes use of a particular component, should have access to the component interface. This is accomplished by having a **REQUIRES** section in the application makefile – **REQUIRES = tmComp8**

Any application, which makes use of a particular component, might also call functions, which are implemented as part of the component itself. This is accomplished by having **LIBS** section in the application makefile – **LIBS = tmComp8**

Application makefile is also comprised of a variable **TARGET**, to define the name of the executable to be built. – **TARGET = test**

(In this case, the name of the built executable is **test.<extension>**, where **<extension>** depends on the configuration class, for example, **test.exe**, **test.hex**, **test.out**, etc.)

The application makefile also defines a target **all**, which further consists of targets **configuration** and **target**. The target **configuration**, which is mandatory, checks configurations and sets locations to store the output executables created for different configurations set in the environment. The target **target** builds an application/executable by including the configuration-specific **maketarget_$(_TMBSL).mk** file. The environment variable **_TMBSL** specifies various board support packages (if applicable), for which applications need to be built.

The contents of an application makefile are listed below:

```
#Application makefile

DIR_LOCAL = comps/tmComp8/tst/Tst1

# Do not change the following include it sets some makefile
# variables. It should always be there
include $(_TMROOT)/sde/environment.mk
#-------------------------------------------------------
# Source environment variables
#-------------------------------------------------------
# relative path of the source files
C_SOURCES      = src/test.c
#-------------------------------------------------------
# Required components
#-------------------------------------------------------
#if it is calling function from tmComp8 library or interfaces is used
#whose implementation is in tmComp1 . Here it is calling the function
#so name of the component is mentioned here
REQUIRES    = tmComp8
# it is calling functions from tmComp8 library . The name of the
# components is mentioned here in LIBS section

LIBS        = tmComp8
#----------------------------------------
# name of the target to build
#----------------------------------------------
# We are building a target executable here test will be name of
# the file generated.
TARGET = test
#*********************************************************
# Do not change this
#*********************************************************
# Following are the 2 targets to build The configuration target is
# mandatory; it sets the
# configuration.  target  builds the executable.
```

```
all: configuration target

#************************************************************
# Do not change the following include
#************************************************************
ifneq ($(DIR_CONFIG),_)
# includes the .mk file to generate excutable
include $(DIR_SDE)/$(DIR_CONFIG)/maketarget$(_TMBSL).mk
endif
```

Complete the following steps to build the component:

1. Open a DOS/Unix shell, and run the site-specific batch file of the required configuration.
2. Change the present working directory to the component directory. For example,

   **<SDE2 Root Directory>\comps\tmComp8**
3. Type the following command to build component **tmComp8**:

   **gmake**
4. Change the present working directory to the example application directory. For example,

   **<SDE2 Root Directory>\comps\tmComp8\tst\tst1**
5. Type the following command to build the example application:

   **gmake**

   The command **gmake** creates an executable corresponding to **tmComp8**. Executables are generated in the <**Build directory**> as explained in the following section, *Section 3.5.3*.

### 3.5.3 Build location for executables

Executables are also located in the <**Build directory**>, as specified by the environment variable **_TMTGTBUILDROOT**. The contents of <**Build directory**> are already explained in *Section 3.3.2, Build location for libraries* on page 15.

In <**Build directory**>, the executables are stored in the component-specific directories. Component-specific directories are located in **<Build_directory>\comps**. Each component-specific directory contains a **tst/Tst1** subdirectory. The directory **Tst1** comprises **bin** and **tmp** subdirectories. While all the temporary files are stored in the **tmp** directory, the **bin** directory contains separate subdirectories named by the configuration, link type, endianness, CPU type and diversity, to hold the corresponding executable.

In the above example of **tmComp8**, the executable is located as shown below:

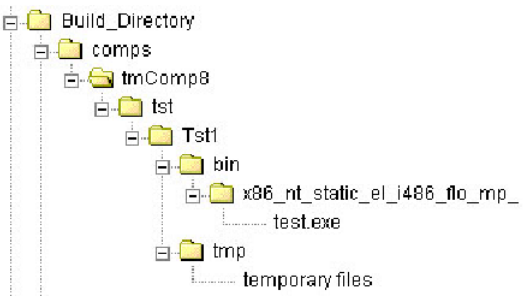



**Figure 3-8: tmComp8 directory structure**

## 3.6 Building components for different configurations

With SDE2, you can create and deliver reusable software components, which can be used across different target platforms, without any changes. In order to make components usable across different platforms, you should be able to build them, accordingly for different platforms. SDE2 provides an easy means to build a single component for various configurations, by setting and running different configuration-specific batch files.

This explains how build the same component for different platforms with the help of an example. So far, we built some example components (**tmComp1**, **tmComp8**) for a specific configuration (**x86_nt**). Now let us see how to build the same components for a different configuration (**tm_psos**).

Complete the following steps to build components for a different configuration:

1. Edit the site-specific batch file of correct configuration for setting the environment variables correctly. For example,

   **tm_psos_retail_static_el_tm32_winnt_default.bat**

2. Open a DOS shell and run the above batch file.
3. Change the current working directory to the component directory (**tmComp1** or **tmComp8**).
4. Type the following command to build components for **tm_psos** configuration:

   **gmake**

   To build **tmComp8**, remember to set **_TMDIVERSITY = _mp_ or _sp_**.
5. Change the directory to the executable directory (**tmComp8\tst\Tst1**).
6. Type the following command to build the executable for **tm_psos** platform:

   **gmake**

The libraries and executables built for different configurations are stored in different configuration-specific subdirectories. The **<Build directory>** structure, with components built for different platforms is shown below:

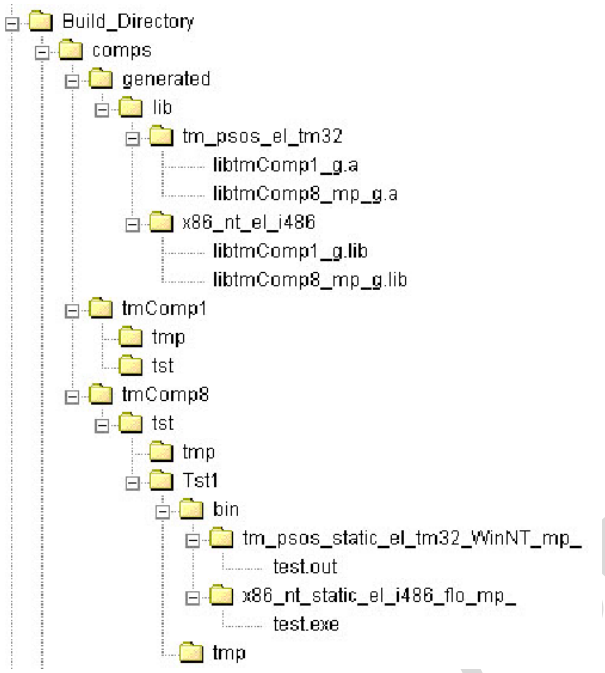



**Figure 3-9: Build Directory structure**

# Appendix A
# Glossary





**Table A-1: Frequently used terms and abbreviations**

| Term | Description |
|---|---|
| Binary release | It consists of two parts. The lib directory contains the libraries/DLLs of the component<br>and documents & interfaces in their respective directories as per MoReUse<br>standards. |
| CODS | Central Online Distribution Server |
| Component | A coherent and encapsulated piece of software with well-defined interfaces, that was designed with reuse in mind so it may be deployed independently. |
| Configuration | Set of options determining which compiler settings are used for building products (libraries/executables) from source code. |
| Configuration class | The combination of CPUClass and operating system, used for building source files. Each configuration is part of a configuration class. |
| CPU | Central Processing Unit |
| CPUClass | CPU Family. For example **x86**, **arm** |
| CPUType | A particular variant of CPUClass. For example **i486**, **r3940** |
| Cygwin | UNIX emulation for PC |
| DOS | Disk Operating System |
| DVP | Digital Video Platform |
| Generic | Describes SDE2's capability of being built for multiple configurations |
| GUI | Graphical User Interface |
| Integrated | Used to describe SDE2's capability of including third-party tools at build time |
| MoReUse | A methodology for reusing software cores |
| OS | Operating System |
| OSClass | Operating systems family. For example **nt**, **pSOS** |
| OSType | A particular version of OSClass. For example **nt4**, **psos250** |
| Production | Building a binary release using SDE2 |
| Reusable | Refers to SDE2's capability of building reusable software |

**Table A-1: Frequently used terms and abbreviations**

| Term | Description |
|---|---|
| LIPP | Library and Intellectual Property Partnership |
| SDE2 | Software Development Environment 2; build environment consisting of a directory structure, makefiles and tools |
| ToolChain | A specific set of build tools (compiler/linker) for a specific Configuration Class For example `ms, tcs` |