SEC-SWC ASP GDL-00004
Ver. 6
2001-07-28

Coding Style Guideline

SAMSUNG
삼성전자

SEC-SWC ASP GDL-00004
Ver. 6
2001-07-28

Copyright notice

작성자 : 홍준성, 이강선, 이선아 (소프트웨어센타)

상 태 : Public

대 상 : Developer

개 요 : 이 문서는 C/C++ programming language로 작성된 program의 일관성과 가독성을 높이기 위해 제안된 style guide이다.

연 혁 :

VERSION	날짜	내용	작성자	승인자
1	1999. 03.25	Initial revision	홍준성, 이강선	홍준성
2	1999.04.02	SA팀 내부 review 결과 반영	홍준성	
3	1999.04.12	센터의 각 팀에서 review한 결과 반영	홍준성	
4	1999.06.18	Copyright notice 보완	홍준성	홍준성
5	1999.08.18	Comment syntax의 약간의 변화, DOC++에 대한 설명 추가	홍준성	홍준성
6	2000.12.16	Documentation Tool(Surveyor)에 맞도록 수정, SHP 진행에서 나온 사항 반영.	홍준성, 이선아	

목 차

1. 개요

1.1. 목적

본 문서는 소프트웨어 개발 단계 중 구현단계에서의 가장 중요한 산출물인 source code를 위한 coding style을 표준화하여 code를 쉽게 읽고 이해할 수 있게 함으로써 프로그램의 유지보수를 위한 비용을 줄일 수 있도록 하기 위하여 작성되었다. 본 표준을 이용하여 프로그램을 구현하였을 시 다음과 같은 기대 효과를 얻을 수 있다.

1) Documentation Tool을 이용한 Source Code의 문서화
2) 프로그램의 판독성 및 유지보수성 향상
3) 규격에 맞는 Code 생산
4) 소프트웨어 제품의 품질 향상
5) 생산성 향상

1.2. 범위

본 문서는 ANSI C를 기준으로 작성하였고, C++에만 존재하는 내용은 [MC++]로 표기하여 설명하였다. 해당 문서는 특정 분야의 program 구현을 위해 수정될 수 있다.

1.3. 정의 및 약어

지침의 정도는 다음의 네 단계로 나누어진다. 각 단계 중 특정한 programming language에 대응되는 규칙에 대해서는 language의 이름을 뒤에 붙인다. (예: MC++)

1) M(Mandatory): 의무적으로 지켜야 함.
2) R(Recommended): 의무적으로 지켜야 하는 것은 아니지만, 지킬 것을 권장함.
3) O(Optional): 지침을 참고하여 지키기로 결정되는 경우 지킴
4) S(Surveyor): Surveyor(Documentation Tool)과 관련된 부분

1.4. 코딩 시작 시 결정 사항과 역할

Project Leader는 coding의 시작 전에 다음의 사항에 대한 결정을 내려야 한다.

1) Project comment의 형태를 정한다.
2) 지침의 정도: 의무적인(M) 지침에 대해 선택 가능한 경우가 있을 경우, 이를 선택하고, 권장(R) 지침과 참고(O) 지침의 경우 이를 수용할 것인지를 결정한다.
3) Hungarian notation에서 <prefix> <base-type>에 대한 사용 여부를 결정하고 상세하게 정의한다.
4) 예외 항목 list를 작성한다.

개발자 중 한명은 다음의 file들을 제공한다.

1) 통합 개발 환경을 사용하지 않는 경우, Makefile의 공통된 형식을 제공한다.
2) Debugging관련 macro를 구현한다.

1.5. 참고 문헌

1. L.W. Cannon, et. al., *Recommended C Style and Coding Standards Revision 6.0*, 1990.
2. Doug Klunder, Hungarian Naming Conventions, 1988.
3. C/C++ Programming Standard, 삼성전자 기간네트워크 사업부
4. Steve McConnell, *Code Complete*, Microsoft Press, 1993.

2. Program Layout

2.1. Working Directories

[M] 작업 디렉토리를 관리하기 편하게 서브 디렉토리들로 나누어야 한다. 프로젝트가 여러 모듈로 이루어진 경우 소스 안에서 다시 서브 디렉토리를 나눈다.
[R] Editor의 프로젝트 옵션에서 outdirectory와 search path 설정도 확인한다.

Example
Src: 프로젝트 소스 파일 Bin: 컴파일된 단위, 실행 파일. Lib: 프로젝트에 필요한 object 파일 Doc: 소스와 관련된 문서. Img: 그림파일 원본 Help: 도움말 Res: Resource 파일

2.2. Source File

프로그램 코딩 시 다음과 같은 사항을 주의하여 작성한다.

1) 한 줄에 하나의 명령문을 사용한다.
2) Source code의 한 line은 80 column을 초과하지 않는 것을 원칙으로 한다.
3) 가능한 하나의 function을 하나의 source file로 구성한다. (여러 개의 function들이 짧고, 서로 연관성을 가질 경우에는 하나의 source file에 포함할 수 있다.)
4) 하나의 function내의 실행문(excutable statement)의 총 line의 합은 300-500 line을 넘지 않도록 한다. (주석문을 포함한 전체는 600-1000 line 정도)

2.2.1. Header File

[M] Header file은 다음과 같은 구조를 가진다. 기본적으로 header file에는 어떠한 variable이나 function의 definition이 기술될 수 없다.

Header File의 구조
Heading comments #ifndef HEADER_FILE_ID #define HEADER_FILE_ID #includes #defines Enumeration typedefs Constant definitions Variable declarations Function declarations (C++의 경우 Class declarations) #endif /* HEADER_FILE_ID */

- **Heading comments:** Header file에 대한 정보(File이름, 내용, 작성자, 날짜, 수정내역 등)를 적는다. 자세한 내용은 4.2.2를 참고한다.
- **#ifndef HEADER_FILE_ID/#define HEADER_FILE_ID/#endif /* HEADER_FILE_ID */:** 이 header file이 여러 번 include되는 경우를 막기위해 사용된다. HEADER_FILE_ID는 _<file-name>_H_의 형태를 가진다. 즉, math.h의 경우, _MATH_H_가 된다.
- **#includes:** 일반적으로 사용되는 library와 같은 이름의 header file은 만들지 않는다. 또한, 제공되는 library의 header file의 경우, 반드시 <>기호를 사용하고, 개발되는 header file의 경우 “”를 사용한다. 특별한 경우, header file의 ‘상대적인’ 위치를 기술할 수 있지만, ‘절대적인’ 위치를 기술할 수는 없다.
- **#defines:** Macro를 정의한다.
- **Enumeration:** Enumerated type에 대한 정의를 한다.
- **typedefs:** 사용자 정의 type을 정의한다.
- **Constant definitions**: 상수를 선언한다.
- **Variable declarations**: 변수를 선언한다.
- **Function prototypes:** 제공되는 function들의 prototype을 나열한다.

Good Example	Bad Example
#ifdef _TYPE_H_ #include <math.h> #include “../inc/myheader.h” #endif /* _TYPE_H_ */	#ifdef TYPES_H #include “math.h” #include </proj/inc/myheader.h> #endif

2.2.2. Implementation File

[M] Implementation file은 다음과 같은 구조를 가진다. 특별히 언급되지 않은 항목들은 header file의 경우와 동일하다.

Implementation File의 구조
Heading comments #includes Global variable definitions Imported variable declarations Imported function declarations Local #defines Local constant definitions Local typedefs Local function prototypes Function definitions (C++의 경우 class definitions)

- **Heading comments:** Implementation file에 대한 정보(File이름, 내용, 작성자, 날짜, 수정내역 등)를 적는다. 자세한 내용은 4.2.2를 참고한다.
- **Global variable definitions:** 이 file에서 제공하는 global variable을 기술한다. 모든 global variable은 반드시 초기화 되어야 한다.

✍✍ **Imported variable/function declarations**: extern으로 선언되는 변수 및 함수를 기술한다.

✍✍ **Function definitions:** 이 file에서 구현해야 하는 function들을 기술한다. 기본적으로 breath-first방식, 즉 functional hierarchy의 상위 부분에서부터, 하위 부분으로 정의하는 것을 권장하지만, 필요에 따라 alphabet순, 혹은 depth-first방식이 사용될 수 있다.

2.2.3. 기타

✍✍ [O] Makefile은 주로 Unix환경에서 program의 개발 시 반드시 필요한 file이다. Makefile의 구조에 대해서는 부록 E를 참조하여 작성한다.

✍✍ [O] Portability를 고려한 code를 위해, platform관련된 정보를 저장할 필요가 있는 경우, config.h라는 file에 저장한다.

✍✍ [R] 각 directory마다 README(혹은 Windows환경에서는 README.TXT)라는 이름의 file을 작성하여 directory에 있는 file들의 이름을 나열하고, 간단한 설명을 추가한다.

README
Description : Author : Files :

2.3. Function

✍✍ [M] Function의 return type은 독립적인 줄에 위치한다.

Good Example	Bad Example

Void
main(void)
{
…
}
 	
Node*
FindRoot(int iStart, Node* pHead)
{
…
}
	main()
	{
	…
	}
	
	Node* FindRoot(int iStart, Node* pHead)
	{
	…
	}

✍✍ [M] Function의 입, 출력은 반드시 기술되어야 한다. int가 return되는 경우, 혹은 input parameter가 없는 경우에도, 각각 int, void가 명시되어야 한다.

✍✍ [M] 내부적으로 사용되는 function은 반드시 static keyword를 사용하여, scope를 명시해야 한다. 필요에 따라 개발 책임자는 scope에 관한 macro를 결정, 배포할 수 있다.

✍✍ [M] Function declaration (prototype)의 경우, input parameter의 이름을 명시해야 한다.

Good Example	Bad Example

#define PUBLIC
#define PRIVATE static
 	
PUBLIC void main(void)
{
}
	main()
	{
	}

PRIVATE int Foo(void)
{
}	

2.4. Variable

✍✍ [M] Global (extern/static) declaration의 경우, column 1에서부터 기술하며, 한 줄에 하나의 declaration만을 한다.

✍✍ [M] 모든 global (extern/static) definition은 초기화 값을 가져야 한다.

✍✍ [M] 모든 external declaration의 경우 extern을 붙인다. Array의 경우에는, array의 크기를 명시한다.

✍✍ [M] Local (auto) declaration의 경우, pointer를 제외한 variable은 필요에 따라 한 줄에 하나 이상의 declaration을 할 수 있지만, 이는 상호 밀접한 관계가 있는 data에 한한다.

✍✍ [M] Pointer기호 (*)는 type쪽에 붙인다.[1]

✍✍ [M] 동일한 이름의 variable을 하나만 사용하도록 한다. 예를 들어, C에서는 scope rule에 따라 내부적으로 같은 이름의 variable을 사용할 수 있지만, 혼동을 일으킬 수 있으므로 금지한다.

Good Example	Bad Example
extern int caWords[ATOZ]; char* pch = NULL;	int iArray1, iArray2; int csWords[]; char *pch;

[1] Type쪽에 붙이는 경우, type에 대한 readability는 증가하지만, char* s, t, u; 와 같은 경우에, t와 u에 대한 type이 혼동될 수 있으므로, pointer declaration은 한 줄에 하나만을 한다.

3. Naming Convention

본 장에서는 programming language에서 사용되는 name들을 분류하고 이를 어떻게 작성해야 하는가 대하여 기술한다.

1) Name은 의미를 쉽게 파악할 수 있도록 한다.
2) 각 Name은 식별이 보장되도록 한다.
3) 혼동될 문자는 가급적 사용하지 않도록 한다.
4) 발음 상 혼동이 되는 Name의 사용을 피한다.
5) Global 변수는 등록한 후 임의 갱신을 피한다.

(주의: Naming convention은 project의 특성상 사용하는 tool, 다른 library와의 호환성, 표준 API등에 의해 영향을 받으며, project의 구현 전단계에 본 guideline을 기초로 변경할 수 있다.)

3.1. File Name

✍✍ [M] File이름은 해당 file에 적혀 있는 함수명과 일치하는 것을 원칙으로 한다. 만약 여러 개의 함수가 같이 들어가 있은 file일 경우에는 대표되는 함수명을 file명으로 한다. 대표 함수명으로 나타내기 어려울 때에는 file의 내용을 잘 설명할 수 있는 이름으로 정한다.

✍✍ [M] 다른 component나 module간의 file name 충돌을 피하기 위하여 각 file name에는 component나 module을 나타내는 미리 약속된 prefix를 사용하여야 한다. 부록 H를 참조하도록 한다.

✍✍ [M] File이름은 <base name>.<suffix>의 형태로 구성된다. <base name>은 영문대문자로 시작되며 하나 이상의 단어로 이루어질 경우 새로운 단어는 대문자로 시작된다.

✍✍ [M] <base name>은 영문자[2]와 숫자, 그리고 '_'만을 사용한다.

✍✍ [MC] <suffix>는 C header file의 경우 .h, C implementation file의 경우 .c를 사용한다.[3]

✍✍ [MC++] <suffix>는 C++ header file의 경우 .h, C++ implementation file의 경우 .cpp를 사용한다.

Good Example	Bad Example
Types.h MachThread.h	wap_types.h mach_thread.c

3.2. Function Name

✍✍ [M] Function 명은 기능을 잘 표현할 수 있도록 정의해야 한다.

✍✍ [M] 동사와 명사로 이름을 정하는 경우, <동사><명사>의 순으로 배열하며 4단어 이상 조합하지 않는다.

✍✍ [R] Function이름은 영문자와 숫자로 구성되며, 31문자를 넘지 않도록 한다[4]. 이름의 시작은 C/C++의 경우 모두 대문자이다. 하나 이상의 단어로 이루어질 경우, 새로운

[2] Samba, DOS와 같은 몇몇 환경에서 대소문자의 구분이 잘 이루어지지 않는다.
[3] Assembly file, object file등은 assembler, compiler에 따라 의존적이다. (예: Unix의 경우, assembly file은 .s, PC의 경우 .asm suffix를 가진다.)

단어는 대문자로 시작한다. 특정 component나 module을 가리키는 prefix의 사용 시 '_'를 사용할 수 있다.

✍✍ [R] External function에 대해서는 component나 module을 나타내는 미리 약속된 prefix를 사용하도록 한다. 부록 H를 참조하도록 한다.

✍✍ [R] 가능한 약자를 피한다.

Good Example	Bad Example
OsInitializeMemory() OsReadFile()	MemoryInitialize() initialize()

3.3. Variable / Function Parameter Name

✍✍ [M] Variable과 function parameter이름은 소문자로 시작하며, 31문자를 넘지 않도록 한다. 또한 하나 이상의 단어로 이루어질 경우, 새로운 단어는 대문자로 시작한다.

✍✍ [M] Variable과 function parameter이름은 기본적으로 Hungarian notation을 따른다. 즉, <prefix><base type><qualifer>의 형태를 가진다.

✍✍ [R] <prefix>와 <base type>은 기본적으로 다음과 같은 사용법을 따라야 하며, project에 따라 추가할 수 있다.

Category	prefix	Description
Prefix	p	Pointer
	a	Array
	i	Index
	c	Count
	d	Difference
	e	Element of an array
	g	Global variable
	m	Module-level variable
Base type	F	Flag : Boolean/ logical value
	ch	character
	sz	String
	fn	Function
	fl	File
	w	Word
	b	Byte
	l	Long
	u	Unsigned
	d	Double
	r	Float
	v	void
Qualifier	First	집합(array등)의 첫 원소
	Last	집합의 마지막 원소
	Lim	집합의 상위 limit, 즉xLast+1
	Min	집합의 절대적 첫 원소
	max	집합의 절대적 마지막 원소 Min <= First <= Last <= Lim <= Max.

[4] ANSI C에서 identifier의 길이는 32문자 이내로 제한된다.

Good Example	Bad Example

pch /* character에 대한 pointer */
ppach /* character array에 대한2차원 pointer */
bool fHoldOn; /* boolean */
Uint16 uRcvTID; /* 16 bit unsigned int */
 	
for (iaReformat = 0;
iaReformat < iaLastReformat;
iaReformat++)
{
…
}
	for (i = 0; i < 100; i++)
	{
	sum += i;
	}

3.4. Type Name

사용자 정의 변수 type이란 typedef를 이용하여 프로그램에 대한 이해를 확실히 해주기 위해 작성된다.

1) 기능을 짐작할 수 있는 타입 이름을 만든다.
2) 미리 정의된 타입은 피한다.
3) 정의된 타입을 재정의하지 않는다.
4) 이식성을 위해 대체되는 타입을 정의한다.
5) Define문을 이용하여 변수를 선언하지 않는다.

✍✍ [M] Type의 이름은 영문자와 숫자로 구성되며, 31문자를 넘지 않도록 한다. 이름의 시작은 대문자이며, 둘 이상의 단어로 이루어질 경우 새로운 단어는 대문자로 시작한다.

✍✍ [M] 각 type은 type으로부터 생성되는 instance들에 대한 추상화이므로, 단수 형태로 작성한다.

✍✍ [M] struct, union, enum의 경우 typedef를 사용하여 새로운 type을 만들어 사용한다. 단 struct등의 내부에서 정의되어 내부에서만 사용되는 경우에는 굳이 type을 만들 필요는 없다.

Good Example	Bad Example

typedef struct tnode* pTree;
typedef struct
{
char* pchWord;
int cOccurences;
TreePtr left;
TreePtr right;
} TreeNode;
	struct splodge_t
	{
	int cSp;
	char* szName;
	};
	
	struct splodge_t s1, s2;

3.5. Enumerated Type Name

✍✍ [M] External type에 대해서는 component나 module을 나타내는 미리 약속된 prefix를 사용하도록 한다. 부록 H를 참조하도록 한다.

✍✍ [M] Enumerated type의 각 Enumerated Constant를 정의할 때 prefix를 사용함으로써 동일 그룹에 속하는 타입을 명백히 한다. prefix는 enumerated type 이름 혹은 축약된 대문자 형태가 된다.

✍✍ [R]각 Enumerated Constant들은 constant 이름 정의 규칙을 따른다.

Good Example	Bad Example
typedef enum { COLOR_INVALID, COLOR_RED, COLOR_GREEN, COLOR_BLUE } Color;	Typedef enum { INVALID, RED, GREEN, BLUE } Color;

3.6. Constant Name

✍✍ [M] Constant이름은 모두 대문자로 구성되며, 단어 사이의 구분은 '_'를 사용한다.

✍✍ [M] Program 내부에는 어떠한 constant (예: 123, "abc")도 직접적으로 사용할 수 없으며, 예외적으로 array의 시작 index인 0을 허용한다.

✍✍ [RC/C++] 특별한 이유(예: Compiler가 ANSI C를 지원하지 않는다. 혹은 array declaration의 size로 사용한다. 등)가 없는 한, #define에 의한 constant의 선언은 하지 않는다.

Good Example	Bad Example

const int MAX_POOL_LEN = 128;
const char* ERR_MSG = "Oops!";
puts(ERR_MSG);
	#define MAX_POOL_LEN 128
	puts("Oops");

3.7. Class Name

✍✍ [MC++] Class란 abstract data type의 implementation이므로 class name은 3.4절의 type에 대한 naming scheme을 따른다.

✍✍ [MC++] Member function name은 3.2절의 function naming convention을 따른다.

✍✍ [MC++] Class의 member variable(attribute) name은 3.3절의 variable naming convention을 따른다.

✍✍ [MC++] Attribute값을 얻는(get) method를 정의하는 경우 'Get'을 attribute 이름 앞에 붙인다. 수정하는(set) 경우 'Set'을 attribute name앞에 붙인다.

ATTRIBUTE	TYPE	GET METHOD	SET METHOD
firstName	String	GetFirstName()	SetFirstName()
address	Address Object	Get Address()	SetAddress()

4. Comments

Comment는 program의 각 구성 요소에 대한 설명을 위해 사용되며, source code 작성시에 같이 작성되어야 한다. 또한 항상 code에 대한 설명보다 idea에 대한 설명이 이루어져야 한다. 기본적으로 comment는 영문으로 작성되어야 하며, project의 특성상 한글로도 작성될 수 있다.

Good Example	Bad Example
compute mean value	sum of values divided by n

4.1. Comments의 분류

Comment는 Block Comments와 End-line Comment로 분류된다. Block comments는 프로그램이나 Function앞, Code 여러 줄을 설명 및 요약하는 경우에 사용되며 End-line comments는 프로그램 내부에 위치하는 하나의 문장이나 실행문의 끝을 표시하기 위해 사용된다.

✍✍ [M] Block comment를 사용하는 경우는 다음과 같다.

1. File의 시작
2. Function definition의 시작
3. Statement앞
4. Declaration등이 grouping이 가능한 경우
5. Pseudo-code로 algorithm을 기술하는 경우 등

✍✍ [M] End-line comment를 사용하는 경우는 다음과 같다.

1. Data declaration & definition
2. Block의 끝 (예: for, while, if등의 compound statement의 길이가 길 때)
3. #endif

4.2. Block Comments

Block comment는 항상 설명할 대상의 앞부분에 위치한다. 표현 방법은 다음과 같다.

✍✍ [M] 한 줄 이상의 block comment 앞에는 한 줄을 띄운다.

한 줄로 표현 가능한 경우	한 줄 이상이 필요한 경우
/* comment */	/* * com * ment */

4.2.1. Project Comment

✍✍ [M] Project comment는 모든 file에 공통적으로 필요한 내용을 기술한 comment이며, file의 맨 앞부분에 위치한다. 주로 다음과 같은 형태를 가진다. 이 부분은 개발 책임자에 의해 정의되어, 각 개발자에게 배포된다.

```
/*
 * Project 이름
 * Copyright 정보 등
 */
```

✍✍ [M] 소프트웨어 센타에서 개발된 모든 source code의 project comment는 copyright 정보를 포함하여야 한다.

Good Example

```
/*
 * Wireless Application Protocol Developed by WAP team++
 *
 * Copyright 1999 by Software Center, Samsung Electronics, Inc.,
 * 599-4 Shinsa-Dong, Kangnam-Gu, Seoul, Korea.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Samsung Electronics, Inc. ("Confidential Information").  You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Samsung.
 */
```

4.2.2. File Comments

✍✍ [M] File comment는 각 file에 대한 정보를 기술한 comment이며, project comment의 바로 다음에 위치한다. File comment의 형태는 다음과 같다.

```
/**[5]
* File name: 이 file에 대한 설명 (완전한 문장으로 기술한다.)
 *
 * @author        [M] 작성자에 대한 정보[6]
 * @version       [O] $Revision$[7]
 * @see           [O] 관련 정보
 */
```

Good Example

```
/**
 * wtp_c2sar.c: This file implements the class 2 transaction using segmentation
 * and re-assembly.
 *
 * @author Joon Sung Hong (3416-0419, jshong@swc.sec.samsung.co.kr)
 * @version $Revision$
 */
```

✍✍ [R] Module 이름의 경우, component에 속한 module임을 나타내기 위하여 '.'을 사용하여 계층적으로 module이름을 나타낸다.
(예) wap.wtp.sar

✍✍ [S] Surveyor4.5의 경우 파일 단위의 comment는 사용하지 않으며 function 단위의 comment만을 on-line document로 추출하게 된다. 각 파일을 Function group으로 묶어 보이려면 관련된 각 파일의 가장 상단에 "/*:Associated with "Function group" */"을 기술한다. 해당 사항과 관련된 Surveyor 4.5의 Rule은 다음과 같다.

---

[5] **가 두개임에 유의한다.

[6] 작성자가 한명 이상인 경우에는, @author를 각각의 작성자의 이름과 전화 번호, e-mail등의 연락처에 대해 붙여 여러 줄에 걸쳐 작성한다. 작성자가 알려지지 않은 경우에는, "unidentified"라고 쓴다.

[7] Version control tool을 사용 하지 않는 경우, 작성자가 직접 작성하고, file의 내용을 수정하지 않는 version control tool의 경우에는 이 부분은 생략한다.

Rule1. /*:Associated with...*/의 위에 있는 주석을 그룹 설명으로 받아들인다.
Rule2. 여러 파일에서 /*:Associated with...*/의 위에 중복된 설명이 있을 경우, 가장 긴 설명을 해당 그룹 설명으로 받아들이고, 나머지 주석은 사용하지 않는다.
Rule3. /*:Associated with...*/는 다음 /*:Associated with...*/가 나올 때까지만 유효하다.

Good Example

```
/*:Associate with class "OS Critical Section" */
/*
 * Samsung Handset Platform
 * Copyright (c) 2000 Software Center, Samsung Electronics, Inc.
 * All rights reserved.
 *
 * This software is the confidential and proprietary information
 * of Samsung Electronics, Inc. ("Confidential Information"). You
 * shall not disclose such Confidential Information and shall use
 * it only in accordance with the terms of the license agreement
 * you entered into with Samsung Electronics.
 */
/**
 * OsCriticalSection.c: This file implements the critical section management functions.
 *
 * @author Joon Sung Hong (3416-0419, jshong@swc.sec.samsung.co.kr)
 * @version $Revision$
 */
                    ……
```

4.2.3. Function Comments

✍✍ [M] Function comment는 각function definition에 관한 정보를 기술한 comment이며, 각function definition의 앞 부분에 위치한다. Function comment의 형태는 다음과 같다.

```
/**
 * Function이 사용하는 algorithm, limitation등을 기술함 (완전한 문장으로 기술한다.)
 *
 * @author      [O] 작성자에 대한 정보
 * @param       [M] 이름 설명([in, out, inout중 하나] Parameter의 종류, 범위 )[8]
 * @return      [M] 설명[9]
 * @exception   [MC++] C++의 경우 throw하는 exception들에 대한 설명
 * @see         [O] 관련 정보 (Function이름, module이름 등)
 * @version     [O] Version 정보
 */
```

Good Example

```
/**
 * This function sorts the numbers between two elements of an array. (No side effects.)
 *
 * @param    aData,       [inout] Sorts array elements iFirstElmt..iLastElmt
 * @param    iFirstElmt,  [in] Index of first element to sort
 * @param    iLastElmt    [in] Index of last element to sort
 */
void
InsertionSort(SortArray aData, int iFirstElmt, int iLastElmt)
{       …
}
```

[8] 하나 이상의 parameter가 있는 경우에는, 한 줄에 하나의 parameter씩, 실제 선언된 순서에 따라 작성한다. Parameter가 없는 경우에는 생략한다.
[9] Return type이 void인 경우에는 생략한다.

4.3. End-line Comments

End-line comment는 설명할 대상이 기술된 위치에 작성하는 comment이며, 다음과 같은 경우 사용한다. End-Line Comment는 다음과 같은 경우 사용한다.

1) 데이터 선언부를 설명하기 위하여
2) 에러수정 등 관리내역을 기록하기 위하여
3) Block 끝을 표시하기 위하여

✍✍ [M] 한 줄로 대상의 뒤에 기술한다.

statement /* comment */ (C++의 경우에는 // comment)

✍✍ [R] Comment가 길어서 한 line에 표현할 수 없을 때에는 block comment 형태를 사용하며 다음 line에 동일한 column에 시작되어야 한다.

✍✍ [R] Source Code가 긴 경우는 다음 line에 comment를 작성하며 comment의 시작은 code 작성 위치에서 한 tab을 띄운 후 시작한다.

Good Example	Bad Example

/* For each element of the set, */
while (pList != NULL)
{
/* special case */
if (a == EXCEPTION)
b = TRUE;
/* works only for odd a */
else
b = CheckPrime(a);
} /* while */
typedef enum
{
REPLY_OK, /* No error */
REPLY_CANCEL, /* Cancel the request */
};
	If (a == EXCEPTION)
	b = TRUE; /* special case */
	else
	b = CheckPrime(a); /* works only for odd a */

5. Programming Style Guide

5.1. Indentation

Indentation은 program의 readability를 결정하는 중요한 요소이다.

- ✍✍ [M] 모든 indentation은 <Tab> key를 사용하여 이루어진다.
- ✍✍ [R] indent의 크기는 4로 정한다.[10] (예외: File/function comment의 작성시, 각 항목이 한 줄을 넘는 경우)
- ✍✍ [R] 복합문이 중첩된 경우 포함될 내용은 복합문장의 Tab위치보다 한 Tab만큼 오른쪽으로 띄워 기록한다.
- ✍✍ [M] '{' 기호와 '}' 기호는 같은 indentation을 가지며, 그 줄에는 어떠한 code도 위치할 수 없다.(Comment제외) {} 사이의 내용이 한 줄 안에 충분히 들어갈 정도로 적은 경우 동일한 줄에 위치할 수 있다.

Good Example	Bad Example

struct boat
{
int wllength;
BoatType type;
long sailArea;
};
struct boat winner[] =
{
{ 40, YAWL, 60000L },
{ 28, MOTOR, 0L },
{ 0 },
};
 	
if (((a + b) / (c + d)) == 0)
	struct boat {
	int wllength;
	BoatType type;
	long sailArea;
	};
	
	if ((a+b)/(c+d)) == 0)

5.2. Blank space

- ✍✍ [M] Keyword (if, while, return, switch, for등) 와 '('사이에 한 칸을 띄운다.
- ✍✍ [M] ','뒤에 새로운 줄이 시작되지 않는 경우, 다음 항목과 한 칸을 띄운다.
- ✍✍ [M] Binary operator의 경우 operator의 양쪽에 한 칸을 띄운다. (주의: ->, ., [] operator는 칸을 띄우지 않는다.)
- ✍✍ [M] Unary operator의 경우 operator와 operand 사이를 붙인다.
- ✍✍ [M] ( )가 nesting되어 나타나는 경우, 최대한 readability를 고려하여, 빈 칸을 띄운다.

[10] 일반적으로 program editor는 <Tab>의 크기를 정하는 기능을 제공한다. 예를 들어, vi의 경우 tabstop과 shiftwidth를 각각 변경하면 되고(Default 8), Microsoft Developer Studio의 경우 Tools/Options menu의 Tabs property window내에서 변경할 수 있다.(Default 4)

5.3. Control Statements and Loops

5.3.1. if

✍✍ [M] If-else statement에서 else 부분은 새로운 줄에 위치한다. If-else if-else의 경우에 else if부분은 독립적인 줄에 기술한다.

✍✍ [R] 정상적인 경우를 먼저 처리한다.

✍✍ [R] 판단문은 단순화한다(Bool 함수 등을 이용한다.)

✍✍ (참고: Compound statement {}의 경우 '{'와 '}'에는 어떠한 code도 위치할 수 없다는 규칙에 대한 적용이다.)

Good Example	Bad Example

if (STREQ(reply, "yes"))
{
statement
}
else if (STREQ(reply, "no"))
{
statement
}
	if (STREQ(reply, "yes")) {
	statement
	} else if (STREQ(reply, "no")) {
	statement
	}

5.3.2. switch

✍✍ [M] Switch statement에서 case는 switch와 같은 indent를 가지며[11], 독립적인 줄에 위치한다.

✍✍ [M] default: 를 제외한 모든 case에는 break statement를 사용한다.

✍✍ [M] statement가 있지만 의도적으로 break statement를 사용하지 않는 경우에는 comment /* FALL THROUGH */를 붙여야 한다.

✍✍ [R] default문은 에러를 찾는 목적으로 사용한다.

Good Example	Bad Example

switch (expr)
{
case ABC:
case DEF:
statement
break;
case UVW:
statement
/* FALL THROUGH */
case XYZ:
statement
break;
}
	switch (expr)
	{
	case ABC:
	case DEF:
	statement break;
	case UVW:
	statement
	case XYZ:
	statement
	}

5.3.3. while

✍✍ [M] For/while statement에서 null body를 가지는 경우에는, null statement를 독립된 줄에 위치시키고, comment /* NULL */을 붙여야 한다.

[11] Switch statement의 경우, 일반적으로 indentation의 깊이가 깊고, case label에 의해 indentation이 없이도 code의 readability가 떨어지지 않으므로, 같은 indent를 사용한다.

Good Example	Bad Example
while (*dest++ = src++) ; /* NULL */	while (*dest++ = src++) ;

✍✍ [M] Do-while 문은 항상 {}를 사용하여야 한다.

5.4. Macros

✍✍ [M] Macro의 이름은 모두 대문자이며, 단어 사이에는 '_'를 추가한다.

✍✍ [M] Macro를 구성하는 모든 expression(각argument, argument들의 expression등)은. ()로 묶는다.

✍✍ [M] Macro에는 global variable을 사용할 수 없다.

✍✍ [M] Macro는 반드시 사용 시에 statement 형태로 사용하도록 한다. 즉 ';'을 macro의 끝에 붙인다.

✍✍ [R] Macro를 구성하는 statement는 'do {} while (0)' 로 묶는다. 다음의 예에서 이와 같이 하지 않는 경우의 문제점을 보인다.

```
#define SP3()    if (b) { int x; av = f(&x); bv += x; }
#define BORK() (zork())
if (x == 3)
    SP3();
else
    BORK();
```

위의 예에서 else에 대응되는 if는 'if (x == 3)'이 아니라, 'if (b)'이다.

Good Example	Bad Example
#define PRODUCT(a, b) ((a)*(b)) #define SP3() \ do\ {\ if (b) { int x; av = f(&x); bv += x; } }\ whiile (0)	#define product(a, b) (a)*(b) #define SP3() \ if (b) { int x; av = f(&x); bv += x; }

5.4.1. Conditional Compilation

✍✍ [M] #ifdef-#else-#endif등과 같은 preprocessing block의 경우에 대해서도 if문에서와 같은 indentation, comment등의 규칙이 적용된다.

✍✍ [M] '#'기호는 모두 column 1에 둔다.[12]

✍✍ [M] #endif 와 같은 줄에 반드시 #ifdef 혹은 #if defined()에서 적용된 condition을 comment 형태로 기술한다.

5.4.2. Conditional Compilation & Platform-Specific Features

[M] Plarform-specific한 경우에 Conditional compilation을 source code에 사용할 수 있다.

[12] Preprocessor의 version에 따라, column 1이외에 있는 '#'기호를 인식하지 못하는 경우가 있다.

Good Example

```
/* wx_x – for code which should work under any X toolkit    */
/* wx_xview – for code which should work under Xview only */
/* wx_motif – for code which should work under Motif only  */
/* wx_msw – for code which should work under Microsoft Windows only */
…
#ifdef wx_x
   (void)wxMessageBox("Sorry, metafiles not available under X");
#endif
#ifdef wx_msw
   wxMetaFileDC dc;
   DrawIt(dc);
   WxMetaFile *mf = dc.Close();
   Mf->SetClipboard();
   Delete mf;
#endif
…
```

[R] Platform-specific한 경우에 따라 함수 정의부터 시작하고 같은 함수명을 사용하는 경우, 이는 Plaform에 따라 함수를 나누어 정의하여 Readability를 높인다.
[S] 해당의 경우 헤더화일에도 같은 함수명이라도 #ifdefined..#else…#endif로 정의한다.

Good Example

```
#if defined(_SHP_OS_REX)
BOOL
OsCleanupSemaphore(HTask hTask)
{
        return OsCleanupQueue(hTask);
}
#else
BOOL
OsCleanupSemaphore(HTask hTask)
{
        register int        i;

        …
                        if (OsDeleteSemaphore(i) == FALSE)
                                return FALSE;
                }

        return TRUE;
}
#endif // _SHP_OS_REX
```

6. Debugging

✍✍ [R] 일관된 debugging code를 위하여 다음과 같은 debugging용 macro를 사용한다.

Macro	Description
SysASSERT(bool)	Expression이 true임을 확인한다.
SysREQUIRE(bool)	Function의 input parameter들의 값의 범위를 확인한다.
SysENSURE(bool)	Function의 수행이 끝나고 난 뒤, return value의 값의 범위와, 다른 variable들의 값의 범위를 확인한다.
SysCHECK(bool)	Conditional compilation에 의해 REQUIRE(), ENSURE()의 input code는 debugging이 끝난 뒤, 삭제되는데 반해, CHECK()의 input code는 남겨진다. 즉, 다음과 같은 code가 가능하다. SysCHECK((pfl = fopen(PARAM_FILE, "r")) != NULL);
SysIMPLIES(bool, bool):	첫번째 expression이 true인 경우, 두번째 expression이 true임을 확인한다.
SysNEVER_GET_HERE():	Control flow가 도달할 수 없는 곳에 두어, 잘못된 flow를 조사한다.
SysVERBOSE((module, code))	DEBUG()보다 보여줄 필요가 많은 message에 대해 사용한다. module은 앞에서 정의한 file prefix를 사용한다. (예: SysVerbose((SMB2, "Hello, %s\n", "World!"));)
SysDEBUG((module, code))	SysDebug((module, code)): module, code는 SysVerbose(())의 경우와 같다.
SysTRACE((module, code))	SysDebug(())와 같은 정도의 message를 보여주는데 사용하지만 그 결과가 file등의 형태로 저장된다. module, code는 SysVerbose(())의 경우와 같다.

Good Example

```
#include "debug.h"
int* mStack;
void
CreateStack(int capacity)
{
     REQUIRE(capacity > 0);
     CHECK(mStack = (int*) malloc(capacity));
}

void
Push(int item)
{
      REQUIRE(!IsFull());
     mStack[++mCount] = item;
     ENSURE(Top() == item);
}
```

✍✍ [R] Macro들의 구현은 개발 담당자가 error발생 시의 처리 등을 개발 환경에 맞도록 일관되게 정의하여, 개발자에게 배포한다. 다음은 구현의 한 예이다.

debug.h

```
#ifndef _DEBUG_H_
#define _DEBUG_H_

typedef enum
{
	ASSERTION_ASSERT,
	ASSERTION_REQUIRE,
	ASSERTION_ENSURE,
	ASSERTION_CHECK,
	ASSERTION_IMPLIES,
	ASSERTION_NEVER_GET_HERE,
} Assertion;

extern void _assert(const char* expr, const char* file, const int line);

#define ASSERT(expr)\
	{\
		if (!(expr))\
		{\
			_assert(#expr, __FILE__, __LINE__);\
			_debug("Intentional abnormal termination.\n");\
			_debug("Use a debugger to keep track of the this point.\n");\
			char* p = (char*)0; *p = 'a';\
		}\
	}

/*
 * REQUIRE: Precondition assertion
 */
#if defined(ALL_ASSERTIONS) || defined(ASSERT_REQUIRE)
#	define REQUIRE(expr)\
	if (!(expr)) { ASSERT(ASSERTION_REQUIRE, #expr); } else {;}
#else
#	define REQUIRE(expr)
#endif /* ALL_ASSERTIONS || ASSERT_REQUIRE */

/*
 * ENSURE: Postcondition assertion
 */
#if defined(ALL_ASSERTIONS) || defined(ASSERT_ENSURE)
#	define ENSURE(expr)\
	if (!(expr)) { ASSERT(ASSERTION_ENSURE, #expr); } else {;}
#else
#	define ENSURE(expr)
#endif /* ALL_ASSERTIONS || ASSERT_REQUIRE */

/*
 * CHECK: Note that this assertion preserves the expression when disabled.
 *		This means that statements such as:
 *		CHECK((fp = fopen("file", "rb+")) != NULL);
 *		are preserved when debugging is disabled.
 */
#if defined(ALL_ASSERTIONS) || defined(ASSERT_CHECK)
#	define CHECK(expr)\
	if (!(expr)) { ASSERT(ASSERTION_CHECK, #expr); } else {;}
#else
#	define CHECK(expr)\
```

```
        if (!(expr)) {;} else {;}
#endif /* ALL_ASSERTIONS || ASSERT_CHECK */

/*
 * IMPLIES: Assertion, which must be true if the first expression is true.
 */
#if defined(ALL_ASSERTIONS)
#       define IMPLIES(expr1, expr2)\
        if ((expr1))\
        {\
                if (!(expr2)) { ASSERT(ASSERTION_IMPLIES, #expr1 "," #expr2); }\
        }\
        else {;}
#else
#       define IMPLIES(expr1, expr2)
#endif /* ALL_ASSERTIONS */

/*
 * NEVER_GET_HERE: Always enabled.
 */
#define NEVER_GET_HERE()\
        ASSERT(ASSERTION_NEVER_GET_HERE, "NEVER_GET_HERE");

/*
 * DEBUG
 */
#if defined(DEBUG)
#       define DEBUG(code) { code; }
#else
#       define DEBUG(code)
#endif /* DEBUG */

#endif /* _DEBUG_H */
```

부 록

A. Wireless Application Protocol Project 적용 예

✍✍Project Comment

WAP(Wireless Application Protocol) 개발 project에서는 다음과 같은 project comment를 사용한다.

```
/*
 * Wireless Application Protocol
 * Developed by WAP team
 * … (Copyright 정보는 4.2.1절 참조)
 */
```

✍✍Naming

다른 project와는 달리 본 project는 일부 variable name, data structure name, type name등이 명세에 포함되어 있다. 그렇지만, 각 layer마다 일관적이지는 못하고, 어느 경우에 대해서 허용을 하는지에 대한 규칙을 case-by-case로 정하는 경우, 더 혼란이 야기될 수 있으므로, 일단을 특별한 이유가 없는 한, 본 문서의 지침을 100% 따르는 것을 기본으로 한다. 예를 들어, GenID라는 variable이 명세에 있는 경우, 구현 시에는 이를 genID라는 이름으로 바꾼다.

본 project에서는 naming convention 중 [M1], 즉 Hungarian notation을 기반으로 하는 convention을 따른다. 하지만, 너무 세부적인 prefix나 base type의 정의는 개발자들에게 부담이 되므로, 다음과 같은 일부 정의만을 따른다.

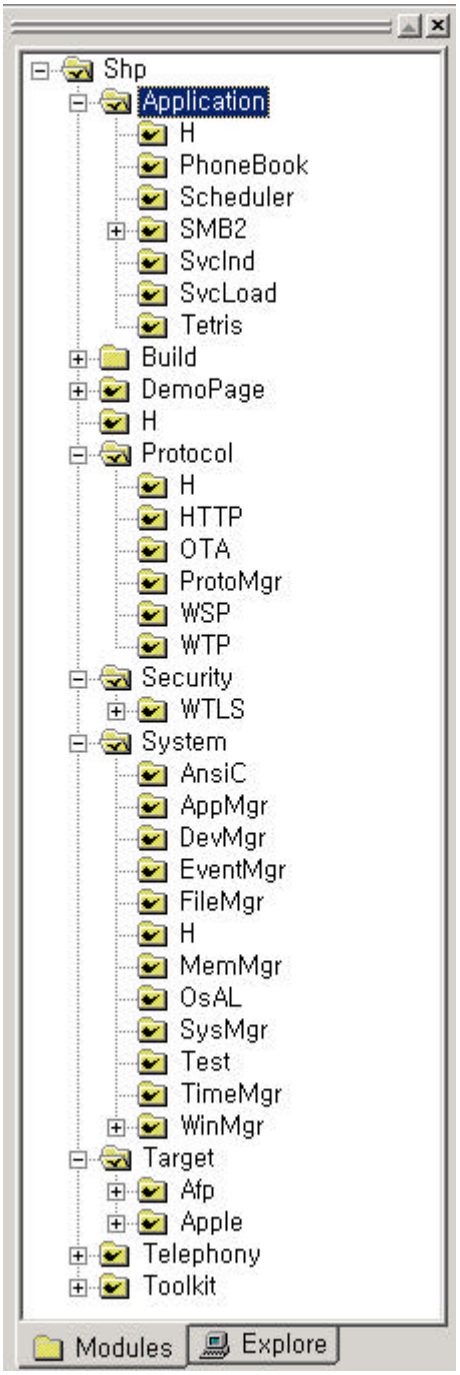
<prefix>
p: pointer
i: index
c: count
g: global variable
m: module-level variable

<base type>:
f: flag (boolean이나 logical값)
sz: string

✍✍예외 항목

모든 [R]에 해당하는 rule도 따르기로 결정하였으므로, 예외 항목은 없다.

B. Sample C Working Directories



그림에서와 같이 SHP의 root directory는 "Shp"이며 모든 Shp component들은 이 directory아래 두게 된다. Top-level directory는 크게 나누어 Application, DemoPage(추후 옮겨질 예정임), H, Protocol, Security, System, Target, Telephony, Toolkit으로 나누어진다.

✍✍ "Application" directory에는 Shp의 모든 application들이 위치하게 되는데 현재까지 개발되었거나 개발 예정인 application들은 각각 하나의 directory를 가지게 된다. SMB2와 같이 규모가 큰 application의 경우 하나 이상의 subdirectory를 가지게 된다.

- ✍✍ "H" directory는 "그 level의 component들이 공통적으로 사용하는" header file들을 저장하는 directory이다. 예를 들면 Shp\H는 Application, Protocol, Telephony 등의 component들이 공유하는 header file들이 위치하게 되며 Shp\Application\H에는 Shp\Application\SMB2, Shp\Application\SvcInd 등이 공유하는 header file들이 위치하게 된다.
- ✍✍ "Protocol" directory는 SHP에서 제공하는 (data) protocol module들을 모아두는 directory이다. (단 여기서 주의할 것은 Security, Telephony와 같이 일정 크기 이상의 grouping이 가능한 'feature'에 대해서는 top-level directory로 정의하는 것이 가능하므로 이 경우에는 (예: WTLS) Protocol directory이외의 영역에 protocol module이 위치할 수도 있다. 이와 같이 feature에 따른 directory를 따로 만드는 이유는 기능간의 configuration을 보다 쉽게 하기 위한 것이다.)
- ✍✍ "Security" directory는 현재까지는 WTLS만이 개발 진행 중이므로 WTLS를 가지며 추후 SSL, WIM, WPKI 등과 관련해서 계속 기능이 추가될 때 이 directory내에서 확장될 것이다.
- ✍✍ "System" directory는 모든 system manager들을 저장하는 directory이다. 앞서 설명한 manager들이 각각 한 subdirectory를 형성하고 있다.
- ✍✍ "Target" directory는 target-dependent한 code들을 일괄적으로 정리하는 directory이다. 현재는 PC simulation을 위한 Apple과 AFP (Auto-Folder Phone) target을 위한 subdirectory가 있다.
- ✍✍ "Telephony" directory의 경우 모든 telephony와 관련된 기능 – WTA, TAPI, SMS 등 – 이 위치하게 된다.
- ✍✍ 마지막으로 "Toolkit" directory는 SHP Toolkit에 필요한 모든 file들을 저장한다.

언제든지 새로운 top-level directory를 추가할 수는 directory구조에 대한 변경은 반드시 SCM engineer에게 미리 공지되어야 한다. 또한 부가적으로 모든 directory 이름은 대문자로 시작하며 다음 단어가 시작되는 경우에는 다시 대문자로 시작하게 됨에 주의한다. (예: ProtoMgr, DemoPage)

C. Sample C Header File

```
/**
 * This file defines the keyword table which counts the number of occurrence of each C keyword.
 * Log: 990305 bwk & dmr Initial revision.
  *
 * @author      Brian W. Kernighan
 * @author      Dennis M. Ritchies
 * @version     1.2
  */
#ifndef _KEYWORD_H_
#define _KEYWORD_H_

typedef struct
{
        char*           szWord;         /* C keyword */
        int             cOccurrence;    /* The number of occurrence */
} Keyword;

int
BinarySearch(char* szWord, Keyword table[], int iMax);

#endif _KEYWORD_H_
```

D. Sample C Implementation File

```
/**
 * This file contains the routine which counts the number of occurrence of each C keyword.
 *
 * Log: 990305 bwk Initial revision.
 *
 * @name        keyword.c
 * @author      Brian W. Kernighan
 * @author      Dennis M. Ritchies
 * @version     1.2
 */
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include "keyword.h"

#define MAX_WORD 100            /* The maximum length of C keywords */
#define NKEYS 50                /* The number of keywords */

static Keyword keywordTable[NKEYS];

void
main(void)
{
        int iKey; /* The index of C keyword table */
        char word[MAX_WORD]; /* Temporary storage for the current read word */

        /* Fills up the C keyword table */
        InitializeKeywordTable();

        /* Gets next word or character from input, and search the read word from
         * the C keyword table
         */
        while (GetWord(word, MAX_WORD) != EOF)
        {
                if (isalpha(word[0]))
                        if ((iKey = BinarySearch(word, keywordTable, NKEYS)) >= 0)
                                keywordTable[iKey].cOccurrence++;
        }

        /*
         * Prints out the results
        */
        for (iKey = 0; iKey < NKEYS; iKey++)
                if (keywordTable[iKey].cOccurrence > 0)
                        printf("%4d %s\n", keywordTable[iKey].cOccurrence,
                                keywordTable[iKey].szWord);
        exit(0);
}

/**
 * This function finds a word in a table. (No side effects)
 *
 * @return      The index of a table. –1 if failed to find a word
 * @param       szWord          [in] The word to search
 * @param       table[],        [in] Keyword table
 * @param       iMax            [in] The maximum index of keyword table
```

```
 */
int
BinarySearch(char* szWord, Keyword table[], int iMax)
{
	int dWords;		/* Stores the difference between two words */
	int iLow, iHigh, iMid;	/* Low, high, and mid index of current searching area */

	/* Sets the search area to the entire region of the table */
	iLow = 0;
	iHigh = iMax – 1;

	/* While there is the remaining region to search, search the table. */
	while (iLow <= iHigh)
	{
		iMid = (iLow + iHigh) / 2;
		if ((dWords = strcmp(szWord, table[iMid].szWord)) < 0)
			iHigh = iMid – 1;
		else if (dWords > 0)
			iLow = iMid + 1;
		else
			return iMid;
	}
	return –1;
}
```

E. Makefile 구조

Makefile은 주로 Unix환경에서 program의 개발 시 반드시 필요한 file이다. Makefile의 syntax에 대한 표준은 없으므로, 상세한 syntax는 make utility의 종류와 version에 따라 약간의 차이를 보이지만, 본 지침서에서는 여러 platform에서 수행 가능한 GNU의 gmake을 기본으로 하여 설명한다.

[R] 일반적인 Makefile의 구조는 다음과 같다.

Heading comments
Macro definitions
Targets
Dependency lines

✍✍ Heading comments
Makefile에 대한 정보(File이름, 내용, 작성자, 날짜, 수정내역 등)를 적는다. 자세한 내용은 4.2.2를 참고한다. (주의: Makefile의 comment는 column 1에 '#'기호를 사용하여 기술한다.)

✍✍ Macro definitions
Makefile에서 사용되는 macro들에 대한 정의를 하는 부분이다. 일반적으로 directory정보, 개발과 관련된 command들 (compiler, linker, assembler등), 그리고 flag등이 포함된다.

✍✍ Targets
Make의 대상이 되는 target을 정의하고, make utility가 제공하는 다양한 keyword를 사용하여, make option을 정의한다.

✍✍ Dependency lines
Make되는 target과 이에 의존적인 source와의 관계를 나타내는 부분이다. 특별한 이유가 없는 한 dependency line은 make rule을 입력하는 경우 외에 직접적인 편집은 금지하며, makedepend와 같은 tool을 사용하여 자동으로 갱신되도록 한다.

Makefile.com (각 Makefile 에 공통적인 내용을 기술: 한 project 에 하나씩)

```
#
# Project:
# File        : Makefile.com
# Author:  Joon Sung Hong (3416-0419, jshong@swc.sec.samsung.co.kr)
#
# M A C R O   D E F I N I T I O N S
#
# Directories
#
BASEDIR                 = /proj
BINDIR                  = ${BASEDIR}/bin
INCDIR                  = ${BASEDIR}/h
LIBDIR                  = ${BASEDIR}/lib
SRCDIR                  = ${BASEDIR}/src
OBJDIR                  = ${BASEDIR}/obj

G++DIR                  = /usr/local/lib
GCCDIR                  = /usr/local/lib/gcc-lib/sparc-sun-solaris2.4/2.7.2

#
# Commands
#
CC                      = g++
MAKEDEPEND              = /usr/bin/X11/makedepend
AR                      = ar
RANLIB                  = /usr/bin/ranlib
```

```
CP                      = /bin/cp
RM                      = /bin/rm
MV                      = /bin/mv
INSTALL                 = /usr/bin/install

#
# Flags
#
INCLUDES                = -I${INCDIR} -I${G++DIR}/g++-include -I${GCCDIR}/include
LDFLAGS                 = -L${LIBDIR}
LIBS                    =
MAKEFLAGS               =
RMFLAGS                 = -f
ARFLAGS                 = rv

#
# T A R G E T S
#
OBJS                    = $(SRCS:%.cpp=$(OBJDIR)/%.o)
IHDRS                   = $(HDRS:%.h=$(INCDIR)/%.h)

# Inhibit the display of commands
#.SILENT:

# Suppress SCCS retrieval
.SCCS_GET:

# Activate command dependency checking
.KEEP_STATE:

# Add suffix rules at the head of the list
#.SUFFIXES:
#.SUFFIXES: ...

# Preserve target against removal due interrupts
.PRECIOUS:

# Retrieve known hidden dependencies
.INIT:

# Debug and profiling
debug    := CFLAGS   = -g
profile  := CFLAGS   = -pg -O

#
# D E P E N D E N C Y    L I N E S
#
${OBJDIR}/%.o: %.cpp
        ${COMPILE.c} $< -o $@

${INCDIR}/%.h: %.h
        ${CP} $(@F) $@
```

Makefile (각 directory(module)에 대한 내용을 기술: directory 마다 하나씩)

```
#
# Module:
# File      : Makefile
# Author: Joon Sung Hong (3416-0419, jshong@swc.sec.samsung.co.kr)
```

```
#
#
# M A C R O   D E F I N I T I O N S
#
CPPFLAGS                = ${INCLUDES}-DALL_ASSERTIONS -DDEBUG
CFLAGS                  = -g –DSOLARIS
DEPENDFLAGS             = ${INCLUDES} ${CFLAGS} -f Makefile -p${OBJDIR}/
TARGETLIB               = ${LIBDIR}/libtarget_g++.a

include Makefile.com

SRCS                     = \
        wtp_invoke.c \
        wtp_assembly.c

HDRS                    = \

IHDRS                   = \
        ${HDRS}

SUBDIRS                 =

#
# T A R G E T S
#
all:  all.local
clean: clean.local
depend:  depend.local
install: install.local

#
# D E P E N D E N C Y   L I N E S
#
all.nested clean.nested depend.nested install.nested:
        ${MAKE} ${SUBDIRS} TARGET=$(@:%.nested=%)

${SUBDIRS}: FORCE
        cd $@; ${MAKE} ${TARGET}

all.local: ${OBJS} ${LIBTARGET}

clean.local:
        ${RM} ${RMFLAGS} ${LIBTARGET}
        ${RM} ${RMFLAGS} ${OBJDIR}/*.o

depend.local: FORCE
        $(MAKEDEPEND) $(DEPENDFLAGS) $(SRCS)

install.local: ${IHDRS} ${LIBTA RGET}
        ${CP} ${LIBTARGET} ${LIBDIR}
        ${CP} ${IHDRS} ${INCDIR}

${LIBTARGET}: ${OBJDIR}/*.o
        ${AR} ${ARFLAGS} $@ $?

FORCE:

# DO NOT DELETE THIS LINE -- make depend depends on it.
```

F. Using Documentation Tool (Surveyor)

✍✍Surveyor란?

Surveyor는 C/C++ 파일들로부터 웹 기반의 문서로 자동적으로 만들어주는 Tool이다. Surveyor는 Source Comment를 생성된 웹 문서의 설명문으로 바꾸어준다.

✍✍Surveyor 설치하기

SHP에서 제공한 service에 대한 on-line document를 생성하기 위해서는 Surveyor를 install하여야 한다. Surveyor는 \\File\공용SW\Surveyor에 있으므로 install하도록 한다. (Key값은 PQNJ VSJX HNAP KALJ 혹은 V6N9 ELE0 VT7P 2VAA를 사용한다.)

✍✍Survyeor 시작하기

Install후 시작 화면은 다음과 같다. 해당의 경우처럼 3개의 윈도우가 보이지 않을 경우 [Window]에서 [Show Files Window][Show Classes Window][Show Log Window]를 선택하면 해당 Window가 보인다.

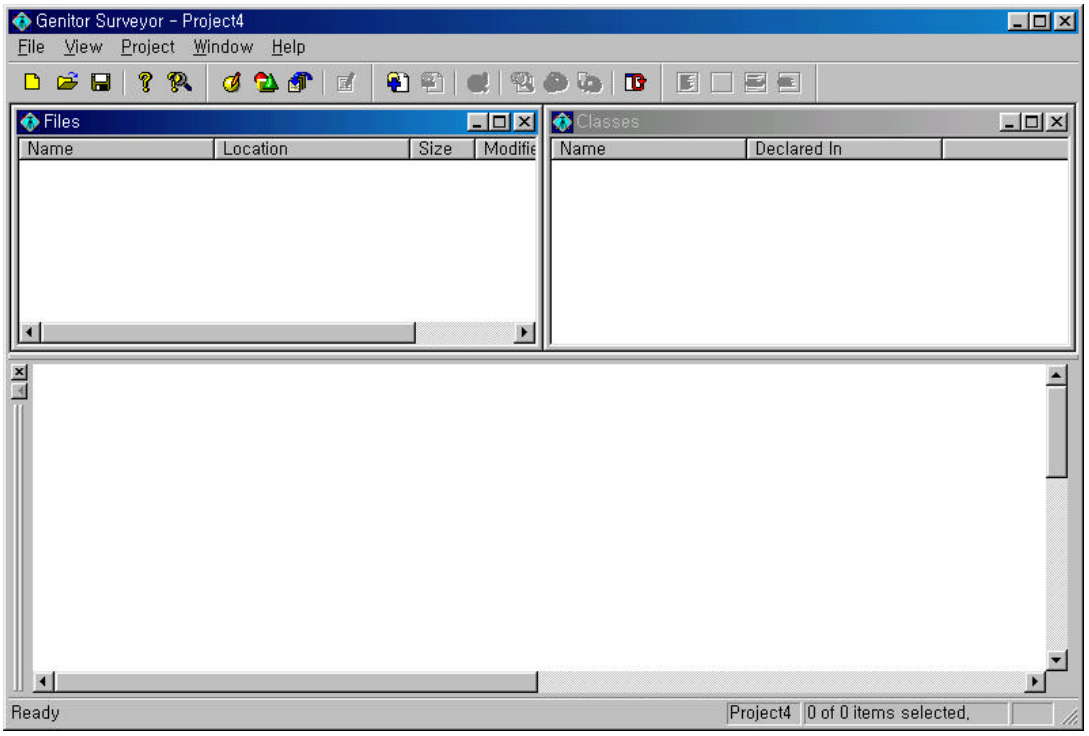


그림 1. Surveyor 시작 화면

✍✍Source File 포함하기

처음 Surveyor를 시작하면 다음과 같은 형태로 Surveyor Window가 보이게 된다. 해당의 경우 [Project Menu]의 [Add Files…]를 선택하고 포함시키고자 하는 Source File을 포함시키면 된다.

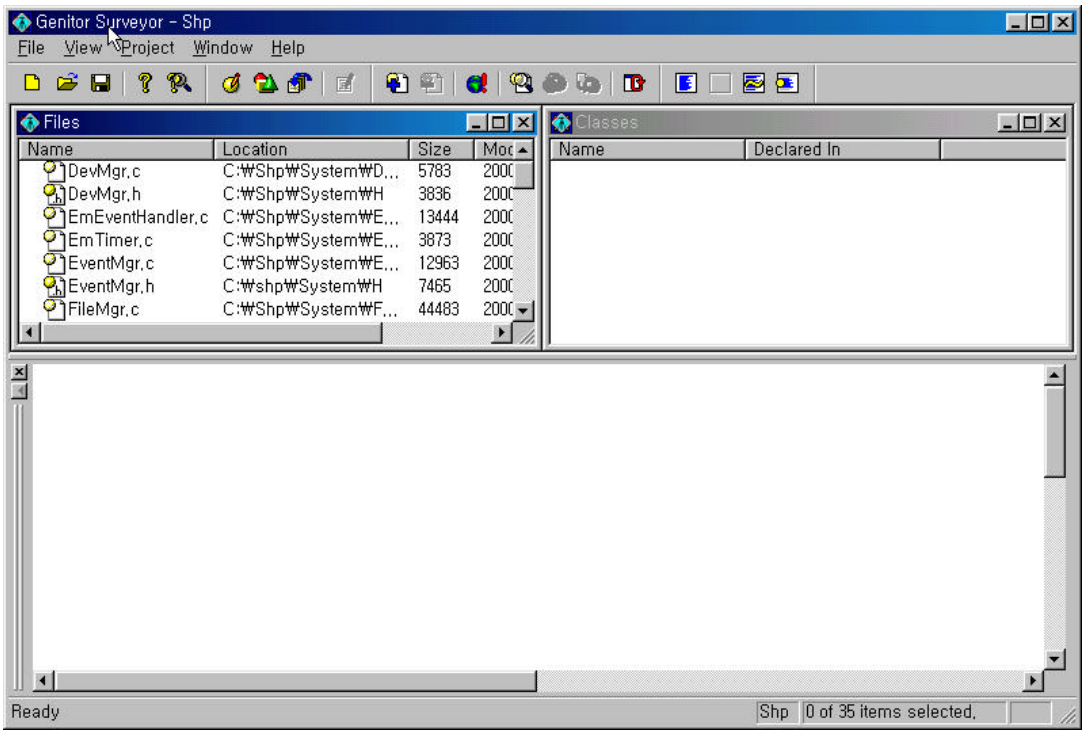


그림 2. Source File이 포함된 모양

✍✍Web 문서로 출력하기

Source File을 포함하면 왼쪽 상단의 Project Window에 포함된 파일 리스트가 보인다. 아래 그림의 상태가 되었다면, [Project Menu]의 [Auto Web Update…]를 선택한다. Surveyor가 자동적으로 Web 문서를 출력하여 보여주게 된다.

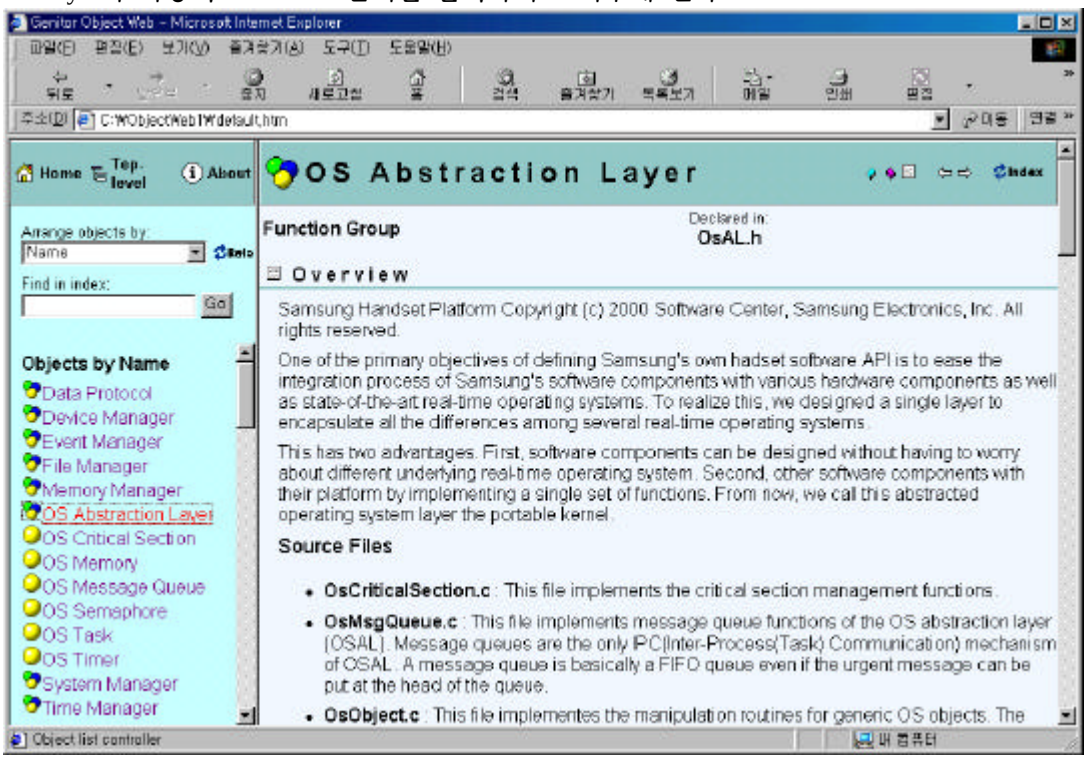


그림 3. Web 문서로 출력된 형태

✍✍Web 문서로 나온 모양 검토하기

위와 같은 방법으로 나온 웹 페이지 모양을 검토한 후, 출력 내용은 [File]/[Preference]/[Generate Web…]을 선택하여 해당 Option을 설정함으로써 조정한다.

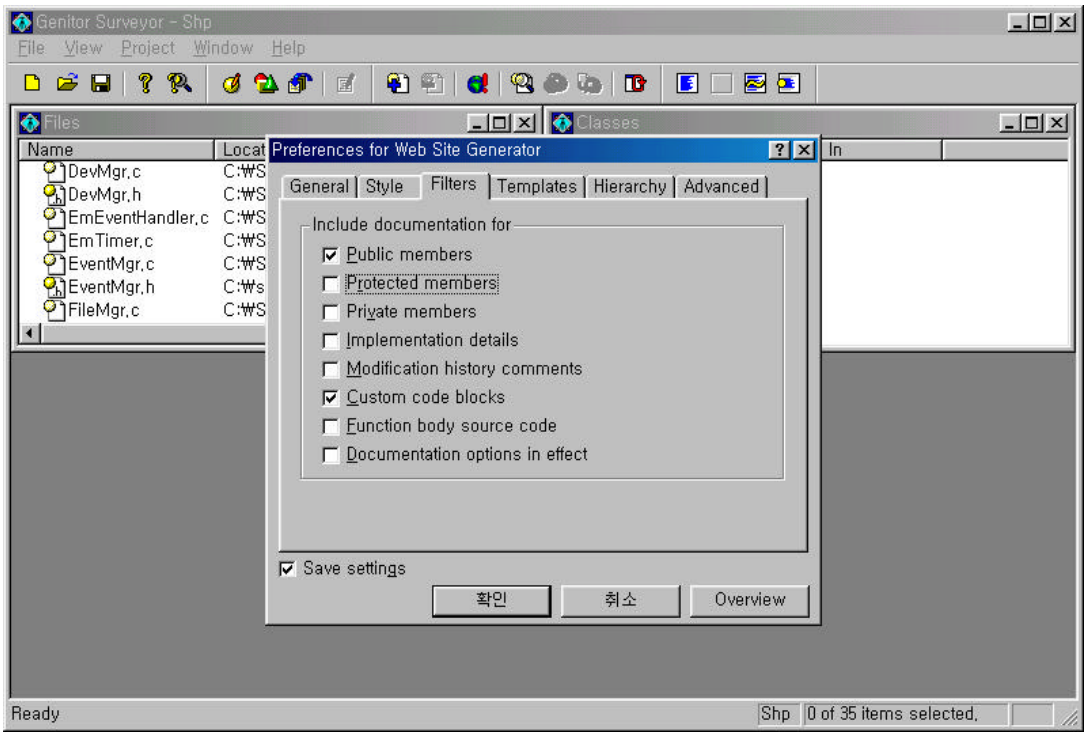


그림 4. Preference for Web Site Generator의 Dialog Box

잘못 나온 정보가 있거나 Grouping을 다르게 하고 싶을 경우, 소프트웨어 센타의 Documentation 담당에게 수정을 요청한다. (E-mail:doc@swc.sec.samsung.co.kr)

✍✍Surveyor의 장단점

Surveyor의 장점

1) 센터에서 지정한 기존 Rule인 javadoc형태의 comment를 지원한다.
2) Function/Class 단위 및 Group Documentation을 지원한다.
3) OCS와 연동하여 Word문서와 WinHelp문서로의 변환도 가능하다.
4) 사용이 편리하다.

Surveyor의 단점

1) File 단위의 Documentation을 지원하지는 않는다.

해당의 미비 사항을 해결할 만한 방법으로는 Surveyor의 상위 버전에 해당 사항이 반영되도록 요청하거나 다른 툴을 사용할 수 밖에 없다. 다른 툴을 사용할 경우 추천할 만한 Tool로는 object-outline(http://www.bbeesoft.com/)로 Freeware이다.
그러나 Surveyor가 웹 출력 모양 등 여러 면에서 우수하므로 가능한 Surveyor 사용을 추천한다.

G. Coding Style Checklist

CODING STYLE CHECKLIST		
Clause	Checkpoint	Check
NAMING CONVENTION		
File	File 이름의 길이, 대소문자, suffix가 적절한가?	
Function	Function 이름의 길이, 대소문자, <동사><명사>형태가 적절한가?	
Constant	Constant 이름이 적절한가?	
	Code 내에 0 이외의 constant가 직접 사용되었는가?	
	특별한 이유없이 constant의 기술을 위해 #define이 사용되었는가?	
Type	Type 이름의 길이, 대소문자가 적절한가?	
	Type의 이름이 단수 형태를 가지는가?	
	struct, union, enum의 경우 typedef를 사용하여 새로운 type을 만들었는가?	
	Enumerated type의 경우, 각 원소들이 constant이름 정의 규칙을 따르며, 각 원소가 enumerated type 이름의 대문자 형태를 prefix로 가지는가?	
Variable / Function Parameter	Variable / Function parameter 이름의 길이, 대소문자, (<prefix>, <base type>, <qualifier>) 가 적절한가?	
COMMENTS		
Block	Block comment의사용 위치가 적절한가?	
	한 줄 이상의 block comment의 경우, 앞 줄을 띄었는가?	
End-line	End-line comment의 사용 위치가 적절한가?	
Project	각 file에 project comment가 있는가?	
File	각 file에 file comment가 적절히 기술되었는가?	
Function	각 function에 function comment가 적절히 기술되었는가?	
	Return type에 대한 설명, 범위가 명시되었는가?	
	각 input parameter에 대한 설명, 범위가 명시되었는가?	
DECLARATIONS & DEFINITIONS		
Global declaration	한 줄에 하나의 declaration만을 하였는가?	
Global definition	모든 global data definition시 초기화가 되었는가?	
extern	모든 external data declaration의 경우 extern을 붙였는가?	
	Array의 경우에, array의 크기를 명시하였는가?	
Local declaration	한 줄에 하나의 pointer variable만을 declaration하였는가?	
	밀접한 관계에 있는 data만이 한 줄에 기술되었는가?	
Pointer	Pointer 기호(*)가 왼쪽에 있는가?	
Scope	동일한 이름이 scope에 따라 중복 사용되었는가?	
Function I/O	Function의 입, 출력이 모두 기술되었는가?	
static	내부적으로 사용되는 function에 대해 static keyword를 사용하였는가?	
Function declaration	Function prototype에서 input parameter의 이름이 기술되었는가?	
INDENTATION		
<Tab> 4	Indentation은 모두 <Tab>을 사용하여 4칸으로 이루어졌는가?	
Block {}	Code의 길이가 일정한 길이(예40 column)를 넘는 경우, '{'와 '}'는 같은 column, 독립된 줄에 위치하는가?	
Comma ,	','뒤에 한 칸을 띄었는가?	

Binary operator	Binary operator의 양쪽은 한 칸씩 띄었는가?	
Unary operator	Unary operator와 operand사이에 빈 칸이 없는가?	
Keyword (	Keyword와 '('사이에 한 칸을 띄었는가?	
(())	()가 nesting되는 경우, 적절히 빈칸을 삽입했는가?	
Function parameter	각function parameter가 한 줄에 한 개씩 기술되었는가?	
if-else if-else	else부가 새로운 줄에 기술되었는가?	
Semicolon ;	Null statement가 독립적인 줄에 기술되고, /* NULL */ comment를 붙였는가?	
switch-case	case가 switch와 같은 indent를 가지는가?	
break	default: 를 제외한 모든 case에는 break statement를 사용하였는가?	
	break가 없는 경우, /* FALL THROUGH */를 붙였는가?	
do-while	{}를 사용하였는가?	
MACROS		
이름	Macro의 이름은 길이, 내용 등이 적절한가?	
Argument	각 argument들은 ( )로 묶었는가?	
Global variable	Macro에 global variable등 context에 따라 오류를 발생시킬 가능성이 있는 부분이 포함되었는가?	
Statement	Statement가 정의되는 경우, do-while로 묶었는가?	
Expression	Macro가 사용 시 ';'을 붙일 수 있도록 만들어졌는가?	
#if-#else-#endif	Indentation은 적절한가?	
	# 기호는 column 1에 있는가?	
	#endif 부에 #if에서 사용된 condition을 기술하였는가?	
PROGRAM 구조		
Header file	Header file이 정의된 구조를 따르는가?	
Impl. file	Implementation file이 정의된 구조를 따르는가?	
Makefile	Makefile이 정의된 구조를 따르는가?	
기타	README file등이 정의된 구조를 따르는가?	
DEBUGGING		
Debugging macro의 사용	(ASSERT) Return값이 있는 모든 function에 대해 error checking이 되었는가?	
	(REQUIRE) 각 function이 잘못된 input data에 대해 조사하는가?	
	(ENSURE) 각 function이 잘못된 output data에 대해 조사하는가?	
	(CHECK) 삭제되어서는 안 되는 code에 ASSERT()를 사용하지 않았는가?	
	(NEVER_GET_HERE) 도달해서는 안 되는 code부분에 대해 충분히 NEVER_GET_HERE() macro가 사용되었는가?	

H. Software Center에서의 Prefix

SAMSUNG HANDSET PLATFORM

Component/Module	Prefix
SMB2 Common	Smb2
Browser Common	Br
Content manager	Ctm
Event Manager	Evm
Presentation Manager	Prm
Request Manager	Rqm
WMLScript	Wmls
E-mail	Em
Phonebook	Phb
Scheduler	Sch
Protocol Common	Proto
WSP	Wsp
WTP	Wtp
WDP	Wdp
System Common	Sys
Os Abstraction Layer	Os
Memory Manager	Mem
File Manager	Fm
Toolkit	Tk