

GSR-020
rev 0.09.01

GIGA Kernel API

변경 내역

버전	배포 날짜	변경사항
0.09.00	2005.11-23	초안작성
0.09.01	2006-01-18	4.4.1절 컨텐츠 개발 요구사항 추가

지적 재산권에 대한 공지

본 문서 및 문서와 함께 제공되는 일체의 정보는 SK 텔레콤의 WIPI 추가 규격을 명시함으로써, SK 텔레콤에 납품하는 단말기 제조사들 및 컨텐츠 개발업체가 WIPI 추가 규격을 구현하도록 하기 위함이다. 본 문서 및 문서와 함께 제공되는 일체의 정보는 SK 텔레콤의 소유물이다. 이의 일부 또는 전부는 SKT WIPI 를 위해서만 SKT 의 승인 아래 복사되거나 배포될 수 있다.

에스케이텔레콤 주식회사

목 차

1 일반사항	5
1.1 개요	5
1.2 목적	5
1.3 규격의 범위.....	5
1.4 용어정의	5
2 의존성	6
2.1 WIPI 의존성.....	6
2.2 GIGA 의존성	6
2.3 GIGA 의존성	6
2.4 OAT 의존성.....	6
3 제조사 포팅 가이드	7
3.1 Define 정의.....	7
3.2 Structure정의.....	7
3.3 HAL API.....	7
4 개발자 가이드	14
4.1 Define 정의.....	16
4.2 Structure정의.....	17
4.3 C API	17

세부목차

1 일반사항	5
1.1 개요	5
1.2 목적	5
1.3 규격의 범위.....	5
1.4 용어정의	5
2 의존성	6
2.1 WIPI 의존성.....	6
2.2 GIGA 의존성	6
2.3 GIGA 의존성	6
2.4 OAT 의존성.....	6
3 제조사 포팅 가이드	7
3.1 Define 정의.....	7
3.2 Structure정의.....	7
3.3 HAL API.....	7
3.3.1 OEMH_gknlSetSystemOperation	9
3.3.2 OEMH_gKnlGetSystemOperation	10
3.3.3 OEMH_gKnlSetSystemProperty.....	11
3.3.4 OEMH_gKnlGetSystemProperty	12
4 개발자 가이드	14
4.1 Define 정의.....	16
4.2 Structure정의.....	17
4.3 C API	17
4.3.1 OEMC_gknlSetSystemOperation	18
4.3.2 OEMC_gKnlGetSystemOperation	20
4.3.3 OEMC_gKnlSetSystemProperty.....	21
4.3.4 OEMC_gKnlGetSystemProperty	22

1 일반사항

1.1 개요

고품질의 GIGA 콘텐츠가 개발됨에 따라 GIGA 콘텐츠의 요구사항을 충족시켜주기 위해 OEM과의 인터페이스 및 플랫폼 제어 인터페이스에 대한 요구사항이 늘어나고 있다. GIGA 콘텐츠는 WIPI 콘텐츠와는 다른 API 접근 제어 정책이 필요하기 때문에, GIGA 콘텐츠의 요구사항을 충족시키기 위해 GIGA Kernel API를 정의하도록 한다.

1.2 목적

본 규격은 SK Telecom(주)(이하 “SK Telecom” 이라 한다.)의 GIGA Kernel API 을 위한 요구사항 및 규격을 명시하는데 목적이 있다.

1.3 규격의 범위

본 문서는 SK Telecom에서 서비스 예정인 GIGA Kernel API를 위한 규격에 대하여 기술한다.

1.4 용어정의

본 문서에서 언급되는 용어들을 정의한다.

2 의존성

본 규격 구현을 위하여, 필요로 하는 WIPI 버전과, 타 WSR 및 본 규격의 검증을 위해 필요한 OAT 버전을 명시한다.

2.1 WIPI 의존성

본 WSR 규격은 아래 WIPI Core 버전 이상이 적용된 단말에 포팅 되어야 한다.

Base WIPI Core 버전
WIPI 1.10.02

2.2 GIGA 의존성

본 GSR 규격은 아래 GIGA Core 버전 이상이 적용된 단말에 포팅 되어야 한다.

Base GIGA Core 버전
GIGA 2.05.00

2.3 GIGA 의존성

본 GSR을 포팅하기 위하여서는 아래 명시된 GSR들을 모두 포팅 하여야 한다.

의존성을 갖는 GSR	설 명

2.4 OAT 의존성

본 규격의 테스트를 위해서는 다음 버전의 OAT가 필요하다.

3 제조사 포팅 가이드

본 절에서는 제조사에서 GIGA Kernel API를 포팅 하는데 필요한 정보에 대해 설명하도록 한다.

3.1 Define 정의

Define 값	설명
MC_GIGA_VIRUP	Up
MC_GIGA_VIRDOWN	Down
MC_GIGA_VIRLEFT	Left
MC_GIGA_VIRRIGHT	Right
MC_GIGA_VIRFIRE	Select
MC_GIGA_VIRGAME_A	Virtual A
MC_GIGA_VIRGAME_B	Virtual B
MC_GIGA_VIRGAME_C	Virtual C
MC_GIGA_VIRGAME_D	Virtual D

3.2 Structure정의

타입	설명
VIRTUAL_KEYMAP	가상키 정보 typedef struct tagVIRTUAL_KEYMAP{ M_Int16 nKeyCode; // OEM Key Code M_Int16 nVirtualKeyCode; // Virtual Key Code } VIRTUAL_KEYMAP;

3.3 HAL API

API	설명
M_Int32 OEMH_gknlSetSystemOperation(M_Char *pszCmd, M_Int32 nCmdCode)	응용 프로그램의 Default System Operation을 변경한다.
M_Int32 OEMH_gknlGetSystemOperation(M_Char *pszCmd, M_Int32 *pnCmdCode)	응용 프로그램의 Default System Operation으로 설정된 값을 가져온다.
M_Int32 OEMH_gknlSetSystemProperty(M_Char *pszID, void	단말기에 특화된 값을 설정한다.

*pBuf)	
M_Int32 OEMH_gknlGetSystemProperty(M_Char *pszID, void *pBuf, M_Int32 *pnBufSize)	단말기에 특화된 값을 가져온다.

3.3.1 OEMH_gknlSetSystemOperation

설명

응용 프로그램의 Default System Operation을 변경한다. Default System Operation이란 플랫폼의 기본적인 동작방식을 말한다.

Default System Operation	설명	기본값
"GIGAMODE"	GIGA 컨텐츠가 실행됨을 알린다. 제 조사는 GIGAMODE가 TRUE로 설정 되면, GIGA 컨텐츠를 실행하는데 필요한 작업을 해줘야 한다.	FALSE

프로토타입

M_Int32 OEMH_gknlSetSystemOperation(M_Char *pszCmd, M_Int32 nCmdCode)

매개변수

pszCmd

[In]

변경하고자 하는 System Operation 명령어.

nCmdCode

[In]

변경하고자 하는 System Operation에 대한 명령 코드

Default System Operation	명령코드	설명
"GIGAMODE"	TRUE	GIGA 컨텐츠 실행 시작
	FALSE	GIGA 컨텐츠 실행 종료

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Operation 명령어

3.3.2 OEMH_gKniGetSystemOperation

설명

응용 프로그램의 Default System Operation에 설정된 값을 가져온다. 사용 가능한 System Operation 명령은 3.3.1 OEMH_gKniSetSystemOperation과 동일하다.

프로토타입

```
M_Int32 OEMH_gkniGetSystemOperation(M_Char *pszCmd, M_Int32 *pnCmdCode)
```

매개변수

pszCmd

[In]

가져오고자 하는 System Operation에 대한 명령어.

pnCmdCode

[out]

System Operation 설정 값.

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Operation 명령어

3.3.3 OEMH_gKniSetSystemProperty

설명

단말기에 특화된 값을 설정한다.

System Property	설명	기본값

(*) 현재 컨텐츠에서 설정할 수 있도록 정의된 System Property는 없다.

프로토타입

M_Int32 OEMH_gkniSetSystemProperty(M_Char *pszID, void *pBuf)

매개변수

pszID

[In]

설정하고자 하는 System Property ID.

pBuf

[In]

설정하고자 하는 System Property ID에 필요한 데이터

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Property ID

3.3.4 OEMH_gKniGetSystemProperty

설명

단말기에 특화된 값을 읽어온다.

프로토타입

```
M_Int32 OEMH_gkniGetSystemProperty(M_Char *pszID, void *pBuf, M_Int32*
pnBufSize)
```

매개변수

pszID

[In]

읽어오고자 하는 System Property ID.

pBuf

[in/out]

System Property가 저장될 버퍼 포인터 또는 System Property ID와 함께 추가적으로 전달해야 할 데이터가 저장된 버퍼 포인터

pnBufSize

[in/out]

컨텐츠는 System Property가 저장될 버퍼의 크기가 지정하며, 플랫폼은 저장된 데이터의 크기를 전달한다.

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Property ID

System Property ID	비고		
"GSR"	단말에 특정 GSR이 포팅 되었는지 여부를 확인한다. 제조사는 pBuf에 지정된 GSR이 포팅 되었는지의 여부를 반환값으로 전달해 주어야 한다.		
	<table border="1"> <tr> <td>아규먼트</td> <td>설명</td> </tr> </table>	아규먼트	설명
아규먼트	설명		

	pBuf	GSR 번호 (예 : "GSR020")						
	pnBufSize	사용되지 않음.						
"VKEYTABLE"	<p>단말에 적용된 Virtual Key Table을 얻어온다. Virtual Key Table은 VIRTUAL_KEYMAP형 배열로 전달된다.</p> <p><u>(*) 제조사는 단말 출시 이전에 반드시 SKT와 Virtual Key Table에 대해 협의를 한 후, 협의된 내용을 Virtual Key Table로 정의해서 단말에 탑재해야 한다.</u></p> <p>(**) pBuf가 NULL인 경우에는 Virtual Key Table을 저장하는데 필요한 메모리의 크기(Byte)를 pnBufSize로 전달하도록 한다.</p> <table border="1" style="width: 100%;"> <thead> <tr> <th>아규먼트</th> <th>설명</th> </tr> </thead> <tbody> <tr> <td>pBuf</td> <td>Virtual Key Table이 저장될 버퍼 포인터</td> </tr> <tr> <td>pnBufSize</td> <td>Virtual Key Table의 크기 (Byte)</td> </tr> </tbody> </table>		아규먼트	설명	pBuf	Virtual Key Table이 저장될 버퍼 포인터	pnBufSize	Virtual Key Table의 크기 (Byte)
아규먼트	설명							
pBuf	Virtual Key Table이 저장될 버퍼 포인터							
pnBufSize	Virtual Key Table의 크기 (Byte)							

4 개발자 가이드

본 절에서는 콘텐츠 개발사에서 GIGA Kernel API를 사용하기 위해 필요한 사항에 대해 설명 하도록 한다.

4.1 콘텐츠 개발 요구사항

4.1.1 Virtual Keymap

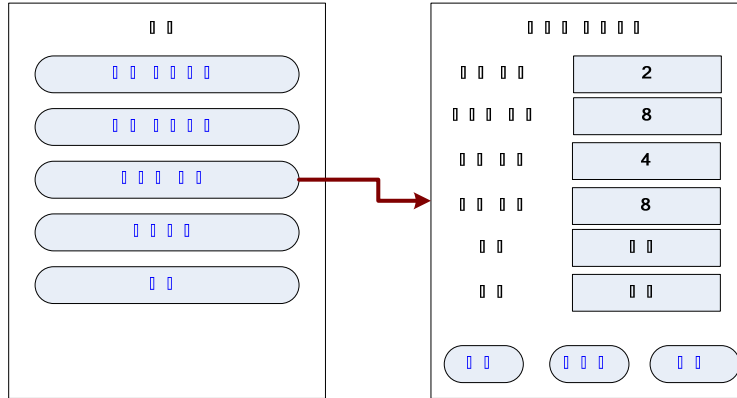
단말은 디자인에 따라서 키배치가 상이할 수 있다. 따라서, 플랫폼에 정의된 키값을 직접적으로 이용해서 콘텐츠를 개발하는 경우, 단말의 디자인에 따른 키배치의 차이로 인해 동일한 게임성을 제공하는데 어려움이 있다.

Virtual Keymap은 게임 콘텐츠에서 반드시 필요로 하는 기능에 대응하는 가상키를 정의하고, 이들 가상키에 대한 실제 키값의 맵핑은 단말기의 디자인과 키배치에 따라 SKT가 제조사에 가이드 함으로써, 서로 다른 단말 상에서도 동일한 게임콘텐츠에 대해서는 동일한 게임성을 보장하기 위한 방법이다.

콘텐츠 개발사는 GIGA에서 정의된 가상키 이외에 추가로 가상키를 정의하여 사용할 수 있으며, 추가로 정의되는 가상키는 4.2절에 정의된 "MC_GIGA_VIRUSER" 이상의 값을 사용해야 한다.

게임 개발사는 콘텐츠 개발 시 아래의 요구사항을 반드시 따라야 한다.

- ① GIGA Class3Q 이상을 지원하는 모든 게임 콘텐츠는 반드시 가상키를 지원해야 한다. 즉, 콘텐츠가 실행될 때 4.4절에 정의된 API를 이용하여 가상키 테이블을 읽어오고, handleClet()함수를 통해 키 이벤트를 전달받으면 이를 가상키 테이블을 참조하여 가상키로 변환한 후 사용해야 한다.
- ② GIGA Class3Q 이상을 지원하는 모든 게임콘텐츠는 사용자의 게임 취향에 따른 게임키 설정이 가능하도록 게임키 설정 변경 기능을 제공해야 하며, 아래 [그림-1]과 같은 UI를 제공해야 한다. 사용자가 게임키를 변경한 후 “설정” 버튼을 누르면 해당 정보를 변경된 가상키 테이블을 별도의 파일로 저장하여 관리해야 하며, “기본값” 버튼을 누르면 단말에 설정된 기본 가상키 테이블이 적용되어야 한다.



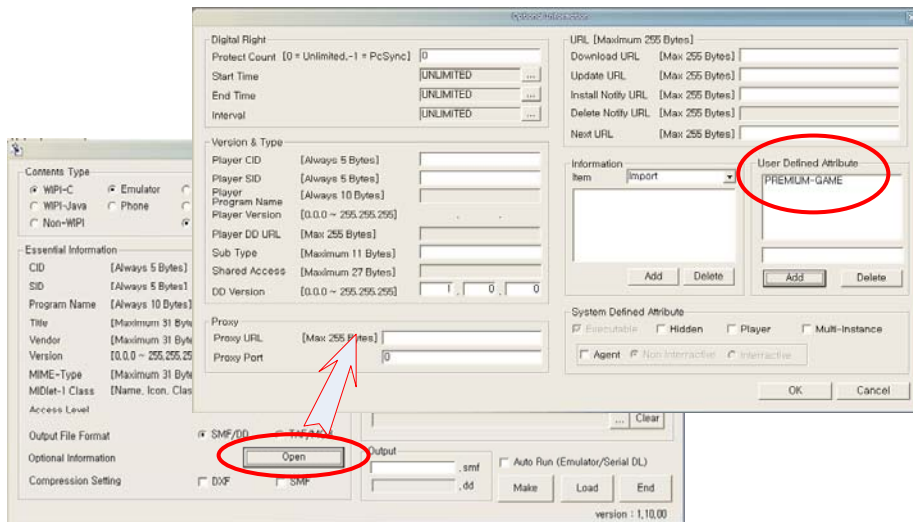
[그림-1] 게임키 설정 UI 가이드라인

4.1.2 LCD 좌표계 변환

기존 Class3 단말까지는 게임 콘텐츠를 가로 UI로 서비스 하기 위해서는, Binary Maker에서 “User Defined Attribute”에 “PREMIUM-GAME”이라는 속성을 넣어서 Binary 를 생성하고, 게임매니저가 콘텐츠 실행 시 “PREMIUM-GAME” 속성을 가진 콘텐츠의 경우에는 LCD를 가로 UI로 전환해준 후, 콘텐츠를 실행시켜주는 방식을 이용했다. 이 방식을 유지할 경우, 세로 UI를 이용하는 게임 콘텐츠를 지원해줄 수 없는 문제가 있기 때문에, 게임 콘텐츠에서 사용하고자 하는 LCD의 좌표계를 지정할 수 있는 API 가 새롭게 추가되었다.

게임 개발사는 콘텐츠 개발 시 아래의 요구사항을 반드시 따라야 한다.

- ① GIGA Class3Q 이상을 지원하는 모든 콘텐츠들은 4.4절에 정의된 API를 이 용해서 사용하고자 하는 LCD의 좌표계를 직접 지정해줘야 한다.
- ② 콘텐츠가 게임 보관함에 등록되기 위해서는 “PREMIUM-GAME” 속성이 필요 하기 때문에, 아래 [그림-2]와 같이 Binary Maker에서 “User Defined Attribute” 에 “PREMIUM-GAME”속성을 반드시 지정해 줘야 한다.



[그림-2] Binary Maker의 Option 화면

4.1.3 GIGA 모드 지정

다양한 OEM 연동 시나리오에 대해, 게임 콘텐츠에 대한 예외 처리 기능을 제공하기 위한 API가 추가되었다.

게임 개발사는 콘텐츠 개발 시 아래의 요구사항을 반드시 따라야 한다.

- ① GIGA Class3Q 이상을 지원하는 모든 콘텐츠들은 4.4절에 정의된 API를 이용해서 게임 콘텐츠가 실행 중인지의 여부를 OEM에 알려줘야 한다.
- ② 게임 콘텐츠는 startClet() 함수에서 GIGA Mode를 TRUE로 설정하고, destoryClet() 함수에서 GIGA Mode를 FALSE로 설정해야 한다.
- ③ 게임 콘텐츠는 백그라운드로 천이시 GIGA Mode를 FALSE로 설정하고, 다시 포그라운드로 천이시 GIGA Mode를 TRUE로 설정해야 한다. 즉, "MV_BACKG_EVENT", "MV_PLTBACKG_EVENT" 이벤트를 받으면 GIGA Mode를 FASLE로 설정하고, "MV_FOREG_EVENT", "MV_PLTFORG_EVENT" 이벤트를 받으면 GIGA Mode를 TRUE로 설정해야 한다.

4.2 Define 정의

Define 값	설명
MC_GIGA_VIRUP	Up
MC_GIGA_VIRDOWN	Down
MC_GIGA_VIRLEFT	Left
MC_GIGA_VIRRIGHT	Right

MC_GIGA_VIRFIRE	Select
MC_GIGA_VIRGAME_A	Virtual A
MC_GIGA_VIRGAME_B	Virtual B
MC_GIGA_VIRGAME_C	Virtual C
MC_GIGA_VIRGAME_D	Virtual D
MC_GIGA_VIRUSER	컨텐츠가 추가적으로 가상키를 정의해서 사용하는 경우, MC_GIGA_VIRUSER 이상의 값을 이용해야 한다.

4.3 Structure정의

타입	설명
VIRTUAL_KEYMAP	가상키 정보 <pre>typedef struct tagVIRTUAL_KEYMAP{ M_Int16 nKeyCode; // OEM Key Code M_Int16 nVirtualKeyCode; // Virtual Key Code } VIRTUAL_KEYMAP;</pre>

4.4 C API

API	설명
M_Int32 OEMC_gknlSetSystemOperation(M_Char *pszCmd, M_Int32 nCmdCode)	응용 프로그램의 Default System Operation을 변경한다.
M_Int32 OEMC_gknlGetSystemOperation(M_Char *pszCmd, M_Int32 *pnCmdCode)	응용 프로그램의 Default System Operation으로 설정된 값을 가져온다.
M_Int32 OEMC_gknlSetSystemProperty(M_Char *pszID, void *pBuf)	단말기에 특화된 값을 설정한다.
M_Int32 OEMC_gknlGetSystemProperty(M_Char *pszID, void *pBuf, M_Int32 *pnBufSize)	단말기에 특화된 값을 가져온다.

4.4.1 OEMC_gknlSetSystemOperation

설명

응용 프로그램의 Default System Operation을 변경한다. Default System Operation이란 플랫폼의 기본적인 동작방식을 말한다.

Default System Operation	설명	기본값
"GIGAMODE"	GIGA 컨텐츠가 실행됨을 알린다.	FALSE
"LCD_COORDINATION"	LCD 방향을 설정한다.	GIGA_LCD_VERTICAL

프로토타입

M_Int32 OEMC_gknlSetSystemOperation(M_Char *pszCmd, M_Int32 nCmdCode)

매개변수

pszCmd

[In]

변경하고자 하는 System Operation 명령어.

nCmdCode

[In]

변경하고자 하는 System Operation에 대한 명령 코드

Default System Operation	명령코드	설명
"GIGAMODE"	TRUE	GIGA 컨텐츠 실행 시작
	FALSE	GIGA 컨텐츠 실행 종료
"LCD_COORDINATION"	GIGA_LCD_VERTICAL	세로 UI 설정
	GIGA_LCD_HORIZONTAL	가로 UI 설정

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Operation 명령어

예제코드

■ "GIGAMODE"

```
// GIGA Mode 설정
nRetCode = OEMC_gknlSetSystemOperation("GIGAMODE", 1);

// GIGA Mode 해제
nRetCode = OEMC_gknlSetSystemOperation("GIGAMODE", 0);
```

■ "LCD_COORDINATION"

```
// 세로 UI 설정 (예: 240x320)
nRetCode = OEMC_gknlSetSystemOperation("LCD_COORDINATION",
                                         GIGA_LCD_VERTICAL);

// 가로 UI 설정 (예: 320x240)
nRetCode = OEMC_gknlSetSystemOperation("LCD_COORDINATION",
                                         GIGA_LCD_HORIZONTAL);
```

4.4.2 OEMC_gKniGetSystemOperation

설명

응용 프로그램의 Default System Operation에 설정된 값을 가져온다. 사용 가능한 System Operation 명령은 4.4.1 OEMC_gKniSetSystemOperation과 동일하다.

프로토타입

```
M_Int32 OEMC_gkniGetSystemOperation(M_Char *pszCmd, M_Int32 *pnCmdCode)
```

매개변수

pszCmd

[In]

가져오고자 하는 System Operation에 대한 명령어.

pnCmdCode

[out]

System Operation 설정 값.

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Operation 명령어

4.4.3 OEMC_gKniSetSystemProperty

설명

단말기에 특화된 값을 설정한다.

System Property	설명	기본값

(*) 현재 컨텐츠에서 설정할 수 있도록 정의된 System Property는 없다.

프로토타입

M_Int32 OEMC_gkniSetSystemProperty(M_Char *pszID, void *pBuf)

매개변수

pszID

[In]

설정하고자 하는 System Property ID.

pBuf

[In]

설정하고자 하는 System Property ID에 필요한 데이터

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Property ID

4.4.4 OEMC_gKniGetSystemProperty

설명

단말기에 특화된 값을 읽어온다.

프로토타입

```
M_Int32 OEMC_gkniGetSystemProperty(M_Char *pszID, void *pBuf, M_Int32*
pnBufSize)
```

매개변수

pszID

[In]

읽어오고자 하는 System Property ID.

pBuf

[in/out]

System Property가 저장될 버퍼 포인터 또는 System Property ID와 함께 추가적으로 전달해야 할 데이터가 저장된 버퍼 포인터

pnBufSize

[in/out]

컨텐츠는 System Property가 저장될 버퍼의 크기가 지정하며, 플랫폼은 저장된 데이터의 크기를 전달한다.

반환값

M_E_SUCCESS - 성공

M_E_ERROR - 실패

M_E_NOTSUP - 지원하지 않는 System Property ID

System Property ID	비고				
"GSR"	단말에 특정 GSR이 포팅 되었는지 여부를 확인한다. 포팅여부는 함수의 반환 값으로 판단한다.				
	<table border="1"> <thead> <tr> <th>아규먼트</th> <th>설명</th> </tr> </thead> <tbody> <tr> <td>pBuf</td> <td>GSR 번호 (예 : "GSR020")</td> </tr> </tbody> </table>	아규먼트	설명	pBuf	GSR 번호 (예 : "GSR020")
아규먼트	설명				
pBuf	GSR 번호 (예 : "GSR020")				

	pnBufSize	사용되지 않음.						
"VKEYTABLE"	<p>단말에 적용된 Virtual Key Table을 얻어온다. Virtual Key Table은 VIRTUAL_KEYMAP형 배열로 전달된다.</p> <p>Virtual Key Table의 크기만 읽어오기 위해서는 pBuf에 NULL을 전달하면 되며, 전달받은 크기는 Virtual Key Table의 크기(Byte)이기 때문에, Virtual Key의 개수는 (*pnBufSize/sizeof(VIRTUAL_KEYMAP))으로 계산하여 사용하도록 한다.</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <thead> <tr> <th>아규먼트</th> <th>설명</th> </tr> </thead> <tbody> <tr> <td>pBuf</td> <td>Virtual Key Table이 저장될 버퍼 포인터</td> </tr> <tr> <td>pnBufSize</td> <td>Virtual Key Table의 크기(Byte)</td> </tr> </tbody> </table>		아규먼트	설명	pBuf	Virtual Key Table이 저장될 버퍼 포인터	pnBufSize	Virtual Key Table의 크기(Byte)
아규먼트	설명							
pBuf	Virtual Key Table이 저장될 버퍼 포인터							
pnBufSize	Virtual Key Table의 크기(Byte)							

예제 코드

■ "GSR"

```
M_Int32 nRetCode;

nRetCode = OEMC_gknlGetSystemProperty("GSR", "GSR020", NULL);
if(nRetCode == M_E_SUCCESS)
{
    // GSR020이 포팅되어 있음.
    // To Do...
}else
{
    // GSR020이 포팅되어 있지 않음.
    // To Do...
}
```

■ "VKEYTABLE "

```
M_Int32 nBufLen;
M_Int32 hVKTable;
VIRTUAL_KEYMAP *pVKTable;

// Virtual Key Table의 크기를 읽어온다.
nRetCode = OEMC_gknlGetSystemProperty("VKEYTABLE", NULL, &nBufLen);

// Virtual Key Table을 저장할 버퍼를 할당한다.
hVKTable = MC_knlAlloc(nBufLen);
pVKTable = (VIRTUAL_KEYMAP *)MC_GETDPTR(hVKTable);

// Virtual Key Table을 읽어온다.
nRetCode = OEMC_gknlGetSystemProperty("VKEYTABLE", pVKTable, &nBufLen);
```

■ Virtual Key 변환처리 예제

```
VIRTUAL_KEYMAP *g_pVKTable;
M_Int32 g_nKeyMapNum;

void handleCletEvent( int type, int param1, int param2 )
```

```
{
  Int i;

  switch( type )
  {
    // Key press Event Received.
    case MV_KEY_PRESS_EVENT:
    {
      // 입력받은 키값에 맵핑된 가상키값을 얻는다.
      for(i=0; i< g_nKeyMapNum; i++)
      {
        if(param1 == g_pVKTable[i]. nKeyCode)
        {
          param1 = g_pVKTable[i]. nVirtualKeyCode;
          break;
        }
      }

      switch(param1)
      {
        case MC_GIGA_VIRGAME_A :
          // To Do...
          break;
        case MC_GIGA_VIRGAME_B :
          // To Do...
          break;
        case MC_GIGA_VIRGAME_C :
          // To Do...
          break;
        case MC_KEY_0 :
          // To Do...
          break;
      }
    }
  }
}
```