KEDACOM 苏州科达科技有限公司
Suzhou Keda Technology Co.,Ltd.

科达工程创建规范

（仅供内部使用）

文 件 编 号：	KDC-
版 本 号：	V 1.0
实 施 日 期：	2007-08-13
保 密 等 级：	■秘密 □机密 □绝密
编 制：	李洪强
审 核：	晋兆龙
会 签：	胡辉、胡小鹏、沈伟平、张明义、陆雪忠、赖齐、万春雷、耿昌明、胡昌威
批 准：	王超

修订记录

日期	版本号	描述	作者
2007-06-13	0.1	根据以前的《KDV 工程约定》等相关资料整理	李洪强
2007-06-27	0.2	根据评审意见，修改工程目录名等章节，增加代码提交章节。	李洪强
2007-07-04	0.3	更新 Linux 和 CCS 部分的内容	李洪强
2007-07-19	0.4	根据评审意见，增加遵从等级的区分，增加 CBBID 号打印	李洪强
2007-08-07	1.0	正式发布 1.0 版本	李洪强

目 录

1 目的

科达目前在开发的产品线逐渐增多，涉及的开发平台及功能模块也非常繁杂。为了规范整个开发团队的开发行为，提高产品质量，有必要对开发工程的创建行为进行规范。同时本规范也可以作为质量检查及相关培训的标准使用。

2 适用范围

本规范适用于科达研发中心所有开发人员。开发人员在创建工程、编码时需要参照本规范的相关规定。QA 人员可以参照本规范制定检查标准，对开发人员的工作进行必要的检查。对于确有特殊情况不能按本规范实施的，需得到相关项目主管批准同意。

3 遵从等级

制定规范的目的是帮助开发人员更好地创建工程，而不是束缚一切的枷锁。划分遵从等级可以让规范更好地发挥作用。本规范主要划分两个级别：

A、 必须遵从。规范中打上对号（“√”）的条目是必须要遵守的内容，不能随意地发挥创建性。

B、 建议遵从。规范中打三角号（“△”）的条目是建议遵守的内容，如果没有更好的理由，就不要违反这些条目。

4 Windows 平台

4.1 工程创建

在 windows 下创建工程时一般采用 VC 的向导的缺省设置一步步创建工程。如有特殊需要，可以对向导的设置进行修改。VC 的工程类型比较多，我们仅对常用的基本于 MFC AppWizard（exe）类型的工程进行约定。以下列出是要注意的一些地方，没有说明的请采用缺省设置。

- √在 General 标签页里面要选择“Use MFC in a Shared DLL”，intermediate files 和 Output files 都选择 debug 或 release。
- △C/C++标签页中 General 分类里，要去掉 Generate browse info 选项。勾上该选项会大大增加编译的速度。该选项缺省是不打勾的，但后续的有些操作会导致此选项选中。所以在提交之前要确保该选项没有选中。
- √C/C++标签页中 Code Generation 分类里，“Use run-time library”选择 Debug Mutithreaded

DLL 或 Mutithreaded DLL。

- √C/C++标签页中 Preprocessor 分类中的“Additional include directories”和 Link 标签页中 Input 分类中的“Additional include directories”，必须采用相对目录。
- △Link 标签页中 General 分类中 Output file name 选项有两种做法。一种是输出在本地目录的 debug 或 release 目录下，另外一种是通过相对路径的方式输出到指定的目录下。对于输出到本地的情况，要在编译完成后用将编译出的程序拷贝到指定的目录，如 10-common 的 version 目录下。拷贝可以用 compile.bat 完成，也可以在 Post-build step 选项中用命令完成。拷贝时也要注意采用相对路径。直接输出指定的目录优点在只会有一个输出文件，不会造成调试时用错文件的情况发生。但是这样缺点是会在输出目录产生少量的垃圾文件，如.ilk，.lib，.exp 文件。这些文件拷到版本机上只会是垃圾，因此要 Post-build step 或 compile.bat 里面将他们删除。

4.2 包含 Osp 的 Windows 应用程序头文件规定

为了避免使用 Osp 的 Windows 应用程序在编译的时候产生头文件的冲突，特规定以下要点：

- √在 Osp 中包含 afx.h，afxwin.h 和 winsock2.h 三个头文件。
- √Windows 应用程序不包含 windows.h 头文件。
- √应用程序使用共享的 MFC DLL。

4.3 静态链接库工程标准

视频通信产品功能复杂，开发人员多，为了并行开发，整个系统采用模块化的思想，将产品划分为不同开发人员负责的功能库。每个功能库都提供完善的接口供使用者调用，相应开发人员负责库的完备性。功能库以静态或者动态链接库的形式提供。对于仅仅用于 Windows 平台或者跨越 Windows、VxWorks、Linux 平台的功能库，特别规定统一输出 C++类型的库。跨越 Equator 和其他平台的静态链接库可以输出 C 类型库。

使用静态链接库，可以减轻发布新版本的压力。静态链接库直接包含已经编译的可执行代码，所以在链接的时候很容易产生链接错误，因此在开发静态链接库的时候，有必要为了保证最终产品的成功编译链接，制定相应的静态链接库的工程标准。

- √使用 VC 的工程向导，创建工程选择 Win32 Static Library。对于仅仅用于 Win 平台的静态链接库，为了使用 MFC 的特性，统一选择使用预编译头（Pre-Compiled Header）和 MFC 支持（MFC Support）选项。跨平台的统一不选择这些选项。通过该工程向导，创建的是

一个缺省选项的静态库工程，开发者向工程中添加自己的代码文件即可。

- √在 General 标签页内，对于仅仅用于 Windows 平台的静态链接库，确定统一选择“Use MFC in a Shared DLL”，其他类型的不使用 MFC。
- √在 C/C++标签页内，确定选择使用多线程 DLL（Multithread DLL）。

4.4 静态链接库工程标准

√动态链接库的工程要求同静态链接库的要求。

4.5 版本信息

在 windows 下创建基于对话框或文档视图等应用程序时，要资源文件中填写相关的版本信息。在工程的 resource 标签页里点击 Version 下面的 VS_VERSION_INFO 会出现以下配置界面：

在上面的界面上需要手动地填写产品信息：

- △PRODUCTVERSION：参照本文后段有关版本的描述
- √CompanyName：苏州科达科技有限公司
- √FileDescription：相关模块的中文描述，如科达会议控制台
- √FileVersion：参照本文后段中有关版本的描述
- △InternalName：相关模块的中文描述，如科达会议控制台
- √LegalCopyright：2003-2007 Suzhou Keda Technology Co.,Ltd。2003 指的是软件最早创建的时间，2007 指当前时间。如果写中文则为：2003-2007 苏州科达科技有限公司

- √ProductName：相关模块中文名称，如科达会议控制台
- √LegalTrademarks：公司产品的商标，KEDACOM

5 VxWorks 平台

5.1 创建

√VxWorks 工程有两种，创建时 ToolChain 分别选用 PPCEC603gnu（8260，8265）和 PENTIUMgnu。

5.2 编译参数

√PENTIUMgnu 工程编译开关统一采用创建时缺省参数，可再加上相应的头文件路径和自定义宏，不得再随便添加编译开关。缺省开关和宏定义如下：

-g -mpentium -ansi -nostdinc -DRW_MULTI_THREAD -D_REENTRANT -fvolatile -nostdlib -fno-builtin -fno-defer-pop -I. -ID:/Tornado/target/h -DCPU=PENTIUM

√PPCEC603gun 工程除了统一采用缺省参数再加上相应的头文件路径和自定义宏以外，还需另外添加字段-mlongcall，除此之外不得再随便添加编译开关。最终缺省开关和宏定义如下：

-g -mstrict-align -ansi -nostdinc -DRW_MULTI_THREAD -D_REENTRANT -fvolatile -fno-builtin -fno-for-scope -msoft-float -mlongcall -I. -ID:/Tornado/target/h -DCPU=PPCEC603

√所有跨平台的.out 或.a 工程必须统一增加宏定义_VXWORKS_，定义方式是在该工程属性 c/c++ compiler 里添加字段-D_VXWORKS_。

5.3 库文件

√.a 工程调用到库文件时时，只需将其它库的头文件 include 进来，而不需将相应的.a 文件加入到该工程中来，所有用到的.a 文件统一由.out 工程加入。

5.4 关于文件只读属性设置

√由于目前的嵌入式系统里，文件会无故损坏，所以要求 vxworks 在启动的时候将所有文件设为只读属性，各模块在写文件完成后，要将文件的属性改变成只读属性。

6 Linux/Unix 平台

Linux 项目工程规范，采用 2004 年公司制订的 Makefile 规范。经过实践证明，该规范能很好满足业务需求，且方便使用，下面以文档形式描述该规范，并以 OSP 为例描述如何使用该规范。

6.1 Makefile 模板

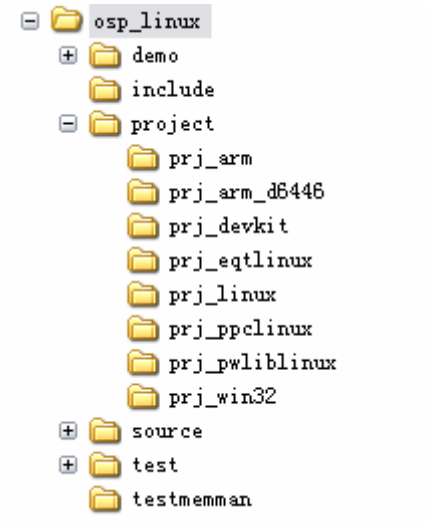
√必须统一使用公司规范 Makefile 模板，common.mk， 

。

- √该工程文件无需使用者编辑修改，可以视为一个黑盒。
- √该工程文件涵盖 ppc8xx、ppc82xx、ppc85xx、eqt、arm、arm 6446、x86 的交叉编译方法。

6.2 目录结构

√编写 Makefile，以 OSP 为例，目录结构如下：

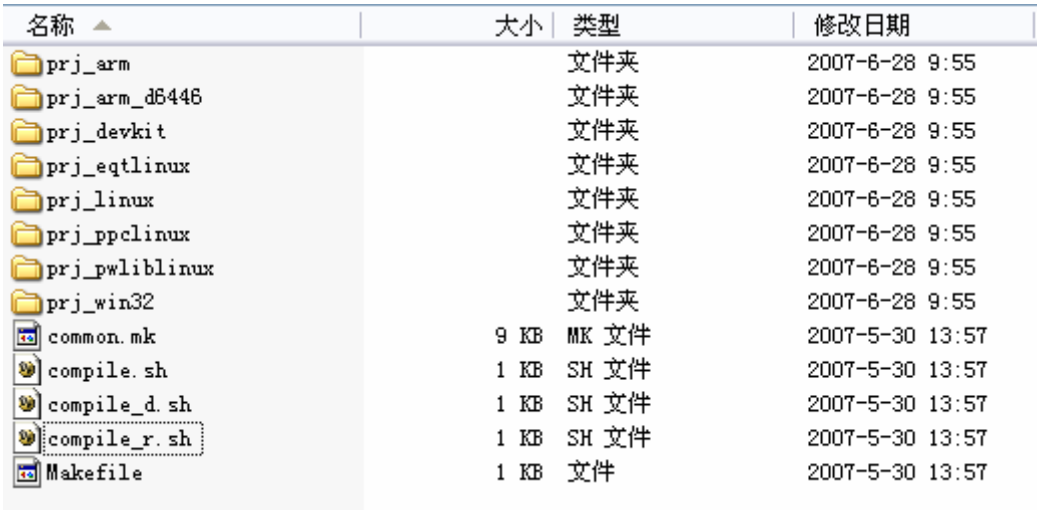


include 为头文件目录。

source 为源代码目录。

project 目录为工程目录。

6.3 project 目录子目录



名称	大小	类型	修改日期
prj_arm		文件夹	2007-6-28 9:55
prj_arm_d6446		文件夹	2007-6-28 9:55
prj_devkit		文件夹	2007-6-28 9:55
prj_eqtlinux		文件夹	2007-6-28 9:55
prj_linux		文件夹	2007-6-28 9:55
prj_ppclinux		文件夹	2007-6-28 9:55
prj_pwliblinux		文件夹	2007-6-28 9:55
prj_win32		文件夹	2007-6-28 9:55
common.mk	9 KB	MK 文件	2007-5-30 13:57
compile.sh	1 KB	SH 文件	2007-5-30 13:57
compile_d.sh	1 KB	SH 文件	2007-5-30 13:57
compile_r.sh	1 KB	SH 文件	2007-5-30 13:57
Makefile	1 KB	文件	2007-5-30 13:57

common.mk 即为 5.1 中所描述的公司 Makefile 模板。

compile.sh 为总体编译脚本，各个模块必须在其 project 目录下提供名为 comile.sh 的编译脚本，完成该模块在各个平台（如 arm、ppc、x86）上的各个版本（如 debug、release）的编译，并保证编译后的目标文件在发布目录（如 10-common/lib）中。

√配置管理员进入各个模块的 project 目录调用 compile.sh，完成每日的版本编译。

compile.sh 的内容如下：

```
#!/bin/sh
make -f Makefile    // 执行当前目录下的 Makefile
```

△开发人员**可以**额外提供 compile_d.sh，compile_r.sh 分别完成 debug、release 版本的编译。

Makefile 的内容如下：

```
#compile for each subdirs

/* 当前需要编译如下子目录，prj_linux、 prj_ppclinux、 prj_arm、 prj_arm_d6446*/
DIRS = prj_linux prj_ppclinux prj_arm prj_arm_d6446

include common.mk    //包含公司 Makefile 模板文件
```

√每个体系结构必须对应一个 prj 子目录，需要编哪些体系结构，则该 Makefile 中的 DIRS 需要包含对应的目录。

6.4 示例

以 ARM 为例说明每个体系结构下的编译脚本如何编写，prj_arm 结构如下：

common.mk 会到指定的子目录（如 prj_arm）下寻找 makefile 并执行，makefile 文件内容如下：

```
MAKE := make --no-print-directory

all :
//  编译 DEBUG
      // make clean，删除编译中间文件；
      @$(MAKE) -f makefile_debug clean;

      // make，并把编译出错信息输出到 comileinfo 下，各个模块输出到到不同的文件；
      @$(MAKE) -f makefile_debug 1> ../../../../10-common/version/compileinfo/osp_linux_arm_cpp_d.txt 2>&1;

      // make clean，删除编译中间文件；
      @$(MAKE) -f makefile_debug clean;

//  编译 RELEASE
      @$(MAKE) -f makefile_release clean;
      @$(MAKE) -f makefile_release 1> ../../../../10-common/version/compileinfo/osp_linux_arm_cpp_r.txt 2>&1;
      @$(MAKE) -f makefile_release clean;
```

makefile 会执行 makefile_debug 和 makefile_release，来完成 debug 和 release 版本的编译，并把编译出错内容输出到 comileinfo 下，各模块根据自己的模块名、体系结构、版本等，决定存放编译信息的文件名。

命名规则为 [模块]_[操作系统]_[体系结构]_[Debug/Release].txt。

6.5 makefile_debug/makefile_release

√makefile_debug/makefile_release 是整个编译规范的核心，也是最复杂的文件，下面详细说明该文件的内容，以 makefile_debug 为例：

```
#############################################################################
###
###   DESCRIPTION:
###       Common definitions for all Makefiles in OSP linux project.
###
```

```
##############################################################################
//模块顶层目录
TOP := ../..

//模块常用目录
COMM_DIR := ../..

//源代码目录
SRC_DIR := $(TOP)/source

## Name and type of the target for this Makefile

//模块名
ARC_TARGET          := Osp

## Define debugging symbols
//Debug 标志打开
DEBUG = 1
//OSP 自定义宏
CFLAGS += -DOSP_DEBUG
//编译器 ARM
LINUX_COMPILER=_ARM_# _EQUATOR_, _ARM_, _LINUX_ and so on
//是否采用 pwlib
PWLIB_SUPPORT = 0

## Object files that compose the target(s)
//需要编译的目标文件
OBJS :=                                 \
      $(SRC_DIR)/ospteleserver          \
      $(SRC_DIR)/osplog                  \
      $(SRC_DIR)/ospnodeman             \
      $(SRC_DIR)/osppost                \
      $(SRC_DIR)/ospsch                  \
      $(SRC_DIR)/osptest                 \
      $(SRC_DIR)/osptestagent            \
      $(SRC_DIR)/osptimer                \
      $(SRC_DIR)/osputil                  \
      $(SRC_DIR)/ospvos

## Libraries to include in shared object file
//需要包含的库
#LIBS :=
```

```
## Add driver-specific include directory to the search path
//需要包含的头文件目录
INC_PATH += $(TOP)/include ../../../../10-Common/include/cbb/platform ../../../../10-Common/include/cbb/system

//pwlib 的头文件、库的路径
ifeq ($(PWLIB_SUPPORT),1)
    INC_PATH += $(PWLIBDIR)/include/ptlib/unix $(PWLIBDIR)/include
endif

//编译目标安装目录
INSTALL_LIB_PATH = ../../../../10-Common/lib/debug/linux_arm_cpp

//包含公司 makefile 模板
include $(COMM_DIR)/common.mk
```

a) TOP **:= ../..**

这句话指名该模块的顶级目录，即 source, include, project 平级那层目录，common.mk 会使用 TOP 变量，因此必须指定。

当前目录为 osp_linux/project/linux_arm，因此 TOP 目录为向上两级 **../..**

b) COMM_DIR **:= ../..**

这句话指名该模块的当前使用目录，可以自己指定，方便后面的目录定位，也可以忽略不用。

c) SRC_DIR := $(TOP)/source

指定源文件路径，即.c 的路径，必须指定。

d) ARC_TARGET := osp

静态库名，生成的静态库会以 lib(ARC_TARGET).a 来命名。

如果需要生成静态库，必须指定 ARC_TARGET。

如果需要生成动态库，必须指定 SO_TARGET。

如果需要生成可执行程序，必须指定 APP_TARGET。

e) DEBUG = 1

DEBUG 标志，DEBUG 版本为 1，RELEASE 版本为 0。

f) LINUX_COMPILER=_ARM_

交叉编译器指定，

ppc82xx ： _HHPPC_

ppc85xx ： _PPC85_

arm ： _ARM_

arm d6446 ： _DAVINCI_

eqt ： _EQUATOR_

x86 ： _LINUX_

g) PWLIB_SUPPORT = 0

是否采用 pwlib，现只有 T120 使用。

h) OBJS := $(SRC_DIR)/ospsch \

需要编译的对象文件，每个对象文件对应一个同名的.c 文件，如 ospsch 对应于 ospsch.c，其位于源文件路径下。

必须指定每个需要编译的.c 文件，并列出其文件名。

i) LIBS :=

LIB_PATH :=

需要包含的库，如果需要链接其他库，则必须指定该这 2 个变量。

如需要链接 OSP，rt，encrypt，pthread，则应该

LIBS := osp rt \

encrypt pthread

LIB_PATH := ../../../../10-common/lib/release/linux

j) INC_PATH += $(TOP)/include ../../../../10-common/include/cbb/platform

../../../../10-common/include/cbb/system

指定头文件相对路径，如果引用公用头文件，其他模块头文件，则必须指定该目录。

k) ifeq ($(PWLIB_SUPPORT),1)

INC_PATH += $(PWLIBDIR)/include/ptlib/unix $(PWLIBDIR)/include

LIB_PATH += $(PWLIBDIR)/lib/

endif

如果支持 pwlib，则必须指定 pwlib 的头文件路径，以及库路径。

l) INSTALL_LIB_PATH = ../../../../10-common/lib/debug/linux_arm_cpp

指定库安装路径，必须指定该目录。

m) include $(COMM_DIR)/common.mk

必须包含公司 makefile 模板。

n) 如果用户需要指定一些额外的宏，可以通过设置 CFLAGS 完成，如

CFLAGS += -DOSP_DEBUG

6.6 注意点

a) 注意相对路径：

调用 comile.sh 的时候，当前路径必须是 project 目录。

编写 prj_linux/makefile_debug 的时候，当前路径是 makefile_debug 文件所在路径，此时头文件、库文件、安装目录的相对路径都必须根据当前路径来计算。

b) 注意区分大小写：

Linux 文件系统区分大小写，因此头文件路径、库文件路径、安装目录必须区分大小写。

c) 注意脚本语言语法

有些变量如 OBJS 换行时必须使用“\”。

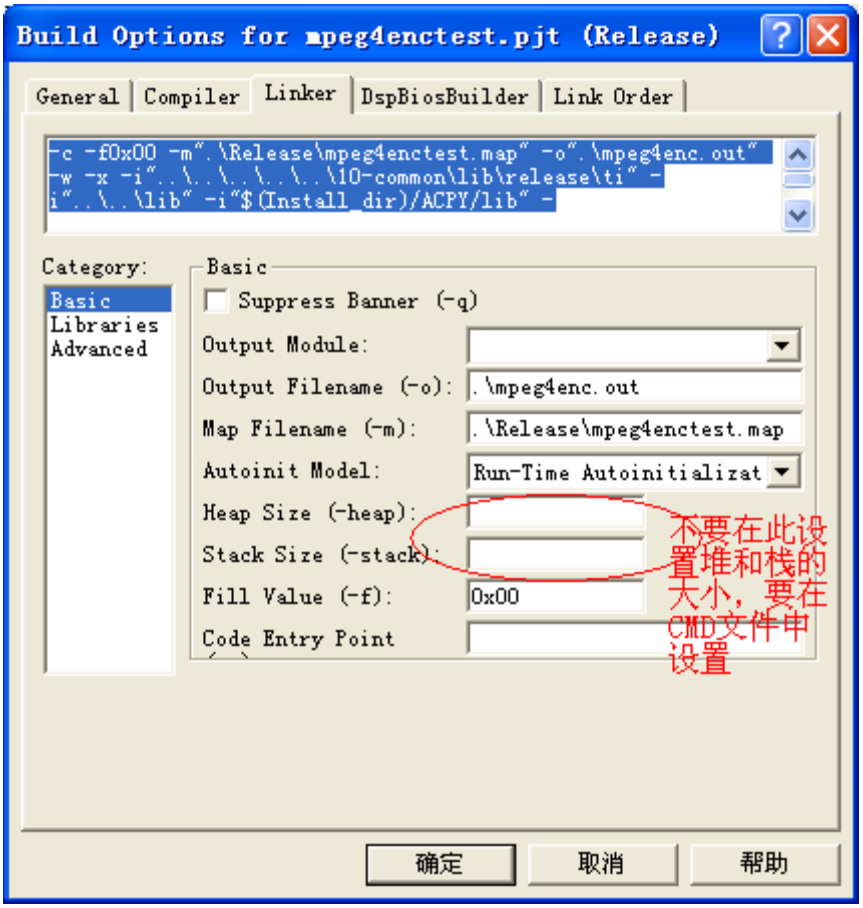
7 CCS 平台

7.1 创建

- √在 CCS 下创建工程时要选择目标 DSP 平台，选择编译目标是可执行程序或者库文件。
- √工程所在目录不允许有中文字符，和不常用字符如“-”等，因为 CCS 对中文和不常用字符不支持。

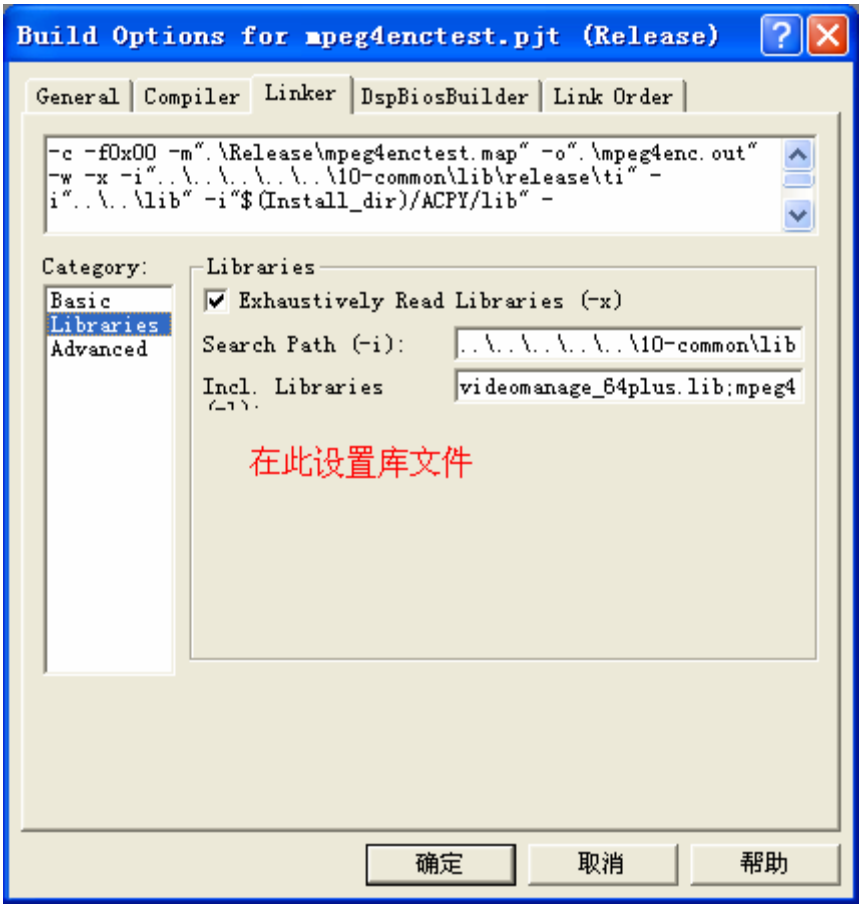
7.2 编译参数

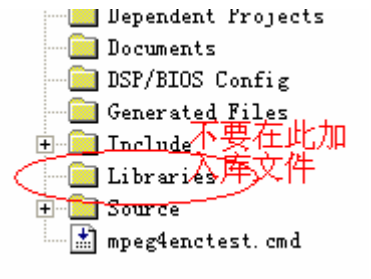
√当选择编译目标是可执行程序时，堆和栈的大小设置必须在 CMD 文件中设置，不要在编译选择对话框中设置。



7.3 库文件

√工程中的库文件要在编译选择对话框中设置

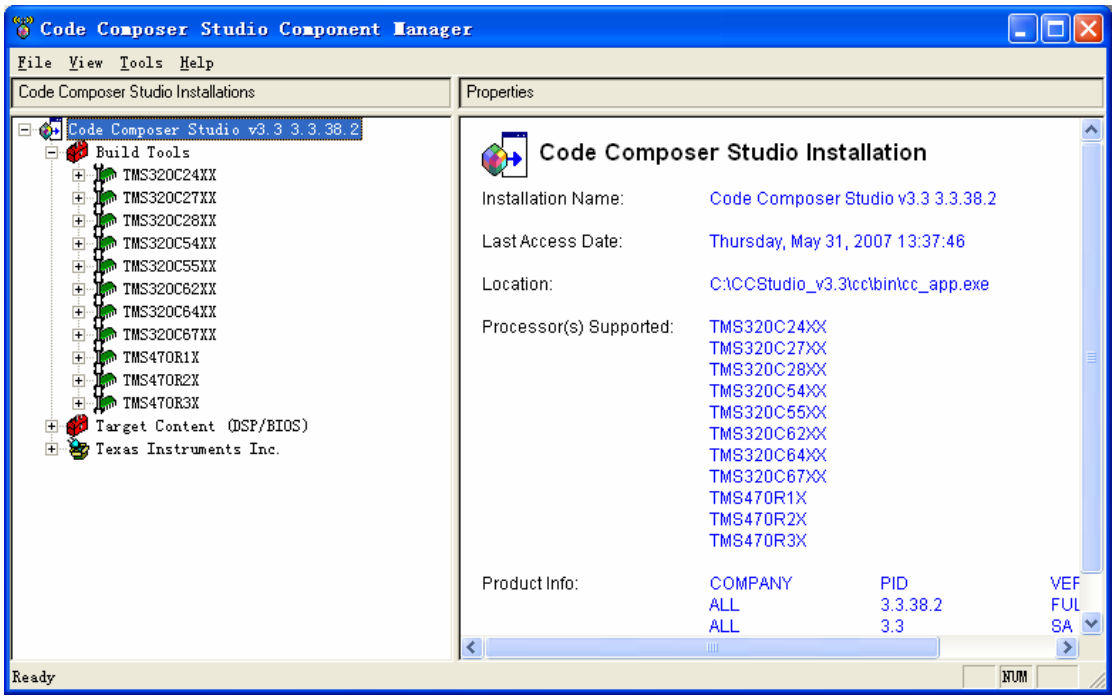




7.4 编译器

√CCS 要注意所用编译器的选择，选择方法如下：

help→about→Component



8 调试手段

√每个模块都需要提供适当的调试手段来帮助使用定位问题，或者查看当前的状态。对于包括 OSP 库的模块最基本的调试手段是提供版本号及帮助命令。必须要能在 telnet 窗口中打印其版本信息和相关的帮助命令。

8.1 版本号命名规则

√模块的版本号命名规则如下：

CBB 的 ID 号（可选）	模块名称	应用名称（应用场合）	系统版本	模块版本	接口版本（改动次数）	实现版本（Bug 修改次数）	修改时间（yyyymmdd）
P0010	Osp	Small	20	12	02	01	20070519

总的结构：mn.an ss.mm.ii.cc.tttttttt，如 P0010 Osp.Small 20.12.02.01.20070519 表示

CBB 的 ID 号：Osp 模块是一个 CBB，在版本号里面应该加入该 CBB 的 ID 号如 P0010。

模块名称 Osp，应用名 Small，系统 2.0，1.2 版本，接口 0.2 版本，实现 0.1 版本，2007 年 5 月 19 号修改。

应用名称：表示是用在哪个场合的版本，如某次测试、某个代码分支或一些特定的应用。

系统版本：当前产品版本为 2.0 版本

模块版本：每一次大的升级加一。

接口版本：每修改一次接口加一。

实现版本：接口没变的情况下每修改一次加 1（修改 BUG，或增加功能）。每次提交代码时要确保实现版本号大于上次提交时的版本号。

注：版本号缺省使用两位表示，累计超过两位按实际位数表示。

8.2 调试命令命名规则

√调试命令统一命名为 modulehelp、modulever。其中 module 为模块名，例如 MCU 提供的命令为 mcuhelp、mcuver，OSP 提供的命令为 osphelp、ospver。ver、help 命令必须全部为小写，其余命令不做限定。

△对于顶层模块，模块名建议与 OspPrompt 提示的名称一致。例如某模块调用了 OspSetPrompt（“abc”），则 telnet 窗口里面的提示就为 abc->。此时模块名取为 abc 就很容易得出相关的命令为 abchelp 和 abcver。相关的底层模块的帮助信息可以列在顶层模块的帮助命令里。这样不需要翻阅手册就可以了解到所有的帮助命令。

√ver 命令必须打印出所调用重要模块的版本信息。

√help 命令必须打印出所调用模块的 help 命令格式及自己的所有帮助命令

△调试命令格式

ver:

XXX Module1 Version: *mn.ss.mm.ii.cc.yyyymmdd.* Compile Time: *time, date,*

XXX Module2 Version: *mn.ss.mm.ii.cc.yyyymmdd.* Compile Time: *time, date,*

help:

module1help（）;

module2help（）;

private command:

Command1:

Parameter: ParameterName1 参数 1 说明

ParameterName2 参数 2 说明

Function: Command1 命令功能说明

Command2:

Parameter: ParameterName1 参数 1 说明

ParameterName2 参数 2 说明

Function: Command2 命令功能说明

斜体表示实际打印的部分，Command 为命令名称，ParameterName 为该命令对应的参数所有输出必须为英文。

√所有打印信息统一用函数 OspPrintf（ TRUE, FALSE, “.....” ）。

△模块内打印输出信息时必须加上模块名，并换行。避免打印过多时，分不清是谁在打印的情况出现。如 mtclib 的打印格式应该为：

OspPrintf（TRUE, FALSE, “[mtclib]Connect mt time out\n”）。

调试打印命令要遵循简单明了的原则，要方便调试者使用，过于冗长的命令会增加敲错命令的机会。

9 编译脚本

编译脚本是自动化编译的根本，发布产品中的所有二进制文件输出必须是在版本机上通过编译脚本编译产生的。编译脚本提交前要检查确保相关模块能够正常输出。

9.1 功能要求

√各产品线的配置管理的目录结构可能会有所不同，但一级目录、二级目录及各子模块的主目录下面必须有一个编译脚本。通过运行脚本，该脚本所在目录及其下所有子目录的模块均要能够编译输出。一个模块如果有多个体系结构的工程目录，则每个目录下面要放置与之对应的编译脚本。上下层脚本调用时要避免循环嵌套现象的出现。

√编译脚本要能同时输出 DEBUG 及 RELEASE 两个版本代码。如果确实没有必要输出 DEBUG 版本，需要经过产品经理认可。

√编译脚本在编译正式代码之前要检查编译模块所依赖的库或文件是否存在。如果不存在则直接退出编译。如果依赖的库或文件出了问题，编了也是白编，浪费时间。

9.2 脚本命名

√编译脚本文件在不同的系统下稍有区别，现规定如下：

Windows：

compile.bat	编译所有版本
compile_d.bat	只编译 DEBUG 版本
compile_r.bat	只编译 RELEASE 版本

Linux/Unix/VxWorks：

compile 或 compile.sh	编译所有版本
compile_d 或 compile_d.sh	只编译 DEBUG 版本
compile_r 或 compile_r.sh	只编译 RELEASE 版本

10 目录结构

各产品线及部门内项目繁多，要在现阶段完全统一所有目录结构还有一定困难。本规范针对一

些底层的目录结构做出规定。

10.1 高层目录结构

在一个确定的配置管理环境中，高层目录（如一、二级目录）结构由产品线或部门根据需要进行划分。划分的原则为：

- △所有的输出放在一起，各模块依据系统架构及功能划分，而不是以小组人员的组合方式划分。
- △层次结构要简明清晰，避免一级目录下面有过多的目录。
- △有联系的模块尽可能地放在一起。

10.2 模块目录名

需要目录名要能清晰地区分出模块的用途，规则为“模块名+功能”，在模块名清晰、唯一，不会产生歧义时“功能”为可选：

- △模块名+ui：存放某模块图形界面的代码。
- △模块名+lib：存放某模块库文件。
- △模块名+server：存放某模块服务器端代码。（server 为可选项）
- √所有需要提交的目录和文件名一律用小写，避免由于配置管理工具大小写敏感带来的一些麻烦。对于编译器或系统自动生成的目录或文件，不强行要求小写。

10.3 模块源码部分的目录结构

△具体到某一个模块，其目录结构都是相似的，主要包含以下几个目录：

- doc（可选）：存放与该模块相关的设计文档
- include：存放该模块的头文件。
- source：存放该模块的源代码。
- project：存放工程文件。
- unittest（可选）：该模块的单元测试模块的相关文件
- systest（可选）：该模块系统测试或者集成测试模块的相关文件
- demo（可选）：该模块的演示模块的相关文件
- install（可选）：该模块安装盘的相关文件。

project 目录下面是模块的工程目录，对于运行于多个平台的模块，需要在 project 目录下面再细

分对应的工程目录，将工程文件，Makefile 及编译脚本等放在下面。细分的目录命名规则为“prj_操作系统_体系结构”。如不区分体系结构，后面的体系结构部分可以不写。以下是一些已有的工程目录名的示例：

- prj_win32：Win32 系统
- prj_linux：桌面 Linux
- prj_linux_eqt：Eqate Linux 系统
- prj_linux_ppc_82xx：
- prj_linux_arm：
- prj_vx_8260：
- prj_vx_860：
- prj_solaris：Solaris 系统

一个典型的模块目录结构为：

```
module
    |-include
    |-project
    |   |-prj_linux
    |   |-prj_win32
    |-source
```

对于明确只会运行在单一操作系统下的模块，project 可以为对应的细分目录名。如只会跑在 windows 上的程序，可以直接在主目录下面建立 prj_win32 目录，而不必建立如 project\prj_win32 这样的两级目录。上面的例子就可以简化为：

```
module
    |-include
    |-prj_win32
    |-source
```

10.4 编译输出的目录结构

√整个产品的编译结果要集中输出到公共的目录下面，便于取用。输出的内容包括两大部分，中间编译的 lib 库和最终编译的二进制版本。这两部分都需要编译 DEBUG 和 RELEASE 两个版本。

对于跑在单板或嵌入式设备上的，无法 debug 的底层库或模块，可以将 release 版的文件拷贝一份到相应的 DEBUG 目录下。

√输出目录按照先平台、后模块的层次进行划分，目录命名规则与工程细分目录类似，也是“操作系统_体系结构”，比如：

- win32：Win32 系统编译
- linux：桌面 Linux 编译
- linux_eqt：Eqate Linux 系统编译
- linux_ppc_82xx：
- linux_arm：
- vx_8260：
- vx_860：
- solaris：Solaris 系统编译

除了以上的模块目录以外，还有一些辅助目录同样需要提交到版本机上：

- √compileinfo：编译信息的输出
- √config：配置文件或数据库脚本。

11 Readme 文件

每个模块都要有自己的 readme 文件，用于记录模块的功能修改及 BUG 解决情况。是模块维护的重要历史记录。

√Readme 文件统一文件名为 readme.txt，放在本模块的最外层目录下，与 compile 文件放在一起。在版本机编译时，由编译脚本统一拷贝到指定目录。

√代码每提交一次，就要在 readme 文件里增加修改的内容。修改的内容至少要包含以下信息：模块名称、提交的版本号、新增了哪些功能、修改的 BUG 编号或简单描述、当前模块的维护人。提交者也可以备注一些必要的说明信息。

△有些项目组可能会要求对 readme 文件进行自动检查。在这种情况下，readme 的格式将有严格的要求。以下是视频会议产品的一份 readme 格式，仅供参考：

```
==========================================================================
3AServer 主控模块
==========================================================================
```

版本号：3as39.10.01.57.061222

一、接口改变，新增功能

二、修改的 bug（对应 bug 单中编号）

1 usbkey 的操作上 win 平台和 linux 平台不同

2 delete 语句有错误，没有 from 关键词

3 更形 my 的脚本

三、文档更新情况

已更新

四、代码走读情况

检查人: 万春雷

五、单元测试情况

六、系统测试建议

基本功能测试

七、模块负责人

万春雷

12 库文件命名

√Lib 库工程的输出在不同的系统命名稍有不同：

Linux：	lib+模块名+.a	如：libosp.a
VxWorks:	模块名+.a	如：osp.a
Windows：	模块名+lib.lib	如：osplib.lib

13 文件提交

所有与开发相关的文件都就该提交到配置库里面，保证工作成果完整、不丢失。另一方面，所有的垃圾文件严禁提交到配置库里面。一般来讲，需要提交的文件有源文件、配置文件、Readme 文件、工程文件、脚本文件和其它必要的文件。

- √源文件包括：*.cpp, *.c, *.h，*.rc
- √配置文件：*.ini
- √Readme 文件：readme.txt
- √工程文件包括：

Windows 工程：*.dsp（集成环境）、*.mak（自动编译环境）

VxWorks 工程：*.wpj（集成环境）、Makefile 和 prjObjs.lst（自动编译环境）

- √脚本文件：数据库脚本、测试脚本、编译脚本（*.bat，*.sh）
- √其它必要文件包括：bmp、ico 图文件，或者其它编译必需的文件

√其它任何诸如 dsw，wsp,以及编译过程中产生的中间文件以及最终可执行文件严禁放入相应目录提交！

√任何文档、代码等，不能以压缩文件的方式放入配置库中。

14 参考资料

1. 《KDV 工程约定》
2. 《KDV4.0 配置管理计划》
3. 《Linux 项目文档》
4. 《CCS 平台》