SMARTX® Framework

프로그래밍 가이드

SmartX Framework Ver. 3.2.4

업데이트 : 2021.02.01

Revision No.2(2020.06.30 기준)

HNS
www.hnsts.co.kr
(주)에이치앤에스
Hardware & Software Total Solution
Expert Embedded Total Solution

SmartX Framework
프로그래밍 가이드 활용 안내

SmartX Framework 프로그래밍 가이드의 내용이 방대하여 하나 하나 살펴보기에 많은 시간이 필요합니다. 따라서 효과적인 학습과 적용을 위한 체계적인 학습 방향을 잡을 수 있도록 아래의 요약정보를 참고하시어 적용하시기 바랍니다.

1. SmartX Framework 사용을 권장하는 이유

1) 업데이트가 중단된 .NET Compact Framework 2.0/3.5 버전과 달리 지속적인 업데이트를 지원합니다.
2) SmartX Framework를 사용하시면 더욱 폭넓고 빠른 기술 지원을 받으실 수 있습니다.
3) .NET Compact Framework 2.0/3.5 보다 쉽고 빠르게 프로그램을 개발하실 수 있습니다.

2. 개발자가 꼭 필독해야 할 내용

SmartX Framework 프로그래밍 가이드의 내용이 방대하여 효과적인 학습과 적용을 위해 두 개의 Section으로 구성했습니다. Essential Learning Section은 SmartX Framework의 적용을 체계적으로 하기 위해 개발 전 필독해야 하며, Reference Guide Section은 필요한 부분을 선택적으로 참고하여 개발에 응용하시기 바랍니다.

Essential Learning -Section 필수 권장

- Part-I. SmartX Framework 소개
- Part-II. 개발 환경 구축하기
- Part-III. 개발 시 환경 관련 TIP
- Part-IV. 개발 시 유의사항
- Part-V. 주요 권장 컴포넌트

Reference Guide -Section

- Part-VI. 하드웨어 장치 안내
- Part-VII. 하드웨어 장치 제어 컴포넌트
- Part-VIII. 사용자 인터페이스 컴포넌트
- Part-IX. 사용자 편의 컴포넌트

SmartX Framework 프로그래밍 가이드

3. 필수 사용 권장 컴포넌트

컴포넌트의 학습 순서는 중요하지 않습니다. 적용 및 필요한 기능을 확인하여 참고 및 학습하시기를 권장합니다. 다만 "Part- V. 주요 권장 컴포넌트"에서 설명하는 컴포넌트는 가장 먼저 학습하기를 바라며 프로젝트에 적용할 것을 추천드립니다.

SmartForm	SmartInnerForm	SmartButton	SmartLabel
SmartRadioButton	SmartCheckBox	SmartSplash	SmartBootLogo

필수 사용 권장 컴포넌트

4. 컴포넌트 간략 설명

※ 하드웨어 제어 관련 기능

컴포넌트	활용도	설명
SmartGPIO	★★★★★	확장 GPIO 포트 제어
SmartADC	★★★★★	ADC(Analog to Digital Converter) 제어
SmartSerialPort	★★★★★	시리얼 통신을 편리하게 처리
SmartMemory	★★★★★	시스템 RAM과 외부 Storage 등의 영역을 실시간 모니터링 및 설정
SmartBootLogo	★★★★★	부트로고를 변경할 수 있도록 지원하는 프로그램
SmartModbus	★★★★☆	Modbus 사용을 편리하게 처리
SmartModbusSlave	★★★★☆	Modbus-Slave 사용을 편리하게 처리
SmartSound	★★★★☆	사운드 관련 기능을 편리하게 제어
SmartDAC	★★★☆☆	DAC(Digital to Analog Convert) 제어
SmartPWM	★★★☆☆	PWM 출력을 편리하게 제어
SmartInputCounter	★★☆☆☆	SmartGPIO의 입력 속도 대비 고속으로 입력을 처리
SmartWatchDog	★★☆☆☆	응용 프로그램에서 처리되지 않은 예외 발생을 감지 및 처리
SmartVideo	★★☆☆☆	별도의 옵션 제품인 SmartVideo을 사용하여 카메라 기능을 지원
SmartIIC	★★☆☆☆	IIC(Inter-Integrated Circuit) 통신을 편리하게 처리
SmartPrint	★☆☆☆☆	인쇄 관련 기능을 편리하게 처리
SmartBattery	★☆☆☆☆	외부 전원 공급 차단 시 시스템의 데이터 및 상태 Backup 기능 지원

※ 사용자 인터페이스(UI) 관련 기능

컴포넌트	활용도	설명
SmartForm	★★★★★	MDI 화면 구성을 편리하게 구현하도록 지원
SmartInnerForm	★★★★★	여러가지(MDI) 프로그램의 성능 및 메모리의 최적화 처리를 지원
SmartDraw	★★★★★	도형(이미지, 선, 호, 글자) 등을 쉽게 그리도록 지원
SmartButton	★★★★★	기존 버튼(Button)의 확장 기능을 지원
SmartLabel	★★★★★	기존 라벨(Label)의 확장 기능을 지원
SmartCheckBox	★★★★★	기존 체크박스(Checkbox)의 확장 기능을 지원
SmartRadioButton	★★★★★	기존 라디오버튼(RadioButton)의 확장 기능을 지원
SmartListBox	★★★★★	기존 리스트박스(Listbox)의 확장 기능을 지원
SmartProgressBar	★★★★☆	기존 프로그레스바(ProgressBar)의 확장 기능을 지원
SmartKeyboard	★★★★☆	가상 키보드(InputPanel)의 단점을 보완한 가상 키보드 패널을 지원
SmartKeyPad	★★★★☆	키보드 레이아웃, 이미지, 키의 위치/개수, 전체 사이즈 등의 변경을 지원
SmartTrackBar	★★★★☆	기존 트랙바(TrackBar)의 확장 기능을 지원
SmartComboBox	★★★★☆	기존 콤보박스(ComboBox)의 확장 기능을 지원
SmartUpDown	★★★★☆	기존 NumericUpDown의 확장 기능을 지원
SmartGroupBox	★★★★☆	.Net Compact Framework에서 지원되지 않는 GroupBox 컨트롤을 지원
SmartMessageBox	★★★★☆	기존 MessageBox의 확장 기능을 지원
SmartSeperatorLine	★★★★☆	폼에 구분선을 직접 그리는 기능을 지원
SmartMonthCalendar	★★★☆☆	일정(스케줄) 표시 및 관리를 편리하게 사용하는 기능을 지원

※ 사용자 편의 기능

컴포넌트	활용도	설명
SmartTCPMultiServer	★★★★★	다중 접속 TCP Socket 서버를 편리하게 구현하도록 지원
SmartTCPClient	★★★★★	TCP Socket 클라이언트를 편리하게 구현하도록 지원
SmartConfigs	★★★★★	IEC-Series 장치의 다양한 환경설정 및 특수 기능을 제어
SmartSplash	★★★★★	응용 프로그램 로딩 시 보여지는 로딩 상태 이미지 표시 기능을 지원
SmartRemote	★★★★★	IEC-Series의 원격제어 솔루션을 지원
SmartTimer	★★★★☆	기존 Timer 기능 확장. 고정밀 특정 구간의 시간 측정 및 표시 기능을 지원
SmartFile	★★★★☆	파일 읽기/쓰기 처리 시 복잡한 파일 처리의 간소화 기능을 지원
SmartUpdate	★★★☆☆	장치 응용 프로그램을 간편하게 업데이트 하는 기능을 지원
SmartLock	★★★☆☆	개발 프로그램의 불법 복제 방지 기능을 지원
SmartFileSetting	★★★☆☆	제품 양산 시 IEC-Series를 업체에 맞게 일괄 Setting하는 프로그램
SmartThread	★★★☆☆	Thread 생성 및 제어를 쉽게 처리
SmartFTP	★★★☆☆	IEC-Series에서 FTP Client의 기능들을 편리하게 처리
SmartScreenSaver	★★☆☆☆	IEC-Series에서 스크린세이버 기능을 편리하게 사용하도록 지원
SmartPlayer	★★☆☆☆	동영상 Play 기능을 처리(O/S Proffessional 지원)
SmartLaunch	★☆☆☆☆	IEC-Series의 바탕화면에 바로가기를 생성해주는 프로그램

주의 각 컴포넌트별 "프로그래밍 적용 가이드"에서 사용되는 코드는 SmartX 네임 스페이스를 포함하였지만 인터페이스 설명의 C#, VB 사용법에는 가독성을 위해 제외했습니다. C#, VB 사용법을 Copy & Paste 하실 경우 유의하여 사용하시기 바랍니다.

5. 이미지 사용 시 주의사항

IEC-Series로 프로그램을 개발하시는 경우 프로그램의 안정적인 동작을 위하여 아래 주의사항을 반드시 준수해 주시기 바랍니다.

※ 이미지 제작 가이드 ※

필수 이미지 저장 시 반드시 포토샵으로 사용하여 저장합니다.

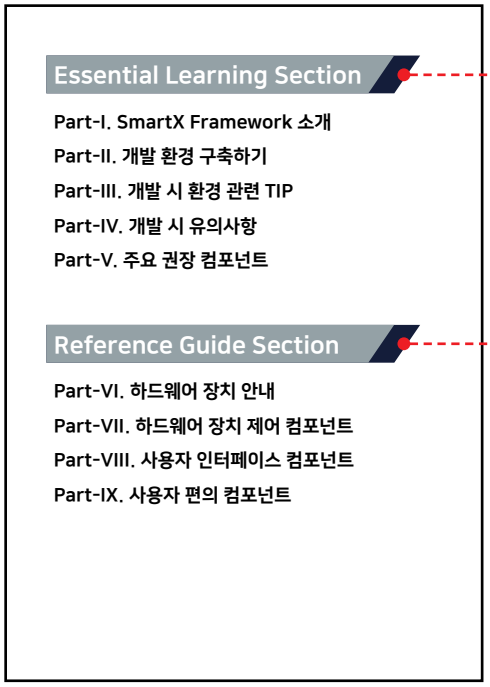
필수 이미지 해상도는 96dpi로 저장해야 합니다.

참고 모든 이미지들은 png 형식을 사용할 것을 권장합니다.

참고 [Essential Learning Section] - [Part-IV. 개발 시 유의사항] - [1. 이미지 관련 주의 사항]

이 책의 구성

이 책은 HNS에서 자체적으로 만든 SmartX Framework를 누구나 쉽게 익힐 수 있도록 다양한 그림과 소스 코드, 인터페이스 설명 등을 곳곳에 만들어 두었습니다. 또한 각 챕터의 내용이 끝나면 예제를 활용하여 본문에서 설명하는 컴포넌트의 사용법을 익힐 수 있습니다.



Essential Learning Section

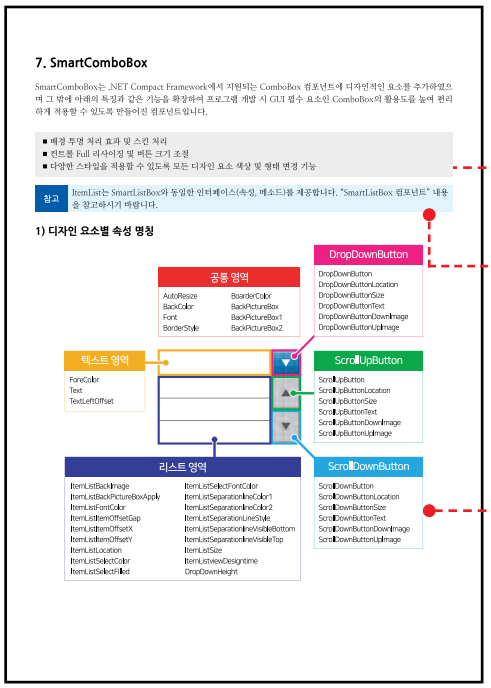
SmartX Framework의 적용을 체계적으로 할 수 있게 구성된 Section으로 개발 전 필독하시기 바랍니다.

- **Part - I~IV** : SmartX Framework에 대한 정보 및 개발 환경에 관한 Part들로 구성되어 있습니다.
- **Part - V** : 프로젝트에 적용을 권장하는 주요 컴포넌트들로 구성되어 있습니다.

Reference Guide Section

필요한 부분을 선택적으로 참고하여 개발에 응용할 수 있는 Section입니다.

- **Part - VI** : 하드웨어 장치의 콘넥터 위치 및 핀 기능에 대한 내용들로 구성되어 있습니다.
- **Part - VII** : 하드웨어를 제어할 수 있는 컴포넌트들로 구성되어 있습니다.
- **Part - VIII** : 사용자 인터페이스를 쉽고 편리하게 개발할 수 있는 컴포넌트들로 구성되어 있습니다.
- **Part - IX** : 장치 응용 프로그램 개발 시 유용하게 사용할 수 있는 컴포넌트들로 구성되어 있습니다.



특징

컴포넌트의 특징을 한눈에 보이도록 합니다.

참고/ 참조/ 주의/ 권장

해당 내용을 학습할 때 보충할 내용을 짚어줍니다.

그림으로 보는 디자인 요소

사용자 인터페이스 컴포넌트인 경우 디자인 요소를 영역으로 분류하여 설명합니다.

5) 프로그래밍 적용 가이드

CASE-1 출력 기능 사용하기

SmartGPIO를 출력 기능으로 사용하기 위해서는 먼저 출력으로 사용할 Pin의 방향을 PORTXDIR 속성을 사용해 출력(Output)으로 설정해야 합니다. 방향 설정이 완료되면 PORTXDATA 속성으로 Pin의 상태를 HIGH 또는 LOW로 설정할 수 있습니다.

※ 자세한 내용은 "PORTXDIR, PORTXDIRS, PORTXDATA, PORTXDATAS 속성"을 참고하시기 바랍니다.

```
smartGPIO1.PORTADIR0 =SmartGPIO_PORTDIRS.OUTPUT;    // Port-A0 Pin을 출력으로 설정
smartGPIO1.PORTBDIRS = 0xFF;                        // Port-B의 모든 Pin을 출력으로 설정
smartGPIO1.PORTADATA0 = true;                       // Port-A0 Pin을 HIGH로 설정
smartGPIO1.PORTBDATAS = 0xFF;                       // Port-B의 모든 Pin을 HIGH로 설정
System.Threading.Thread.Sleep(1000);                // 1초(1000ms) 대기
smartGPIO1.PORTADATA0 = false;                      // Port-A0 Pin을 LOW로 설정
smartGPIO1.PORTBDATAS = 0x00;                       // Port-B의 모든 Pin을 LOW로 설정
```

프로그래밍 적용 가이드

사용자가 해당 컴포넌트를 프로젝트에 쉽게 적용하기 위한 과정을 소스 코드 위주로 자세히 설명합니다.

해당 소스 코드에는 SmartX 네임 스페이스가 모두 포함되어 있기 때문에 추가 과정없이 Copy & Paste하여 바로 사용할 수 있습니다.

3) SmartLabel 인터페이스 설명

SmartLabel Component Interface

속성		
BackColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
BackPictureBox2 : SmartGroupBox	BorderColor : Color	BorderStyle : BorderStyle
ForeColor : Color	InitVisible : bool	Text : string
TextHAlign : SmartLabel.TextHorAlign	TextVAlign : SmartLabel.TextVerAlign	WordWrap : bool
메소드		
SetDoubleClickEnable(bool Enable) : void		
이벤트		
Click : EventHandler	DoubleClick : EventHandler	

인터페이스 설명

컴포넌트의 인터페이스를 속성, 메소드, 이벤트 별로 분류했으며, 각 인터페이스와 사용법을 설명합니다.

해당 C#, VB 소스 코드에는 가독성을 위해 SmartX 네임 스페이스가 제외되었습니다. 소스 코드를 Copy & Paste 하여 사용하실 경우 아래의 방법 중 하나를 반드시 포함하여 소스 코드를 수정한 뒤 사용하시기 바랍니다.

[방법-1]	소스 코드에 SmartX를 추가하여 사용

```
SmartX.SmartSplash.Finished();
```

[방법-2]	소스 코드 최상단에 SmartX 네임 스페이스를 선언하여 사용

- C# 사용하는 경우 : `using SmartX;`
- VB 사용하는 경우 : `Imports SmartX;`

메소드(함수)	Finished

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다.

스플래쉬 이미지(애니메이션 GIF)의 출력을 종료합니다. 반드시 객체를 생성해야 사용이 가능했던 기존 방식을 개선한 Static 메소드로 사용을 권장하며, 기존 Finish() 메소드와 동일하게 동작합니다.

권장	Finished() 정적 메소드 사용을 권장합니다.
중요	장치 응용프로그램이 실행 완료하여 화면에 출력되는 시점에 Finished() 정적 메소드를 호출하여 스플래쉬를 종료하시기 바랍니다. 폼 로드 이벤트에서 처리할 내용이 많은 경우 Finish() 정적 메서드의 호출 위치를 변경하여 최적의 상태를 찾아 호출하시기 바랍니다.

```
static void Finished()
```

사용법	
참고	Start() 정적 메소드의 사용법을 참고하시기 바랍니다.

4) SmartComboBox 예제 사용하기

SmartComboBox를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제파일 다운로드 → "SmartComboBox"

예제 경로

본문에서 설명하는 컴포넌트를 프로젝트에 적용하기 위한 참고 소스를 제공합니다.

결과 이미지

예제를 실행한 후 제품에 출력되는 화면을 나타냅니다.

SmartX Framework

| SmartX Framework 시스템 요구사항

- **개발 PC 권장 운영 체제 :** Windows 7, 8, 8.1, 10
- **개발 Tool :** Visual Studio 2008 Pro. 또는 Team Edition
- **Download&Debugging :** Windows Mobile Device Center
- **대상 장치 :** IEC-Series(IEC266-Series, IEC667-Series, IEC1000-Series)
- **지원 개발 언어 :** C#(Full 지원), Basic(Full 지원), C++(하드웨어 제어 기능만 지원)
  언어별 지원 컴포넌트 표 참고(Part-I → 3. 개발 언어별 SmartX Framework 지원사항)

※ .Net Compact Framework 2.0, .Net Compact Framework 3.5(권장)

| 예제 및 기술지원 안내

1. SmartX Framework 관련 예제의 사용에 앞서 관련 문제가 발생하실 경우 Part-II. 개발 환경 구축하기 → 3. SmartX Framework 참조 ERROR 문제 해결 방법을 참고하시기 바랍니다.

2. SmartX Framework를 사용하여 프로그램 개발이 완료되어 Run 폴더에 배포하여 Runtime 모드로 동작해야 하는 경우 실행파일(EXE)과 함께 SmartX 관련 DLL(SmartX_IECXXX.dll, SmartXCommon.dll, SmartXCommonExt.dll)파일들도 함께 복사하시기 바랍니다.

3. 그 밖의 SmartX Framework 관련 문의 사항 및 기술지원은 회사 홈페이지 Q&A 및 전화 기술문의와 원격기술지원을 이용하여 주시기 바랍니다.

목차

[Memo]

Section
Essential Learning

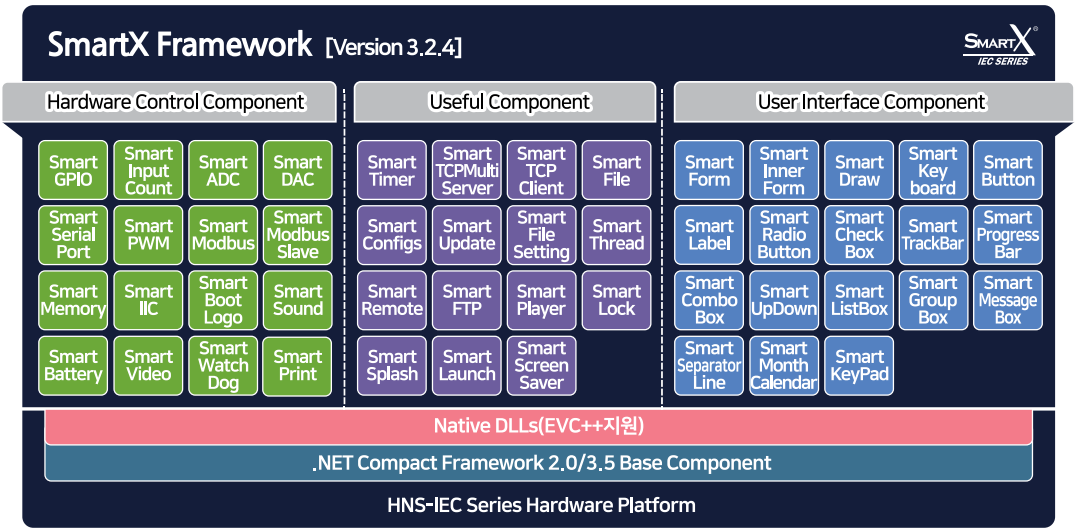
SmartX Framework 프로그래밍 가이드의 내용이 방대하여 효과적인 학습과 적용을 위해 두 개의 Section으로 구성했습니다.
Essential Learning Section은 개발 전 필독하여 SmartX Framework의 적용을 체계적으로 하시기 바랍니다.

Essential Learning Section

Part- Ⅰ. SmartX Framework 소개

1. SmartX Framework란?

SmartX Framework는 HNS에서 독자적으로 개발한 .NET Compact Framework 기반의 임베디드 컴포넌트(라이브러리)의 집합을 통칭하여 SmartX Framework라 합니다. 장치 응용 프로그램 개발 시 SmartX Framework를 적용할 경우 IEC-Series의 하드웨어 제어 및 사용자 인터페이스의 구현을 쉽게 직관적으로 처리할 수 있습니다.



SmartX Framework는 다양한 기능들로 구성되어 있습니다. 크게 하드웨어 제어 기능(Hardware Control Component), 사용자 인터페이스 기능(User Interface Component), 사용자 편의 기능(Useful Component) 총 3가지 기능으로 나누어져 있습니다.

하드웨어 제어에 따른 펌웨어 및 디바이스 드라이버의 개발 없이 하드웨어를 제어할 수 있는 환경을 제공하며 간편하게 이미지 설정 및 적용만으로 완성도 높은 사용자 인터페이스(User Interface)를 쉽고 편리하게 개발할 수 있습니다.

SmartX Framework 솔루션 운영정책

- 지속적인 업데이트 및 기능 개선
- 빠르고 섬세한 기술지원
- 기술자료 및 레퍼런스 자료 제공
- 무상 지원 솔루션

SmartX Framework의 사용자 인터페이스 디자인 구현방식

SmartX Framework 사용자 인터페이스 관련 컴포넌트들은 대부분 디자인 템플릿이 준비되어 있어야 사용할 수 있도록 설계되어 있습니다.

사용자 인터페이스 구성에 따른 프로그램을 구현하기 위해서는 프로그래머가 사용자 인터페이스에 적용할 수 있는 이미지가 제공되어야 합니다. 여기서 말하는 제공되는 이미지는 디자이너에 의해 만들어진 사용자 인터페이스에 적용될 요소별 이미지들을 말합니다. 개발자는 준비된 이미지를 이용하여 완성도 높은 사용자 인터페이스를 지원하는 프로그램을 편리하게 제작할 수 있습니다.

SmartX Framework와 연동할 수 있는 다양한 옵션 제품

Smart-KIT를 구입하시면 SmartX Framework의 하드웨어 제어 기능(Hardware Control Component)을 편리하게 테스트 및 스터디를 할 수 있습니다.

Smart I/O – Series 제품은 IEC-Series와 연동하여 산업 환경에서 외부 장치들과 인터페이스를 편리하게 적용할 수 있는 솔루션이며 기존의 HMI(Human Machine Interface)와 PLC(Programmable Logic Controller)의 기능을 하나의 제품으로 구성할 수 있는 옵션 제품입니다.

그 밖의 다양한 옵션 제품들이 준비되어 있으며 SmartX Framework 옵션 제품의 활용을 편리하게 도와줍니다.

참조 옵션 제품 관련 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 옵션 제품 관련"을 참조하시기 바랍니다.

주의 SmartX Framework는 HNS에서 제작된 하드웨어(IEC-Series)에서만 정상 동작되며, 그 외의 하드웨어에서 사용할 경우 발생되는 문제는 당사가 책임을 지지 않습니다.

2. SmartX Framework 사용을 권장하는 이유

1) 업데이트가 중단된 .NET Compact Framework 2.0/3.5 버전과 달리 지속적인 업데이트를 지원합니다.
2) SmartX Framework를 사용하시면 더욱 폭넓고 빠른 기술 지원을 받으실 수 있습니다.
3) .NET Compact Framework 2.0/3.5 보다 쉽고 빠르게 프로그램을 개발하실 수 있습니다.

3. 개발 언어별 SmartX Framework 지원사항

※ SmartX Framework 2021년 1월 현재 기준

1) Hardware Control Class

Component	C++	C#	Basic
SmartGPIO	○	○	○
SmartADC	○	○	○
SmartSerialPort	×	○	○
SmartMemory	×	○	○
SmartBootLogo	별도 App	별도 App	별도 App
SmartModbus	×	○	○
SmartModbusSlave	×	○	○
SmartSound	○	○	○
SmartDAC	○	○	○
SmartPWM	○	○	○
SmartInputCounter	×	○	○
SmartWatchDog	×	○	○
SmartVideo	○	○	○
SmartIIC	○	○	○
SmartPrint	○	○	○
SmartBattery	○	○	○

2) User Interface Component Class

Component	C++	C#	Basic
SmartForm	×	○	○
SmartInnerForm	×	○	○
SmartDraw	×	○	○
SmartButton	×	○	○
SmartLabel	×	○	○
SmartCheckBox	×	○	○
SmartRadioButton	×	○	○
SmartListBox	×	○	○
SmartProgressBar	×	○	○
SmartKeyboard	×	○	○
SmartKeyPad	×	○	○
SmartTrackBar	×	○	○
SmartComboBox	×	○	○
SmartUpDown	×	○	○
SmartGroupBox	×	○	○
SmartMessageBox	×	○	○
SmartSeparatorLine	×	○	○
SmartMonthCalendar	×	○	○

3) Useful Component Class

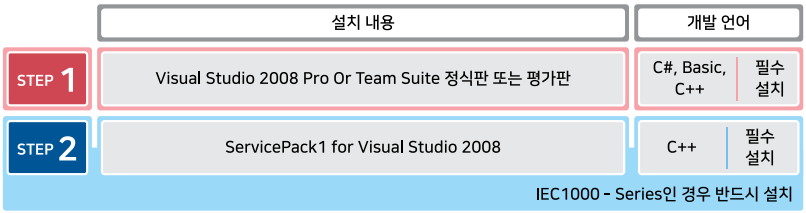
Component	C++	C#	Basic
SmartTCPMultiServer	×	○	○
SmartTCPClient	×	○	○
SmartConfigs	×	○	○
SmartSplash	×	○	○
SmartRemote	지원 예정	○	○
SmartTimer	×	○	○
SmartFile	×	○	○
SmartUpdate	○	○	○
SmartLock	○	○	○
SmartFileSetting	별도 App	별도 App	별도 App
SmartThread	×	○	○
SmartFTP	○	○	○
SmartScreenSaver	×	○	○
SmartPlayer	○	○	○
SmartLaunch	별도 App	별도 App	별도 App

참고 SmartX Framework은 지속적인 업데이트를 하고 있습니다.

Part-Ⅱ. 개발 환경 구축하기

1. 사전 설치 개발 Tools

IEC-Series 장치 응용 프로그램 개발 시 Visual Studio 2008 Tool만을 사용합니다. IEC-Series는 Windows CE 5.0/6.0을 탑재하고 있으며 Visual Studio 2008은 Windows CE 장치 응용 프로그램을 지원하고 있습니다. 개발 언어에 따라 설치해야 하는 것이 다르므로 아래의 STEP을 참고하시기 바랍니다.



[STEP-1] Visual Studio 2008 설치하기

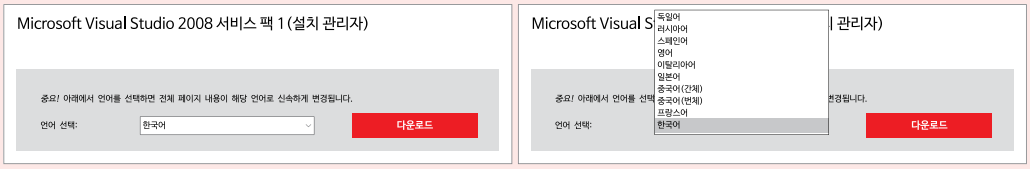
Visual Studio 2008 Professional Edition 또는 Visual Studio 2008 Team Suite 개발 Tool이 설치되어 있어야 합니다.
※ Visual Studio 2008 구매 안내 ▶ 홈페이지(www.hnsts.co.kr) → 소프트웨어 → 개발 툴

[STEP-2] Service Pack 1 for Visual Studio 2008 설치 확인하기

Service Pack 1의 설치 여부와 Visual Studio 2008 Version을 확인하시려면 아래 이미지를 참고하시기 바랍니다.
만약 설치가 안 되어 있다면 Microsoft Download Center에서 다운로드받아 설치합니다.

주의 Service Pack 1 설치 시 주의사항

Service Pack 1 언어 선택 시 설치된 Visual Studio 2008 언어와 동일하게 다운로드받아 설치하시기 바랍니다.



참조 STEP1, STEP2 설치 방법

※ Visual Studio 2008 설치 방법
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 관련 → 개발 환경 구축 가이드 → 개발 환경 설정 → Visual Studio 2008 설치

※ ServicePack1 for Visual Studio 2008 설치 방법
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 관련 → 개발 환경 구축 가이드 → 개발 환경 설정 → ServicePack1 for Visual Studio 2008 설치하기

주의 반드시 비주얼 스튜디오를 관리자 권한으로 실행합니다.

비주얼 스튜디오를 관리자 권한으로 실행하지 않을 시 다음과 같은 문제점이 발생할 수 있습니다.
- 디버깅이 안 될 수 있습니다.

- 사용 권한이 없다는 메시지 창이 뜰 수 있습니다.
- 디버깅 시 액세스가 거부되어 실행이 안 될 수 있습니다.
- 보안상 절대 경로에 프로젝트를 저장하거나 파일 생성이 안 됩니다.

따라서 반드시 비주얼 스튜디오 실행 시 [관리자 권한]을 주어 안정적으로 실행하시기 바랍니다.

그 밖의 다른 설치 항목들은 SmartX Framework 설치 시 추가로 설치하실 수 있습니다.

STEP 3	SmartX Framework 설치	C#, Basic, C++	필수 설치

파일 위치	홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 관련 → 4. SmartX Framework 설치파일

※ SmartX Framework 3.2.1 이후 버전을 다운로드 시 개발 환경 자동 설치 기능을 지원하기 때문에 개발 환경 구축 및 관련 파일이 설치되어 Window 10 버전 업데이트 이후 발생하는 다양한 오류를 해결 가능합니다.

참 고	자세한 내용은 "SmartX Framework 설치하기"의 [STEP-3]를 확인하시기 바랍니다.

1. IEC-Series SDK 설치 확인 및 진행

파일 위치	홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → 6. IEC-Series SDK 설치파일
주 의	IEC1000-Series일 때 반드시 설치 합니다.

2. Windows Mobile Device Center

3. Windows Mobile 2003 기반 장치 연결 서비스 시작 (SmartX 설치 시 자동으로 설정됨)

4. .NET Compact Framework Target 파일 설치

파일 위치	홈페이지(www.hnsts.co.kr) → 커뮤니티 → 공지사항 → 99. Windows 10의 Windows 버전 업데이트 관련

5. SmartX Framework 설치

2. SmartX Framework 설치하기

SmartX Framework를 설치하기에 앞서 설치 파일을 다운로드합니다. 개발 환경에 맞는 설치가 필요하므로 아래 표를 참고하여 설치하시기 바랍니다.

홈페이지 www.hnsts.co.kr → 자료실 → SmartX Framework 관련 → SmartX Framework 설치 파일

.NET Compact Framework 2.0인 경우		권장 .NET Compact Framework 3.5인 경우	
.NET Compact Framework 2.0 Base SmartX 설치 파일		.NET Compact Framework 3.5 Base SmartX 설치 파일	
지원 장치	IEC266-Series / IEC667-Series	지원 장치	IEC667-Series / IEC1000-Series
지원 툴	Visual Studio 2005 / 2008	지원 툴	Visual Studio 2008
.NET Compact Framework 2.0 Base SmartX Framework 설치 파일		.NET Compact Framework 3.5 Base SmartX Framework 설치 파일	

참고 SmartX Framework 2.8.0 미만 Version의 경우 SmartX의 Version을 모델별로 설치할 수 없었으나, 2.8.0 이상 Version은 SmartX의 Version을 모델별로 독립적으로 설치가 가능하여 다양한 SmartX Framework Version의 설치 및 사용이 보다 편리합니다.

참고 SmartX Framework Version Major / Minor / Release 각 Version에 대한 설명

★ 본 내용은 2.8.0 버전부터 적용

3.2.4
① ② ③

① Major Version	SmartX Framework의 구조 변경 또는 신규 컴포넌트가 동시에 여러 개 추가될 경우 변경
② Minor Version	신규 컴포넌트 추가 또는 구조 변경될 경우 변경
③ Release Version	SmartX Framework의 기능 추가, 버그 및 기능 개선될 경우 변경

[STEP-1] SmartX Framework 설치 프로그램 실행

[STEP-2] 대상 장치 선택

[STEP-3] 개발 프로그램 이미지 제작 가이드 안내 및 개발 환경 자동 설치 확인

SmartX Framework 설치 프로그램은 다음과 같이 2가지의 기능을 설치할 수 있습니다.

- 장치 응용 프로그램을 개발하기 위한 설정 및 설치
- SmartX Framework 설치

SmartX Framework 설치 프로그램을 사용하는 상황에 따라 설치해야 하는 기능이 달라집니다. 따라서 아래의 표를 참고하여 필요한 기능을 알맞게 선택 후 설치하시기 바랍니다.

		개발 환경 자동 설치	SmartX Framework 설치
CASE-1	- SmartX Framework를 처음 설치 - O/S 업데이트 이후 - Visual Studio 새로 설치	☑	☑
CASE-2	SmartX Framework 업데이트	☐	☑

권장 SmartX Framework를 사용하지 않는 경우에도 개발 환경 자동 설치를 설치하시기를 권장합니다.

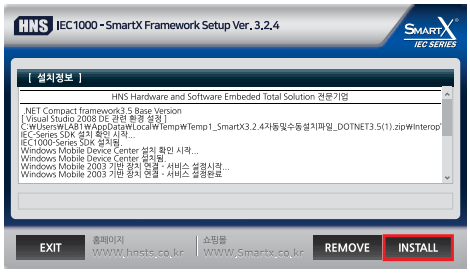
개발 환경 자동 설치[설치 권장]

- IEC-Series SDK 설치 확인 및 진행
- Windows Mobile Device Center 설치 확인 및 진행
- Windows Mobile 2003 기반 장치 연결 서비스 시작
- .NET Compact Framework Target 파일 설치
- SmartX Framework 설치

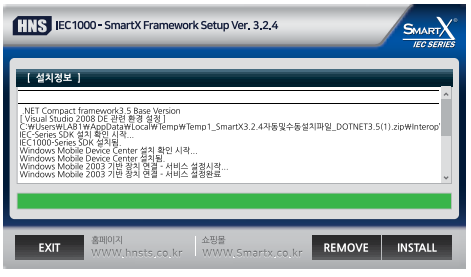
SmartX Framework 설치

- SmartX Framework 설치

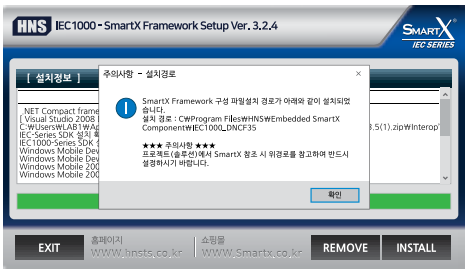
[STEP-4] [INSTALL] 클릭



[STEP-5] 설치 진행



[STEP-6] 설치 경로 표시 및 설치 완료



참고 설치가 정상적으로 되지 않을 경우 실행 중인 모든 프로그램을 종료 및 컴퓨터를 재부팅한 뒤, 다시 설치하시기 바라며 다음으로 이어지는 "SmartX Framework 수동 설치하기"를 참고하시기 바랍니다.

Visual Studio 2005 / 2008 개발 Tool을 실행한 다음 새로운 장치 프로젝트(솔루션)를 생성하시면 설치된 대상 장치에 따라 각각 HNS-SmartX for IEC266", "HNS-SmartX for IEC667", "HNS-SmartX for IEC1000"이 Toolbox에 추가된 것을 확인하실 수 있습니다.

1) SmartX Framework 수동 설치하기

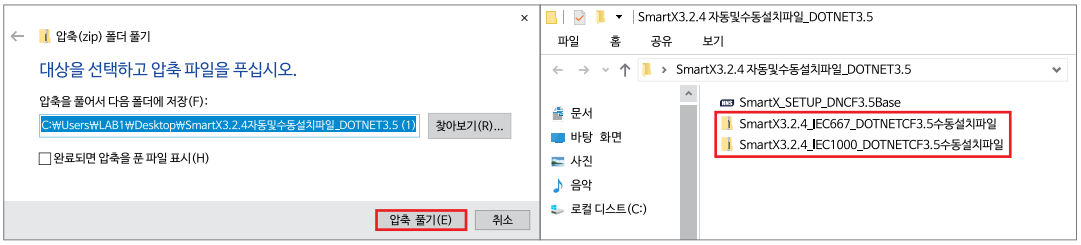
수동 설치를 하실 경우 홈페이지에서 SmartX Framework 설치 파일을 다운로드합니다. 개발 환경에 맞는 파일을 설치하시기 바랍니다.

참고 개발 환경에 맞는 파일을 다운로드하시려면 "SmartX Framework 설치하기"의 표를 참고하시기 바랍니다.

※ 본 STEP은 IEC1000-Series 기준으로 작성되었습니다.

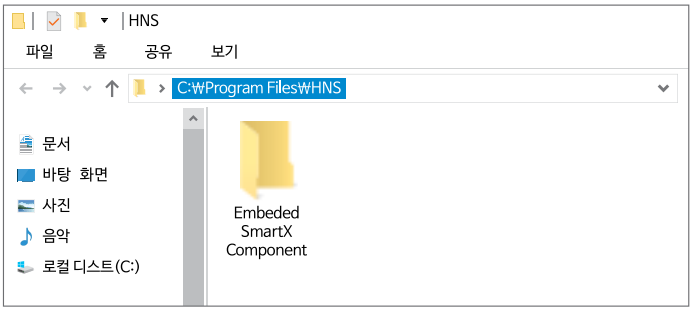
[STEP-1] 다운로드 받은 파일 압축 풀기

수동 설치 파일은 다운로드 받은 SmartX Framework 설치 파일 폴더 내부에 있습니다.



[STEP-2] SmartX Framework 파일을 설치할 폴더 만들기

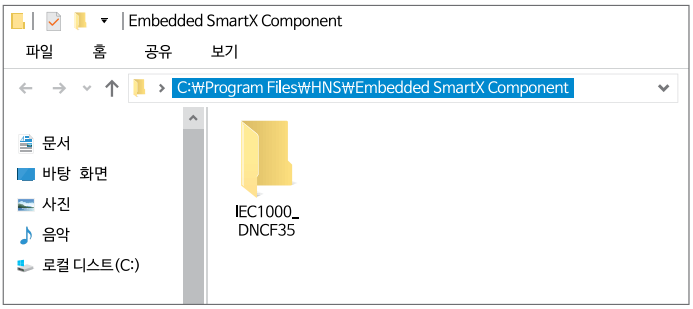
SmartX Framework 파일의 설치 경로는 "C:₩Program Files₩HNS₩Embedded SmartX Component"이어야 하므로 Program Files 폴더 안에 HNS 폴더를 만든 뒤 Embedded SmartX Component 폴더도 만들어줍니다.
(앞의 드라이브는 설치된 하드 디스크 드라이브에 따라 달라질 수 있습니다.)



[STEP-3] DLL 파일을 담을 폴더 만들기

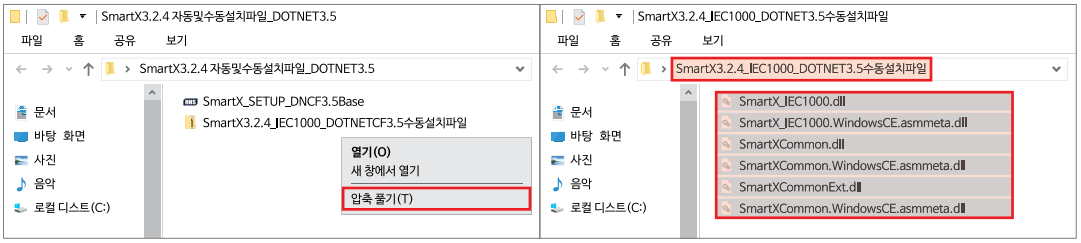
[STEP-2]에서 만든 Embedded SmartX Component 폴더 안에 IEC1000_DNCF35 폴더를 만들어줍니다. 개발 환경과 대상 장치에 따라 폴더명이 다르므로 아래를 참고하시기 바랍니다.

참고 DLL 폴더명은 "제품 및 .NET Compact Framework Version에 따른 DLL 파일 경로"를 참고하시기 바랍니다.



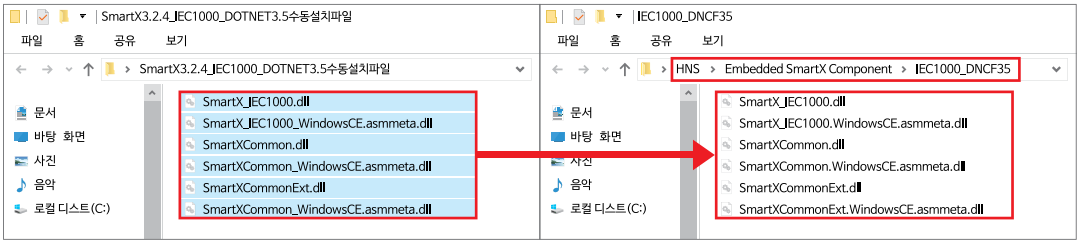
[STEP-4] 개발 환경과 대상 장치에 따른 폴더 압축 풀기

[STEP-1]에서 설명한 수동 설치 파일의 압축을 풀어줍니다. 개발 환경과 대상 장치를 확인하시고 압축을 풀기 바랍니다.



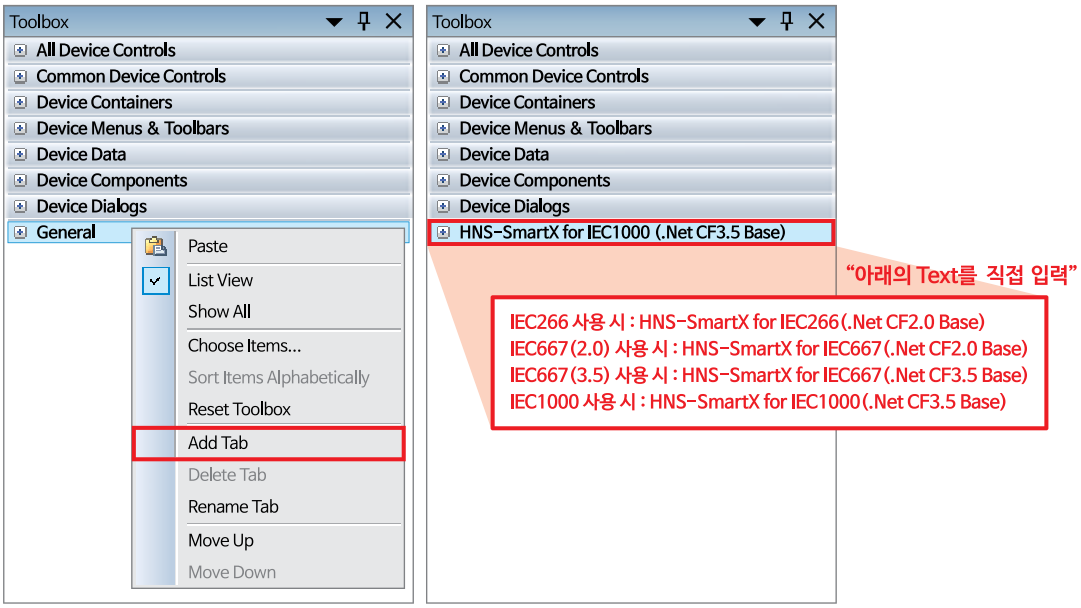
[STEP-5] SmartX 관련 DLL 파일 옮기기

DLL 파일을 [STEP-3]에서 만든 IEC1000_DNCF35 폴더 안에 복사 붙여넣기합니다.



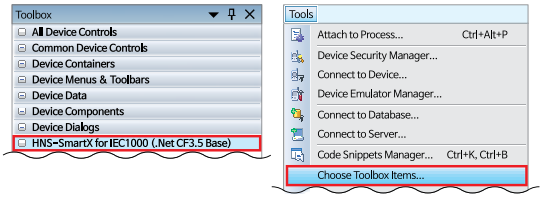
[STEP-6] Visual Studio 2008 실행 → [도구 상자] → 마우스 오른쪽 버튼 클릭 → [탭 추가] 클릭 → SmartX 탭 생성

[탭 추가]를 클릭하시면 탭 이름을 직접 입력할 수 있습니다. 아래의 박스를 참고하여 작성하시기 바랍니다.

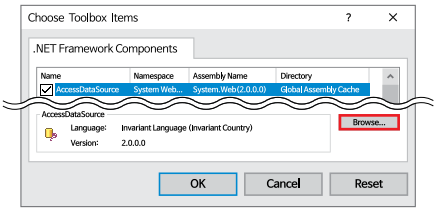


[STEP-7] 만든 탭을 선택한 상태로 [도구] → [도구 상자 항목 선택] 클릭

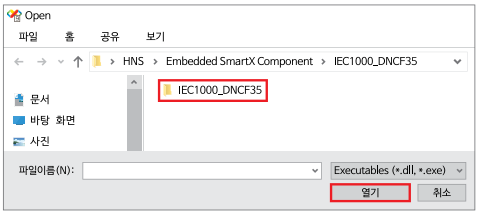
만드신 탭을 클릭하여 선택한 상태로 진행해야 만든 탭 안에 SmartX Component가 추가됩니다.



[STEP-8] [.NET Framework 구성 요소] → [찾아보기] 클릭

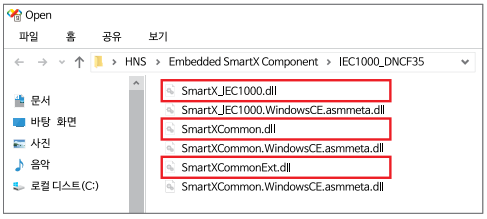


[STEP-9] 개발 대상 장치에 따른 DLL 폴더를 선택 → [열기] 클릭

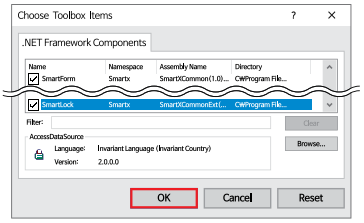


참고 개발 대상 장치에 따른 DLL 파일 경로는 “제품 및 .NET Compact Framework Version에 따른 DLL 파일 경로”를 참고하시기 바랍니다.

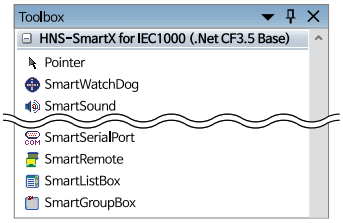
[STEP-10] SmartX_IEC1000.dll, SmartXCommon.dll, SmartXCommonExt.dll 파일 선택 → [열기] 클릭



[STEP-11] [확인] 클릭 → SmartX Component 수동 설치



[STEP-12] 도구 상자 확인

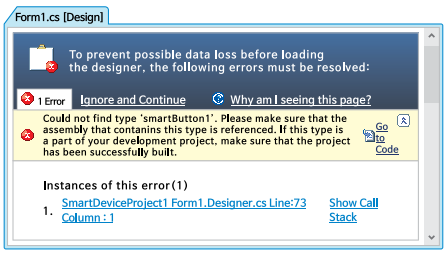


참고 “DLL 파일별 포함되는 컴포넌트”를 참고하시기 바랍니다.

3. SmartX Framework 참조 ERROR 문제 해결 방법

프로젝트 파일을 열었을 때, 참조 문제로 ERROR가 발생할 수 있습니다. 이럴 경우 참조를 다시 설정하시기 바랍니다.

참조 에러 발생되는 모습



프로젝트 파일을 열기(Open)할 때 Form 디자인 화면에서 오류 메시지

프로젝트 파일 배포 시 IEC-Series에서의 오류 메시지

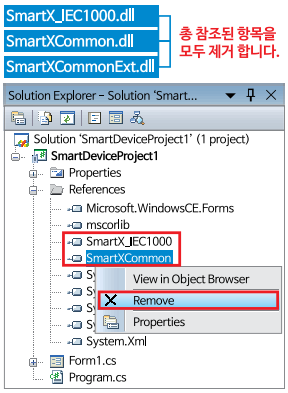
1) 문제 발생 원인

CASE	유형
CASE-1	대상 장치가 변경되어 제품 간 참조가 변경된 경우 (ex : IEC667-Series → IEC1000-Series)
CASE-2	개발 PC가 변경되어 SmartX DLL 파일의 경로가 변경된 경우 (ex : Windows 10 OS 32bit 운영체제로 개발 → Windows 10 OS 64bit 운영체제로 변경)
CASE-3	참조 파일이 해당 경로에 없거나 잘못된 참조 파일을 참조하는 경우
CASE-4	SmartX Framework 업데이트했을 경우 (ex : SmartX Framework 3.2.4 → SmartX Framework 3.2.4 이후 버전)
CASE-5	프로젝트 파일이 복사되어 다른 PC에서 사용할 경우

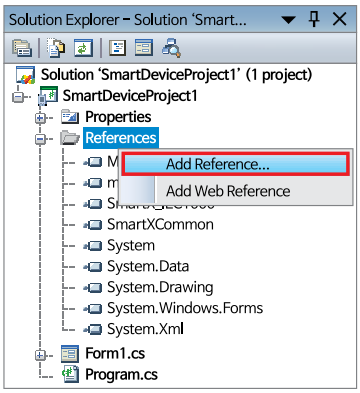
2) 참조 문제 해결 방법

※ 본 STEP은 IEC1000-Series 기준으로 작성되었습니다.

[STEP-1] [솔루션 탐색기] → [참조] → SmartX와 관련된 모든 DLL 파일 선택 후 마우스 오른쪽 버튼 클릭 → [제거] 클릭



[STEP-2] [참조] 선택 후 마우스 오른쪽 버튼 클릭 → [참조 추가] 클릭

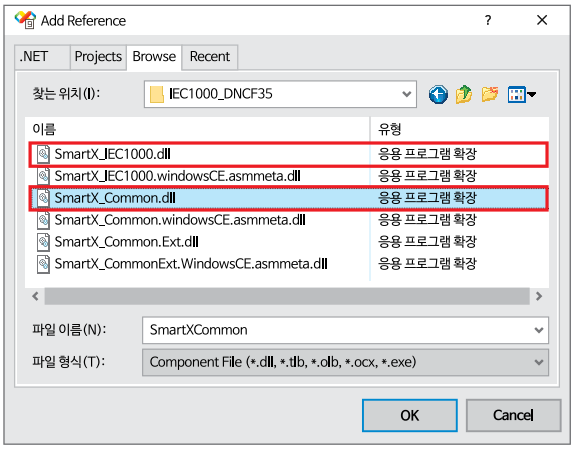


SmartX 컴포넌트에 따라 참조되는 DLL 파일과 파일의 용량이 다르므로 사용자가 프로젝트에 사용한 SmartX 컴포넌트를 확인해 보시고 참조 제거와 추가를 해주시기 바랍니다.

참고 컴포넌트에 따른 DLL 파일과 용량은 "DLL 파일별 포함되는 컴포넌트"와 "제품별 DLL 파일의 용량"을 참고하시기 바랍니다.

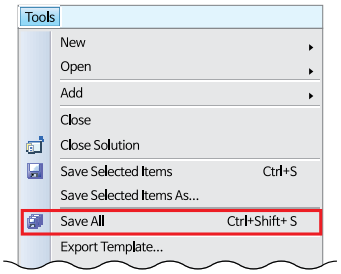
[STEP-3] [찾아보기] – [찾는 위치] 파일 경로 위치 지정 → 프로젝트에 사용한 SmartX DLL 파일 선택 → [확인] 클릭

[파일 경로] "C:₩Program Files₩HNS₩Embedded SmartX Component₩IEC1000_DNCF35" 제거한 SmartX DLL 파일과 동일한 DLL 파일을 참조 추가합니다.



참고 개발 대상 장치에 따른 DLL 파일 경로는 "제품 및 .NET Compact Framework Version에 따른 DLL 파일 경로"를 참고하시기 바랍니다.

[STEP-4] [파일] – [모두 저장] 클릭 → 빌드 되는지 확인 → 프로젝트 파일 다시 열기 → 에러 발생 없어짐



참조 DLL 참조 에러 발생 시 해결 방법

※ 홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 29. Visual Studio의 폼 디자이너 화면에서 오류가 발생하는 경우 해결 방법

만약 위와 같은 방법으로 해결이 안 된다면 당사로 문의하시기 바랍니다.

4. DLL 파일별 포함되는 컴포넌트 정보

1) 제품 및 .NET Compact Framework Version에 따른 DLL 파일 경로

기준경로 C:₩Program Files₩HNS₩Embedded SmartX Component

파일 경로	.NET Compact Framework Version	제품	해당 DLL 파일	
기준 경로₩IEC266_DNCF20	.NET CF 2.0 Base	IEC266-Series	SmartX_IEC266.dll	SmartXCommon.dll SmartXCommonExt.dll
기준 경로₩IEC667_DNCF20		IEC667-Series	SmartX_IEC667.dll	
기준 경로₩IEC667_DNCF35	.NET CF 3.5 Base	IEC667-Series	SmartX_IEC667.dll	SmartXCommon.dll SmartXCommonExt.dll
기준 경로₩IEC1000_DNCF35		IEC1000-Series	SmartX_IEC1000.dll	

2) DLL 파일별 포함되는 컴포넌트

SmartX_IEC266 / 667 / 1000	SmartXCommon			SmartXCommonExt
SmartVideo(667Only)	SmartRemote	SmartButton	SmartUpDown	SmartLock
SmartSound	SmartListBox	SmartTrackBar	SmartMonthCalendar	SmartUpdate
SmartInputCounter	SmartGroupBox	SmartPlayer	SmartThread	
SmartPWM	SmartFile	SmartKeyPad	SmartRadioButton	
SmartIIC	SmartTCPMultiServer	SmartInnerForm	SmartModbusSlave	
SmartUART	SmartTCPClient	SmartModbus	SmartLabel	
SmartWatchDog	SmartPrint	SmartFTP	SmartForm	
SmartBattery	SmartMemory	SmartConfigs	SmartTimer	
SmartDAC	SmartComboBox	SmartSplash	SmartKeyboard	
SmartGPIO	SmartSeparatorLine	SmartProgressBar	SmartCheckBox	
SmartADC	SmartDraw	SmartScreenSaver		
SmartSerialPort				

3) 제품별 DLL 파일의 용량

제품별	DLL 파일명	용량
IEC1000	SmartX_IEC1000.dll	257KB
	SmartXCommon.dll	1070KB
	SmartXCommonExt.dll	1137KB
IEC667(3.5)	SmartX_IEC667.dll	334KB
	SmartXCommon.dll	1070KB
	SmartXCommonExt.dll	1137KB
IEC667(2.0)	SmartX_IEC667.dll	321KB
	SmartXCommon.dll	977KB
	SmartXCommonExt.dll	1163KB
IEC266	SmartX_IEC266.dll	177KB
	SmartXCommon.dll	977KB
	SmartXCommonExt.dll	1163KB

※ SmartX Framework Ver3.2.4 기준

참고 SmartX Framework 업데이트에 따라 용량이 변경될 수 있습니다.

주의 사용자 정의 컨트롤 개발 시 주의사항

스마트 장치 프로젝트에서 사용자 정의 컨트롤(User-Control) 내에 SmartX Framework 사용자 인터페이스 관련 컴포넌트를 사용할 경우 다양한 문제가 발생할 수 있으므로 사용을 권장하지 않습니다.

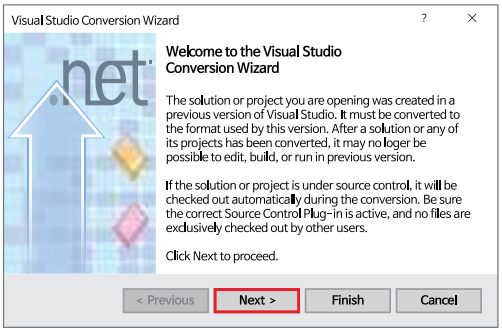
Part-Ⅲ. 개발 환경 관련 TIP

1. Visual Studio 2005 프로젝트를 Visual Studio 2008에서 읽어오기 (Migration)

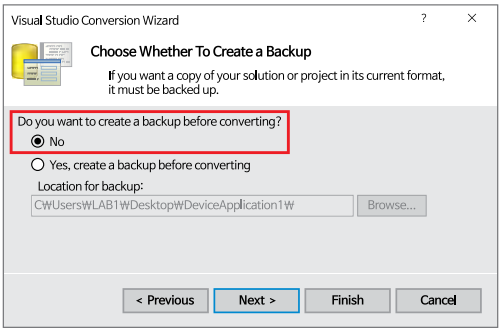
Visual Studio 2005에서 작성된 프로젝트를 Visual Studio 2008로 Migration 하기 위한 방법입니다.

[STEP-1] Visual Studio 2005에서 작성된 프로젝트 파일을 Visual Studio 2008로 열기

[STEP-2] Visual Studio 변환 마법사 시작

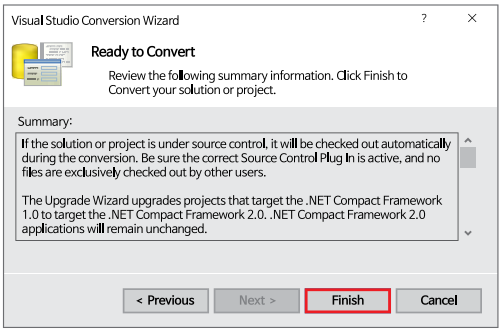


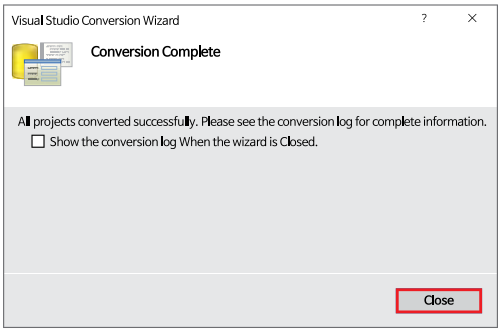
[STEP-3] 백업파일(Visual Studio 2005) 설정 [아니요] 선택



[STEP-4] [마침] 클릭 → Visual Studio 2008 프로젝트로 변환 완료

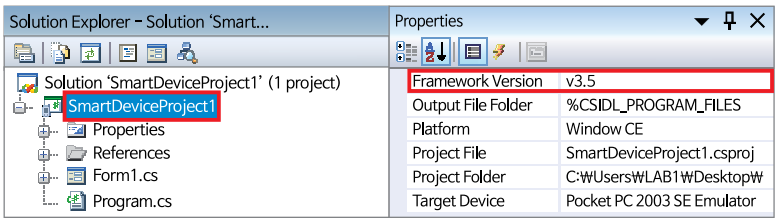
Visual Studio 2005에서 작성된 프로젝트 파일이 이제 Visual Studio 2008에서도 사용 가능합니다.





2. 현재 프로젝트의 .NET Compact Framework Version 확인하기

솔루션 탐색기에서 해당 프로젝트를 선택 후 프로젝트 속성에서 Framework 버전을 확인하시면 됩니다.
[단축키 : F4]



3. .NET CF 2.0 버전 솔루션을 .NET CF 3.5 솔루션으로 업그레이드 하기

.NET Compact Framework 2.0으로 개발된 프로젝트를 .NET Compact Framework 3.5로 업그레이드할 수 있는 방법을 설명합니다.

주의 .NET Compact Framework 3.5는 IEC1000-Series와 IEC667-Series (O/S Core Standard 이상)만 지원됩니다.

[STEP-1] [.NET Compact Framework 3.5 Base SmartX Framework 설치 파일] 다운로드

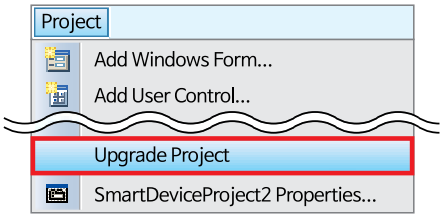
파일 다운로드 ▶ 홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 관련 → SmartX Framework 설치 파일에서 **[.NET Compact Framework 3.5 Base SmartX Framework 설치 파일]** 다운로드

※ .NET Compact Framework 3.5 Base SmartX Framework가 이미 설치되어 있다면 [STEP-2]로 넘어가시기 바랍니다.

참고 SmartX Framework 설치에 관한 자세한 설명은 "SmartX Framework 설치하기"를 참고하시기 바랍니다.

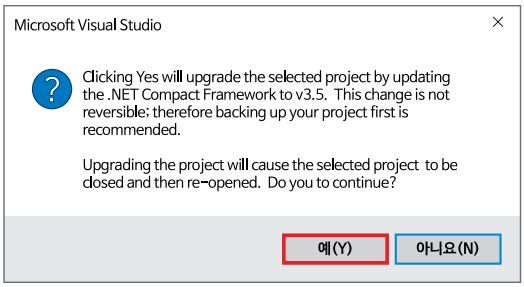
[STEP-2] .NET CF 2.0으로 개발된 프로젝트를 Visual Studio 2008에서 열기

[STEP-3] [프로젝트] – [프로젝트 업그레이드] 클릭



[STEP-4] [예] 클릭

업그레이드 여부를 확인합니다. 프로젝트 백업은 필수입니다.



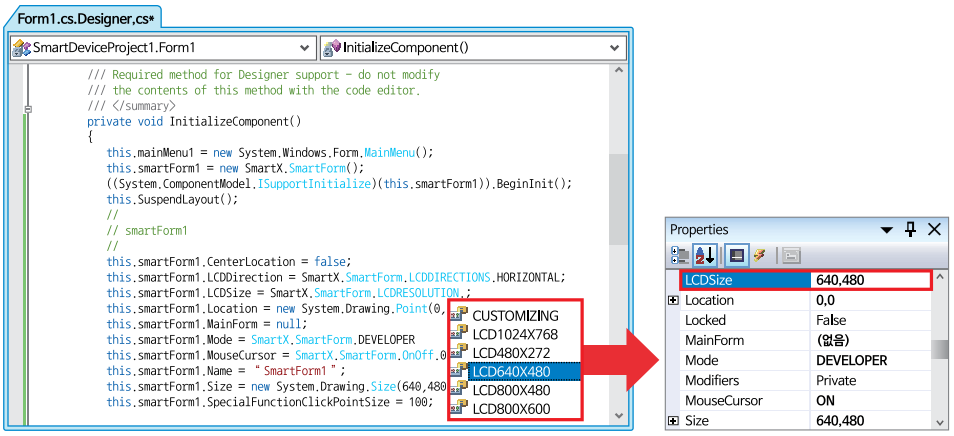
[STEP-5] SmartX Framework와 관련된 모든 DLL(SmartX_IEC266/667/1000.dll, SmartXCommon.dll, SmartX CommonExt.dll) 파일의 참조를 제거하고 새로 참조

주의 개발 환경과 대상 장치에 따라 참조하는 DLL 파일이 다르므로 잘못 참조한다면 ERROR가 발생할 수 있습니다.

참고 참조에 관한 자세한 설명은 "SmartX Framework 참조 ERROR 문제 해결 방법"을 참고하시기 바랍니다.

4. Designer.cs에서 코드 수정 시 일어나는 현상

xxx.Designer.cs에서 코드 수정 시 디자이너 속성 창의 해당 내용이 변경되는 것을 확인할 수 있습니다.



주의 xxx.Designer.cs에서 코드 위치 수정 시 주의사항

xxx.Designer.cs에서 코드의 위치를 바꾼 뒤 컴포넌트를 추가하시거나 디자이너 속성 창에서 속성 수정을 하신다면 원래 위치로 복원이 되므로 xxx.Designer.cs에서 직접 코드를 수정하기보다는 디자이너 속성 창에서 수정하시기 바랍니다.

SuspendLayoutInterval 코드 위치 이동

```
private void InitializeComponent()
{
  this.mainMenu1 = new System.Windows.Forms.MainMenu();
  this.smartForm1 = new SmartX.SmartForm();
  ((System.ComponentModel.ISupportInitialize)(this.smartForm1)).BeginInit();
  this.SuspendLayout();
  //
  // smartForm
  //
  this.smartForm1.SuspendLayoutInterval = 0;

  this.smartForm1.CenterLocation = false;
  this.smartForm1.LCDDirection =SmartX.SmartForm.LCDDIRECTIONS.HORIZONTAL;
  this.smartForm1.LCDSize = SmartX.SmartForm.LCDRESOLUTION.LCD640X480;
  this.smartForm1.Location = new System.Drawing.Point(0, 0)
  this.smartForm1.MainForm = null;
  this.smartForm1.Mode = SmartX.SmartForm.RUNMODE.DEVELOPER;
  this.smartForm1.MouseCusor = SmartX.SmartForm.OnOff.OFF;
  this.smartForm1.Name = "smartForm1";
  this.smartForm1.Size = new System.Drawing.Size(640, 480);
  this.smartForm1.SpecialFunctionClickPointSize = 100;
```

원래 위치로 복원

```
private void InitializeComponent()
{
  this.mainMenu1 = new System.Windows.Forms.MainMenu();
  this.smartFowm1 = new SmartX.SmartForm();
  ((System.ComponentModel.ISupportInitialize)(this.smartForm1)).BeginInit();
  this.SuspendLayout();
  //
  // smartForm
  //
  this.smartForm1.CenterLocation = false;
  this.smartForm1.LCDDirection =SmartX.SmartForm.LCDDIRECTIONS.HORIZONTAL;
  this.smartForm1.LCDSize = SmartX.SmartForm.LCFRESOLUTION.LCD640X480;
  this.smartForm1.Location = new System.Drawing.Point(0, 0)
  this.smartForm1.MainForm = null;
  this.smartForm1.Mode = SmartX.SmartForm.RUNMODE.DEVELOPER;
  this.smartForm1.MouseCusor = SmartX.SmartForm.OnOff.OFF;
  this.smartForm1.Name = "smartForm1";
  this.smartForm1.Size = new System.Drawing.Size(640, 480);
  this.smartForm1.SpecialFunctionClickPointSize = 100;
  this.smartForm1.SuspendLayoutInterval = 0;
```

5. 배경 이미지 삭제하기

배경 이미지를 삽입한 사용자 인터페이스 컴포넌트는 디자이너 속성 창에서 이미지 삭제가 안 됩니다. 이런 경우 xxx.Designer.cs에서 사용자가 직접 소스를 삭제하셔야 합니다.

참고 이미지를 설정할 수 있는 사용자 인터페이스 컴포넌트

SmartForm, SmartInnerForm, SmartButton, SmartRadioButton, SmartUpDown, SmartGroupBox, SmartCheck Box, SmartTrackBar, SmartKeyboard, SmartKeyPad, SmartDraw

SmartInnerForm와 SmartGroupBox로 예제를 들어보겠습니다. 폼의 이름을 Form1이라고 가정한다면, 아래의 코드를 참고하여 배경 이미지를 삭제하시기 바랍니다.

Form1.Designer.cs

```
// 변경 전
this.smartInnerForm1.BackImage =
((System.Drawing.Image)(resources.GetObject( " smartInnerForm1.BackImage " )));
this.smartGroupBox1.Image = ((System.Drawing.Image)(resources.GetObject( " smartGroupBox1.Image " )));

// 변경 후
this.smartInnerForm1.BackImage = null;
this.smartGroupBox1.Image = null;  // 배경 이미지 소스값을 null로 주어 이미지를 삭제합니다.
// 코드 수정 후 저장해야 합니다.
```

6. IEC-Series의 운영체제 사양별 기본 탑재 Font

※ IEC-Series는 기본적으로 Standard 버전의 OS 라이선스를 탑재

Optimum	Standard	Professional
tahoma.ttf	cour.ttf tahoma.ttf gulim.ttc	cour.ttf tahoma.ttf gulim.ttc wingding.ttf

주의 IEC-Series는 Windows CE 운영체제로 TrueType의 글꼴만을 지원하며, Font의 확장자는 ttc, ttf로 제한합니다.

개발 PC에는 다양한 Font들이 설치되어 있지만, IEC-Series는 기능 최적화를 위해 기본 탑재된 Font의 종류가 적습니다. 이러한 이유로 개발 시 프로젝트에 사용한 Font가 IEC-Series에서 적용되지 않을 수 있습니다. 따라서 IEC-Series에 탑재된 Font가 아닌 다른 Font를 사용하거나 다국어를 사용하고 싶은 경우에는 아래를 참조하시기 바랍니다.

참조 다양한 Font 사용 방법

※ 홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 20. IEC-Series에서 다양한 폰트(Fonts 폴더, 트루타입) 사용 방법 안내

참조 다국어 출력 방법

※ 홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 62. IEC-Series에서 다국어 출력 방법

Part-Ⅳ. 개발 시 유의사항

1. 이미지 관련 주의사항

1) 이미지 제작 가이드

문제점 이미지 제작 가이드를 지키지 않는 경우에 발생되는 오류 유형

※ 프로그램 사용 도중 임의의 시점에서 응용 프로그램 오류가 발생할 수 있습니다.
(이 시점은 짧게 1일, 길게는 몇 개월 이후일 수 있습니다.)
※ 프로그램 종료 시 프로세스가 올바르게 종료되지 않을 수 있습니다.
※ 프로젝트에 이미지 적용 시 이미지의 크기가 임의로 변경(축소)되어 보일 수 있습니다.

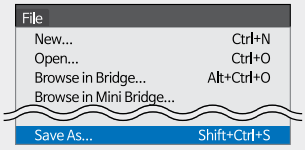
[이미지 제작 가이드를 지키지 않는 경우 발생되는 에러 메시지 형태]

프로그램에 적용할 이미지 작업 시 반드시 지켜야 할 사항으로 디자이너가 숙지해야 할 사항입니다.

STEP-1 포토샵을 사용하여 PNG로 저장

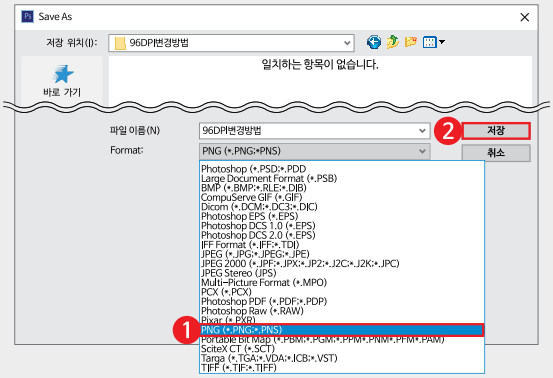
포토샵이 아닌 다른 Tool을 사용하시는 경우 호환성 문제로 프로그램 ERROR가 발생할 수 있습니다. 포토샵으로 이미지 저장 시 "File" – "Save for Web"을 통한 이미지 저장은 권장하지 않습니다. 반드시 "Save As"를 통해 저장하시기 바랍니다.

포토샵 열기 → [메뉴] → [File] → [Save As]



[Format] PNG 선택 → [저장]

이미지 파일의 Format을 PNG (*.PNG;*.PNS)로 선택하여 저장하는 것을 권장합니다.



참고 확장자 PNG와 달리 JPG와 GIF는 손실 압축 파일이기 때문에 투명 효과 처리 시 Masking이 고르지 않는 현상이 발생하거나 이미지의 해상도가 떨어져 보일 수 있습니다.
(단, BMP는 용량이 크므로 메모리 최적화를 위해 PNG를 권장합니다.)

주의 Form에 적용한 이미지(리소스)를 추출하여 재사용을 금지합니다.

참고 포토샵이 없는 경우

개발 프로그램의 이미지를 [이미지 제작 가이드]에 맞게 변경이 불가능한 경우 HNS로 신청서를 작성하여 이미지와 같이 메일로 보내주시면 권장 이미지로 변경해드리겠습니다.

이미지 변경 신청서			
업체명		담당자	
연락처		E-Mail	

[E-mail] app@smartx.co.kr 또는 design@smartx.co.kr

STEP-2 이미지 해상도 96DPI

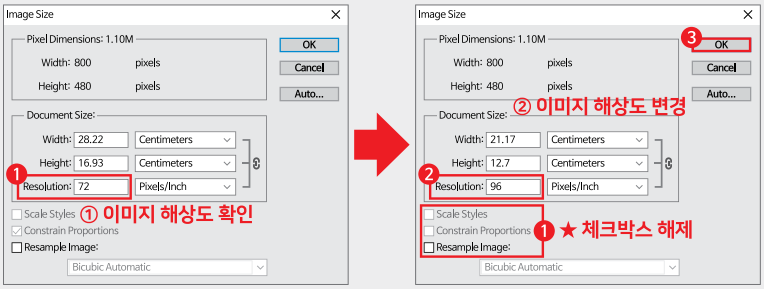
이미지 파일(BMP, JPG, PNG)의 해상도를 96DPI가 아닌 다른 해상도를 사용하시는 경우 프로젝트에 이미지 적용 시 이미지의 크기가 임의로 변경(축소)되어 보일 수 있습니다. 포토샵에서 이미지 해상도를 변경하시기 바랍니다.

사용 Tool	DPI	이미지 Format
Photoshop(★필수)	96(★필수)	PNG (권장)

포토샵 열기 → [메뉴] → [File] → [Open]

변경할 이미지 파일을 불러옵니다.

[메뉴] → [Image] → [Image Size] 해상도 확인 및 변경



※ 반드시 체크박스 해제 후 해상도를 변경해야 합니다.

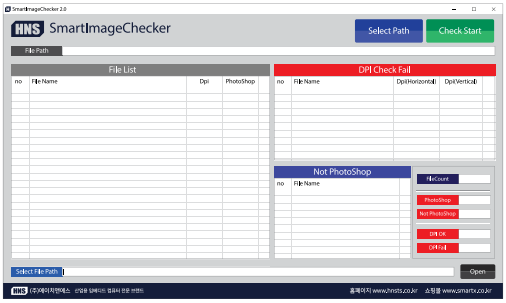
참고 SmartImageChecker 프로그램으로 올바른 이미지 형식인지 확인방법

SmartImageChecker 프로그램으로 이미지가 포토샵으로 제작되었는지 여부와 이미지의 DPI를 확인할 수 있습니다. 이미지를 적용하기 전 반드시 체크하시는 것을 권장합니다.

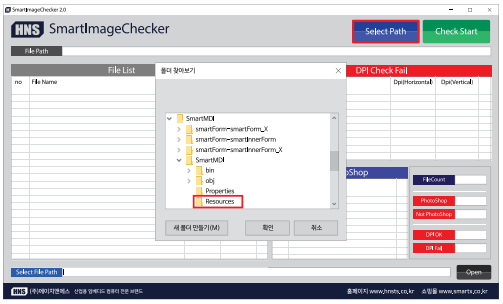
[STEP-1] SmartImageChecker 프로그램을 개발 컴퓨터에 설치

"홈페이지(www.hnsts.co.kr) - 자료실 - SmartXFramework 관련 - 8. SmartImageChecker 프로그램"을 다운받은 후 압축을 해제합니다.
(해당 프로그램은 IEC-Series에서 실행되지 않습니다.)

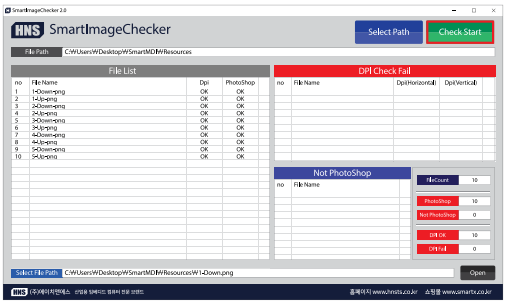
[STEP-2] SmartImageChecker 프로그램을 실행



[STEP-3] [Select Path] 버튼을 클릭해 검사할 원본 이미지가 있는 폴더를 선택

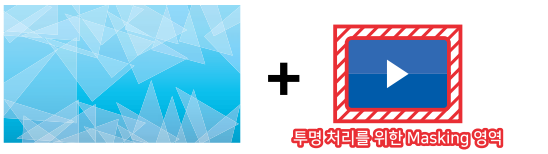


[STEP-4] [Check Start]버튼을 클릭해 올바른 이미지 형식인지 확인



2) 투명(Masking) 처리 영역 제작 가이드

배경 이미지(SmartForm) 위에 컴포넌트들의 이미지를 올려야 할 경우 경계선을 깔끔하게 처리하기 위해서는 Masking 처리를 진행해야 합니다. 그 경계선을 최소화하기 위한 방법으로 반드시 아래의 사항들을 숙지 해주시기 바랍니다.



배경 컴포넌트 적용된 이미지　　사용자 인터페이스 컴포넌트에 적용된 이미지

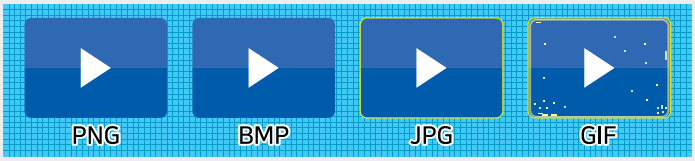
Masking 처리된 이미지

[그림] 투명 Masking 영역 안내

컴포넌트 이미지 Masking 처리 시 지켜야 하는 항목은 아래와 같습니다.

STEP-1 이미지 형식은 PNG로 통일하여 사용

확장자 PNG와 달리 JPG와 GIF는 손실 압축 파일이기 때문에 투명 효과 처리 시 Masking이 고르지 않는 현상이 발생하거나 이미지의 해상도가 떨어져 보일 수 있습니다. (단, BMP는 용량이 크므로 메모리 최적화를 위해 PNG를 권장합니다.)



STEP-2 Masking 영역의 색상은 이미지 영역에 사용되지 않는 색상이며 단색으로 지정

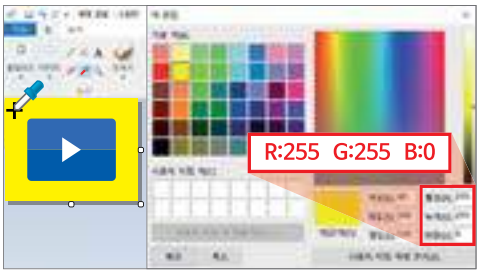
Masking 영역의 색상이 이미지 영역에 사용되는 색상과 같을 경우 이미지 영역도 같이 투명 처리되며, 단색으로만 지정되지 않을 경우 투명 처리가 안 될 수 있습니다.

Masking 영역의 색상이 단색인지 확인하는 방법

Masking 영역의 색상이 육안상으로 단색으로 보이지만 경우에 따라서 Pixel 또는 일정 영역의 RGB값이 다를 수 있습니다. 이를 확인하기 위해서 아래 방법을 설명합니다.

방법-1 [색 선택] → 확인하고자 하는 Masking 영역 클릭 → [색 편집] → RGB값 확인

그림판에서 투명 처리할 이미지를 불러온 뒤 그림판 도구의 "색 선택"으로 Masking 색상의 RGB값을 확인합니다.



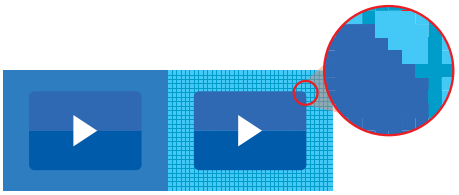
방법-2 다른 색상으로 [색 채우기] → [색 선택] → 안 채워진 영역 클릭 → [색 편집] → RGB값 확인

"색 채우기"로 Masking 영역에 다른 색상을 채우면 안 채워지는 부분을 발견할 수 있습니다. 그 부분을 다시 "색 선택"으로 RGB값을 확인해보면 처음의 RGB값과 다르므로 Masking 영역의 색상이 단색이 아닌 것을 확인할 수 있습니다.

STEP-3 Masking 영역의 색상은 배경 이미지의 색상과 유사한 색상을 사용

배경 이미지 색상은 하늘색, Masking 영역의 색상을 하늘색으로 설정한다면 투명 효과 처리했을 때, Masking 색상이 자연스러워 보입니다.

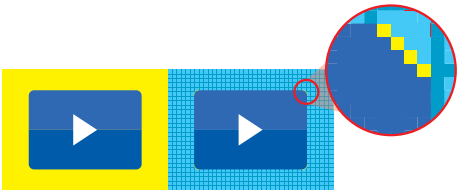
유사한 색상 (좋은 예)



Masking 적용 시

만약 Masking 영역의 색상을 노란색으로 설정한다면 투명 효과 처리했을 때, Masking 색상인 노란색이 명확하게 보이는 것을 확인할 수 있습니다.

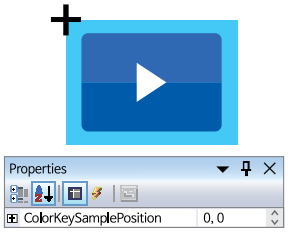
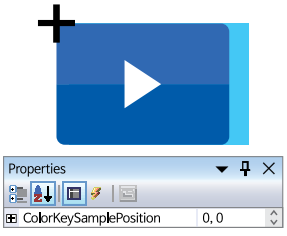
상반된 색상 (나쁜 예)



Masking 적용 시

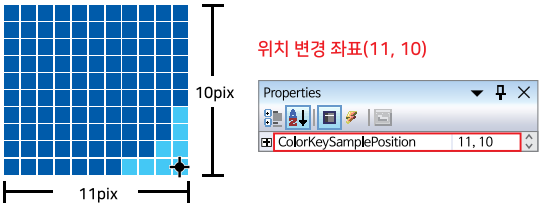
3) 이미지 Masking 영역 위치 변경 방법

이미지 Masking 처리 색상은 컴포넌트 속성 중 ColorKeySamplePosition 속성에 지정된 좌표값의 색상을 사용합니다. 속성의 기본값은 좌측 상단 기준 [0, 0]이며 이 위치에 Masking 영역 색상이 아닌 이미지 색상을 지정할 경우 속성값을 변경해야 합니다.

속성값 변경 불필요	속성값 변경 필요
[0, 0] 위치에 Masking 영역이 있는 경우 (좌측 상단 기준)	[0, 0] 위치에 Masking 영역이 없는 경우
	

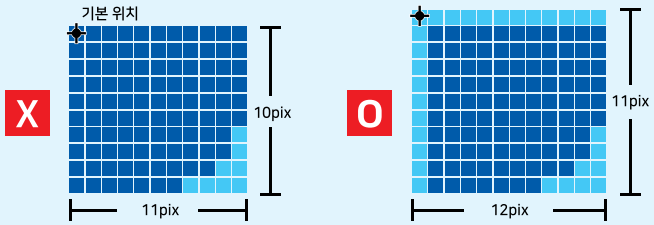
속성값 변경 방법

ColorKeySamplePosition 속성을 사용하여 이미지 영역에서 Masking 영역으로 좌표값을 변경합니다.



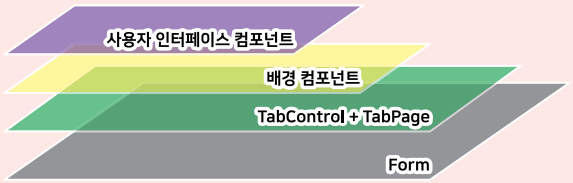
참고 SmartUpDown 컴포넌트에서 Masking 영역 위치 변경 방법

SmartUpDown에서는 ColorKeySamplePosition 속성이 지원되지 않습니다. 만약 Masking 영역의 좌표([0, 0])에 이미지가 존재할 경우 Masking이 불가능합니다. ColorKeySamplePosition을 사용하여 Masking 영역을 지정할 수 없으므로 디자이너가 강제로 좌표값에 Masking 영역을 만들면 이미지의 Masking 처리가 적용됩니다.



주의 TabControl에서 배경 컴포넌트를 사용하여 사용자 인터페이스 컴포넌트의 투명 처리 시 주의사항

TabControl에 배경 컴포넌트를 배치하여 사용자 인터페이스 컴포넌트의 투명 처리 후 프로젝트를 닫고 다시 켠 뒤 새로운 사용자 인터페이스 컴포넌트를 추가해 투명 처리를 하면 배포 시 기존 컴포넌트의 투명 효과가 풀리는 현상이 발생합니다.



문제 발생의 원인은 xxxx.Designer.cs에서 각 컴포넌트의 세부 속성을 설정하는 부분의 위치가 신규로 추가된 사용자 인터페이스 컴포넌트보다 아래에 위치하게 되면서 발생합니다.

해결 방법은 xxxx.Designer.cs에서 각 컴포넌트의 세부 속성 설정 부분에서 배경 컴포넌트 관련 소스 코드를 상단으로 이동시키면 되며 아래 예시를 참고하시기 바랍니다.

TabControl에 배경 컴포넌트는 pictureBox1, 기존 사용자 인터페이스 컴포넌트는 smartCheckBox1이라고 가정한다면 각 컴포넌트의 세부 속성 설정 부분에서 pictureBox1을 맨 위로 이동합니다.

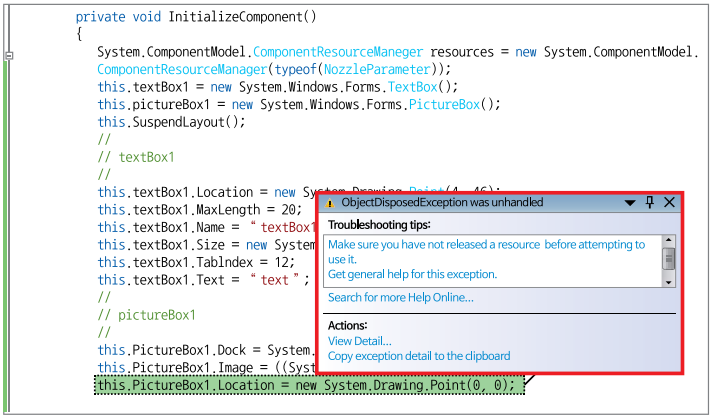
Form1.Designer.cs

```
private void InitializeComponent()
{
  // -- 중략 --
  // 각 Component의 세부 속성 설정 시작 부분

  // tabControl1
  // tabPage1
  // smartCheckBox1
  //
  // pictureBox1
  //
```

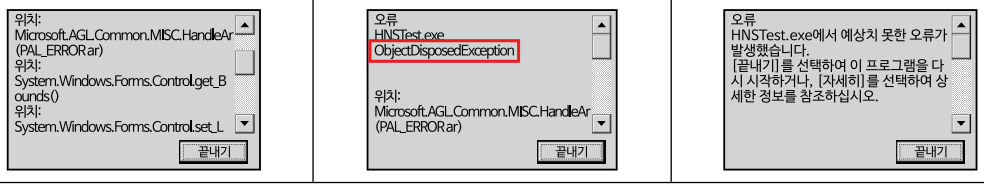
```
    this.pictureBox1.Image
    = ((System.Drawing.Image)(resources.GetObject("pictureBox1.Image")));
    this.pictureBox1.Location = new System.Drawing.Point(13, 16);
    this.pictureBox1.Name = "pictureBox1";
    this.pictureBox1.Size = new System.Drawing.Size(526, 322);
    //
    // smartCheckBox2
    // -- 중략 --
}
```

4) InitializeComponent() Exception 문제 해결 방법

Visual Studio



IEC-Series 장치에서 출력되는 오류 메시지 형태



위와 같이 InitializeComponent() 함수에서 원인을 찾기 어려운 예외가 발생할 경우 아래의 CHECK POINT를 참고하시어 점검 및 해결하시기 바랍니다.

CHECK POINT-1 문법적인 오류 또는 논리적인 오류가 있는지 확인

문법적인 오류 또는 논리적인 오류가 없음을 확인 후에도 예외가 발생할 경우 CHECK POINT-2로 넘어가시기 바랍니다.

CHECK POINT-2 특정 이미지를 추가한 뒤 예외가 발생할 경우

프로그램에서 Resources 이미지를 사용하고 예외가 다음에 해당하는지 확인하시기 바랍니다.

Exception 종류 형식		
Object Disposed Exception	Not Supported Exception	Null Reference Exception

해당된다면 프로그램에 사용한 이미지 형식에 따른 문제이므로 [이미지 제작 가이드]에 맞춰 다시 설정하시기 바랍니다.

참고 이미지 설정 방법의 자세한 설명은 "이미지 제작 가이드"를 참고하시기 바랍니다.

2. SmartX Component 동적 생성 시 유의사항

SmartX Framework 사용자 인터페이스(User Interface) 관련 컴포넌트는 기본적으로 Form Designer를 통하여 생성 및 조작을 권장합니다.

주의 사용자 인터페이스 관련 컴포넌트의 동적 생성을 권장하지 않습니다.

동적으로 생성된 사용자 인터페이스 컴포넌트를 사용해야 할 경우 아래의 Controls.Add() 메소드를 Form Designer를 통하여 ControlCollection에 추가하시기 바랍니다.

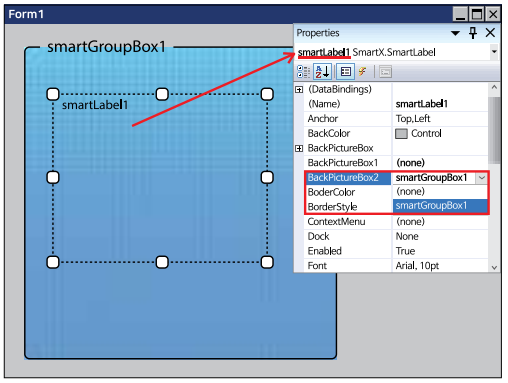
Form1.Designer.cs

```
private void InitializeComponent()
{
  // -- 중략 --
  this.Controls.Add( [ ] );  // 동적 생성된 Component를 빈칸에 추가
}
```

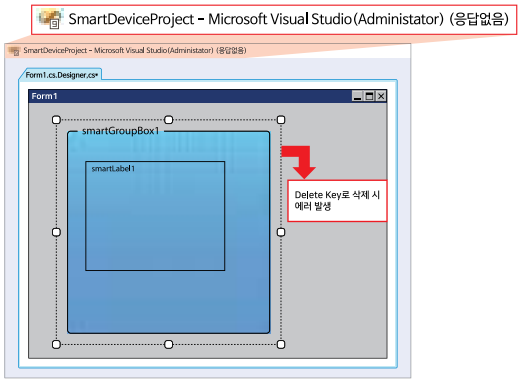
3. 컨테이너 객체를 디자이너에서 삭제 시 유의사항

SmartX Framework 사용자 인터페이스(User Interface) 컴포넌트 중 SmartGroupBox는 컨테이너 객체입니다. Group에 묶이는 컴포넌트들의 투명효과를 적용하기 위해 BackPictureBox2의 속성을 SmartGroupBox로 설정했다면 SmartGroupBox를 삭제 시 Visual Studio가 멈추면서 창이 닫혔다 다시 켜지는 Error가 발생합니다. 이 경우 컨테이너 내부의 컴포넌트들을 먼저 지우고 SmartGroupBox를 삭제하시기 바랍니다.

SmartLabel1 BackPictureBox2 속성 설정



SmartGroupBox1 삭제 시 ERROR



4. IEC266-Series에서 가용메모리 관련 유의사항

IEC266-Series의 가용메모리는 IEC667-Series, IEC1000-Series 제품에 비해 상대적으로 낮습니다. 하나의 Form에 많은 컴포넌트를 사용한다면 프로그램 실행 시 Form이 느리게 열리거나 Form 간의 전환이 느려집니다. 많은 Form을 사용할 경우 Out Of Memory Exception이 발생할 수 있습니다. 제품별 가용 메모리는 아래를 참고하시기 바랍니다.

제품별 가용 프로그램 메모리	IEC266-Series	IEC667-Series	IEC1000-Series
	29MB	172MB	335MB

참조 메모리에 대한 자세한 설명은 "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 46. IEC-Series의 메모리 설명과 프로그램 배포 오류 발생 시 해결 방법"을 참조하시기 바랍니다.

[Memo]

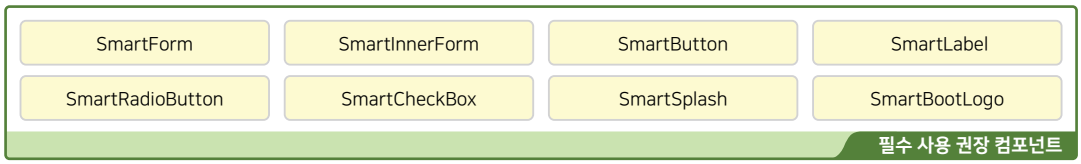
Part-V
주요 권장 컴포넌트

SmartX Framework를 프로젝트에 사용할 경우 적용을 권장하는 주요 컴포넌트들로 구성되어 있습니다.
되도록 주요 권장 컴포넌트를 사용하여 개발하시기를 적극적으로 권장합니다.

1. SmartForm
2. SmartInnerForm
3. SmartButton
4. SmartLabel
5. SmartRadioButton
6. SmartCheckBox
7. SmartSplash
8. SmartBootLogo

Part-Ⅴ. 주요 권장 컴포넌트

컴포넌트의 학습 순서는 중요하지 않습니다. 적용 및 필요한 기능을 확인하여 참고 및 학습하시기를 권장합니다. 아래의 컴포넌트는 가장 먼저 학습하시기 바라며 프로젝트에 적용할 것을 추천해 드립니다.



1. SmartForm

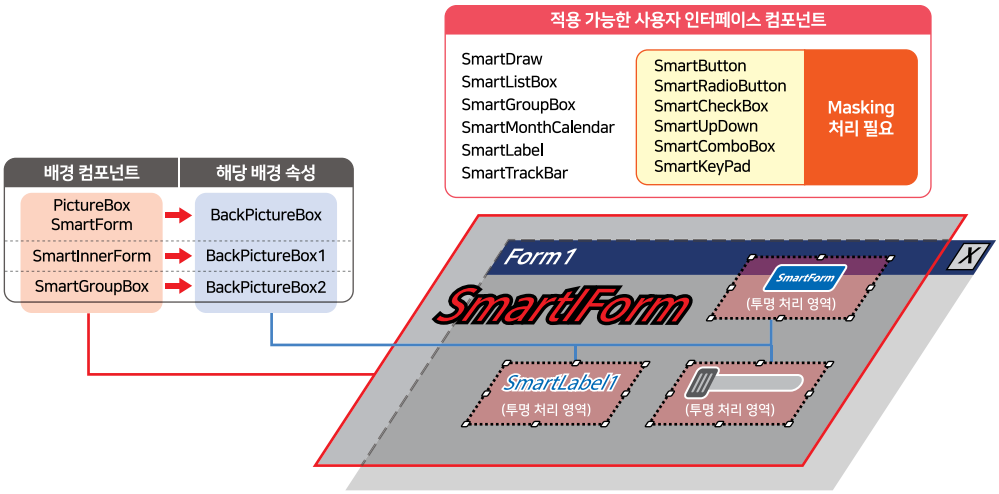
SmartForm은 화면 디자인 시 이미지 적용 및 컴포넌트들의 투명 처리를 지원하여 사용자가 원하는 깔끔하고 다양한 디자인을 적용할 수 있습니다. 또한 MDI(Multiple Document Interface) 프로그램 구현 시 초기 공통적인 구현 사항을 간편하게 할 수 있으며, 화면 전환할 때 발생하는 잔상 및 겹침 현상을 개선하는 기능을 지원합니다. 이외에도 SmartForm은 .NET Compact Framework에서 빠진 기능들이 추가되어 장치 응용 프로그램 개발 작업 시 불필요한 코딩을 줄여 줄 것입니다.

- 배경 이미지 설정 기능
- 투명 효과 지원되는 사용자 인터페이스 컴포넌트와 연동
- 다중 창(MDI) 기능 지원
- 공통 영역 표시 기능
- SmartInnerForm과 연동
- 화면 잔상 개선 기능
- 모달 창(Modal Window) 출력 기능
  - ㄴ Blocking-Modal, NoneBlocking-Modal
- 모델별 편리한 화면 크기 지정 기능(속성창에서 간편하게 설정)
- RUNTIME 시 불필요한 폼 속성값 변경
- 디버그 모드 시 편리한 프로그램 종료 기능
- 장치 응용 프로그램 개발 모드 시 작업 표시줄 자동 숨김 기능
  - ㄴ 작업 표시줄이 폼의 하단 영역을 가리는 경우 자동 숨김 처리
- Mouse Cursor Hide 기능
- 시스템 설정을 위한 특수 터치 이벤트 처리 기능

1) 사용자 인터페이스 투명 처리 조합

SmartX Framework 컴포넌트 중 사용자 인터페이스(User Interface) 관련 컴포넌트들은 서로의 조합에 따라서 투명 효과를 처리할 수 있습니다. 다음과 같이 배경 컴포넌트와 배경 위에서 사용되는 컴포넌트들로 구성되며 적용은 아래 이미지의 배경 컴포넌트의 속성값을 설정하시면 구현할 수 있습니다.

참고 배경 컴포넌트의 배경 이미지가 없고 BackColor 속성만 설정된 경우 투명 효과 기능을 사용하지 않고 사용자 인터페이스 컴포넌트의 BackColor 속성을 배경 컴포넌트와 같은 색상으로 설정하시기 바랍니다.



참고 "투명(Masking) 처리 영역 제작 가이드"를 참고하시기 바랍니다.

2) SmartForm을 이용한 MDI 조합 구성

2-1) SmartForm-SmartForm MDI 조합과 SmartForm-SmartInnerForm MDI 조합의 차이점

SmartForm을 이용해 MDI 조합 구성 시 자식 폼이 SmartForm인지 SmartInnerForm인지에 따라 크게 2가지 방법으로 나눌 수 있습니다. 두 방법 모두 부모 폼은 반드시 SmartForm이어야 하며, 자세한 차이점은 아래 표를 확인하여 용도에 맞게 프로젝트에 적용하시기 바랍니다.

권장 Case에 따른 MDI 조합 선택기준

CASE	내용	권장 MDI 조합
1	- 폼이 많은 경우 - 각 폼에 컴포넌트가 많이 배치되는 경우	SmartForm-SmartForm
2	- 각 폼간 데이터의 공유가 잦은 경우	SmartForm-SmartInnerForm

[표] SmartForm-SmartForm MDI 조합과 SmartForm-SmartInnerForm MDI 조합의 차이점

MDI 조합	SmartForm-SmartForm	SmartForm-SmartInnerForm
부모 폼	SmartForm	
자식 폼	SmartForm	SmartInnerForm
물리적인 폼 수	N개	1개
장점	1. 폼 디자인 작업이 SmartInnerForm 조합에 비해 편함 2. 각 폼이 물리적으로 분리되어 Visual Studio의 디자인 모드에서 폼 갱신(Reload) 시간이 SmartInnerForm 조합보다 짧음 (SmartX와 상관없이 Visual Studio에서 발생)	1. 폼간 컨트롤 객체 접근 및 데이터 공유가 SmartForm 조합에 비해 편함
단점	1. 폼간 컨트롤 객체 접근 및 데이터 공유가 SmartInner Form 조합에 비해 불편함	1. 폼 디자인 작업이 SmartForm 조합에 비해 불편함 2. 각 폼이 하나의 물리적인 폼에 구성되어 Visual Studio의 디자인모드에서 폼 갱신(Reload) 시간이 SmartForm 조합보다 김 (SmartX와 상관없이 Visual Studio에서 발생)

참조 폼 디자인의 컴포넌트 개수에 따른 디자인 창에서 화면 갱신 시간 안내

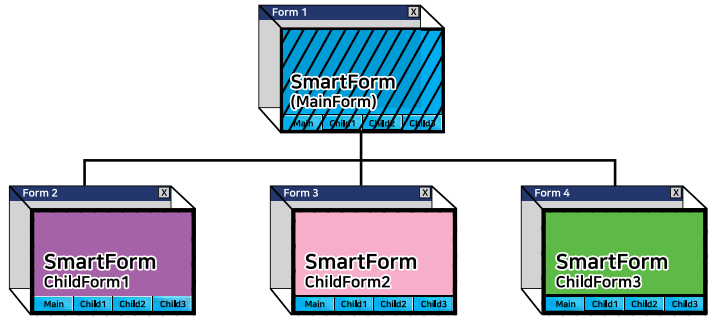
Visual Studio를 사용하여 C# 또는 VB.NET 용 프로젝트를 생성하여 디자인 창에서 컴포넌트를 사용하는 경우 프로젝트의 소스 코드를 수정하게 되면 디자인 창을 갱신(Reload)합니다. 사용되는 컴포넌트의 개수에 비례하여 화면 갱신 시간이 늘어나게 되며 그 이외 다른 문제점이 발생하지는 않습니다. 자세한 내용은 아래 경로에서 확인하시기 바랍니다.

※ 홈페이지(www.hnsts.co.kr) → 커뮤니티 → 공지사항 → 100. 폼 디자인의 컴포넌트 개수에 따른 디자인 창에서 화면 갱신 시간 안내

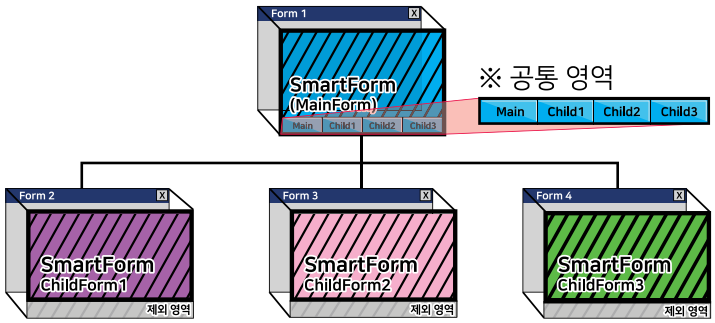
2-2) SmartForm-SmartForm MDI 조합의 형태

SmartForm을 이용하여 구현할 수 있는 MDI 조합의 형태는 크게 2가지입니다. 아래 CASE를 확인하여 프로젝트에 적용해 보시기 바랍니다.

CASE-1	모든 폼의 크기가 같은 경우
사용 시점	각 폼에 공통 영역이 없는 경우



CASE-2	부모 폼과 자식 폼의 크기가 다른 경우(각 자식 폼의 크기는 동일)
사용 시점	각 폼에 공통 영역이 있는 경우



참고 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

참조 SmartForm-SmartForm 구조의 MDI에서 데이터를 공유하는 방법

SmartForm-SmartInnerForm MDI 조합은 하나의 물리적인 폼에 구성되어 폼 간 데이터 공유가 간편하지만, SmartForm-SmartForm MDI 조합은 각각의 폼이 서로 물리적으로 분리되어 있어 폼 간 데이터 공유가 상대적으로 불편합니다. 이때 각 폼을 참조하는 방법으로 데이터의 공유를 간편하게 할 수 있으며, Tech Note에 관련 내용이 개재되어 있으니 참조하시기 바랍니다.

※ 홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 80. [C#, VB.NET] SmartForm-SmartForm 구조의 MDI에서 데이터를 공유하는 방법

참조 MDI 구성 시 ChildForm의 크기를 각각 다르게 구성하는 방법

Main Form과 ChildForm의 크기를 각각 다르게 하는 방법은 Tech Note에 관련 내용이 게재되어 있으니 참조하시기 바랍니다.

※ 홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 6. SmartForm과 SmartInnerForm으로 만들 수 있는 MDI 구성 조합

3) 화면 전환 잔상 및 겹침 현상 개선 방법

MDI(Multiple Document Interface) 구조는 화면 전환 시 잔상 및 겹침(Overlap) 현상이 발생할 수 있습니다.
이 현상은 SmartX 컴포넌트뿐만 아니라 .Net Compact Framework에서 제공하는 사용자 인터페이스에서도 동일하게 발생되며 사용되는 컨트롤의 수와 배경 이미지에 따라서 쉽게 나타날 수 있는 현상입니다.
SmartX Framework의 SmartForm을 사용하여 MDI 구성 후 SuspendLayoutInterval 속성값을 1로 설정하면 잔상 및 겹침 현상을 개선할 수 있습니다.

3-1) SuspendLayoutInterval 속성값에 따른 장단점 및 사용 시점

MDI 구조에서 화면 전환 시 SuspendLayoutInterval 속성값에 따른 차이점이 있으며, 아래 표에서 장단점을 확인하여 프로젝트에 적용하시기 바랍니다.

[표] SuspendLayoutInterval 속성값에 따른 장단점 및 사용 시점

속성값	SuspendLayoutInterval = 1 권장	SuspendLayoutInterval = 0
장점	화면 전환 시 컨트롤 잔상 및 겹침 현상 방지	SuspendLayoutInterval = 1 일 때 보다 화면 전환 속도가 상대적으로 빠름
단점	1. SuspendLayoutInterval = 0 일 때 보다 폼 전환 속도가 상대적으로 지연됨 2. 화면 전환 시 사용자 인터페이스 컨트롤의 Visible, Enabled 속성이 임의로 변경됨	화면 전환 시 컨트롤 잔상 및 겹침 현상 발생
권장 사용 시점	각 화면들이 다른 색 또는 이미지 사용 사용자 인터페이스 컨트넌트 위치가 다름	각 화면들이 동일 색 또는 이미지 사용 사용자 인터페이스 컨트넌트 위치가 동일
사용자 인터페이스 예시	※ 폼 간 유사성 낮음  [부모 폼]  [자식 폼]	※ 폼 간 유사성 높음  [부모 폼]  [자식 폼]

※ SuspendLayoutInterval 속성은 반드시 부모 폼의 SmartForm에서 설정해야 합니다.

주의 SuspendLayoutInterval 속성값을 1로 할 경우 주의사항

MDI 프로그램에서 SmartForm의 속성 중 SuspendLayoutInterval 속성값이 1일 경우 화면 전환 시 사용자 인터페이스 컴포넌트의 Visible, Enabled 속성값이 임의로 변경되는 문제가 발생합니다. 이러한 경우 SmartForm의 OnEventFormChanged 이벤트를 활용하시면 위 문제를 해결할 수 있습니다.

참조 화면 전환 시 잔상 및 겹침 현상(Overlap) 관련 공지사항

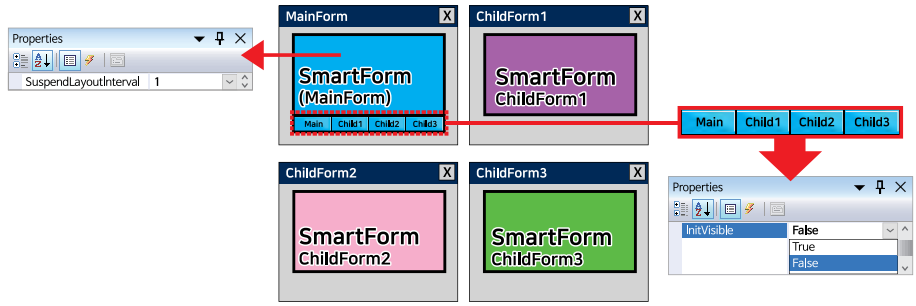
화면 전환 시 발생하는 잔상 및 겹침 현상에 대한 자세한 내용은 공지사항을 참조하시기 바랍니다.

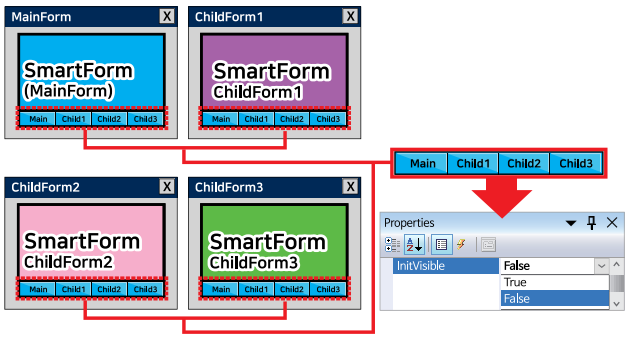
※ 홈페이지(www.hnsts.co.kr) → 공지사항 → 79. 화면(Form)전환 시 잔상 및 겹침(Overlap)현상 최소화 기능 추가

3-2) 공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정

공통 영역이 있는 MDI를 구성할 때 공통 영역에 위치한 컴포넌트의 InitVisible 속성이 True이고 부모 폼 (SmartForm)의 SuspendLayoutInterval 속성값이 1이라면 해당 컴포넌트는 화면 전환 시 사라지게 됩니다. 이 현상을 방지하기 위해서는 공통 영역에 위치한 컴포넌트들의 InitVisible 속성값을 반드시 False로 설정해야 합니다. 아래 표를 확인하여 상황에 맞게 적용하시기 바랍니다.

[표] SmartForm-SmartForm MDI 구성일 경우 공통 영역 여부에 따른 InitVisible 속성값 설정

CASE-1	공통 영역이 있으며, 공통 영역에 컴포넌트가 위치한 경우
설명	공통 영역이 있는 MDI 구성 시, 공통 영역에 위치한 컴포넌트가 부모 폼에 위치한 SmartForm의 SuspendLayoutInterval 속성값이 1일 때 공통 영역에 위치한 컴포넌트들이 사라지는 현상을 방지하기 위해 각 컴포넌트들의 InitVisible 속성값을 반드시 False로 설정해야 함
설정 방법	 1. SuspendLayoutInterval = 1인 경우 InitVisible 속성값을 False로 설정(필수) 2. SuspendLayoutInterval = 0인 경우 InitVisible 속성값은 False 권장

CASE-2	공통 영역이 없으며, 특정 영역에 컴포넌트들의 위치와 크기가 같은 경우
설명	공통 영역이 없는 MDI 구성 시, 각 폼에 배치된 컴포넌트의 위치와 크기가 같은 경우 해당 컴포넌트들의 InitVisible 속성값을 False로 설정 시 화면 전환이 자연스럽게 됨 (SuspendLayoutInterval 속성값과 관계없음)
설정 방법	 1. 각 폼에 배치된 컴포넌트의 위치와 크기가 같은 경우 InitVisible 속성값은 False 권장 2. 각 폼에 배치된 컴포넌트의 위치와 크기가 다른 경우 InitVisible 속성값은 True 권장

참고 InitVisible 속성 적용 사용자 인터페이스 컴포넌트

SmartDraw, SmartButton, SmartLabel, SmartProgressBar, SmartUpDown, SmartCheckBox, SmartListBox, SmartGroupBox, SmartRadioButton, SmartSeparatorLine, SmartTrackBar

4) 모달 창(Modal Window) 출력 방법

SmartForm에서는 Modal Window로 폼을 출력하는 기능을 제공하고 있습니다. 여기서 Modal Window란 Dialog Window, Pop Window 등을 말합니다. SmartForm에서 Modal Window 사용 시 AddChildForm() 메소드로 추가하지 않는 특징이 있습니다. 일반적으로 Modal Window는 Blocking 처리 방식으로 동작하지만 SmartForm에서는 별도로 None-Blocking 형태의 Modal Window를 출력할 수 있는 기능을 추가로 지원하고 있습니다.

[표] 메시지 폼의 Blocking 출력 방식과 NoneBlocking 출력 방식의 비교

구분	Modal + Blocking	Modal + NoneBlocking
메소드	ShowDialog()	ShowDialog_NoneBlocking()
사용 시점	모달 창 출력이 필요한 경우	프로그램 및 작업 로딩 처리
특징	1. 모달 방식의 출력 2. 인스턴싱 없이 폼 출력 가능 3. Blocking 방식으로 출력되어 호출한 폼은 작업이 중지됨	1. 모달 방식의 출력 2. NoneBlocking 방식으로 출력되어 호출한 폼에서 Back Ground 처리 가능
적용 예		

C# 소스 제어 흐름	C# 소스 / 제어 흐름
	private void Form1_Load(object sender, EventArgs e)
	{
	// Alert를 모달(Modal) 방식으로 띄움
	// 부모 폼을 Blocking 방식으로 처리
	smartForm1.ShowDialog (typeof(Alert));
	// Alert창에서 선택 입력(확인, 취소)이
	// 발생되기 전까지 Return 되지 않아
	// 아래 Code는 대기 상태가 된다.
	Alert창에서 선택 입력(확인, 취소)이 발생하여 Return됨
	Application.DoEvents();
	frm2 = new Form2();
	frm3 = new Form3();
	smartForm1.AddChildForm(frm2);
	smartForm1.AddChildForm(frm3);
	smartForm1.Show(0);
	}
	Alert창 호출 / Close 대기 시간 / Close 후 Return
		C# 소스 / 제어 흐름
		private void Form1_Load(object sender, EventArgs e)
		{
		// Alert를 모달(Modal) 방식으로 띄움
		// 부모 폼을 NoneBlocking 방식으로 처리
		smartForm1.ShowDialog_NoneBlocking (typeof(Alert));
		// Alert창 호출 후 바로 Return하여
		// 아래의 코드가 진행된다.
		// Close 전까지 터치가 되지 않음(modal)
		Application.DoEvents();
		frm2 = new Form2();
		frm3 = new Form3();
		smartForm1.AddChildForm(frm2);
		smartForm1.AddChildForm(frm3);
		// 부모 폼에서 Alert창을 닫는다.
		smartForm1.CloseDialog_NoneBlocking();
		smartForm1.Show(0);
		}
		Alert창 호출 / 바로 Return / Alert창 닫힘

VB 소스 제어 흐름	VB.NET 소스 / 제어 흐름 Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load ' 부모 폼의 작업 처리가 중지 ' Alert를 모달(Modal) 방식으로 띄움 ' 부모 폼을 Blocking 방식으로 처리 smartForm1.ShowDialog (GetType(Alert)) ' Alert창에서 선택 입력(확인, 취소)이 ' 발생되기 전까지 Return 되지 않아 아래 Code는 ' 대기 상태가 된다. Alert창에서 선택 입력(확인, 취소)이 발생하여 Return됨 Application.DoEvents() frm2 = New Form2() frm3 = New Form3() smartForm1.AddChildForm(frm2) smartForm1.AddChildForm(frm3) smartForm1.Show(0) End Sub Alert창 호출 / Close 대기 시간 / Close 후 Return	VB.NET 소스 / 제어 흐름 Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load ' Alert를 모달(Modal) 방식으로 띄움 ' 부모 폼을 NoneBlocking 방식으로 처리 smartForm1.ShowDialog_NoneBlocking (GetType(Alert)) ' Alert창 호출 후 바로 Return하여 ' 아래의 코드가 진행된다. ' Close 전까지 터치가 되지 않음(modal) Application.DoEvents() frm2 = New Form2() frm3 = New Form3() smartForm1.AddChildForm(frm2) smartForm1.AddChildForm(frm3) ' 부모 폼에서 Alert창을 닫는다. smartForm1.CloseDialog_NoneBlocking() smartForm1.Show(0) End Sub Alert창 호출 / 바로 Return / Alert창 닫힘

5) 프로그래밍 적용 가이드

CASE-1 MDI 구조로 사용하기

SmartForm을 사용한 MDI 구조는 SmartForm-SmartForm 구조와 SmartForm-SmartInnerForm 구조가 있습니다. 아래 CASE를 확인하여 프로그램에 적용하시기 바랍니다.

※ 자세한 내용은 "SmartForm을 이용한 MDI 조합 구성"을 참고하시기 바랍니다.

[CASE-1-1] SmartForm-SmartForm 구조의 MDI인 경우

MainForm 속성으로 메인 폼으로 사용할 폼을 지정합니다. 그 다음 자식 폼으로 추가할 폼들을 생성 후 AddChildForm() 메소드로 자식 폼을 추가합니다. 자식 폼 추가가 완료되면 Show() 메소드의 인자값을 설정해 출력할 폼을 선택할 수 있습니다.

※ 자세한 내용은 "MainForm 속성", "AddChildForm(), Show() 메소드"를 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  // 자식 폼을 생성합니다.
  childForm1 chfrm1 = new childForm1();  childForm2 chfrm2 = new childForm2();
  // Main 폼에서 자신이 Main 폼임을 설정 후 Main 폼에 자식 폼을 추가합니다.
  smartForm1.MainForm = this;
  // 추가한 순서대로 Index가 정해지며, Show(Index) 또는 ChildForms에서 사용됩니다.
  smartForm1.AddChildForm(chfrm1);   smartForm1.AddChildForm(chfrm2);
  smartForm1.Show(1);  // 인덱스를 설정해 화면에 출력할 폼을 선택합니다.
}
```

[CASE-1-2] SmartForm-SmartInnerForm 구조의 MDI인 경우

MainForm 속성으로 메인 폼으로 사용할 폼을 지정합니다. 그다음 자식 폼으로 추가할 SmartInnerForm을 AddChildForm() 메소드를 사용해 추가합니다. 자식 폼 추가가 완료되면 Show() 메소드의 인자값을 설정해 출력할 폼을 선택할 수 있습니다.

※ 자세한 내용은 "MainForm 속성", "AddChildForm(), Show() 메소드"를 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  smartForm1.MainForm = this;  // Main 폼에서 자신이 Main 폼임을 설정합니다.
  // Main 폼에 자식 폼을 추가합니다.
  // 추가한 순서대로 Index가 정해지며, Show(Index) 또는 ChildForms에서 사용됩니다.
  smartForm1.AddChildForm(smartInnerForm1);  smartForm1.AddChildForm(smartInnerForm2);
  smartForm1.Show(1);  // 인덱스를 설정해 화면에 출력할 폼을 선택합니다.
}
```

CASE-2 모달 창으로 사용하기

SmartForm은 모달 창을 출력하는 기능을 제공하고 있으며, Blocking 출력 방식과 None-Blocking 출력 방식이 있습니다.

※ 자세한 내용은 "모달 창(Modal Window) 출력 방법"을 참고하시기 바랍니다.

[CASE-2-1] Blocking 방식으로 출력하기

Blocking 방식으로 출력하는 경우 출력 시점에서 코드의 진행이 멈추며, 출력된 폼이 닫히면 다음 코드가 진행되는 방식입니다. ShowDialog() 메소드를 사용해 출력할 수 있으며, DialogResult를 리턴하여 메시지 폼 용도로 사용할 수 있습니다.

※ 자세한 내용은 "ShowDialog() 메소드"를 참고하시기 바랍니다.

```
// FormMessage 폼을 모달 방식으로 화면 중앙에 출력합니다.
DialogResult dResult = smartForm1.ShowDialog(typeof(FormMessage));
if (dResult == DialogResult.Yes) { /* Yes 입력 시 처리 코드 작성 */ }
```

[CASE-2-2] None-Blocking 방식으로 출력하기

None-Blocking 방식으로 출력하는 경우 폼이 출력 돼도 다음 코드가 멈추지 않고 진행되는 방식입니다. Show Dialog_NoneBlocking() 메소드를 사용해 출력할 수 있으며, CloseDialog_NoneBlocking() 메소드를 호출해 출력시킨 None-Blocking 폼을 닫을 수 있습니다.

※ 자세한 내용은 "ShowDialog_NoneBlocking(), CloseDialog_NoneBlocking() 메소드"를 참고하시기 바랍니다.

```
// FormMessage 폼을 NoneBlocking-모달 방식으로 화면 중앙에 출력합니다.
smartForm1.ShowDialog_NoneBlocking(typeof(FormMessage));
// ...배경 처리 코드...
smartForm1.CloseDialog_NoneBlocking();  // 모달 창을 닫습니다.
```

6) SmartForm 인터페이스 설명

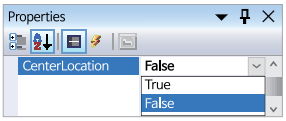
SmartForm Component Interface		
속성		
CenterLocation : bool	ChildForms : List<Form>	Image : Image
LCDDirection : SmartForm.LCDDIRECTIONS	LCDSize : SmartForm.LCDRESOLUTION	MainForm : Form
Mode : SmartForm.RUNMODE	MouseCursor : SmartForm . OnOff	SpecialFunctionClickPointSize : int
SuspendLayoutInterval : int		
메소드		
AddchildForm() : void (+1개 오버로드)	ChildFormLocation(int iLeft, int iTop): void	CloseDialog_NoneBlocking() : void
Show(int index) : void	ShowDialog(Type formDialogClass) : DialogResult (+1개 오버로드)	ShowDialog_NoneBlocking (Type formDialogClass) : void (+1개 오버로드)
이벤트		
OnEventFormChanged : EventFormChange	OnSpecialFunctionClick : EventHandler	OnTouchVerified : EventHandler

프로퍼티(속성) CenterLocation

폼의 크기가 화면 크기보다 작은 경우 폼의 출력 위치를 화면 중앙에 정렬하여 표시하도록 합니다.

- bool : 화면 중앙 정렬 여부
  - True : 화면 중앙에 정렬하여 표시
  - False : 기본 설정 위치에 표시

사용법

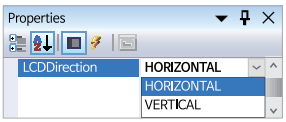


프로퍼티(속성) LCDDirection

LCD의 회전 기능으로 화면이 90도 회전할 LCDDirection 속성값을 변경하여 가로, 세로 방향을 설정합니다.

- SmartForm.LCDDIRECTIONS.HORIZONTAL : 화면을 수평 방향으로 설정합니다.
- SmartForm.LCDDIRECTIONS.VERTICAL : 화면을 수직 방향으로 설정합니다.

사용법



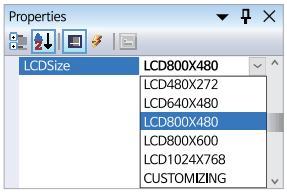
프로퍼티(속성) LCDSize

Form 크기를 제품 모델별로 편리하게 설정할 수 있습니다.

참고 SmartBootLogo의 "제품별 BootLogo 이미지 사양" 내용을 참고하시기 바랍니다.

- SmartForm.LCDRESOLUTION.LCD480X272 → 4.3 Inch
- SmartForm.LCDRESOLUTION.LCD640X480 → 5.6 Inch
- SmartForm.LCDRESOLUTION.LCD800X480 → 7, 10.2 Inch
- SmartForm.LCDRESOLUTION.LCD800X600 → 8, 10.4 Inch
- SmartForm.LCDRESOLUTION.LCD1024X768 → 10.4, 15 Inch
- SmartForm.LCDRESOLUTION.CUSTOMIZING → 사용자 정의 크기

사용법



속성창에서 설정을 권장합니다.
변경되는 크기에 따라 폼의 크기도 같이 변경됩니다.

프로퍼티(속성) Image

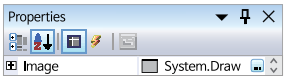
Image

배경 이미지를 설정합니다. 통상 LCD 해상도와 같은 사이즈의 이미지를 사용하여 배경을 설정합니다.

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법 속성창에서 설정하시면 편리하게 설정할 수 있습니다.



프로퍼티(속성) ChildForms

부모 폼에서 자식 폼들을 참조할 수 있습니다. 자식 폼의 수는 여러 개일 수 있으므로 배열과 같이 Index를 사용합니다.

참고 추가한 자식 폼의 Index는 "AddChildForm() 메소드"를 참고하시기 바랍니다.

주의 ChildForms Index 순서

ChildForms의 Index는 AddChildForm() 메소드로 자식 폼을 추가한 순서대로 설정됩니다.
Show() 메소드에서는 Index 0은 부모 폼이고 1부터 자식 폼 Index가 시작됐지만, ChildForms 속성에서는 Index 0부터 자식 폼 Index가 시작됩니다.

C# 사용법

```
// childForm1에서 smartButton1의 접근 한정자는 Public이어야 합니다.
((childForm1)(smartForm1.ChildForms[0])).smartButton1.Text = " 111 ";
```

VB 사용법

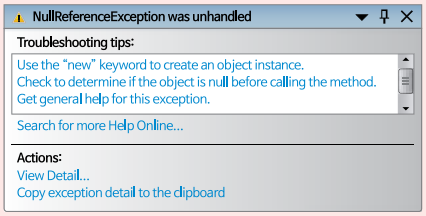
```
DirectCast(smartForm1.ChildForms(0), childForm1).smartButton1.Text = " 111 "
```

프로퍼티(속성) MainForm

다중 창(MDI) 구현 시 부모 폼에서 자신이 부모 폼임을 설정합니다. 반드시 부모 폼에서만 설정합니다.

주의 MainForm 설정 생략 시 폼을 Show 하는 부분에서 NullReferenceException이 발생합니다.

```
//smartForm1.MainForm = this;
smartForm1.AddChildForm(m_frm2);
smartForm1.Show(0);
```



C# 사용법

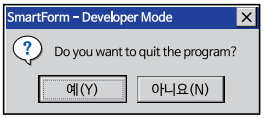
```
// Main 폼에서 자신이 Main 폼임을 설정합니다.
smartForm1.MainForm = this;
```

VB 사용법

```
smartForm1.MainForm = Me
```

프로퍼티(속성) Mode

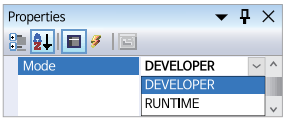
장치 응용 프로그램 개발 시 프로그램의 종료를 편리하게 할 수 있도록 합니다. 종료 방법은 Smart Form에서 다른 컨트롤이 없는 공간을 더블 클릭하면 종료 확인 메시지가 출력되어 프로그램을 쉽게 종료할 수 있습니다.



중요

- SmartForm.Runmode.DEVELOPER : 개발 모드로 더블 클릭 시 프로그램 종료합니다. 제품 개발이 완료되고 생산되는 시점에서는 시스템 보안상 RUNTIME 모드로 사용해야 합니다.
- SmartForm.Runmode.RUNTIME : 프로그램 종료 기능을 사용하지 않습니다.

C# 사용법

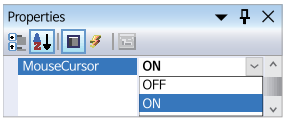


프로퍼티(속성) MouseCursor

마우스 커서를 숨김 처리합니다. 시스템 전역적으로 처리됩니다.

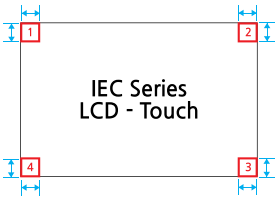
- SmartForm.OnOff.ON : 마우스 커서를 보이게 합니다.
- SmartForm.OnOff.OFF : 마우스 커서를 숨깁니다.

사용법



프로퍼티(속성) SpecialFunctionClickPointSize

환경 설정 기능과 같이 특수 기능의 접근을 하기 위한 터치 입력 기능 사용 시 입력 영역 크기(가로, 세로, 크기)를 설정합니다. 기본 설정값은 100Pixel입니다.



• int : 특수 입력 터치 영역 크기 (기본값 : 100, 단위 : Pixel)

C# 사용법

```
smartForm1.SpecialFunctionClickPointSize = 50;
```

VB 사용법

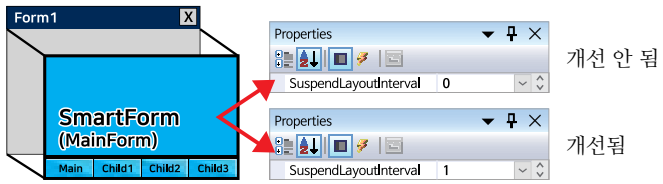
```
smartForm1.SpecialFunctionClickPointSize = 50
```

프로퍼티(속성) SuspendLayoutInterval

MDI 프로그램에서 화면 전환 시 발생하는 잔상 및 겹침(Overlap) 현상을 개선하기 위한 속성으로, 반드시 부모 폼(SmartForm)에서 설정해야 합니다. ※ Main Form에서만 설정합니다.

참고 "화면 전환 잔상 및 겹침 현상 개선 방법"을 참고하시기 바랍니다.

사용법



메소드(함수) AddChildForm

부모 폼에서 다른 폼을 자식 폼으로 설정합니다. 추가되는 순서에 의해 Index가 추가됩니다.

[표] MDI 구성 시 코드 호출 순서에 따른 Show() 메소드와 ChildForms 속성의 Index

호출 순서	Show()	ChildForms
smartForm1.MainForm = this;	0	-
smartForm1.AddChildForm(chfrm1);	1	0
smartForm1.AddChildForm(chfrm2);	2	1
smartForm1.AddChildForm(chfrm3);	3	2

주의 AddChildForm() 메소드 호출 순서

반드시 MainForm 속성을 설정 후 AddChildForm() 메소드를 호출하시기 바랍니다.

올바른 사용 예	잘못된 사용 예

//부모 폼에서 자신이 부모 폼임을 설정합니다.
smartFrom1.MainForm = this;
//부모 폼 설정 이후 자식 폼을 추가합니다.
smartForm1.AddChildForm(chfrm1);
	//부모 폼 설정이전 자식 폼을 추가합니다.
	smartForm1.AddChildForm(chfrm1);
	//부모 폼에서 자신이 부모 폼임을 설정합니다.
	smartFrom1.MainForm = this;

◉ void AddChildForm(Form childForm)
◉ void AddChildForm(SmartInnerForm childForm)

[인자]

• Form childForm : SmartForm-SmartForm 구조의 MDI일 때 사용하며, SmartForm이 배치된 폼을 인자로 설정합니다.

• SmartInnerForm childForm : SmartForm-SmartInnerForm 구조의 MDI일 때 사용하며, 자식 폼으로 추가할 SmartInnerForm을 인자로 설정합니다.

참고 SmartInnerForm의 사용 방법은 SmartInnerForm 부분을 참고하시기 바랍니다.

C# 사용법

```
// 자식 폼을 생성 후 Main 폼에 자식 폼을 추가합니다.
childForm1 chfrm1 = new childForm1();
childForm2 chfrm2 = new childForm2();
smartForm1.MainForm = this;
smartForm1.AddChildForm(chfrm1);
smartForm1.AddChildForm(chfrm2);
```

VB 사용법

```
Dim chfrm1 As childForm1 = New childForm1()
Dim chfrm2 As childForm2 = New childForm2()
smartForm1.MainForm = Me
smartForm1.AddChildForm(chfrm1)
smartForm1.AddChildForm(chfrm2)
```

메소드(함수) ChildFormLocation

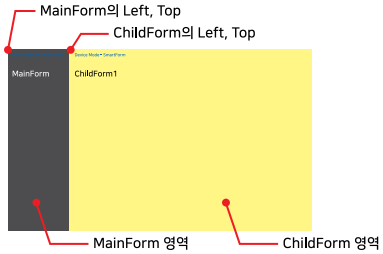
모든 자식 폼의 표시되는 위치를 부모 폼의 기준으로 설정합니다. 아래 그림처럼 ChildForm이 MainForm 기준으로 일정 거리가 떨어진 원하는 지점에서 표시되도록 할 수 있습니다.

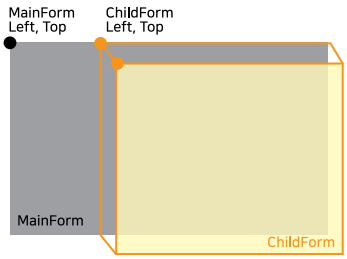
◉ void ChildFormLocation(int iLeft, int iTop)

[인자]

• int iLeft : 부모 폼을 기준으로 좌측부터 Offset

• int iTop : 부모 폼을 기준으로 위부터 Offset





C# 사용법

```
// 자식 폼의 위치를 부모 폼 기준으로 Left = 100, Top = 0으로 Offset합니다.
smartForm1.ChildFormLocation(100, 0);
```

VB 사용법

```
smartForm1.ChildFormLocation(100, 0)
```

메소드(함수) Show

자식 폼의 표시를 제어합니다. Index는 0(부모 폼)부터 시작되며 자식 폼을 추가한 순서대로 자식 폼에 Index가 부여됩니다.

주의 반드시 smartForm1.Show() 메소드를 호출하여 최초에 표시되는 폼을 설정하시기 바랍니다.

참고 Index에 대한 자세한 설명은 "AddChildForm() 메소드"를 참고하시기 바랍니다.

◉ void Show(int index)

[인자]

• int index : 자식 폼의 순서

C# 사용법

```
// 부모 폼에서의 코드. Index가 1번인 자식 폼 표시
smartForm1.Show(1);
// 자식 폼에서 index가 1번인 자식 폼 표시
((MainForm)(Owner)).smartForm1.Show(1);
```

VB 사용법

```
smartForm1.Show(1)
DirectCast((Owner), MainForm).smartForm1.Show(1)
```

메소드(함수) ShowDialog

자식 폼으로 추가되지 않은 폼을 모달(Modal)/PopUp 창 방식으로 호출합니다. DialogResult를 Return하여 메시지 폼 용도로 사용하실 수 있습니다.

참고 "모달 창(Modal Windows) 출력 방법"을 참고하시기 바랍니다.

◉ DialogResult ShowDialog(Type formDialogClass)
◉ DialogResult ShowDialog(Type formDialogClass, int iLeft, int iTop)

[인자]

• Type formDialogClass : 모달 방식으로 출력할 폼의 타입
• int iLeft : 폼이 출력될 X축 위치를 설정(생략 시 화면 중앙에 출력)
• int iTop : 폼이 출력될 Y축 위치를 설정(생략 시 화면 중앙에 출력)

C# 사용법

```
// FormMessage 폼을 모달 방식으로 화면 중앙에 출력합니다.
DialogResult dResult = smartForm1.ShowDialog(typeof(FormMessage));
if (dResult == DialogResult.Yes)
{
  // Yes 입력
}
```

VB 사용법

```
Dim dResult As DialogResult = smartForm1.ShowDialog(GetType(FormMessage))
If dResult = DialogResult.Yes Then
End If
```

메소드(함수) CloseDialog_NoneBlocking

ShowDialog_NoneBlocking() 메소드로 출력한 NoneBlocking 방식의 모달 창을 닫습니다.

참고 "모달 창(Modal Windows) 출력 방법"을 참고하시기 바랍니다.

사용법

참고 "ShowDialog_NoneBlocking() 메소드"를 참고하시기 바랍니다.

메소드(함수) ShowDialog_NoneBlocking

다이얼로그 창을 모달(Modal) 방식으로 띄우고 NoneBlocking 방식으로 처리하여 다이얼로그 창이 오픈된 동안에도 부모 폼의 작업 처리가 진행됩니다. 다이얼로그 창을 띄운 후 CloseDialog_NoneBlocking() 메소드를 호출하거나 호출된 창에서도 닫기 처리를 하실 수 있습니다.

주의 프로그램 초기 로딩에 사용하는 경우 다이얼로그 창 이외의 영역 클릭 시 전체 폼의 로딩 완료 후 클릭 이벤트(Event)가 처리되는 현상이 발생합니다.

참고 "모달 창(Modal Windows) 출력 방법"을 참고하시기 바랍니다.

◉ `void ShowDialog_NoneBlocking(Type formDialogClass)`
◉ `void ShowDialog_NoneBlocking(Type formDialogClass, int iLeft, int iTop)`

[인자]

- `Type formDialogClass` : NoneBlocking-Modal 방식으로 출력할 폼의 타입
- `int iLeft` : 폼이 출력될 X축 위치를 설정(생략 시 화면 중앙에 출력)
- `int iTop` : 폼이 출력될 Y축 위치를 설정(생략 시 화면 중앙에 출력)

C# 사용법

```
// FormMessage 폼을 NoneBlocking-모달 방식으로 화면 중앙에 출력합니다.
smartForm1.ShowDialog_NoneBlocking(typeof(FormMessage));
// 모달 창을 닫습니다.
smartForm1.CloseDialog_NoneBlocking();
```

VB 사용법

```
smartForm1.ShowDialog(GetType(FormMessage))
smartForm1.CloseDialog_NoneBlocking()
```

이벤트 OnEventFormChanged

SmartForm - SmartForm 또는 SmartForm - SmartInnerForm MDI 조합 구현 시 화면 전환 후 발생되는 이벤트로, OnEventFormChanged 이벤트는 SmartForm에서 Main으로 설정된 Form에서만 발생되는 이벤트입니다.
OnEventFormChanged 이벤트를 사용하면 SuspendLayoutInterval 속성값이 1일 때 Enabled/Visible 속성값이 임의로 변경되는 문제를 개선할 수 있습니다.

참고 "화면 전환 잔상 및 겹침 현상 개선 방법"을 참고하시기 바랍니다.

참고 Index에 대한 자세한 설명은 "AddChildForm() 메소드"를 참고하시기 바랍니다.

[인자]

- `int iFormIndex` : 폼의 Index 값을 얻을 수 있습니다.

C# 사용법

```
private void smartForm1_OnEventFormChanged(int iFormIndex)
{
  // 사용자 인터페이스 컴포넌트의 Visible, Enable 속성을 True로 설정합니다.
  smartTrackBar2.Visible = true;
  smartbutton2.Enabled = true;
}
```

VB 사용법

```
Private Sub smartForm1_OnEventFormChanged(ByVal iFormIndex As System.Int32) Handles
smartForm1.OnEventFormChanged
  smartTrackBar2.Visible = True
  smartbutton2.Enabled = True
End Sub
```

이벤트 OnSpecialFunctionClick

환경 설정 기능과 같이 특수 기능의 접근을 하기 위한 터치 입력 시 발생되는 이벤트로, 사용자 시스템의 Maintainance 관련 작업(File 복사, 시스템 설정 변경 등)을 할 경우에 유용하게 사용할 수 있습니다.

※ [1 → 2 → 3 → 4 → 1] 순서로 화면을 터치하면 이벤트가 발생됩니다.

C# 사용법

```
private void smartForm1_OnSpecialFunctionClick(object sender, EventArgs e)
{
  // 패스워드 입력을 위한 창 출력
  smartForm1.ShowDialog(typeof(FormSpecialFunc));
}
```

VB 사용법

```
Private Sub smartForm1_OnSpecialFunctionClick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles smartForm1.OnSpecialFunctionClick
  smartForm1.ShowDialog(GetType(FormSpecialFunc))
End Sub
```

주의 OnSpecialFunctionClick 이벤트 주의사항

1. OnSpecialFunctionClick 이벤트 코드에서 폼 이동 시 주의사항

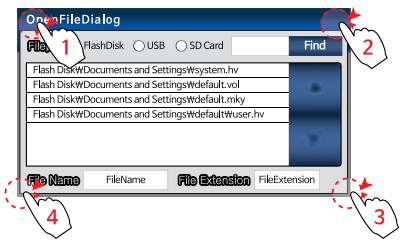
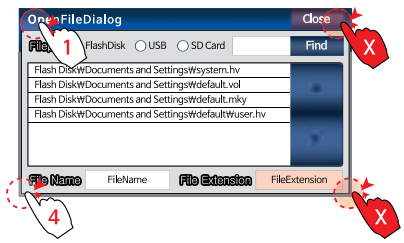
폼의 Form_Load에서 다른 폼을 참조하거나 복잡한 소스 코드의 처리를 하는 경우, 메시지 큐의 메시지가 즉시 처리되지 않아 사용자가 폼을 터치하여도 응용 프로그램이 응답하지 않는 문제가 발생합니다. 이때 Application.DoEvents()를 호출하면 현재 메시지 큐의 모든 메시지를 처리하여 응용 프로그램이 응답 되도록 합니다.

[C#] 예시 코드

```
// 부모 폼의 소스 코드(C#) : OnSpecialFunctionClick 이벤트를 사용하여 자식 폼3을 호출
private void smartForm1_OnSpecialFunctionClick(object sender, EventArgs e)
{
  smartForm1.Show(3);
}

// 자식 폼3의 Form_Load 부분. 부모 폼에서 smartForm1.Show(3); 하는 경우 한 번만 호출됨
private void childForm3_Load(object sender, EventArgs e)
{
  // 복잡한 소스 코드의 처리 또는 다른 폼을 참조. 다른 코드 생략됨
  // 마지막에 Application.DoEvents() 메서드 호출
  Application.DoEvents();
}
```

2. OnSpecialFunctionClick 이벤트 설정 시 컨트롤 배치 주의사항

다른 컴포넌트(터치 이벤트) 부분이 겹치지 않는 영역에 적용해 주시기 바랍니다.

컨트롤 배치가 잘된 예	컨트롤 배치가 잘못된 예
	

※ 터치 부분의 이벤트(button, TextBox 등)가 겹칠 경우 동작이 안 됩니다.

이벤트	OnTouchVerified

프로그램 로딩 시 비정상적인 터치의 입력을 감지하여 이벤트를 발생합니다.
터치가 눌려있거나 파손된 경우 진단 기능으로 사용하시면 됩니다. 만약 프로그램 로딩 중 터치를 누르고 있을 경우 이벤트가 발생됩니다.

C# 사용법

```
// 부팅 시 비정상적인 터치 입력 감지 시 발생하는 이벤트
private void smartForm1_OnTouchVerified(object sender, EventArgs e)
{
  // 부팅 시 최초 터치 입력 상태 확인
  MessageBox.Show( " Touch Error.!!! " );
}
```

VB 사용법

```
Private Sub smartForm1_OnTouchVerified(ByVal sender As System.Object, ByVal e As System.
EventArgs) Handles smartForm1.OnTouchVerified
  MessageBox.Show( " Touch Error.!!! " )
End Sub
```

7) SmartForm 예제 사용하기

SmartForm을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartForm"

2. SmartInnerForm

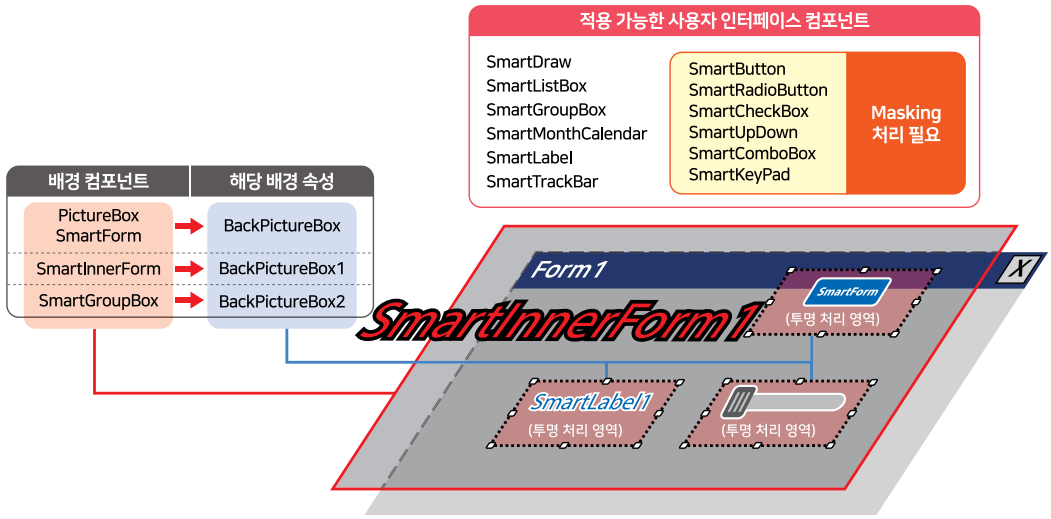
SmartInnerForm은 SmartForm과 연동하여 간편하게 MDI(Multiple Document Interface) 프로그램을 구현하기 위한 컴포넌트입니다. MDI 구조는 SmartForm이 메인 폼 역할을 하고 SmartInnerForm이 자식 폼이 되는 구조로 구성되며, SmartForm만을 이용한 MDI 구조보다 구성 및 데이터의 공유가 간편하다는 장점이 있습니다. 또한, 화면 디자인 시 이미지 적용 및 컴포넌트들의 투명 처리를 지원하여 사용자가 원하는 깔끔하고 다양한 디자인을 적용할 수 있습니다.

- 폼간 데이터 및 처리의 공유가 간편함
- 배경 이미지 설정 기능
- 투명 효과 지원되는 사용자 인터페이스 컴포넌트와 연동
- 다중 창(MDI) 기능 지원
- 공통 영역 표시 기능
- SmartForm과 연동
- 화면 잔상 개선 기능
- 모델별 편리한 화면 크기 지정 기능(속성창에서 간편하게 설정)
- RUNTIME 시 불필요한 폼 속성값 변경
- 디버그 모드 시 편리한 프로그램 종료 기능
- 장치 응용 프로그램 개발 모드 시 작업 표시줄 자동 숨김 기능
  ↳ 작업 표시줄이 폼의 하단 영역을 가리는 경우 자동 숨김 처리
- 시스템 설정을 위한 특수 터치 이벤트 처리 기능

중요 반드시 SmartForm 컴포넌트와 함께 사용해야 하며, 자세한 내용은 “SmartForm 컴포넌트”를 참고하시기 바랍니다.

1) 사용자 인터페이스 투명 처리 조합

SmartX Framework 컴포넌트 중 사용자 인터페이스(User Interface) 관련 컴포넌트들은 서로의 조합에 따라서 투명 효과를 처리할 수 있습니다. 다음과 같이 배경 컴포넌트와 배경 위에서 사용되는 컴포넌트들로 구성되며 적용은 아래 이미지의 배경 컴포넌트의 속성값을 설정하시면 구현할 수 있습니다.



참고

1. 배경 컴포넌트의 배경 이미지가 없고 BackColor 속성만 설정된 경우 투명 효과 기능을 사용하지 않고 사용자 인터페이스 컴포넌트의 BackColor 속성을 배경 컴포넌트와 같은 색상으로 설정하시기 바랍니다.
2. “투명(Masking) 처리 영역 제작 가이드”를 참고하시기 바랍니다.

2) SmartForm을 이용한 MDI 조합 구성

2-1) SmartForm-SmartForm MDI 조합과 SmartForm-SmartInnerForm MDI 조합의 차이점

SmartForm을 이용해 MDI 조합 구성 시 자식 폼이 SmartForm인지 SmartInnerForm인지에 따라 크게 두 가지 방법으로 나눌 수 있습니다. 두 방법 모두 부모 폼은 반드시 SmartForm이어야 하며, 자세한 차이점은 아래 표를 확인하여 용도에 맞게 프로젝트에 적용하시기 바랍니다.

권장 Case에 따른 MDI 조합 선택 기준

CASE	내용	권장 MDI 조합
1	- 폼이 많은 경우 - 각 폼에 컴포넌트가 많이 배치되는 경우	SmartForm-SmartForm
2	- 각 폼 간 데이터의 공유가 잦은 경우	SmartForm-SmartInnerForm

[표] SmartForm-SmartForm MDI 조합과 SmartForm-SmartInnerForm MDI 조합의 차이점

MDI 조합	SmartForm-SmartForm	SmartForm-SmartInnerForm
부모 폼	SmartForm	
자식 폼	SmartForm	SmartInnerForm
물리적인 폼 수	N개	1개
장점	1. 폼 디자인 작업이 SmartInnerForm 조합에 비해 편함 2. 각 폼이 물리적으로 분리되어 Visual Studio의 디자인 모드에서 폼 갱신(Reload) 시간이 SmartInnerForm 조합보다 짧음 (SmartX와 상관없이 Visual Studio에서 발생)	1. 폼 간 컨트롤 객체 접근 및 데이터 공유가 SmartForm 조합에 비해 편함
단점	1. 폼간 컨트롤 객체 접근 및 데이터 공유가 SmartInnerForm 조합에 비해 불편함	1. 폼 디자인 작업이 SmartForm 조합에 비해 불편함 2. 각 폼이 하나의 물리적인 폼에 구성되어 Visual Studio의 디자인 모드에서 폼 갱신(Reload) 시간이 SmartForm 조합보다 김 (SmartX와 상관없이 Visual Studio에서 발생)

참조 폼 디자인의 컴포넌트 개수에 따른 디자인 창에서 화면 갱신 시간 안내

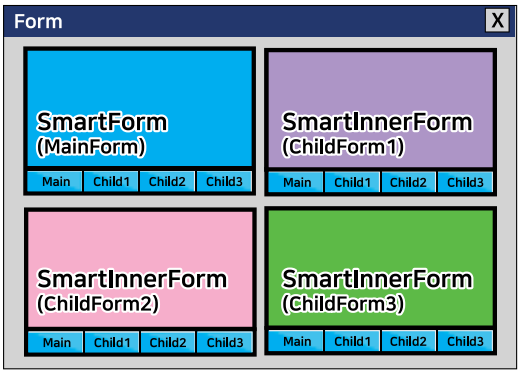
Visual Studio를 사용하여 C# 또는 VB.NET 용 프로젝트를 생성하여 디자인 창에서 컴포넌트를 사용하는 경우 프로젝트의 소스 코드를 수정하게 되면 디자인 창을 갱신(Reload)합니다. 사용되는 컴포넌트의 개수에 비례하여 화면 갱신 시간이 늘어나게 되며 그 이외 다른 문제점이 발생하지는 않습니다. 자세한 내용은 아래 경로에서 확인하시기 바랍니다.

※ 홈페이지(www.hnsts.co.kr) → 커뮤니티 → 공지사항 → 100. 폼 디자인의 컴포넌트 개수에 따른 디자인 창에서 화면 갱신 시간 안내

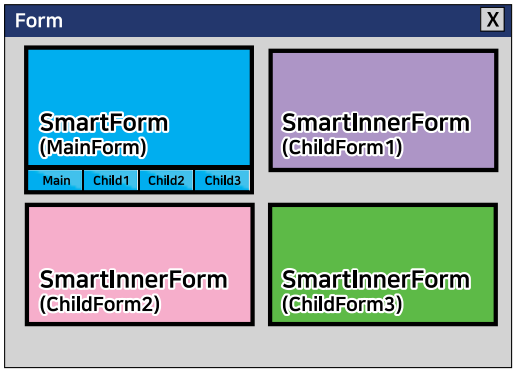
2-2) SmartForm-SmartInnerForm MDI 조합의 형태

SmartInnerForm을 이용하여 구현할 수 있는 MDI 조합의 형태는 크게 두 가지입니다. 아래 CASE를 확인하여 프로젝트에 적용해 보시기 바랍니다.

CASE-1	SmartForm(부모 폼)과 SmartInnerForm(자식 폼)의 크기가 같은 경우
사용 시점	각 폼에 공통 영역이 없는 경우



CASE-2	SmartForm(부모 폼)과 SmartInnerForm(자식 폼)의 크기가 같은 경우 (각 자식 폼의 크기는 동일)
사용 시점	각 폼에 공통 영역이 있는 경우



참고 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

참조 MDI 구성 시 ChildForm의 크기를 각각 다르게 구성하는 방법

Main Form과 ChildForm의 크기를 각각 다르게 하는 방법은 Tech Note에 관련 내용이 개재되어 있으니 참조하시기 바랍니다.

※ 홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 6. SmartForm과 SmartInnerForm으로 만들 수 있는 MDI 구성 조합

3) 화면 전환 잔상 및 겹침 현상 개선 방법

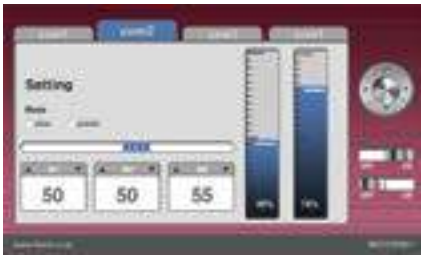
MDI(Multiple Document Interface) 구조는 화면 전환 시 잔상 및 겹침(Overlap) 현상이 발생할 수 있습니다.
이 현상은 SmartX 컴포넌트뿐만 아니라 .Net Compact Framework에서 제공하는 사용자 인터페이스에서도 동일하게 발생되며 사용되는 컨트롤의 수와 배경 이미지에 따라서 쉽게 나타날 수 있는 현상입니다.
SmartX Framework의 SmartForm을 사용하여 MDI 구성 후 SuspendLayoutInterval 속성값을 1로 설정하면 잔상 및 겹침 현상을 개선할 수 있습니다.

3-1) SuspendLayoutInterval 속성값에 따른 장단점 및 사용 시점

중요 SuspendLayoutInterval 속성은 SmartForm 컴포넌트의 기능으로 반드시 SmartForm에서 값을 설정하시기 바랍니다.

MDI 구조에서 화면 전환 시 SuspendLayoutInterval 속성값에 따른 차이점이 있으며, 아래 표에서 장단점을 확인하여 프로젝트에 적용하시기 바랍니다.

[표] SuspendLayoutInterval 속성값에 따른 장단점 및 사용 시점

속성값	SuspendLayoutInterval = 1 권장	SuspendLayoutInterval = 0
장점	화면 전환 시 컨트롤 잔상 및 겹침 현상 방지	SuspendLayoutInterval = 1 일 때 보다 화면 전환 속도가 상대적으로 빠름
단점	1. SuspendLayoutInterval = 0 일 때 보다 폼 전환 속도에 미세한 차이가 발생됨 2. 화면 전환 시 사용자 인터페이스 컨트롤의 Visible, Enabled 속성이 임의로 변경됨	화면 전환 시 컨트롤 잔상 및 겹침 현상 발생
권장 사용 시점	- 각 화면들이 다른 색 또는 이미지 사용 - 사용자 인터페이스 컴포넌트 위치가 다름	- 각 화면들이 동일 색 또는 이미지 사용 - 사용자 인터페이스 컴포넌트 위치가 동일
사용자 인터페이스 예시	※ 폼 간 유사성 낮음  [부모 폼]  [자식 폼]	※ 폼 간 유사성 높음  [부모 폼]  [자식 폼]

※ SuspendLayoutInterval 속성은 반드시 부모 폼의 SmartForm에서 설정해야 합니다.

주의 SuspendLayoutInterval 속성값을 1로 할 경우 주의사항

MDI 프로그램에서 SmartForm의 속성 중 SuspendLayoutInterval 속성값이 1일 경우 화면 전환 시 사용자 인터페이스 컴포넌트의 Visible, Enabled 속성값이 임의로 변경되는 문제가 발생합니다. 이러한 경우 SmartForm의 OnEventFormChanged 이벤트를 활용하시면 위 문제를 해결할 수 있습니다.

참조 화면 전환 시 잔상 및 겹침 현상(Overlap) 관련 공지사항

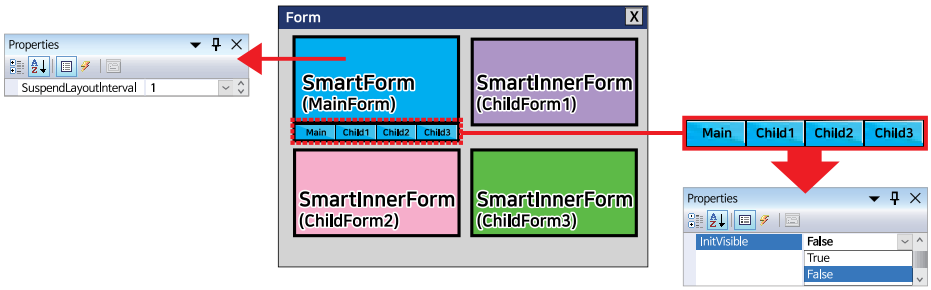
화면 전환 시 발생하는 잔상 및 겹침 현상에 대한 자세한 내용은 공지사항을 참조하시기 바랍니다.

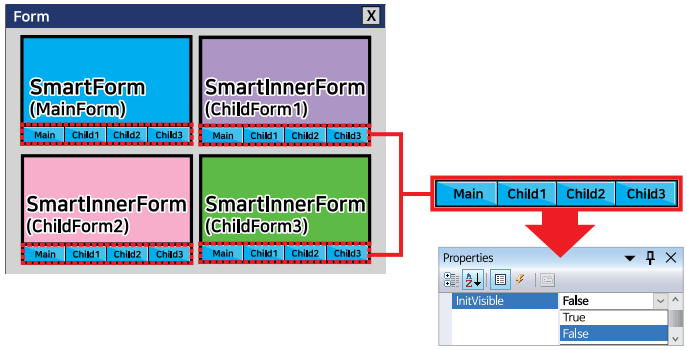
※ 홈페이지(www.hnsts.co.kr) → 공지사항 → 79. 화면(Form)전환 시 잔상 및 겹침(Overlap)현상 최소화 기능 추가

3-2) 공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정

공통 영역이 있는 MDI를 구성할 때 공통 영역에 위치한 컴포넌트의 InitVisible 속성이 True이고 부모 폼 (SmartForm)의 SuspendLayoutInterval 속성값이 1이라면 해당 컴포넌트는 화면 전환 시 사라지게 됩니다. 이 현상을 방지하기 위해서는 공통 영역에 위치한 컴포넌트들의 InitVisible 속성값을 반드시 False로 설정해야 합니다. 아래 표를 확인하여 상황에 맞게 적용하시기 바랍니다.

[표] SmartForm-SmartInnerForm MDI 구성일 경우 공통 영역 여부에 따른 InitVisible 속성값 설정

CASE-1	공통 영역이 있으며, 공통 영역에 컴포넌트가 위치한 경우
설명	공통 영역이 있는 MDI 구성 시, 공통 영역에 위치한 컴포넌트가 부모 폼에 위치한 SmartForm의 SuspendLayoutInterval 속성값이 1일 때 공통 영역에 위치한 컴포넌트들이 사라지는 현상을 방지하기 위해 각 컴포넌트들의 InitVisible 속성값을 반드시 False로 설정해야 함
설정 방법	 1. SuspendLayoutInterval = 1인 경우 InitVisible 속성값을 False로 설정(필수) 2. SuspendLayoutInterval = 0인 경우 InitVisible 속성값은 False 권장

CASE-2	공통 영역이 없으며, 특정 영역에 컴포넌트들의 위치와 크기가 같은 경우
설명	공통 영역이 없는 MDI 구성 시, 각 폼에 배치된 컴포넌트의 위치와 크기가 같은 경우 해당 컴포넌트들의 InitVisible 속성값을 False로 설정 시 화면 전환이 자연스럽게 됨 (SuspendLayoutInterval 속성값과 관계없음)
설정 방법	 1. 각 폼에 배치된 컴포넌트의 위치와 크기가 같은 경우 InitVisible 속성값은 False 권장 2. 각 폼에 배치된 컴포넌트의 위치와 크기가 다른 경우 InitVisible 속성값은 True 권장

참고 InitVisible 속성 적용 사용자 인터페이스 컴포넌트

SmartDraw, SmartButton, SmartLabel, SmartProgressBar, SmartUpDown, SmartCheckBox, SmartListBox, SmartGroupBox, SmartRadioButton, SmartSeparatorLine, SmartTrackBar

4) SmartInnerForm 인터페이스 설명

SmartInnerForm Component Interface		
속성		
BackImage : Image	LCDDirection : SmartInnerForm.LCDDIRECTIONS	LCDSize : SmartInnerForm.LCDRESOLUTION
Mode : SmartInnerForm.RUNMODE	SpecialFunctionClickPointSize : int	
이벤트		
OnSpecialFunctionClick : EventHandler		

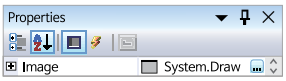
프로퍼티(속성) BackImage Image

배경 이미지를 설정합니다. 통상 LCD 해상도와 같은 사이즈의 이미지를 사용하여 배경을 설정합니다.

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법 속성창에서 설정하시면 편리하게 설정할 수 있습니다.

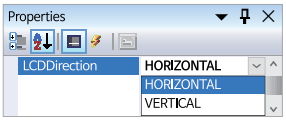


프로퍼티(속성) LCDDirection

LCD의 회전 기능으로 화면이 90도 회전할 LCDDirection 속성값을 변경하여 가로, 세로 방향을 설정합니다.

- SmartInnerForm.LCDDIRECTIONS.HORIZONTAL : 화면을 수평 방향으로 설정
- SmartInnerForm.LCDDIRECTIONS.VERTICAL : 화면을 수직 방향으로 설정

사용법



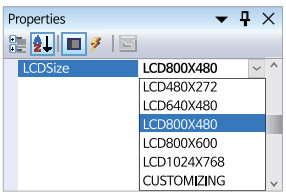
프로퍼티(속성) LCDSize

SmartInnerForm의 크기를 제품 모델별로 편리하게 설정할 수 있습니다.

참고 SmartBootLogo의 "제품별 BootLogo 이미지 사양" 내용을 참고하시기 바랍니다.

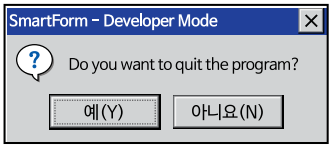
- SmartForm.LCDRESOLUTION.LCD480X272 → 4.3 Inch
- SmartForm.LCDRESOLUTION.LCD640X480 → 5.6 Inch
- SmartForm.LCDRESOLUTION.LCD800X480 → 7, 10.2 Inch
- SmartForm.LCDRESOLUTION.LCD800X600 → 8, 10.4 Inch
- SmartForm.LCDRESOLUTION.LCD1024X768 → 10.4, 15 Inch
- SmartForm.LCDRESOLUTION.CUSTOMIZING → 사용자 정의 크기

사용법



프로퍼티(속성) Mode

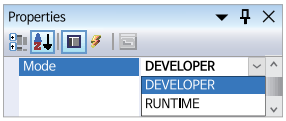
장치 응용 프로그램 개발 시 프로그램의 종료를 편리하게 할 수 있도록 합니다. 종료 방법은 SmartInnerForm에서 다른 컨트롤이 없는 공간을 더블 클릭하면 종료 확인 메시지가 출력되어 프로그램을 쉽게 종료할 수 있습니다.



중요

- SmartInnerForm.Runmode.DEVELOPER : 개발 모드로 더블 클릭 시 프로그램 종료합니다. 제품 개발이 완료되고 생산되는 시점에서는 시스템 보안상 RUNTIME 모드로 사용해야 합니다.
- SmartInnerForm.Runmode.RUNTIME : 프로그램 종료 기능을 사용하지 않습니다.

사용법



프로퍼티(속성) SpecialFunctionClickPointSize

환경 설정 기능과 같이 특수 기능의 접근을 하기 위한 터치 입력 기능 사용 시 입력 영역 크기(가로 세로 크기)를 설정합니다.

- int : 특수 입력 터치 영역 크기 (기본값 : 100, 단위 : Pixel)

C# 사용법

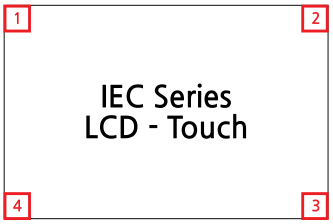
```
// 특수 입력 터치 영역 크기를 50 Pixel로 설정합니다.
smartInnerForm1.SpecialFunctionClickPointSize = 50;
```

VB 사용법

```
smartForm1.SpecialFunctionClickPointSize = 50
```

이벤트 OnSpecialFunctionClick

환경 설정 기능과 같이 특수 기능의 접근을 하기 위한 터치 입력 시 발생되는 이벤트로, 사용자 시스템의 Maintainance 관련 작업(File 복사, 시스템 설정 변경 등)을 할 경우에 유용하게 사용할 수 있습니다.
※ [1 → 2 → 3 → 4 → 1] 순서로 화면을 터치하면 이벤트가 발생됩니다.



사용법

참고 SmartForm의 OnSpecialFunctionClick 이벤트와 동일한 기능으로, “SmartForm” 내용을 참고하시기 바랍니다.

5) SmartInnerForm 예제 사용하기

SmartInnerForm을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

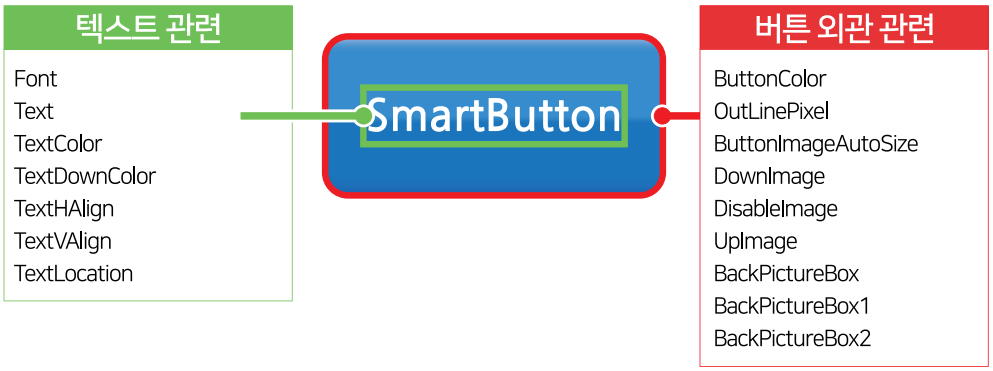
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → “SmarInnerForm1, 2”

3. SmartButton

SmartButton은 .NET Compact Framework에서 지원되는 Button 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 Button의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- ■ 단순 버튼 기능과 다양한 특수 기능(SpecialFunction) 버튼 지원
  - ㄴ 일반, 연속 입력, 안전 입력, 연속+안전 입력 기능 지원
- ■ 다양한 동작 모드 지원
  - ㄴ 일반 버튼, RADIO 버튼, PUSH 버튼 모드 지원
- ■ 버튼 이미지 설정 기능
- ■ 버튼 이미지 관련 투명 효과 지원
- ■ 버튼을 누르고 있을 경우(연속) 입력 가속 처리 기능 지원
- ■ 연속으로 버튼 동작 빠르게 입력 시 발생하는 Stack OverFlow 예외 방지 기능 지원

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

STEP-1 디자인하기

SmartButton은 색상 및 테두리를 설정하거나, 이미지를 사용해 버튼에 다양한 디자인을 적용할 수 있으며, 마우스를 사용해 버튼의 크기를 조절할 수 있습니다. 또한, 이미지를 사용하는 경우 버튼에 투명 효과를 적용할 수 있습니다. 버튼 디자인이 완료되면 버튼에 보여지는 텍스트를 시스템 상에서 출력하도록 하거나, 이미지에 포함시켜 표현할 수 있습니다.

※ 자세한 내용은 아래 CASE의 내용 중 표의 관련 속성 내용을 참고하시기 바랍니다.

[CASE-1] 이미지를 사용하지 않는 경우

[표1] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
ButtonColor	DeepSkyBlue	TextColor	Black
OutLinePixel	8	TextDownColor	DarkRed
Font	NanumBarunGothic, 14pt, style=Bold	TextHAlign	Middle
Text	"SmartButton"	TextVAlign	Middle

[표2] 관련 속성 적용에 따른 디자인 변화

디자인 적용 전	디자인 적용 후
	

[CASE-2] 이미지를 사용하는 경우

※ 본 가이드에서는 BackPictureBox 속성을 설정해 SmartButton 이미지에 마스킹 처리를 하여 투명 효과를 적용하였습니다.

[표1] 관련 속성의 속성값 예시

DownImage	DisableImage	UpImage
		

관련 속성	속성값	관련 속성	속성값
BackPictureBox	smartForm1(이미지 적용됨)	Text	“”(공백)

[표2] 관련 속성 적용에 따른 디자인 변화

디자인 적용 전	디자인 적용 후
	

STEP-2 특수 기능 적용하기

SmartButton은 단순 버튼 기능과 다양한 특수 기능(일반, 연속 입력, 안전 입력, 연속+안전 입력)을 지원합니다. 특수 기능은 SpecialFunction 속성에서 설정할 수 있으며, 특수 기능에 따라 RepeatInterval, SafeInterval 속성에서 입력 시간을 설정해야 합니다. 또한, 반복 입력 기능 사용 시 RepeatIntervalAccelerate 속성을 사용해 입력 시간을 가속시킬 수 있습니다.

※ 자세한 내용은 “SpecialFunction, RepeatInterval, SafeInterval, RepeatIntervalAcceler 속성”을 참고하시기 바랍니다.

[CASE-1] 연속 입력 및 가속 입력 처리 기능 사용하기

연속 입력 기능은 버튼을 여러 번 클릭해야 하는 경우 클릭을 여러 번할 필요 없이 버튼을 누르고 있으면 연속으로 클릭 이벤트가 발생합니다. 또한 RepeatIntervalAccelerate 속성을 사용해 입력 간격을 가속시킬 수 있습니다.

```
// REPT 기능 사용 설정
smartButton1.SpecialFunction = SmartX.SmartButton.SPECIALFUNC.REPT;  // REPT 기능 사용 설정
// RepeatAccelerate 타입의 배열변수 repAcc 선언 및 생성. 3단계 가속
SmartX.SmartButton.RepeatAccelerate[] repAcc = new SmartX.SmartButton.RepeatAccelerate[3];
repAcc[0].iClickCount = 10;  repAcc[0].iInterval = 300;  // 처음 10회 입력 동안 Interval : 300
repAcc[1].iClickCount = 10;  repAcc[1].iInterval = 100;  // 다음 10회 입력 동안 Interval 100
repAcc[2].iClickCount = 10;  repAcc[2].iInterval = 10;   // 이후 입력 되는 Interval : 10
smartButton1.RepeatIntervalAccelerate = repAcc;  // 연속 입력에 따른 가속 입력 처리 시간을 설정
```

[CASE-2] 안전 입력 기능 사용하기

안전 입력 기능은 버튼 클릭 시 체터링(Chattering)을 방지해야 하거나, 장비 동작에 있어 신중하게 버튼의 입력을 해야 하는 경우 사용하는 기능입니다. 버튼 클릭 시 바로 입력되는 것이 아니라 SafeInterval 속성에서 설정된 시간 이상 버튼을 누르고 있어야 클릭 이벤트가 발생됩니다.

```
smartButton1.SpecialFunction = SmartX.SmartButton.SPECIALFUNC.SAFE;  // SAFE 기능 사용 설정
smartButton1.SafeInterval = 2000;  // 버튼의 입력 시간을 2초(2000ms)설정
```

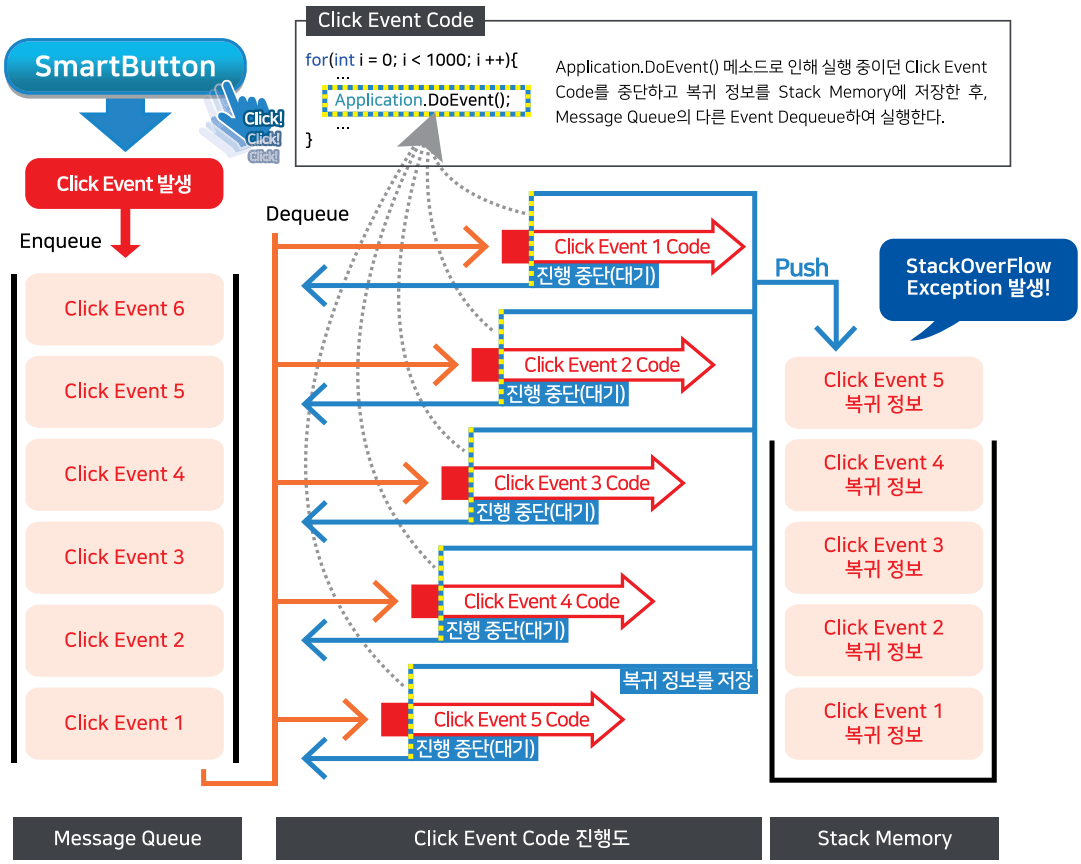
[CASE-3] 연속+안전 입력 기능 사용하기

연속+안전 입력 기능은 연속 입력 기능과 안전 입력 기능을 모두 사용하는 기능입니다. 즉, SafeInterval 속성에서 설정된 시간 이상 버튼을 누르고 있으면 클릭 이벤트가 발생되고, 누른 상태를 유지하면 RepeatInterval 속성에서 설정한 간격마다 클릭 이벤트가 발생하게 됩니다. 또한 RepeatIntervalAccelerate 속성을 사용해 입력 간격을 가속시킬 수 있습니다.

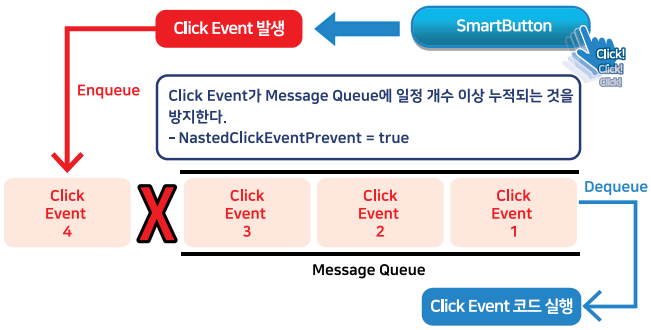
```
// SAFE + REPT 기능 사용 설정
smartButton1.SpecialFunction = SmartX.SmartButton.SPECIALFUNC.SAFE_REPT;
// 반복 입력 간격 : 1초(1000ms)
smartButton1.RepeatInterval = 1000;
// 버튼 입력 유지 시간 : 2초(2000ms)
smartButton1.SafeInterval = 2000;
```

3) 버튼 연속 클릭 시 생기는 문제 해결 방법

버튼을 빠르게 연속해서 클릭하게 되면 Message Queue에 클릭 이벤트가 과도하게 누적되어 다른 코드의 처리가 지연될 수 있으며, 클릭 이벤트 코드의 반복문 내에서 DoEvent() 메소드를 호출하는 경우 StackOverFlow 예외가 발생할 수 있습니다. 이 같은 경우를 방지하기 위해서는 두 가지 방법이 있으며, 아래 CASE를 확인하시기 바랍니다.



[그림] StackOverFlow Exception 발생 원인

CASE-1	클릭 이벤트 처리 중 다른 클릭 이벤트 발생을 무시하는 방법
그림	
해결 방법	버튼 클릭 이벤트의 처리 중에는 Message Queue에 클릭 이벤트가 Enqueue되지 않도록 방지합니다. 즉, 클릭 이벤트 처리 중에는 클릭 이벤트를 무시합니다.
적용 방법	- SpecialFunction을 SAFE로 설정 및 SafeInterval을 1로 설정
CASE-2	**Message Queue에 누적되는 클릭 이벤트의 개수를 제한시키는 방법**
그림	
해결 방법	Message Queue에 일정 개수 이상 누적되지 못하도록 방지합니다. 즉, 버튼을 여러 번 클릭할 수 있지만 누적된 클릭 이벤트가 일정 개수를 넘기면 발생된 클릭 이벤트를 무시합니다.
적용 방법	NestedClickEventPrevent를 True로 설정

참고 "SpecialFunction, SafeInterval, NestedClickEventPrevent 속성" 내용을 참고하시기 바랍니다.

4) SmartButton 인터페이스 설명

SmartButton Component Interface		
속성		
BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm	BackPictureBox2 : SmartGroupBox
ButtonColor : Color	ButtonImageAutoSize : bool	ButtonStatus : SmartButton.BUTSTATUS
ColorKeySamplePosition : Point	DisableImage : Image	DownImage : Image
Font : Font	GroupID : int	InitVisible : bool
Mode : SmartButton.BUTTONMODE	NestedClickEventPrevent : bool	OutLinePixel : int
RepeatInterval : int	RepeatIntervalAccelerate : SmartButton.RepeatAccelerate[]	SafeInterval : int
SpecialFunction : SmartButton.SPECIALFUNC	Text : string	TextColor : Color
TextDownColor : Color	TextHAlign : SmartButton.TextHorAlign	TextLocation : Point
TextVAlign : SmartButton.TextVerAlign	UpImage : Image	
메소드		
ButtonDown() : void	ButtonUp() : void	SetPosXY(int iXPos, int iYPos) : void

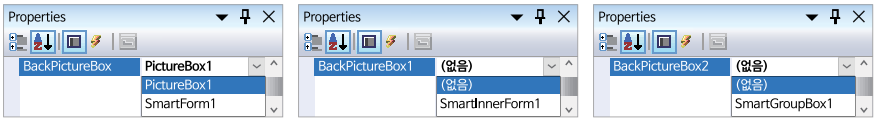
프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartButton의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartButton에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 BackColor 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법



프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartButton의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) ButtonColor, OutLinePixel, Font, Text, TextColor, TextDownColor

SmartButton에 표현되는 각 요소의 색상을 설정합니다.

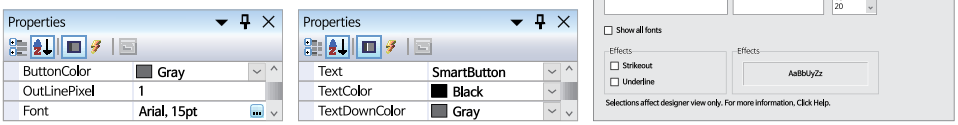
- `ButtonColor` : SmartButton에 버튼 색상을 설정합니다.
- `OutLinePixel` : SmartButton에 이미지를 사용하지 않을 경우 테두리의 두께를 설정합니다.
- `Font` : SmartButton에 표현되는 텍스트의 폰트를 설정합니다.
- `Text` : SmartButton의 텍스트를 가져오거나 설정합니다.
- `TextColor` : 버튼이 눌리지 않은 상태의 텍스트 색상을 설정합니다.
- `TextDownColor` : 버튼이 눌린 상태의 텍스트 색상을 설정합니다.

참고 멀티라인 기능은 지원하지 않으며, 멀티라인을 표현하시려면 이미지를 사용하시기 바랍니다.

[표] OutlinePixel 속성값에 따른 테두리 변화

OutlinePixel = 1	OutlinePixel = 5
SmartButton	SmartButton

사용법



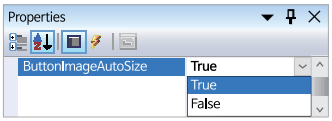
프로퍼티(속성) ButtonImageAutoSize

SmartButton의 크기를 이미지 크기에 맞게 자동으로 조절시킬지 설정합니다.

- `bool` : 크기 자동 조절 여부
  - `true` : 이미지에 맞게 조절함
  - `false` : 이미지에 맞게 조절하지 않음

True	False
	

사용법



프로퍼티(속성) UpImage, DownImage, DisableImage

Masking Image

버튼 각각의 상태별 보여지는 이미지를 설정합니다.

- `UpImage` : 버튼이 눌리지 않은 상태의 버튼 이미지를 지정합니다.
- `DownImage` : 버튼이 눌려진 상태의 버튼 이미지를 지정합니다.
- `DisableImage` : 버튼이 비활성화(Enabled = false) 시 보여지는 버튼 이미지를 지정합니다.

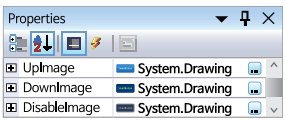
속성	UpImage	DownImage	DisableImage
이미지			
설명	버튼이 눌리지 않은 경우	버튼이 눌린 경우	Enable 속성이 False인 경우

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 “투명(Masking) 처리 영역 제작 가이드”에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 “이미지 제작 가이드”와 “투명(Masking) 처리 영역 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) ColorKeySamplePosition

SmartButton에 사용한 이미지에 마스킹 처리를 한 경우, 마스킹 컬러의 위치가 (0, 0)이 아닐 때 마스킹 컬러가 위치하는 좌표를 설정합니다.

참고 "투명(Masking) 처리 영역 제작 가이드" 내용을 참고하시기 바랍니다.

사용법

프로퍼티(속성) TextHAlign, TextVAlign

SmartButton에 표현되는 텍스트의 정렬 상태를 설정합니다.

- `TextHAlign` : 수평 정렬 상태를 설정합니다.
- `TextVAlign` : 수직 정렬 상태를 설정합니다.

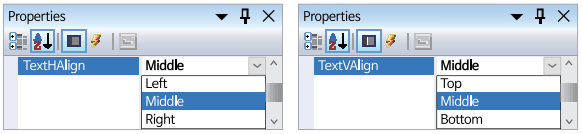
- `SmartButton.TextHorAlign` : 수평 정렬 상태

열거값	설명	열거값	설명	열거값	설명
Left	좌측 정렬	Middle	중앙 정렬	Right	우측 정렬

- `SmartButton.TextVerAlign` : 수직 정렬 상태

열거값	설명	열거값	설명	열거값	설명
Top	상단 정렬	Middle	중앙 정렬	Bottom	하단 정렬

사용법



프로퍼티(속성) TextLocation

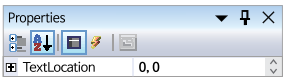
SmartButton에 표시되는 텍스트의 위치를 설정합니다.

참고 TextLocation의 기준점(0, 0)은 TextHAlign, TextVAlign 속성값에 따라 변경됩니다.
"TextHAlign, TextVAlign 속성" 내용을 참고하시기 바랍니다.

[표] TextHAlign, TextVAlign 속성값이 Middle일 때 TextLocation 속성값에 따른 텍스트의 위치

(-15, -15)	(0, 0)	(15, 15)
SmartButton	SmartButton	SmartButton

사용법



프로퍼티(속성) ButtonStatus

SmartButton의 현재 상태(눌림 여부)를 얻습니다.

중요 ButtonStatus 속성은 읽기 전용입니다.

- SmartButton.BUTTONSTATUS : 버튼의 현재 상태
  - SmartButton.BUTTONSTATUS.DOWN : 눌려있음
  - SmartButton.BUTTONSTATUS.UP : 눌려있지 않음

C# 사용법

```
if (smartButton1.ButtonStatus == SmartButton.BUTTONSTATUS.UP)
{
  // 버튼이 눌리지 않은 상태
}
```

VB 사용법

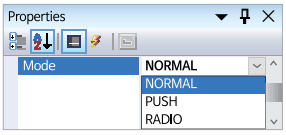
```
If smartButton1.ButtonStatus = SmartButton.BUTTONSTATUS.UP Then
End If
```

프로퍼티(속성) Mode

SmartButton의 동작 모드를 설정합니다.

- SmartButton.BUTTONMODE.NORMAL : 일반적인 버튼
- SmartButton.BUTTONMODE.PUSH : 버튼 클릭 시 다시 클릭하기 전까지 눌린 상태를 유지함
- SmartButton.BUTTONMODE.RADIO : GroupID 속성이 동일한 SmartButton이 그룹이 되어 RadioButton과 동일하게 동작함

사용법



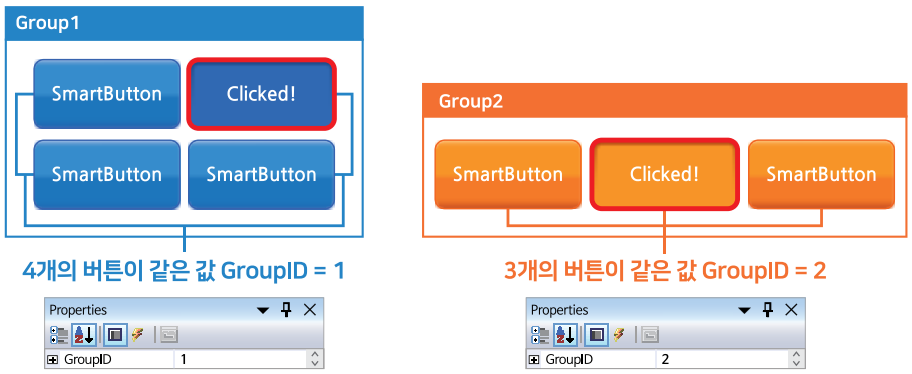
프로퍼티(속성) GroupID

SmartButton의 Mode 속성이 RADIO 일 때, SmartButton을 RadioButton처럼 사용하기 위해 그룹을 설정합니다.

참고 눌려진 버튼을 다시 눌려지지 않은 상태로 변경하려면 ButtonUp() 메소드를 호출하시면 됩니다. "ButtonUp() 메소드" 내용을 참고하시기 바랍니다.

• `int` : Radio 모드로 동작할 SmartButton의 그룹 ID

사용법



프로퍼티(속성) SpecialFunction

SmartButton의 특수 기능을 설정합니다. Mode 속성에서 설정된 동작 모드에 따라 지원되는 기능이 다르니 아래 표를 확인하시기 바랍니다.

[표] Mode 속성에 따른 특수 기능 지원 여부

Mode	REPT	SAFE	SAFE_REPT
NORMAL	지원	지원	지원
PUSH	미지원	지원	미지원
RADIO	미지원	지원	미지원

참고 "RepeatInterval, SafeInterval, RepeatIntervalAccelerate 속성" 내용을 참고하시기 바랍니다.

• `SmartButton.SPECIALFUNC` : 특수 기능 설정
  – `SmartButton.SPECIALFUNC.NONE` : 특수 기능을 사용하지 않음 (기본값)
  – `SmartButton.SPECIALFUNC.REPT` : 버튼을 계속 누르고 있을 경우 클릭(Click) 이벤트를 일정 간격(RepeatInterval 또는 RepeatIntervalAccelerate 속성값) 발생시킴
  – `SmartButton.SPECIALFUNC.SAFE` : 버튼 클릭 시 바로 입력되는 것이 아니라 설정된 시간(SafeInterval 속성값)이상 Button 클릭을 유지해야만 클릭 이벤트가 발생됨
  ※ 버튼 클릭 시 체터링(Chattering)을 방지해야 하거나 혹은 장비 동작에 있어 신중하게 버튼의 입력을 해야 하는 경우 사용하는 기능입니다.
  – `SmartButton.SPECIALFUNC.SAFE_REPT` : REPT 기능과 SAFE 기능을 모두 사용함

C# 사용법

```
// REPT 기능 사용 설정
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.REPT;
// 반복 입력 간격 시간을 1초(1000ms) 설정
smartButton1.RepeatInterval = 1000;

// SAFE 기능 사용 설정
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE;
// 버튼의 입력 시간을 2초(2000ms) 설정
smartButton1.SafeInterval = 2000;

// SAFE + REPT 기능 사용 설정
```

```
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE_REPT;
// 반복 입력 간격 시간을 1초(1000ms) 설정, 버튼의 입력 유지 시간을 2초(2000ms) 설정
smartButton1.RepeatInterval = 1000;
smartButton1.SafeInterval = 2000;
```

VB 사용법

```
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.REPT
smartButton1.RepeatInterval = 1000
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE
smartButton1.SafeInterval = 200
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE_REPT
smartButton1.RepeatInterval = 1000
smartButton1.SafeInterval = 2000
```

프로퍼티(속성) RepeatInterval, SafeInterval

SpecialFunction 속성을 설정한 경우 각 특수 기능(SAFE, REPT)에서 각각의 시간 간격(Interval)을 설정합니다.

- RepeatInterval : SpecialFunction이 REPT인 경우 반복 입력 시간 간격을 설정합니다.
- SafeInterval : SpecialFunction이 SAFE인 경우 입력 유지 시간을 설정합니다.

참고 SpecialFunction 속성값이 SAFE_REPT인 경우 두 속성을 모두 사용하며, "SpecialFunction 속성" 내용을 참고하시기 바랍니다.

- int : 시간 간격 (단위 : ms)

사용법

참고 "SpecialFunction 속성의 사용법" 내용을 참고하시기 바랍니다.

프로퍼티(속성) RepeatIntervalAccelerate

SpecialFunction 속성을 REPT 또는 SAVE_REPT로 설정하여 버튼 반복 입력 기능을 사용하도록 설정했을 때, 버튼을 누르고 있으면 자동으로 반복하여 입력됩니다. 여기서 반복 입력 속도(Interval)를 점차 증가시키도록 처리할 수 있는 속성입니다.

- SmartButton.RepeatAccelerate[] : 버튼 반복 입력 횟수 및 입력 속도 배열
  - int iClickCount : 반복 입력 횟수
  - int iInterval : 입력 속도

C# 사용법

```
// 버튼을 계속 누르고 있을 경우 클릭(Click) 이벤트를 일정 간격으로 발생시킨다.
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.REPT;
// RepeatAccelerate 타입의 배열 변수 repAcc 선언 및 생성. 3단계 가속
SmartButton.RepeatAccelerate[] repAcc = new SmartButton.RepeatAccelerate[3];
// 처음 10회 입력 동안 Interval : 300
repAcc[0].iClickCount = 10;
repAcc[0].iInterval = 300;
// 다음 10회 입력 동안 Interval : 100
repAcc[1].iClickCount = 10;
repAcc[1].iInterval = 100;
// 이후 입력 되는 Interval : 10
repAcc[2].iClickCount = 10;
repAcc[2].iInterval = 10;
smartButton1.RepeatIntervalAccelerate = repAcc;
```

VB 사용법

```
smartButton1.SpecialFunction = SmartButton.SPECIALFUNC.REPT
Dim repAcc As SmartButton.RepeatAccelerate() = New SmartButton.RepeatAccelerate(2) {}
repAcc(0).iClickCount = 10
repAcc(0).iInterval = 300
repAcc(2).iClickCount = 10
repAcc(2).iInterval = 10
smartButton1.RepeatIntervalAccelerate = repAcc
```

프로퍼티(속성) NestedClickEventPrevent

SmartButton을 인위적으로 지나치게 빨리 연속적으로 클릭하는 경우 시스템의 오작동 및 시스템 성능 저하가 발생될 수 있습니다. NestedClickEventPrevent 속성은 이를 개선할 수 있는 기능입니다. 또한, StackOverflow 예외를 방지 할 수 있습니다.

참고 Click Event 처리 코드가 많은 경우 즉, 처리 코드의 실행 시간이 오래 걸리는 경우 적용하시기 바랍니다. 자세한 내용은 "버튼 연속 클릭 시 생기는 문제 해결 방법" 내용을 참고하시기 바랍니다.

- `bool` : 버튼 연속 입력 방지 기능 설정 여부
  - `true` : 중복 Click Event 발생을 허용하지 않음 (Stack Overflow 예외 발생 방지)
  - `false` : 중복 Click Event 발생을 허용함 (기본값)

사용법

메소드(함수) ButtonDown, ButtonUp

SmartButton의 눌림 상태를 설정합니다.

- `ButtonDown()` : SmartButton을 눌려진 상태로 설정합니다.
- `ButtonUp()` : SmartButton을 눌려지지 않은 상태로 설정합니다.

참고 ButtonDown(), ButtonUp() 메소드는 Mode 속성이 PUSH, RADIO로 설정된 경우 프로그램 상에서 버튼의 눌림 상태를 표현하기 위해 주로 사용됩니다. "Mode 속성" 내용을 참고하시기 바랍니다.

중요 ButtonDown(), ButtonUp() 메소드 호출 시 보여지는 상태만 변경되며, 각종 이벤트(Click 이벤트 등)는 발생하지 않습니다.

◉ `void ButtonDown()`
◉ `void ButtonUp()`

C# 사용법

```
// 버튼의 상태를 마치 버튼을 누른 것처럼 표시합니다.
smartButton1.ButtonDown();
// 버튼이 만약 눌려있을 경우 원상태로 됩니다.
smartButton1.ButtonUp();
```

VB 사용법

```
smartButton1.ButtonDown()
smartButton1.ButtonUp()
```

메소드(함수) **SetPosXY**

SmartButton의 위치를 변경합니다. Form의 좌측 상단(0, 0)이 기준이며, SmartButton의 좌측 상단 모서리의 위치를 설정합니다.

◉ `void SetPosXY(int iXPos, int iYPos)`

[인자]

- `int iXPos` : X좌표 위치
- `int iYPos` : Y좌표 위치

C# 사용법

```
// 버튼의 위치를 X : 100, Y : 200으로 설정합니다.
smartButton1.SetPosXY(100, 200);
```

VB 사용법

```
smartButton1.SetPosXY(100, 200)
```

5) SmartButton 예제 사용하기

SmartButton을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

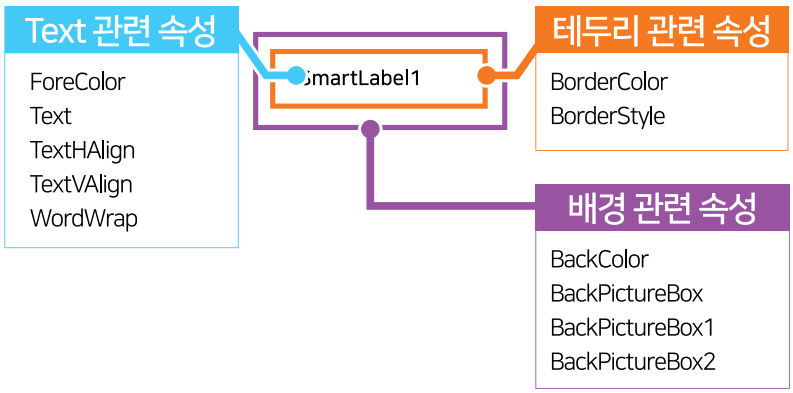
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartButton"

4. SmartLabel

SmartLabel은 .NET Compact Framework에서 지원되는 Label 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 Label의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 배경 투명 처리 효과 기능
  - SmartForm, SmartInnerForm, SmartGroupBox 연동
- Click, Double Click 이벤트 지원
- 수직/수평 정렬 기능
- WordWrap 기능
- 화면 갱신 시 깜박임을 최소화

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartLabel 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartLabel에서 이미지는 배경 컴포넌트에 사용됩니다.

STEP-1 배경 속성 설정하기

SmartLabel의 배경 색상을 BackColor 속성을 사용해 색상으로 설정하는 방법과 BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있습니다.

[CASE-1] 색상으로 설정하는 경우

배경 컴포넌트의 배경 색상과 동일하게 설정하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackColor	Yellow

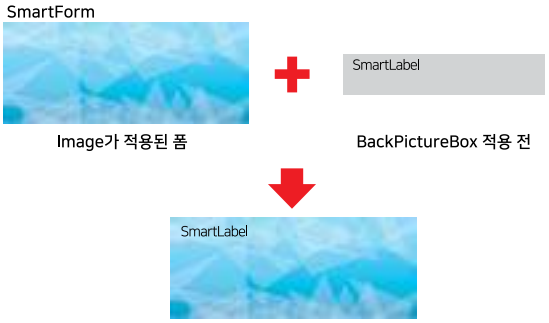
※ 자세한 내용은 "BackColor 속성" 내용을 참고하시기 바랍니다.

[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackPictureBox	SmartForm

※ 자세한 내용은 "BackPictureBox 속성" 내용을 참고하시기 바랍니다.



STEP-2 Text 관련 속성 설정하기

SmartLabel에 표시되는 Text를 Text 속성으로 설정하며, ForeColor 속성으로 색상을 설정합니다. 또한 TextHAlign, TextVAlign 속성을 사용해 Text의 수평, 수직 정렬을 설정합니다. 만약 Text의 길이가 SmartLabel의 폭보다 길어질 경우 WordWrap 속성을 True로 설정하여 멀티라인을 자동으로 출력 처리하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
Text	SmartLabelTest	TextVAlign	Middle
ForeColor	Black	WordWrap	True
TextHAlign	Left	-	-

※ 자세한 내용은 "Text, ForeColor, TextHAlign, TextVAlign 속성" 내용을 참고하시기 바랍니다.

[그림] WordWrap 속성값에 따른 화면 출력

3) SmartLabel 인터페이스 설명

SmartLabel Component Interface		
속성		
BackColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
BackPictureBox2 : SmartGroupBox	BorderColor : Color	BorderStyle : BorderStyle
ForeColor : Color	InitVisible : bool	Text : string
TextHAlign : SmartLabel.TextHorAlign	TextVAlign : SmartLabel.TextVerAlign	WordWrap : bool
메소드		
SetDoubleClickEnable(bool Enable) : void		
이벤트		
Click : EventHandler	DoubleClick : EventHandler	

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartLabel의 InitVisible 속성 값을 False로 설정하시면 화면 전환 시 좀 더 부드럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartLabel의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartLabel에 의해 배경이 가려지는 현상을 방지할 수 있습니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

프로퍼티(속성) BackColor

SmartLabel 컨트롤의 내부 배경색을 설정합니다. 배경 이미지를 사용하지 않을 경우에 적용됩니다.

C# 사용법

```
// 또는 Color.FromArgb() 메소드로 적용 가능
smartLabel1.BackColor = Color.Blue;
```

VB 사용법

```
smartLabel1.BackColor = Color.Blue
```

프로퍼티(속성) BorderColor

SmartLabel의 테두리를 표시하는 경우 색상을 설정합니다.

C# 사용법

```
// R, G, B
smartLabel1.BorderColor = Color.FromArgb(100, 100, 0);
```

VB 사용법

```
smartLabel1.BorderColor = Color.FromArgb(100, 100, 0)
```

프로퍼티(속성) BorderStyle

SmartLabel의 테두리를 표시할지를 설정합니다.

- BorderStyle.None : 테두리 없음
- BorderStyle.FixedSingle : 선 테두리
- BorderStyle.Fixed3D : 3D 테두리

C# 사용법

```
// 선 테두리 표시
smartLabel1.BorderStyle = BorderStyle.FixedSingle;
```

VB 사용법

```
smartLabel1.BorderStyle = BorderStyle.FixedSingle
```

프로퍼티(속성) ForeColor

SmartLabel의 Text 색상을 설정합니다.

C# 사용법

```
// R, G, B
smartLabel1.ForeColor = Color.FromArgb(100, 100, 0);
```

VB 사용법

```
smartLabel1.ForeColor = Color.FromArgb(100, 100, 0)
```

프로퍼티(속성) Text

SmartLabel에서 표시되는 문자열을 설정합니다.

C# 사용법

```
smartLabel1.Text = " SmartLabel Test ";
```

VB 사용법

```
smartLabel1.Text = " SmartLabel Test "
```

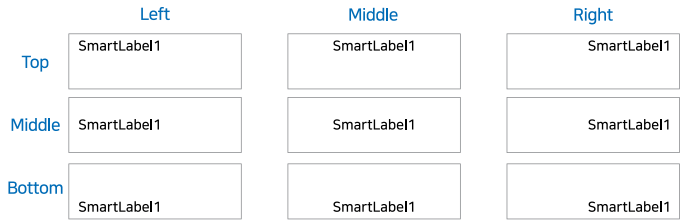
프로퍼티(속성) TextHAlign, TextVAlign

SmartLabel에서 출력되는 Text의 수평, 수직 정렬을 설정합니다.

- TextHAlign : Text의 수평 정렬을 설정합니다.
- TextVAlign : Text의 수직 정렬을 설정합니다.

- SmartLabel.TextHorAlign.Left : 수평 왼쪽 정렬
- SmartLabel.TextHorAlign.Middle : 수평 중앙 정렬
- SmartLabel.TextHorAlign.Right : 수평 오른쪽 정렬

- SmartLabel.TextVerAlign.Top : 수직 상위 정렬
- SmartLabel.TextVerAlign.Middle : 수직 중앙 정렬
- SmartLabel.TextVerAlign.Bottom : 수직 하위 정렬



C# 사용법

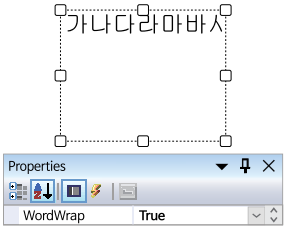
```
// 수평 중앙 정렬
smartLabel1.TextHAlign = SmartLabel.TextHorAlign.Middle;
```

VB 사용법

```
smartLabel1.TextHAlign = SmartLabel.TextHorAlign.Middle
```

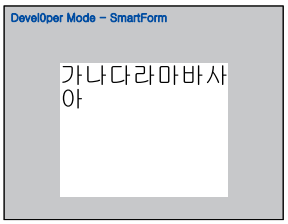
프로퍼티(속성) WordWrap

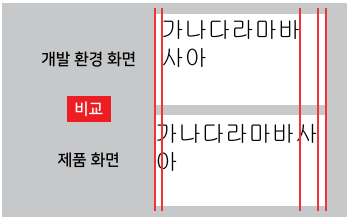
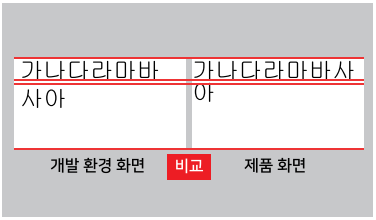
속성을 True로 설정 시 Text의 길이가 SmartLabel의 폭보다 길어질 경우 자동으로 멀티 라인 출력 처리되며, 또한 라인 개행 문자 (“₩n”)를 사용하여 멀티라인 출력 처리할 수 있습니다.

True	False
	

주의 WordWrap 속성 사용 시 개행이 달라지는 현상

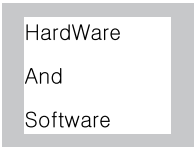
IEC-Series 제품과 PC(개발 환경)에서 동일한 서체를 사용해도 개행의 위치가 다르게 보일 수 있습니다. 이 현상은 IEC-Series 제품과 PC(개발 환경)의 차이(여백, 행간)로 발생하며, 소스 코드에서 개행을 원하는 위치에 (“₩n”)를 추가하여 사용하시기 바랍니다.

개발 환경(Visual Studio) 화면	IEC-Series 제품 화면
	

① 여백 차이	② 행간 차이
	

C# 사용법

```
// Wordwrap 의 속성값을 True로 설정. 개행을 원하는 위치에 ₩n을 삽입
smartLabel1.Text = " HardWare₩n₩nAnd₩n₩nSoftWare ";
```



VB 사용법

```
smartLabel1.Text = " HardWare " & vbCrLf & vbCrLf & " And " & vbCrLf & vbCrLf & " SoftWare "
```

메소드(함수) SetDoubleClickEnable

DoubleClick 이벤트 발생을 설정 또는 해제합니다. SmartLabel을 빠르게 여러 번 클릭하는 경우 DoubleClick으로 인식되어 Click 이벤트가 발생하지 않는 현상을 방지할 수 있습니다.

주의 Click 이벤트만 사용하실 경우 반드시 SetDoubleClickEnable()을 False로 설정하여 사용하시기 바랍니다.

[표] SetDoubleClickEnable() 설정에 따른 Click, DoubleClick 이벤트 발생 횟수

설정값	True	False
Click 횟수	100회	
Click 이벤트 발생 횟수	50회	100회
DoubleClick 이벤트 발생 횟수	50회	0회

◉ `void SetDoubleClickEnable(bool bEnable)`

[인자]

- `bool bEnable` : DoubleClick 이벤트 발생 방지 유무
  - `true` : DoubleClick 이벤트 발생
  - `false` : DoubleClick 이벤트 발생 방지

C# 사용법

```
// DoubleClick 이벤트 발생 방지
smartLabel1.SetDoubleClickEnable(false);
```

VB 사용법

```
smartLabel1.SetDoubleClickEnable(false)
```

이벤트 Click

SmartLabel을 Click했을 경우 발생하는 이벤트입니다.(기존 Label에서는 없는 기능)

C# 사용법

```
private void smartLabel1_Click(object sender, EventArgs e)
{
  MessageBox.Show(" SmartLabel1이 Click되었습니다. ");
}
```

VB 사용법

```
Private Sub smartLabel1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartLabel1.Click
  MessageBox.Show(" SmartLabel1이 Click되었습니다. ")
End Sub
```

이벤트	DoubleClick

Label을 DoubleClick했을 경우 발생하는 이벤트입니다.(기존 Label에서는 없는 기능)

참고 SetDoubleClickEnable() 메소드를 이용하여 DoubleClick 이벤트 발생을 방지할 수 있습니다. 자세한 내용은 SetDoubleClickEnable() 메소드를 참고하시기 바랍니다.

C# 사용법

```
private void smartLabel1_DoubleClick(object sender, EventArgs e)
{
  MessageBox.Show( " SmartLabel1이 DoubleClick되었습니다. " );
}
```

VB 사용법

```
Private Sub smartLabel1_DoubleClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartLabel1.DoubleClick
  MessageBox.Show( " SmartLabel1이 DoubleClick되었습니다. " )
End Sub
```

4) SmartLabel 예제 사용하기

SmartLabel을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

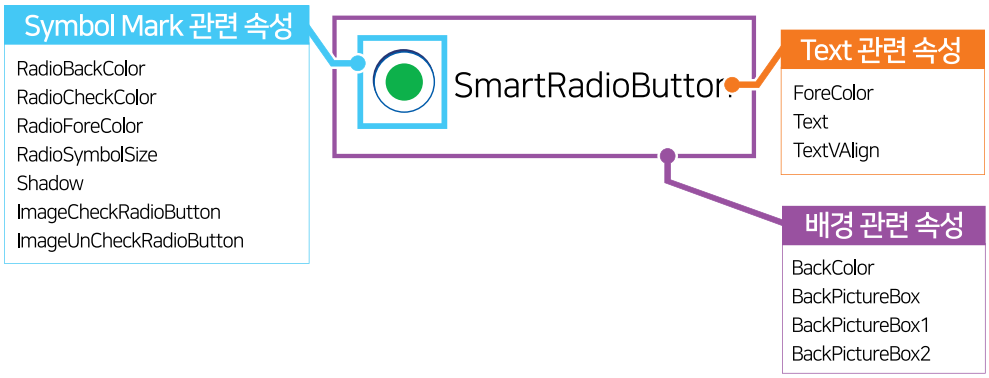
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartLabel"

5. SmartRadioButton

SmartRadioButton은 .NET Compact Framework에서 지원되는 RadioButton 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 RadioButton의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 배경 투명 처리 효과 기능
  - ↳ PictureBox, SmartForm, SmartInnerForm, SmartGroupBox 연동
- Radio Mark의 크기 조절 및 색상 변경 기능
- Radio Mark의 사용자 이미지 설정 기능

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartRadioButton 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartRadioButton에서 이미지는 배경 컴포넌트와 Symbol Mark에 사용됩니다.

STEP-1 배경 속성 설정하기

SmartRadioButton의 배경 색상을 BackColor 속성을 사용해 색상으로 설정하는 방법과 BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있습니다.

[CASE-1] 색상으로 설정하는 경우

배경 컴포넌트의 배경 색상과 동일하게 설정하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackColor	Yellow

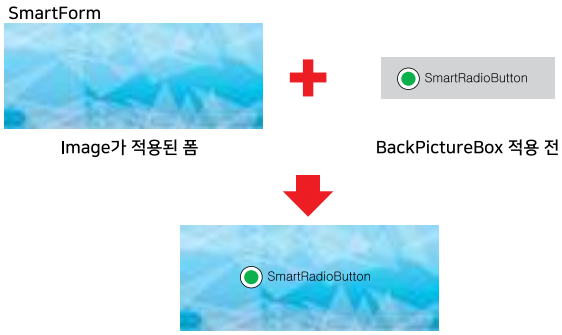
※ 자세한 내용은 "BackColor 속성" 내용을 참고하시기 바랍니다.

[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackPictureBox	SmartForm

※ 자세한 내용은 "BackPictureBox 속성" 내용을 참고하시기 바랍니다.



STEP-2 Symbol Mark 관련 속성 설정하기

Symbol Mark는 이미지를 사용하지 않고 디자인하는 방법과 사용자 이미지를 사용하여 디자인하는 방법이 있습니다.

[CASE-1] 이미지를 사용하지 않을 경우

RadioBackColor, RadioForeColor, RadioSymbolSize, RadioCheckColor 속성으로 Symbol Mark의 배경 및 테두리 색상, 크기와 체크되었을 때의 색상을 설정합니다. 또한, 안쪽 음영을 주고 싶을 때는 Shadow 속성값을 True로 설정하면 됩니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
RadioBackColor	Gray	RadioSymbolSize	20
RadioCheckColor	White	Shadow	True
RadioForeColor	Gray	-	-

※ 자세한 내용은 "RadioBackColor, RadioCheckColor, RadioForeColor, RadioSymbolSize, Shadow 속성"과 내용을 참고하시기 바랍니다.

[그림] 적용된 모습

[CASE-2] 이미지를 사용할 경우

Symbol Mark 이미지를 ImageCheckRadioButton, ImageUnCheckRadioButton 속성을 사용하여 출력합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
ImageCheckRadioButton		ImageUnCheckRadioButton	

※ 자세한 내용은 "ImageCheckRadioButton, ImageUnCheckRadioButton 속성" 내용을 참고하시기 바랍니다.

[그림] 적용된 모습

STEP-3 Text 관련 속성 설정하기

SmartRadioButton에 표시되는 Text를 Text 속성으로 설정하며, ForeColor 속성으로 색상을 설정합니다. 또한, TextV Align 속성을 사용해 Text의 수직 정렬을 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
Text	SmartRadioButton	TextVAlign	Middle
ForeColor	Black	-	-

※ 자세한 내용은 "Text, ForeColor, TextVAlign 속성" 내용을 참고하시기 바랍니다.

STEP-4 그룹 ID 설정하기

SmartRadioButton의 특성상 그룹으로 설정된 버튼들은 하나의 버튼만 선택되므로 이를 구별하기 위해 GroupID 속성으로 그룹 ID값을 설정하여 그룹화합니다. Group ID를 다르게 하여 RadioButton 선택 시 같은 Group ID인 Radio Button의 선택 상태를 변경시킬 수 있습니다.

※ 자세한 내용은 "GroupID 속성" 내용을 참고하시기 바랍니다.

```
// Group-1
smartRadioButton1.GroupID = 1;  smartRadioButton2.GroupID = 1;  smartRadioButton3.GroupID = 1;
// Group-2
smartRadioButton4.GroupID = 2;  smartRadioButton5.GroupID = 2;  smartRadioButton6.GroupID = 2;
```

3) SmartRadioButton 인터페이스 설명

SmartRadioButton Component Interface		
속성		
BackColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
BackPictureBox2 : SmartGroupBox	Checked : bool	ForeColor : Color
GroupID : int	ImageCheckRadioButton : Image	ImageUncheckRadioButton : Image
InitVisible : bool	OutputOnly: bool	RadioBackColor : Color
RadioCheckColor : Color	RadioForeColor : Color	RadioSymbolSize : int
Shadow : bool	Text : string	TextVAlign : SmartRadioButton.TextVerAlign

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartRadioButton의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 부드럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartRadioButton의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartRadioButton에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

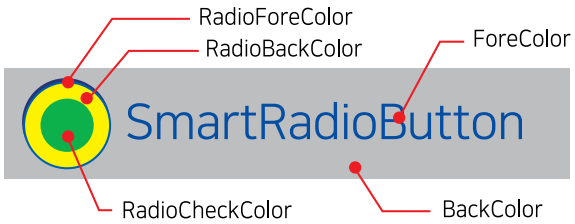
주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

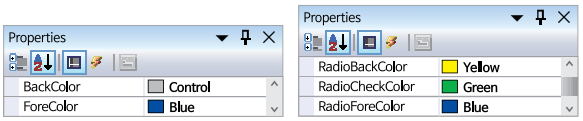
프로퍼티(속성) BackColor, ForeColor, RadioBackColor, RadioCheckColor, RadioForeColor

SmartRadioButton 관련 각 요소별 색상을 설정합니다.

- `BackColor` : SmartRadioButton의 배경 색상
- `ForeColor` : SmartRadioButton의 Font 색상
- `RadioBackColor` : Radio Symbol Mark의 배경 색상
- `RadioCheckColor` : Radio Symbol Mark의 선택(Check) 색상
- `RadioForeColor` : Radio Symbol Mark의 외곽선 색상



사용법



프로퍼티(속성) RadioSymbolSize

Radio Symbol Mark의 크기를 설정합니다.

- `int` : Radio Symbol Mark의 크기 (기본값 : 14)

RadioSymbolSize : 14 (기본값)	RadioSymbolSize : 30
SmartRadioButton1	SmartRadioButton1

C# 사용법

```
smartRadioButton1.RadioSymbolSize = 30;
```

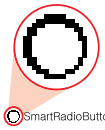
VB 사용법

```
smartRadioButton1.RadioSymbolSize = 30
```

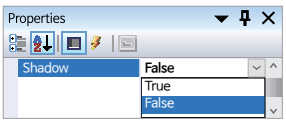
프로퍼티(속성) Shadow

Radio Symbol Mark의 음영 부분의 효과를 처리 여부를 설정합니다.
(RadioSymbolSize 속성값이 커질 경우 사용하시면 효과적입니다.)

- bool : 음영 효과 처리 사용 유무
  - true : 음영 효과 사용
  - false : 음영 효과 사용 안 함

True	False
	

사용법



프로퍼티(속성) ImageCheckRadioButton, ImageUnCheckRadioButton

Masking Image

SmartRadioButton의 Radio Symbol Mark 부분을 사용자 이미지로 설정합니다.

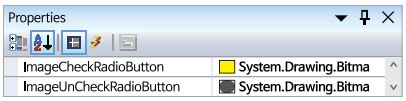
- ImageCheckRadioButton : 선택된 상태의 Radio Symbol Mark 이미지를 설정합니다.
- ImageUnCheckRadioButton : 선택되지 않은 상태의 Radio Symbol Mark 이미지를 설정합니다.

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 “투명(Masking) 처리 영역 제작 가이드”에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 “이미지 제작 가이드”와 “투명(Masking) 처리 영역 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) Text

SmartRadioButton에 표시되는 문자열을 설정합니다.

C# 사용법

```
smartRadioButton1.Text = “ SmartRadioButton1 Test ”;
```

VB 사용법

```
smartRadioButton1.Text = “ SmartRadioButton1 Test ”
```

프로퍼티(속성) TextVAlign

SmartRadioButton의 출력되는 Text의 수직 정렬을 설정합니다.

- SmartRadioButton.TextVerAlign.Top : 수직 상위 정렬
- SmartRadioButton.TextVerAlign.Middle : 수직 중앙 정렬
- SmartRadioButton.TextVerAlign.Bottom : 수직 하위 정렬

C# 사용법

```
// 수직 중앙 정렬
smartRadioButton1.TextVAlign = SmartRadioButton.TextVerAlign.Middle;
```

VB 사용법

```
smartRadioButton1.TextVAlign = SmartRadioButton.TextVerAlign.Middle
```

프로퍼티(속성) Checked

SmartRadioButton의 선택(Check) 상태를 가져오거나 설정합니다.

- bool : SmartRadioButton의 선택(Check) 유무
  - true : Check됨
  - false : Check 안 됨

C# 사용법

```
// SmartRadioButton이 체크됨
if (smartRadioButton2.Checked == true)
{
  smartLabel1.Text = " Checked " ;
}
```

VB 사용법

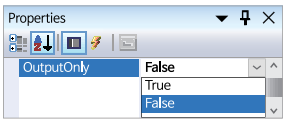
```
If (smartRadioButton1.Checked = True) Then
  smartLabel1.Text = " Checked "
End If
```

프로퍼티(속성) OutputOnly

외부의 입력(Click)으로 선택(Check) 상태가 변경되지 않으며, 오직 프로그램에서 Checked 속성값에 의해서만 선택 상태가 변경됩니다.

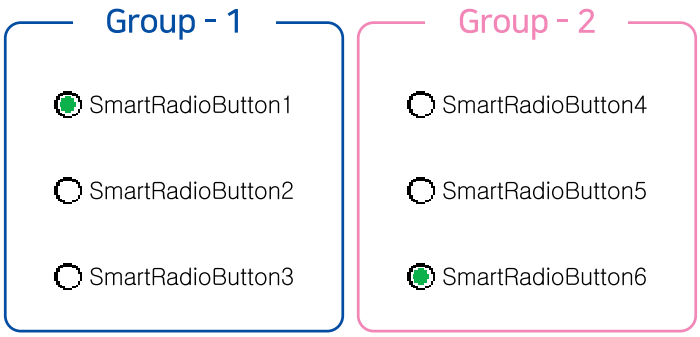
- bool : 외부의 입력으로 선택(Check) 상태 변경 유무
  - true : 상태 변경 안 됨
  - false : 상태 변경됨

사용법



프로퍼티(속성) GroupID

SmartRadioButton의 특성상 그룹으로 설정된 버튼들은 하나의 버튼만 선택되며, 이를 구별하기 위한 그룹 ID값을 설정하여 SmartRadioButton을 그룹화합니다.



- int : 그룹 ID값

C# 사용법

```
// Group-1
smartRadioButton1.GroupID = 1;
smartRadioButton2.GroupID = 1;
smartRadioButton3.GroupID = 1;
smartRadioButton1.Checked = true;
// Group-2
smartRadioButton4.GroupID = 2;
smartRadioButton5.GroupID = 2;
smartRadioButton6.GroupID = 2;
smartRadioButton6.Checked = true;
```

VB 사용법

```
smartRadioButton1.GroupID = 1
smartRadioButton2.GroupID = 1
smartRadioButton3.GroupID = 1
smartRadioButton1.Checked = True

smartRadioButton4.GroupID = 2
smartRadioButton5.GroupID = 2
smartRadioButton6.GroupID = 2
smartRadioButton6.Checked = True
```

4) SmartRadioButton 예제 사용하기

SmartRadioButton를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

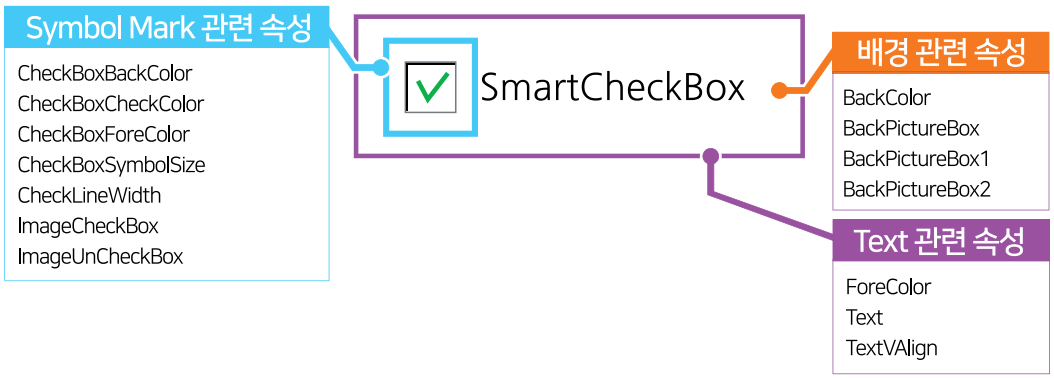
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartRadioButton"

6. SmartCheckBox

SmartCheckBox은 .NET Compact Framework에서 지원되는 CheckBox 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 CheckBox의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 배경 투명 처리 효과 기능
  - PictureBox, SmartForm, SmartInnerForm, SmartGroupBox 연동
- CheckBox Mark의 크기 조절 및 색상 변경 기능
- CheckBox Mark의 사용자 이미지 설정 기능

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartCheckBox 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartCheckBox에서 이미지는 배경 컴포넌트와 Symbol Mark에 사용됩니다.

STEP-1 배경 속성 설정하기

SmartCheckBox의 배경 색상을 BackColor 속성을 사용해 색상으로 설정하는 방법과 BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있습니다.

[CASE-1] 색상으로 설정하는 경우

배경 컴포넌트의 배경 색상과 동일하게 설정하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackColor	Yellow

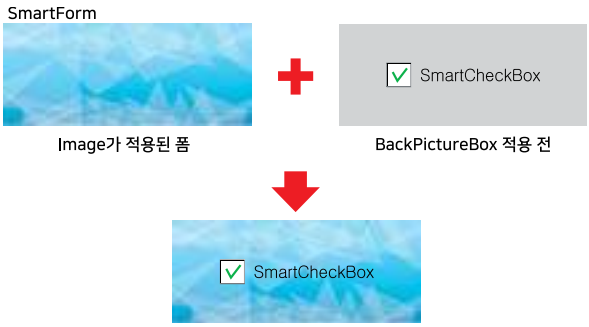
※ 자세한 내용은 “BackColor 속성” 내용을 참고하시기 바랍니다.

[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackPictureBox	SmartForm

※ 자세한 내용은 "BackPictureBox 속성" 내용을 참고하시기 바랍니다.



STEP-2 Symbol Mark 관련 속성 설정하기

Symbol Mark는 이미지를 사용하지 않고 디자인하는 방법과 사용자 이미지를 사용하여 디자인하는 방법이 있습니다.

[CASE-1] 이미지를 사용하지 않을 경우

CheckBoxBackColor, CheckBoxForeColor, CheckBoxSymbolSize, CheckBoxCheckColor, CheckLineWidth 속성으로 Symbol Mark의 배경 및 외곽선 색상, 크기, 체크되었을 때의 색상과 선 두께를 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
CheckBoxBackColor	Gray	CheckBoxCheckColor	White
CheckBoxForeColor	Black	CheckLineWidth	5
CheckBoxSymbolSize	20	-	-

※ 자세한 내용은 "CheckBoxBackColor, CheckBoxForeColor, CheckBoxSymbolSize, CheckBoxCheckColor, CheckLineWidth 속성"과 내용을 참고하시기 바랍니다.

[그림] 적용된 모습

[CASE-2] 이미지를 사용할 경우

Symbol Mark 이미지를 ImageCheckBox, ImageUnCheckBox 속성을 사용하여 출력합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
ImageCheckBox		ImageUnCheckBox	

※ 자세한 내용은 "ImageCheckBox, ImageUnCheckBox 속성" 내용을 참고하시기 바랍니다.

[그림] 적용된 모습

STEP-3 Text 관련 속성 설정하기

SmartCheckBox에 표시되는 Text를 Text 속성으로 설정하며, ForeColor 속성으로 색상을 설정합니다. 또한, TextV Align 속성을 사용해 Text의 수직 정렬을 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값	관련 속성	속성값
Text	SmartCheckBox	ForeColor	Black	TextVAlign	Middle

※ 자세한 내용은 “Text, ForeColor, TextVAlign 속성” 내용을 참고하시기 바랍니다.

3) SmartCheckBox 인터페이스 설명

SmartCheckBox Component Interface		
속성		
BackColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
BackPictureBox2 : SmartGroupBox	CheckBoxBackColor : Color	CheckBoxCheckColor : Color
CheckBoxForeColor : Color	CheckBoxSymbolSize : int	Checked : bool
CheckLineWidth : int	ForeColor : Color	ImageCheckBox : Image
ImageUnCheckBox : Image	InitVisible : bool	Text : string
TextVAlign : SmartCheckBox.TextVerAlign		

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartCheckBox의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 “공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정”을 참고하시기 바랍니다.

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartCheckBox의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartCheckBox에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

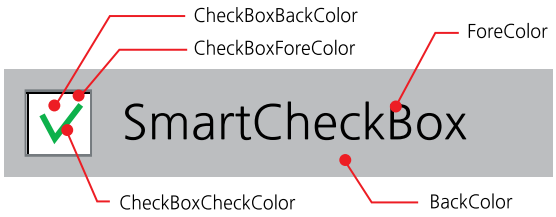
주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

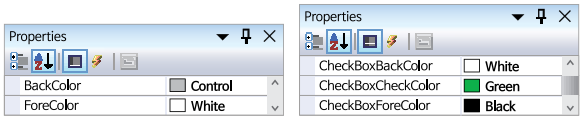
프로퍼티(속성) BackColor, ForeColor, CheckBoxBackColor, CheckBoxCheckColor, CheckBoxForeColor

SmartCheckBox 관련 각 요소별 색상을 설정합니다.

- `BackColor` : SmartCheckBox의 배경 색상
- `ForeColor` : SmartCheckBox의 Font 색상
- `CheckBoxBackColor` : CheckBox Symbol Mark의 배경 색상
- `CheckBoxCheckColor` : CheckBox Symbol Mark의 선택(Check) 색상
- `CheckBoxForeColor` : CheckBox Symbol Mark의 외곽선 색상



사용법



프로퍼티(속성) CheckBoxSymbolSize

CheckBox Symbol Mark의 크기를 설정합니다.

- `int` : CheckBox Symbol Mark의 크기 (기본값 : 14)

CheckBoxSymbolSize : 14 (기본값)	CheckBoxSymbolSize : 30
SmartCheckBox	SmartCheckBox

C# 사용법

```
smartCheckBox1.CheckBoxSymbolSize = 30;
```

VB 사용법

```
smartCheckBox1.CheckBoxSymbolSize = 30
```

프로퍼티(속성) CheckLineWidth

CheckBox Symbol Mark의 선 두께를 설정합니다.
(CheckBoxSymbolSize 속성값이 커질 경우 사용하시면 효과적입니다.)

- `int` : CheckBox Symbol Mark의 Check 선 두께 (기본값 : 3)

CheckLineWidth : 3 (기본값)	CheckLineWidth : 5
	

C# 사용법

```
smartCheckBox1.CheckLineWidth = 5;
```

VB 사용법

```
smartCheckBox1.CheckLineWidth = 5
```

프로퍼티(속성) ImageCheckBox, ImageUnCheckBox Image

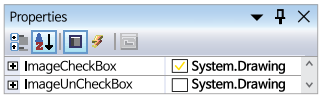
SmartCheckBox의 CheckBox Symbol Mark 부분을 사용자 이미지로 설정합니다.

- ImageCheckBox : 선택된 상태의 CheckBox Symbol Mark 이미지를 설정합니다.
- ImageUnCheckBox : 선택되지 않은 상태의 CheckBox Symbol Mark 이미지를 설정합니다.

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) Checked

SmartCheckBox의 선택(Check) 상태를 가져오거나 설정합니다.

- bool : SmartCheckBox의 선택(Check) 유무
  - true : Check됨
  - false : Check 안 됨

C# 사용법

```
// Check됨
if (smartCheckBox2.Checked == true)
{
  smartLabel1.Text = " Checked ";
}
else
{
  smartLabel1.Text = " UnChecked ";
}
```

VB 사용법

```
If (smartCheckBox2.Checked = True) Then
  smartLabel1.Text = " Checked "
Else
  smartLabel1.Text = " UnChecked "
End If
```

프로퍼티(속성) Text

SmartCheckBox에 표시되는 문자열을 설정합니다.

C# 사용법

```
smartCheckBox1.Text = " SmartCheckBox1 Test ";
```

VB 사용법

```
smartCheckBox1.Text = " SmartCheckBox1 Test "
```

프로퍼티(속성) `TextVAlign`

SmartCheckBox의 출력되는 Text의 수직 정렬을 설정합니다.

- SmartCheckBox.TextVerAlign.Top : 수직 상위 정렬
- SmartCheckBox.TextVerAlign.Middle : 수직 중앙 정렬
- SmartCheckBox.TextVerAlign.Bottom : 수직 하위 정렬

C# 사용법

```
// 수직 중앙 정렬
smartCheckBox1.TextVAlign = SmartCheckBox.TextVerAlign.Middle;
```

VB 사용법

```
smartCheckBox1.TextVAlign = SmartCheckBox.TextVerAlign.Middle
```

4) SmartCheckBox 예제 사용하기

SmartCheckBox를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

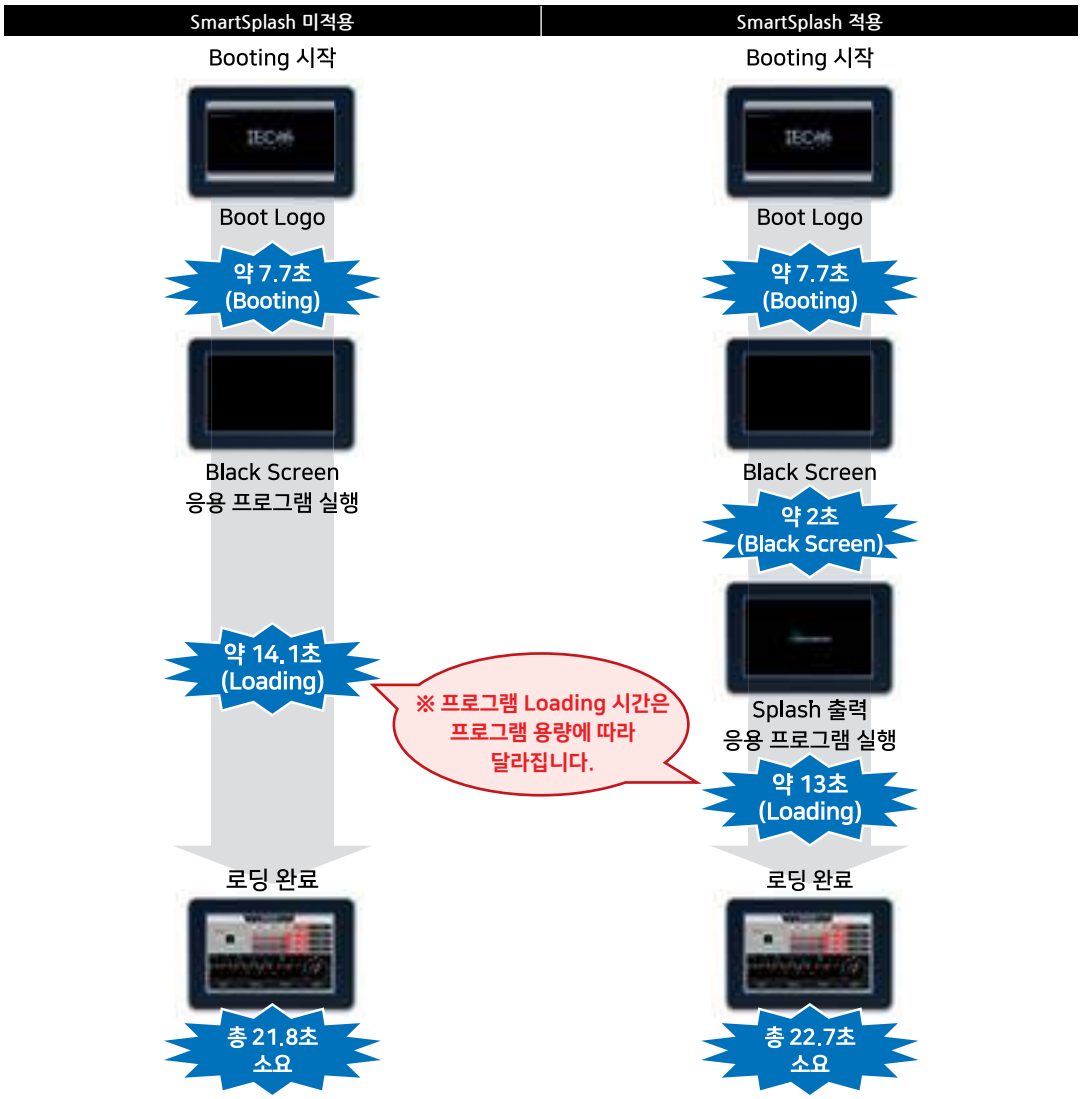
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartCheckBox"

7. SmartSplash

스플래쉬(Splash)란 응용 프로그램이 실행 시 초기에 로딩하는 시간이 발생하며, 이에 따라 사용자가 프로그램이 실행되고 있는지 알 수 있도록 화면에 프로그램 정보나 진행(실행) 중임을 보여주는 화면을 스플래쉬라고 합니다. IEC-Series가 RunTime 모드일 때 부팅 후 응용 프로그램이 실행되어 화면에 표시되기까지 약간의 시간이 소요됩니다. 이 시간 동안 화면은 검은색(Black Screen)으로 표시되어 동작 중임을 알 수 없는 상태로 진행됩니다. SmartSplash는 이러한 검은색(Black Screen) 화면 대신 장치 응용 프로그램이 로딩 중임을 화면에 애니메이션(Animation GIF)으로 표현할 수 있도록 만들어진 컴포넌트입니다.

- 로딩 시 검은색 화면(Black Screen) 최소화
- 로딩 화면 애니메이션(Animation GIF) 지원
- 단 한 줄의 코드로 사용 가능



[그림] 복잡한 구성일 경우 SmartSplash 미적용 VS 적용 동작 비교

SmartSplash 미적용보다 적용 동작이 로딩 완료까지 약 1초 정도 더 소요되지만 Black Screen 출력 시간을 최소화하여 화면에 프로그램이 로딩 중임을 직관적으로 알 수 있습니다.

권장 SmartSplash 사용 시 메소드 권장사항

기존 방식은 스플래쉬 이미지 경로와, Interval, 출력 위치를 각 속성에 직접 적용해야 했지만, 개선된 방식에서는 인자값으로 각 속성값을 설정할 수 있습니다. 아래 표를 참고하시기 바랍니다.

[표] Start(), Finished() 정적 메소드(권장) VS Start(), Finish() 일반 메소드(기존) 비교

정적 메소드(권장)

특징 객체 생성(Instancing)과정 없이 클래스만으로 메소드 호출 가능

```
public Form1()
{
  SmartX.SmartSplash.Start( " SmartLoading1 " , 200);
  InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{
  SmartX.SmartSplash.Finished();
}
```

일반 메소드(기존)

특징 객체 생성(Instancing)을 해야만 메소드 호출 가능

```
private SmartX.SmartSplash m_smartSplash;
public Form1()
{
  m_smartSplash = new SmartX.SmartSplash();
  m_smartSplash.LoadingImagePathname = " SmartLoading1 " ;
  m_smartSplash.AnimationInterval = 200;
  m_smartSplash.CenterPosition = true;
  m_smartSplash.Start();
  InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{
  m_smartSplash.Finish();
}
```

1) 응용 프로그램 용량에 따른 스플래쉬 출력 시간

SmartSplash는 Black Screen 출력 시간을 최소화하여 스플래쉬를 출력하며, 응용 프로그램 용량에 따라 스플래쉬 출력 시간이 달라집니다. 또한, 프로그램의 Black Screen 시간을 스플래쉬 방식보다 더 단축하는 방법도 있습니다. 아래의 표와 참조를 확인하시기 바랍니다.

[표] 응용 프로그램 용량에 따른 스플래쉬 출력 시간

프로그램 용량	7KB (아무것도 없는 상태)	375KB (간단한 구성)	2.43MB (복잡한 구성)
스플래쉬 출력 시간	약 1초	약 2초	약 13초

※ IEC1000-Series (OS : Standard) 기준이며, 출력 시간은 대략적인 수치이므로 차이가 있을 수 있습니다.

참조 프로그램의 Black Screen 시간을 스플래쉬 방식보다 더 단축하는 방법

SmartForm의 NoneBlockingDialog을 사용하면 프로그램의 Black Screen 시간을 단축할 수 있습니다.

자세한 설명은 "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 69. 프로그램 로딩 중 화면 처리 방법"을 참조하시기 바랍니다.

2) 프로그래밍 적용 가이드

SmartSplash는 원래 프로그램 시작 시 사용되도록 만든 용도이지만 프로그램 동작 중 기능 전환 시 로딩 시간이 오래 걸려 화면이 멈춘 것처럼 보일 때, 로딩 중임을 표현하기 위해 사용할 수도 있습니다. 아래 CASE를 참고하여 사용하시기 바랍니다.

CASE-1 프로그램 시작 시 로딩 화면을 처리하는 경우

프로그램 실행 시 프로그램의 로딩 작업에 소요되는 시간 동안 스플래쉬 화면을 출력 처리하는 경우로 폼 생성자 함수에 Start() 정적 메소드를 호출하여 스플래쉬 이미지 관련 인자값을 설정합니다. 이후 프로그램의 실행이 완료되어 화면에 출력되는 시점에 Finished() 메소드를 호출하여 스플래쉬를 종료합니다.

※ 자세한 내용은 "Start(), Finished() 정적 메소드" 내용을 참고하시기 바랍니다.

```
public Form1()
{
  // 내부 이미지 SMARTLOADING1 사용, GIF 이미지의 프레임 간격 300, 출력 위치 (50, 50)
  SmartX.SmartSplash.Start(SmartX.SmartSplash.BuiltInLoadingImages.SMARTLOADING1, 300, 50, 50);
  InitializeComponent();
}
private void Form1_Load(object sender, EventArgs e)
{
  // -- 중략 --
  // 폼 Load 이벤트 함수가 종료되는 시점에서 폼 출력되며, 스플래쉬 종료
  SmartX.SmartSplash.Finished();
}
```

CASE-2 프로그램 실행 중 기능 전환 시 시간이 오래 걸리는 경우

기능 전환 시 시간이 오래 걸리는 코드 앞에 Start() 정적 메소드를 호출하여 스플래쉬 이미지 관련 인자값을 설정합니다. 이후 기능 전환이 완료되어 화면에 출력되는 시점에 Finished() 메소드를 호출하여 스플래쉬를 종료합니다.

※ 자세한 내용은 "Start(), Finished() 정적 메소드" 내용을 참고하시기 바랍니다.

```
// 프로그램 중간에 기능 전환 시 시간이 오래 걸리는 코드 앞에 Start() 메소드 호출
// 내부 이미지 SMARTLOADING2 사용, GIF 이미지의 프레임 간격 200, 출력 위치 정중앙
SmartX.SmartSplash.Start(SmartX.SmartSplash.BuiltInLoadingImages.SMARTLOADING2, 200);
// -- 프로그램 중간에 기능 전환 시 시간이 오래 걸리는 코드 --
// -- 중략 -- 코드가 종료되는 시점에서 스플래쉬 종료
SmartX.SmartSplash.Finished();
```

3) SmartSplash 인터페이스 설명

SmartSplash Component Interface		
속성		
AnimationInterval : int	CenterPosition : bool	Left : int
LoadingImagePathname : string	Top : int	
메소드		
Finish() : void	Finished() : static void	Start() : void
Start(SmartSplash.BuiltInLoadingImages eBuiltinLoadingImage, int iAnimation Interval) : static void (+3개 오버로드)		

프로퍼티(속성) AnimationInterval

애니메이션(Animation) GIF 파일의 프레임 간 출력 지연 시간을 설정합니다. 설정값은 200ms 이상으로 설정하시기 바랍니다.

- int : 동영상 위치 (단위 : ms)

C# 사용법

```
// 애니메이션의 프레임 간격을 설정
smartSplash1.AnimationInterval = 300;
```

VB 사용법

```
smartSplash1.AnimationInterval = 300
```

프로퍼티(속성) CenterPosition

스플래쉬 이미지(애니메이션 GIF)의 출력 위치를 설정하지 않고 자동으로 화면의 정중앙에 표시하도록 설정합니다.

- bool : 화면의 정중앙에 스플래쉬 이미지 출력 유무
  - true : 정중앙에 출력
  - false : Left와 Top 속성값에 따라서 출력

참고 "Left, Top 속성" 내용을 참고하시기 바랍니다.

C# 사용법

```
smartSplash1.CenterPosition = true;
```

VB 사용법

```
smartSplash1.CenterPosition = true
```

프로퍼티(속성) Left, Top

스플래쉬 이미지(애니메이션 GIF)의 출력 위치를 설정합니다.

- Left : 출력 위치의 X축을 설정합니다.
- Top : 출력 위치의 Y축을 설정합니다.
- int : X, Y축값 (기준점 : 좌측 상단)

C# 사용법

```
smartSplash1.Left = 300;
smartSplash1.Top = 200;
```

VB 사용법

```
smartSplash1.Left = 300
smartSplash1.Top = 200
```

프로퍼티(속성) LoadingImagePathname

Image

스플래쉬 이미지(애니메이션 GIF) 파일의 이름과 경로를 설정합니다. 아래와 같이 경로 값을 설정하시면 SmartSplash 포함된 Splash 애니메이션 이미지를 사용하실 수 있습니다.

SmartLoading1	SmartLoading2
	

SmartLoading3	SmartLoading4
	

Image 이미지 사용 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 “이미지 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
// Flash Disk에 있는 스플래쉬 이미지 사용
smartSplash1.LoadingImagePathname = “Flash Disk₩₩Run₩₩Loading.gif”;
// 내부에 포함된 스플래쉬-1 이미지를 사용
smartSplash1.LoadingImagePathname = “SmartLoading1”;
// …중략…
// 내부에 포함된 스플래쉬-4 이미지를 사용
smartSplash1.LoadingImagePathname = “SmartLoading4”;
```

VB 사용법

```
smartSplash1.LoadingImagePathname = “Flash Disk₩₩Run₩₩Loading.gif”
smartSplash1.LoadingImagePathname = “SmartLoading1”
smartSplash1.LoadingImagePathname = “SmartLoading4”
```

메소드(함수) Start

스플래쉬 이미지(애니메이션 GIF)의 출력을 시작합니다.

◉ void Start()

중요 Start() 메소드의 호출은 장치 응용 프로그램 실행 시 가장 먼저 호출되어야 하는 메소드입니다. 따라서 반드시 폼의 생성자 함수에서 실행하시기 바라며, 장치 응용 프로그램이 실행 완료하여 화면에 출력되는 시점에 Finish() 메소드를 호출하여 스플래쉬를 종료하시기 바랍니다.

사용법

권장 정적 메소드인 Start() 사용을 권장합니다.

메소드(함수) Finish

스플래쉬 이미지(애니메이션 GIF)의 출력을 종료합니다.

◉ void Finsh()

중요 장치 응용 프로그램이 실행 완료하여 화면에 출력되는 시점에 Finish() 메소드를 호출하여 스플래쉬를 종료하시기 바랍니다. 폼 로드 이벤트에서 처리할 내용이 많은 경우 Finish() 메서드의 호출 위치를 변경하여 최적의 상태를 찾아 호출하시기 바랍니다.

사용법

권장 정적 메소드인 Finished() 사용을 권장합니다.

정적 메소드(함수) Start 권장

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

스플래쉬 이미지(애니메이션 GIF)의 출력을 시작합니다. 반드시 객체를 생성해야 사용이 가능했던 기존 방식을 개선한 Static 메소드로 기존 Start() 메소드와 동일하게 동작합니다.

중요 Start() 정적 메소드의 호출은 장치 응용 프로그램의 실행 시 가장 먼저 호출되어야 하는 메소드입니다. 따라서 반드시 폼의 생성자 함수에서 실행하시기 바라며, 장치 응용 프로그램이 실행 완료하여 화면에 출력되는 시점에 Finished() 정적 메소드를 호출하여 스플래쉬를 종료하시기 바랍니다.

- `static void Start(SmartSplash.BuiltInLoadingImages eBuiltinLoadingImage, int iAnimationInterval)`
- `static void Start(string strLoadingImagePathname, int iAnimationInterval)`
- `static void Start(SmartSplash.BuiltInLoadingImages eBuiltinLoadingImage, int iAnimationInterval, int iLeft, int iTop)`
- `static void Start(string strLoadingImagePathname, int iAnimationInterval, int iLeft, int iTop)`

[인자]

- `SmartSplash.BuiltInLoadingImages eBuiltinLoadingImage` : 내부에 포함된 스플래쉬 이미지 중 사용할 이미지를 설정
- `int iAnimationInterval` : 스플래쉬 이미지의 프레임 간격 설정
- `string strLoadingImagePathname` : 외부 경로의 스플래쉬 이미지를 사용할 경우 경로를 설정
- `int iLeft` : 스플래쉬 이미지가 출력될 X축 위치 설정 (인자값을 설정하지 않을 경우 중앙에 출력)
- `int iTop` : 스플래쉬 이미지가 출력될 Y축 위치 설정 (인자값을 설정하지 않을 경우 중앙에 출력)

[표] 내부에 포함된 스플래쉬 이미지 열거값에 따른 출력 이미지

SmartLoading1	SmartLoading2
	
SmartLoading3	**SmartLoading4**
	

C# 사용법

```
public Form1()
{
  // 사용법1 : 내부 이미지 사용, Gif 이미지의 프레임 간격 200
  SmartSplash.Start(SmartSplash.BuiltInLoadingImages.SMARTLOADING1, 200);
}
private void Form1_Load(object sender, EventArgs e)
{
  // 폼 Load 이벤트 함수가 종료되는 시점에서 스플래쉬 종료
  SmartSplash.Finished();
}
```

VB 사용법

```
Private Sub New()
  SmartSplash.Start(SmartSplash.BuiltInLoadingImages.SMARTLOADING1, 100)
End Sub
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
```

```
    SmartSplash.Finished()
End Sub
```

메소드(함수)	Finished	권장

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

스플래쉬 이미지(애니메이션 GIF)의 출력을 종료합니다. 반드시 객체를 생성해야 사용이 가능했던 기존 방식을 개선한 Static 메소드로 사용을 권장하며, 기존 Finish() 메소드와 동일하게 동작합니다.

중요 장치 응용 프로그램이 실행 완료하여 화면에 출력되는 시점에 Finished() 정적 메소드를 호출하여 스플래쉬를 종료하시기 바랍니다. 폼 로드 이벤트에서 처리할 내용이 많은 경우 Finish() 정적 메서드의 호출 위치를 변경하여 최적의 상태를 찾아 호출하시기 바랍니다.

◉ `static void Finished()`

사용법

참고 Start() 정적 메소드의 사용법을 참고하시기 바랍니다.

4) SmartSplash 예제 사용하기

SmartSplash를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartSplash"

8. SmartBootLogo

SmartBootLogo는 IEC-Series 제품의 부팅 시 화면에 출력되는 이미지(BootLogo)를 사용자가 변경할 수 있도록 해주는 컴포넌트가 아닌 프로그램입니다.

■ 제품 BootLogo 이미지 변경 지원
■ BootLogo 이미지 변환 프로그램 지원

주의 BootLogo 변경 시 최초의 Logo로 복원하는 기능은 없습니다. 만약 BootLogo 변경 후 원상태로 복구하기를 원하는 경우 HNS에 제품을 보내주시기 바랍니다. (단, 복구 비용이 발생합니다.)

1) 제품별 BootLogo 이미지 사양

이미지 사양을 제품별 이미지 사이즈(해상도) 조건과 이미지 형식 조건에 맞게 설정하여 부팅 이미지를 제작하시기 바랍니다.

중요 제품별 이미지 사양

1. 제품별 이미지 사이즈(해상도) 조건
아래 표를 참고하여 사용하시는 제품의 해상도를 확인 후 해상도와 동일하게 이미지 사이즈를 제작합니다.

2. 이미지 형식 조건
이미지를 BMP(Bitmap) - 16Bit(R5 G6 B5) 형식으로 제작합니다.

ex) IEC1000-07을 사용하신다면 해상도는 800 * 480이므로 가로 800, 세로 480인 BMP 16Bit 565 RGB 형식으로 이미지를 제작하시면 됩니다.

[표] IEC-Series 제품별 해상도

제품	LCD Size	해상도
IEC1000Lite-43 [B1 or B2] IEC266Lite-43 [B1 or B2]	10.9cm (4.3")	480 * 272
IEC1000Lite-56 [B1 or B2] IEC667Lite-56 [B1 or B2] IEC266Lite-56 [B1 or B2]	14.2cm (5.6")	640 * 480
IEC1000-07N [B1 or B2] IEC1000Lite-07N [B1 or B2]	17.7cm (6.95")	800 * 480
IEC1000-07 [B1 or B2] IEC1000Lite-07 [B1 or B2] IEC667-07 [B1 or B2] IEC667Lite-07 [B1 or B2] IEC266-07 [B1 or B2] IEC266Lite-07 [B1 or B2]	17.8cm (7")	800 * 480
IEC1000-08 [B1 or B2] IEC1000Lite-08 [B1 or B2] IEC667-08 [B1 or B2] IEC667Lite-08 [B1 or B2] IEC266-08 [B1 or B2]	20.3cm (8")	800 * 600
IEC1000-102 [B1 or B2] IEC1000Lite-102 [B1 or B2] IEC667-102 [B1 or B2] IEC667Lite-102 [B1 or B2] IEC266-102 [B1 or B2]	25.9cm (10.2")	800 * 480

IEC1000-104 [B1 or B2] IEC1000Lite-104 [B1 or B2] IEC667-104 [B1 or B2] IEC667Lite-104 [B1 or B2]	26.4cm (10.4")	800 * 600
IEC1000-104H [B1 or B2]	26.4cm (10.4H")	1024 * 768
IEC1000-150 [B]	38.1cm (15")	1024 * 768
IEC1000XGA- I	XGA (모니터 미지원)	1024 * 768

2) BootLogo 이미지 제작 및 변환 방법

SmartBootLogo를 사용하여 제품의 BootLogo 이미지를 변경할 경우 사용하시는 제품의 LCD 해상도에 따른 이미지 형식이 맞게 설정되어 있어야 합니다. 따라서 이미지 형식을 변환하기 위한 작업이 필요하며 이 과정에서 포토샵을 사용해야 합니다. 만약 포토샵이 없는 경우 자사에서 제공하는 BootLogo 이미지 변환 프로그램을 사용하여 쉽고 편리하게 형식을 변환할 수 있습니다.

주의 사용하는 제품에 맞는 BootLogo 이미지 사양을 제작하시기 바랍니다. 만약 맞지 않게 제작하실 경우 Boot Logo 이미지가 올바르게 적용되지 않을 수 있습니다.

CASE-1 BootLogo 이미지 변환 프로그램을 사용하는 경우

[STEP-1] "BootLogo 이미지 변환 프로그램" 다운로드 후 실행 → [이미지 열기] 클릭

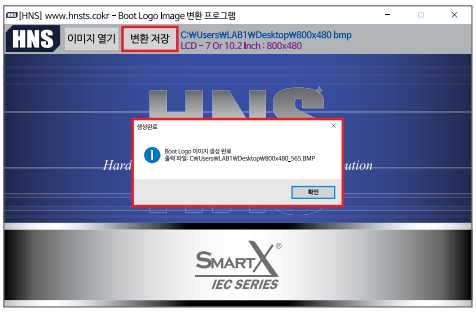
다운로드 경로 ▶ 홈페이지 → 자료실 → 제품관련 → "BootLogo 이미지 변환 프로그램"

사용하시는 제품의 LCD 해상도에 따른 이미지 파일을 불러옵니다. 이미지 확장자는 BMP 16Bit, BMP 24Bit, PNG, JPG, GIF를 지원합니다.

[STEP-2] LCD 해상도 및 인치 확인

[STEP-3] [변환 저장] 클릭

"변환 저장"을 클릭하면 자동으로 BootLogo 이미지를 생성합니다. 이미지의 이름은 "원본이미지이름_565.BMP"의 형태로 생성됩니다.



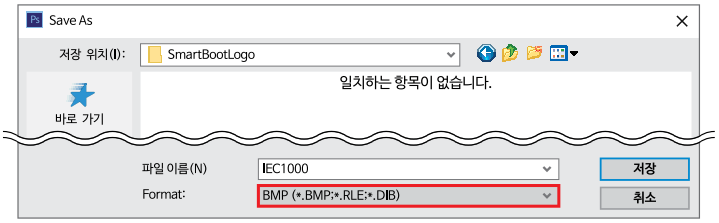
CASE-2 포토샵을 사용하는 경우

[STEP-1] 포토샵 열기 - [메뉴] - [File] - [Open]

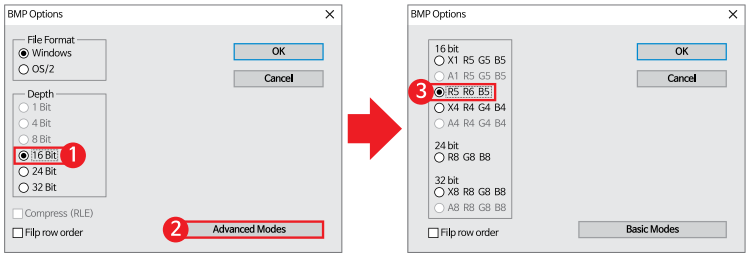
사용하시는 제품의 LCD 해상도에 따른 이미지 파일을 불러옵니다.

[STEP-2] [메뉴] - [File] - [Save As] - [Format] BMP 선택

이미지 파일의 Format을 BMP (*.BMP;*.RLE;*.DIB)로 선택합니다.



[STEP-3] [Depth] 16Bit 선택- [Advanced Modes] 클릭 - [16Bit] R5 G6 B5 선택 - [OK] 클릭



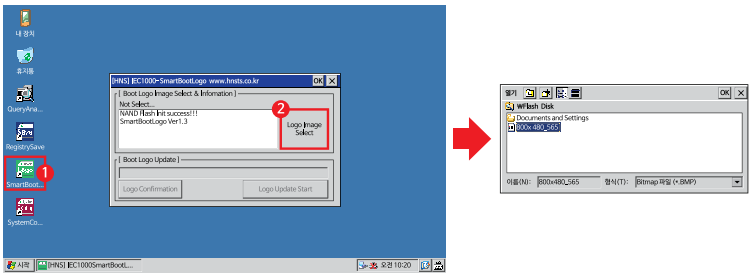
3) BootLogo 적용하기

IEC-Series에 맞게 포토샵 또는 BootLogo 이미지 변환 프로그램을 이용해 이미지를 제작하셨다면 SmartBootLogo를 사용하여 제품에 적용하시기 바랍니다.

[STEP-1] 변환한 BootLogo 이미지 파일을 USB Memory 또는 SD Card Memory에 저장 후 제품에 삽입

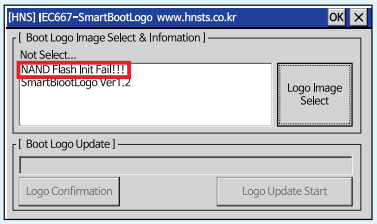
[STEP-2] [SmartBootLogo] 실행 - [Logo Image Select] 클릭 - 이미지 파일 선택 - [OK] 클릭

IEC-Series 제품의 바탕화면에 있는 "SmartBootLogo"를 실행해 [STEP-1]에서 삽입한 BootLogo 이미지 파일을 선택합니다.

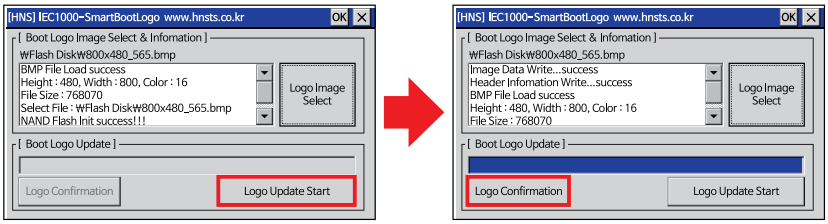


참고 IEC667-Series 제품으로 SmartBootLogo 실행 시 참고사항

IEC667-Series의 경우 SmartBootLogo 실행 시 [Boot Logo Image Select & Infomation]에서 메시지 창에 "NAND Flash Init success!!!"가 아닌 "NAND Flash Init Fail!!!"라는 문구가 뜨게 됩니다. 이 경우 무시하고 진행하시기 바랍니다.



[STEP-3] [Logo Update Start] 클릭 – [Logo Confirmation] 클릭 – 부팅 이미지 적용 상태 확인



주의 업데이트 중에는 전원을 반드시 유지하시기 바랍니다.

주의 SmartBootLogo에서는 애니메이션 형식의 BootLogo는 지원하지 않습니다.

Essential Learning Section의 내용은 여기까지이며, SmartX Framework를 적용할 경우 반드시 알아야 할 중요한 내용을 위주로 구성했습니다. 반드시 숙지하여 장치 응용 프로그램 개발에 참고하시기 바랍니다.

Section
Reference Guide

SmartX Framework 프로그래밍 가이드의 내용이 방대하여 효과적인 학습과 적용을 위해 두 개의 Section으로 구성했습니다.
Reference Guide Section은 필요한 부분을 선택적으로 참고하여 개발에 응용하시기 바랍니다.

Reference Guide Section

Part-VI. 하드웨어 장치 안내

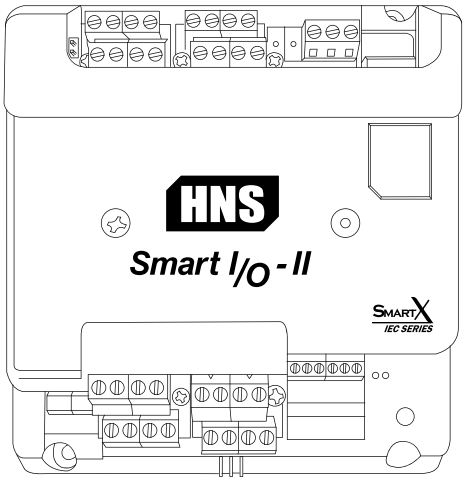
IEC-Series는 아래와 같은 하드웨어 기능이 내장되어 있습니다. 이러한 기능들을 외부에서 쉽게 인터페이스할 수 있도록 확장 포트(Extension Port - I, II)로 구성했습니다.

참고 SmartX Framework를 활용한 확장 포트(Extension Port - I, II)의 제어 실험 및 테스트를 편리하게 하시려면 SmartKit를 구매하시기 바랍니다.

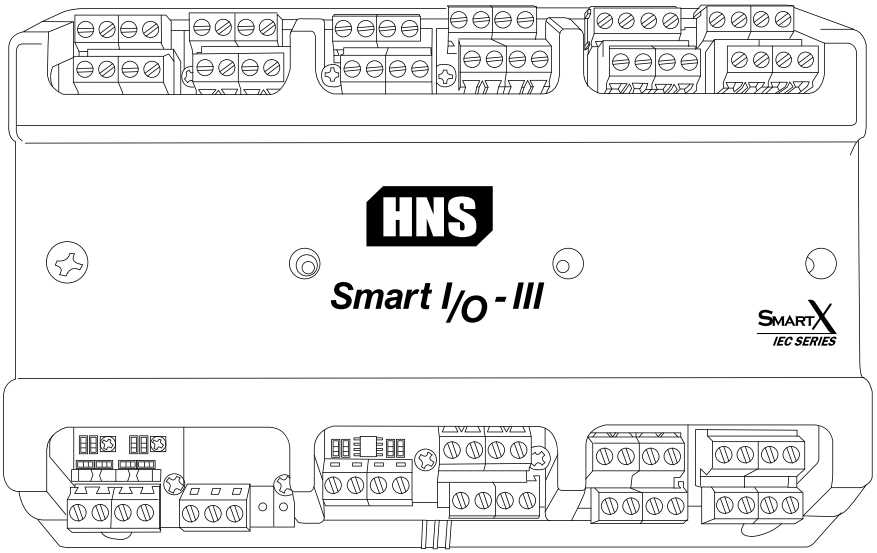
[표] 제품별 인터페이스 장치 사양

장치기능	제품군	채널 수	전기적 사양
GPIO	IEC266-Series	16EA	입력/출력 전압 : DC 0 or 3.3V - 전류 : 7mA Port-B 6, 7 Pin Open Drain 입/출력 **참고** 초기 입/출력 상태에 따른 출력값은 SmartGPIO를 참고하시기 바랍니다.
	IEC266Lite-Series		
	IEC667-Series	60EA	
	IEC667Lite-Series	16EA	
	IEC1000-Series	60EA	
	IEC1000Lite-Series	16EA	
ADC	IEC266-Series	4EA	입력 전압 : DC 0 ~ 3.3V 분해능 : 10Bit(0 ~ 1023)
	IEC266Lite-Series		
	IEC667-Series	4EA	입력 전압 : DC 0 ~ 3.3V 분해능 : 12Bit(0 ~ 4095)
	IEC667Lite-Series		
	IEC1000-Series	6EA	
	IEC1000Lite-Series	4EA	
PWM	IEC-Series	2EA	0.25Hz ~ 66MHz Carrier Frequency 출력 전압 : DC 0 or 3.3V
I2C	IEC-Series	1EA	Port-B 6, 7 Pin Open Drain 입/출력 입력/출력 전압 : DC 0 or 3.3V
SerialPort	IEC266-Series	3 EA	COM1 : RS485(S/W 방식 자동 송/수신) or DC 5V - TTL COM2,3 : RS232
	IEC266Lite-Series		COM1 : DC 3.3V - TTL(RS485 Option SN485-Board) COM2,3 : RS232
	IEC667-Series IEC1000-Series	4 EA	COM1 : RS485(H/W 방식 자동 송/수신) or DC 5V - TTL COM4 : DC 5V - TTL COM2,3 : RS232
	IEC667Lite-Series IEC1000Lite-Series		COM1 : RS485(S/W 방식 자동 송/수신) or DC 5V - TTL COM4 : DC 5V - TTL COM2,3 : RS232
DAC	IEC-Series	2EA Option	출력 전압 : DC 0 ~ 5V or 0 ~ 10V 분해능 : 12Bit(0 ~ 4095)
LAN	IEC266-Series	1EA	10Mbps
	IEC667-Series		
	IEC1000-Series		100Mbps
	IEC266Lite-Series IEC667Lite-Series IEC1000Lite-Series	-	미지원

1. IEC-Series 의 Extension Port - I, II 활용 옵션 제품



Smart I/O - II



Smart I/O - III

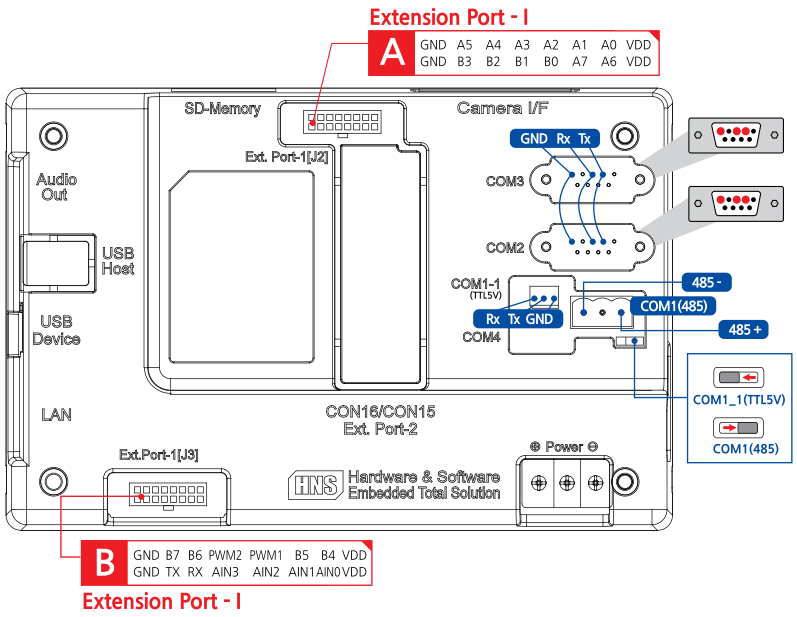
참조 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → Smart I/O Series 매뉴얼"을 참조하시기 바랍니다.

2. 인터페이스 콘넥터 핀 명칭

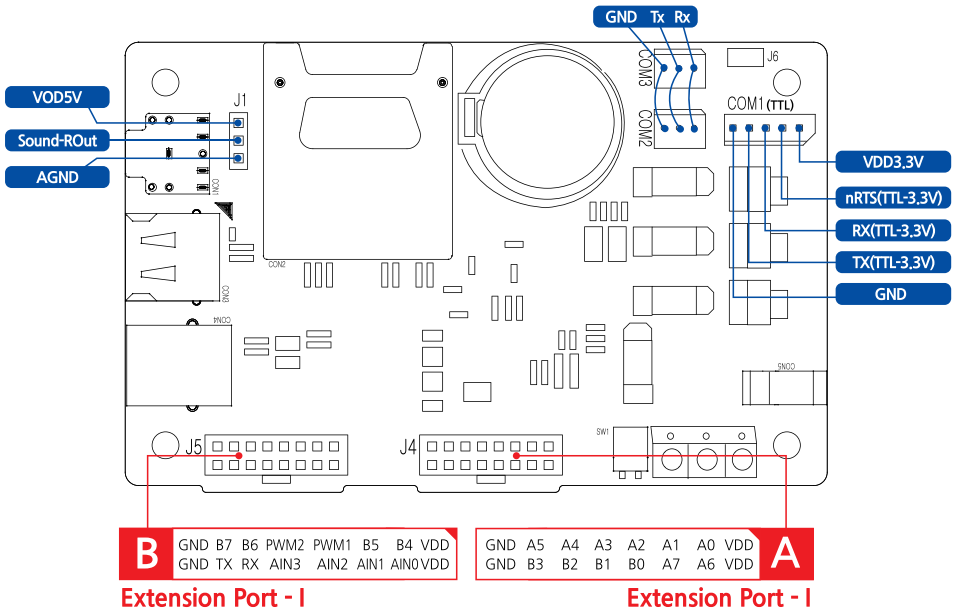
참고 콘넥터별 핀 기능관련은 “확장 포트(Extension Port – I,II) 콘넥터별 핀 기능 정의”를 참고하시기 바랍니다.

참조 IEC-Series 제품별 Extension Port 위치(Dimension)도면 안내는 “홈페이지(www.hnsts.co.kr) → 자료실 → 도면 및 승인원”을 참조하시기 바랍니다.

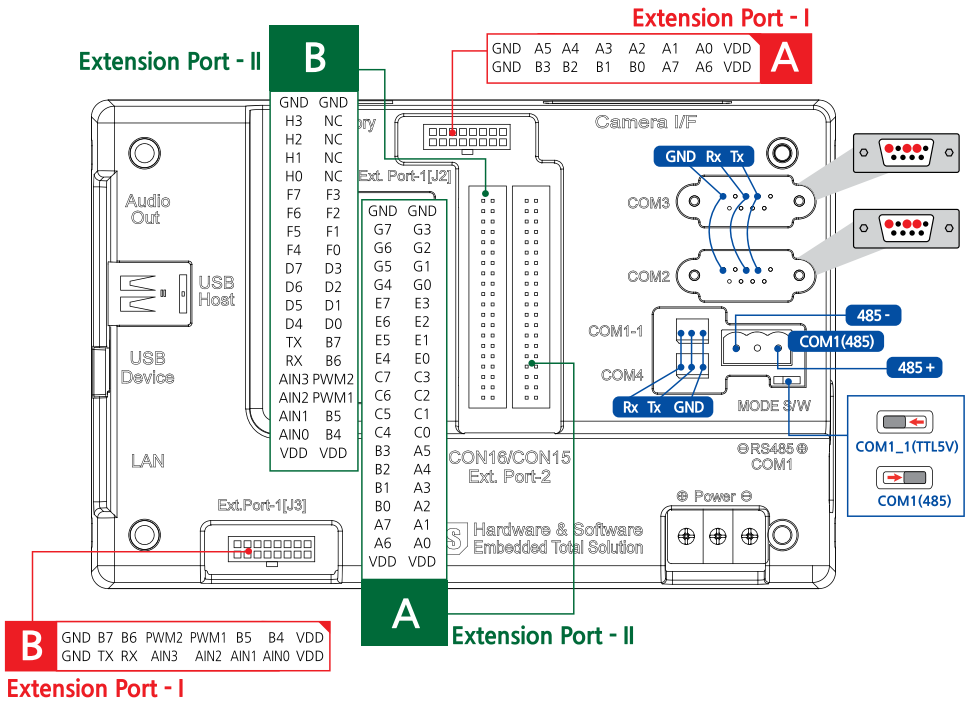
1) IEC266-Series 인터페이스 콘넥터 위치 및 핀 기능 정의



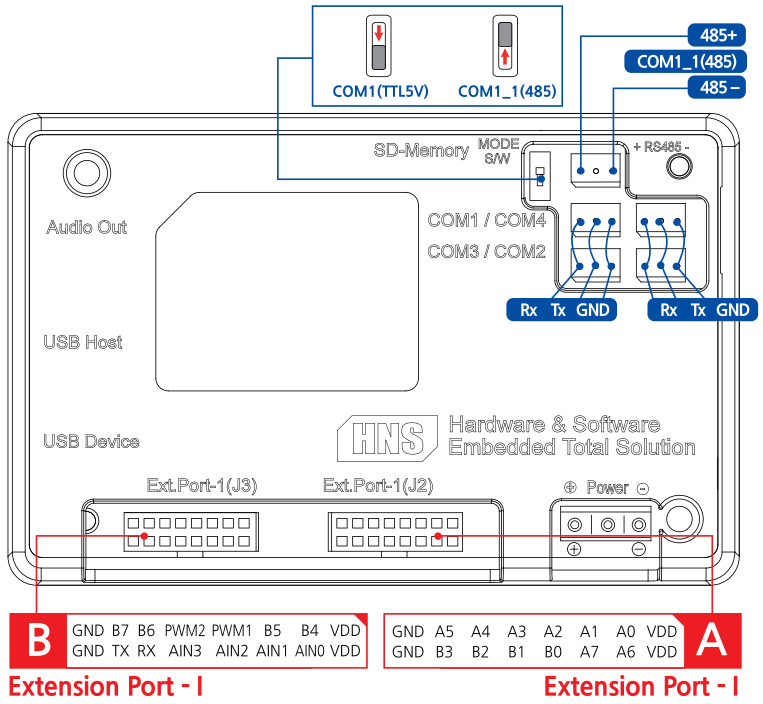
2) IEC266Lite-Series 인터페이스 콘넥터 위치 및 핀 기능 정의



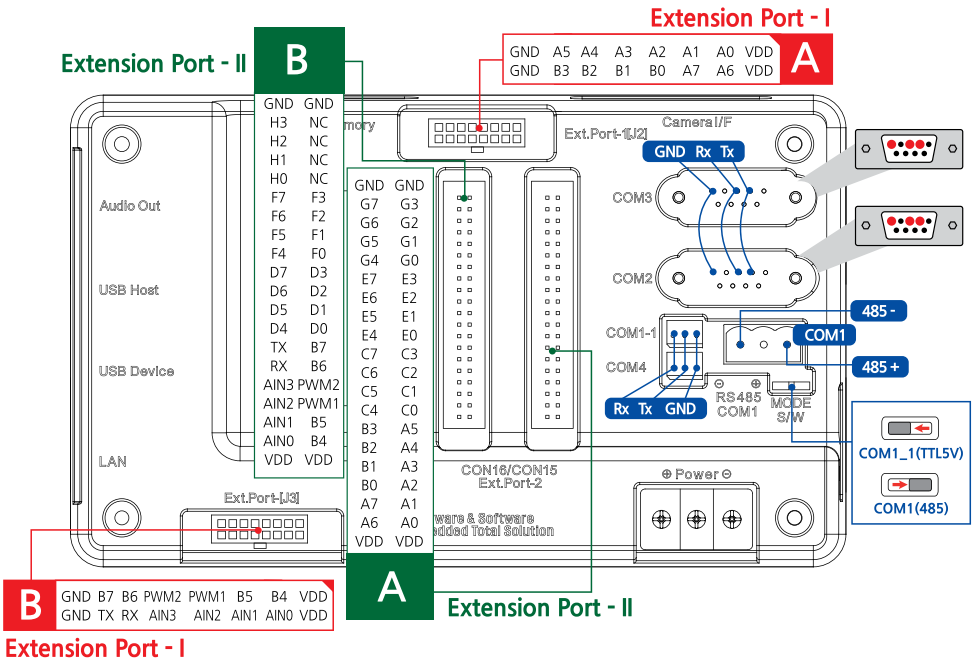
3) IEC667-Series 인터페이스 콘넥터 위치 및 핀 기능 정의



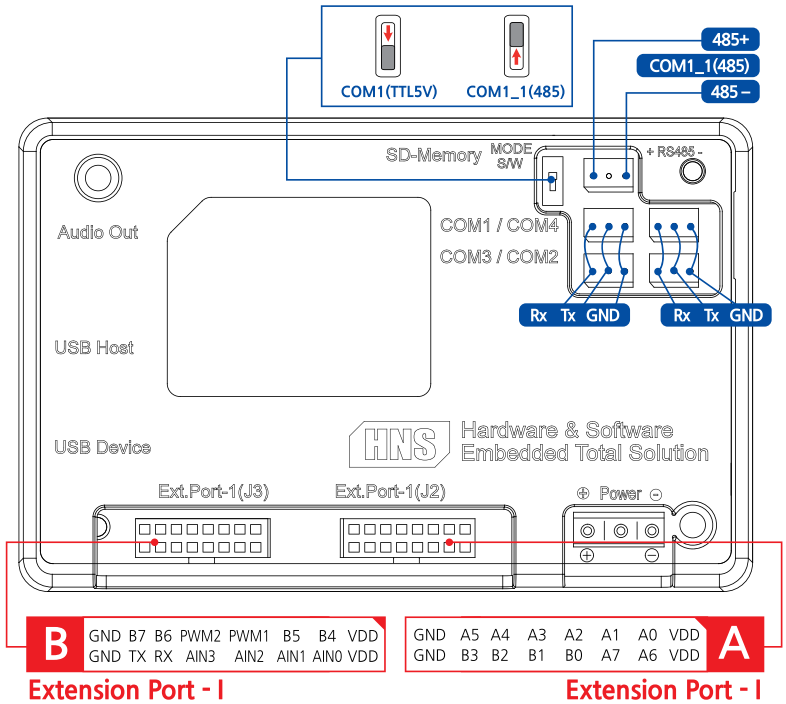
4) IEC667Lite-Series 인터페이스 콘넥터 위치 및 핀 기능 정의



5) IEC1000-Series 인터페이스 콘넥터 위치 및 핀 기능 정의

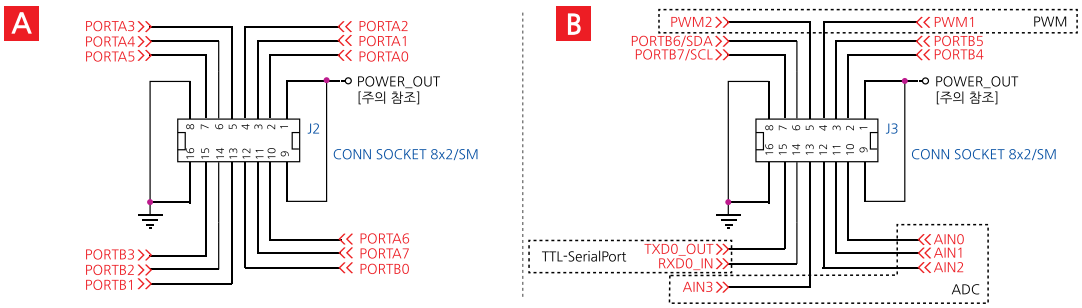


6) IEC1000Lite-Series 인터페이스 콘넥터 위치 및 핀 기능 정의



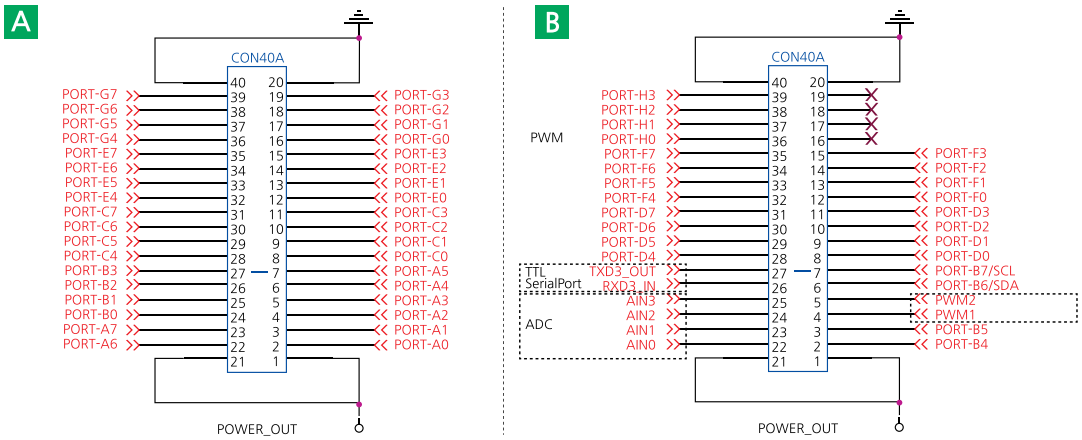
3. 확장 포트(Extension Port - I, II) 콘넥터별 핀 기능 정의

1) Extension Port - I 확장 포트

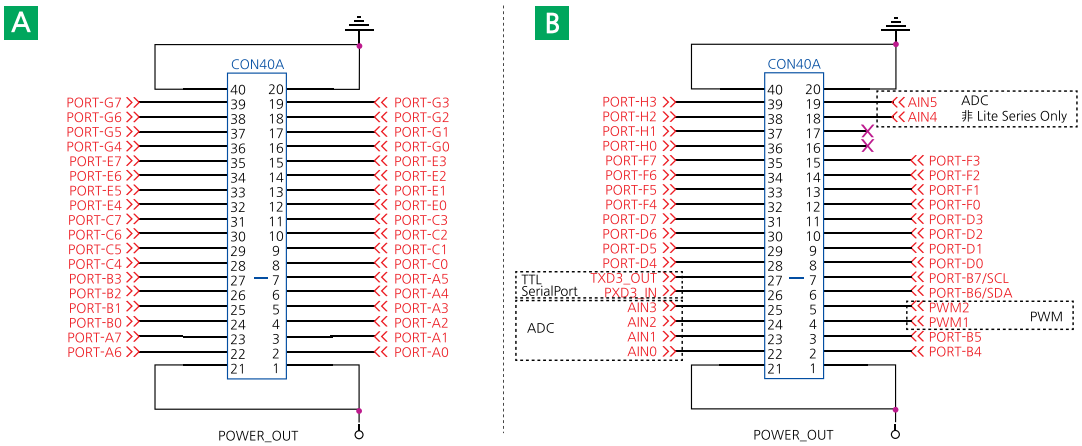


2) Extension Port - II 확장 포트

2-1) IEC667 - Series



2-2) IEC1000 - Series



3) 제품별 Extension Port - I, II 사양

IEC-Series는 Lite, 非 Lite 여부에 따라 지원하는 Extension Port가 다르며, Extension Port – II는 Extension Port – I을 포함하고 있습니다. 아래 주의사항과 표를 확인하시기 바랍니다.

[표] 제품별 Extension Port - I, II 기능(전압/채널 수)

IEC-Series	266	266Lite	667	667Lite	1000	1000Lite
Extension Port	I		I, II	I	I, II	I
GPIO	16EA(3.3V)		60EA(3.3V)	16EA(3.3V)	60EA(3.3V)	16EA(3.3V)
I2C	1EA					
SerialPort	COM1 (TTL 3.3V)		COM4 (TTL 5V)		COM4 (TTL 5V)	
PWM	2EA(3.3V)					
ADC	4EA – 10Bit (0 ~ 3.3V)		4EA – 12Bit (0 ~ 3.3V)		6EA – 12Bit (0 ~ 3.3V)	4EA – 12Bit (0 ~ 3.3V)

주의 Extension Port 및 IEC-Series에 따른 VDD 전압 주의사항

Extension Port의 특정 Pin은 제품에 입력되는 전원(터미널)단자와 직접 연결되어 있어 결선에 주의가 필요합니다. 아래 표를 확인하시기 바랍니다.

[표1] Extension Port - I

IEC-Series	Pin	출력 전압 (제품 인가 전압과 동일)
非 Lite - Series	Extension Port – I– A, Extension Port – I– B Pin1, 9(VDD)	DC 9 ~ 24V
Lite - Series		DC 5V

[표2] Extension Port - II

IEC-Series	Pin	출력 전압 (제품 인가 전압과 동일)
非 Lite - Series	Extension Port – II – A, Extension Port – II – B Pin1, 21(VDD)	DC 9 ~ 24V

참고 SmartGPIO의 “IEC-Series Port 초기 상태값” 표를 참고하시기 바랍니다.

Part-VII

하드웨어 장치 제어 컴포넌트

하드웨어 장치 제어(Hardware Device Control) 컴포넌트는 IEC-Series 제품의 하드웨어 제어에 따른 펌웨어 및 디바이스 드라이버의 개발 없이 하드웨어를 제어할 수 있도록 다양한 컴포넌트를 지원하며, 편리하게 프로젝트에 적용할 수 있도록 컴포넌트의 인터페이스를 직관적으로 구성하였습니다.

Part-Ⅶ. 하드웨어 장치 제어 컴포넌트

1. SmartGPIO

SmartGPIO는 IEC-Series에서 입출력 기능을 간편하게 구현할 수 있도록 지원하는 컴포넌트입니다.
기본적인 입출력 기능과 고속 입력되는 신호를 캡처하는 기능을 지원하며, 사용자가 원하는 모드(입력, 출력)를 설정하여 제어 가능하도록 구현되어 있습니다. 또한, SmartI/O-Series와 Block-Series를 조합해 입출력 기능을 구성하여 사용할 수 있습니다.

- GPIO 입출력 기능(입력 출력 방향 설정) 지원
- 최대 A~H 8개의 Port(60Pin) 사용 가능(제품에 따라 다름)
- SmartI/O-Series와 연동 지원
- 입력 신호 감지 Event 기능 지원
- 고속 입력 Capture 기능 지원
- 프로그래밍에 편리한 각종 기능 및 인터페이스 지원

참고 콘넥터 핀 정보에 관한 설명은 "Part-VI. 하드웨어 장치 안내"를 참고하시기 바랍니다.

주의 SmartGPIO 사용 시 주의사항

1. MDI 구조에서 SmartGPIO를 사용할 경우 하나의 프로그램(솔루션)에서 하나의 SmartGPIO만 생성할 수 있습니다. 폼이 다른 경우에는 다른 폼에 SmartGPIO의 참조를 넘겨서 처리하시기 바랍니다.

2. SmartGPIO와 I2C를 함께 사용하는 경우 프로그램이 정상 동작하지 않을 수 있습니다.
이 경우 SmartGPIO의 속성 중 Port B의 DIR6, DIR7을 OUTPUT으로 설정하기 바랍니다.

3. SmartGPIO와 SmartDAC를 함께 사용하는 경우에는 SmartDAC 출력이 정상적으로 이루어지지 않을 수 있습니다. 디자이너 코드에서 SmartDAC의 초기화 코드 순서를 SmartGPIO 초기화 코드 이후에 실행되도록 강제로 변경해 주어야 합니다.

4. Extension Port-I/II-A, B의 Pin 1, 9, 21은 메인 전원과 연결되어 있어 잘못 연결하실 경우 보드의 파손이 생길 수 있으니 반드시 Pin 연결 시 주의하시기 바랍니다.

1) SmartGPIO Port와 Extension Port의 관계

IEC-Series는 제품 후면에 위치한 Extension Port로 입출력 기능을 수행합니다. Extension Port는 Ⅰ, Ⅱ가 있으며, Extension Port-Ⅰ은 Port-A, B를 지원하여 총 GPIO 16EA를 사용할 수 있고, Extension Port-Ⅱ 는 Port-A ~ H를 지원하여 총 GPIO 60EA를 사용할 수 있습니다. 단, Port-H는 4개만 사용할 수 있습니다. IEC-Series 별로 지원되는 Extension Port가 상이하니 아래 표를 확인하시기 바랍니다.

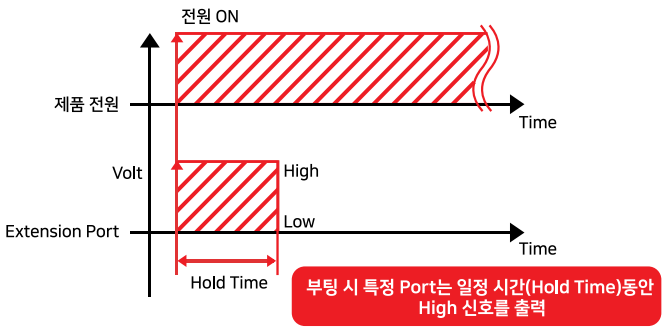
[표] IEC-Series별 지원되는 Extension Port와 GPIO 채널 수

제품	IEC266-Series		IEC667-Series		IEC1000-Series	
	Lite	非Lite	Lite	非Lite	Lite	非Lite
Extension Port	Ⅰ	Ⅰ	Ⅰ	Ⅱ	Ⅰ	Ⅱ
지원 Port	A, B	A, B	A, B	A ~ H (H0 ~ H3)	A, B	A ~ H (H0 ~ H3)
GPIO 개수	16EA	16EA	16EA	60EA	16EA	60EA

※ 각 Port는 0~7번까지 총 8개 사용할 수 있으며, Port - H의 경우 0~3번까지 총 4개 사용할 수 있습니다.

2) 제품별 GPIO-Port 초기 상태값

IEC-Series 부팅 시 특정 Port가 일정 시간 동안(Hold Time) High Level 신호가 출력됩니다. Port-A, B, C, D, E, F, G, H는 각각 Pull Up(Port B의 6, 7Pin은 오픈 드레인 출력) 상태의 Port이며, 초기값이 Floating 되거나 High로 표시되는 경우에는 Pull down/Pull up 저항을 걸어서 사용하시기 바랍니다. 아래 표는 IEC-Series별 각 Port의 초기 상태값을 측정한 결과입니다. 각 Port에 Pull Down 저항 장착 전과 장착 후를 각각 측정하였습니다.



주의 측정 OS는 Standard 기준이며, OS 버전별(OPT, STD, PRO)로 부팅 시간에 차이가 있으므로 측정값이 다소 차이가 날 수 있습니다. 하기의 데이터는 대략적인 데이터로서 약간의 시간 오차가 발생할 수 있습니다.

[표1] IEC266 B15 OS - Port 초기 상태값("※" Pull Down 저항 장착 후 테스트)

Port-Pin		0	1	2	3	4	5	6	7
Port A	Hold Time 구간	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port B	Hold Time 구간	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	HIGH (84ms)	LOW	LOW	LOW	LOW
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW

[표2] IEC667 B23 OS - Port 초기 상태값("※" Pull Down 저항 장착 후 테스트)

Port-Pin		0	1	2	3	4	5	6	7
Port A	Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port B	Hold Time 구간	LOW (8.04s)	LOW (8.04s)	LOW (8.04s)	HIGH (240ms) LOW (8.08s)	HIGH (240ms) LOW (8.08s)	HIGH (240ms) LOW (8.08s)	LOW	LOW

Port B	Hold Time 이후	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	HIGH (315ms)	HIGH (315ms)	HIGH (315ms)	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port C	Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port D	Hold Time 구간	LOW (1.4s)	LOW (1.4s)	LOW (210ms) HIGH (1.6ms) LOW (1.13s)	HIGH (210ms) LOW (1.14s)	HIGH (210ms) LOW (1.14s)	HIGH (210ms) LOW (1.14s)	HIGH (210ms) LOW (1.14s)	LOW (1.4s)
	Hold Time 이후	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH	HIGH
	※Hold Time 구간	LOW	LOW	LOW (210ms) HIGH (1.6ms) LOW (1.13s)	HIGH (250ms)	HIGH (250ms)	HIGH (250ms)	HIGH (250ms)	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port E	Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port F	Hold Time 구간	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)
	Hold Time 이후	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)	LOW (1.4s)
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port G	Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	HIGH (1.4s) LOW (0.3s)
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	HIGH
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW (1.7s)
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
Port H	Hold Time 구간	LOW	LOW	LOW	LOW	–	–	–	–
	Hold Time 이후	LOW	LOW	LOW	LOW	–	–	–	–

Port H	※Hold Time 구간	LOW	LOW	LOW	LOW	-	-	-	-
	※Hold Time 이후	LOW	LOW	LOW	LOW	-	-	-	-

주의 IEC667-Series Port 초기 상태값

IEC667-Series에서 Port-B, Port-D, Port-F, Port-G는 입력으로 사용하는 것을 권장합니다.
Port-B0, 1, 2, 3, 4, 5 Pin은 전원 인가 후 9.3s 경과하여 HIGH로 출력되며, Port-D0 ~ 7 Pin, Port-F0 ~7 Pin,Port-G7 Pin은 전원 인가 후 1.4s 경과하여 HIGH로 출력됩니다. 제시한 Port는 Pull Down 저항을 걸어서 사용하는 경우에는 문제가 되지 않으며, 자세한 사항은 [표2] IEC667 B23 OS - Port 초기 상태값을 참고하시기 바랍니다.

[표3] IEC1000 B1 OS(Windows Embedded CE 6.0) - Port 초기 상태값

Port-Pin		0	1	2	3	4	5	6	7
Port A ~ H	Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 구간	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW
	※Hold Time 이후	LOW	LOW	LOW	LOW	LOW	LOW	LOW	LOW

3) 출력 기능

SmartGPIO를 사용하면 간단하게 Extension Port의 출력 제어를 할 수 있습니다. 출력 전압은 HIGH 상태일 때 3.3V이며 LOW 상태일 때는 0V가 출력됩니다. 관련 속성 및 속성값 설명은 아래 표를 참고하시기 바랍니다.

[표] 출력 관련 속성 및 속성값

	속성명	속성값	설명
Pin 단위 방향 설정	PORTXDIR0~7	OUTPUT	X Port의 특정 Pin을 출력으로 설정합니다.
Pin 단위 출력 제어	PORTXDATA0~7	True (HIGH 3.3V)	X Port의 특정 Pin을 HIGH로 설정합니다.
		False (LOW 0V)	X Port의 특정 Pin을 LOW로 설정합니다.
Port 단위 방향 설정	PORTXDIRS	0xFF	X Port의 모든 Pin을 출력으로 설정합니다.
Port 단위 출력 제어	PORTXDATAS	0xFF (HIGH 3.3V)	X Port의 모든 Pin을 HIGH로 설정합니다.
		0x00 (LOW 0V)	X Port의 모든 Pin을 LOW로 설정합니다.

참고 [C#] 출력 기능 사용 예시 코드

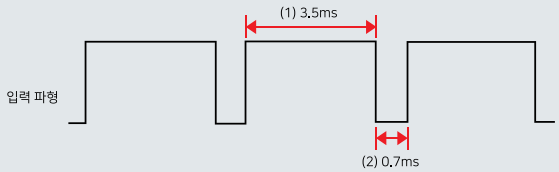
```
private void GPIO_Output()
{
  // Port-A0 Pin의 방향을 출력으로 설정
  smartGPIO1.PORTADIR0 = SmartX.SmartGPIO.PORTDIRS.OUTPUT;
  // Port-B의 모든 Pin의 방향을 출력으로 설정
  smartGPIO1.PORTBDIRS = 0xFF;
  // Port-A0 Pin을 HIGH로 설정
  smartGPIO1.PORTADATA0 = true;
  // Port-B의 모든 Pin을 LOW로 설정
  smartGPIO1.PORTBDATAS = 0x00;
}
```

4) 입력 기능

SmartGPIO는 하드웨어 인터럽트 방식의 입력 처리를 지원하지 않습니다. 따라서 S/W적인 방식으로 입력 신호를 처리하게 되어있습니다. 입력 신호의 감지는 크게 폴링(Polling) 또는 이벤트(Event) 방식으로 처리하며, 각각의 방식은 장단점이 있으므로 개발자가 사용하고자 하는 방식 등을 고려하여 선정하시기 바랍니다. SmartGPIO를 사용하여 Port 입력 상태 값을 확인하는 방법은 총 4가지가 있으며, 이 중 PortDetection() 메소드는 PORTXDATAS 속성 또는 PortXDataChange 이벤트와 달리 이전과 현시점에서 Port의 상태 변화에 따른 결과를 리턴합니다.
아래 중요 내용과 참고 내용을 확인해보시기 바랍니다.

중요 SmartGPIO 입력 주파수 계산 방식

입력 주파수의 해석은 주기뿐만 아니라 Duty Rate도 함께 고려해야 합니다. 즉, Duty Rate가 작은 쪽을 기준으로 계산해야 합니다.



위 그림처럼 Duty Rate가 50이 아닌 파형에서 큰 쪽의 주기는 3.5ms, 작은 쪽의 주기는 0.7ms입니다.
여기서 주파수를 구해보면 다음과 같습니다.

(1) 큰 쪽의 주파수 f1 = 1 / 0.0035(s) = 285.71(Hz)
(2) 작은 쪽의 주파수 f2 = 1 / 0.0007(s) = 1.43(kHz)

따라서 입력 파형의 주파수 적용 값은 1.43kHz입니다.

[표] 포트 입력 상태값 확인 방법

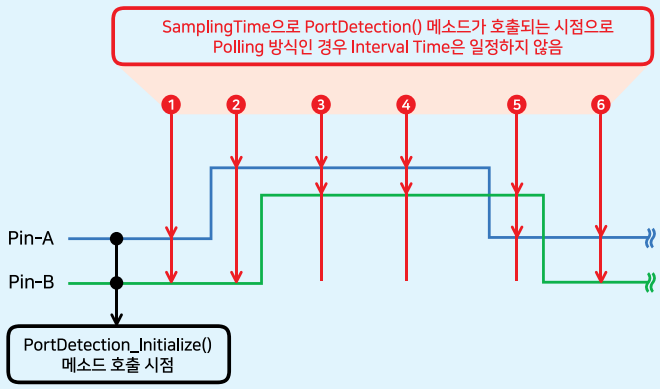
1	PORTXDATAS 속성값	2	PortXDatasChange 이벤트에서 PORTXDATAS 속성값
3	PortDetection() 메서드 결과	4	PortXDatasChange 이벤트에서 PortDetection() 메서드 결과

참고 PortDetection() 메소드 결과값 설명

PortDetection() 메소드 호출 시 eTriggerPins 인자를 사용하지 않는 경우와 사용하는 경우의 결과값이 다르니 아래 표를 확인하여 프로젝트에 적용하시기 바랍니다.

1. eTriggerPins 인자를 사용하지 않는 경우
각 Pin별 이전 상태와 현재 상태값에 의해 Return 값이 결정됩니다. 즉, 임의로 입력되는 단일 Pin의 상태 변화를 감지하는데 사용합니다.

2. eTriggerPins 인자를 사용하는 경우
eTriggerPins 인자값에 사용한 Pin 조합에 따른 이전 상태와 현재 상태값에 의해 Return 값이 결정됩니다. 즉, 특정 Pin(단일, 복수)의 상태 변화를 감지하는데 사용합니다. (Pin 조합은 And 연산 처리됨)



[표] PortDetection() 메소드 호출 시 eTriggerPins 인자값 사용 여부에 따른 각 Pin의 상태

		Pin 번호	호출시점					
			①	②	③	④	⑤	⑥
입력 신호 레벨(High/Low)		Pin-A	0	1	1	1	0	0
		Pin-B	0	0	1	1	1	0
1. eTriggerPins 인자 미사용	HighActive	Pin-A	0	1	0	0	0	0
		Pin-B	0	0	1	0	0	0
	LowActive	Pin-A	0	0	0	0	1	0
		Pin-B	0	0	0	0	0	1
	LevelChange	Pin-A	0	1	0	0	1	0
		Pin-B	0	0	1	0	0	1
2. eTriggerPins 인자 사용 (Pin-A|Pin-B)	HighActive	Pin-A	0	0	1	0	0	0
		Pin-B	0	0	1	0	0	0
	LowActive	Pin-A	0	0	0	0	0	1
		Pin-B	0	0	0	0	0	1
	LevelChange	Pin-A	0	0	1	0	0	1
		Pin-B	0	0	1	0	0	1

[표] 폴링 방식과 이벤트 방식의 차이

방식	폴링(Polling)	이벤트(Event)	
처리 방법	반목문(For, While)	EvtPortXDatasChange	EvtPortXDatasChangeCapture
처리 시점	즉시 처리	즉시 처리	일괄 처리
사용 시점	중요한 데이터를 수신해야 하는 경우	데이터 수신 중 다른 처리가 필요한 경우	일정 시간 동안 빠른 속도로 카운팅을 해야하는 경우
장점	EvtPortXDatasChange 이벤트 보다 입력 처리가 약간 빠름	사용자 인터페이스 응답이 가능하고 코드 처리가 간단함	고속의 입력 신호를 처리 가능하며 신호를 거의 놓치지 않음
단점	Loop문으로 입력을 Check 하므로 UI의 다른 처리가 불가능함	Polling 방식보다 입력 처리가 약간 늦을 수 있음	입력 신호 캡처 중에는 다른 작업의 처리가 어려움
용도	CPU 가용 상태의 영향을 적게 받으면서 일정 시간 동안 입력 신호의 변화에 따른 처리를 해야 하는 경우 사용됩니다.	사용자 인터페이스의 반응 처리를 자유롭게 처리하면서 입력 신호의 변화를 처리하는 경우 사용됩니다.	CPU의 가용 상태의 영향을 적게 받으면서 일정 시간 동안 입력 신호의 변화를 개수하기 위한 용도로 사용됩니다.

참고 "4-3) SmartGPIO 입력 성능 관련"을 참고하시기 바랍니다.

4-1) 폴링(Polling) 방식

폴링(Polling) 방식은 While, For문 등을 이용한 반복문으로 입력 신호를 체크하는 방식입니다. 폴링 방식은 이벤트 방식보다 비교적 정확하게 입력 신호를 캐치할 수 있지만, 입력 신호를 받는 동안은 화면 터치 등 다른 처리가 불가능하다는 단점이 있습니다.

참고 [C#] 폴링(Polling) 방식 사용 예시 코드

```
private void Input_Polling()
{
  // Polling 방식으로 사용하시는 경우 반드시 해당 포트를 초기화하시기 바랍니다.
  smartGPIO1.PortDetection_Initialize(SmartX.SmartGPIO.PORTID.PORTA);
  SmartX.SmartGPIO.PORTPIN iBit;
  while (true)
  {
    // 현재 Pin의 상태를 읽은 후 입력 신호가 변경된 핀을 체크 (LOW -> HIGH)
    iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.TRIGGERMODE.HighAct
```

```
ive)
    // Counting
    if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) == SmartX.SmartGPIO.PORTPIN.PIN0)
    {
      ++iPortA[0];
      break;
    }
    //…중략…
  }
}
```

4-2) 이벤트(Event) 방식

이벤트(Event) 방식은 사용자가 따로 입력 신호를 체크할 필요 없이 내부의 프로세스가 입력 신호의 상태 변화를 체크하는 방식입니다. SmartGPIO에서 이벤트 방식은 입력 신호의 상태 변화를 즉시 캐치하는 EvtPortXDatasChange 이벤트와 고속으로 입력되는 신호의 상태 변화를 캡처하여 큐에 저장시키는 EvtPortXDatasChangeCapture 이벤트로 크게 두 가지가 있습니다.

주의 동일한 Port에서 DatasChange와 DatasChangeCapture 이벤트 동시 사용 불가

동일한 Port에서 EvtPortXDatasChange 이벤트와 EvtPortXDatasChangeCapture 이벤트 두 개를 동시에 사용할 수 없습니다. 저속으로 데이터를 입력받아 실시간 출력으로 사용 시 SmartGPIO의 EvtPortXDatasChange 이벤트를 사용하시고 고속으로 데이터를 입력받아 일괄 출력으로 사용 시 EvtPortXDatasChangeCapture 이벤트를 사용하시기 바랍니다.

참고 SmartGPIO의 “4-3-2) EvtPortXDatasChange vs EvtPortXDatasChangeCapture 비교”를 참고하시기 바랍니다.

4-2-1) EvtPortXDatasChange 이벤트 방식

Port에 입력되는 신호의 변화에 따라 이벤트가 발생되는 방식으로 동작합니다. 따라서 입력 상태 변화에 즉시 응답 처리할 수 있으며, 사용이 편리하다는 장점이 있습니다. 단, 신호 변화에 따른 코드가 이벤트 코드에 위치하게 되어 이벤트 코드의 처리 시간에 의해 응답 처리가 지연되거나 누락될 수 있습니다. 결국 CPU의 가용상태에 따라서 이벤트 처리 및 응답 처리 성능에 영향을 주게 됩니다.

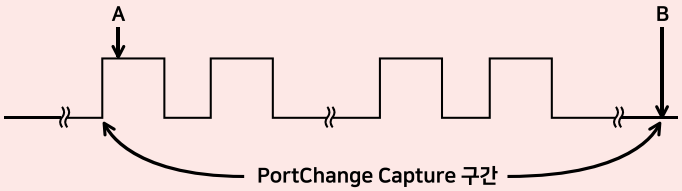
참고 [C#] EvtPortXDatasChange 이벤트 사용 예시 코드

```
private int[] iPortA = new int[8];  // 입력 신호를 카운팅할 배열
private void smartGPIO1_EvtPortADatasChange(object sender, EventArgs e)
{
  SmartX.PORTDataEvtArgs PortDatas = (SmartX.PORTDataEvtArgs)e;  // 인자값으로 받은 핀 상태를 저장
  int iPortDatas = PortDatas.iPortDatas;
  SmartX.SmartGPIO.PORTPIN iBit;
  // 입력 신호가 변경된 핀을 체크 (HIGH -> LOW, LOW -> HIGH)
  iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.TRIGGERMODE.LeveChange,
                                  iPortDatas);
  // Counting
  if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) == SmartX.SmartGPIO.PORTPIN.PIN0)
  {
    lblCount.Text = (++iPortA[0]).ToString();  // 라벨에 데이터를 표시
  }
  // …중략…
}
```

4-2-2) EvtPortXDatasChangeCapture 이벤트 방식

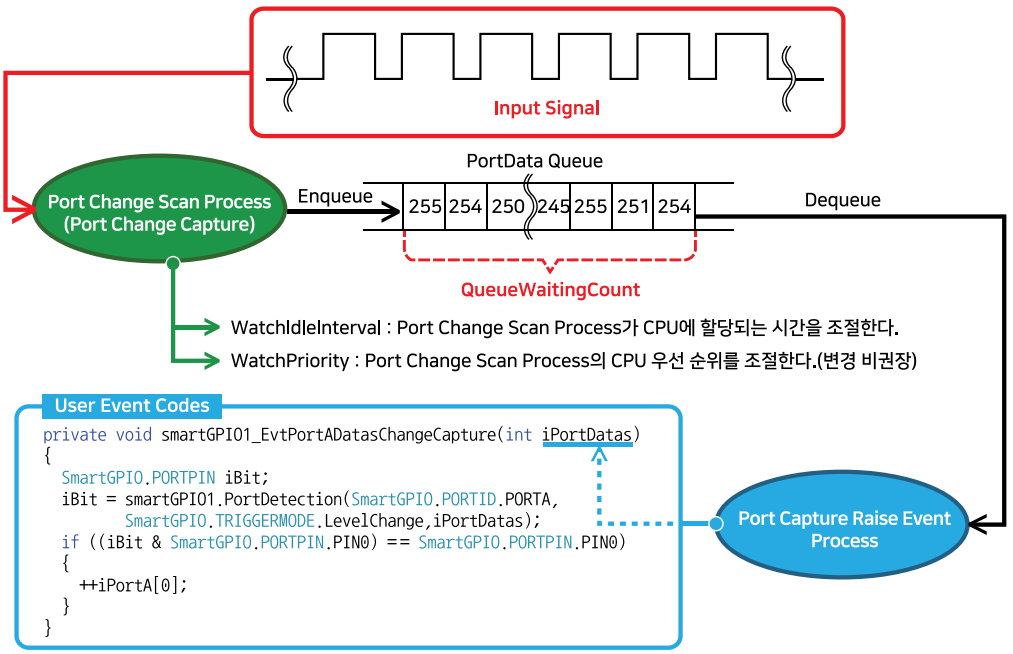
EvtPortXDatasChangeCapture 이벤트는 EvtPortXDatasChange 이벤트를 사용할 때 외부 입력 신호가 이벤트 내 코드 처리 시간보다 더 빠르게 입력되어 입력 신호의 누락이 발생하는 경우를 개선하기 위한 이벤트로,입력 신호 감지 및 감지에 따른 변화 발생 시 이벤트를 발생하지 않고 내부 큐(Queue)에 상태 값을 저장하여 입력 신호의 변화를 캡처(Capture) 하는 방식으로 사용됩니다. 고속으로 입력 신호를 감지하므로 주로 일정 시간 동안 입력 신호의 상태 변화를 카운팅(Counting) 해야 하는 경우 사용됩니다. 단, 입력 신호를 캡처(Port-Scan)하는 동안 CPU의 모든 사용은 Port Change Scan Process(Scan Process)가 사용하게 되어 다른 Process는 일정 시간 동안 CPU를 사용할 수 없게 됩니다. Scan Process의 CPU 점유율과 스캔 간격은 SerPortXWatchPriority(), SetPortXWatchIdleInterval() 메소드를 사용하여 조절할 수 있습니다.

주의 SetPortXWatchIdleInterval(), SetPortXWatchPriority() 메소드 사용 시 주의사항



- SetPortXWatchIdleInterval() 메소드는 Port Change Scan Process가 CPU에 할당되는 시간을 조절하는 메소드로, 입력 신호가 들어오는 시점(A)에 값을 높게 설정하면 Scan Process가 CPU에 할당되는 시간이 높아져 일정 기간 동안 입력되는 신호를 더욱 잘 감지하게 됩니다. 단, PortChange Capture 구간에서는 CPU를 최대한 사용하기 때문에 타작업을 수행하기 어려워집니다. 즉, 일정 시간이 경과하거나 입력 신호가 들어오지 않는 시점(B)에서는 값을 낮게 설정하여 Scan Process가 CPU에 할당되는 시간을 줄여야 타 작업의 수행이 가능하게 됩니다.

- SetPortXWatchPriority() 메소드는 Port Change Scan Process의 CPU 우선 순위를 변경하는 메소드로, 우선 순위를 변경하는 경우 다른 기능이 원활하게 동작하지 않을 수 있으니 변경하지 않는 것을 권장합니다. 만약 우선 순위를 변경해야 하는 경우 프로그램을 충분히 검증하시기 바랍니다.

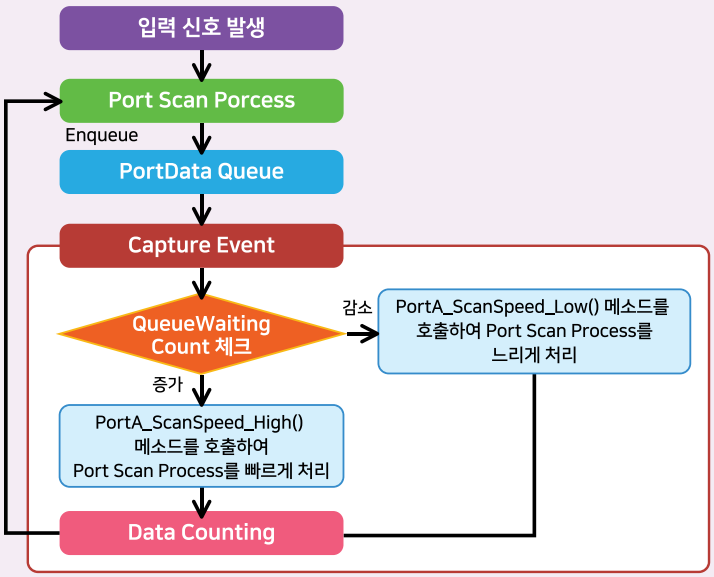


[EvtPortXDatasChangeCapture 이벤트 동작 개요]

권장 EvtPortXDatasChangeCapture 이벤트 권장 사용 방법

EvtPortXDatasChangeCapture 이벤트 사용 시 아래 STEP과 FlowChart 순서로 프로그램을 작성하여 사용하시길 권장합니다.

STEP-1	입력 신호가 감지되기 시작하거나, 사용자의 요청이 있는 경우 WatchIdleInterval 값을 높게 변경하여 Port Change Scan Process(이하 Scan Process)가 CPU에 할당되는 시간을 늘려서 입력되는 신호를 놓치지 않도록 합니다. 적정값으로 설정하여 간헐적으로 Port Capture Raise Event Process(이하 Capture Event)가 발생할 수 있도록 합니다. (적정값 : 1000 ~ 100000)
STEP-2	입력 신호의 상태가 변경 됐을 때 Scan Process는 상태값을 PortDataQueue에 저장합니다.
STEP-3	Capture Event에서 QueueWaitingCount를 체크하여 PortData Queue에 저장된 데이터의 개수가 일정 개수에 도달하거나 더 이상 증가하지 않는 경우 WatchIdleInterval 값을 낮게 변경하여 ScanProcess가 CPU에 할당되는 시간을 줄여서 Capture Event가 빠르게 발생할 수 있도록 합니다. (적정값 : 10 ~ 100)



참고 [C#] EvtPortXDatasChangeCapture 이벤트 사용 예시 코드

```
private int[] iPortA = new int[8];  // 입력 신호를 카운팅할 배열
private uint m_uiQueueCount = 0;   // ChangeCapture 이벤트가 실행된 다음의 큐의 개수
private bool m_bIsQueueIncrease = false;  // PortData Queue의 데이터 증가 여부를 확인할 플래그
private void smartGPIO1_EvtPortADatasChangeCapture(int iPortDatas)
{
  // PortData Queue의 현재 저장된 데이터의 개수와 직전의 개수를 체크
  if (smartGPIO1.GetPortAQueueWaitingCount() > m_uiQueueCount)
  {
    // 플래그를 설정하여 한 번만 설정하도록 한다.
    if (m_bIsQueueIncrease == false)
    {
      // 현재 저장된 데이터의 개수가 더 크다면, 입력 신호가 감지되고 있는 것임
      m_bIsQueueIncrease = true;
      PortA_ScanSpeed_High();  //PortA_ScanSpeed_High() 메소드를 호출한다.
    }
  }
  else
  {
    // 현재 저장된 데이터의 개수가 더 작다면, 입력 신호가 감지되고 있지 않는 것임
    m_bIsQueueIncrease = false;
```

```
    PortA_ScanSpeed_Low();  // PortA_ScanSpeed_Low() 메소드를 호출한다.
  }
  // 신호 감지 시점을 선택하여 수신받는다. (LevelChange,NegativeEdge, PositiveEdge 중 선택)
  SmartX.SmartGPIO.PORTPIN iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA,
                                  SmartX.SmartGPIO.TRIGGERMODE.LevelChange, iPortDatas);
  if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) == SmartX.SmartGPIO.PORTPIN.PIN0)
  {
    lblCount.Text = (++iPortA[0]).ToString();  // 라벨에 데이터를 표시
  }
  // .. 중략 ..
  // 저장된 데이터 개수를 현재 큐에 저장된 데이터 개수로 설정
  m_uiQueueCount = smartGPIO1.GetPortAQueueWaitingCount();
}
// Port-A의 WatchIdleInterval을 증가시키는 메소드
// WatchIdleInterval 값을 높게 변경하여 Port Scan Process 속도를 빠르게 처리한다.(적정값 1000~100000)
private void PortA_ScanSpeed_High()
{
  smartGPIO1.SetPortAWatchIdleInterval(100000);
}
// Port-A의 WatchIdleInterval을 감소시키는 메소드
// WatchIdleInterval 값을 낮게 변경하여 Port Scan Process 속도를 느리게 처리한다.(적정값 10~100)
private void PortA_ScanSpeed_Low()
{
  smartGPIO1.SetPortAWatchIdleInterval(10);
}
```

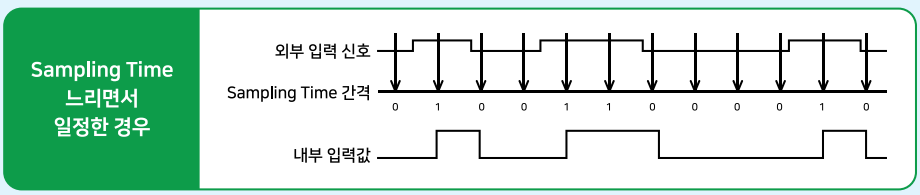
4-3) SmartGPIO 입력 성능 관련

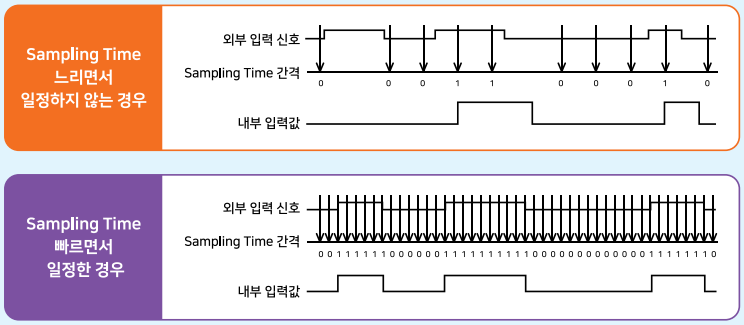
SmartGPIO의 입력 성능은 실시간 동작과 관련해서 보장하기 어려운 부분이 있습니다. 이는 Windows CE가 멀티태스크 구조의 운영체제이므로 이런 문제를 가지고 있으며, 또한 응용 프로그램의 구성에 따라 많은 성능의 차이가 있을 수 있습니다. 일반적으로 실시간성의 IO를 타임 크리티컬하게 처리하려면 모든 기능을 드라이버 단에서 처리하도록 구성해야 합니다. 그렇지만 일반 개발자가 디바이스 드라이버를 개발하는 것은 사실상 힘든 실정입니다. 또한 드라이버를 개발하여도 드라이버 성능 개선을 위해 시스템의 프로세서 우선 순위 등을 변경해야 하는 문제도 있습니다. 하지만 이 방법 또한 펌웨어 방식보다 성능 면에서는 떨어지게 됩니다.

IEC-Series는 Extension Port를 통하여 GPIO 기능을 지원하고 있어 신호의 입력 및 출력 처리를 할 수 있습니다. 출력과 달리 입력 신호의 처리는 Random하게 처리되고 있어 계속 CPU가 감시해야 하는 부담이 발생하는 처리로(Interrupt 처리가 아님) 많은 오차를 가지고 있습니다. 즉, 입력 신호의 처리는 CPU의 가용 상태에 따라 많은 차이를 나타낼 수 있으므로 충분한 검증을 하셔야 합니다. 여기서 가용 상태라 함은 CPU가 처리하는 코드의 양으로 in Process에서는 이벤트 코드의 양을 말하며, Out of Process에서는 각종 Process, 서비스, 디바이스 드라이버 등등의 프로세서가 CPU를 Time Slice 하여 처리하는 양을 말합니다. 아래 표에서 Event 코드에 따른 입력 신호 처리 결과를 확인해 보시기 바랍니다.

참고 CPU의 외부 입력 신호 Sampling Time에 따른 내부 입력값

CPU의 사용량에 따라 Sampling Time이 일정하지 않을 수 있습니다. 아래 그림에서 Sampling Time 간격에 따른 내부 입력값의 변화를 참고해 보시기 바랍니다.





4-3-1) Event 코드에 따른 입력 신호 처리 성능

Event 코드의 복잡도에 따라 입력 신호 처리 성능이 크게 차이가 납니다. 특히 사용자 인터페이스를 사용해 화면 UI를 갱신하는 작업을 하는 경우 차이는 더욱 커집니다. 아래 표는 IEC1000-Series의 Port-A에서 EvtPortADatasChange 이벤트를 사용하여 테스트한 결과입니다.

<table>
<tr><th></th><th>이벤트 코드가 단순한 경우</th><th>이벤트 코드가 복잡한 경우</th></tr>
<tr><td>사용자
인터페이스</td><td> 
</td><td> 
</td></tr>
<tr><td>테스트
순서</td><td>1. 외부 입력 신호가 LOW → HIGH가 될 때마다 이벤트 코드 내부에서 SmartLabel에 입력 Count 표시
2. 입력 가능한 최대 주파수를 측정</td><td>1. 외부 입력 신호가 LOW → HIGH 가 될 때마다 이벤트 코드 내부에서 SmartLabel에 입력 Count 표시, SmartDraw에 차트 그리기, SmartListBox에 입력 Count 추가
2. 입력 가능한 최대 주파수를 측정</td></tr>
<tr><td>테스트
코드</td><td><pre>
private int m_iCount = 0; // Count를 저장할 변수
private void smartGPIO1_EvtPortADatasChange
(object sender, EventArgs e)
{
  // 인자값으로 받은 핀 상태를 저장
  SmartX.PORTDataEvtArgs PortDatas = (SmartX.
    PORTDataEvtArgs)e;
  int iPortDatas = PortDatas.iPortDatas;
  SmartX.SmartGPIO.PORTPIN iBit;
  // 입력 신호가 변경된 핀을 체크
  iBit = smartGPIO1.PortDetection(SmartX.
  SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.
    TRIGGERMODE.HighActive, iPortDatas);
  if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) ==
       SmartX.SmartGPIO.PORTPIN.PIN0)
  {
    ++m_iCount; // Counting
    // 라벨에 데이터를 표시
    lblCount.Text = m_iCount.ToString();
  }
}
</pre>
2줄 코드가 추가됨</td><td><pre>
private int m_iCount = 0; // Count를 저장할 변수
private void smartGPIO1_EvtPortADatasChange
(object sender, EventArgs e)
{
  // 인자값으로 받은 핀 상태를 저장
  SmartX.PORTDataEvtArgs PortDatas = (SmartX.
    PORTDataEvtArgs)e;
  int iPortDatas = PortDatas.iPortDatas;
  SmartX.SmartGPIO.PORTPIN iBit;
  // 입력 신호가 변경된 핀을 체크
  iBit = smartGPIO1.PortDetection(SmartX.
    SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.
      TRIGGERMODE.HighActive, iPortDatas);
  if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) ==
       SmartX.SmartGPIO.PORTPIN.PIN0)
  {
    ++m_iCount; // Counting
    // 라벨에 데이터를 표시
    lblCount.Text = m_iCount.ToString();
    // 사용자 인터페이스 컴포넌트에 데이터를 추가
    smartDraw1.PutData(m_iCount);
    smartListBox1.AddItem(m_iCount.ToString());
  }
}
</pre></td></tr>
<tr><td>입력 가능한
최대 주파수</td><td>22H</td><td>5.5Hz</td></tr>
<tr><td>결론</td><td colspan="2">이벤트 코드가 복잡한 경우에 사용자 인터페이스 컴포넌트 작업을 하는 두 줄의 코드가 추가되어,
단순한 경우에 비해 입력 성능의 차이가 발생함</td></tr>
</table>

참고 SmartGPIO의 "4-2-1) EvtPortXDatasChange 이벤트 방식"을 참고하시기 바랍니다.

4-3-2) EvtPortXDatasChange vs EvtPortXDatasChangeCapture 비교

EvtPortXDatasChange 이벤트와 EvtPortXDatasChangeCapture 이벤트를 비교한 표입니다.

참고 본 측정 데이터는 입력 파형의 Duty Rate가 50일 때 기준이며, Duty Rate가 다르거나 사용자 인터페이스에 따라서 속도가 크게 차이나게 됩니다. 따라서 측정 결과에 비해 고속으로 데이터 입력을 권장하지 않습니다.
만약 측정 결과보다 고속의 신호를 계수(Counting)할 경우 SmartInputCounter 내용을 참고하시기 바랍니다.

이벤트	EvtPortXDatasChange		EvtPortXDatasChangeCapture	
사용환경	외부 입력 속도가 저속일 때 사용		외부 입력 속도가 고속일 때 사용	
용도	포트 상태에 따른 즉시 처리가 필요할 경우		포트 상태 변화를 카운트해야 하는 경우	
처리 시점	즉시 처리		일괄 처리	
실시간성	실시간성		비실시간성	
입력 속도	1. 외부 입력 신호를 받으면 이벤트가 발생하여 입력된 신호를 즉시처리 2. 연속적인 신호나 처리 코드의 복잡도가 높은 경우 입력 신호의 누락이 발생될 수 있음 3. 응답성능은 EvtPortXDatasChangeCapture에 비해 좋음		1. 외부 입력 신호를 큐에 저장 후 이벤트 프로세스가 이벤트를 발생하여 입력된 신호를 일괄로 처리 2. 일정 시간 빠른 속도의 연속적인 입력 신호 처리에 적합하여 입력 신호의 누락이 발생되지 않음 3. 응답성능은 EvtPortXDatasChange에 비해 떨어짐	
관련 메소드	PortXWatchStart() PortXWatchStop()		PortXWatchStart() PortXWatchStop() SetPortXWatchPriority() SetPortXWatchIdleInterval() GetPortXQueueWaitingCount()	
테스트 코드	"4-3-1) Event 코드에 따른 입력 신호 처리 성능" 내용을 참고하시기 바랍니다.		"4-2-2) EvtPortXDatasChangeCapture 이벤트 방식" 내용을 참고하시기 바랍니다.	
최대 입력 주파수 (Port A기준)	IEC1000-Series	22Hz → 45.45ms	IEC1000-Series	30kHz → 33.33us
	IEC667-Series	13.5Hz → 74.07ms	IEC667-Series	5.5kHz → 181.81us
	IEC266-Series	17.6Hz → 56.8ms	IEC266-Series	기능 지원 안 함

5) 프로그래밍 적용 가이드

CASE-1 출력 기능 사용하기

SmartGPIO를 출력 기능으로 사용하기 위해서는 먼저 출력으로 사용할 Pin의 방향을 PORTXDIR 속성을 사용해 출력(Output)으로 설정해야 합니다. 방향 설정이 완료되면 PORTXDATA 속성으로 Pin의 상태를 HIGH 또는 LOW로 설정할 수 있습니다.

※ 자세한 내용은 "PORTXDIR, PORTXDIRS, PORTXDATA, PORTXDATAS 속성"을 참고하시기 바랍니다.

```
smartGPIO1.PORTADIR0 = SmartX.SmartGPIO.PORTDIRS.OUTPUT;  // Port-A0 Pin을 출력으로 설정
smartGPIO1.PORTBDIRS = 0xFF;                              // Port-B의 모든 Pin을 출력으로 설정
smartGPIO1.PORTADATA0 = true;                             // Port-A0 Pin을 HIGH로 설정
smartGPIO1.PORTBDATAS = 0xFF;                             // Port-B의 모든 Pin을 HIGH로 설정
System.Threading.Thread.Sleep(1000);                      // 1초(1000ms) 대기
smartGPIO1.PORTADATA0 = false;                            // Port-A0 Pin을 LOW로 설정
smartGPIO1.PORTBDATAS = 0x00;                             // Port-B의 모든 Pin을 LOW로 설정
```

CASE-2 폴링 방식으로 입력 신호 감지하기

폴링 방식은 반복문을 사용해 신호를 입력받은 방식으로, 이벤트 방식보다는 비교적 정확하여 중요한 데이터를 수신해야 하는 경우에 사용합니다. 입력으로 사용할 Pin의 방향을 PORTXDIR 속성을 사용해 입력(Input)으로 설정합니다. Pin 방향 설정이 완료되면 PortDetection_Initialize() 메소드를 호출해 입력으로 사용할 Port를 초기화합니다. 모든 설정이 완료되면 반복문에서 PortDetection() 메소드를 반복하여 호출해 Pin의 상태를 읽습니다.

※ 자세한 내용은 "폴링(Polling) 방식", "PORTXDIR, PORTXDIRS, PORTXDATA, PORTXDATAS 속성", "PortDetection() 메소드"를 참고하시기 바랍니다.

```
private void ReadData_Polling()
{
  smartGPIO1.PORTADIR0 = SmartX.SmartGPIO.PORTDIRS.INPUT;  // PortA0 Pin을 입력으로 설정
  smartGPIO1.PORTADIRS = 0x00;                             // PortA의 모든 Pin을 입력으로 설정
  // Polling 방식으로 사용하시는 경우 반드시 해당 Port를 초기화하시기 바랍니다.
  smartGPIO1.PortDetection_Initialize(SmartX.SmartGPIO.PORTID.PORTA);
  SmartX.SmartGPIO.PORTPIN iBit;

  // 반복하며 Pin의 상태를 읽습니다.
  while (true)
  {
    // 현재 Pin의 상태를 읽은 후 입력 신호가 변경된 핀을 체크 (LOW->HIGH)
    iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.TRIGGERMODE.
           HighActive)
    if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) == SmartX.SmartGPIO.PORTPIN.PIN0)
    {
      ++iPortA[0];  // Counting
      break;
    }
    // …중략…
  }
}
```

CASE-3 이벤트 방식으로 입력 신호 감지하기

이벤트 방식은 Port에 입력되는 신호의 변화에 따라 이벤트가 발생되는 방식입니다. 따라서 입력 신호 변화를 대기할 필요 없이 변화 발생 즉시 응답 처리할 수 있습니다.

※ 자세한 내용은 "이벤트(Event) 방식, EvtPortXDatasChange 이벤트 방식"을 참고하시기 바랍니다.

[STEP-1] 입력 방향 설정 및 감지 시작하기

먼저 입력 신호를 감지할 Pin의 방향을 PORTXDIR 또는 PORTXDIRS 속성을 사용해 입력으로 설정합니다. 설정이 완료되면 PortXWatchStart() 메소드를 호출해 입력 신호 감지를 시작합니다. 입력 신호가 감지되면 EvtPortXDatasChange 이벤트가 발생합니다.

※ 자세한 내용은 "PORTXDIR, PORTXDIRS 속성", "PortXWatchStart() 메소드"를 참고하시기 바랍니다.

```
smartGPIO1.PORTADIRS = 0x00;      // PORTA의 모든 Pin을 입력으로 설정
smartGPIO1.PortAWatchStart();     // PORTA의 입력 감지 시작
```

[STEP-2] 이벤트에서 입력 신호 감지하기

입력 신호의 변화가 발생하면 EvtPortXDatasChange 이벤트가 발생합니다. 이벤트의 인자값을 확인해 현재 Pin의 상태를 얻고, PortDetection() 메소드를 사용해 상태가 변경된 Pin을 TRIGGERMODE를 설정해 입력 신호 감지 시점에 따라 확인할 수 있습니다. 이후 조건문을 사용해 변경된 특정 Pin을 간추릴 수 있습니다.

※ 자세한 내용은 "PortDetection() 메소드", "EvtPortXDatasChange 이벤트"를 참고하시기 바랍니다.

```
private void smartGPIO1_EvtPortADatasChange(object sender, EventArgs e)
{
  // 인자값에서 현재 Pin의 상태를 얻습니다.
  SmartX.PORTDataEvtArgs PortDatas = (SmartX.PORTDataEvtArgs)e;
  int iPortDatas = PortDatas.iPortDatas;
  // 입력 신호가 변경된 핀을 체크 (HIGH -> LOW, LOW -> HIGH)
  SmartX.SmartGPIO.PORTPIN iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA,
                                  SmartX.SmartGPIO.TRIGGERMODE.LevelChange, iPortDatas);
  if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) == SmartX.SmartGPIO.PORTPIN.PIN0)
  {
     ++iPortA[0];  // Counting
  }
  // …중략…
}
```

[STEP-3] 입력 감지 종료하기

PortXWatchStop() 메소드를 호출해 입력 신호 감지를 종료할 수 있습니다.

※ 자세한 내용은 “PortXWatchStop() 메소드”를 참고하시기 바랍니다.

```
smartGPIO1.PortAWatchStop();  // PORTA의 입력 감지 종료
```

CASE-4 이벤트 방식으로 입력 신호 카운팅(캡처)하기

SmartGPIO는 고속의 입력 신호를 카운팅 하기 위한 기능을 제공하고 있습니다. 이 방식은 CASE-3의 방식에서 입력 신호의 누락이 발생하는 경우를 개선할 수 있습니다.

※ 자세한 내용은 “EvtPortXDatasChangeCapture 이벤트 방식”, “EvtPortXDatasChange vs EvtPortXDatasChangeCapture 비교”를 참고하시기 바랍니다.

[STEP-1] 입력 방향 설정 및 캡처 시작하기

먼저 입력 신호를 감지할 Pin의 방향을 PORTXDIR 또는 PORTXDIRS 속성을 사용해 입력으로 설정합니다. 설정이 완료되면 PortXWatchStart() 메소드를 호출해 입력 신호 캡처를 시작합니다. 입력 신호가 감지되면 EvtPortXDatasChangeCapture 이벤트가 발생합니다.

※ 자세한 내용은 “PORTXDIR, PORTXDIRS 속성”, “PortXWatchStart() 메소드”를 참고하시기 바랍니다.

```
smartGPIO1.PORTADIRS = 0x00;   // PORTA의 모든 Pin을 입력으로 설정
smartGPIO1.PortAWatchStart();  // PORTA의 입력 캡처 시작
```

[STEP-2] 이벤트에서 입력 신호 감지하기

입력 신호의 변화가 발생하면 EvtPortXDatasChangeCapture 이벤트가 발생합니다. 이벤트가 발생하면 GetPortXQueueWaitingCount() 메소드를 호출해 Queue에 저장된 데이터의 개수를 확인하고 개수가 증가한다면, SetPortXWatchIdleInterval() 메소드의 인자값을 크게 설정해 Port Scan Process 속도를 증가시켜 입력 신호를 놓치지 않도록 합니다. 반대로 줄고 있다면 인자값을 작게 설정해 Port Scan Process 속도를 감소시켜 EvtPortXDatasChangeCapture 이벤트가 빠르게 발생할 수 있도록 합니다. 이후 이벤트의 인자값을 확인해 현재 Pin의 상태를 얻고, PortDetection() 메소드를 사용해 상태가 변경된 Pin의 TRIGGERMODE를 설정해 입력 신호 감지 시점에 따라 확인할 수 있습니다. 이후 조건문을 사용해 변경된 특정 Pin을 간추릴 수 있습니다.

※ 자세한 내용은 “GetPortXQueueWaitingCount(), SetPortXWatchIdleInterval(), PortDetection() 메소드”, “EvtPortXDatasChangeCapture 이벤트”를 참고하시기 바랍니다.

```
private int[] iPortA = new int[8];          // 입력 신호를 카운팅할 배열
private uint m_uiQueueCount = 0;            // ChangeCapture 이벤트가 실행된 다음의 큐의 개수
private bool m_bIsQueueIncrease = false;    // PortData Queue의 데이터 증가 여부를 확인할 플래그
```

```
private void smartGPIO1_EvtPortADatasChangeCapture(int iPortDatas)
{
   // PortData Queue의 현재 저장된 데이터의 개수와 직전의 개수를 체크
   if (smartGPIO1.GetPortAQueueWaitingCount() > m_uiQueueCount)
   {
     // 플래그를 설정하여 한 번만 설정하도록 한다.
     if (m_bIsQueueIncrease == false)
     {
       // 현재 저장된 데이터의 개수가 더 크다면, 입력 신호가 감지되고 있는 것으로,
       m_bIsQueueIncrease = true;
       PortA_ScanSpeed_High();  // PortA_ScanSpeed_High() 메소드를 호출한다.
     }
   }
   else
   {
     // 현재 저장된 데이터의 개수가 더 작다면, 입력 신호가 감지되고 있지 않은 것으로,
     m_bIsQueueIncrease = false;
     PortA_ScanSpeed_Low();  // PortA_ScanSpeed_Low() 메소드를 호출한다.
   }

  // 신호 감지 시점을 선택하여 수신받는다. (LevelChange, NegativeEdge, PositiveEdge 중 선택)
  SmartX.SmartGPIO.PORTPIN iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA,
                                  SmartX.SmartGPIO.TRIGGERMODE.LevelChange, iPortDatas));

  if ((iBit & SmartX.SmartGPIO.PORTPIN.PIN0) == SmartX.SmartGPIO.PORTPIN.PIN0)
  {
     ++iPortA[0]; // PortData Counting
  }
  // .. 중략 ..

  // 직전의 저장된 데이터 개수를 현재 큐에 저장된 데이터 개수로 설정
  m_uiQueueCount = smartGPIO1.GetPortAQueueWaitingCount();
}

// Port-A의 WatchIdleInterval을 증가시키는 메소드
// WatchIdleInterval 값을 높게 변경하여 Port Scan Process 속도를 빠르게 처리한다.
private void PortA_ScanSpeed_High()
{
  smartGPIO1.SetPortAWatchIdleInterval(100000);  // 적정값 1000~100000
}

// Port-A의 WatchIdleInterval을 감소시키는 메소드
// WatchIdleInterval 값을 낮게 변경하여 Port Scan Process 속도를 느리게 처리한다.
private void PortA_ScanSpeed_Low()
{
  smartGPIO1.SetPortAWatchIdleInterval(10);  // 적정값 10~100
}
```

[STEP-3] 입력 신호 캡처 종료하기

PortXWatchStop() 메소드를 호출해 입력 신호 캡처를 종료할 수 있습니다.
※ 자세한 내용은 “PortXWatchStop() 메소드”를 참고하시기 바랍니다.

```
smartGPIO1.PortAWatchStop();  // PORTA의 입력 감지 종료
```

6) SmartGPIO 인터페이스 설명

SmartGPIO Component Interface		
속성		
IsPortAWatchStart : bool	IsPortBWatchStart : bool	IsPortCWatchStart : bool
IsPortDWatchStart : bool	IsPortEWatchStart : bool	IsPortFWatchStart : bool
IsPortGWatchStart : bool	IsPortHWatchStart : bool	PORTADIR0 ~ PORTADIR7 : bool
PORTBDIR0 ~ PORTBDIR7 : bool	PORTCDIR0 ~ PORTCDIR7 : bool	PORTDDIR0 ~ PORTDDIR7 : bool
PORTEDIR0 ~ PORTEDIR7 : bool	PORTFDIR0 ~ PORTFDIR7 : bool	PORTGDIR0 ~ PORTGDIR7 : bool
PORTHDIR0 ~ PORTHDIR3 : bool	PORTADIRS : int	PORTBDIRS : int
PORTCDIRS : int	PORTDDIRS : int	PORTEDIRS : int
PORTFDIRS : int	PORTGDIRS : int	PORTHDIRS : int
PORTADATA0 ~ PORTADATA7 : bool	PORTBDATA0 ~ PORTBDATA7 : bool	PORTCDATA0 ~ PORTCDATA7 : bool
PORTDDATA0 ~ PORTDDATA7 : bool	PORTEDATA0 ~ PORTEDATA7 : bool	PORTFDATA0 ~ PORTFDATA7 : bool
PORTGDATA0 ~ PORTGDATA7 : bool	PORTHDATA0 ~ PORTHDATA3 : bool	PORTADATAS : int
PORTBDATAS : int	PORTCDATAS : int	PORTDDATAS : int
PORTEDATAS : int		
메소드		
GetPortAQueueWaitingCount() : int	GetPortBQueueWaitingCount() : int	GetPortCQueueWaitingCount() : int
GetPortDQueueWaitingCount() : int	GetPortEQueueWaitingCount() : int	GetPortFQueueWaitingCount() : int
GetPortGQueueWaitingCount() : int	GetPortHQueueWaitingCount() : int	PortDetection(SmartGPIO.PORTID ePortID, SmartGPIO.TRIGGERMODE eTriggerMode) : SmartGPIO.PORTPIN (+3개 오버로드)
PortDetection_Initialize(SmartGPIO.PORTID ePortID) : void	SetPortAWatchIdleInterval(int iInterval) : void	SetPortBWatchIdleInterval(int iInterval) : void
SetPortCWatchIdleInterval(int iInterval) : void	SetPortDWatchIdleInterval(int iInterval) : void	SetPortEWatchIdleInterval(int iInterval) : void
SetPortFWatchIdleInterval(int iInterval) : void	SetPortGWatchIdleInterval(int iInterval) : void	SetPortHWatchIdleInterval(int iInterval) : void
SetPortAWatchPriority(int iPriority) : void	SetPortBWatchPriority(int iPriority) : void	SetPortCWatchPriority(int iPriority) : void
SetPortDWatchPriority(int iPriority) : void	SetPortEWatchPriority(int iPriority) : void	SetPortFWatchPriority(int iPriority) : void
SetPortGWatchPriority(int iPriority) : void	SetPortHWatchPriority(int iPriority) : void	PortAWatchStart() : void
PortBWatchStart() : void	PortCWatchStart() : void	PortDWatchStart() : void
PortEWatchStart() : void	PortFWatchStart() : void	PortGWatchStart() : void
PortHWatchStart() : void	PortAWatchStop() : void	PortBWatchStop() : void
PortCWatchStop() :void	PortDWatchStop() : void	PortEWatchStop() : void
PortFWatchStop() : void	PortGWatchStop() : void	PortHWatchStop() : void
이벤트		
EvtPortADatasChange : EventHandler	EvtPortBDatasChange : EventHandler	EvtPortCDatasChange : EventHandler
EvtPortDDatasChange : EventHandler	EvtPortEDatasChange : EventHandler	EvtPortFDatasChange : EventHandler
EvtPortGDatasChange : EventHandler	EvtPortHDatasChange : EventHandler	
EvtPortADatasChangeCapture: SmartGPIO.PortDataEventHandler	EvtPortBDatasChangeCapture: SmartGPIO.PortDataEventHandler	EvtPortCDatasChangeCapture: SmartGPIO.PortDataEventHandler
EvtPortDDatasChangeCapture: SmartGPIO.PortDataEventHandler	EvtPortEDatasChangeCapture: SmartGPIO.PortDataEventHandler	EvtPortFDatasChangeCapture: SmartGPIO.PortDataEventHandler
EvtPortGDatasChangeCapture: SmartGPIO.PortDataEventHandler	EvtPortHDatasChangeCapture: SmartGPIO.PortDataEventHandler	

프로퍼티(속성) IsPortAWatchStart, IsPortBWatchStart, IsPortCWatchStart, IsPortDWatchStart, IsPortEWatchStart, IsPortFWatchStart, IsPortGWatchStart, IsPortHWatchStart

해당 Port의 입력 감지 시작(WatchStart) 여부를 확인합니다.

- bool : 입력 감지 시작 여부
  - True : 시작
  - False : 중지

C# 사용법

```
// 만약 Port-A의 입력 감지가 시작 상태이면 Port-A의 입력 감지를 중지
if(smartGPIO1.IsPortAWatchStart == true)
{
  smartGPIO1.PortAWatchStop();
}
```

VB 사용법

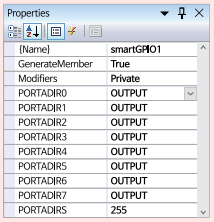
```
If smartGPIO1.IsPortAWatchStart = True Then
   smartGPIO1.PortAWatchStop()
End If
```

프로퍼티(속성) PORTADIR0 ~ 7, PORTBDIR0 ~ 7, PORTCDIR0 ~ 7, PORTDDIR0 ~ 7, PORTEDIR0 ~ 7, PORTFDIR0 ~ 7, PORTGDIR0 ~ 7, PORTHDIR0 ~ 3

Port-A, B, C, D, E, F, G, H의 방향(입력/출력)을 Pin 단위로(1Bit) 설정합니다.

주의 프로그램 로딩 시 초기 Port(Pin)방향 설정 주의사항

C#, VB.NET에서 내부 소스 코드의 실행 순서는 InitializeComponent() → Form1_Load()의 순서로 실행됩니다. 큰 용량의 프로그램인 경우 Form1_Load 이벤트에서 초기값을 설정하게 되면 Form_Load 이벤트가 실행되기 전까지 사용자가 설정한 Port(Pin)별 방향(입/출력)이 설정되지 않아 포트 동작이 지연될 수 있습니다.
따라서 포트의 지연을 방지하기 위해 Port(Pin)별 방향(입/출력) 설정은 반드시 디자인 속성창에서 설정하시기 바랍니다.



- SmartGPIO.PORTDIRS.OUTPUT : 특정 Port의 Pin 방향을 출력으로 설정합니다.
- SmartGPIO.PORTDIRS.INPUT : 특정 Port의 Pin 방향을 입력으로 설정합니다.

C# 사용법

```
smartGPIO1.PORTADIR0 = SmartGPIO.PORTDIRS.INPUT;  // Port-A의 0번 Pin을 입력으로 사용함
```

VB 사용법

```
smartGPIO1.PORTADIR0 = SmartGPIO.PORTDIRS.INPUT
```

프로퍼티(속성) PORTADIRS, PORTBDIRS, PORTCDIRS, PORTDDIRS, PORTEDIRS, PORTFDIRS, PORTGDIRS, PORTHDIRS

Port-A, B, C, D, E, F ,G ,H의 방향(입력/출력)을 Port 단위로(8Bit) 설정합니다.
각 Pin의 Bit 값이 1이면 출력(Output) / Bit 값이 0이면 입력(Input)입니다.

[표] Port-X의 각 Pin 방향 설정 방법

Pin 번호	0	1	2	3	4	5	6	7
Bit	1	1	0	0	1	1	0	0
방향	출력	출력	입력	입력	출력	출력	입력	입력
Hex	0x33							
Int	51							

C# 사용법

```
smartGPIO1.PORTADIRS = 0xFF;  // Port-A의 모든 Pin을 출력으로 설정한다.
```

VB 사용법

```
smartGPIO1.PORTADATA0 = &HFF
```

프로퍼티(속성) PORTADATA0 ~ 7, PORTBDATA0 ~ 7, PORTCDATA0 ~ 7, PORTDDATA0 ~ 7, PORTEDATA0 ~ 7, PORTFDATA0 ~ 7, PORTGDATA0 ~ 7, PORTHDATA0 ~ 3

Port-A, B, C, D, E, F, G, H의 Pin 단위로(1Bit) 데이터를 읽거나 씁니다.

[표] 속성 사용 방식에 따른 결과

속성값	속성값을 읽는 경우	속성값을 쓰는 경우
True	해당 Pin의 상태는 HIGH입니다	Pin의 상태를 HIGH로 설정합니다.
False	해당 Pin의 상태는 LOW입니다.	Pin의 상태를 LOW로 설정합니다.

C# 사용법

```
smartGPIO1.PORTADATA0 = true;            // Port-A의 0 Pin(Bit)을 HIGH로 합니다.
bool bPortData = smartGPIO1.PORTADATA0;  // Port-A의 0 Pin(Bit)의 상태를 읽음
```

VB 사용법

```
smartGPIO1.PORTADATA0 = True
Dim bPortData As Boolean = smartGPIO1.PORTADATA0
```

프로퍼티(속성) PORTADATAS, PORTBDATAS, PORTCDATAS, PORTDDATAS, PORTEDATAS, PORTFDATAS, PORTGDATAS, PORTHDATAS

Port-A, B, C, D, E, F, G, H를 Port 단위로(8Bit) 데이터를 읽거나 씁니다.

[표1] 속성 사용 방식에 따른 결과

속성값	속성값을 읽는 경우	속성값을 쓰는 경우
Hex : 0x33, Int : 55	Port-X의 각 Pin 상태는 [표2]와 같습니다.	Port-X의 각 Pin 상태를 [표2]로 설정합니다.

[표2] 속성값에 따른 Port-X의 각 Pin별 상태

Pin 번호	0	1	2	3	4	5	6	7
Bit	1	1	0	0	1	1	0	0
Status	HIGH	HIGH	LOW	LOW	HIGH	HIGH	LOW	LOW

C# 사용법

```
smartGPIO1.PORTADATAS = 0xFF;          // Port-A 출력(모든 Pin을 High)
int iPortData = smartGPIO1.PORTADATAS; // Port-A 입력(모든 Pin의 상태을 읽음)
```

VB 사용법

```
smartGPIO1.PORTADATAS = &HFF
Dim iPortData As Integer = smartGPIO1.PORTADATAS
```

메소드(함수) PortAWatchStart, PortBWatchStart, PortCWatchStart, PortDWatchStart, PortEWatchStart, PortFWatchStart, PortGWatchStart, PortHWatchStart

Port가 입력 모드일 경우 Port의 상태 변경 감지를 Event 방식으로 시작합니다.
Port의 상태가 변경되면 EvtPortXDatasChange/EvtPortXDatasChangeCapture 이벤트가 발생합니다.

주의 PortXWatchStart() 메소드 사용 시 주의사항

1. PortXWatchStart() 메소드를 호출한 경우 프로그램 종료 시 반드시 PortXWatchStop() 메소드를 호출해야 합니다.
2. Polling 방식으로 사용 시 PortXWatchStart() 메소드를 호출하지 마시기 바랍니다.

◉ void PortAWatchStart() ~ PortHWatchStart()

C# 사용법

```
private void Form1_Load(object sender, EventArgs e)
{
  // Port-A의 입력 상태가 변경되는지 감지를 시작함
  smartGPIO1.PortAWatchStart();
}
private void Form1_Closing(object sender, CancelEventArgs e)
{
  // Port-A의 입력 상태가 변경되는지 감지를 중단함 (프로그램 종료 시 반드시 처리해야 함)
  smartGPIO1.PortAWatchStop();
}
```

VB 사용법

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
  smartGPIO1.PortAWatchStart()
End Sub
Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As System.ComponentModel.
CancelEventArgs) Handles MyBase.Closing
  smartGPIO1.PortAWatchStop()
End Sub
```

메소드(함수) PortAWatchStop, PortBWatchStop, PortCWatchStop, PortDWatchStop, PortEWatchStop, PortFWatchStop, PortGWatchStop, PortHWatchStop

Port 상태 변경 감지를 종료합니다.

◉ void PortAWatchStop() ~ PortHWatchStop()

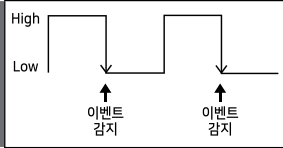
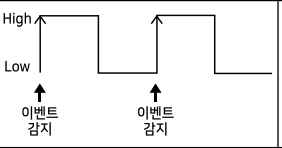
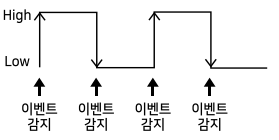
사용법

참고 PortXWatchStart() 메소드를 참고하시기 바랍니다.

메소드(함수) PortDetection

Port-A, B, C, D, E, F, G, H로 입력되는 신호의 변화 시점에 따라 신호를 감지합니다. 입력 신호 감지 시점에 따라 TRIGGERMODE를 LowActive, HighActive, LevelChange로 설정할 수 있으며, 특정 Pin 또는 Or 연산을 통한 복수 개 Pin의 동시 입력을 감지할 수 있습니다. 아래 표를 참고하여 상황에 맞게 사용하시기 바랍니다.

[표] TRIGGERMODE에 따른 입력 신호 감지 시점

TRIGGERMODE	LowActive	HighActive	LevelChange
입력 파형			
입력신호 감지 시점	High → Low	Low → High	High → Low / Low → High

참고 "4) 입력 기능 → PortDetection() 메소드 결과값 설명"을 참고하시기 바랍니다.

◉ SmartGPIO.PORTPIN PortDetection(SmartGPIO.PORTID ePortID, SmartGPIO.TRIGGERMODE eTriggerMode)

◉ SmartGPIO.PORTPIN PortDetection(SmartGPIO.PORTID ePortID, SmartGPIO.TRIGGERMODE eTriggerMode, int iNowPortData)

◉ SmartGPIO.PORTPIN PortDetection(SmartGPIO.PORTID ePortID, SmartGPIO.TRIGGERMODE eTriggerMode, SmartGPIO.PORTPIN eTriggerPins)

◉ SmartGPIO.PORTPIN PortDetection(SmartGPIO.PORTID ePortID, SmartGPIO.TRIGGERMODE eTriggerMode, int iNowPortData, SmartGPIO.PORTPIN eTriggerPins)

[인자]

- SmartGPIO.PORTID ePortID : 변화를 감지할 Port를 설정
- SmartGPIO.TRIGGERMODE eTriggerMode : 입력 신호의 변화를 감지할 시점을 설정
- SmartGPIO.PORTPIN eTriggerPins : 변화를 감지할 Pin을 설정. 즉, 특정 Pin을 트리거로 설정합니다. Or 연산을 통한 복수개 Pin의 동시 입력 감지 가능
- int iNowPortData : 현재 Port에서 감지한 모든 Pin의 상태. 생략할 경우 메소드 호출 시점의 Pin 상태를 기준으로 입력 신호 변화를 감지합니다.

[리턴값]

- SmartGPIO.PORTPIN : 신호가 감지된 Pin의 정보. 복수개의 Pin이 감지된 경우, 감지된 Pin 상태값이 모두 더한 값을 리턴합니다.

[표] SmartGPIO.PORTPIN 열거값에 따른 Bit 값과 리턴값 예시

열거값	ZERO	PIN0	PIN1	PIN2	PIN3	PIN4	PIN5	PIN6	PIN7
Bit값	0	1	2	4	8	16	32	64	128
예시	1. PIN0 입력 감지 시 리턴값 : PIN0 2. PIN0, PIN7 입력 감지 시 리턴값 : 129(1 + 128)								

사용법

참고 특정 Pin 또는 복수 Pin 동시 입력 시 신호 변화를 감지하는 방법

PortDetection() 메소드는 임의의 Pin에 입력되는 신호 변화를 감지합니다. 이때 특정 Pin 또는 복수 Pin의 신호 변화를 감지(트리거)하고 싶은 경우, 오버로드 메소드의 인자값 "SmartGPIO.PORTPIN eTriggerPins"를 사용하면 구현이 가능합니다. 아래 표에서 사용법을 확인해 프로젝트에 적용해 보시기 바랍니다.

※ 사용법은 Event 방식을 기준으로 설명하였으며, Polling 방식도 이벤트 방식과 큰 차이는 없습니다.

메소드(함수) PortDetection_Initialize

PortDetection() 함수와 관련하여 Port의 이전 상태를 초기화 합니다.

주의 Polling 방식으로 입력 신호를 감지하는 경우 초기에 반드시 호출해야 합니다.

◉ void PortDetection_Initialize(SmartGPIO.PORTID ePortID)

[인자]

- SmartGPIO.PORTID ePortID : 초기화할 Port를 설정합니다.

열거값	설명	열거값	설명	열거값	설명	열거값	설명
PORTA	Port-A	PORTC	Port-C	PORTE	Port-E	PORTG	Port-G
PORTB	Port-B	PORTD	Port-D	PORTF	Port-F	PORTH	Port-H

C# 사용법

```
// Port-A의 이전 상태를 초기화
smartGPIO1.PortDetection_Initialize(SmartGPIO.PORTID.PORTA);
```

VB 사용법

```
smartGPIO1.PortDetection_Initialize(SmartGPIO.PORTID.PORTA)
```

[표] 입력 신호 변화 감지 메소드 사용법

[C#] 사용법

```
private void smartGPIO1_EvtPortADatasChange(object sender, EventArgs e)
{
  int iPortDatas                          // 현재 Port 상태를 저장할 변수
  SmartX.SmartGPIO.PORTPIN iBit;          // 신호 변화가 감지된 핀 정보를 저장할 변수
  SmartX.SmartGPIO.PORTPIN iTriggerPins;  // 트리거로 설정할 핀의 정보를 저장할 변수
  // Event 방식일 경우 인자값을 이용하여 Port-A의 상태를 변수에 저장합니다.
  iPortDatas = ((SmartX.PORTDataEvtArgs)e).iPortDatas;
  // Polling 방식일 경우 속성값을 이용하여 Port-A의 상태를 변수에 저장합니다.
  // iPortDatas = smartGPIO1.PORTADATAS;

  // 1.Port-A의 복수 Pin(Pin0, Pin1)의 동시 입력 신호 변화(High -> Low)를 감지하는 경우
  // 신호 변화를 감지할 특정 핀을 설정합니다.(트리거 설정할 핀) : Pin0, Pin1
  iTriggerPins = SmartX.SmartGPIO.PORTPIN.PIN0 | SmartX.SmartGPIO.PORTPIN.PIN1;
  iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.TRIGGERMODE.
         LowActive, iPortDatas, iTriggerPins);
  if (iBit == iTriggerPins)
  {
    // 동작 처리
  }
  // …중략…

  //-------------------------------------------------------------------------
  // 2.Port-A의 특정 Pin(Pin0)에 입력되는 신호 변화(Low -> High)를 감지하는 경우
  // 신호 변화를 감지할 특정 핀을 설정합니다.(트리거 설정할 핀) : Pin0
  iTriggerPins = SmartX.SmartGPIO.PORTPIN.PIN0;
  iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.TRIGGERMODE.
         HighActive, iPortDatas, iTriggerPins);
  if (iBit == iTriggerPins)
  {
    // 동작 처리
  }
  //-------------------------------------------------------------------------
  // ..Port-A의 임의의 Pin에 입력되는 신호 변화(Low -> High, High -> Low)를 감지하는 경우
  iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA, SmartX.SmartGPIO.TRIGGERMODE.
         LevelChange, iPortDatas);

  iTriggerPins = SmartX.SmartGPIO.PORTPIN.PIN0;  // PIN0의 입력을 체크
  if ((iBit & iTriggerPins) == iTriggerPins)
  {
    // 동작 처리
  }
}
```

메소드(함수)

```
SetPortAWatchIdleInterval, SetPortBWatchIdleInterval,
SetPortCWatchIdleInterval, SetPortDWatchIdleInterval,
SetPortEWatchIdleInterval, SetPortFWatchIdleInterval,
SetPortGWatchIdleInterval, SetPortHWatchIdleInterval
```

Port Change Scan Process가 CPU에 할당되는 시간을 설정합니다.

참고 “EvtPortXDatasChangeCapture 이벤트 방식”을 참고하시기 바랍니다.

주의 본 기능은 EvtPortXDatasChangeCapture 이벤트에서만 동작합니다.

◉ `void SetPortAWatchIdleInterval(int iInterval) ~ SetPortHWatchIdleInterval(int iInterval)`

[인자]

• int iInterval : Scan Porcess가 CPU에 할당되는 시간

C# 사용법

```
// 입력 큐의 데이터가 증가(입력 신호 수신 중)
smartGPIO1.SetPortAWatchIdleInterval(100000);
// 입력 큐의 데이터가 감소(입력 신호 중지)
smartGPIO1.SetPortAWatchIdleInterval(10);
```

VB 사용법

```
smartGPIO1.SetPortAWatchIdleInterval(100000)
smartGPIO1.SetPortAWatchIdleInterval(10)
```

메소드(함수) SetPortAWatchPriority, SetPortBWatchPriority, SetPortCWatchPriority, SetPortDWatchPriority, SetPortEWatchPriority, SetPortFWatchPriority, SetPortGWatchPriority, SetPortHWatchPriority

※ 사용을 권장하지 않습니다.※

Port Change Scan Process의 우선 순위를 설정합니다.

참고 Windows CE 운영체제 상에서의 프로세서 우선 순위의 정의

모든 프로세서(스레드)는 0~255의 우선 순위를 설정하여 사용할 수 있으며 정의된 우선 순위는 아래와 같습니다.

[표1] 일반적으로 정의된 우선 순위

우선 순위	설명
0 ~ 96	실시간 성능을 요구하는 장치 드라이버보다 높은 우선 순위를 요하는 경우 사용
97 ~152	기본 장치 드라이버에서 사용
153 ~ 247	실시간 성능이 필요 없는 장치 드라이버에서 사용
248 ~ 255	비 실시간 우선 순위로 장치 응용 프로그램에서 사용

[표2] 장치 응용 프로그램 내에서 사용되는 우선 순위

우선 순위	설명
248	장치 응용 프로그램 TIME_CRITICAL
249	장치 응용 프로그램 HIGHEST
250	장치 응용 프로그램 ABOVE_NORMAL
251	장치 응용 프로그램 NORMAL : 기본적으로 사용되는 응용 프로그램(스레드)의 우선 순위
252	장치 응용 프로그램 BELOW_NORMAL
253	장치 응용 프로그램 LOWEST
254	장치 응용 프로그램 ABOVE_IDLE
255	장치 응용 프로그램 IDLE

주의 본 기능은 EvtPortXDatasChangeCapture 이벤트에서만 동작합니다.

◉ void SetPortAWatchPriority(int iPriority) ~ SetPortHWatchPriority(int iPriority)

[인자]

• int iPriority : 입력 우선 순위 조절

C# 사용법

```
// 입력 우선 순위 높게 설정(실시간 성능이 필요한 경우)
smartGPIO1.SetPortAWatchPriority(1);
// 입력 우선 순위 낮게 설정(실시간 성능이 필요없는 경우)
smartGPIO1.SetPortAWatchPriority(251);
```

VB 사용법

```
smartGPIO1.SetPortAWatchPriority(1)
smartGPIO1.SetPortAWatchPriority(251)
```

메소드(함수) `GetPortAQueueWaitingCount, GetPortBQueueWaitingCount, GetPortCQueueWaitingCount, GetPortDQueueWaitingCount, GetPortEQueueWaitingCount, GetPortFQueueWaitingCount, GetPortGQueueWaitingCount, GetPortHQueueWaitingCount`

각 Port별 PortDataQueue에 저장된 데이터의 개수(QueueWaitingCount)를 가져옵니다.

참고 "EvtPortXDatasChangeCapture 이벤트 방식"을 참고하시기 바랍니다.

주의 본 기능은 EvtPortXDatasChangeCapture 이벤트에서만 동작합니다.

◉ `uint GetPortAQueueWaitingCount() ~ GetPortHQueueWaitingCount()`

[리턴값]

• `uint` : PortDataQueue에 저장된 데이터의 개수

C# 사용법

```
// 큐에 저장된 데이터의 개수를 확인할 변수
private uint m_uiQueueCnt;
private SetPortASpeed()
{
  // 만약 Port A 입력 신호를 저장하는 큐가 증가 중이면
  if (smartGPIO1.GetPortAQueueWaitingCount() > m_uiQueueCnt)
  {
    smartGPIO1.SetPortAWatchIdleInterval(100000);  // Port-A의 입력 큐의 호출 속도를 느리게 설정
  }
  else
  {
    smartGPIO1.SetPortAWatchIdleInterval(10);  // Port-A의 입력 큐의 호출 속도를 빠르게 설정
  }
  m_uiQueueCnt = smartGPIO1.GetPortAQueueWaitingCount();
}
```

VB 사용법

```
Private m_uiQueueCnt As UInteger
Private Sub SetPortASpeed()
  If smartGPIO1.GetPortAQueueWaitingCount() > m_uiQueueCnt Then
    smartGPIO1.SetPortAWatchIdleInterval(100000)
  Else
    smartGPIO1.SetPortAWatchIdleInterval(10)
  End If
  m_uiQueueCnt = smartGPIO1.GetPortAQueueWaitingCount()
End Sub
```

이벤트 `EvtPortADatasChange, EvtPortBDatasChange, EvtPortCDatasChange, EvtPortDDatasChange, EvtPortEDatasChange, EvtPortFDatasChange, EvtPortGDatasChange, EvtPortHDatasChange`

Port-A, B, C, D, E, F, G, H 각 Port에 입력되는 신호의 상태가 변경되었을 경우 발생되는 이벤트입니다.
입력 신호에 따른 즉시 처리가 필요한 경우 사용됩니다.

사용법

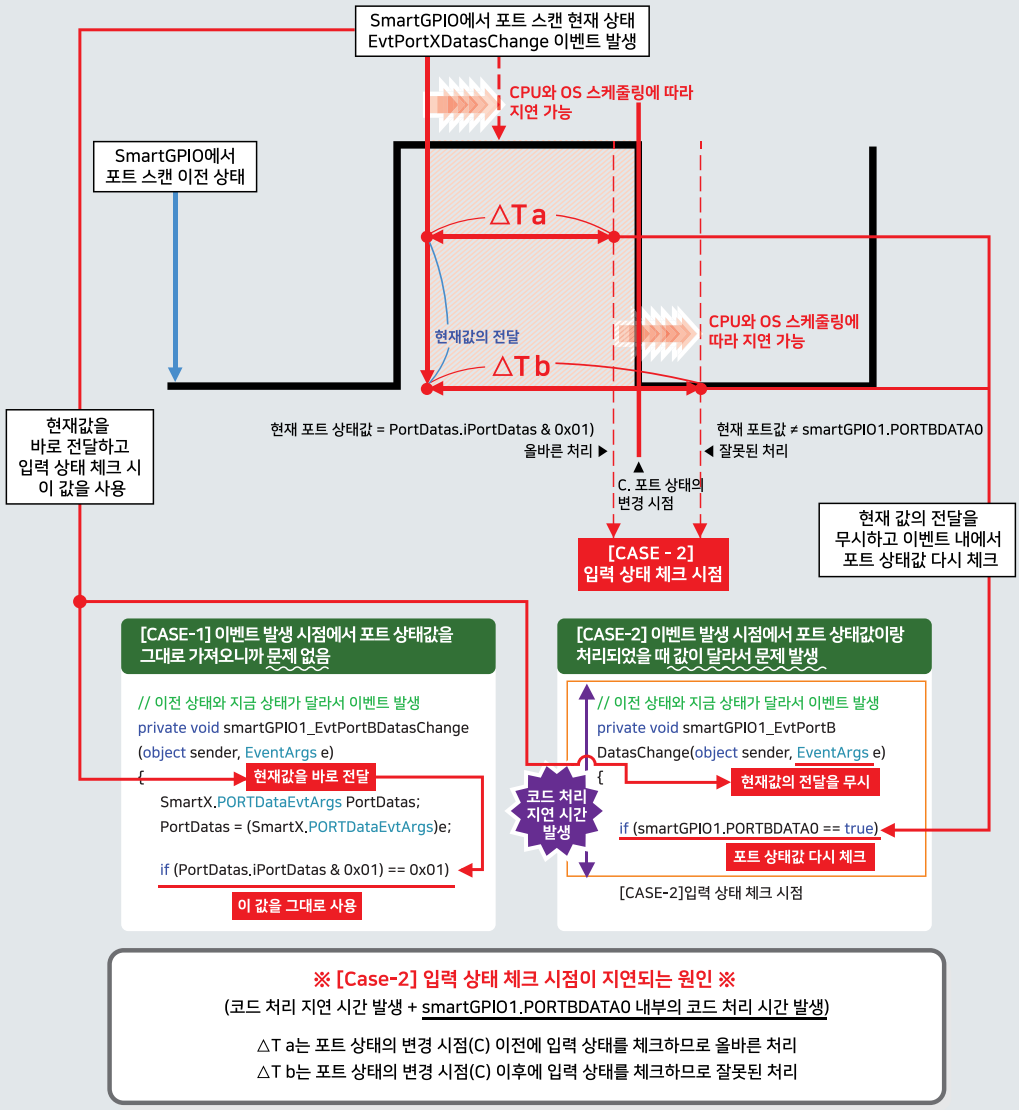
참고 "EvtPortXDatasChange 이벤트 방식"과 "EvtPortXDatasChange vs EvtPortXDatasChangeCapture 비교"를 참고하시기 바랍니다.

중요 EvtPortXDatasChange 이벤트 인자 관련 사항

1. EventArgs 인자값을 이용한 핀 상태 확인 방법

Port의 상태 변경 즉 HIGH → LOW 또는 LOW → HIGH 등으로 변경될 경우 이벤트가 발생되며 발생된 상태에서의 Port 데이터를 인자로 전달하도록 설계되었습니다. 그러므로 이벤트가 발생된 상태에서 별도의 Port 확인 작업이 필요 없이 인자를 타입 캐스트 하여 전달된 값을 확인하면 어느 Port에서 Pin의 상태가 변경되었는지 확인할 수 있습니다.

2. PortChange Event에서 인자를 사용하지 않을 경우 발생될 수 있는 문제점



※ [Case-2] 입력 상태 체크 시점이 지연되는 원인 ※

(코드 처리 지연 시간 발생 + smartGPIO1.PORTBDATA0 내부의 코드 처리 시간 발생)

△T a는 포트 상태의 변경 시점(C) 이전에 입력 상태를 체크하므로 올바른 처리
△T b는 포트 상태의 변경 시점(C) 이후에 입력 상태를 체크하므로 잘못된 처리

이벤트

```
EvtPortADatasChangeCapture, EvtPortBDatasChangeCapture,
EvtPortCDatasChangeCapture, EvtPortDDatasChangeCapture,
EvtPortEDatasChangeCapture, EvtPortFDatasChangeCapture,
EvtPortGDatasChangeCapture, EvtPortHDatasChangeCapture
```

Port-A, B, C, D, E, F, G, H 각 Port에 입력되는 신호의 상태가 변경되었을 경우 발생되는 이벤트입니다.
일정 시간 빠른 속도의 연속적인 외부 입력 신호를 카운팅하는 경우 사용됩니다.

◉ PortDataEventHandler EvtPortADatasChangeCapture(int iPortDatas)
~ EvtPortHDatasChangeCapture(int iPortDatas)

[인자]

- int iPortDatas : 데이터 Pin의 상태값

사용법

참고 "EvtPortXDatasChange 이벤트 방식"과 "EvtPortXDatasChange vs EvtPortXDatasChangeCapture 비교"를 참고하시기 바랍니다.

7) SmartGPIO 예제 사용하기

SmartGPIO를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartGPIO", "SmartGPIO Capture"

2. SmartADC

IEC-Series 제품 내부에 4~6채널의 10/12Bit ADC(Analog to Digital Converter)가 내장되어 있습니다. ADC 입력 단자는 Extension Port-Ⅰ,Ⅱ 커넥터의 AIN0~AIN3(AIN4, 5)번 Pin을 사용하고 있으며, 입력 전압 레벨은 0 ~ 3.3V입니다. IEC1000-Series의 경우 ReadDataFast() 메소드를 이용하여 빠르게 ADC 값을 읽어올 수 있으며, ReadDataDetailFiltering() 메소드 또한 최적화를 통하여 IEC266/667-Series에 비해 빠르게 ADC 값을 읽어올 수 있습니다. SmartADC를 사용하시면 외부 센서의 인터페이스를 아주 쉽게 처리하실 수 있습니다.

- 제품별 최대 4~6 채널 지원
- 12Bit(4096) 분해능 지원
- SmartI/O – ADC Block 연동 지원
- Software 방식의 필터링 기능 지원
- 입력 전압
  - ↳ Extension Port 직접 입력 시 : DC 0~3.3V
  - ↳ SmartI/O – ADC Block 입력 시 : DC 0~5V or DC 0~10V(스위치 선택)

참고 자세한 Pin Map 정보는 “Part-VI. 하드웨어 장치 안내”를 참고하시기 바랍니다.

1) ADC 변환 성능

SmartADC를 이용한 ADC 변환 성능은 IEC-Series별로 차이가 있습니다. 아래 제품별 사양 및 성능표를 확인해보시기 바랍니다.

[표] IEC266-Series / IEC667-Series / IEC1000-Series ADC 사양

제품군	IEC266-Series		IEC667-Series		IEC1000-Series	
	Lite	非Lite	Lite	非Lite	Lite	非Lite
채널 수	4Ch		4Ch		4Ch	6Ch
분해능	10Bit(1024)		12Bit(4096)		12Bit(4096)	
필터링 지원 여부	지원		지원		지원	
고속 입력 지원 여부	미지원		미지원		지원	

1-1) 제품에 따른 언어별 변환 성능

SmartADC의 ADC Sampling Time(측정 시간)과 Tollerance(떨림값)은 IEC-Series와 개발 언어에 따라 차이가 발생합니다. SmartADC는 ADC Sampling을 위해 ReadData(), ReadDataFast(), ReadDataDetailFiltering() 메소드를 지원하며 아래 CASE에서 각 메소드의 성능을 확인해보시기 바랍니다.

ReadData(), ReadDataFast() 메소드 성능 테스트	
테스트 방법	1) IEC-Series의 전원부에 페라이트 코어를 장착하고, ADC로 인가되는 전압은 배터리를 이용한다. 2) ReadData(C++ : GetADCData) 메소드의 FilteringEnable을 변경해가며 1000회 반복하여 SamplingTime과 Tollerance(떨림값)을 측정한다. 3) IEC1000-Series의 경우 ReadDataFast(C++ : GetADCDataFast) 메소드를 1000회 반복하여 Sampling Time과 Tollerance(떨림값)을 측정한다.

테스트 방법	C#	`int[] iADC = new int[1000];`
		`for(int i = 0; i < 1000; i++)`
		`{`
		`  iADC[i] = smartADC1.ReadData(0);`
		`  // iADC[i] = smartADC1.ReadDataFast(0);`
		`}`

	VB	`Dim iADC As Integer() = New Integer(999) {}`
		`For i As Integer = 0 To 1000 - 1`
		`  iADC(i) = smartADC1.ReadData(0)`
		`  ' iADC(i) = smartADC1.ReadDataFast(0)`
		`Next`

	C++	`int iADC[1000];`
		`for(i=0;i<1000;i++)`
		`{`
		`  iADC[i] = m_SmartADC.GetADCData(0);`
		`  // iADC[i] = m_SmartADC.GetADCDataFast(0);`
		`}`

Sampling Time(단위 : Second) 및 Tollerance 측정 결과

메소드	Filtering Enable	항목	1000 CE6.0(B6버전)		667 CE6.0(B23버전)		266 CE5.0(B15버전)	
			C#, VB	C++	C#, VB	C++	C#, VB	C++
ReadData()	true	SamplingTime	0.0916	0.0907	3.0499	3.0511	3.701	3.701
		Tollerance	8	8	2	3	8	4
	false	SamplingTime	0.0413	0.0408	0.3084	0.3055	0.617	0.317
		Tollerance	13	14	13	13	17	17
ReadDataFast()	-	SamplingTime	0.0161	0.0158	-	-	-	-
		Tollerance	23	23	-	-	-	-

주의 ReadData(), ReadDataFast() 메소드 테스트 시 주의사항

1. ADC와 외부 케이블을 연결하여 사용할 경우 케이블이 길어지면 ADC 값이 흔들릴 수 있습니다.
따라서 오차가 발생할 수 있으므로, 케이블 길이를 되도록 짧게 하여 사용하시기 바랍니다.

2. ReadDataFast() 메소드는 IEC1000-Series에만 적용되며 FilteringEnable 값에 영향을 받지 않습니다.

3. FilteringEnable 값에 따라 SamplingTime과 Tollerance 값에 많은 차이가 있습니다.

4. Tollerance는 최대 측정값과 최소 측정값의 차이이며, 케이블 및 측정 전압의 전기적인 환경(노이즈 등)에 따라서 오차는 달라질 수 있습니다.

ReadData(), ReadDataFast() 메소드 성능 테스트

테스트 방법	1) IEC-Series의 전원부에 페라이트 코어를 장착하고, ADC로 인가되는 전압은 배터리를 이용합니다. 2) ReadDataDetailFiltering(C++ : GetDetailFilteringADCData) 메소드의 iRawDataCnt와 iAvailable Cnt 값을 아래의 표와 같이 변경할 때마다 테스트하여 Sampling Time을 측정합니다.	
테스트 코드	C#	`int iADC = smartADC1.ReadDataDetailFiltering(0, iRaw, iAve);`
	VB	`Dim iADC As Integer = smartADC1.ReadDataDetailFiltering(0, iRaw, iAve)`
	C++	`int iADC = m_SmartADC.GetDetailFilteringADCData(0, iRaw, iAve);`

Sampling Time 측정 결과(단위 : Second)							
ReadDataDetailFiltering		1000 CE6.0(B6버전)		667 CE6.0(B23버전)		266 CE5.0(B15버전)	
iRawDataCnt	iAvailableCnt	C#, VB	C++	C#, VB	C++	C#, VB	C++
100	80	0.0009	0.0008	0.3045	0.3049	0.370	0.371
	40	0.0009	0.0008	0.3045	0.3049	0.370	0.371
	20	0.0009	0.0008	0.3045	0.3049	0.371	0.371
500	400	0.0042	0.0042	1.5242	1.5242	1.856	1.855
	200	0.0042	0.0042	1.521	1.5241	1.858	1.858
	100	0.0042	0.0042	1.5218	1.5236	1.855	1.853
1000	800	0.0084	0.0084	3.0447	3.0506	3.721	3.717
	400	0.0083	0.0084	3.0467	3.0523	3.714	3.713
	200	0.0083	0.0084	3.0464	3.0511	3.717	3.718

주의 ReadDataDetailFiltering() 메소드 테스트 시 주의사항

SamplingTime은 iRawDataCnt의 값에 따라 달라지며, iAvailableCnt 값은 ADC 값의 Tollerance(떨림값)과 응답성에 관련이 있습니다.

2) 출력 데이터에 따른 입력 전압 계산 방법

IEC-Series는 제품별로 최대 10Bit(1024)/12Bit(4096) 분해능의 ADC 성능을 가지고 있습니다. 따라서 현재 ADC 값과 최대 분해능의 비율을 최대 입력 전압에 적용하면 입력 받은 전압값을 계산할 수 있습니다. 마찬가지로, 센서의 출력 범위를 알고 있다면 현재 ADC 값에 따른 센서값을 계산할 수 있습니다. 아래 표에서 공식과 예시를 확인 후 프로젝트에 적용해보시기 바랍니다.

[표] ADC 출력 데이터에 따른 입력 전압 및 센서값 계산 공식 및 예시

공식 1 ADC 입력값에 따른 입력 전압 계산 공식

$$\text{Input Voltage} = \frac{\text{MAX Input Voltage}}{\text{ADC Resolution}} \times \text{ADC Read Data}$$

- Input Voltage : 입력되는 전압값
- ADC Read Data : ADC 입력 데이터
- ADC Resolution : ADC 최대 해상도(ex : 12비트의 경우 4095)
- Max Input Voltage : 최대 입력 전압 (IEC-Series에 직접 입력 시: 3.3V), (ADC Block 사용 시 : 5V 또는 10V)

공식 2 ADC 입력값에 따른 센서의 물리적인 측정값 계산 공식

$$\text{Sensor Value} = \left(\frac{(\text{Smax}-\text{Smin})}{\text{ADC Resolution}} \times \text{ADC Read Data}\right) + \text{Smin}$$

- Sensor Value : ADC로 입력되는 전압에 따른 센서값
- Smax : 센서의 물리적인 최대 측정 범위
- Smin : 센서의 물리적인 최소 측정 범위
- ADC Read Data : ADC 입력 데이터
- ADC Resolution : ADC 최대 해상도(ex : 12비트의 경우 4095)

예시 0~250Kg을 센싱하는 로드셀 센서에서 출력되는 0~5V 사이의 전압값을 IEC1000-Series에서 ADC로 입력받은 후, 화면에 입력 받은 전압값과 센서값을 표시

먼저 IEC1000-Series을 사용하므로 ADC Resolution 값은 4095이며, 입력받은 ADC Read Data 값은 2048이라고 가정합니다.

- ADC Max Resolution : 4095
- ADC Read Data : 2048

로드셀 센서는 0~5V 전압을 출력하고, 0~250Kg을 센싱하므로 Max Input Voltage 값은 5이며, Smax는 250, Smin은 0 입니다.
- Max Voltage : 5
- Smax : 250
- Smin : 0

1. 입력받은 전압값을 화면에 표시하기 위해 [공식 1]을 적용하여 입력 전압을 계산합니다.

$$\text{Input Voltage} = \frac{5}{4095} \times 2048 = 2.5$$

즉, 입력받은 전압값은 2.5V입니다.

2. 입력받은 전압값에 따른 센서값을 화면에 표시하기 위해 [공식 2]를 적용하여 센서값을 계산합니다.

$$\text{Sensor Value} = \left(\frac{(250 - 0)}{4095} \times 2048\right) + 0 = 125$$

즉, 입력받은 전압값에 따른 센서값은 125Kg입니다.

[결과]
1. 입력받은 전압값 : 2.5V
2. 입력받은 센서값 : 125Kg

3) 프로그래밍 적용 가이드

STEP-1 A/D 변환값 읽기

SmartADC를 사용해 A/D 변환값을 읽는 방법은 크게 필터링 되지 않은 값을 읽는 방법과 필터링된 값을 읽는 방법이 있습니다. 필터링 되지 않은 값을 읽는 방법은 ReadData() 또는 ReadDataFast() 메소드를 호출하는 방법이 있으며, ReadDataFast() 메소드는 IEC1000-Series에서만 지원됩니다.
필터링 된 값을 읽는 방법은 ReadData() 메소드와 FilteringEnable 속성을 사용하는 방법과 ReadDataDetailFiltering() 메소드를 호출하는 방법이 있습니다.

※ 자세한 내용은 "FilteringEnable 속성", "ReadData(), ReadDataDetailFiltering(), ReadData() 메소드"를 참고하시기 바랍니다.

```
int iADCData;   // A/D 변환값을 저장할 변수
iADCData = smartADC1.ReadDataFast(0); // Type-1
iADCData = smartADC1.ReadData(0);     // Type-2
smartADC1.FilteringEnable = true;     // 필터링 설정(기본값)
iADCData = smartADC1.ReadData(0);     // Type-3
iADCData = smartADC1.ReadDataDetailFiltering(0, 500, 400);  // Type-4
```

아래로 갈수록 Sampling 속도는 느려지고, 떨림값은 작아집니다.

4) SmartADC 인터페이스 설명

SmartADC Component Interface		
속성		
bTouchEnable : bool	FilteringEnable : bool	PreScaleVal : ulong
메소드		
ReadData(int ch) : int	ReadDataDetailFiltering(int iCh, int iRawdataCnt, int iAvailableCnt) : int	ReadDataFast(int ch) : int
TouchEnable(bool bEnable) : void		

프로퍼티(속성) bTouchEnable

화면 터치 활성화 여부를 확인합니다. 읽기 전용 속성입니다.

참고 TouchEnable() 메소드를 참고하시기 바랍니다.

주의 IEC1000-Series에서만 지원되는 기능이므로 사용 시 주의하시기 바랍니다.

- bool : 화면 터치 활성화 여부
  - true : 활성화 됨
  - false : 비활성화 됨

C# 사용법

```
if (smartADC1.bTouchEnable == true)
{
  // 현재 터치가 활성화 됨
}
```

VB 사용법

```
If smartADC1.bTouchEnable = True Then
  ' 현재 터치가 활성화 됨
End If
```

프로퍼티(속성) FilteringEnable

A/D 변환 시 노이즈 성분의 신호를 저감할 수 있도록 상위 20% 하위 20%의 데이터를 무시한 상태의 데이터들의 평균을 계산하여 A/D 결과로 반환합니다.

- bool : 필터 사용 여부
  - true : 사용
  - false : 사용 안 함

C# 사용법

```
smartADC1.FilteringEnable = true;
```

VB 사용법

```
smartADC1.FilteringEnable = True
```

프로퍼티(속성) PreScaleVal

A/D 변환 시간 관련 설정값입니다. (PreScaleVal 값은 49가 권장값입니다.)

주의 PreScaleVal의 설정값은 49 이상으로 설정하시기 바랍니다

- ulong : 프리스케일러 값 (권장값 : 49)

C# 사용법

```
smartADC1.PreScaleVal = 49;
```

VB 사용법

```
smartADC1.PreScaleVal = 49
```

메소드(함수) ReadData

A/D 변환된 결과값(데이터 범위는 0~1023 또는 0~4095)을 반환합니다. 입력 채널의 선택은 메소드(함수)인자를 통해 선택됩니다. 만약 FilteringEnable 속성값이 true일 경우 리턴값은 노이즈를 감소시키기 위한 알고리즘이 동작되어 결과 데이터를 적용하여 반환합니다.

◉ int ReadData(int ch)

[인자]

• int ch : 입력 채널 번호

[표] ADC Channel에 따른 인자값

ADC Channel	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5
ch	0	1	2	3	4	5

※ AIN4, AIN5의 경우 IEC1000非Lite-Series에서만 지원됩니다.

[리턴값]

• int : A/D 변환된 결과값(데이터 범위 : 0~1023 또는 0~4095)

C# 사용법

```
int iADCData = smartADC1.ReadData(0);
```

VB 사용법

```
Dim iADCData As Integer = smartADC1.ReadData(0)
```

메소드(함수) ReadDataDetailFiltering

ADC를 통하여 외부의 Analog 신호를 읽을 경우 노이즈 등으로 인하여 데이터가 흔들리게 되며 이러한 이유로 데이터의 오차가 발생합니다. 이렇게 흔들리는 신호를 H/W적인 Filter를 설계하여 원인을 해결할 수도 있지만 실상 Filter를 설계해서 적용하기란 그리 쉽지 않으며 외부 환경의 변화에 따라서 많은 영향을 받는 부분이 있어 적용이 어렵습니다. 따라서 일반적으로 S/W적인 방법으로 Filtering한 효과를 얻을 수 있도록 Filtering Algorithm을 개발하여 적용하고 있습니다. 이에 따라 SmartADC에서 쉽게 Filtering 기능을 제어하여 원하는 입력값을 얻을 수 있도록 S/W적인 Filtering 기능을 지원하고 있습니다.

◉ int ReadDataDetailFiltering(int iCh, int iRawdataCnt, int iAvailableCnt)

[인자]

• int iCh : 아날로그 입력 채널

• int iRawdataCnt : 필터링 전의 외부 아날로그 신호 개수

• int iAvailableCnt : RawdataCnt에서 Max, Min 값을 제거한 데이터 신호 개수

[표] ADC Channel에 따른 인자값

ADC Channel	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5
ch	0	1	2	3	4	5

※ AIN4, AIN5의 경우 IEC1000非Lite-Series에서만 지원됩니다.

[리턴값]

• int : A/D 변환된 결과값(데이터 범위 : 0~1023 또는 0~4095)

참고

1. ReadData 메소드를 사용하여 Filtering Enable을 체크하는 경우 iRawdataCnt는 10, iAvailableCnt는 4 값을 사용합니다.
2. IEC1000-Series의 경우 최적화를 통하여 IEC266/667-Series에 비해 Sampling 속도가 훨씬 빠릅니다.

C# 사용법

```
// 0 : AD 입력 채널, 500 : RawData, 400 : Available. ADC 값을 구해 iADC 변수에 저장
int iADC = smartADC1.ReadDataDetailFiltering(0, 500, 400);
```

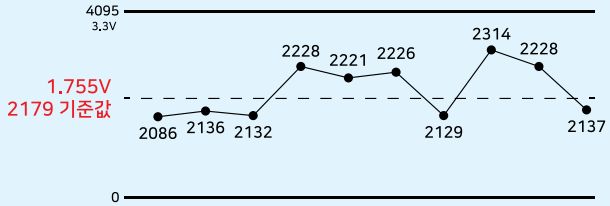
VB 사용법

```
Dim iADC As Integer = smartADC1.ReadDataDetailFiltering(0, 500, 400)
```

참고 ReadDataDetailFiltering() 메소드 동작 알고리즘

ReadDataDetailFiltering() 메소드 호출 시 iRawdataCnt 인자값은 11 ~ 5000 범위 내에서 사용 가능하며, iAvailableCnt 인자값은 1 ~ (iRawdataCnt - 2) 범위 내에서 사용 가능합니다. 여기서 2를 빼주는 이유는 RawData에서

Max, Min 값을 빼주기 위함입니다. 아래 Step을 확인해 보시기 바랍니다.



[STEP-1] RawData(Analog Input Data)는 상기와 같이 불규칙적으로 입력됩니다. 이때 iRawdatCnt 인자값의 크기만큼 RawData를 추출합니다.

EX) iRawDataCnt가 10인 경우

Index	0	1	2	3	4	5	6	7	8	9
AD 값	2086	2136	2132	2228	2221	2226	2129	2314	2228	2137

[STEP-2] ReadDataDetailFiltering을 통해 오름차순으로 Sorting 됩니다.

Index	0	1	2	3	4	5	6	7	8	9
AD 값	2086	2129	2132	2136	2137	2221	2226	2228	2228	2314

[STEP-3] iAvailableCnt 값에 따라 A/D 값의 평균을 구하여 리턴합니다

Ex1) iAvailableCnt가 6인 경우. 호출 코드 : ReadDataDetailFiltering(0, 10, 6)

Index	~~0~~	~~1~~	2	3	4	5	6	7	~~8~~	~~9~~
AD 값	~~2086~~	~~2129~~	2132	2136	2137	2221	2226	2228	~~2228~~	~~2314~~
리턴값	(2132 + 2136 + 2137 + 2221 + 2226 + 2228) / 6 = 2180(계산값) 리턴값 : 2180									

Ex2) iAvailableCnt가 8인 경우. 호출 코드 : ReadDataDetailFiltering(0, 10, 8)

Index	~~0~~	1	2	3	4	5	6	7	8	~~9~~
AD 값	~~2086~~	2129	2132	2136	2137	2221	2226	2228	2228	~~2314~~
리턴값	(2129 + 2132 + 2136 + 2137 + 2221 + 2226 + 2228 + 2228) / 8 = 2179.625(계산값) 리턴값 : 2179									

메소드(함수) ReadDataFast

IEC1000-Series에서만 사용 가능한 메소드이며 FilteringEnable 값에 영향을 받지 않습니다.
Sampling 속도가 ReadData() 메소드에 비해 빠르지만 Tollerance(떨림값)이 큽니다.

	ReadData()	ReadDataFast()
장점	Tollerance(떨림값)이 ReadDataFast() 메소드에 비해 작음	Sampling속도가 ReadData() 메소드에 비해 빠름
단점	Sampling속도가 ReadDataFast() 메소드에 비해 느림	Tollerance(떨림값)이 ReadData() 메소드에 비해 큼

참고 "제품에 따른 언어별 변환 성능"를 참고하시기 바랍니다.

주의 ReadDataFast() 메소드는 IEC1000-Series에만 적용됩니다.

◉ `int ReadDataFast(int ch)`

[인자]

• `int ch` : 입력 채널 번호

[표] ADC Channel에 따른 인자값

ADC Channel	AIN0	AIN1	AIN2	AIN3	AIN4	AIN5
ch	0	1	2	3	4	5

※ AIN4, AIN5의 경우 IEC1000非Lite-Series에서만 지원됩니다.

[리턴값]

• int : A/D 변환된 결과값(데이터 범위 : 0~4095)

C# 사용법

```
int iADCDataFast = smartADC1.ReadDataFast(0);
```

VB 사용법

```
Dim iADCDataFast As Integer = smartADC1.ReadDataFast(0)
```

메소드(함수) TouchEnable

화면 터치 활성화 비활성화 설정

주의 IEC1000-Series에서만 지원되는 기능이므로 사용 시 주의하시기 바랍니다.

◉ void TouchEnable(bool bEnable)

[인자]

• bool : 화면 터치 활성화 여부
  – true : 활성화
  – false : 비활성화

C# 사용법

```
smartADC1.TouchEnable(true);  // 터치 활성화
```

VB 사용법

```
smartADC1.TouchEnable(True)
```

5) SmartADC 예제 사용하기

SmartADC를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartADC"

3. SmartSerialPort

SmartSerialPort는 Serial 통신을 처리하기 위한 컴포넌트로 동기-Polling 또는 비동기-Event 방식의 수신 기능을 지원합니다. 또한, Frame 구조와 None-Frame 구조의 통신을 모두 지원하여 다양한 통신 환경에 쉽고 편리하게 적용하여 개발할 수 있도록 만들어졌습니다.

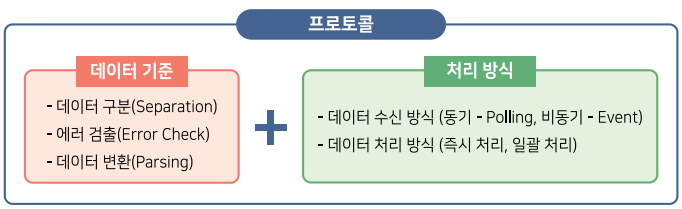
- ■ 동기-Polling 또는 비동기-Event 방식의 수신 기능 지원
- ■ 수신 인디케이터 기능 지원
  - └ Open된 Port에 데이터가 수신됨을 확인하는 모니터링 기능 지원
- ■ 비동기-Event 방식으로 수신 시 수신 Queue 지원
- ■ Frame 구조의 송/수신 기능 지원
  - └ 다양한 Frame 구조를 지원
  - └ Text Protocol 및 Binary Protocol 모두 지원
  - └ 식별 코드 변경 가능
- ■ None-Frame 구조의 송/수신 기능 지원
  - └ 데이터의 구분 기준은 ReadTimeOut 또는 FrameBufferSize 기준
- ■ Error Check 기능 지원
  - └ Frame/None-Frame 구조 모두 적용 가능
  - └ CheckSum8, CheckSum16, CRC16, CRC32 방식 지원
  - └ Error Code 위치 설정 가능
  - └ Error Code 표현 방식 지원(Binary, ASCII, Unicode)
- ■ 송/수신 데이터 표현 방식
  - └ Binary, ASCII, Unicode
- ■ RS485 통신 지원
  - └ IEC-Series중 RS485-S/W 방식의 제품인 경우 송/수신 전환 시 초기 Null 값 수신 데이터 제거 기능 지원

참고 SmartSerialPort 사용 시 참고사항

1. SmartSerialPort를 사용하여 외부의 디바이스와 연동(통신 프로그래밍) 시 문제 및 적용에 어려움이 있는 경우 HNS로 연락주시면 IEC-Series와 디바이스 연동 작업을 지원하도록 하겠습니다. 지원 시 SmartSerialPort를 기준으로 합니다.
2. 콘넥터 핀 정보에 관한 설명은 “Part-VI. 하드웨어 장치 안내”를 참고하시기 바랍니다.

주의 Port Open 후 SmartSerialPort의 설정 값을 변경하지 마십시오.

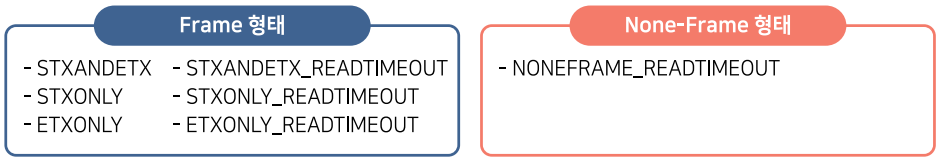
1) 데이터 구조 및 수신 처리 방식



프로토콜(Protocol)은 크게 “데이터 기준”과 “처리 방식” 2가지로 나눌 수 있습니다. 여기서 “데이터 기준”은 데이터 구분(Separation), 에러 검출(Error Check), 데이터 변환(Parsing)의 기준. 즉, 데이터 구조의 기준을 제시합니다. “처리 방식”은 데이터 수신 방식(동기-Polling, 비동기-Event), 데이터 처리 방식(즉시 처리, 일괄 처리)의 기준을 제시합니다. 다음 내용에서 SmartSerialPort에서 지원하는 “데이터 기준”과 “처리 방식”의 내용을 확인하시기 바랍니다.

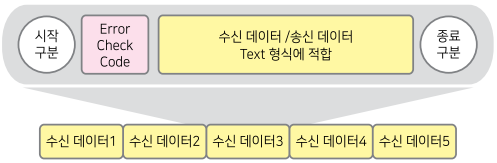
1-1) 데이터 구조 및 지원 형태

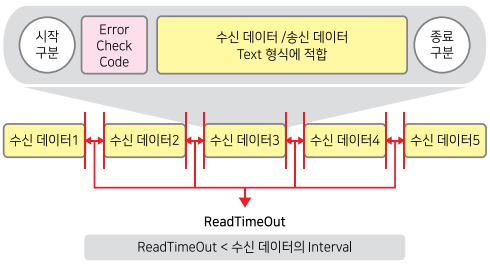
SmartSerialPort는 데이터 구분을 위해 Frame 구조와 None-Frame 구조 총 2가지 통신 구조를 지원합니다. Frame 구조의 경우 STX(시작 구분자)와 ETX(종료 구분자)를 통해 각 데이터를 구분하며, None-Frame 구조의 경우 ReadTimeOut(각 수신 데이터 사이의 시간 간격)과 FrameBufferSize(한 Frame에 저장될 수 있는 데이터의 최대 크기)를 통해 각 데이터를 구분합니다. 또한, 에러 검출을 위해 다양한 에러 체크 기능을 제공하고 있습니다. 아래 표를 확인하여 적합한 데이터 구조 및 에러 체크 기능을 적용하시기 바랍니다.

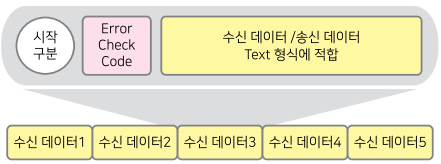


[그림] 지원하는 통신 데이터 구조

[표1] SmartSerialPort에서 지원하는 데이터 구조

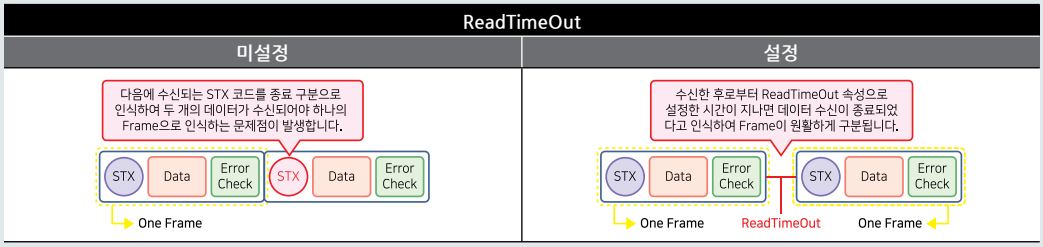
FrameSeparationType	STXANDETX
설명	STX Code와 ETX Code를 모두 사용하며, 수신받은 데이터의 STX Code부터 ETX Code까지를 하나의 Frame으로 구분하는 방식
적합 데이터 형식	Text(ASCII, Unicode 등) 형식
Frame 구조	

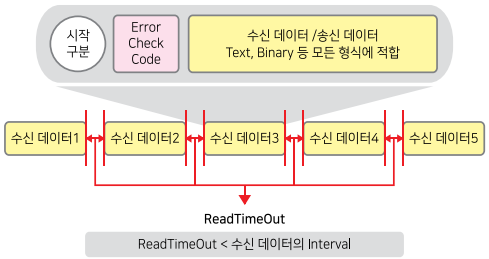
FrameSeparationType	STXANDETX_READTIMEOUT
설명	STX Code와 ETX Code를 모두 사용하며, 각 데이터를 ReadTimeOut으로 구분 후, 구분한 데이터의 STX Code부터 ETX Code까지를 하나의 Frame으로 구분하는 방식
적합 데이터 형식	Text(ASCII, Unicode 등), Binary 등 모든 형식
Frame 구조	

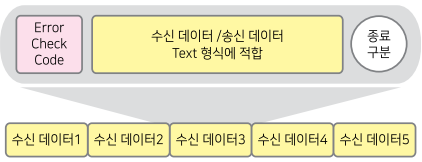
FrameSeparationType	STXONLY
설명	1Byte 또는 2Byte의 STX Code만을 사용하며, 수신받은 데이터의 STX Code부터 다음에 수신되는 STX Code까지를 하나의 Frame으로 구분하는 방식
적합 데이터 형식	Text(ASCII, Unicode 등) 형식
Frame 구조	

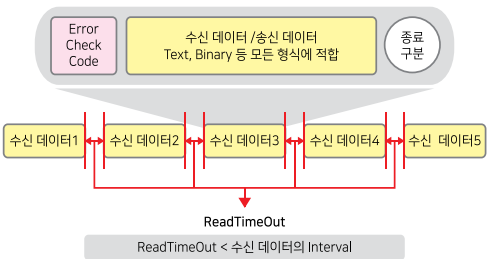
중요 STXONLY인 경우 ReadTimeOut을 설정하지 않았을 때의 문제점

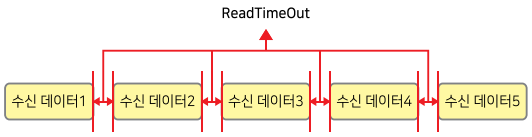
STXONLY 프레임 구조의 시작 구분은 해당 데이터의 STX 코드이지만 종료 구분은 다음에 송신되는 데이터의 STX 코드입니다. 이러한 이유로 STXONLY 구조를 사용 시 ReadTimeOut 속성을 설정해 일정 시간이 지난 후 더 이상의 송신이 없다면 수신을 강제 종료하는 과정이 필요합니다.



FrameSeparationType	STXONLY_READTIMEOUT
설명	1Byte 또는 2Byte의 STX Code만을 사용하며, 각 데이터를 ReadTimeOut으로 구분 후, 구분한 데이터의 STX Code부터 다음에 수신되는 STX Code까지를 하나의 Frame으로 구분하는 방식
적합 데이터 형식	Text(ASCII, Unicode 등), Binary 등 모든 형식
Frame 구조	

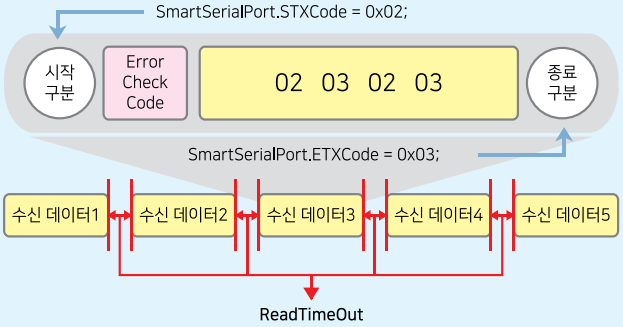
FrameSeparationType	ETXONLY
설명	1Byte 또는 2Byte의 ETX Code만을 사용하며, 수신받은 데이터의 ETX Code부터 다음에 수신되는 ETX Code까지를 하나의 Frame으로 구분하는 방식
적합 데이터 형식	Text(ASCII, Unicode 등) 형식
Frame 구조	

FrameSeparationType	ETXONLY_READTIMEOUT
설명	1Byte 또는 2Byte의 ETX Code만을 사용하며, 각 데이터를 ReadTimeOut으로 구분 후, 구분한 데이터의 ETX Code부터 다음에 수신되는 ETX Code까지를 하나의 Frame으로 구분하는 방식
적합 데이터 형식	Text(ASCII, Unicode 등), Binary 등 모든 형식
Frame 구조	

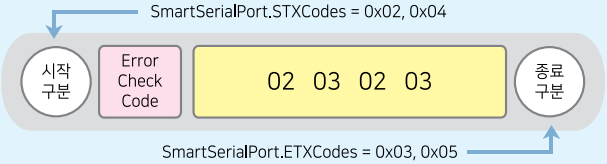
FrameSeparationType	NONEFRAME_READTIMEOUT
설명	STX Code와 ETX Code를 모두 사용하지 않으며, 각 데이터를 ReadTimeOut으로 구분하며. FrameBufferSize로 각 프레임에 저장 가능한 데이터의 최대 크기를 설정합니다.
적합 데이터 형식	Text(ASCII, Unicode 등), Binary 등 모든 형식
Frame 구조	

참고 Binary 통신 시 데이터에 시작 구분자 또는 종료 구분자가 중복되는 경우 해결 방법

이 경우 온전한 데이터를 수신하지 못 하는 문제가 발생합니다. 이에 대한 해결 방법은 두 가지가 있습니다.



1. FrameSeperationType 속성을 STXANDETX_READTIMEOUT / STXONLY_READTIMEOUT / ETXONLY READTIMEOUT으로 설정하여 해결할 수 있습니다.



2. STX, ETX Code를 2byte로 설정하여 해결할 수 있습니다.

주의 ReadTimeOut 속성값 설정 시 주의사항

FrameSeparationType이 XXX_READTIMEOUT인 경우 반드시 ReadTimeOut 속성값을 설정해야 합니다. 또한, ReadTimeOut 속성값은 데이터 Frame 간 Interval보다 작아야 합니다. 만약 ReadTimeOut 속성값이 데이터 Frame 간 Interval보다 같거나 크다면 데이터 Frame이 올바르게 구분되지 않을 수 있습니다. 아래 표를 확인하시기 바랍니다.

[표] ReadTimeOut 속성값과 데이터 Frame 간 Interval에 따른 데이터 Frame 처리

Interval	100ms		10ms	
ReadTimeOut	90ms	110ms	9ms	11ms
결과	정상 수신	데이터 Frame이 묶임	정상 수신	데이터 Frame이 묶임

※ 본 설정값은 예시이며 실제 적용 시 통신 환경에 따라 결과가 달라질 수 있으므로, 충분히 테스트하여 적정값으로 조절하시기 바랍니다.

[표2] SmartSerialPort에서 지원하는 에러 체크 기능

관련속성	설명
ErrorCheckMode	에러 체크 방식을 설정 (None, CheckSum8, CheckSum16, CRC16, CRC32 지원)
ErrorCode_Location	에러 체크 코드의 위치를 설정
ErrorCheckCodeType	에러 체크 코드의 타입을 설정 (Binary, ASCII, Unicode 지원)

1-2) 통신 방식 및 데이터 구조에 따른 송/수신 메소드

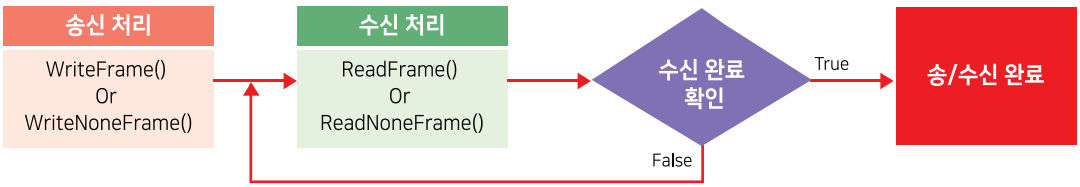
SmartSerialPort는 동기-Polling 방식과 비동기-Event 방식의 통신 방식을 지원하며 각 통신 방식 및 데이터 구조에 따라 사용하는 메소드가 상이합니다. 아래 표에서 내용을 확인하시기 바랍니다.

통신방식	동기-Polling		비동기-Event	
데이터 구조	Frame	None-Frame	Frame	None-Frame
처리 방식	즉시 수신 처리		수신 Queue 처리	
전송 메소드	WriteFrame()	WriteNoneFrame()	WriteFrame()	WriteNoneFrame()
수신 메소드	ReadFrame()	ReadNoneFrame()	ReadQueue()	

중요 해당 내용은 SmartSerialPort를 사용하기 전 반드시 숙지하기를 권장합니다.

1-3) 동기-Polling 처리 방식

동기-Polling 방식은 동기 처리 방식으로 수신 처리 메소드 호출 시 데이터가 수신될 때까지 Blocking 되어 처리되는 방식입니다. 비동기-Event 방식보다 비교적 정확하게 수신 데이터를 캐치할 수 있지만, 입력 신호를 받는 동안은 화면 터치 등 다른 처리가 불가능하다는 단점이 있습니다.



[그림] 동기 - Poling 방식 처리 과정

※ 테스트 전 반드시 "시리얼 데이터 구조 및 통신 방식에 따른 기본 설정" 내용을 참고하여 기본 설정을 진행하시기 바랍니다.

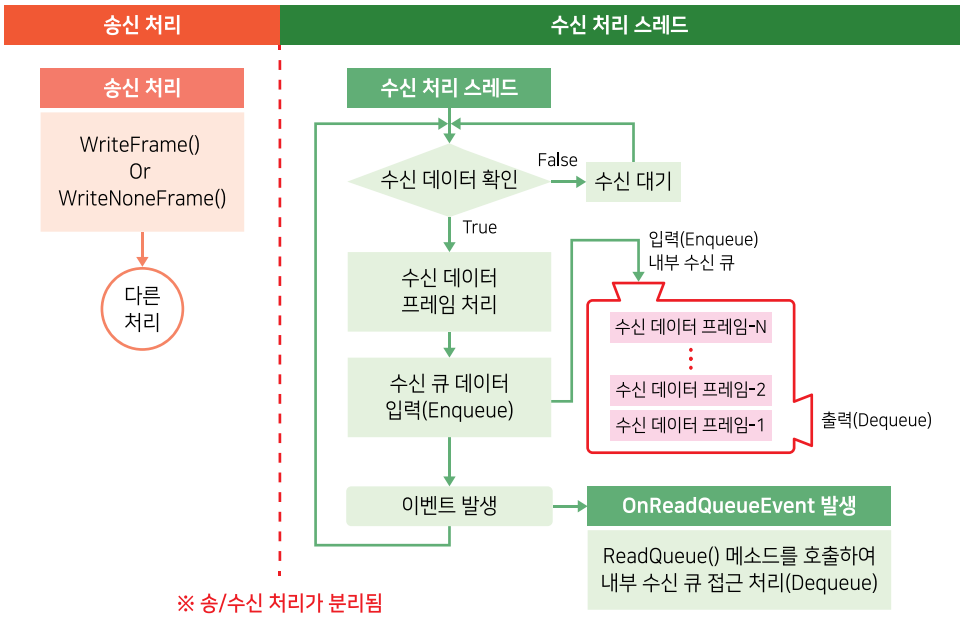
참고 [C#] 동기-Polling 방식 사용 예시 코드

```
private void Recv_Polling(byte[] bSendData, bool bIsNoneFrame)
{
  byte[] readByte;  // 데이터를 수신받을 바이트 배열
  // 초기 수신 데이터 상태를 Empty로 설정
  SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus = SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.EMPTY;
  // NoneFrameType인 경우
  if (bIsNoneFrame == true)
  {
    smartSerialPort1.WriteNoneFrame(bSendData);
  }
  else
  {
    smartSerialPort1.WriteFrame(bSendData);  // FrameType인 경우
  }
  // 데이터 전송 후 동기-Polling 방식으로 데이터를 수신받음
  while (true)
  {
    // NoneFrameType인 경우
    if (bisNoneFrame == true)
    {
```

```
      eStatus = smartSerialPort1.ReadNoneFrame(out readByte);
    }
    // FrameType인 경우
    else
    {
      eStatus = smartSerialPort1.ReadFrame(out readByte);
    }
    if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
    {
      break;  // 수신받은 데이터가 정상인 경우 데이터 처리 후 반복문을 빠져나온다. 코드 생략됨
    }
  }
}
```

1-4) 비동기-Event 처리 방식

비동기-Event 방식은 비동기 처리방식으로 사용자가 따로 수신 데이터를 체크할 필요 없이 내부의 프로세스가 수신 데이터를 수신 큐(Queue)에 저장하여 이벤트를 발생시키는 방식입니다. SmartSerialPort에서는 데이터 수신 시 OnReadQueueEvent를 발생 시켜 데이터 수신 처리를 할 수 있습니다.



[그림] 비동기 Event 처리 과정

※ 테스트 전 반드시 "시리얼 데이터 구조 및 통신 방식에 따른 기본 설정" 내용을 참고하여 기본 설정을 진행하시기 바랍니다.

참고 [C#] 비동기-Event 방식 사용 예시 코드

```
// FrameType에 따라 데이터를 전송한다.
private void SendData(bool bIsNoneFrame)
{
  byte[] bSendData = Encoding.ASCII.GetBytes( " Event Data " );  // 전송할 데이터를 변환
  // NoneFrameType인 경우
  if (bIsNoneFrame == true)
```

```
  {
    smartSerialPort1.WriteNoneFrame(bSendData);
  }
  // FrameType인 경우
  else
  {
    smartSerialPort1.WriteFrame(bSendData);
  }
}
//DetectType이 Event일 경우 데이터 수신 시 발생하는 이벤트
private void smartSerialPort1_OnReadQueueEvent()
{
  // 데이터를 수신받을 바이트 배열
  byte[] readByte;
  // 초기 수신 데이터 상태를 Empty로 설정
  SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus = SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.EMPTY;
  // 수신 데이터를 바이트 배열에 저장하고 상태를 체크
  eStatus = smartSerialPort1.ReadQueue(out readByte);
  // 수신 데이터 처리한다.
  // …중략…
}
```

2) 시리얼 데이터 구조 및 통신 방식에 따른 기본 설정

SmartSerialPort를 사용하기 전 데이터 구조 및 통신 방식에 따라 속성값을 설정해야 합니다. 아래 Step을 확인하여 통신 환경에 맞게 적용하시기 바랍니다.

[STEP-1] BaudRate, Port, 에러 체크 방식을 설정합니다.

```
smartSerialPort1.Baud_Rate = SmartX.SmartSerialPort.BAUDRATE.XXXXX;
smartSerialPort1.PortNo = SmartX.SmartSerialPort.COMPORTNO.COMX;
smartSerialPort1.ErrorCheckCodeType = SmartX.SmartSerialPort.ERRORCHECKCODETYPES.XXXXX;
smartSerialPort1.ErrorCheckMode = SmartX.SmartSerialPort.ERRORCHECK.XXXXX;
smartSerialPort1.ErrorCode_Location = SmartX.SmartSerialPort.ERRORCODELOCATION.XXXXX;
smartSerialPort1.ReceiveFrameDebugMode = XXX;
smartSerialPort1.RS485SoftwareDetection = XXX;
```

[STEP-2] 데이터 구조를 설정합니다.

```
smartSerialPort1.FrameSeparationType = SmartX.SmartSerialPort.FRAMESEPARATIONTYPES.XXXXX;
smartSerialPort1.STXCode = XXX;
smartSerialPort1.ETXCode = XXX;
smartSerialPort1.FrameBufferSize = XXX;
smartSerialPort1.ReadTimeout = XXX;
```

[STEP-3] 통신 방식을 설정합니다.

```
smartSerialPort1.ReceiveDetect = SmartX.SmartSerialPort.RECEIVEDETECTTYPE.XXXXX;
```

[STEP-4] Port를 Open합니다.

```
smartSerialPort1.Open();
```

주의 Port Open 후 프로그램 종료 시 반드시 Close() 메소드를 호출하시기 바랍니다.

3) 수신 버퍼 레지스터 크기에 따른 주의사항

IEC-Series의 CPU 내에 Serial 통신 기능이 포함되어 있으며 3~4개의 COM Port를 가지고 있습니다. 각각의 COM Port 마다 수신 버퍼 레지스터(Receive Buffer Register)를 가지고 있으며, IEC-Series별로 크기가 상이합니다. 또한, 수신 버퍼 레지스터의 크기에 따라서 즉, IEC-Series별로 시리얼 통신 중 터치 입력이나 다른 작업으로 CPU의 사용량이 증가할 경우, 시리얼 디바이스 드라이버에서 수신 버퍼 레지스터의 데이터를 가져오는 시간이 길어져 수신 데이터의 유실, 데이터의 깨짐 또는 Buffer OverRun Error가 발생될 수 있습니다. 이는 수신 데이터의 크기가 레지스터 크기보다 큰 상태에서 연속적으로 수신될 경우 버퍼를 비울 수 없어 발생되는 현상입니다. 이 현상은 특히 CPU의 수신 버퍼 레지스터에서 메모리로 데이터 복사 시 가용 CPU Resource가 낮거나, 통신 BaudRate가 높거나, 통신 데이터가 많은 경우 쉽게 발생합니다. 아래 표에서 IEC-Series별 수신 버퍼 레지스터의 크기를 확인하시기 바랍니다.

[표] IEC-Series별 수신 버퍼 레지스터의 크기

IEC-Series	COM1 & COM1_1	COM2	COM3	COM4
IEC266-Series	16Byte	16Byte	16Byte	-
IEC667-Series	64Byte	64Byte	64Byte	64Byte
IEC1000-Series	256Byte	64Byte	16Byte	16Byte

참조 수신 버퍼 레지스터 크기와 관련된 자세한 내용은 "홈페이지 → 자료실 → Tech Note → 11. 제품별 시리얼 포트의 대량 수신 데이터 처리관련 주의사항" 내용을 참조하시기 바랍니다.

4) RS485 통신 시 주의사항

IEC-Series는 제품별로 RS485 송/수신 제어 방식이 H/W 방식인 제품과 S/W 방식인 제품이 있습니다. H/W 방식의 경우 첫 데이터 수신 시 별다른 처리를 하지 않아도 되지만, S/W 방식은 속성값을 설정하지 않으면 첫 데이터 수신 시 Null 값이 수신되어 에러가 발생합니다. 이 현상은 SmartSerialPort를 사용하여 개선할 수 있으며, .NET Compact Framework에서 제공하는 SerialPort 사용 시 RtsEnable 속성을 true로 설정하여 개선할 수 있습니다. 또한, Request-Response 통신 구조인 RS485 통신에서 H/W 방식은 S/W 방식보다 빠르게 Request 수신에 대한 Response 응답을 빠르게 처리할 수 있습니다. 아래 표에서 자세한 내용을 확인하시기 바랍니다.

[표1] H/W 방식과 S/W 방식의 차이점

구분	H/W 방식	S/W 방식
특징	1. Request 수신에 대한 Response 응답을 S/W 방식에 비해 빠르게 할 수 있음 2. Port Open 후 첫 데이터 수신 시 Null 값이 수신되지 않음	1. Request 수신에 대한 Response 응답이 H/W 방식에 비해 느림 2. Port Open 후 첫 데이터 수신 시 Null 값이 수신됨 (SmartSerialPort를 사용하거나, 속성값 설정 필요)

[표2] IEC-Series별 송/수신 제어 방식과 RtsEnable/RS485SoftwareDetection 속성 설정 필요 여부

IEC-Series		송/수신 제어 방식	SerialPort를 사용하는 경우 (관련 속성 : RtsEnable)	SmartSerialPort를 사용하는 경우 (관련 속성 : RS485SoftwareDetection)
IEC1000-Series	非Lite	H/W	설정 필요 없음(설정값 : False)	설정 필요 없음(설정값 : False)
	Lite	S/W		
IEC667-Series	非Lite	H/W	설정 필요 없음(설정값 : False)	설정 필요 없음(설정값 : False)
	Lite	S/W	**설정 필요함(설정값 : True)**	설정 필요 없음(설정값 : False)
IEC266-Series	Lite	S/W	**설정 필요함(설정값 : True)**	**설정 필요함(설정값 : True)**

주의 RtsEnable, RS485SoftwareDetection 속성은 Port Open 전 한 번만 설정합니다.

중요 RS485 송/수신 처리 시 권장사항

1. 데이터 수신 시 비동기-Event 방식이 아닌 동기-Polling 방식으로 프로그래밍하시기 바랍니다.

- 비동기 처리(Event) 방식인 경우 모드 제어의 명확성 및 Timing에 문제가 발생할 수 있으므로, 동기 처리(Polling) 방식으로 프로그램하시기를 권장합니다.

[표] RS485 통신 시 Half-Duflex 방식에 의한 DE(송/수신 방향) 처리 상태에 따른 모드 제어

수신 처리 상태	→	수신 모드
송신 처리 상태	→	송신 모드
대기 상태	→	수신 모드

2. 반드시 Master-Slave 즉, Request-Response 구조의 통신을 하도록 구성하시기 바랍니다.
- RS485 통신은 Half-Duflex 통신 방식이므로 동시에 송/수신을 할 수 없습니다. 따라서 반드시 Request-Response 구조의 통신을 하도록 프로토콜을 구성하시기 바랍니다.

5) 프로그래밍 적용 가이드

SmartSerialPort의 통신 방식은 크게 동기-Polling 방식과 비동기-Event 방식으로 나눌 수 있으며, 세부적으로 구현 가능한 통신 모델은 총 4가지가 있습니다. 아래 CASE를 확인하시기 바랍니다.

공통 공통 속성 설정하기

아래 CASE 모두에 공통적으로 적용되는 통신 관련 속성을 설정합니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성을 참고하시기 바랍니다.

```
smartSerialPort1.Baud_Rate = SmartX.SmartSerialPort.BAUDRATE._9600bps;  // 보드레이트를 설정합니다.
smartSerialPort1.PortNo = SmartX.SmartSerialPort.COMPORTNO.COM3;        // 통신 포트를 설정합니다.
// 에러 체크와 관련된 속성을 설정합니다.
smartSerialPort1.ErrorCheckCodeType = SmartX.SmartSerialPort.ERRORCHECKCODETYPES.ASCIICODE;
smartSerialPort1.ErrorCheckMode = SmartX.SmartSerialPort.ERRORCHECK.NONE;
smartSerialPort1.ErrorCode_Location = SmartX.SmartSerialPort.ERRORCODELOCATION.TAIL;
smartSerialPort1.ReceiveFrameDebugMode = false;   // 수신받은 프레임에서 데이터만 표시합니다.
smartSerialPort1.RS485SoftwareDetection = false;  // RS485 소프트웨어 감지를 사용하지 않습니다.
smartSerialPort1.FrameBufferSize = 50000;  // 버퍼의 크기를 설정합니다.
smartSerialPort1.ReadTimeout = 700;        // Timeout 시간을 설정합니다.
```

CASE-1 동기-Polling, None-Queue, None-Frame 방식인 경우

동기-Polling 방식의 통신 방식으로 Queue를 사용하지 않으며, None-Frame 타입의 통신 모델입니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성을 참고하시기 바랍니다.

[STEP-1] 설정 및 Open하기

동기-Polling 방식으로 사용하기 위해 ReceiveDetect 속성을 POLLING_NONEQUEUE로 설정합니다. FrameSeparationType 속성으로 프레임 타입을 None-Frame으로 설정합니다. 모든 속성 설정이 완료되면 Open() 메소드를 호출해 Port를 Open할 수 있습니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성 및 메소드를 참고하시기 바랍니다

```
// 기본적인 통신 관련 속성 설정 코드는 생략되었으며, 공통 속성 설정하기를 참고하시기 바랍니다.
smartSerialPort1.FrameSeparationType = SmartX.SmartSerialPort.FRAMESEPARATIONTYPES
                                       .NONEFRAME_READTIMEOUT;  // 프레임 타입을 설정합니다.
// 데이터 수신 방식을 설정합니다. (동기-Polling)
smartSerialPort1.ReceiveDetect = SmartX.SmartSerialPort.RECEIVEDETECTTYPE.POLLING_NONEQUEUE;
// 포트 오픈 상태 확인 후 포트를 오픈합니다.
if (smartSerialPort1.IsOpen == false)
{
```

```
  smartSerialPort1.Open();
}
```

[STEP-2] 데이터 송신하기

WriteNoneFrame() 메소드를 호출해 None-Frame 형식의 데이터를 전송할 수 있습니다. 동기-Polling 방식의 통신을 하므로, 데이터 송신 후 수신 처리를 진행합니다.

※ 자세한 내용은 "WriteNoneFrame() 메소드"를 참고하시기 바랍니다.

```
private void SendPolling_NoneFrame(byte[] bSendData)
{
  smartSerialPort1.WriteNoneFrame(bSendData);  // 바이트 배열을 None-Frame으로 전송
  RecvPolling_NoneFrame();                     // 데이터 수신을 위해 메소드를 호출
}
```

[STEP-3] 데이터 수신하기

데이터 송신 후 반복문에서 ReadNoneFrame() 메소드를 반복 호출하여 응답 데이터를 수신합니다. 리턴값을 확인해 수신한 데이터가 있다면 반복문을 빠져나오도록 코드를 작성합니다.

※ 자세한 내용은 "ReadNoneFrame() 메소드"를 참고하시기 바랍니다.

```
private void RecvPolling_NoneFrame()
{
  byte[] bRecvData;  // 데이터 수신을 위한 바이트 배열
  SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus;  // 데이터 수신 상태를 체크할 변수
  while (true)
  {
    eStatus = smartSerialPort1.ReadNoneFrame(out bRecvData);  // 데이터 수신 후 상태를 저장
    if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
    {
      // 수신 성공 처리 코드 작성
      break;  // 반복문을 빠져나옴
    }
    else if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.EMPTY)
    {
      // 수신 데이터가 없을 경우 처리 코드 작성
    }
    else
    {
      // 수신 실패 처리 코드 작성
      break;  // 반복문을 빠져나옴
    }
  }
}
```

CASE-2 동기-Polling, None-Queue, Frame 방식인 경우

동기-Polling 방식의 통신 방식으로 Queue를 사용하지 않으며, Frame 타입의 통신 모델입니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성을 참고하시기 바랍니다.

[STEP-1] 설정 및 Open하기

동기-Polling 방식으로 사용하기 위해 ReceiveDetect 속성을 POLLING_NONEQUEUE로 설정합니다. FrameSeparationType 속성으로 프레임 타입을 None-Frame이 아닌 값으로 설정합니다. 설정값에 따라 STXCode, EtxCode 속성값을 설정하시기 바랍니다. 본 가이드는 STXANDETX로 설정하였습니다. 모든 속성 설정이

완료되면 Open() 메소드를 호출해 Port를 Open할 수 있습니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성 및 메소드를 참고하시기 바랍니다.

```
// 기본적인 통신 관련 속성 설정 코드는 생략되었으며, 공통 속성 설정하기를 참고하시기 바랍니다.
// 프레임 타입을 설정합니다.
smartSerialPort1.FrameSeparationType = SmartX.SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX;
smartSerialPort1.STXCode = 0x02; smartSerialPort1.ETXCode = 0x03;
// 데이터 수신 방식을 설정합니다. (동기-Polling)
smartSerialPort1.ReceiveDetect = SmartX.SmartSerialPort.RECEIVEDETECTTYPE.POLLING_NONEQUEUE;
// 포트 오픈 상태 확인 후 포트를 오픈합니다.
if (smartSerialPort1.IsOpen == false)
{
  smartSerialPort1.Open();
}
```

[STEP-2] 데이터 송신하기

WriteFrame() 메소드를 호출해 Frame 형식의 데이터를 전송할 수 있습니다. 동기-Polling 방식의 통신을 하므로, 데이터 송신 후 수신 처리를 진행합니다.

※ 자세한 내용은 "WriteFrame() 메소드"를 참고하시기 바랍니다.

```
private void SendPolling_Frame(byte[] bSendData)
{
  smartSerialPort1.WriteFrame(bSendData);  // 바이트 배열을 Frame으로 전송
  RecvPolling_Frame();  // 데이터 수신을 위해 메소드를 호출
}
```

[STEP-3] 데이터 수신하기

데이터 송신 후 반복문에서 ReadFrame() 메소드를 반복 호출하여 응답 데이터를 수신합니다.
리턴값을 확인해 수신한 데이터가 있다면 반복문을 빠져나오도록 코드를 작성합니다.

※ 자세한 내용은 "ReadFrame() 메소드"를 참고하시기 바랍니다.

```
private void RecvPolling_Frame()
{
  byte[] bRecvData;  // 데이터 수신을 위한 바이트 배열
  SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus;  // 데이터 수신 상태를 체크할 변수
  while (true)
  {
    eStatus = smartSerialPort1.ReadFrame(out bRecvData);  // 데이터 수신 후 상태를 저장
    if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
    {
      // 수신 성공 처리 코드 작성
      break;  // 반복문을 빠져나옴
    }
    else if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.EMPTY)
    {
      // 수신 데이터가 없을 경우 처리 코드 작성
    }
    else
    {
      // 수신 실패 처리 코드 작성
      break;  // 반복문을 빠져나옴
    }
```

```
  }
}
```

CASE-3 비동기-Event, Queue, None-Frame 방식인 경우

비동기-Event 방식의 통신 방식으로 Queue를 사용하며, None-Frame 타입의 통신 모델입니다

※ 자세한 내용은 "데이터 구조 및 수신 처리 방식"을 참고하시기 바랍니다.

[STEP-1] 설정 및 Open하기

비동기-Event 방식으로 사용하기 위해 ReceiveDetect 속성을 Event_QUEUE로 설정합니다. FrameSeparationType 속성으로 프레임 타입을 None-Frame으로 설정합니다. 모든 속성 설정이 완료되면 Open() 메소드를 호출해 Port를 Open할 수 있습니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성 및 메소드를 참고하시기 바랍니다

```
// 기본적인 통신 관련 속성 설정 코드는 생략되었으며, 공통 속성 설정하기를 참고하시기 바랍니다.
smartSerialPort1.FrameSeparationType = SmartX.SmartSerialPort.FRAMESEPARATIONTYPES
                                       .NONEFRAME_READTIMEOUT;  // 프레임 타입을 설정합니다.
// 데이터 수신 방식을 설정합니다. (비동기-Event)
smartSerialPort1.ReceiveDetect = SmartX.SmartSerialPort.RECEIVEDETECTTYPE.EVENT_QUEUE;
// 포트 오픈 상태 확인 후 포트를 오픈합니다.
if (smartSerialPort1.IsOpen == false)
{
  smartSerialPort1.Open();
}
```

[STEP-2] 데이터 송신하기

WriteNoneFrame() 메소드를 호출해 None-Frame 형식의 데이터를 전송할 수 있습니다. 비동기-Event 방식의 통 신을 하므로, 데이터 송신 후 OnReadQueueEvent에서 송신에 따른 수신 처리를 하시기 바랍니다.

※ 자세한 내용은 "WriteNoneFrame() 메소드"를 참고하시기 바랍니다.

```
private void SendEvent_NoneFrame(byte[] bSendData)
{
  smartSerialPort1.WriteNoneFrame(bSendData);  // 바이트 배열을 None-Frame으로 전송
}
```

[STEP-3] 데이터 수신하기

데이터 송신 후 응답 데이터 수신 시 OnReadQueueEvent가 발생하며, 이벤트 코드 내에서 ReadQueue() 메소드 를 호출해 응답 데이터를 확인할 수 있습니다. 이후 리턴값을 확인해 데이터가 정상적인지 확인합니다.

※ 자세한 내용은 "ReadQueue() 메소드", "OnReadQueueEvent 이벤트"를 참고하시기 바랍니다

```
private void smartSerialPort1_OnReadQueueEvent()
{
  byte[] bRecvData;  // 데이터 수신을 위한 바이트 배열
  SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus;  // 데이터 수신 상태를 체크할 변수
  // 데이터를 수신하고 상태를 저장한다.
  eStatus = smartSerialPort1.ReadQueue(out bRecvData);
  if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
  {
    // 수신 성공 처리 코드
  }
  else if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.FAILDATA)
```

```
  {
    // 수신 실패 처리 코드
  }
}
```

CASE-4 비동기-Event, Queue, Frame 방식인 경우

비동기-Event 방식의 통신 방식으로 Queue를 사용하며, Frame 타입의 통신 모델입니다.

※ 자세한 내용은 "데이터 구조 및 수신 처리 방식"을 참고하시기 바랍니다.

[STEP-1] 설정 및 Open하기

비동기-Event 방식으로 사용하기 위해 ReceiveDetect 속성을 Event_QUEUE로 설정합니다. FrameSeparationType 속성으로 프레임 타입을 None-Frame이 아닌 값으로 설정합니다. 설정값에 따라 STXCode, EtxCode 속성값을 설정하시기 바랍니다. 본 가이드는 STXANDETX로 설정하였습니다. 모든 속성 설정이 완료되면 Open() 메소드를 호출해 Port를 Open할 수 있습니다.

※ 자세한 통신 관련 속성은 아래 코드에서 사용한 속성 및 메소드를 참고하시기 바랍니다

```
// 기본적인 통신 관련 속성 설정 코드는 생략되었으며, 공통 속성 설정하기를 참고하시기 바랍니다.
// 프레임 타입을 설정합니다.
smartSerialPort1.FrameSeparationType = SmartX.SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX;
smartSerialPort1.STXCode = 0x02; smartSerialPort1.ETXCode = 0x03;
// 데이터 수신 방식을 설정합니다. (비동기-Event)
smartSerialPort1.ReceiveDetect = SmartX.SmartSerialPort.RECEIVEDETECTTYPE.EVENT_QUEUE;
// 포트 오픈 상태 확인 후 포트를 오픈합니다.
if (smartSerialPort1.IsOpen == false)
{
  smartSerialPort1.Open();
}
```

[STEP-2] 데이터 송신하기

WriteFrame() 메소드를 호출해 Frame 형식의 데이터를 전송할 수 있습니다. 비동기-Event 방식의 통신을 하므로, 데이터 송신 후 OnReadQueueEvent에서 송신에 따른 수신 처리를 하시기 바랍니다.

※ 자세한 내용은 "WriteFrame() 메소드"를 참고하시기 바랍니다.

```
private void SendEvent_Frame(byte[] bSendData)
{
  smartSerialPort1.WriteFrame(bSendData);  // 바이트 배열을 Frame으로 전송
}
```

[STEP-3] 데이터 수신하기

데이터 송신 후 응답 데이터 수신 시 OnReadQueueEvent가 발생하며, 이벤트 코드내에서 ReadQueue() 메소드를 호출해 응답 데이터를 확인할 수 있습니다. 이후 리턴값을 확인해 데이터가 정상적인지 확인합니다.

※ 자세한 내용은 "ReadQueue() 메소드", "OnReadQueueEvent 이벤트"를 참고하시기 바랍니다.

```
private void smartSerialPort1_OnReadQueueEvent()
{
  byte[] bRecvData;  // 데이터 수신을 위한 바이트 배열
  SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus;  // 데이터 수신 상태를 체크할 변수
  eStatus = smartSerialPort1.ReadQueue(out bRecvData);  // 데이터를 수신하고 상태를 저장
  if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
```

```
    {
      // 수신 성공 처리 코드
    }
    else if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.FAILDATA)
    {
      // 수신 실패 처리 코드
    }
  }
```

6) SmartSerialPort 인터페이스 설명

SmartSerialPort Component Interface		
속성		
Baud_rate : SmartSerialPort.BAUDRATE	ErrorCheckCodeType : SmartSerialPort.ERRORCHECKCODETYPES	ErrorCheckMode : SmartSerialPort.ERRORCHECK
ErrorCode_Location : SmartSerialPort.ERRORCODELOCATION	ETXCode : byte	ETXCodes : byte[]
FrameBufferSize : uint	FrameSeparationType : SmartSerialPort.FRAMESEPARATIONTYPES	HeadErrorCodeOffset : int
IsOpen : bool	IsReadStart : bool	PortNo : SmartSerialPort.COMPORTNO
RawSerialPort : SerialPort	ReadTimeOut : int	ReceiveDetect : SmartSerialPort.RECEIVEDETECTTYPE
ReceiveFrameDebugMode : bool	ReceiveIndicator : SmartSerialPort.RECEIVE_INDICATOR	RS485SoftwaeDetection : bool
STXCode : byte	STXCodes : byte[]	TailErrorCodeOffset : int
메소드		
Close() : void	ConvertASCIIByteToString(byte[] byteData) : static string	ConvertBytesToHexDecString(byte[] byteBinary, string strDelimiter) : static string
ConvertHexDecStringToBytes(string strHexadecimal, string strDelimiter) : static byte[]	ConvertKSC5601ByteToUnicodeString (byte[] byteKSC5601) : static string	ConvertUnicodeByteToString(byte[] byteBinary, string strDelimiter) : static string
ConvertUnicodeStringToKS5601Byte (string strUnicode): static byte[]	GetCheckSum8Gen(byte[] rawData) : byte, byte[] (+3개 오버로드)	GetCheckSum16Gen(byte[] rawData) : ushort, byte[] (+3개 오버로드)
GetCRC16Gen(byte[] rawData) : ushort, byte[] (+3개 오버로드)	GetCRC32Gen(byte[] rawData): uint, byte[] (+3개 오버로드)	Open(): void (+1개 오버로드)
ReadFrame(out byte[] bytesRead) : SmartSerialPort.FRAMEDATAREADSTATUS (+3개 오버로드)	ReceiveIndicatorStart() : void	ReceiveIndicatorStop() : void
ReadNoneFrame(out byte[] bytesRead) : SmartSerialPort.FRAMEDATAREADSTATUS (+3개 오버로드)	ReadQueue(out byte[] bytesRead) : SmartSerialPort.ReceiveDataHandler (+1개 오버로드)	WriteFrame(byte[] bytesWrite) : void (+1개 오버로드)
WriteNoneFrame(byte[] bytesWrite) : void (+1개 오버로드)	SetWriteOnlySTX(byte writeSTXCode) : void (+1개 오버로드)	SetWriteOnlyETX(byte writeETXCode) : void (+1개 오버로드)
이벤트		
OnPortError : SmartSerialPort.PortErrorHandler	OnReadQueueEvent : SmartSerialPort.ReceiveDataHandler	

프로퍼티(속성) **Baud_Rate**

통신 속도(Baud Rate)를 설정 합니다.

주의 자사 예제를 갖고 테스트 시 보드레이트 최대 속도를 256000으로 설정하면 에러가 발생합니다. 최대 속도를 115200으로 사용해주시기 바랍니다. Tech Note 49. [C#, VB.NET]시리얼 통신 예제도 동일하게 적용됩니다.

- SmartSerialPort.BAUDRATE.CBR_110 : 110bps
- SmartSerialPort.BAUDRATE.CBR_300 : 300bps
- SmartSerialPort.BAUDRATE.CBR_600 : 600bps
- SmartSerialPort.BAUDRATE.CBR_1200 : 1200bps
- SmartSerialPort.BAUDRATE.CBR_2400 : 2400bps
- SmartSerialPort.BAUDRATE.CBR_4800 : 4800bps
- SmartSerialPort.BAUDRATE.CBR_9600 : 9600bps
- SmartSerialPort.BAUDRATE.CBR_14400 : 14400bps
- SmartSerialPort.BAUDRATE.CBR_19200 : 19200bps
- SmartSerialPort.BAUDRATE.CBR_38400 : 38400bps
- SmartSerialPort.BAUDRATE.CBR_56000 : 56000bps
- SmartSerialPort.BAUDRATE.CBR_57600 : 57600bps
- SmartSerialPort.BAUDRATE.CBR_115200 : 115200bps

C# 사용법

```
smartSerialPort1.Baud_Rate = SmartSerialPort.BAUDRATE.CBR_19200;  // 19200bps으로 설정함
```

VB 사용법

```
smartSerialPort1.Baud_Rate = SmartSerialPort.BAUDRATE.CBR_19200
```

프로퍼티(속성) ErrorCheckCodeType

ErrorCheckCode(CheckSum8/16, CRC16/32)의 적용 시 표현되는 형식을 설정합니다.

- SmartSerialPort.ERRORCHECKCODETYPES.ASCIICODE : ASCII로 표현
- SmartSerialPort.ERRORCHECKCODETYPES.BINARY : Binary로 표현
- SmartSerialPort.ERRORCHECKCODETYPES.UNICODE : Unicode로 표현

[표1] 전송 데이터

Index	0	1	2	3	4
데이터	0x41	0x42	0x43	0x44	0x45

[표2] ErrorCheckCodeType 속성값에 따른 ErrorCode ([표1] 데이터 기준이며 16진수 표시)

ErrorCheckMode	ERRORCHECKCODETYPES		
	BINARY	ASCIICODE	UNICODE
CheckSum8	4F	34-46	34-00-46-00
CheckSum16	01-4F	30-31-34-46	30-00-31-00-34-00-46-00
CRC16	0F-50	30-46-35-30	30-00-46-00-35-00-30-00
CRC32	72-D3-1A-D5	37-32-44-33-31-41-44-35	37-00-32-00-44-00-33-00-31-00-41-00-44-00-35-00

C# 사용법

```
// ErrorCheckCode를 ASCII로 표현
smartSerialPort1.ErrorCheckCodeType = SmartSerialPort.ERRORCHECKCODETYPES.ASCIICODE;
```

VB 사용법

```
smartSerialPort1.ErrorCheckCodeType = SmartSerialPort.ERRORCHECKCODETYPES.ASCIICODE
```

프로퍼티(속성) ErrorCheckMode

데이터를 송/수신 시 사용할 ErrorCheckMode를 설정합니다. SmartSerialPort는 내부에 Error Check를 자동으로 수행하도록 구현되어 있어 사용자가 별도로 에러 코드를 계산할 필요가 없습니다.

- SmartSerialPort.ERRORCHECK.NONE : ErrorCheck 사용 안 함

- SmartSerialPort.ERRORCHECK.CHECKSUM8 : CheckSum 8Bit 사용
- SmartSerialPort.ERRORCHECK.CHECKSUM16 : CheckSum 16Bit 사용
- SmartSerialPort.ERRORCHECK.CRC16 : CRC 16Bit 사용
- SmartSerialPort.ERRORCHECK.CRC32 : CRC 32Bit 사용

C# 사용법

```
smartSerialPort1.ErrorCheckMode = SmartSerialPort.ERRORCHECK.NONE;  // Error Check 하지 않음
```

VB 사용법

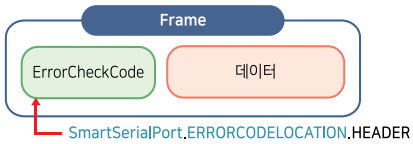
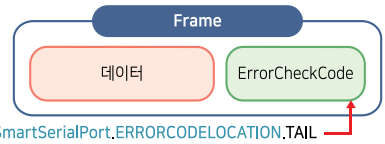
```
smartSerialPort1.ErrorCheckMode = SmartSerialPort.ERRORCHECK.NONE
```

프로퍼티(속성) ErrorCode_Location

ErrorCheckMode가 NONE이 아닌 경우, ErrorCheckCode의 위치를 설정합니다.

- SmartSerialPort.ERRORCODELOCATION.HEADER : ErrorCheckCode의 위치를 데이터 앞으로 설정
- SmartSerialPort.ERRORCODELOCATION.TAIL : ErrorCheckCode의 위치를 데이터 뒤로 설정

[표] ErrorCode_Location 속성값에 따른 ErrorCheckCode 위치

HEADER	TAIL
	

C# 사용법

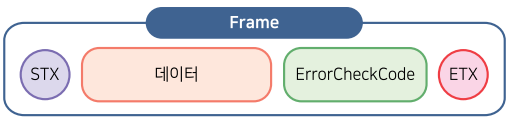
```
// ErrorCheckCode를 앞쪽에 위치시킴
smartSerialPort1.ErrorCode_Location = SmartSerialPort.ERRORCODELOCATION.HEADER;
```

VB 사용법

```
smartSerialPort1.ErrorCode_Location = SmartSerialPort.ERRORCODELOCATION.HEADER
```

프로퍼티(속성) FrameSeperationType

통신 Frame 구조(형태)를 설정합니다.



참고 "데이터 구조 및 지원 형태"를 참고하시기 바랍니다.

- SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX : STX와 ETX를 사용
- SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX_READTIMEOUT : ReadTimeOut을 통해 데이터를 구분하며, STX와 ETX를 사용
- SmartSerialPort.FRAMESEPARATIONTYPES.STXONLY : STX만 사용
- SmartSerialPort.FRAMESEPARATIONTYPES.STXONLY_READTIMEOUT : ReadTimeOut을 통해 데이터를 구분하며, STX만 사용
- SmartSerialPort.FRAMESEPARATIONTYPES.ETXONLY : ETX만 사용
- SmartSerialPort.FRAMESEPARATIONTYPES.ETXONLY_READTIMEOUT : ReadTimeOut을 통해 데이터를 구분하며, ETX만 사용
- SmartSerialPort.FRAMESEPARATIONTYPES.NONEFRAME_READTIMEOUT : ReadTimeOut 또는 FrameBufferSize를 통해 데이터를 구분하며, STX와 ETX를 사용하지 않음

C# 사용법

```
// Frame 구분 방식을 STX+ETX로 설정
smartSerialPort1.FrameSeparationType = SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX;
```

VB 사용법

```
smartSerialPort1.FrameSeparationType = SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX
```

프로퍼티(속성) ETXCode, STXCode

크기가 1Byte인 STX 코드 또는 ETX 코드를 설정합니다.

주의 Port Open 후 설정값을 변경하지 마시기 바랍니다.

- STXCode : 1Byte 크기의 송/수신 데이터의 STX(시작 구분) 코드를 설정(기본값 : 0x02)
- ETXCode : 1Byte 크기의 송/수신 데이터의 ETX(종료 구분) 코드를 설정(기본값 : 0x03)
- byte : STX 또는 ETX 코드

[표] FrameSeparaionType에 따른 사용 속성

FrameSeparationType	사용 속성
STXANDETX	STXCode, ETXCode
STXANDETX_READTIMEOUT	STXCode, ETXCode, ReadTimeOut
STXONLY (1Byte)	STXCode
STXONLY_READTIMEOUT (1Byte)	STXCode, ReadTimeOut
ETXONLY (1Byte)	ETXCode
ETXONLY_READTIMEOUT (1Byte)	ETXCode, ReadTimeOut

C# 사용법

```
// 시작 및 종료 구분 코드 설정
smartSerialPort1.STXCode = 0x02; smartSerialPort1.ETXCode = 0x03;
```

VB 사용법

```
smartSerialPort1.STXCode = &H2 : smartSerialPort1.ETXCode = &H3
```

프로퍼티(속성) ETXCodes, STXCodes

- ETXCodes : FrameSeparationType이 ETXONLY 또는 ETXONLY_READTIMEOUT이고(ETX 만을 사용), ETX를 2Byte로 사용하는 경우 설정합니다.

중요 FrameSeparationType을 반드시 ETXONLY 또는 ETXONLY_READTIMEOUT으로 설정하고 사용해야합니다.

- STXCodes : FrameSeparationType이 STXONLY 또는 STXONLY_READTIMEOUT이고(STX 만을 사용), STX를 2Byte로 사용하는 경우 설정합니다.

중요 FrameSeparationType을 반드시 STXONLY 또는 STXONLY_READTIMEOUT으로 설정하고 사용해야합니다.

참고 “데이터 구조 및 지원 형태”를 참고하시기 바랍니다

[표] FrameSeparaionType에 따른 사용 속성

FrameSeparationType	사용 속성
STXONLY (2Byte)	STXCodes
STXONLY_READTIMEOUT (2Byte)	STXCodes, ReadTimeOut
ETXONLY (2Byte)	ETXCodes
ETXONLY_READTIMEOUT (2Byte)	ETXCodes, ReadTimeOut

- byte[] : 2Byte 배열이며, 중복된 패턴의 멤버는 지원되지 않습니다.

C# 사용법

```
// etxstxCods를 멀티 바이트(2Byte)로 설정하고 ETX 또는 STX 값을 CR, LF로 각각 설정
byte[] etxstxCods = new byte[2];
etxstxCods[0] = (byte)'\r'; etxstxCods[1] = (byte)'\n';
smartSerialPort1.ETXCodes = etxstxCods;  // smartSerialPort1.STXCodes = etxstxCods;
```

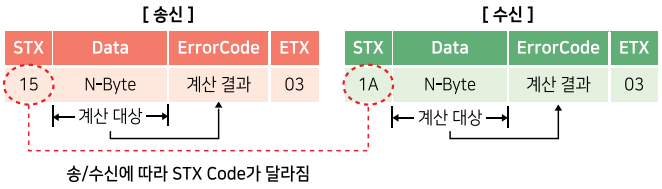
VB 사용법

```
Dim etxstxCods As Byte() = New Byte(1) {}
etxstxCods(0) = CByte(Val(ControlChars.Cr)) : etxstxCods(1) = CByte(Val(ControlChars.Lf))
smartSerialPort1.ETXCodes = etxstxCods  ' smartSerialPort1.STXCodes = etxstxCods;
```

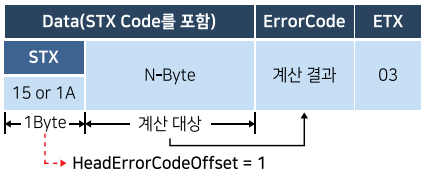
프로퍼티(속성) HeadErrorCodeOffset, TailErrorCodeOffset

SmartSerialPort는 송/수신 시 Frame 데이터의 오류를 검증하기 위하여 CheckSum, CRC 등의 오류 코드를 적용할 수 있는 기능을 포함하고 있습니다. 일반적으로 오류 코드는 데이터 부분만 계산하게 되어 있습니다. 통신 중 STX 또는 ETX의 크기가 가변되는 경우 데이터의 실질적인 길이가 달라져 CheckSum이나 CRC 값에 차이가 발생합니다. 이때 STX의 경우 HeadErrorCodeOffset, TailErrorCodeOffset 속성을 사용하면 실제 데이터의 길이를 설정할 수 있어 이러한 현상을 해결할 수 있습니다.

IEC-Series & 장치 간 송/수신 Protocol 구성



SmartSerialPort에서 처리할 수 있는 Protocol 처리 방법

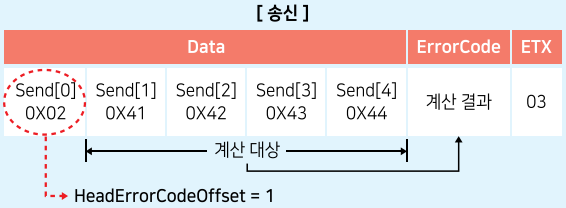


사용법

참고 아래의 "HeadErrorCodeOffset, TailErrorCodeOffset 사용법"을 참고하시기 바랍니다.

참고 HeadErrorCodeOffset, TailErrorCodeOffset 사용법

1. 송신일 경우 Data 앞에 STX 코드를 추가하여 함께 처리해야 합니다. (TailErrorCodeOffset 사용 시 Data 뒤에 ETX 코드를 추가)



[C#] 송신부 코드

```
// 데이터에 포함된 STX 코드의 길이가 1BYTE이므로 1로 지정
smartSerialPort1.HeadErrorCodeOffset = 1;
```

```
// 데이터에 포함된 ETX 코드의 길이가 1BYTE이므로 1로 지정
// smartSerialPort1.TailErrorCodeOffset = 1;
byte[] send = new byte[5];
send[0] = 0x02;  // Data에 포함된 STX 코드.TailErrorCodeOffset 사용 시 맨 뒤에 ETX 코드를 추가
// ASCII Code : ABCD
send[1] = 0x41; send[2] = 0x42; send[3] = 0x43; send[4] = 0x44;
smartSerialPort1.WriteFrame(send);  // WriteFrame을 사용하여 데이터를 Send
```

2. 수신일 경우 Data에 STX 코드가 포함되어 있으므로, 앞쪽의 STX 코드의 크기만큼 데이터를 제거하거나 무시하고 처리해야 합니다.

[수신]

Data					ErrorCode	ETX
recv[0] 0X02	recv[1] 0X41	recv[2] 0X42	recv[3] 0X43	recv[4] 0X44	계산 결과	03

수신받은 Byte 배열의 0번째 데이터는 STX 코드이므로 첫번째 데이터부터 Read 합니다.

[C#] 송신부 코드

```
byte[] recv;
string strRecData = " ";
SmartX.SmartSerialPort.FRAMEDATAREADSTATUS eStatus = smartSerialPort1.ReadQueue(out recv);
if (eStatus == SmartX.SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
{
// HeadErrorCodeOffset 사이즈 만큼을 제외한 길이의 데이터(실제 데이터)를 strRecData에 저장
// 즉, 0x02 를 제외한 ABCD를 읽어서 strRecData에 저장
strRecData = Encoding.ASCII.GetString(recv, 1, recv.Length - smartSerialPort1.HeadErrorCodeOffset);
}
```

프로퍼티(속성) FrameBufferSize

한 Frame의 최대 크기를 설정합니다. 만약 데이터의 Frame 크기가 고정이 아니라 가변되는 경우, 통신 시 발생할 수 있는 최대 크기를 설정합니다.

참고 "데이터 구조 및 지원 형태"를 참고하시기 바랍니다

- uint : 한 Frame의 최대 크기 (단위 : Byte, 기본값 : 1024)

C# 사용법

```
smartSerialPort1.FrameBufferSize = 5000;  // 한 Frame에 저장 가능한 최대 크기를 5000Byte로 설정
```

VB 사용법

```
smartSerialPort1.FrameBufferSize = 5000
```

프로퍼티(속성) IsOpen

현재 통신 Port가 Open 되었는지를 확인합니다.

- bool : 통신 Port Open 여부
  - true : Open 됨
  - false : Close 됨

C# 사용법

```
// 통신 Port가 열려있는 경우 Port를 닫음
if (smartSerialPort1.IsOpen == true)
```

```
{
  smartSerialPort1.Close();
}
```

VB 사용법

```
If smartSerialPort1.IsOpen = True Then
  smartSerialPort1.Close()
End If
```

프로퍼티(속성) IsReadStart

ReceiveDetect 속성이 EVENT_QUEUE일 때 SmartSerialPort가 Serial Data 감지를 시작했는지 확인합니다.

- bool : Serial Data 감지 시작 여부
  - true : 시작됨
  - false : 중지됨

C# 사용법

```
// SmartSerialPort가 Serial Data 감지를 시작한 경우 포트를 닫음
if (smartSerialPort1.IsReadStart == true)
{
  smartSerialPort1.Close();
}
```

VB 사용법

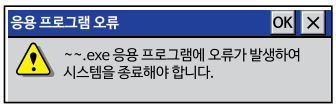
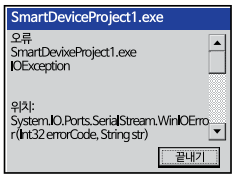
```
If smartSerialPort1.IsReadStart = True Then
  smartSerialPort1.Close()
End If
```

프로퍼티(속성) PortNo

사용할 통신 Port 번호를 설정합니다. IEC-Series 별로 내장된 통신 Port의 개수가 다르며, COM7의 경우 USB FTDI VCP 드라이버를 사용합니다.

주의 USB to Serial 컨버터(COM7) 사용 시 주의사항

USB to Serial 컨버터를 USB에 연결해 COM7로 Port 사용 시, 통신 중 USB 컨버터를 제거하지 마시기 바랍니다. 통신 중에 USB 컨버터를 제거하는 경우 다음과 같은 오류가 발생할 수 있습니다.

에러 유형 1	에러 유형 2
	

[표] IEC-Series별 지원 통신 Port

IEC-Series	COM1_1	COM1	COM2	COM3	COM4	COM7
IEC266	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	미지원	미지원
IEC667	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC667Lite	지원 (RS485)	지원 (5V-TTL)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)

IEC1000	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC1000Lite	지원 (RS485)	지원 (5V-TTL)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
Comport No.	COM1		COM2	COM3	COM4	COM7

- SmartSerialPort.COMPORTNO.COM1 : COM1
- SmartSerialPort.COMPORTNO.COM2 : COM2
- SmartSerialPort.COMPORTNO.COM3 : COM3
- SmartSerialPort.COMPORTNO.COM4 : COM4 (IEC266-Series 미지원)
- SmartSerialPort.COMPORTNO.COM7 : COM7 (IEC266-Series 미지원)

C# 사용법

```
smartSerialPort1.PortNo = SmartSerialPort.COMPORTNO.COM3;  // 통신 Port를 COM3로 설정
```

VB 사용법

```
smartSerialPort1.PortNo = SmartSerialPort.COMPORTNO.COM3
```

프로퍼티(속성) ReadTimeOut

읽기 작업의 타임아웃 시간 설정 및 None-Frame 구조에서 Data의 구분을 하기 위하여 통신 환경에 적절한 값으로 설정합니다.

- int : 타임아웃 시간 (단위 : ms)

참고 "데이터 구조 및 지원 형태"를 참고하시기 바랍니다.

중요 FrameSeparationType 속성값이 XXX_READTIMEOUT인 경우 반드시 설정하시기 바랍니다.

C# 사용법

```
smartSerialPort1.ReadTimeOut = 500;  // ReadTimeOut 시간을 500ms로 설정
```

VB 사용법

```
smartSerialPort1.ReadTimeOut = 500
```

프로퍼티(속성) ReceiveDetect

수신 방식을 비동기-Event 방식으로 할지 동기-Polling 방식으로 할지 설정합니다.

참고 "통신 방식 및 데이터 구조에 따른 송/수신 메소드", "동기-Polling 처리 방식", "비동기-Event 처리 방식" 내용을 참고하시기 바랍니다.

- SmartSerialPort.RECEIVEDETECTTYPE.EVENT_QUEUE : 비동기-Event 방식으로 데이터를 수신. 데이터 수신 시 수신받은 데이터는 내부 Queue에 추가되고 OnReadQueue 이벤트가 발생됩니다. Frame, None-Frame과 관계없이 이벤트 코드에서 ReadQueue() 메소드를 호출하여 데이터 처리 가능
- SmartSerialPort.RECEIVEDETECTTYPE.POLLING_NONEQUEUE : 동기-Polling 방식으로 데이터를 수신. 데이터 수신 시 ReadFrame() 또는 ReadNoneFrame() 메소드를 호출해야 데이터 처리 가능

C# 사용법

```
// 수신 방식을 비동기-Event 방식으로 설정합니다.
smartSerialPort1.ReceiveDetect = SmartSerialPort.RECEIVEDETECTTYPE.EVENT_QUEUE;
```

VB 사용법

```
smartSerialPort1.ReceiveDetect = SmartSerialPort.RECEIVEDETECTTYPE.EVENT_QUEUE
```

프로퍼티(속성) ReceiveFrameDebugMode

데이터 수신 시 데이터만 출력할 것인지 또는 다른 요소(STX 코드, ETX 코드, ErrorCheck 코드)도 포함한 모든 데이터를 출력할 것인지 설정합니다.

- bool : 수신받은 데이터 전체 출력 여부
  - true : 수신받은 모든 데이터를 출력
  - false : 수신받은 데이터에서 STX 코드, ETX 코드, ErrorCheck 코드를 제외하고 출력(기본값)

[표1] 수신받은 모든 데이터

STX	ErrorCheck (CheckSum8-ASCII)	DATA	ETX
02	34-46	41-42-43-44	03

[표2] ReceiveFrameDebugMode 속성값에 따른 수신 데이터

ReceiveFrameDebugMode	출력 데이터
true	02-34-46-41-42-43-44-03
false	41-42-43-44

C# 사용법

```
smartSerialPort1.ReceiveFrameDebugMode = true;  // 수신받은 모든 데이터를 출력하도록 설정
```

VB 사용법

```
smartSerialPort1.ReceiveFrameDebugMode = true
```

프로퍼티(속성) ReceiveIndicator

현재 Open된 Port의 데이터 수신 상태를 확인합니다. 시리얼 통신에 문제가 발생하는 경우 ReceiveIndicator 속성을 이용해 Open된 Port에 실제 데이터 수신 여부를 확인하여 문제 발생 원인을 파악할 수 있습니다.

- SmartSerialPort.RECEIVE_INDICATOR.Empty : 수신된 데이터가 없음
- SmartSerialPort.RECEIVE_INDICATOR.Onebyte : 1Byte 데이터가 수신됨
- SmartSerialPort.RECEIVE_INDICATOR.Multiplebytes : 2Byte 이상의 데이터가 수신됨

참고 1Byte의 데이터가 연속적으로 들어오는 경우 Onebyte가 아닌 Multiplebytes로 체크됩니다.

C# 사용법

```
private void timer1_Tick(object sender, EventArgs e)
{
  // 타이머를 이용해 주기적으로 Port에 수신되는 데이터를 감지
  switch (smartSerialPort1.ReceiveIndicator)
  {
    case SmartSerialPort.RECEIVE_INDICATOR.Empty:
      label2.Text = "Empty";  // Port에 수신된 데이터가 없는 경우
      timer1.Interval = 200;    // 데이터 체크를 위해 타이머 인터벌을 줄임
      Status_Off();
      break;
    case SmartSerialPort.RECEIVE_INDICATOR.Onebyte:
      label2.Text = "Onebyte";  // Port에 수신된 데이터가 1Byte인 경우
      timer1.Interval = 200;  // 데이터 체크를 위해 타이머 인터벌을 줄임
      Status_Toggle();
      break;
    case SmartSerialPort.RECEIVE_INDICATOR.Multiplebytes:
      label2.Text = "Multiplebytes";  // Port에 수신된 데이터가 2Byte 이상인 경우
      timer1.Interval = 1000;  // 다수의 데이터가 수신되어 다음 데이터 체크를 늦춤
      // 다수의 데이터가 수신됨을 표현
      for (int i = 0; i < 6; i++)
```

```
      {
        Status_Toggle();
        Application.DoEvents();
        System.Threading.Thread.Sleep(50);
      }
      break;
  }
}
```

VB 사용법

```
Private Sub timer1_Tick(ByVal sender As System.Object, ByVal e As System.ComponentModel.EventArgs)
  Select case smartSerialPort1.ReceiveIndicator
    case SmartSerialPort.RECEIVE_INDICATOR.Empty
       label2.Text = " Empty "
       Status_Off()
    case SmartSerialPort.RECEIVE_INDICATOR.Onebyte
       label2.Text = " Onebyte "
       timer1.Interval = 200
       Status_Toggle()
    case SmartSerialPort.RECEIVE_INDICATOR.Multiplebytes
       label2.Text = " Multiplebytes "
       timer1.Interval = 1000
       For i As Integer = 0 To 5
         Status_Toggle()
         Application.DoEvents()
         System.Threading.Thread.Sleep(50)
       Next
  End Select
End Sub
```

프로퍼티(속성) RS485SoftwareDetection

RS485 송/수신 제어 방식이 S/W 방식인 경우, RS485SoftwareDetection 속성값을 true로 설정하면 첫 데이터 수신 시 에러가 발생하는 현상을 방지할 수 있습니다.

참고 "RS485 통신 시 주의사항" 내용을 참고하시기 바랍니다.

- bool : RS485SoftwareDetection 기능 활성화 여부
  - true : 활성화
  - false : 비활성화

C# 사용법

```
smartSerialPort1.RS485SoftwareDetection = true;  // RS485SoftwareDetection 기능 활성화
```

VB 사용법

```
smartSerialPort1.RS485SoftwareDetection = true
```

프로퍼티(속성) RawSerialPort

.Net Compact Framework의 SerialPort 컴포넌트 인터페이스를 참조할 수 있습니다. SmartSerialPort는 .NET Compact Framework의 SerialPort 컴포넌트에서 파생되었습니다. 그러므로 SerialPort의 인터페이스 또한 참조하실 수 있으며, SmartSerialPort에서 지원되지 않은 속성값들은 RawSerialPort를 통해서 설정하시면 됩니다.

C# 사용법

```
smartSerialPort1.RawSerialPort.StopBits = System.IO.Ports.StopBits.One;  // 정지 비트 1Bit 설정
smartSerialPort1.RawSerialPort.Parity = System.IO.Ports.Parity.None;  // 패리티 없음 설정
```

VB 사용법

```
smartSerialPort1.RawSerialPort.StopBits = System.IO.Ports.StopBits.One
smartSerialPort1.RawSerialPort.Parity = System.IO.Ports.Parity.None
```

메소드(함수) Open

Port를 Open합니다. 각 속성값으로 설정 후 Open하거나, 간단한 속성값(BaudRate, Port, ReadTimeOut)을 설정하며 Open할 수 있습니다.

주의 1. Port Open 후 SmartSerialPort의 설정값을 변경하지 마십시오.
2. Port Open 후 프로그램 종료 시 반드시 Port를 Close 하시기 바랍니다.

참고 "BaudRate, Port, ReadTimeOut 속성" 내용을 참고하시기 바랍니다.

◉ `void Open()`
◉ `void Open(SmartSerialPort.BAUDRATE baudRate, SmartSerialPort.COMPORTNO portNo, int iReadTimeout)`

[인자]

- `SmartSerialPort.BAUDRATE baudRate` : 통신 속도를 설정
- `SmartSerialPort.COMPORTNO portNo` : 통신 Port를 설정
- `int iReadTimeout` : 읽기 작업의 타임아웃 시간 설정

C# 사용법

```
smartSerialPort1.Open();   // 속성값을 변경하지 않고 Port를 Open합니다.
// BaudRate : 19200, Port : COM3, ReadTimeOut : 1000
smartSerialPort1.Open(SmartSerialPort.BAUDRATE.CBR_19200, SmartSerialPort.COMPORTNO.COM3, 1000);
```

VB 사용법

```
smartSerialPort1.Open()
smartSerialPort1.Open(SmartSerialPort.BAUDRATE.CBR_19200, SmartSerialPort.COMPORTNO.COM3, 1000)
```

메소드(함수) Close

Port를 닫습니다.

주의 Port Open 후 프로그램 종료 시 반드시 Port를 Close 하시기 바랍니다.

◉ `void Close()`

C# 사용법

```
// smartSerialPort1이 Open된 경우 Port를 닫는다.
if (smartSerialPort1.IsOpen == true)
{
  smartSerialPort1.Close();
}
```

VB 사용법

```
If smartSerialPort1.IsOpen = True Then
  smartSerialPort1.Close()
End If
```

메소드(함수) WriteFrame, WriteNoneFrame

• WriteFrame() : Frame 형식의 데이터를 송신합니다.
• WriteNoneFrame() : None-Frame 형식의 데이터를 송신합니다.

◉ void WriteFrame(byte[] bytesWrite)
◉ void WriteFrame(string strWrite, SmartSerialPort.CODETYPES eCode)
◉ void WriteNoneFrame(byte[] bytesWrite)
◉ void WriteNoneFrame(string strWrite, SmartSerialPort.CODETYPES eCode)

[인자]

• byte[] bytesWrite : 전송할 바이트 배열 데이터
• string strWrite : 전송할 string 데이터
• SmartSerialPort.CODETYPES eCode : string 데이터의 전송 타입을 설정 (ASCII, Unicode)

C# 사용법

```
byte[] send = new byte[3];
// ASCII Code : ABC
send[0] = 0x41; send[1] = 0x42; send[2] = 0x43;
smartSerialPort1.WriteFrame(send);
// 바이트 배열 데이터를 Frame 형식으로 전송
// ASCIICODE 타입의 string 데이터를 Frame 형식으로 전송
smartSerialPort1.WriteFrame( " ABC " , SmartSerialPort.CODETYPES.ASCIICODE);
smartSerialPort1.WriteNoneFrame(send);
// 바이트 배열 데이터를 None-Frame 형식으로 전송
// ASCIICODE 타입의 string 데이터를 None-Frame 형식으로 전송
smartSerialPort1.WriteNoneFrame( " ABC " , SmartSerialPort.CODETYPES.ASCIICODE);
```

VB 사용법

```
Dim send As Byte() = New Byte(2) {}
send[0] = &H41 : send[1] = &H42 : send[2] = &H43
smartSerialPort1.WriteFrame(send);
smartSerialPort1.WriteNoneFrame( " ABC " , SmartSerialPort.CODETYPES.ASCIICODE)
```

메소드(함수) ReadFrame, ReadNoneFrame

• ReadFrame() : 데이터가 Frame 형식이고 ReceiveDetect 속성값이 POLLING_NONEQUEUE(동기-Polling 방식)인 경우 데이터를 수신합니다.
• ReadNoneFrame() : 데이터가 None-Frame 형식이고 ReceiveDetect 속성값이 POLLING_NONEQUEUE(동기-Polling 방식)인 경우 데이터를 수신합니다.

참고 "동기-Polling 처리 방식" 내용을 참고하시기 바랍니다.

◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadFrame(out byte[] bytesRead)
◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadFrame(out string stringRead)
◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadFrame(out byte[] bytesRead, int iReadTimeout)
◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadNoneFrame(out byte[] bytesRead)
◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadNoneFrame(out string stringRead)
◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadNoneFrame(out byte[] bytesRead, int iReadTimeout)

[인자]

• out byte[] bytesRead : 데이터를 수신받을 바이트 배열 변수
• out string stringRead : 데이터를 수신받을 string 변수
• int iReadTimeout : 읽기 작업의 타임아웃 시간 설정

[리턴값]

- SmartSerialPort.FRAMEDATAREADSTATUS.EMPTY : 수신된 데이터가 없음
- SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA : 정상적인 데이터가 수신됨
- SmartSerialPort.FRAMEDATAREADSTATUS.FAILDATA : 비정상적인 데이터가 수신됨

C# 사용법

```
byte[] readByte;  // 데이터를 수신받을 바이트 배열
SmartSerialPort.FRAMEDATAREADSTATUS eStatus;  // 수신된 데이터의 상태를 저장할 변수
eStatus = smartSerialPort1.ReadFrame(out readByte);  // Frame 형식의 데이터를 수신받는다.
eStatus = smartSerialPort1.ReadNoneFrame(out readByte);  // None-Frame 형식의 데이터를 수신받는다.
```

VB 사용법

```
Dim readByte As Byte(){}
Dim eStatus As SmartSerialPort.FRAMEDATAREADSTATUS
eStatus = smartSerialPort1.ReadFrame(readByte)
eStatus = smartSerialPort1.ReadNoneFrame(readByte)
```

메소드(함수) ReadQueue

Frame, None-Frame 형식의 데이터를 모두 지원하며, ReceiveDetect 속성값이 EVENT_QUEUE(비동기-Event 방식)인 경우 데이터를 수신합니다. 데이터 수신 시 발생하는 OnReadQueue 이벤트 코드 내에서 호출하여 사용합니다.

참고 "비동기-Event 처리 방식" 내용을 참고하시기 바랍니다.

◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadQueue(out byte[] bytesRead)
◉ SmartSerialPort.FRAMEDATAREADSTATUS ReadQueue(out string stringRead)

[인자]

- out byte[] bytesRead : 데이터를 수신받을 바이트 배열 변수
- out string stringRead : 데이터를 수신받을 string 변수

[리턴값]

- SmartSerialPort.FRAMEDATAREADSTATUS.EMPTY : 수신된 데이터가 없음
- SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA : 정상적인 데이터가 수신됨
- SmartSerialPort.FRAMEDATAREADSTATUS.FAILDATA : 비정상적인 데이터가 수신됨

중요 ReadQueue() 메소드 호출 시 주의사항

OnReadQueueEvent 이벤트는 데이터의 정상/비정상 유무와 상관없이 데이터 수신 시 발생하므로, 반드시 ReadQueue() 메소드의 리턴값을 확인하여 데이터의 정상/비정상 유무를 확인하시기 바랍니다.

C# 사용법

```
private void smartSerialPort1_OnReadQueueEvent()
{
  byte[] readByte;
  // 데이터를 수신받음
  SmartSerialPort.FRAMEDATAREADSTATUS eStatus = smartSerialPort1.ReadQueue(out readByte);
  if (eStatus == SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA)
  {
    // 수신 성공에 따른 처리 코드 작성
  }
}
```

VB 사용법

```
Private Sub smartSerialPort1_OnReadQueueEvent() Handles smartSerialPort1.OnReadQueueEvent
  Dim readByte As Byte()
  Dim eStatus As SmartSerialPort.FRAMEDATAREADSTATUS = smartSerialPort1.ReadQueue(readByte)
  If eStatus = SmartSerialPort.FRAMEDATAREADSTATUS.VALIDDATA Then
```

```
  End If
End Sub
```

메소드(함수) ReceiveIndicatorStart

Serial Port 데이터 수신 감지를 시작합니다.

◉ void ReceiveIndicatorStart()

C# 사용법

```
smartSerialPort1.ReceiveIndicatorStart();  // 데이터 수신 감지 시작
```

VB 사용법

```
smartSerialPort1.ReceiveIndicatorStart()
```

메소드(함수) ReceiveIndicatorStop

Serial Port 데이터 수신 감지를 중지합니다.

◉ void ReceiveIndicatorStop()

C# 사용법

```
smartSerialPort1.ReceiveIndicatorStop();  // 데이터 수신 감지 중지
```

VB 사용법

```
smartSerialPort1.ReceiveIndicatorStop()
```

메소드(함수) SetWriteOnlySTX, SetWriteOnlyETX

• SetWriteOnlySTX() : 송신부와 수신부의 STX 코드가 다른 경우 송신 시에만 사용되는 STX 코드를 별도로 지정합니다.

• SetWriteOnlyETX() : 송신부와 수신부의 ETX 코드가 다른 경우 송신 시에만 사용되는 ETX 코드를 별도로 지정합니다.

중요 별도로 지정한 송신부의 코드와 수신부의 코드는 반드시 일치해야 합니다.

◉ void SetWriteOnlySTX(byte writeSTXCode)
◉ void SetWriteOnlySTX(byte[] writeSTXCodes)
◉ void SetWriteOnlyETX(byte writeETXCode)
◉ void SetWriteOnlyETX(byte[] writeETXCodes)

[인자]

• byte writeSTXCode : 송신 시 별도로 지정할 STX 코드
• byte[] writeSTXCodes : 송신 시 별도로 지정할 2byte의 STX 코드
• byte writeETXCode : 송신 시 별도로 지정할 ETX 코드
• byte[] writeETXCodes : 송신 시 별도로 지정할 2byte의 ETX 코드

C++ 사용법

```
// 프레임 구조가 STXANDETX이고 송/수신부의 ETX가 불일치하는 경우
// 수신부 STX Code : 0x02, ETX Code : 0x04
smartSerialPort1.FrameSeparationType = SmartSerialPort.FRAMESEPARATIONTYPES.STXANDETX;
smartSerialPort1.STXCode = 0x02;
smartSerialPort1.ETXCode = 0x03;
// 송신 시에만 별도로 사용되는 ETX 코드를 0x04로 지정
SmartSerialPort1.SetWriteOnlyETX(0x04);
```

정적 메소드(함수) ConvertASCIIByteToString

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

ASCII 바이트 배열 데이터를 문자열로 변환합니다.

◉ `static string ConvertASCIIByteToString (byte[] byteData)`

[인자]

• `byte[] byteData` : ASCII 바이트 배열 데이터

[리턴값]

• `string` : 변환된 문자열

C# 사용법

```
byte[] testByte = new byte[3];
testByte[0] = 65;
testByte[1] = 66;
testByte[2] = 67;
// ASCII 바이트 배열 데이터를 ASCII 문자열 데이터로 반환. 리턴값 : " ABC "
string strData = SmartSerialPort.ConvertASCIIByteToString(readByte);
```

VB 사용법

```
Dim testByte As Byte() = New Byte(2) {}
testByte(0) = 65 : testByte(1) = 66 : testByte(2) = 67
Dim strData As String = SmartSerialPort.ConvertASCIIByteToString(readByte)
```

정적 메소드(함수) ConvertUnicodeByteToString

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

Unicode 바이트 배열 데이터를 문자열로 변환합니다.

◉ `static string ConvertUnicodeByteToString(byte[] byteData)`

[인자]

• `byte[] byteData` : Unicode 바이트 배열 데이터

[리턴값]

• `string` : 변환된 문자열

C# 사용법

```
byte[] testByte = new byte[6];
testByte[0] = 65; testByte[1] = 0;   testByte[2] = 66;
testByte[3] = 0;  testByte[4] = 67;  testByte[5] = 0;
// 바이트 배열(65:00:66:00:67:00)을 Unicode 문자열 데이터로 반환. 리턴값 : " ABC "
string strData = SmartSerialPort.ConvertUnicodeByteToString(readByte);
```

VB 사용법

```
Dim testByte As Byte() = New Byte(5) {}
testByte(0) = 65 : testByte(1) = 0 : testByte(2) = 66
testByte(3) = 0 : testByte(4) = 67 : testByte(5) = 0
Dim strData As String = SmartSerialPort.ConvertUnicodeByteToString(readByte)
```

정적 메소드(함수) ConvertBytesToHexDecString

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

Hex 바이트 배열 데이터를 16진수 문자열로 변환합니다.

참고 ConvertHexDecStringToBytes() 메소드와 반대의 기능을 합니다

◉ static string ConvertBytesToHexDecString(byte[] byteBinary, string strDelimiter)

[인자]

• byte[] byteBinary : 16진수 형식의 문자열로 변환할 Hex 바이트 배열 데이터

• string strDelimiter : Hex 바이트 배열 데이터를 16진수 문자열로 변환 시 문자 사이에 삽입할 구분자를 설정

[리턴값]

• static string : 변환된 16진수 문자열

C# 사용법

```
byte[] testByte = new byte[3];
testByte[0] = 41; testByte[1] = 42; testByte[2] = 43;
// 리턴값은 문자열 데이터. "41:42:43"
string strData = SmartSerialPort.ConvertBytesToHexDecString(testByte, ":");
```

VB 사용법

```
Dim testByte As Byte() = New Byte(2) {41, 42, 43}
Dim strData As String = SmartSerialPort.ConvertBytesToHexDecString(testByte, ":")
```

정적 메소드(함수) ConvertHexDecStringToBytes

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

16진수 문자열을 Hex 바이트 배열 데이터로 변환합니다.

참고 ConvertBytesToHexDecString() 메소드와 반대의 기능을 합니다.

◉ static byte[] ConvertHexDecStringToBytes(string strHexadecimal, string strDelimiter)

[인자]

• string strHexadecimal : Hex 바이트 배열 데이터로 변환할 10진수 문자열 데이터

• string strDelimiter : Hex 바이트 배열 데이터로 변환 시 데이터를 구분할 기준 문자를 설정

[리턴값]

• static byte[] : 변환된 Hex 바이트 배열 데이터

C# 사용법

```
string strData= "41:42:43";
// 리턴값은 바이트 배열 0x41, 0x42, 0x43
byte[] readByte = SmartSerialPort.ConvertHexDecStringToBytes(strData, ":");
```

VB 사용법

```
Dim strData As String = "41:42:43"
Dim readByte As Byte() = SmartSerialPort.ConvertHexDecStringToBytes(strData, ":")
```

정적 메소드(함수) ConvertKSC5601ByteToUnicodeString

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

KSC5601 바이트 배열 데이터를 문자열로 변환합니다.

참고 ConvertUnicodeStringToKSC5601Byte() 메소드와 반대의 기능을 합니다.

◉ static string ConvertKSC5601ByteToUnicodeString(byte[] byteKSC5601)

[인자]

• byte[] byteData : KSC5601 바이트 배열 데이터

[리턴값]

• string : 변환된 문자열

C# 사용법

```
byte[] readByte = new byte[8];
// 가나다라
readByte[0] = 0xb0; readByte[1] = 0xa1; readByte[2] = 0xb3; readByte[3] = 0xaa;
readByte[4] = 0xb4; readByte[5] = 0xd9; readByte[6] = 0xb6; readByte[7] = 0xf3;
string strData = SmartSerialPort.ConvertKSC5601ByteToUnicodeString(readByte);
```

VB 사용법

```
Dim readByte As Byte() = New Byte(7) {}
readByte(0) = &HB0 : readByte(1) = &HA1 : readByte(2) = &HB3 : readByte(3) = &HAA
readByte(4) = &HB4 : readByte(5) = &HD9 : readByte(6) = &HB6 : readByte(7) = &HF3
Dim strData As String = SmartSerialPort.ConvertKSC5601ByteToUnicodeString(readByte)
```

정적 메소드(함수) ConvertUnicodeStringToKSC5601Byte

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

문자열을 KSC5601 바이트 배열 데이터로 변환합니다.

참고 ConvertKSC5601ByteToUnicodeString() 메소드와 반대의 기능을 합니다.

◉ `static byte[] ConvertUnicodeStringToKSC5601Byte(string strUnicode)`

[인자]

• `string strUnicode` : KSC5601 바이트 배열 데이터로 변환할 문자열

[리턴값]

• `static byte[]` : 변환된 KSC5601 바이트 배열 데이터

C# 사용법

```
string strData = " 가나다라 ";
byte[] readByte = SmartSerialPort.ConvertUnicodeStringToKSC5601Byte(strData);
```

VB 사용법

```
Dim strData As String = " 가나다라 "
Dim readByte As Byte() = SmartSerialPort.ConvertUnicodeStringToKSC5601Byte(strData)
```

정적 메소드(함수) GetCheckSum8Gen, GetCheckSum16Gen, GetCRC16Gen, GetCRC32Gen

GetCheckSum8Gen()/GetCheckSum16Gen()/GetCRC16Gen()/GetCRC32Gen() 각각의 메소드는 ErrorCheckCode의 계산된 값을 리턴합니다.

참고 SmartSerialPort에서 ErrorCode를 사용할 경우 위 함수들이 내부에서 호출되어 송/수신 데이터를 자동으로 검증합니다.

◉ `byte GetCheckSum8Gen(byte[] rawData)`
◉ `byte[] GetCheckSum8Gen(byte[] rawData, SmartSerialPort.CODETYPES eCode)`
◉ `byte GetCheckSum8Gen(byte[] rawData, int iStart, int iLength)`
◉ `byte[] GetCheckSum8Gen(byte[] rawData, int iStart, int iLength, SmartSerialPort.CODETYPES eCode)`
◉ `ushort GetCheckSum16Gen(byte[] rawData)`
◉ `byte[] GetCheckSum16Gen(byte[] rawData, SmartSerialPort.CODETYPES eCode)`
◉ `ushort GetCheckSum16Gen(byte[] rawData, int iStart, int iLength)`
◉ `byte[] GetCheckSum16Gen(byte[] rawData, int iStart, int iLength, SmartSerialPort.CODETYPES eCode)`
◉ `ushort GetCRC16Gen(byte[] rawData)`
◉ `byte[] GetCRC16Gen(byte[] rawData, SmartSerialPort.CODETYPES eCode)`
◉ `ushort GetCRC16Gen(byte[] rawData, int iStart, int iLength)`

◉ byte[] GetCRC16Gen(byte[] rawData, int iStart, int iLength, SmartSerialPort.CODETYPES eCode)
◉ uint GetCRC32Gen(byte[] rawData)
◉ byte[] GetCRC32Gen(byte[] rawData, SmartSerialPort.CODETYPES eCode)
◉ uint GetCRC32Gen(byte[] rawData, int iStart, int iLength)
◉ byte[] GetCRC32Gen(byte[] rawData, int iStart, int iLength, SmartSerialPort.CODETYPES eCode)

[인자]
• byte[] rawData : 바이트 배열 형식의 원본 데이터
• int iStart : 데이터의 시작 지점을 설정
• int iLength : 데이터의 길이를 설정
• SmartSerialPort.CODETYPES eCode : 데이터의 타입이 ASCII 또는 Unicode인지 설정

[리턴값]
– GetCheckSum8Gen() 메소드 리턴값
• byte : 데이터를 CheckSum 8Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터
– GetCheckSum16Gen() 메소드 리턴값
• ushort : 데이터를 CheckSum 16Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터
– GetCRC16Gen() 메소드 리턴값
• ushort : 데이터를 CRC 16Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터
– GetCRC32Gen() 메소드 리턴값
• uint : 데이터를 CRC 32Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터

C# 사용법

```
byte[] testByte = new byte[5];
testByte[0] = 65; testByte[1] = 66; testByte[2] = 67; testByte[3] = 68; testByte[4] = 69;
ushort result = smartSerialPort1.GetCRC16Gen(testByte)
```

VB 사용법

```
Dim testByte(4) As Byte
testByte(0) = 65 : testByte(1) = 66 : testByte(2) = 67 : testByte(3) = 68 : testByte(4) = 69
Dim result As UShort = smartSerialPort1.GetCRC16Gen(testByte)
```

이벤트 OnReadQueueEvent

데이터 수신 방식이 비동기-Event 방식인 경우 Frame 또는 None-Frame 형식의 데이터가 수신되면 내부 수신 Queue에 데이터가 저장된 후 발생되는 Event입니다. 따라서 OnReadQueueEvent가 발생될 경우 ReadQueue() 함수를 호출하여 수신된 데이터를 얻을 수 있습니다.

사용법

참고 "비동기-Event 처리 방식" 내용을 참고하시기 바랍니다.

이벤트 OnPortError

데이터 송/수신 시 수신 측 Port에서 발생하는 에러를 캐치하는 이벤트
◉ OnPortError(SerialError eErrorState)

[인자]
• SerialError.Frame : 하드웨어에서 프레이밍 오류 발생
• SerialError.Overrun : 문자 버퍼 오버런이 발생(수신 데이터 손실)

- SerialError.RXOver : 입력 버퍼 오버플로가 발생(하드웨어 수신 버퍼를 모두 사용함)
- SerialError.RXParity : 하드웨어에서 패리티 오류 발생
- SerialError.TXFull : 응용 프로그램에서 문자를 전송하려고 했지만 출력 버퍼가 꽉 참

C# 사용법

```
private void smartSerialPort1_OnPortError(SerialError eErrorState)
{
  switch (eErrorState)
  {
    case SerialError.Frame:
      lblErr.Text = " HardWare Framing Error ";
      break;
    // 중략
    case SerialError.TXFull:
      lblErr.Text = " Write Buffer was Fulled ";
      break;
    default:
      break;
  }
}
```

VB 사용법

```
Private Sub smartSerialPort1_OnPortError(ByVal eErrorState As SerialError)
  Select Case eErrorState
    Case SerialError.Frame
      lblErr.Text = " HardWare Framing Error "
  End Select
End Sub
```

7) SmartSerialPort 예제 사용하기

SmartSerialPort를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

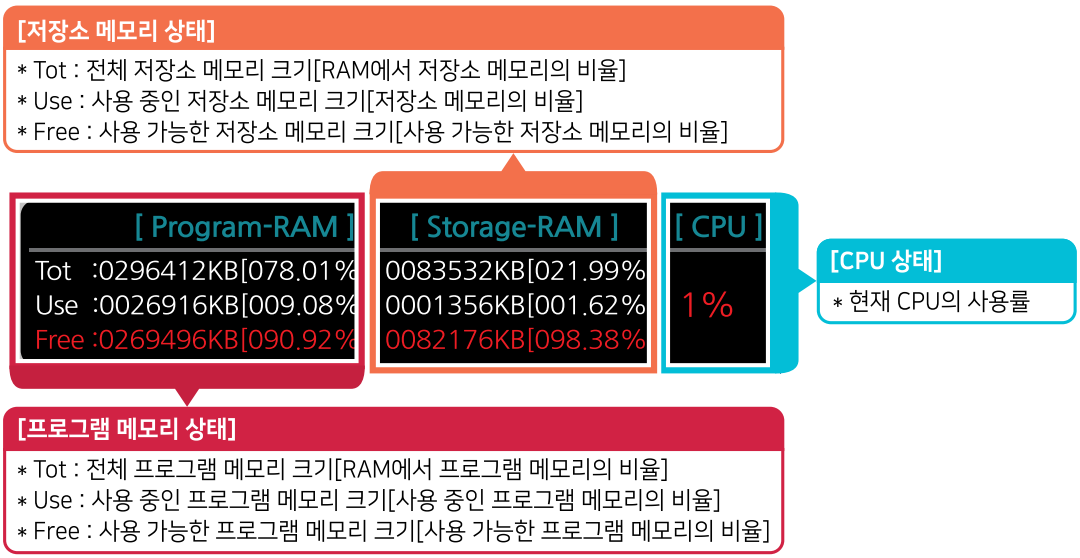
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartSerialPort"

4. SmartMemory

SmartMemory는 IEC-Series 장치의 각종 메모리 시스템의 사용을 모니터링하는 기능을 제공하고 있습니다. 여러 기능 중에서 대표적으로 프로그램 메모리의 사용량을 모니터링하여 프로그램의 메모리 누수(Leak)을 확인할 수 있는 기능과 CPU의 사용량을 확인하여 프로그램 성능을 확인할 수 있도록 지원하고 있습니다. 부가적인 기능으로는 사용 가능한 외부 Storage 등의 영역을 실시간으로 모니터링 및 설정 기능을 제공합니다.

- 주요 모니터링 데이터를 작은 모니터링 창으로 출력 및 자동 갱신 기능
- 모니터링 창 대상 응용 프로그램의 상위(Top Z-Order)로 출력 및 위치 변경 기능
- 개발된 장치 응용 프로그램의 메모리 누수(Leak) 확인 기능
- CPU 사용량 실시간 모니터링
- 프로그램 메모리(Program Memory)와 저장소 메모리(Storage Memory) 영역의 비율 설정 기능
- 외부 메모리(SD Card, USB Memory)의 인식 및 제거에 따른 이벤트 지원
- 현재 사용 중인 시스템 메모리의 상태 값 확인 기능
- 프로그램 메모리 경보 이벤트 기능
- Flash Disk와 SD Card의 사용 상태 확인 기능



참조 프로그램 메모리와 저장소 메모리의 차이

1. 프로그램 메모리

프로그램 메모리는 일반적인 컴퓨터의 RAM과 동일한 역할을 하며, 시스템 및 실행 중인 응용 프로그램의 Heap과 Stack 등이 할당되는 영역입니다. 즉, 프로그램 실행 시 사용되는 메모리 영역을 말합니다.

2. 저장소 메모리

저장소 메모리는 Virtual RAM Disk와 비슷한 개념으로, 응용 프로그램이 저장되는 영역입니다. IECSeries의 Explorer에서 Flash Disk를 제외한 모든 폴더는 저장소 메모리 영역에 해당되어 전원이 OFF될 때 초기화 됩니다.

※ 메모리 관련 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 메모리 관련"을 참조하시기 바랍니다.

1) 프로그래밍 적용 가이드

STEP-1 프로그램 메모리의 누수 확인 및 CPU 사용량 모니터링하기

프로그램 메모리의 누수가 발생하는 경우를 확인하거나 CPU 사용량을 모니터링하기 위해 시스템 메모리 정보창을 출력합니다. 시스템 메모리 정보창의 위치는 별도로 설정 가능합니다.

※ 자세한 내용은 " MemoryStatusInfomationShow 속성" 내용을 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  // 화면 중앙에 메모리 모니터링 창을 출력합니다.
  smartMemory1.MemoryStatusInfomationShow(SmartX.SmartMemory.VIEWPOSITION.CENTER);
}
```

STEP-2 외부저장장치 연결 및 연결 해제 감지하기

외부저장장치(SD Card, USB Memory)의 연결 및 연결 해제를 감지합니다. 사용자는 별도의 코드 처리없이 간편하게 외부저장장치의 확인이 가능합니다.

※ 자세한 내용은 "EvtExternalStorageAttached 이벤트" 내용을 참고하시기 바랍니다.

```
// 외부저장장치의 감지를 시작합니다.
smartMemory1.ExtStorageDetectionStart();
// 외부저장장치 연결 및 연결 해제 시 발생하는 이벤트
private void smartMemory1_EvtExternalStorageAttached(bool bAttached, string strDeviceName)
{
  // 인자값으로 외부저장장치의 상태를 확인
  if (bAttached == true)
  {
    if (strDeviceName == " SD Card " )
    {
      // SD Card가 연결됨(IEC266-Series : ₩₩ SD Card)
    }
    else if (strDeviceName == " 하드 디스크 " )
    {
      // USB 메모리가 연결됨(IEC266-Series : ₩₩ 하드 디스크)
    }
  }
  else
  {
    if (strDeviceName == " SD Card " )
    {
      // SD Card가 연결 해제됨(IEC266-Series : ₩₩ SD Card)
    }
    else if (strDeviceName == " 하드 디스크 " )
    {
      // USB 메모리가 연결 해제됨(IEC266-Series : ₩₩ 하드 디스크)
    }
  }
}
```

2) SmartMemory 인터페이스 설명

SmartMemory Component Interface		
속성		
AutoFreeProMemoryAlarmPercent : double	FD_Drive_FreeBytes : ulong	FD_Drive_UseBytes : ulong
FreeProgramMemoryPercent : double	FreeProgramMemorySize : uint	FreeStorageMemoryPercent : double
FreeStorageMemorySize : uint	ParentWindow : Form	ProgramMemoryPercent : double
ProgramMemorySize : uint	SD_Drive_FreeBytes : ulong	SD_Drive_UseBytes : ulong
StorageMemoryPercent : double	StorageMemorySize : uint	UseProgramMemoryPercent : double
UseProgramMemorySize : uint	UseStorageMemoryPercent : double	UseStorageMemorySize : uint
메소드		
ExtStorageDetectionStart() : void	ExtStorageDetectionStop() : void	GetCPUUsage() : int
MemoryStatusInfomationClose() : void	MemoryStatusInfomationShow() : void (+1개 오버로드)	MemoryStatusInformationUpdate() : void
Release() : void	SetProgramRamPercentSize(double fRamPercentSize) : bool	SetProgramRamSize(uint dwRamByte Size) : bool
StartProMemoryAlarm() : void	StopProMemoryAlarm() : void	
이벤트		
EvtExternalStorageAttached : SmartMemory.DeviceAttachedEvent Handler	OnProgramMemoryAlarmEvent : EventHandler	

프로퍼티(속성) ParentWindow

MemoryStatusInfomationShow() 메소드를 호출하여 출력한 메모리 모니터링 창이 폼에 가려져 보이지 않는 현상을 방지하기 위해 메모리 모니터링 창의 부모 폼을 설정합니다. 즉, 모니터링 창의 Top Z-Order를 설정합니다.

참고 "MemoryStatusInfomationShow() 메소드" 내용을 참고하시기 바랍니다.

- Form : 메모리 모니터링 창의 부모 폼

C# 사용법

```
smartMemory1.ParentWindow = this;  // 메모리 모니터링 창의 부모 폼을 설정
```

VB 사용법

```
smartMemory1.ParentWindow = Me
```

프로퍼티(속성) AutoFreeProMemoryAlarmPercent

StartProMemoryAlarm() 메소드를 사용하여 프로그램 메모리의 사용량 감지를 시작한 경우. OnProgramMemory Alarm 이벤트를 발생시키기 위한 비율을 설정합니다.

참고 "StartProMemoryAlarm() 메소드, OnProgramMemoryAlarmEvent" 내용을 참고하시기 바랍니다.

- double : 이벤트를 발생시킬 프로그램 메모리 영역의 비율(%)

C# 사용법

```
// 사용 가능한 프로그램 메모리가 20% 이하로 떨어지면 이벤트 발생
smartMemory1.AutoFreeProMemoryAlarmPercent = 20;
```

VB 사용법

```
smartMemory1.AutoFreeProMemoryAlarmPercent = 20
```

프로퍼티(속성) FD_Drive_FreeBytes, FD_Drive_UseBytes

Flash Disk의 사용 가능한 용량과 사용된 용량을 Byte 단위로 얻습니다.

- FD_Drive_FreeBytes : Flash Disk의 사용 가능한 용량을 Byte 단위로 얻습니다.
- FD_Drive_UseBytes : Flash Disk의 사용된 용량을 Byte 단위로 얻습니다.
- ulong : Flash Disk의 사용 가능한 용량 또는 사용된 용량(Byte)

C# 사용법

```
// Flash Disk의 사용 가능한 용량을 표시
smartLabel1.Text = smartMemory1.FD_Drive_FreeBytes.ToString();
// Flash Disk의 사용된 용량을 표시
smartLabel2.Text = smartMemory1.FD_Drive_UseBytes.ToString();
```

VB 사용법

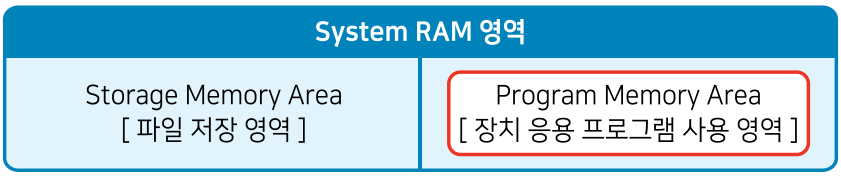
```
smartLabel1.Text = smartMemory1.FD_Drive_FreeBytes.ToString()
smartLabel2.Text = smartMemory1.FD_Drive_UseBytes.ToString()
```

프로퍼티(속성) ProgramMemorySize, ProgramMemoryPercent

System RAM 영역에 할당된 프로그램 메모리 영역의 전체 용량과 System RAM 영역에서의 비율을 얻습니다.

- ProgramMemorySize : 프로그램 메모리 영역의 전체 용량을 Byte 단위로 얻습니다.
- ProgramMemoryPercent : System RAM 영역에서 프로그램 메모리 영역의 비율(%)을 얻습니다.

중요 현재 메모리의 상태를 확인하기 위해서는 반드시 MemoryStatusInformationUpdate() 메소드를 호출하시기 바랍니다.



- uint : 프로그램 메모리 영역의 전체 용량(Byte)
- double : System RAM 영역에서 프로그램 메모리 영역의 비율(%)

C# 사용법

```
// 현재 메모리 상태를 업데이트
smartMemory1.MemoryStatusInformationUpdate();
// 프로그램 메모리 영역의 전체 용량을 KB로 표시
smartLabel1.Text = (smartMemory1.ProgramMemorySize / 1024).ToString() + " KB ";
// 프로그램 메모리 영역의 비율을 표시
smartLabel2.Text = smartMemory1.ProgramMemoryPercent.ToString( " 0.00 " ) + " % ";
```

VB 사용법

```
smartMemory1.MemoryStatusInformationUpdate()
smartLabel1.Text = (smartMemory1.ProgramMemorySize / 1024).ToString() + " KB "
smartLabel2.Text = smartMemory1.ProgramMemoryPercent.ToString( " 0.00 " ) + " % "
```

프로퍼티(속성) FreeProgramMemorySize, FreeProgramMemoryPercent, UseProgramMemorySize, UseProgramMemoryPercent

프로그램 메모리 영역의 사용 가능한 용량과 사용된 용량을 Byte 단위 또는 비율(%)로 얻습니다.

- FreeProgramMemorySize : 프로그램 메모리 영역의 사용 가능한 용량을 Byte 단위로 얻습니다.
- FreeProgramMemoryPercent : 프로그램 메모리 영역의 사용 가능한 용량의 비율(%)을 얻습니다.

• UseProgramMemorySize : 프로그램 메모리 영역의 사용된 용량을 Byte 단위로 얻습니다.
• UseProgramMemoryPercent : 프로그램 메모리 영역의 사용된 용량의 비율(%)을 얻습니다.

중요 현재 메모리의 상태를 확인하기 위해서는 반드시 MemoryStatusInformationUpdate() 메소드를 호출하시기 바랍니다.

• uint : 프로그램 메모리 영역의 사용 가능한 용량 또는 사용된 용량(Byte)
• double : 프로그램 메모리 영역의 사용 가능한 용량의 비율 또는 사용된 용량의 비율(%)

C# 사용법

```
// 현재 메모리 상태를 업데이트
smartMemory1.MemoryStatusInformationUpdate();
// 프로그램 메모리 영역의 사용된 용량을 KB로 표시
smartLabel1.Text = (smartMemory1.UseProgramMemorySize / 1024).ToString() + " KB ";
// 프로그램 메모리 영역의 사용된 용량의 비율을 표시
smartLabel2.Text = smartMemory1.UseProgramMemoryPercent.ToString( " 0.00 " ) + " % ";
// 프로그램 메모리 영역의 사용 가능한 용량을 KB로 표시
smartLabel3.Text = (smartMemory1.FreeProgramMemorySize / 1024).ToString() + " KB ";
// 프로그램 메모리 영역의 사용 가능한 용량의 비율을 표시
smartLabel4.Text = smartMemory1.FreeProgramMemoryPercent.ToString( " 0.00 " ) + " % ";
```

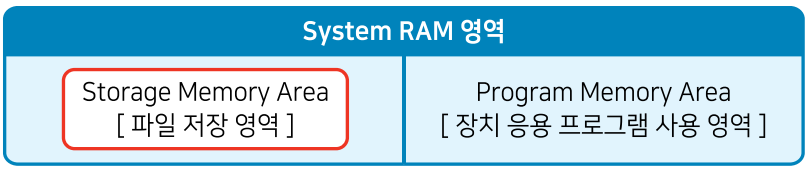
VB 사용법

```
smartMemory1.MemoryStatusInformationUpdate()
smartLabel1.Text = (smartMemory1.UseProgramMemorySize / 1024).ToString() + " KB "
smartLabel2.Text = smartMemory1.UseProgramMemoryPercent.ToString( " 0.00 " ) + " % "
smartLabel3.Text = (smartMemory1.FreeProgramMemorySize / 1024).ToString() + " KB "
smartLabel4.Text = smartMemory1.FreeProgramMemoryPercent.ToString( " 0.00 " ) + " % "
```

프로퍼티(속성) StorageMemorySize, StorageMemoryPercent

System RAM 영역에 할당된 저장소 메모리 영역의 전체 용량과 System RAM 영역에서의 비율을 얻습니다.
• StorageMemorySize : 저장소 메모리 영역의 전체 용량을 Byte 단위로 얻습니다.
• StorageMemoryPercent : System RAM 영역에서 저장소 메모리 영역의 비율(%)을 얻습니다.

중요 현재 메모리의 상태를 확인하기 위해서는 반드시 MemoryStatusInformationUpdate() 메소드를 호출하시기 바랍니다.



• uint : 저장소 메모리 영역의 전체 용량(Byte)
• double : System RAM 영역에서 저장소 메모리 영역의 비율(%)

C# 사용법

```
// 현재 메모리 상태를 업데이트
smartMemory1.MemoryStatusInformationUpdate();
// 저장소 메모리 영역의 전체 용량을 KB로 표시
smartLabel1.Text = (smartMemory1.StorageMemorySize / 1024).ToString() + " KB ";
// 저장소 메모리 영역의 비율을 표시
smartLabel2.Text = smartMemory1.StorageMemoryPercent.ToString( " 0.00 " ) + " % ";
```

VB 사용법

```
smartMemory1.MemoryStatusInformationUpdate()
```

```
smartLabel1.Text = (smartMemory1.StorageMemorySize / 1024).ToString() + " KB "
smartLabel2.Text = smartMemory1.StorageMemoryPercent.ToString( " 0.00 " ) + " % "
```

프로퍼티(속성) FreeStorageMemorySize, FreeStorageMemoryPercent, UseStorageMemorySize, UseStorageMemoryPercent

저장소 메모리 영역의 사용 가능한 용량과 사용된 용량을 Byte 단위 또는 비율(%)로 얻습니다.

중요 현재 메모리의 상태를 확인하기 위해서는 반드시 MemoryStatusInformationUpdate() 메소드를 호출하시기 바랍니다.

- FreeStorageMemorySize : 저장소 메모리 영역의 사용 가능한 용량을 Byte 단위로 얻습니다.
- FreeStorageMemoryPercent : 저장소 메모리 영역의 사용 가능한 용량의 비율(%)을 얻습니다.
- UseStorageMemorySize : 저장소 메모리 영역의 사용된 용량을 Byte 단위로 얻습니다.
- UseStorageMemoryPercent : 저장소 메모리 영역의 사용된 용량의 비율(%)을 얻습니다.
- uint : 저장소 메모리 영역의 사용 가능한 용량 또는 사용된 용량(Byte)
- double : 저장소 메모리 영역의 사용 가능한 용량의 비율 또는 사용된 용량의 비율(%)

C# 사용법

```
// 현재 메모리 상태를 업데이트
smartMemory1.MemoryStatusInformationUpdate();
// 저장소 메모리 영역의 사용된 용량을 KB로 표시
smartLabel1.Text = (smartMemory1.UseStorageMemorySize / 1024).ToString() + " KB ";
// 저장소 메모리 영역의 사용된 용량의 비율을 표시
smartLabel2.Text = smartMemory1.UseStorageMemoryPercent.ToString( " 0.00 " ) + " % ";
// 저장소 메모리 영역의 사용 가능한 용량을 KB로 표시
smartLabel3.Text = (smartMemory1.FreeStorageMemorySize / 1024).ToString() + " KB ";
// 저장소 메모리 영역의 사용 가능한 용량의 비율을 표시
smartLabel4.Text = smartMemory1.FreeStorageMemoryPercent.ToString( " 0.00 " ) + " % ";
```

VB 사용법

```
smartMemory1.MemoryStatusInformationUpdate()
smartLabel1.Text = (smartMemory1.UseStorageMemorySize / 1024).ToString() + " KB "
smartLabel2.Text = smartMemory1.UseStorageMemoryPercent.ToString( " 0.00 " ) + " % "
smartLabel3.Text = (smartMemory1.FreeStorageMemorySize / 1024).ToString() + " KB "
smartLabel4.Text = smartMemory1.FreeStorageMemoryPercent.ToString( " 0.00 " ) + " % "
```

프로퍼티(속성) SD_Drive_FreeBytes, SD_Drive_UseBytes

SD Card의 사용 가능한 용량과 사용된 용량을 Byte 단위로 얻습니다.

- SD_Drive_FreeBytes : SD Card의 사용 가능한 용량을 Byte 단위로 얻습니다.
- SD_Drive_UseBytes : SD Card의 사용된 용량을 Byte 단위로 얻습니다.
- uint : SD Card의 사용 가능한 용량 또는 사용된 용량(Byte)

C# 사용법

```
// SD Card의 사용 가능한 용량을 표시
smartLabel3.Text = smartMemory1.SD_Drive_FreeBytes.ToString();
// SD Card의 사용된 용량을 표시
smartLabel4.Text = smartMemory1.SD_Drive_UseBytes.ToString();
```

VB 사용법

```
smartLabel3.Text = smartMemory1.SD_Drive_FreeBytes.ToString()
smartLabel4.Text = smartMemory1.SD_Drive_UseBytes.ToString()
```

메소드(함수) MemoryStatusInformationUpdate

현재의 메모리 상태 정보를 갱신합니다.

중요 최신의 메모리 상태 정보를 얻기 위해 메모리 정보 관련 속성값 확인 전 반드시 MemoryStatusInformationUpdate() 메소드를 호출하시기 바랍니다.

◉ void MemoryStatusInformationUpdate()

C# 사용법

```
smartMemory1.MemoryStatusInformationUpdate();
```

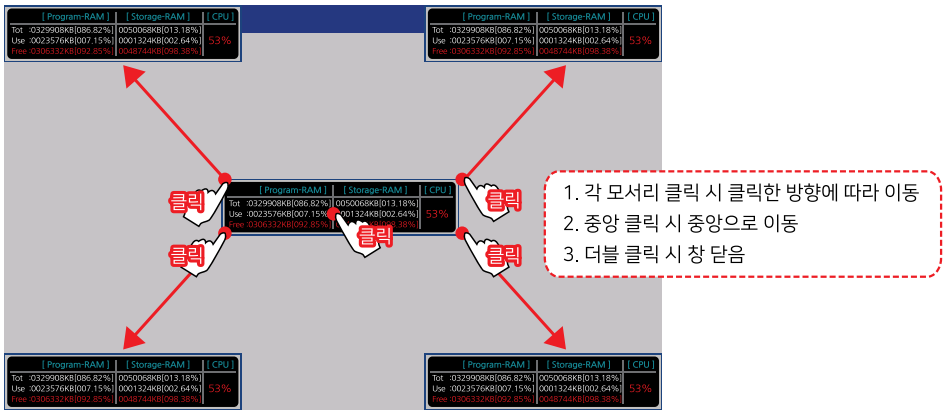
VB 사용법

```
smartMemory1.MemoryStatusInformationUpdate()
```

메소드(함수) MemoryStatusInfomationShow

아래와 같이 현재 시스템 메모리 정보를 모니터링 하기 위한 창을 출력합니다.

주의 메모리 모니터링 창이 폼에 가려지는 경우 ParentWindow 속성을 해당 폼으로 설정하시기 바랍니다.



◉ void MemoryStatusInfomationShow()

◉ void MemoryStatusInfomationShow(SmartMemory.VIEWPOSITION iPosCode)

[인자]

- SmartMemory.VIEWPOSITION iPosCode : 메모리 모니터링 창의 위치
  - SmartMemory.VIEWPOSITION.CENTER : 화면 중앙
  - SmartMemory.VIEWPOSITION.TOPLEFT : 화면 좌측 상단
  - SmartMemory.VIEWPOSITION.TOPRIGHT : 화면 우측 상단
  - SmartMemory.VIEWPOSITION.BOTTOMLEFT : 화면 좌측 하단
  - SmartMemory.VIEWPOSITION.BOTTOMRIGHT : 화면 우측 하단

C# 사용법

```
// 메모리 모니터링 창이 표시될 폼 설정
smartMemory1.ParentWindow = this;
// 메모리 모니터링 창을 표시
smartMemory1.MemoryStatusInfomationShow();
```

VB 사용법

```
smartMemory1.ParentWindow = Me
smartMemory1.MemoryStatusInfomationShow()
```

메소드(함수) MemoryStatusInfomationClose

MemoryStatusInfomationShow() 메소드를 호출하여 출력한 메모리 모니터링 창을 닫습니다.

◉ void MemoryStatusInfomationClose()

C# 사용법

```
smartMemory1.MemoryStatusInfomationClose();
```

VB 사용법

```
smartMemory1.MemoryStatusInfomationClose()
```

메소드(함수) Release

CPU 사용량 관련 기능의 리소스를 모두 해제합니다.

주의 프로그램 종료 시 프로세서가 정상적으로 종료되기 위해 Release() 메소드를 반드시 호출하시기 바랍니다.

◉ void Release()

C# 사용법

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartMemory1.Release();
}
```

VB 사용법

```
Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As System.ComponentModel.
CancelEventArgs) Handles MyBase.Closing
  smartMemory1.Release()
End Sub
```

메소드(함수) GetCPUUsage

CPU 사용률(%)을 얻습니다.

◉ int GetCPUUsage()

[리턴값]

- int : CPU의 사용률(%)

C# 사용법

```
// CPU 사용률을 표시
TextBox1.Text = smartMemory1.GetCPUUsage().ToString();
```

VB 사용법

```
TextBox1.Text = smartMemory1.GetCPUUsage().ToString()
```

메소드(함수) SetProgramRamSize, SetProgramRamPercentSize

프로그램 메모리 영역의 할당량을 조절합니다. 프로그램 메모리 영역의 용량을 변경하는 경우 저장소 메모리 영역의 용량은 연동되어 변경됩니다.

◉ bool SetProgramRamPercentSize(double fRamPercentSize)

◉ bool SetProgramRamSize(uint dwRamByteSize)

[인자]

- double fRamPercentSize : 프로그램 메모리 영역의 비율(%). 80% ~ 현재 사용량까지 설정 가능
- uint dwRamByteSize : 프로그램 메모리의 영역의 용량을 Byte 단위로 설정

[리턴값]

• bool : 프로그램 메모리 설정 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// 프로그램 메모리 영역의 용량을 설정합니다. (Byte 단위 및 퍼센트)
smartMemory1.SetProgramRamSize(32000000);  smartMemory1.SetProgramRamPercentSize(80);
```

VB 사용법

```
smartMemory1.SetProgramRamSize(32000000) : smartMemory1.SetProgramRamPercentSize(80)
```

메소드(함수) StartProMemoryAlarm, StopProMemoryAlarm

프로그램 메모리의 사용량이 AutoFreeProMemoryAlarmPercent 속성에서 설정한 값 이하로 감소될 경우 이벤트가 발생하도록 프로그램 메모리의 감지를 시작 및 종료합니다.

• StartProMemoryAlarm() : 프로그램 메모리 감지를 시작합니다.
• StopProMemoryAlarm() : 프로그램 메모리 감지를 중지합니다.

참고 "AutoFreeProMemoryAlarmPercent 속성, AutoFreeProMemoryAlarmPercent 이벤트" 내용을 참고하시기 바랍니다.

◉ void StartProMemoryAlarm()
◉ void StopProMemoryAlarm()

C# 사용법

```
private void Form1_Load(object sender, EventArgs e)
{
  smartMemory1.StartProMemoryAlarm();  // 프로그램 메모리 감지 시작
}
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartMemory1.StopProMemoryAlarm();  // 프로그램 메모리 감지 중지
}
```

VB 사용법

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Load
  smartMemory1.StartProMemoryAlarm()
End Sub
Private Sub Form1_Closing(ByVal sender As Object, ByVal e As CancelEventArgs)
  smartMemory1.StopProMemoryAlarm()
End Sub
```

메소드(함수) ExtStorageDetectionStart

외부저장장치(SD Card, USB Memory)의 연결 감지를 시작합니다. 연결이 감지된 경우 EvtExternalStorageAttached 이벤트가 발생합니다.

◉ void ExtStorageDetectionStart()

C# 사용법

```
private void Form1_Load(object sender, EventArgs e)
{
  smartMemory1.ExtStorageDetectionStart();  // 외부저장장치 연결 상태 감지 시작
}
```

VB 사용법

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Load
 smartMemory1.ExtStorageDetectionStart()
End Sub
```

메소드(함수) ExtStorageDetectionStop

외부저장장치(SD Card, USB Memory)의 연결 감지를 종료합니다. 외부저장장치가 연결 또는 해제되면 EvtExternal StorageAttached 이벤트가 발생합니다.

주의 ExtStorageDetectionStart() 메소드를 호출한 경우 프로그램 종료 시 반드시 호출하시기 바랍니다.

참고 "EvtExternalStorageAttached 이벤트" 내용을 참고하시기 바랍니다.

◉ `void ExtStorageDetectionStop()`

C# 사용법

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
  // 외부저장장치 연결 상태 감지 중지
  smartMemory1.ExtStorageDetectionStop();
}
```

VB 사용법

```
Private Sub Form1_Closing(ByVal sender As Object, ByVal e As CancelEventArgs)
  smartMemory1.ExtStorageDetectionStop()
End Sub
```

이벤트 EvtExternalStorageAttached

IEC-Series에서 외부저장장치(SD Card, USB Memory)가 연결 및 해제 시 발생되는 이벤트로서 ExtStorageDetectionStart() 메소드를 호출하면 이벤트 사용이 활성화되고 ExtStorageDetectionStop() 메소드를 호출하면 이벤트 사용이 비활성화 됩니다. IEC-Series에서 외부저장장치(USB Memory, SD Card)의 연결 및 연결 해제(UnMount) 과정에서 메모리와 관련하여 접근이 일어나는 경우 사용자 프로그램의 동작 지연(내부 Timer 동작이 일시 정지, 터치 입력 안 됨 등)현상이 발생됩니다. EvtExternalStorageAttached 이벤트를 사용하게 되면 IEC-Series에 외부저장장치가 연결 되거나 연결 해제되는 것을 바로 감지하여 외부저장장치의 접근을 금지시켜 지연현상을 방지 할 수 있습니다. 또한 Timer를 사용하여 주기적으로 외부 장치의 특정 폴더나 파일을 체크하는 방식에 비해 CPU 리소스를 상대적으로 적게 사용합니다.

참조 IEC-Series에 외부저장장치를 연결 시 IEC-Series에 외부저장장치가 연결 되는데까지 일정 시간이 소요되고 연결 해제 시에도 IEC-Series에서 외부 장치가 언마운트 되는데까지 일정 시간이 소요됩니다. "홈페이지 → 자료실 → Tech Note → 68. 외부저장장치(USB 메모리, SD Card) 연결 해제 시 시스템 동작 지연 개선 방법" 내용을 참조하시기 바랍니다

◉ `EvtExternalStorageAttached(bool bAttached, string strDeviceName)`

[인자]

- `bool bAttached` : 연결 상태
  - `true` : 연결됨
  - `false` : 연결 해제
- `string strDeviceName` : 외부 장치 이름
  - 하드 디스크 : USB Memory (IEC266-Series의 경우 "₩₩하드 디스크")
  - `SD Card` : SD Card (IEC266-Series 의 경우 "₩₩SD Card")

C# 사용법

```
private void smartMemory1_EvtExternalStorageAttached(bool bAttached, string strDeviceName)
{
  // 인자값으로 외부저장장치의 상태를 확인
  if ((bAttached == true) & (strDeviceName = " SD Card " )
  {
    // SD Card가 연결(IEC266 : ₩₩ SD Card)
    FileSearch();
   }
   else if ((bAttached == true) & (strDeviceName = " 하드 디스크 " )
   {
     // USB 메모리 연결(IEC266 : ₩₩ 하드 디스크)
     FileSearch()
   }
  else if ((bAttached == false) & (strDeviceName = " SD Card " )
  {
    // SD Card가 연결 해제(IEC266-Series : ₩₩ SD Card)
  }
  else if ((bAttached == false) & (strDeviceName = " 하드 디스크 " )
  {
    // USB 메모리 연결 해제(IEC266 : ₩₩ 하드 디스크)
  }
}
private void FileSearch()
{
  if (System.IO.File.Exists( " 하드 디스크₩₩Log₩₩20210101.txt " ) == true)
  {
    // 하드 디스크₩Log 폴더에 있는 20210101.txt 파일이 있는 경우 처리할 코드
  }
}
```

VB 사용법

```
Private Sub smartMemory1_EvtExternalStorageAttached(ByVal bAttached As Boolean, ByVal
strDeviceName As String) Handles SmartMemory1.EvtExternalStorageAttached
  If(bAttached = True) And (strDeviceName = " SD Card " ) Then
    ' SD Card 연결(IEC266 : ₩₩ SD Card)
    FileSearch()
  Else If(bAttached = True) And (strDeviceName = " 하드 디스크 " ) Then
    ' USB 메모리 연결(IEC266 : ₩₩ 하드 디스크)
    FileSearch()
  Else If(bAttached = False) And (strDeviceName = " SD Card " ) Then
    ' SD Card 연결(IEC266 : ₩₩ SD Card)
  Else If(bAttached = False) And (strDeviceName = " 하드 디스크 " ) Then
    ' USB 메모리 연결(IEC266 : ₩₩ 하드 디스크)
  End If
End Sub
Private Sub FileSearch()
  If System.IO.File.Exists( " 하드 디스크\\Log\\20210101.txt " ) = True Then
    ' 하드 디스크₩Log 폴더에 있는 20210101.txt 파일이 있는 경우 처리할 코드
  End If
End Sub
```

이벤트	OnProgramMemoryAlarmEvent

사용 가능한 프로그램 메모리가 AutoFreeProMemoryAlarmPercent 속성값 이하로 될 경우 발생되는 이벤트입니다.

참조 SmartBattery 관련하여 장착 방법 및 하드웨어 관련 설명은 "홈페이지 → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼"을 참조하시기 바랍니다.

C# 사용법

```
// 프로그램 메모리가 AutoFreeProMemoryAlarmPercent 속성값 이하로 될 경우 발생되는 이벤트
private void smartMemory1_OnProgramMemoryAlarmEvent(object sender, EventArgs e)
{
  // 메모리 부족 알림 메시지 출력
  MessageBox.Show( " 프로그램 메모리가 부족합니다. !!! " );
}
```

VB 사용법

```
Private Sub smartMemory1_OnProgramMemoryAlarmEvent(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles smartMemory1.OnProgramMemoryAlarmEvent
  MessageBox.Show( " 프로그램 메모리가 부족합니다. !!! " )
End Sub
```

3) SmartMemory 예제 사용하기

SmartMemory를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartMemory"

5. SmartModbus

Modbus는 전 세계적으로 널리 보급되어 있는 자동화 분야에서 주로 사용되는 프로토콜이며, 현재 PLC, 제어 시스템, HMI, 계측기 등 수많은 산업용 장비들의 통신 표준으로 사용되고 있습니다. SmartModbus는 IEC-Series 제품에서 Master 측 기능을 지원하며 Modbus 사용을 편리하게 할 수 있는 컴포넌트입니다.

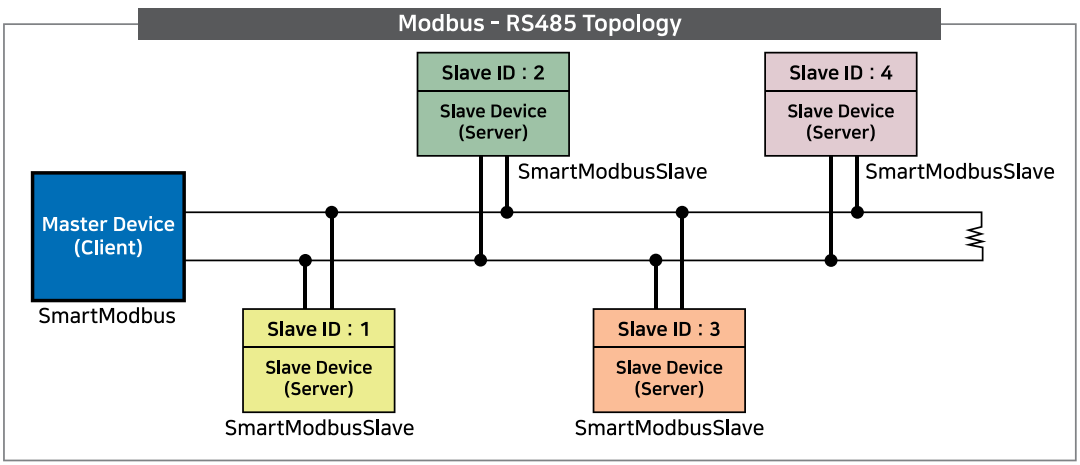
Modbus 프로토콜은 Master-Slave 기법을 사용합니다. 즉 Master가 Slave에게 데이터를 요청하고 Slave는 Master에 응답하는 구조로 되어있습니다. Master에서는 모든 Slave에게 정보를 요청할 수도 있습니다. Slave에는 각자의 Address가 지정돼있어 Master에서 요청한 Address를 확인하여 Address가 일치하면 요청에 응답하고 아니면 무시합니다. Modbus 프로토콜의 메시지(데이터) 전송 방식은 크게 ASCII(American Standard Code for Information Interchange) 모드와 RTU(Remote Terminal Unit) 모드로 나눌 수 있는데 ASCII 모드는 데이터를 ASCII 형태로 구성하는 것이며, RTU 모드는 데이터를 Binary 형태로 구성하는 것입니다. 현재 SmartModbus와 Smart ModbusSlave는 RTU 방식만 지원합니다.

- Modbus-Master 측 기능 지원
- RTU 모드 방식 처리
- Read Function (Function : 1 ~ 4) 지원
- Write Function (Function : 5, 6, 15, 16) 지원
- 송/수신 모드 전환 시간 설정 지원
  ↳ SetResponseInterval() 메소드

참고 SmartModbus 적용 시 어려움이 있는 경우, IEC-Series 제품과 특정 I/O 보드를 구성하여 보내주시면 호환성을 검증해드리도록 하겠습니다. 또한 적용 시 발생하는 문제를 HNS에 알려주시면 조치해 드리도록 하겠습니다. Modbus를 지원하는 산업용 장비가 다양하여 일일이 대응하지 못한 점 양해 바랍니다.

1) RS485 & Modbus Protocol 설명

Modbus 프로토콜은 RS485 통신을 기반으로 하며, 다음과 같은 Topology Group을 기준으로 구성됩니다. SmartModbus는 미리 정의된 Function을 이용해 Master가 Slave로 Request하면 Slave는 Master로 수신받은 Function에 해당하는 데이터를 Response하는 구조로 되어있습니다. 이때 Request에 따른 Response 데이터가 잘못되거나 올바르게 수신되지 않는 예외가 발생하는 경우 Master는 해당 예외에 따른 Exception Code를 사용자에게 리턴합니다. SmartModbus를 사용하시면 별도로 Data Frame을 구성할 필요 없이 메소드를 사용하여 간단히 Function에 따른 Data Frame을 전송할 수 있습니다. 아래 표에서 내용을 확인하시기 바랍니다.



참고 RS485 – Modbus Master and Slave 송/수신 규약

1. Master-Device는 1개만 존재한다.
2. Master-Device 데이터를 임의로 송/수신할 수 있으며 반드시 송신(Request) → 수신(Response)의 구조로 통신되어야 한다.
3. Slave-Device는 여러 개가 존재하며 각각 Slave Address를 가지고 있다.
4. Slave-Device는 데이터를 임의로 송신할 수 없으며, Master-Device의 Request 데이터 수신 시 Slave Address가 일치하는 Slave-Device만이 Response 데이터를 송신할 수 있다.
5. 모든 Slave-Device는 기본적으로 수신(대기)상태를 유지한다.

[표1] SmartModbus에서 지원하는 Read 관련 Function Code 및 Data Frame 구조

Function Code	Name	Description (업체별 정의가 다를 수 있음)
01	Read Coil Status	Digital Output 접점 읽기
02	Read Input Status	Digital Input 접점 읽기
03	Read Holding Registers	Analog Output 데이터 읽기
04	Read Input Registers	Analog Input 데이터 읽기

※ Request[Master->Slave] Data Frame 구조

Slave Address	Function Code	Start Address		Data Length		CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	상위	하위	상위	하위
		데이터 주소		읽을 데이터 개수		CRC16	

※ Response[Slave->Master] Data Frame 구조

Slave Address	Function Code	Byte Count	Read Datas				CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	…	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	데이터 바이트 수	1번째 데이터	2번째 데이터	…	N번째 데이터	상위	하위
							CRC16	

[표2] SmartModbus에서 지원하는 Write(Single) 관련 Function Code 및 Data Frame 구조

Function Code	Name	Description (업체별 정의가 다를 수 있음)
05	Force Single Coil(Write Single Coil)	단일 Output 접점 쓰기
06	Preset Single Register(Write Single Register)	단일 Output 데이터 쓰기

참고 Single Coil/Register Write의 경우 Request와 Response의 Frame 구조는 같습니다.

※ Request[Master -> Slave] Data Frame 구조

Slave Address	Function Code	Start Address		Write Data	CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	쓰기 데이터	상위	하위
		데이터 주소			CRC16	

※ Response[Slave -> Master] Data Frame 구조

Slave Address	Function Code	Start Address		Write Data	CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	쓰기 데이터	상위	하위
		데이터 주소			CRC16	

[표3] SmartModbus에서 지원하는 Write(Multiple) 관련 Function Code 및 Data Frame 구조

Function Code	Name	Description (업체별 정의가 다를 수 있음)
15	Force Multiple Coil(Write Multiple Coil)	여러 Output 접점 쓰기
16	Preset Multiple Registers(Write Multiple Registers)	여러 Output 데이터 쓰기

※ Request[Master -> Slave] Data Frame 구조

Slave Address	Function Code	Start Address		NumberOfCoil/ NumberOfRegister		Byte Cnt	Write Datas			CRC16 (내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	…	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	상위	하위	바이트 개수	1번째 데이터	…	n번째 데이터	상위	하위
		데이터 주소		쓰기 코일(레지스터) 개수						CRC16	

※ Response[Slave -> Master] Data Frame 구조

Slave Address	Function Code	Start Address		NumberOfCoil/ NumberOfRegister		CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	상위	하위	상위	하위
		데이터 주소		쓰기 코일(레지스터) 개수		CRC16	

[표4] SmartModbus에서 지원하는 Exception Code 및 Data Frame 구조

Exception Code	Name	Description (업체별 정의가 다를 수 있음)
0	SUCCESS	성공
1	ILLEGAL_FUNCTION	잘못된 기능 코드 사용
2	ILLEGAL_DATA_ADDRESS	잘못된 주소 지정
3	ILLEGAL_DATA_VALUE	잘못된 값 설정
4	SLAVE_DEVICE_FAILURE	Slave 장치 오류
5	ACKNOWLEDGE	응답 시간이 수신 응답 시간보다 클 경우 수신 응답 오류를 연장하기 위하여 사용됨
6	SLAVE_DEVICE_BUSY	Slave가 다른 처리로 인하여 송/수신 처리가 어려운 상태
223	READBUFFER_OVER	수신 버퍼에 수신 데이터가 꽉찬 경우
224	READDATA_ERROR	비정상적인 수신 데이터
225	SLAVEADDRESS_ERROR	송신 데이터의 Slave Address와 수신 데이터의 Slave Address가 일치하지 않음
226	READDATACOUNT_ERROR	Read 관련 Function에서 실제 읽어들인 데이터의 수가 Byte count와 일치하지 않음
227	CRC16_ERROR	수신 데이터의 CRC16 값과 계산된 CRC16 값이 일치하지 않음
228	READTIMEOUT_ERROR	수신 응답 시간 초과

※ Response[Slave -> Master] Data Frame 구조(MSB(Most Significant Bit) : 최상위 Bit)

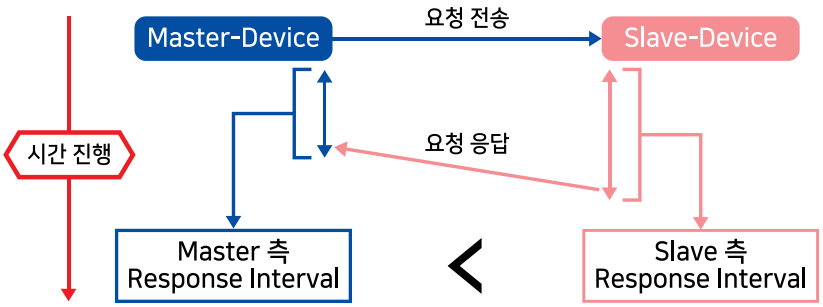
SlaveAddress	FunctionCode	eExceptionCode	CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	MSB를 1로 OR 연산한 기능 코드	발생한 Exception 코드	상위	하위
			CRC16	

참조 "SimplyModbus(www.simplymodbus.ca)" 홈페이지를 참조하시기 바랍니다.

2) Master 와 Slave-Device 의 Response Interval 관계

Modbus는 RS485 기반의 프로토콜이므로 Half-Duflex 방식의 통신을 합니다. 따라서 송신과 수신을 동시에 할 수 없

으며, 만약 데이터 송신 중에 데이터를 수신받게 된다면 데이터의 충돌이 발생하게 됩니다. 따라서 Master-Device의 요청 전송에 따른 Slave-Device의 요청 응답에는 일정 시간만큼의 대기 시간이 필요하게 됩니다. 아래 표에서 내용을 확인하시기 바랍니다.



2-1) Master-Device와 Slave-Device의 ResponseInterval 값 설정 기준 및 방법

1 Master-Device의 ResponseInterval은 1차적으로 Slave-Device의 ResponseInterval 값에 의해 결정됩니다.

Master-Device의 ResponseInterval은 Slave-Device의 요청 응답 시간(ResponseInterval)에 맞게 설정해야 합니다.

2 Slave-Device의 ResponseInterval 값은 Master-Device의 ResponseInterval 값보다 크게 설정합니다. (Slave-Device ResponseInterval 〉 Master-Device ResponseInterval)

Master-Device에서 요청 전송 후 Slave-Device는 요청 응답 처리 시 Master-Device가 수신받을 준비가 되기 전에 응답을 전송할 경우 데이터의 충돌이 발생할 수 있습니다. 따라서 Master-Device가 수신 상태에서 요청 응답을 수신 받을 수 있도록 Slave-Device의 ResponseInterval 값은 Master-Device의 ResponseInterval 값보다 크게 설정해야 합니다.

3 RS485 H/W 자동 제어 방식은 RS485 S/W 자동 제어 방식보다 상대적으로 ResponseInterval 값을 작게 사용할 수 있습니다. (H/W 방식이 S/W 방식보다 빠르게 응답 가능)

H/W 자동 제어 방식은 H/W 단에서 송/수신 모드를 변경하여 S/W 자동 제어 방식보다 ResponseInterval 값을 작게 사용할 수 있습니다.

※ 아래의 "H/W 방식과 S/W 방식의 차이점 및 IEC-Series별 송/수신 제어 방식"을 참고하시기 바랍니다.

4 ReponseInterval은 Slave-Device와 Master-Device 양쪽 모두 설정되야합니다.

Master-Device의 ResponseInterval은 송/수신 모드의 전환 중에 요청 응답 수신을 방지하기 위해 설정해야 하며, Slave-Device의 ResponseInterval은 Master-Device의 송/수신 모드 전환 중에 요청 응답 송신을 방지하기 위해 설정해야 합니다. 또한 ResponseInterval은 충분한 테스트를 거쳐 값을 설정해야 합니다.

참고 H/W 방식과 S/W 방식의 차이점 및 IEC-Series별 송/수신 제어 방식

IEC-Series별로 송/수신 제어 방식의 차이가 있습니다. 아래 표를 확인하시기 바랍니다.

[표] H/W 방식과 S/W 방식의 차이점 및 IEC-Series별 송/수신 제어 방식

구분	H/W 방식			S/W 방식		
특징	Request 수신에 대한 Response 응답을 S/W 방식에 비해 빠르게 할 수 있음			Request 수신에 대한 Response 응답이 H/W 방식에 비해 느림		
IEC - Series	IEC266-Series		IEC667-Series		IEC1000-Series	
	非Lite	Lite	非Lite	Lite	非Lite	Lite
송/수신 제어 방식	S/W		H/W	S/W	H/W	S/W

3) 프로그래밍 적용 가이드

STEP-1 통신 관련 설정값 적용 및 시작하기

Modbus-Master 구성을 위해 기본 설정을 진행합니다. Buad_Rate 속성으로 보드레이트, PortNo 속성으로 통신 포트, ProtocolType 속성으로 프로토콜 타입(RTU만 지원), StopBits, Parity 속성으로 에러 검출 설정을 할 수 있습니다. 또한, ReadTimeout 속성으로 수신 응답 대기 시간을, SetResponseInterval() 메소드로 수신 모드 전환 시간을 설정합니다. 이중 SetResponseInterval 값은 Master 측과 Slave 측을 동일하게 설정해야합니다. 모든 설정이 완료되면 PortOpen() 메소드를 호출해 SmartModbus를 시작할 수 있습니다.

※ 자세한 내용은 "Master-Device와 Slave-Device의 ResponseInterval 값 설정 기준 및 방식", "Buad_Rate, Port No, ProtocolType, StopBits, Parity, ReadTimeout 속성", "SetResponseInterval(), PortOpen() 메소드"를 참고하시기 바랍니다.

```
// 보드레이트 설정
smartModbus1.Buad_Rate = SmartX.SmartModbus.BUADRATE.CBR_9600;
// 통신 포트 설정 (COM1 : RS485)
smartModbus1.PortNo = SmartX.SmartModbus.COMPORTNO.COM1;
// 프로토콜 타입을 설정 (RTU만 지원됨)
smartModbus1.ProtocolType = SmartX.SmartModbus.PROTOCOL.RTU;
// 에러 검출 코드 설정
smartModbus1.StopBits = System.IO.Ports.StopBits.None;
smartModbus1.Parity = System.IO.Ports.Parity.None;
// 응답 대기 시간 설정
smartModbus1.ReadTimeout = 3000;
// 송신 모드에서 수신 모드로 전환 시간을 설정
smartModbus1.SetResponseInterval(70);
if (smartModbus1.IsOpen == false)
{
  // SmartModbus 시작
  smartModbus1.PortOpen();
}
```

STEP-2 Function 송신 및 응답 수신하기

SmartModbus는 각 Function을 메소드로 구성하여 Modbus-Slave로 간편하게 Request를 전송할 수 있으며, 리턴 값을 확인해 올바르게 처리되었는지를 확인할 수 있습니다. SmartModbus는 총 8가지 Function을 지원하며 본 가이드는 FC01(ReadCoilStatus)을 예로 설명합니다.

※ 자세한 내용은 "RS485 & Modbus Protocol 설명" 및 Read/Write 계열 메소드를 참고하시기 바랍니다.

```
private void ReadCoil()
{
  // 데이터를 전송할 Slave Address를 설정
  smartModbus1.SlaveAddress = 1;
  // Request 전송 후 수신 모드로 전환하는 시간(70m초)
  smartModbus1.SetResponseInterval(70);
  // Response 응답 제한 시간(3초)
  smartModbus1.ReadTimeout = 3000;

  // Response 받은 데이터를 저장할 배열
  byte[] bReadByte = new byte[6];
  // 데이터 전송 후 결과를 저장할 변수
  SmartX.SmartModbus.EXCEPTIONCODE eCode;
  // Slave 장비로 Request 요청을 전송 후 데이터와 결과를 저장합니다.
```

```
  eCode = smartModbus1.ReadCoilStatus(1, bReadByte.Length / 2, ref bReadByte);

  // SetResponseInterval()에서 설정한 시간인 1초 동안 송신 모드에서 수신 모드로 전환되어
  // 데이터를 수신할 수 없음. 1초 후 ReadTimeout으로 설정한 시간인 3초 동안 Response 수신을 대기함
  // 즉, Read/Write 메소드 호출 후 SetResponseInterval과 ReadTimeout 시간을 더한 시간만큼
  // Response 수신을 대기하며 다른 작업은 Blocking 됨.
  // " Master-Device와 Slave-Device의 ResponseInterval값 설정 기준 및 방식 " 참고

 // 리턴값으로 결과를 확인
  if (eCode == SmartX.SmartModbus.EXCEPTIONCODE.SUCCESS)
  {
    // Request-Response 성공. 라벨에 수신 받은 데이터를 표시
    lblCoil1.Text = bReadByte[0].ToString( " X2 " ) + bReadByte[1].ToString( " X2 " );
    lblCoil2.Text = bReadByte[2].ToString( " X2 " ) + bReadByte[3].ToString( " X2 " );
    lblCoil3.Text = bReadByte[4].ToString( " X2 " ) + bReadByte[5].ToString( " X2 " );
  }
  else if (eCode == SmartX.SmartModbus.EXCEPTIONCODE.READTIMEOUT_ERROR)
  {
    // 수신 응답 시간이 초과됨
  }
  // ...중략...
}
```

STEP-3 프로그램 종료 시 SmartModbus 종료하기

프로그램 종료 시 SmartModbus의 동작 여부를 IsOpen 속성으로 확인 후, 동작 중이라면 PortClose() 메소드를 호출해 SmartModbus가 사용 중인 Port를 닫습니다. 만약 Port를 닫지 않고 프로그램을 종료한다면, 리소스에 남게되어 프로그램이 정상적으로 종료되지 않을 수 있습니다.

※ 자세한 내용은 "IsOpen 속성", "PortClose() 메소드"를 참고하시기 바랍니다.

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
  if (smartModbus1.IsOpen == true)
  {
    // SmartModbus를 종료한다.
    smartModbus1.PortClose();
  }
}
```

4) SmartModbus 인터페이스 설명

SmartModbus Component Interface		
속성		
Buad_Rate : SmartModbus.BUADRATE	IsOpen : bool	Parity : Parity
PortNo : SmartModbus.COMPORTNO	ProtocolType : SmartModbus.PROTOCOL	ReadTimeout : int
SlaveAddress : int	StopBits : StopBits	
메소드		
PortClose() : void	PortOpen() : bool	ReadCoilStatus(int iStartAddress, int iDataLength, ref byte[] retDatas) : SmartMobus.EXCEPTIONCODE

ReadHoldingRegister(int iStartAddress, int iDataLength, ref byte[] retDatas) : SmartMobus.EXCEPTIONCODE	ReadInputRegister(int iStartAddress, int iDataLength, ref byte[] retDatas) : SmartMobus.EXCEPTIONCODE	ReadInputStatus(int iStartAddress, int iDataLength, ref byte[] retDatas) : SmartMobus.EXCEPTIONCODE
SetResponseInterval(int iResponseInterval) : void	WriteMultipleCoil(int iStartAddress, int iDataLength, byte[] iWriteDatas) : SmartMobus.EXCEPTIONCODE (+1개 오버로드)	WriteMultipleRegister(int iStartAddress, int iDataLength, byte[] iWriteDatas) : SmartMobus.EXCEPTIONCODE (+1개 오버로드)
WriteSingleCoil(int iStartAddress, int iWriteData) : SmartMobus.EXCEPTIONCODE	WriteSingleRegister(int iStartAddress, int iWriteData) : SmartMobus.EXCEPTIONCODE	

프로퍼티(속성) Buad_Rate

통신 속도(Baud Rate)를 설정합니다.

주의 485 지원 통신 속도(Baud Rate) 안내

IEC-Series의 485 Port를 사용하여 송/수신 시 발생할 수 있는 문제점으로 9600, 19200, 38400, 57600, 115200 이외의 통신 속도(BaudRate)를 사용하는 경우 수신 측의 송/수신 Interval 문제가 발생할 수 있으므로 송/수신 (Request & Response) Interval을 조절하여 사용해야 합니다.

- SmartModbus.BUADRATE.CBR_9600 : 통신 속도를 9600bps로 설정
- SmartModbus.BUADRATE.CBR_19200 : 통신 속도를 19200bps로 설정
- SmartModbus.BUADRATE.CBR_38400 : 통신 속도를 38400bps로 설정
- SmartModbus.BUADRATE.CBR_57600 : 통신 속도를 57600bps로 설정
- SmartModbus.BUADRATE.CBR_115200 : 통신 속도를 115200bps로 설정

C# 사용법

```
smartModbus1.Buad_Rate = SmartModbus.BUADRATE.CBR_19200;  // 통신 속도를 19200bps로 설정
```

VB 사용법

```
smartModbus1.Buad_Rate = SmartModbus.BUADRATE.CBR_19200
```

프로퍼티(속성) Parity

패리티 검사 프로토콜 방식을 설정합니다.

- System.IO.Ports.Parity.Even : Bit 집합의 합계가 짝수가 되도록 패리티 Bit를 설정
- System.IO.Ports.Parity.Mark : 패리티 Bit를 1로 설정된 상태로 유지
- System.IO.Ports.Parity.None : 패리티 검사를 수행하지 않음
- System.IO.Ports.Parity.Odd : Bit 집합의 합계가 홀수가 되도록 패리티 Bit를 설정
- System.IO.Ports.Parity.Space : 패리티 Bit를 0으로 설정된 상태로 유지

C# 사용법

```
// Parity 검사를 사용하지 않음으로 설정
smartModbus1.Parity = System.IO.Ports.Parity.None;
```

VB 사용법

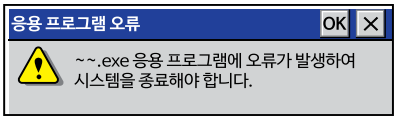
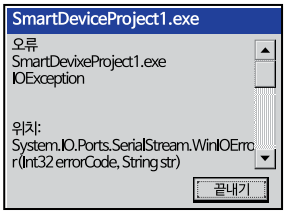
```
smartModbus1.Parity = System.IO.Ports.Parity.None
```

프로퍼티(속성) PortNo

사용할 통신 Port 번호를 설정합니다. IEC-Series별로 내장된 통신 Port의 개수가 다르며, COM7의 경우 USB FTDI VCP 드라이버를 사용합니다.

주의 USB to Serial 컨버터(COM7) 사용 시 주의사항

USB to Serial 컨버터를 USB에 연결해 COM7로 Port 사용 시, 통신 중 USB 컨버터를 제거하지 마시기 바랍니다. 통신 중에 USB 컨버터를 제거하는 경우 다음과 같은 오류가 발생할 수 있습니다

에러 유형 1	에러 유형 2
	

[표] IEC-Series별 지원 통신 Port

IEC-Series	COM1_1	COM1	COM2	COM3	COM4	COM7
IEC266	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	미지원	미지원
IEC266Lite	미지원 (RS485-옵션)	지원 (3.3V-TTL)	지원 (RS232)	지원 (RS232)	미지원	미지원
IEC667	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC667Lite	지원 (RS485)	지원 (5V-TTL)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC1000	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC1000Lite	지원 (RS485)	지원 (5V-TTL)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
Comport No.	COM1		COM2	COM3	COM4	COM7

- SmartModbus.COMPORTNO.COM1 : COM1
- SmartModbus.COMPORTNO.COM2 : COM2
- SmartModbus.COMPORTNO.COM3 : COM3
- SmartModbus.COMPORTNO.COM4 : COM4 (IEC266-Series 미지원)
- SmartModbus.COMPORTNO.COM7 : COM7 (IEC266-Series 미지원)

C# 사용법

```
// COM1을 사용합니다.
smartModbus1.PortNo = SmartModbus.COMPORTNO.COM1;
```

VB 사용법

```
smartModbus1.PortNo = SmartModbus.COMPORTNO.COM1
```

프로퍼티(속성) ProtocolType

Modbus 통신에서 데이터 표현 방식을 설정합니다. SmartModbus는 RTU 방식만 지원합니다.

- SmartModbus.PROTOCOL.RTU : 데이터 표현 방식을 RTU(Binary)로 설정
- SmartModbus.PROTOCOL.ASCII : 미지원

C# 사용법

```
smartModbus1.ProtocolType = SmartModbus.PROTOCOL.RTU;
```

VB 사용법

```
smartModbus1.ProtocolType = SmartModbus.PROTOCOL.RTU
```

프로퍼티(속성) ReadTimeout

Modbus 통신 시 데이터의 응답 대기 시간을 설정합니다. 명령 전송 후 Response 응답이 없을 경우 Time Out 처리하기 위한 시간을 설정합니다. (기본값 : -1(무한 대기))

- int : Time Out 처리 시간 (단위 : ms)

C# 사용법

```
smartModbus1.ReadTimeout = 800;  // 응답 대기 시간을 800ms로 설정
```

VB 사용법

```
smartModbus1.ReadTimeout = 800
```

프로퍼티(속성) SlaveAddress

Slave-Device의 주소(Address)를 설정합니다.

- int : Slave-Device 주소(Address)

C# 사용법

```
smartModbus1.SlaveAddress = 1;  // Slave-Device 주소를 1로 설정
```

VB 사용법

```
smartModbus1.SlaveAddress = 1
```

프로퍼티(속성) StopBits

Byte당 정지 Bit의 표준 개수를 가져오거나 설정합니다.

- System.IO.Ports.StopBits.None : 정지 Bit를 사용하지 않음
- System.IO.Ports.StopBits.One : 1Bit의 정지 Bit를 사용
- System.IO.Ports.StopBits.OnePointFive : 1.5Bit의 정지 Bit를 사용
- System.IO.Ports.StopBits.Two : 2Bit의 정지 Bit를 사용

C# 사용법

```
smartModbus1.StopBits = System.IO.Ports.StopBits.One;  // 1Bit의 정지 Bit를 사용
```

VB 사용법

```
smartModbus1.StopBits = System.IO.Ports.StopBits.One
```

프로퍼티(속성) IsOpen

현재 통신 Port가 Open 되었는지를 확인합니다.

- bool : 통신 Port Open 여부
  - true : Open 됨
  - false : Close 됨

C# 사용법

```
// 통신 Port가 열려있는 경우 Port를 닫는다.
if (smartModbus1.IsOpen == true)
{
  smartModbus1.PortClose();
}
```

VB 사용법

```
If smartModbus1.IsOpen = True Then : smartModbus1.PortClose () : End If
```

메소드(함수) PortOpen, PortClose

- PortOpen() : 통신(Serial) Port를 Open합니다.
- PortClose() : 통신(Serial) Port를 Close합니다.

◉ bool PortOpen()
◉ bool PortClose()

[리턴값]

- bool : Port Open 또는 Close 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
if (smartModbus1.PortOpen() == true)
{
  // Port Open 성공
}
```

VB 사용법

```
If smartModbus1.PortOpen() = True Then
  ' Port Open 성공
End If
```

메소드(함수) SetResponseInterval

송신 모드에서 수신 모드로의 전환 시간을 설정합니다.

참고 "Master와 Slave-Device의 Response Interval 관계" 내용을 참고하시기 바랍니다.

◉ void SetResponseInterval(int iResponseInterval)

[인자]

- int iResponseInterval : 전환 시간을 설정합니다. (단위 ms)

C# 사용법

```
smartModbus1.SetResponseInterval(70);  // 전환 시간을 70m초로 설정
```

VB 사용법

```
smartModbus1.SetResponseInterval(70)
```

메소드(함수) ReadCoilStatus, ReadInputStatus, ReadHoldingRegister, ReadInputRegister

Read 명령(Function Code) 관련 동작을 수행합니다.

참고 "RS485 & Modbus Protocol 설명" 내용을 참고하시기 바랍니다.

- ReadCoilStatus() : Function code 1(0x01). Read Coil Status의 기능을 수행합니다.
- ReadInputStatus() : Function code 2(0x02). Read Input Status의 기능을 수행합니다.
- ReadHoldingRegister() : Function code 3(0x03). Read Holding Register의 기능을 수행합니다.
- ReadInputRegister() : Function code 4(0x04). Read Input Register의 기능을 수행합니다.

◉ SmartModbus.EXCEPTIONCODE ReadCoilStatus(int iStartAddress, int iDataLength, ref byte[] retDatas)
◉ SmartModbus.EXCEPTIONCODE ReadInputStatus(int iStartAddress, int iDataLength, ref byte[] retDatas)
◉ SmartModbus.EXCEPTIONCODE ReadHoldingRegister(int iStartAddress, int iDataLength, ref byte[] retDatas)

◉ SmartModbus.EXCEPTIONCODE ReadInputRegister(int iStartAddress, int iDataLength, ref byte[] retDatas)

[인자]

- int iStartAddress : 시작 주소값
- int iDataLength : 데이터 개수
- ref byte[] retDatas : 응답 데이터

[리턴값]

- SmartModbus.EXCEPTIONCODE : 명령 수행 결과

열거값	설명	열거값	설명
SUCCESS	정상 처리 완료	READBUFFER_OVER	수신 버퍼 오버됨
ILLEGAL_FUNCTION	잘못된 기능 코드	READDATA_ERROR	비정상적인 수신 데이터
ILLEGAL_DATA_ADDRESS	잘못된 데이터 주소	SLAVEADDRESS_ERROR	Slave Address 에러
ILLEGAL_DATA_VALUE	잘못된 데이터	READDATACOUNT_ERROR	읽어들인 데이터의 수가 잘못됨
SLAVE_DEVICE_FAILURE	Slave 장치 오류	CRC16_ERROR	내부 CRC16 검사 에러
ACKNOWLEDGE	수신 응답 오류를 연장	READTIMEOUT_ERROR	수신 응답 시간 초과
SLAVE_DEVICE_BUSY	Slave 처리 중	-	

C# 사용법

```
byte[] bRecv = new byte[4];
if(smartModbus1.ReadHoldingRegister(20, 2, ref bRecv) == SmartModbus.EXCEPTIONCODE.SUCCESS)
{
  // SmartSerialPort의 정적 메소드를 사용해 수신받은 데이터를 라벨에 출력
  labComStatus.Text = " 읽기 성공: " + SmartSerialPort.ConverByteToHexDecString(bRecv, "  ");
}
```

VB 사용법

```
Dim temp As Byte() = New Byte(3) {}
If smartModbus1.ReadHoldingRegister(20, 2, temp) = SmartModbus.EXCEPTIONCODE.SUCCESS Then
  labComStatus.Text = " 읽기 성공: " + SmartSerialPort.ConverByteToHexDecString(bRecv, "  ")
End If
```

메소드(함수) WriteSingleCoil, WriteSingleRegister, WriteMultipleCoil, WriteMultipleRegister

Write 명령(Function Code) 관련 동작을 수행합니다.

참고 "RS485 & Modbus Protocol 설명" 내용을 참고하시기 바랍니다.

- WriteSingleCoil() : Function code 5(0x05). Write Single Coil의 기능을 수행합니다.
- WriteSingleRegister() : Function code 6(0x06). Write Single Register의 기능을 수행합니다.
- WriteMultipleCoil() : Function code 15(0x0F). Write Multiple Coil의 기능을 수행합니다.
- WriteMultipleRegister() : Function code 16(0x10). Write Multiple Register의 기능을 수행합니다.

◉ SmartModbus.EXCEPTIONCODE WriteSingleCoil(int iStartAddress, int iWriteData)

◉ SmartModbus.EXCEPTIONCODE WriteSingleRegister(int iStartAddress, int iWriteData)

◉ SmartModbus.EXCEPTIONCODE WriteMultipleCoil(int iStartAddress, int iDataLength, byte[] iWriteDatas)

◉ SmartModbus.EXCEPTIONCODE WriteMultipleCoil(int iStartAddress, int iNumberOfCoil, int iByteCount, byte[] iWriteDatas)

◉ SmartModbus.EXCEPTIONCODE WriteMultipleRegister(int iStartAddress, int iDataLength, byte[] iWriteDatas)

◉ SmartModbus.EXCEPTIONCODE WriteMultipleRegister(int iStartAddress, int iNumberOfRegister, int iByteCount, byte[] iWriteDatas)

[인자]

- `int iStartAddress` : 시작 주소값
- `int iDataLength` : 데이터 개수
- `int iWriteDatas` : 쓰기 데이터 배열
- `int iWriteData` : 쓰기 데이터
- `int iNumberOfCoil` : 코일 개수
- `int iByteCount` : 총 바이트 개수
- `int iNumberOfRegister` : 레지스터 개수
- `byte[] retDatas` : 응답 데이터 (통상적 쓰기 계열의 명령에서는 iWriteDatas/iWriteData와 같은 값 또는 데이터 개수를 리턴)

[리턴값]

- `SmartModbus.EXCEPTIONCODE` : 명령 수행 결과

열거값	설명	열거값	설명
SUCCESS	정상 처리 완료	READBUFFER_OVER	수신 버퍼 오버됨
ILLEGAL_FUNCTION	잘못된 기능 코드	READDATA_ERROR	비정상적인 수신 데이터
ILLEGAL_DATA_ADDRESS	잘못된 데이터 주소	SLAVEADDRESS_ERROR	Slave Address 에러
ILLEGAL_DATA_VALUE	잘못된 데이터	READDATACOUNT_ERROR	읽어들인 데이터의 수가 잘못됨
SLAVE_DEVICE_FAILURE	Slave 장치 오류	CRC16_ERROR	내부 CRC16 검사 에러
ACKNOWLEDGE	수신 응답 오류를 연장	READTIMEOUT_ERROR	수신 응답 시간 초과
SLAVE_DEVICE_BUSY	Slave 처리 중	-	

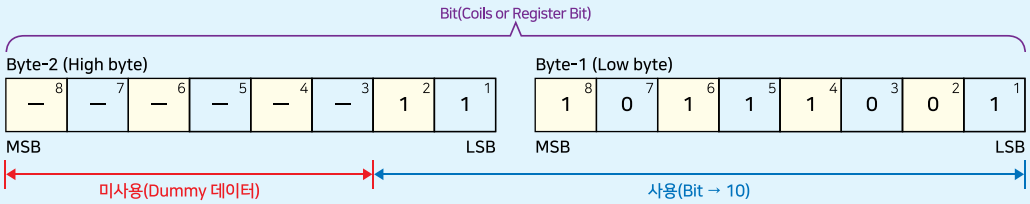
주의 iByteCount와 iWriteDatas의 크기는 반드시 같아야 합니다.

참고 코일(Coil)과 레지스터(Register)의 차이점

Modbus 프로토콜에서 데이터를 Read/Write 하는 대상은 크게 Coil과 Register 2가지입니다. 각각의 Read/Write Function의 프레임은 비슷하거나 동일하지만, 데이터의 처리 단위가 서로 달라 데이터 처리 시 약간의 차이점이 발생합니다.

1. Coil의 경우 데이터를 Bit 단위로 처리하지만, Modbus 프로토콜상 데이터의 처리 단위가 Byte이므로 데이터의 일부 Bit가 사용하지 않는 Dummy 데이터일 수 있습니다. 예를 들어 FC=01(Read Coil Status)이고, 10개의 Coil(iData Length = 10) 상태를 읽는 경우. Slave에서 Response하는 Byte 크기는 2Byte(16Bit)가 됩니다. 즉, Response하는 16Bit 데이터 중 6Bit는 사용하지 않는 값(Dummy 데이터)이 됩니다.

※ "Byte 수 = Math.Ceiling(Bit 수 / 8)" 이며, 여기서 Bit 수는 Coil의 개수 또는 Register Bit의 개수를 의미합니다.



2. Register의 경우 데이터를 Word 단위(1개 Register당 2Byte)로 처리합니다. 예를 들어 FC=03(ReadHolding Registers)이고, 2개의 Register(iDataLength = 2) 상태를 읽는 경우. Slave에서 Response하는 Byte 크기는 4Byte(2 X 2)가 됩니다.

C# 사용법

```
// 34번 주소의 코일에 0을 씀
SmartModbus.EXCEPTIONCODE bResult = smartModbus1.WriteSingleCoil(34, 0);
// 34번 주소의 레지스터에 100을 씀
bResult = smartModbus1.WriteSingleRegister(34, 100);
byte[] writeData = new byte[2];  // 10Bit를 쓰기 위해 2Byte 배열 생성
writeData[0] = 0xCD;  // 1100 1101
```

```
writeData[1] = 0x03;  // 0000 0011 <- 2Byte(14Bit) 중 사용하지 않는 데이터는 0으로 처리
// 20번 주소부터 14개의 코일에 데이터를 씀
bResult = smartModbus1.WriteMultipleCoil(20, 14, writeData.Length, writeData);
writeData = new byte[8];  // 전송할 데이터 배열
writeData[0] = 0x00; writeData[1] = 0xFF; writeData[2] = 0x00; writeData[3] = 0x20;
writeData[4] = 0x00; writeData[5] = 0x50; writeData[6] = 0x00; writeData[7] = 0xFF;
// 1000번 주소부터 4개의 레지스터에 데이터를 씀
bResult = smartModbus1.WriteMultipleRegister(1000, 4, writeData.Length, writeData);
if (bResult == SmartModbus.EXCEPTIONCODE.SUCCESS)
{
  // 쓰기 성공에 따른 처리 코드 작성
}
```

VB 사용법

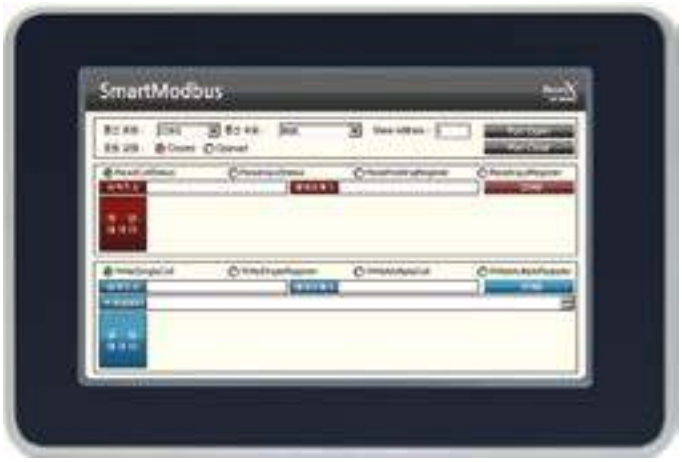
```
Dim bResult As SmartModbus.EXCEPTIONCODE = smartModbus1.WriteSingleCoil(34, 0)
bResult = smartModbus1.WriteSingleRegister(34, 100)
Dim writeData() As Byte = New Byte() {&HCD, &H3}
bResult = smartModbus1.WriteMultipleCoil(20, 14, writeData.Length, writeData);
writeData() As Byte = New Byte() {&H0, &HFF, &H0, &H20, &H0, &H50, &H0, &HFF}
bResult = smartModbus1.WriteMultipleRegister(1000, 4, writeData.Length, writeData);
If bResult = SmartModbus.EXCEPTIONCODE.SUCCESS Then
End If
```

5) SmartModbus 예제 사용하기

SmartModbus를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartModbus"



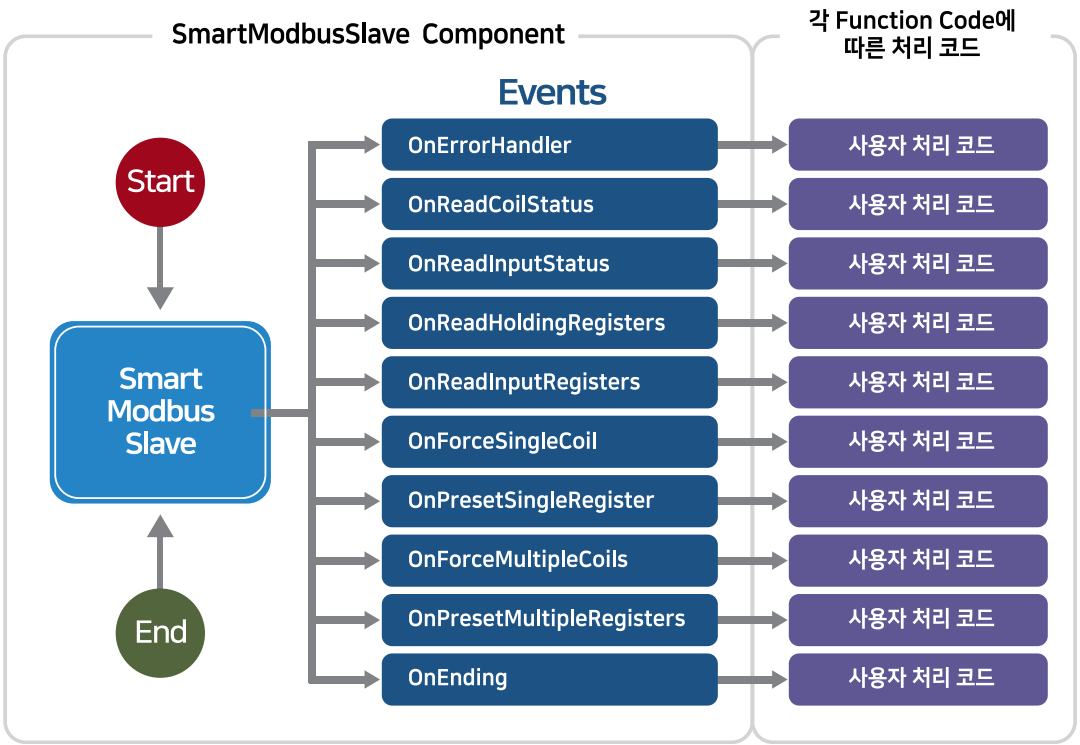
6. SmartModbusSlave

Modbus는 전 세계적으로 널리 보급되어 있는 자동화 분야에서 주로 사용되는 프로토콜이며, 현재 PLC, 제어 시스템, HMI, 계측기 등 수많은 산업용 장비들의 통신 표준으로 사용되고 있습니다. SmartModbusSlave는 IEC-Series 제품에서 Slave 측 기능을 지원하며 Event 방식으로 처리하도록 설계되어 Modbus 사용을 편리하게 할 수 있는 컴포넌트입니다.

Modbus 프로토콜은 Master-Slave 기법을 사용합니다. 즉 Master가 Slave에게 데이터를 요청하고 Slave는 Master에 응답하는 구조로 되어있습니다. Master에서는 모든 Slave에게 정보를 요청할 수도 있고 Slave에는 각자의 Address가 지정돼있어 Master에서 요청한 Address를 확인하여 Address가 일치하면 요청에 응답하고 아니면 무시합니다. Modbus 프로토콜의 메시지(데이터) 전송 방식은 크게 ASCII(American Standard Code for Information Interchange) 모드와 RTU(Remote Terminal Unit) 모드로 나눌 수 있는데 ASCII 모드는 데이터를 ASCII 형태로 구성하는 것이며, RTU 모드는 데이터를 Binary 형태로 구성하는 것입니다. 현재 SmartModbus와 SmartModbusSlave는 RTU 방식만 지원합니다.

- Modbus-Slave 측 기능 지원
- RTU 모드 방식
- Read Function (Function : 1 ~ 4) 지원
- Write Function (Function : 5, 6, 15, 16) 지원
- 송/수신 모드 전환 시간 설정 지원(SetResponseInterval() 메소드)

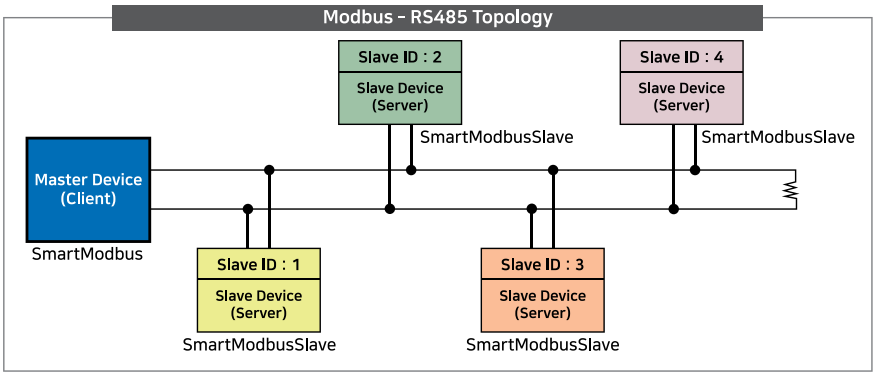
참고 SmartModbusSlave 적용 시 어려움이 있는 경우, IEC-Series 제품과 특정 I/O 보드를 구성하여 보내주시면 호환성을 검증해드리도록 하겠습니다. 또한 적용 시 발생하는 문제를 HNS에 알려주 시면 조치해 드리도록 하겠습니다. Modbus를 지원하는 산업용 장비가 다양하여 일일이 대응하지 못한 점 양해 바랍니다.



[그림] SmartModbusSlave 구조

1) RS485 & Modbus Protocol 설명

Modbus 프로토콜은 RS485 통신을 기반으로 하며, 다음과 같은 Topology Group을 기준으로 구성됩니다. SmartModbusSlave는 미리 정의된 Function을 이용해 Master로부터 Slave로 Request 하면 Slave는 Master로 수신 받은 Function에 해당하는 데이터를 Response 하는 구조로 되어있으며, 이때 Request에 따른 Response 데이터가 잘못되거나 올바르게 수신되지 않는 예외가 발생하는 경우 Master는 해당 예외에 따른 Exception Code를 사용자에게 리턴 합니다. SmartModbusSlave를 사용하시면 별도로 Data Frame을 구성할 필요 없이 해당 Function의 이벤트에서 간단히 Function에 따른 Data Frame을 전송할 수 있습니다. 아래 표에서 내용을 확인하시기 바랍니다.



참고 RS485-Modbus Master and Slave 송/수신 규약

1. Master-Device는 1개만 존재한다.
2. Master-Device 데이터를 임의로 송/수신할 수 있으며 반드시 송신(Request) → 수신(Response)의 구조로 통신되어야 한다.
3. Slave-Device는 여러 개가 존재하며 각각 Slave Address를 가지고 있다.
4. Slave-Device는 데이터를 임의로 송신할 수 없으며, Master-Device의 Request 데이터 수신 시 SlaveAddress가 일치하는 Slave-Device 만이 Response 데이터를 송신할 수 있다.
5. 모든 Slave-Device는 기본적으로 수신(대기) 상태를 유지한다.

[표1] SmartModbusSlave에서 지원하는 Read 관련 Function Code 및 Data Frame 구조

Function Code	Name	Description (업체별 정의가 다를 수 있음)
01	Read Coil Status	Digital Output 접점 읽기
02	Read Input Status	Digital Input 접점 읽기
03	Read Holding Registers	Analog Output 데이터 읽기
04	Read Input Registers	Analog Input 데이터 읽기

※ Request[Master -> Slave] Data Frame 구조

Slave Address	Function Code	Start Address		Data Length		CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	상위	하위	상위	하위
		데이터 주소		읽을 데이터 개수		CRC16	

※ Response[Slave -> Master] Data Frame 구조

Slave Address	Function Code	Byte Count	Read Datas				CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	···	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	데이터 바이트 수	1번째 데이터	2번째 데이터	...	N번째 데이터	상위	하위
							CRC16	

[표2] SmartModbusSlave에서 지원하는 Write(Single) 관련 Function Code 및 Data Frame 구조

Function Code	Name	Description (업체별 정의가 다를 수 있음)
05	Force Single Coil(Write Single Coil)	단일 Output 접점 쓰기
06	Preset Single Register(Write Single Register)	단일 Output 데이터 쓰기

참고 Single Coil/Register Write의 경우 Request와 Response의 Frame 구조는 같습니다.

※ Request[Master –〉 Slave] Data Frame 구조

Slave Address	Function Code	Start Address		Write Data	CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	쓰기 데이터	상위	하위
		데이터 주소			CRC16	

※ Response[Slave –〉 Master] Data Frame 구조

Slave Address	Function Code	Start Address		Write Data	CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	쓰기 데이터	상위	하위
		데이터 주소			CRC16	

[표3] SmartModbusSlave에서 지원하는 Write(Multiple) 관련 Function Code 및 Data Frame 구조

Function Code	Name	Description (업체별 정의가 다를 수 있음)
15	Force Multiple Coil(Write Multiple Coil)	여러 Output 접점 쓰기
16	Preset Multiple Registers(Write Multiple Registers)	여러 Output 데이터 쓰기

※ Request[Master –〉 Slave] Data Frame 구조

Slave Address	Function Code	Start Address		NumberOfCoil/ NumberOfRegister		Byte Cnt	Write Datas			CRC16 (내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	…	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	상위	하위	바이트 개수	1번째 데이터	…	n번째 데이터	상위	하위
		데이터 주소		쓰기 코일(레지스터) 개수						CRC16	

※ Response[Slave –〉 Master] Data Frame 구조

Slave Address	Function Code	Start Address		NumberOfCoil/ NumberOfRegister		CRC16(내부 자동 처리)	
8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	메소드의 기능 코드	상위	하위	상위	하위	상위	하위
		데이터 주소		쓰기 코일(레지스터) 개수		CRC16	

[표4] SmartModbus에서 지원하는 Exception Code 및 Data Frame 구조

Exception Code	Name	Description (업체별 정의가 다를 수 있음)
0	SUCCESS	성공
1	ILLEGAL_FUNCTION	잘못된 기능 코드 사용
2	ILLEGAL_DATA_ADDRESS	잘못된 주소 지정
3	ILLEGAL_DATA_VALUE	잘못된 값 설정
224	READDATA_ERROR	비정상적인 수신 데이터
255	SLAVEADDRESS_ERROR	송신 데이터의 Slave Address와 수신 데이터의 Slave Address가 일치하지 않음
226	READDATACOUNT_ERROR	Read관련 Function에서 실제 읽어 들인 데이터의 수가 Byte count와 일치하지 않음
227	CRC16_ERROR	수신 데이터의 CRC16 값과 계산된 CRC16 값이 일치하지 않음

※ Response[Slave -> Master] Data Frame 구조(MSB(Most Significant Bit) : 최상위 Bit)

SlaveAddress	FunctionCode	eExceptionCode	CRC16(내부자동처리)	
8Bit	8Bit	8Bit	8Bit	8Bit
Slave 장비 번호	MSB를 1로 OR 연산한 기능 코드	발생한 Exception 코드	상위	하위
			CRC16	

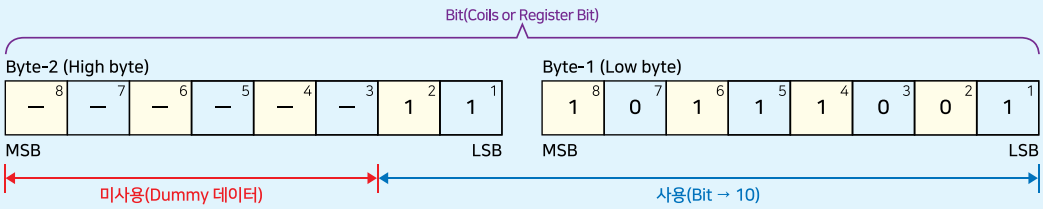
참조 "SimplyModbus(www.simplymodbus.ca)" 홈페이지를 참조하시기 바랍니다.

참고 코일(Coil)과 레지스터(Register)의 차이점

Modbus 프로토콜에서 데이터를 Read/Write 하는 대상은 크게 Coil과 Register 2가지입니다. 각각의 Read/Write Function의 프레임은 비슷하거나 동일하지만, 데이터의 처리 단위가 서로 달라 데이터 처리 시 약간의 차이점이 발생합니다.

1. Coil의 경우 데이터를 Bit 단위로 처리하지만, Modbus 프로토콜상 데이터의 처리 단위가 Byte이므로 데이터의 일부 Bit가 사용하지 않는 Dummy 데이터일 수 있습니다. 예를 들어 FC=01(Read Coil Status)이고, 10개의 Coil(iData Length = 10) 상태를 읽는 경우. Slave에서 Response하는 Byte 크기는 2Byte(16Bit)가 됩니다. 즉, Response하는 16Bit 데이터 중 6Bit는 사용하지 않는 값(Dummy 데이터)이 됩니다.

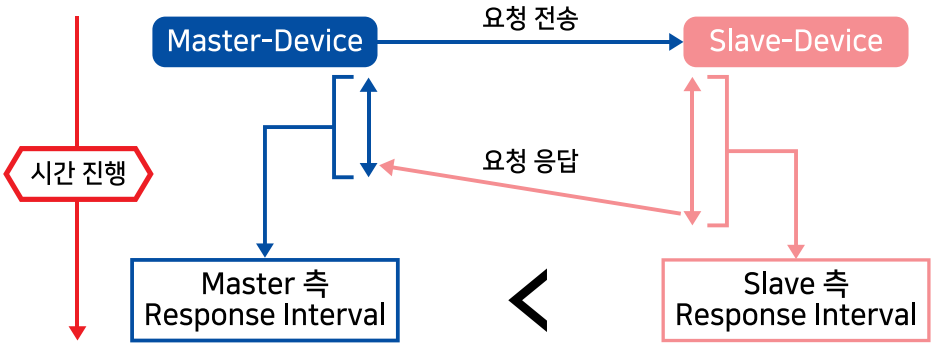
※ "Byte 수 = Math.Ceiling(Bit 수 / 8)" 이며, 여기서 Bit 수는 Coil의 개수 또는 Register Bit의 개수를 의미합니다.



2. Register의 경우 데이터를 Word 단위(1개 Register당 2Byte)로 처리합니다. 예를 들어 FC=03(ReadHolding Registers)이고, 2개의 Register(iDataLength = 2) 상태를 읽는 경우. Slave에서 Response하는 Byte 크기는 4Byte(2 X 2)가 됩니다.

2) Master와 Slave-Device의 Response Interval 관계

Modbus는 RS485 기반의 프로토콜이므로 Half-Duplex 방식의 통신을 합니다. 따라서 송신과 수신을 동시에 할 수 없으며, 만약 데이터 송신 중에 데이터를 수신받게 된다면 데이터의 충돌이 발생하게 됩니다. 따라서 Master-Device의 요청 전송에 따른 Slave-Device의 요청 응답에는 일정 시간만큼의 대기 시간이 필요하게 됩니다. 아래 표에서 내용을 확인하시기 바랍니다.



2-1) Master-Device 와 Slave-Device 의 ResponseInterval 값 설정 기준 및 방법

1 Master-Device의 ResponseInterval은 1차적으로 Slave-Device의 ResponseInterval 값에 의해 결정됩니다.

Master-Device의 ResponseInterval은 Slave-Device의 요청 응답 시간(ResponseInterval)에 맞게 설정해야 합니다.

2 Slave-Device의 ResponseInterval 값은 Master-Device의 ResponseInterval 값보다 크게 설정합니다. (Slave-Device ResponseInterval 〉 Master-Device ResponseInterval)

Master-Device에서 요청 전송 후 Slave-Device는 요청 응답 처리 시 Master-Device가 수신받을 준비가 되기 전에 응답을 전송할 경우 데이터의 충돌이 발생할 수 있습니다. 따라서 Master-Device가 수신 상태에서 요청 응답을 수신 받을 수 있도록 Slave-Device의 ResponseInterval 값은 Master-Device의 ResponseInterval 값보다 크게 설정해야 합니다.

3 RS485 H/W 자동 제어 방식은 RS485 S/W 자동 제어 방식보다 상대적으로 ResponseInterval 값을 작게 사용할 수 있습니다. (H/W 방식이 S/W 방식보다 빠르게 응답 가능)

H/W 자동 제어 방식은 H/W 단에서 송/수신 모드를 변경하여 S/W 자동 제어 방식보다 ResponseInterval 값을 작게 사용할 수 있습니다.

※ 아래의 "H/W 방식과 S/W 방식의 차이점 및 IEC-Series별 송/수신 제어 방식"을 참고하시기 바랍니다.

4 ReponseInterval은 Slave-Device와 Master-Device 양쪽 모두 설정되어야 합니다.

Master-Device의 ResponseInterval은 송/수신 모드의 전환 중에 요청 응답 수신을 방지하기 위해 설정해야 하며, Slave-Device의 ResponseInterval은 Master-Device의 송/수신 모드 전환 중에 요청 응답 송신을 방지하기 위해 설정해야 합니다. 또한 ResponseInterval은 충분한 테스트를 거쳐 값을 설정해야 합니다.

참고 H/W 방식과 S/W 방식의 차이점 및 IEC-Series별 송/수신 제어 방식

IEC-Series별로 송/수신 제어 방식의 차이가 있습니다. 아래 표를 확인하시기 바랍니다.

[표] H/W 방식과 S/W 방식의 차이점 및 IEC-Series별 송/수신 제어 방식

구분	H/W 방식		S/W 방식			
특징	Request 수신에 대한 Response 응답을 S/W 방식에 비해 빠르게 할 수 있음		Request 수신에 대한 Response 응답이 H/W 방식에 비해 느림			
IEC - Series	IEC266-Series		IEC667-Series		IEC1000-Series	
	非Lite	Lite	非Lite	Lite	非Lite	Lite
송/수신 제어 방식	H/W	S/W	H/W	S/W	H/W	S/W

3) Function별 Master 요청에 따른 Event 코드 작성 과정

다음과 같은 STEP을 참고하여 구현하시기 바랍니다.

STEP-1 요청된 대상 Slave가 맞는지 확인

STEP-2 요청된 정보에 문제가 없는지 확인

만약 문제가 있는 경우 잘못된 데이터를 수신하여 예외 발생 시 Master-Device로 Exception Code를 전송합니다.

참고 해당 내용은 "4) 프로그래밍 적용 가이드 – [STEP-3]"의 내용을 참고하시기 바랍니다.

STEP-3 datas.RequestDatas 필드의 요청 정보를 확인하여 응답할 데이터를 구성

STEP-4 구성된 응답 정보를 datas.ResponseDatas 필드에 채워 넣는 것으로 처리 완료

4) 프로그래밍 적용 가이드

STEP-1 통신 관련 설정값 적용 및 시작하기

Modbus-Slave는 Master의 요청에 의해서만 응답을 처리하는 형태이며 이를 위하여 통신 속도, Slave 주소, Response Interval 값들을 설정합니다.

※ Response Interval 값은 Master 측과 연동하여 설정하시기 바라며 자세한 내용은 "Master-Device와 Slave-Device의 ResponseInterval 값 설정 기준 및 방법"을 참고하시기 바랍니다.

```
smartModbusSlave1.Buad_Rate = SmartX.SmartModbusSlave.BUADRATE.CBR_9600;  // 통신 속도 설정
smartModbusSlave1.PortNo = SmartX.SmartModbusSlave.COMPORTNO.COM1;        // 통신 포트 설정
smartModbusSlave1.ProtocolType = SmartX.SmartModbusSlave.PROTOCOL.RTU;    // 프로토콜 타입 설정
smartModbusSlave1.SlaveAddress = 1;                                       // Slave 주소 설정
smartModbusSlave1.StopBits = System.IO.Ports.StopBits.One;  // StopBits 설정
smartModbusSlave1.Parity = System.IO.Ports.Parity.None;     // Parity 설정
smartModbusSlave1.SetResponseInterval(1000);                // 모드 전환 시간 설정
// SmartModbusSlave 시작
if (smartModbusSlave1.IsOpen == false)
{
  smartModbusSlave1.Start();
}
```

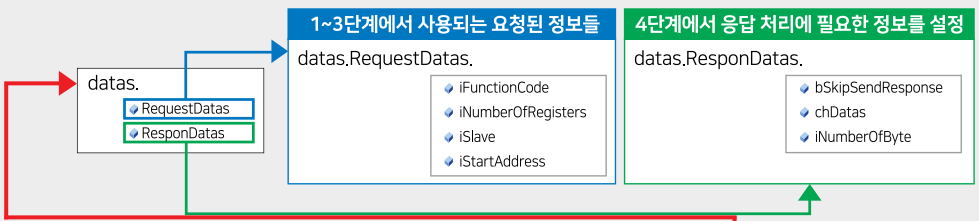
STEP-2 Function 수신에 따른 Event 처리 방법

Master로부터 요청에 따른 수신이 발생할 경우 SmartModbusSlave는 요청에 해당하는 이벤트를 발생합니다. 이벤트는 Function Code 마다 각각 다른 이벤트를 발생하며 이벤트의 인자를 통하여 요청된 정보도 함께 전달됩니다. 따라서 전달된 인자의 정보에 해당되는 내용의 정보를 구성하여 이 정보를 다시 응답 인자에 설정하여 처리하는 구조로 되어있습니다.
정리하면 다음의 단계와 같이 코드를 구현하시기 바라며 다른 Function Code의 이벤트 코드도 이와 유사한 방식으로 처리하시면 됩니다.

중요

[1 단계] 요청된 대상 Slave가 맞는지 확인
[2 단계] 요청된 정보에 문제가 없는지 확인 (만약 문제가 있는 경우 STEP-3 내용을 참고)
[3 단계] datas.RequestDatas 필드의 요청 정보를 확인하여 응답할 데이터를 구성
[4 단계] 구성된 응답 정보를 datas.ResponseDatas 필드에 채워 넣는 것으로 처리 완료

```
private byte[] m_SourceByteDatas = new byte[1000];  // 데이터를 수신 받을 변수
private bool IsCheckRequest(SmartX.SmartModbusSlave.MODBUSREQUESTDATAS_FC03 RequestDatas)
{
  // 수신된 정보를 검증 처리한다. 코드 생략됨
  return true;  // or false
}
```




```
// Read Holding Registers (FC=03) 요청에 의해 발생되는 이벤트
private void smartModbusSlave1_OnReadInputRegisters (SmartX.SmartModbusSlave.READINPUTREGISTERS datas)
{
  // [1 단계] 요청된 대상 Slave가 맞는지 확인
```

```
  if (datas.RequestDatas.iSlave == smartModbusSlave1.SlaveAddress)
  {
    // [2 단계] 요청된 정보에 문제가 없는지 확인한다.
    if (IsCheckRequest(datas.RequestDatas) == false)
    {
      datas.ResponDatas.bSkipSendResponse = true;
      // 예외 응답 처리 코드 생략됨 -> STEP-3 코드 참고
      // 발생 예외(Exception)을 Master-Device로 전송
      smartModbusSlave1.SendExceptionResponse(retException);
      return;
    }
    // [3 단계] datas.RequestDatas 필드의 요청 정보를 확인하여 응답할 데이터를 구성한다.
    byte[] retData = new byte[datas.RequestDatas.iNumberOfRegisters * 2];
    for (int i = 0; i < retData.Length; i++)
    {
      retData[i] = m_SourceByteDatas[datas.RequestDatas.iStartAddress + i];
    }
    // [4 단계] 구성된 응답 정보를 datas.ResponseDatas 필드에 채워 넣는것으로 처리 완료
    datas.ResponDatas.iNumberOfByte = datas.RequestDatas.iNumberOfRegisters;
    datas.ResponDatas.chDatas = retData;
  }
}
```

STEP-3 예외(잘못된 데이터 수신) 발생 시 Master-Device로 Exception Code를 전송하기

Master-Device로부터 수신받은 데이터가 잘못된 경우, 사용자는 Master-Device로 잘못된 데이터가 Response 되지 않도록 설정해야 하며, 사용자가 직접 EXCEPTIONRESPONSEINFO 열거값을 구성하여 Master-Device로 전송해야 합니다.

※ Exception Code 및 Function Code 관련하여 자세한 내용은 "RS485 & Modbus Protocol 설명"을 참고하시기 바랍니다.

[Exception Code 전송을 위한 설정값]

- `datas.ResponDatas.bSkipSendResponse` : Master-Device로 잘못된 데이터가 Response 되지 않도록 반드시 true로 설정해야 함
- `SmartModbusSlave.EXCEPTIONRESPONSEINFO` : Master-Device로 전송할 Exception 정보
  - `SmartModbusSlave.EXCEPTIONCODE eExceptionCode` : 발생한 오류에 따른 Exception Code
  - `SmartModbusSlave.FUNCTIONCODE eFuncCode` : 오류가 발생한 Function. 반드시 MSB를 0x80과 Or 연산해야 함
  - `int iSlaveAddress` : 오류가 발생한 SlaveAddress

```
// 예외(잘못된 데이터 수신) 발생 시 사용자가 직접 Exception Code를 구성하여 Master-Device로 전송
// 생략됨
// Master-Device로 전송할 Exception Code 정보
SmartX.SmartModbusSlave.EXCEPTIONRESPONSEINFO retException;
retException = new SmartX.SmartModbusSlave.EXCEPTIONRESPONSEINFO();
// Slave Address를 설정
retException.iSlaveAddress = datas.RequestDatas.iSlave;
// Function Code와 MSB 0x80을 OR 연산
retException.eFuncCode = ((SmartX.SmartModbusSlave.FUNCTIONCODE)(0x80)) | SmartX.SmartModbusSlave.
                                                                    FUNCTIONCODE.READ_COIL_STATUS;
// Exception Code를 설정
retException.eExceptionCode = SmartX.SmartModbusSlave.EXCEPTIONCODE.ILLEGAL_DATA_ADDRESS;
// 발생 예외(Exception)을 Master-Device로 전송
smartModbusSlave1.SendExceptionResponse(retException);
```

STEP-4 프로그램 종료 시 SmartModbusSlave 종료하기

SmartModbusSlave는 스레드로 동작하므로 프로그램 종료 시 스레드를 종료하지 않는다면, 리소스에 남게 되어 정상적으로 종료되지 않을 수 있습니다. 따라서 프로그램 종료 시 SmartModbusSlave의 동작 여부를 확인하여 동작 중이라면 반드시 종료해야 합니다.

```
// Form_Closing Event에서 SmartModbusSlave의 동작 여부를 확인
private void Form1_Closing(object sender, CancelEventArgs e)
{
  // 현재 SmartModbusSlave가 동작 중이라면 동작을 중지시킴
  if (smartModbusSlave1.IsOpen == true)
  {
    smartModbusSlave1.End();
  }
}
```

5) SmartModbusSlave 인터페이스 설명

SmartModbusSlave Component Interface		
속성		
bSkipSendResponse : bool	Buad_Rate : SmartModbusSlave.BUADRATE	IsOpen : bool
IsStart : bool	Parity : Parity	PortNo : SmartModbusSlave.COPORTNO
ProtocolType : SmartModbusSlave.PROTOCOL	SlaveAddress : int	StopBits : StopBits
메소드		
End() : void	SendExceptionResponse(SmartModbusSlave.EXCEPTIONRESPONSEINFO sendResp onse) : void	SetResponseInterval(int iResponseInterval) : void
Start() : void		
이벤트		
OnEnding : SmartModbusSlave.EndingHandler	OnErrorHandler : SmartModbusSlave.ErrorCodeHandler	OnForceMultipleCoils : SmartModbusSlave.ForceMultipleCoilsHandler
OnForceSingleCoil : SmartModbusSlave.ForceSingleCoilHandler	OnPresetMultipleRegisters : SmartModbusSlave.PresetMultipleRegisterHandler	OnPresetSingleRegister : SmartModbusSlave.PresetSingleRegisterHandler
OnReadCoilStatus : SmartModbusSlave.ReadCoilStatusHandler	OnReadHoldingRegisters : SmartModbusSlave.ReadHoldingRegisterHandler	OnReadInputRegisters : SmartModbusSlave.ReadInputRegisterHandler
OnReadInputStatus : SmartModbusSlave.ReadInputStatusHandler		

프로퍼티(속성) bSkipSendResponse

예외 발생 시 잘못된 데이터가 Master-Device로 전송되지 않도록 Response 동작을 수행하지 않도록 합니다.
또한 예외가 발생하는 경우 SendExceptionResponse() 메소드를 사용하여 Master-Devcie로 Exception Code를 전송해야 합니다.

주의 예외 발생 시 주의사항

Slave-Device에서 잘못된 데이터를 수신하거나, 예외가 발생했을 때 bSkipSendResponse를 true로 설정하지 않으면 Master-Device로 잘못된 데이터가 Response 될 수 있으며, 이때는 Master-Device로 예외에 따른 Exception Code를 전송해야 합니다. 따라서 예외 발생 시 잘못된 데이터가 Masters-Device로 Response 되지 않도록 반드시 bSkipSendResponse를 true로 설정하여 설정한 후 SendExceptionResponse() 메소드로 직접 Exception Code를 구성하여 Master-Device로 Response합니다.

- bool : Master-Device로 Response 데이터 전송 여부
  - true : 전송하지 않음
  - false : 전송 함 (기본값)

C# 사용법

참고 SendExceptionResponse() 메소드를 참고하시기 바랍니다.

프로퍼티(속성) Buad_Rate

통신 속도(Baud Rate)를 설정합니다.

주의 485 지원 통신 속도(Baud Rate) 안내

IEC-Series의 485포트를 사용하여 송/수신 시 발생할 수 있는 문제점으로 9600, 19200, 38400, 57600, 115200 이외의 통신 속도(BaudRate)를 사용하는 경우 수신 측의 송/수신 Interval 문제가 발생할 수 있으므로 송/수신 (Request & Response) Interval을 조절하여 사용해야 합니다.

- SmartModbusSlave.BUADRATE.CBR_9600 : 통신 속도를 9600bps로 설정
- SmartModbusSlave.BUADRATE.CBR_19200 : 통신 속도를 19200bps로 설정
- SmartModbusSlave.BUADRATE.CBR_38400 : 통신 속도를 38400bps로 설정
- SmartModbusSlave.BUADRATE.CBR_57600 : 통신 속도를 57600bps로 설정
- SmartModbusSlave.BUADRATE.CBR_115200 : 통신 속도를 115200bps로 설정

C# 사용법

```
smartModbusSlave1.Buad_Rate = SmartModbusSlave.BUADRATE.CBR_19200;  // 19200bps로 설정
```

VB 사용법

```
smartModbusSlave1.Buad_Rate = SmartModbusSlave.BUADRATE.CBR_19200
```

프로퍼티(속성) IsOpen

현재 통신 Port가 Open 되었는지를 확인합니다.

- bool : 통신 Port Open 여부
  - true : 통신 Port가 열려있음
  - false : 통신 Port가 닫혀있음

C# 사용법

```
if (smartModbusSlave1.IsOpen == true)
{
  smartModbusSlave1.End();  // 통신 Port가 열려있는 경우 Port를 닫는다.
}
```

VB 사용법

```
If smartModbusSlave1.IsOpen = True Then
  smartModbusSlave1.End()
End If
```

프로퍼티(속성) IsStart

SmartModbusSlave가 데이터 수신 감지를 시작했는지 확인합니다.

- bool : Serial Data 감지 시작 여부
  - true : 시작됨
  - false : 중지됨

C# 사용법

```
if (smartModbusSlave1.IsStart == false)
{
  smartModbusSlave1.Start();  // Modbus Slave 통신 Port 처리 시작
}
```

VB 사용법

```
If (smartModbusSlave1.IsStart = False) Then
  smartModbusSlave1.Start()
End If
```

프로퍼티(속성) Parity

패리티 검사 프로토콜 방식을 설정합니다.

- System.IO.Ports.Parity.Even : Bit 집합의 합계가 짝수가 되도록 패리티 Bit를 설정
- System.IO.Ports.Parity.Mark : 패리티 Bit를 1로 설정된 상태로 유지
- System.IO.Ports.Parity.None : 패리티 검사를 수행하지 않음
- System.IO.Ports.Parity.Odd : Bit 집합의 합계가 홀수가 되도록 패리티 Bit를 설정
- System.IO.Ports.Parity.Space : 패리티 Bit를 0으로 설정된 상태로 유지

C# 사용법

```
smartModbusSlave1.Parity = System.IO.Ports.Parity.None;  // Parity 검사를 사용하지 않음으로 설정
```

VB 사용법

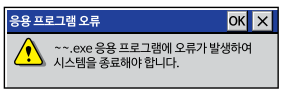
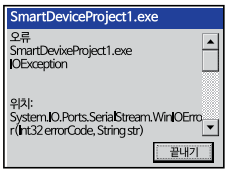
```
smartModbusSlave1.Parity = System.IO.Ports.Parity.None
```

프로퍼티(속성) PortNo

사용할 통신 Port 번호를 설정합니다. IEC-Series별로 내장된 통신 Port의 개수가 다르며, COM7의 경우 USB FTDI VCP 드라이버를 사용합니다.

주의 USB to Serial 컨버터(COM7) 사용 시 주의사항

USB to Serial 컨버터를 USB에 연결해 COM7로 Port 사용 시, 통신 중 USB 컨버터를 제거하지 마시기 바랍니다. 통신 중에 USB 컨버터를 제거하는 경우 다음과 같은 오류가 발생할 수 있습니다.

에러 유형 1	에러 유형 2
	

[표] IEC-Series별 지원 통신 Port

IEC-Series	COM1_1	COM1	COM2	COM3	COM4	COM7
IEC266	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	미지원	미지원
IEC266Lite	미지원 (RS485-옵션)	지원 (3.3V-TTL)	지원 (RS232)	지원 (RS232)	미지원	미지원
IEC667	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC667Lite	지원 (RS485)	지원 (5V-TTL)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)

IEC1000	지원 (5V-TTL)	지원 (RS485)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
IEC1000Lite	지원 (RS485)	지원 (5V-TTL)	지원 (RS232)	지원 (RS232)	지원 (5V-TTL)	지원 (USB)
Comport No.	COM1		COM2	COM3	COM4	COM7

- SmartModbusSlave.COMPORTNO.COM1 : COM1
- SmartModbusSlave.COMPORTNO.COM2 : COM2
- SmartModbusSlave.COMPORTNO.COM3 : COM3
- SmartModbusSlave.COMPORTNO.COM4 : COM4 (IEC266-Series 미지원)
- SmartModbusSlave.COMPORTNO.COM7 : COM7 (IEC266-Series 미지원)

C# 사용법

```
smartModbusSlave1.PortNo = SmartModbusSlave.COMPORTNO.COM1;  // COM1을 사용합니다.
```

VB 사용법

```
smartModbusSlave1.PortNo = SmartModbusSlave.COMPORTNO.COM1
```

프로퍼티(속성) ProtocolType

Modbus통신에서 데이터 표현 방식을 설정합니다. SmartModbusSlave는 RTU 방식만 지원합니다.

- SmartModbusSlave.PROTOCOL.RTU : 데이터 표현 방식을 RTU(Binary)로 설정

C# 사용법

```
smartModbusSlave1.ProtocolType = SmartModbusSlave.PROTOCOL.RTU;
```

VB 사용법

```
smartModbusSlave1.ProtocolType = SmartModbusSlave.PROTOCOL.RTU
```

프로퍼티(속성) SlaveAddress

Slave-Device(자신)의 주소(Address)를 설정합니다.

- int : Slave-Device 주소(Address)

C# 사용법

```
smartModbusSlave1.SlaveAddress = 1;  // Slave-Device 주소를 1로 설정
```

VB 사용법

```
smartModbusSlave1.SlaveAddress = 1
```

프로퍼티(속성) StopBits

Byte당 정지 Bit의 표준 개수를 가져오거나 설정합니다.

- System.IO.Ports.StopBits.None : 정지 Bit를 사용하지 않음
- System.IO.Ports.StopBits.One : 1Bit의 정지 Bit를 사용
- System.IO.Ports.StopBits.OnePointFive : 1.5Bit의 정지 Bit를 사용
- System.IO.Ports.StopBits.Two : 2Bit의 정지 Bit를 사용

C# 사용법

```
smartModbusSlave1.StopBits = System.IO.Ports.StopBits.One;  // 1Bit의 정지 Bit를 사용
```

VB 사용법

```
smartModbusSlave1.StopBits = System.IO.Ports.StopBits.One
```

메소드(함수) Start, End

• Start() : SmartModbusSlave를 시작합니다.
• End() : SmartModbusSlave를 종료합니다.

◉ void Start()
◉ void End()

C# 사용법

```
smartModbusSlave1.Start();
smartModbusSlave1.End();
```

VB 사용법

```
smartModbusSlave1.Start()
smartModbusSlave1.End()
```

메소드(함수) SetResponseInterval

송신 모드에서 수신 모드로의 전환 시간을 설정합니다.

참고 "Master와 Slave-Device의 Response Interval 관계" 내용을 참고하시기 바랍니다

◉ void SetResponseInterval(int iResponseInterval)

[인자]

• int iResponseInterval : 전환 시간 (단위 ms)

C# 사용법

```
smartModbusSlave1.SetResponseInterval(1000);  // 전환 시간을 1초로 설정
```

VB 사용법

```
smartModbusSlave1.SetResponseInterval(1000)
```

메소드(함수) SendExceptionResponse

사용자가 구성한 예외 응답 정보를 Master-Device로 전송합니다.

참고 Exception Code 및 Function Code 관련하여 자세한 내용은 "RS485 & Modbus Protocol 설명"을 참고하시기 바랍니다.

주의 예외 응답 정보를 Master-Device로 전송하기 전 주의사항

1. 반드시 bSkipSendResponse를 true로 설정하셔야 합니다.
bSkipSendResponse를 true로 설정하지 않고 SendExceptionResponse() 메소드를 이용하여 마스터로 예외 응답 정보를 전송할 경우 잘못된 데이터와, 예외 응답 정보가 모두 Master-Device로 전송됩니다.
2. 예외 코드임을 알려주기 위해 반드시 기능코드의 MSB(Most Significant Bit)를 1로 설정합니다.
3. 예외 응답 정보를 올바르게 설정합니다.

◉ void SendExceptionResponse(SmartModbusSlave.EXCEPTIONRESPONSEINFO sendResponse)

[인자]

• SmartModbusSlave.EXCEPTIONRESPONSEINFO sendResponse
– SmartModbusSlave.EXCEPTIONCODE eExceptionCode

열거값	설명	열거값	설명
SUCCESS	정상 처리 완료	READDATA_ERROR	비정상적인 수신 데이터
ILLEGAL_FUNCTION	잘못된 기능 코드	SLAVEADDRESS_ERROR	Slave Address 에러
ILLEGAL_DATA_ADDRESS	잘못된 데이터 주소	READDATACOUNT_ERROR	읽어들인 데이터의 수가 잘못됨
ILLEGAL_DATA_VALUE	잘못된 데이터	CRC16_ERROR	내부 CRC16 검사 에러

- SmartModbusSlave.FUNCTIONCODE eFuncCode

열거값	설명	열거값	설명
READ_COIL_STATUS	Function code 1	WRITE_SINGLE_COIL	Function code 5
READ_INPUT_STATUS	Function code 2	WRITE_SINGLE_REGISTER	Function code 6
READ_HOLDING_REGISTER	Function code 3	WRITE_MULTIPLE_COIL	Function code 15
READ_INPUT_REGISTER	Function code 4	WRITE_MULTIPLE_REGISTER	Function code 16

- int iSlaveAddress : Slave Address(자기 자신의 주소)

C# 사용법

```
private void smartModbusSlave1_OnReadCoilStatus(SmartModbusSlave.READCOILSTATUS datas)
{
  if (datas.RequestDatas.iSlave == smartModbusSlave1.SlaveAddress)
  {
    byte[] retChDatas = new byte[1];
    switch (datas.RequestDatas.iStartAddress)
    {
      case 10000:
        // 중략
      default:
        datas.ResponDatas.bSkipSendResponse = true;  // true인 경우 데이터를 보내지 않음
        SmartModbusSlave.EXCEPTIONRESPONSEINFO retException;
        retException = new SmartModbusSlave.EXCEPTIONRESPONSEINFO();
        retException.iSlaveAddress = datas.RequestDatas.iSlave;
        // MSB 0x80을 OR 연산
        retException.eFuncCode = ((SmartModbusSlave.FUNCTIONCODE)(0x80)) |
                                 SmartModbusSlave.FUNCTIONCODE.READ_COIL_STATUS;
        // ExceptionCode를 ILLEGAL_DATA_ADDRESS로 설정
        retException.eExceptionCode = SmartModbusSlave.EXCEPTIONCODE.ILLEGAL_DATA_ADDRESS;
        // 발생 예외(EXCEPTION)을 Master로 전송
        smartModbusSlave1.SendExceptionResponse(retException);
        break;
    }
  }
}
```

VB 사용법

```
Private Sub smartModbusSlave1_OnReadCoilStatus(ByVal datas As SmartModbusSlave.READCOILSTATUS)
  If datas.RequestDatas.iSlave = smartModbusSlave1.SlaveAddress Then
    Dim retChDatas As Byte() = New Byte(0) {}
    Select Case datas.RequestDatas.iStartAddress
      Case Else
        datas.ResponDatas.bSkipSendResponse = True
        Dim retException As SmartModbusSlave.EXCEPTIONRESPONSEINFO = New SmartModbusSlave.
                                                                     EXCEPTIONRESPONSEINFO()
        retException.iSlaveAddress = datas.RequestDatas.iSlave
        retException.eFuncCode = (CType((&H80), SmartModbusSlave.FUNCTIONCODE)
                                  Or SmartModbusSlave.FUNCTIONCODE.WRITE_MULTIPLE_REGISTER
        retException.eExceptionCode = SmartModbusSlave.EXCEPTIONCODE.ILLEGAL_DATA_ADDRESS
        smartModbusSlave1.SendExceptionResponse(retException)
    End Select
  End If
End Sub
```

이벤트 OnEnding

SmartModbusSlave 종료 시 호출되는 이벤트

C# 사용법

```
private void smartModbusSlave1_OnEnding()
{
  labStatus.Text = " SmartModbusSlave가 종료됨 ";
}
```

VB 사용법

```
Private Sub smartModbusSlave1_OnEnding() Handles smartModbusSlave1.OnEnding
 labStatus.Text = " SmartModbusSlave가 종료됨 "
End Sub
```

이벤트 OnErrorHandler

Master-Device로부터 수신된 데이터에 오류가 있는 경우 발생되는 이벤트입니다.

참고 Exception Code 관련하여 자세한 내용은 "RS485 & Modbus Protocol 설명"을 참고하시기 바랍니다.

◉ `void smartModbusSlave1_OnErrorHandler(SmartModbusSlave.EXCEPTIONCODE errCode, int iSlaveAddress, int iFunctionCode)`

[인자]

• `SmartModbusSlave.EXCEPTIONCODE errCode` : 발생한 오류 유형 정보

열거값	설명	열거값	설명
SUCCESS	정상 처리 완료	READDATA_ERROR	비정상적인 수신 데이터
ILLEGAL_FUNCTION	잘못된 기능 코드	SLAVEADDRESS_ERROR	Slave Address 에러
ILLEGAL_DATA_ADDRESS	잘못된 데이터 주소	READDATACOUNT_ERROR	읽어들인 데이터의 수가 잘못됨
ILLEGAL_DATA_VALUE	잘못된 데이터	CRC16_ERROR	내부 CRC16 검사 에러

• `int iSlaveAddress` : Master-Device가 보낸 대상 Slave 주소

• `int iFunctionCode` : Master-Device가 보낸 Function Code

C# 사용법

```
// Master로부터 수신된 데이터에 오류가 있는 경우 발생되는 Event
private void smartModbusSlave1_OnErrorHandler(SmartModbusSlave.EXCEPTIONCODE err Code, int
iSlaveAddress, int iFunctionCode)
{
  labRequest.Text = " - ";
  SmartModbusSlave.EXCEPTIONRESPONSEINFO retException;  // Exception Code를 전송하기 위한 변수
  retException = new SmartModbusSlave.EXCEPTIONRESPONSEINFO();
  retException.iSlaveAddress = iSlaveAddress; // Slave Address를 수신한 값으로 설정
  // Exception Code임을 알리기 위해 MSB를 1로 설정
  retException.eFuncCode = ((SmartModbusSlave.FUNCTIONCODE)(0x80)) |
                           (SmartModbusSlave.FUNCTIONCODE)(iFunctionCode);
  switch (errCode)
  {
    case SmartModbusSlave.EXCEPTIONCODE.CRC16_ERROR:
      labRequest2.Text = " CRC16_ERROR ";  // CRC16 에러
      break;
      // …중략…
  }
  retException.eExceptionCode = errCode;  // 송신할 Exception Code를 설정
```

```
  smartModbusSlave1.SendExceptionResponse(retException);  // Master로 Exception Frame을 전송
}
```

VB 사용법

```
Private Sub smartModbusSlave1_OnErrorHandler(ByVal errCode As SmartModbusSlave.EXCEPTIONCODE,
ByVal iSlaveAddress As System.Int32, ByVal iFunctionCode As System.Int32) Handles
smartModbusSlave1.OnErrorHandler
  labRequest.Text = " - "
  Dim retException As SmartModbusSlave.EXCEPTIONRESPONSEINFO
  retException = New SmartModbusSlave.EXCEPTIONRESPONSEINFO()
  retException.iSlaveAddress = iSlaveAddress
  retException.eFuncCode = (CType((&H80), SmartModbusSlave.FUNCTIONCODE)) Or
                           CType((iFunctionCode), SmartModbusSlave.FUNCTIONCODE)
  Select Case errCode
    Case SmartModbusSlave.EXCEPTIONCODE.CRC16_ERROR
      labRequest2.Text = " CRC16_ERROR "
      ' …중략…
  End Select
  retException.eExceptionCode = errCode
  smartModbusSlave1.SendExceptionResponse(retException)
End Sub
```

이벤트 OnForceMultipleCoils

Master-Device로부터 요청된 Function Code가 Force Multiple Coils(FC 15)일 경우 발생되는 이벤트입니다.

참고
1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

◉ OnForceMultipleCoils(SmartModbusSlave.FORCEMULTIPLECOILS datas)

[인자]

• SmartModbusSlave.FORCEMULTIPLECOILS datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ byte[] datas.RequestDatas.chDatas : Master-Device로부터 요청된 쓰기 데이터 바이트 배열 (여러 개의 Coil 쓰기 데이터)
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iNumberOfByte : Master-Device로부터 요청된 바이트 크기
  - ↳ int datas.RequestDatas.iNumberOfCoilRegister : Master-Device로부터 요청된 Coil의 개수
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
- datas.RequestDatas : Master-Device로 Response할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ int datas.RequestDatas.iNumberOfCoilRegister : Master-Device로 Response 할 Coil의 개수
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로 Response 할 시작 주소

C# 사용법

```
private byte[] m_SlaveByteDatas = new byte[1000];
private bool IsCheckRequest(SmartModbusSlave.MODBUSREQUESTDATAS_FC15 RequestDatas)
{
  // 요청된 정보를 검증 처리한다.
  // 코드 생략
```

```
  return true;  // or false
}

// Force Multiple Coils (FC=15) 요청에 의해 발생되는 이벤트
private void smartModbusSlave1_OnForceMultipleCoils(SmartModbusSlave.FORCEMULTIPLECOILS datas)
{
  // [1 단계] 요청된 대상 Slave가 맞는지 확인
  if (datas.RequestDatas.iSlave == smartModbusSlave1.SlaveAddress)
  {
    // [2 단계] 요청된 정보에 문제가 없는지 확인한다.
    if (IsCheckRequest(datas.RequestDatas) == false)
    {
       datas.ResponDatas.bSkipSendResponse = true;
       // 예외 응답 처리 코드
       // 생략됨 -> STEP-3 코드 참고
       // 발생 예외(Exception)을 Master-Device로 전송
       smartModbusSlave1.SendExceptionResponse(retException);
       return;
    }
    // [3 단계] datas.RequestDatas 필드의 요청 정보를 확인하여 응답할 데이터를 구성한다.
    double dByteCount;  // Response할 ByteCount를 저장할 변수

    // Coil의 데이터에 접근하므로 Bit 단위 처리를 하도록 해야함
    // iNumberOfCoilRegister 값을 8(8Bit = 1Byte)로 나눈 후 올림(나머지가 없도록)
    dByteCount = Math.Ceiling(datas.RequestDatas.iNumberOfCoilRegister / 8);

    // Response 할 데이터의 크기를 iByteCount 값으로 설정
    byte[] retData = new byte[Convert.ToInt32(dByteCount)];

    for (int i = 0; i < retData.Length; i++)
    {
      m_SlaveByteDatas[datas.RequestDatas.iStartAddress + i] = datas.RequestDatas.chDatas[i];
    }

    // [4 단계] 구성된 응답 정보를 datas.ResponseDatas 필드에 채워 넣는것으로 처리 완료
    datas.ResponDatas.iNumberOfCoilRegister = datas.RequestDatas.iNumberOfCoilRegister;
    datas.ResponDatas.iStartAddress = smartModbusSlave1.SlaveAddress;
  }
}
```

VB 사용법

```
Dim m_SlaveByteDatas As Byte() = New Byte(999) {}
Private Function IsCheckRequest(ByVal RequestDatas As SmartModbusSlave.MODBUSREQUESTDATAS_FC03)
As Boolean
   ' 요청된 정보를 검증 처리한다.
   ' 코드 생략
  Return True
   ' or false
End Function

Private Sub smartModbusSlave1_OnForceMultipleCoils(ByVal datas As SmartModbusSlave.FORCEMULTIPLE
COILS) Handles smartModbusSlave1.OnForceMultipleCoils
  If datas.RequestDatas.iSlave = smartModbusSlave1.SlaveAddress Then
    datas.ResponDatas.bSkipSendResponse = True
```

```
    ' 예외 응답 처리 코드
    ' 생략됨 -> STEP-3 코드 참고
    ' 발생 예외(Exception)을 Master-Device로 전송
    smartModbusSlave1.SendExceptionResponse(retException)
    Return
  End If
  Dim dByteCount As Double
  dByteCount = Math.Ceiling(datas.RequestDatas.iNumberOfCoilRegister / 8)
  Dim retData As Byte() = New Byte(Convert.ToInt32(dByteCount)) {}
  For i As Integer = 0 To retData.Length
    m_SlaveByteDatas(datas.RequestDatas.iStartAddress + i) = retData(i)
  Next
  datas.ResponDatas.iNumberOfCoilRegister = datas.RequestDatas.iNumberOfCoilRegister
  datas.ResponDatas.iStartAddress = smartModbusSlave1.SlaveAddress
End Sub
```

이벤트 OnForceSingleCoil

Master-Device로부터 요청된 Function Code가 Force Single Coil(FC 05)일 경우 발생되는 이벤트입니다.

참고
1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

⦿ OnForceSingleCoil(SmartModbusSlave.FORCESINGLECOIL datas)

[인자]

• SmartModbusSlave.FORCESINGLECOIL datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
  - ↳ int datas.RequestDatas.iWriteValue : Master-Device로부터 요청된 쓰기 데이터값 (Coil 쓰기 데이터 : 2Byte)
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로 Response 할 시작 주소
  - ↳ int datas.RequestDatas.iWriteValue : Master-Device로 Response 할 저장된(실제로 쓴) 데이터

C# 사용법

```
private byte[] m_SlaveByteDatas = new byte[1000];
private bool IsCheckRequest(SmartModbusSlave.MODBUSREQUESTDATAS_FC05 RequestDatas)
{
  // 요청된 정보를 검증 처리한다.
  // 코드 생략
  return true;  // or false
}

// Force Single Coil (FC=05) 요청에 의해 발생되는 이벤트
private void smartModbusSlave1_OnForceSingleCoil(SmartModbusSlave.FORCESINGLECOIL datas)
{
  // [1 단계] 요청된 대상 Slave가 맞는지 확인
  if (datas.RequestDatas.iSlave == smartModbusSlave1.SlaveAddress)
  {
```

```
    // [2 단계] 요청된 정보에 문제가 없는지 확인한다.
    if (IsCheckRequest(datas.RequestDatas) == false)
    {
      datas.ResponDatas.bSkipSendResponse = true;
      // 예외 응답 처리 코드
      // 생략됨 -> STEP-3 코드 참고
      // 발생 예외(Exception)을 Master-Device로 전송
      smartModbusSlave1.SendExceptionResponse(retException);
      return;
    }

    // [3 단계] datas.RequestDatas 필드의 요청 정보를 확인하여 응답할 데이터를 구성한다.
    // Request 받은 데이터를 Slave-Device에 적용
    m_SlaveByteDatas[datas.RequestDatas.iStartAddress] = Convert.ToByte(datas.
                                                         RequestDatas.iWriteValue);
    // [4 단계] 구성된 응답 정보를 datas.ResponseDatas 필드에 채워 넣는것으로 처리 완료
    datas.ResponDatas.iStartAddress = datas.RequestDatas.iStartAddress;
    datas.ResponDatas.iWriteValue = m_SlaveByteDatas[datas.RequestDatas.iStartAddress];
  }
}
```

VB 사용법

```
Dim m_SlaveByteDatas As Byte() = New Byte(999) {}
Private Function IsCheckRequest(ByVal RequestDatas As SmartModbusSlave.MODBUSREQUESTDATAS_FC05)
As Boolean
   ' 요청된 정보를 검증 처리한다.
   ' 코드 생략
  Return True
   ' or false
End Function
Private Sub smartModbusSlave1_OnForceSingleCoil(ByVal datas As SmartModbusSlave.FORCESINGLECOIL)
Handles smartModbusSlave1.OnForceSingleCoil
  If datas.RequestDatas.iSlave = smartModbusSlave1.SlaveAddress Then
    datas.ResponDatas.bSkipSendResponse = True
     ' 예외 응답 처리 코드
     ' 생략됨 -> STEP-3 코드 참고
     ' 발생 예외(Exception)을 Master-Device로 전송
    smartModbusSlave1.SendExceptionResponse(retException)
    Return
  End If
  m_SlaveByteDatas(datas.RequestDatas.iStartAddress) = Convert.ToByte(datas.RequestDatas.
                                                       iWriteValue)
  datas.ResponDatas.iStartAddress = datas.RequestDatas.iStartAddress
  datas.ResponDatas.iWriteValue = m_SlaveByteDatas(datas.RequestDatas.iStartAddress)
End Sub
```

이벤트 OnPresetMultipleRegisters

Master-Device로부터 요청된 Function Code가 Preset Multiple Registers(FC 16)일 경우 발생되는 이벤트입니다.

참고
1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

◉ OnPresetMultipleRegisters(SmartModbusSlave.PRESETMULTIPLEREGISTER datas)

[인자]

• SmartModbusSlave.PRESETMULTIPLEREGISTER datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ byte[] datas.RequestDatas.chDatas : Master-Device로부터 요청된 쓰기 데이터 바이트 배열 (Register 쓰기 바이트 배열)
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iNumberOfByte : Master-Device로부터 요청된 데이터의 바이트 크기
  - ↳ int datas.RequestDatas.iNumberOfCoilRegister : Master-Device로부터 요청된 Register의 개수
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ int datas.RequestDatas.iNumberOfCoilRegister : Master-Device로 Response 할 Register의 개수
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로 Response 할 시작 주소

사용법

참고 OnPresetMultipleRegisters의 사용법은 OnForceMultipleCoils 사용법과 크게 다르지 않으므로 "OnForce MultipleCoils 사용법"내용을 참고하시기 바랍니다.

이벤트 `OnPresetSingleRegister`

Master-Device로부터 요청된 Function Code가 Preset Single Register(FC 06)일 경우 발생되는 이벤트입니다.

참고
1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

◉ OnPresetSingleRegister(SmartModbusSlave.PRESETSINGLEREGISTER datas)

[인자]

• SmartModbusSlave.FORCESINGLECOIL datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장되어 있음
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
  - ↳ int datas.RequestDatas.iWriteValue : Master-Device로부터 요청된 쓰기 데이터값 (Register 쓰기 데이터 : 2Byte)
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로 Response 할 시작 주소
  - ↳ int datas.RequestDatas.iWriteValue : Master-Device로 Response 할 저장된(실제로 쓴) 데이터

사용법

참고 OnPresetSingleRegister의 사용법은 OnForceSingleCoil 사용법과 크게 다르지 않으므로 "OnForceSingle Coil 사용법"내용을 참고하시기 바랍니다.

이벤트 `OnReadCoilStatus`

Master-Device로부터 요청된 Function Code가 Read Coil Status(FC 01)일 경우 발생되는 이벤트입니다.

참고 1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

◉ OnReadCoilStatus(SmartModbusSlave.READCOILSTATUS datas)

[인자]

• SmartModbusSlave.READCOILSTATUS datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iNumberOfCoil : Master-Device로부터 요청된 Coil의 개수
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ byte[] datas.RequestDatas.chDatas : Master-Device로 Response 할 데이터
  - ↳ int datas.RequestDatas.iNumberOfByte : Master-Device로 Response 할 Coil 개수에 따른 바이트 크기

※ iNumberOfByte = Math.Ceiling(iNumberOfCoil / 8)

C# 사용법

```
private byte[] m_SourceByteDatas = new byte[1000];
private bool IsCheckRequest(SmartModbusSlave.MODBUSREQUESTDATAS_FC01 RequestDatas)
{
  // 요청된 정보를 검증 처리한다. 코드 생략
 return true;  // or false
}
// Read Coil Status (FC=01) 요청에 의해 발생되는 이벤트
private void smartModbusSlave1_OnReadCoilStatus(SmartModbusSlave.READCOILSTATUS datas)
{
  // [1 단계] 요청된 대상 Slave가 맞는지 확인
  if (datas.RequestDatas.iSlave == smartModbusSlave1.SlaveAddress)
  {
    // [2 단계] 요청된 정보에 문제가 없는지 확인한다.
    if (IsCheckRequest(datas.RequestDatas) == false)
    {
      datas.ResponDatas.bSkipSendResponse = true;
      // 예외 응답 처리 코드 생략됨 -> STEP-3 코드 참고
      // 발생 예외(Exception)을 Master-Device로 전송
      smartModbusSlave1.SendExceptionResponse(retException);
      return;
    }
    // [3 단계] datas.RequestDatas 필드의 요청 정보를 확인하여 응답할 데이터를 구성한다.
    double dByteCount;  // Response 할 ByteCount를 저장할 변수
    // Coil의 데이터에 접근하므로 Bit 단위 처리를 하도록 해야함
    // iNumberOfCoilRegister 값을 8(8Bit = 1Byte)로 나눈 후 올림(나머지가 없도록)
    dByteCount = Math.Ceiling(datas.RequestDatas.iNumberOfCoil / 8);
    // Response 할 데이터의 크기를 iByteCount 값으로 설정
    byte[] retData = new byte[Convert.ToInt32(dByteCount)];
    for (int i = 0; i < retData.Length; i++)
    {
      retData[i] = m_SourceByteDatas[datas.RequestDatas.iStartAddress + i];
    }
    // [4 단계] 구성된 응답 정보를 datas.ResponseDatas 필드에 채워 넣는 것으로 처리 완료
```

```
    datas.ResponDatas.iNumberOfByte = iByteCount;
    datas.ResponDatas.chDatas = retData;
  }
}
```

VB 사용법

```
Dim m_SourceByteDatas As Byte() = New Byte(999) {}
Private Function IsCheckRequest(ByVal RequestDatas As SmartModbusSlave.MODBUSREQUESTDATAS_FC01)
As Boolean
  ' 요청된 정보를 검증 처리한다. 코드 생략
  Return True
  ' or false
End Function
Private Sub smartModbusSlave1_OnReadCoilStatus(ByVal datas As SmartModbusSlave.READCOILSTATUS)
Handles smartModbusSlave1.OnReadCoilStatus
  If datas.RequestDatas.iSlave = smartModbusSlave1.SlaveAddress Then
    datas.ResponDatas.bSkipSendResponse = True
    ' 예외 응답 처리 코드 생략됨 -> STEP-3 코드 참고
    ' 발생 예외(Exception)을 Master-Device로 전송
    smartModbusSlave1.SendExceptionResponse(retException)
    Return
  End If
  Dim dByteCount As Double
  dByteCount = Math.Ceiling(datas.RequestDatas.iNumberOfCoil / 8)
  Dim retData As Byte() = New Byte(Convert.ToInt32(dByteCount)) {}
  For i As Integer = 0 To retData.Length
    m_SourceByteDatas (datas.RequestDatas.iStartAddress + i) = retData(i)
  Next
  datas.ResponDatas.iNumberOfByte = Convert.ToInt32(dByteCount)
  datas.ResponDatas.chDatas = retData
End Sub
```

이벤트 OnReadHoldingRegisters

Master-Device로부터 요청된 Function Code가 Read Holding Registers(FC 03)일 경우 발생되는 이벤트입니다.

참고
1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

⦿ OnReadHoldingRegisters(SmartModbusSlave.READHOLDINGREGISTERS datas)

[인자]

• SmartModbusSlave.READHOLDINGREGISTERS datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iNumberOfRegisters : Master-Device로부터 요청된 Register의 개수
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ byte[] datas.RequestDatas.chDatas : Master-Device로 Response 할 Register 바이트 배열 (Register Word 단위 배열)
  - ↳ int datas.RequestDatas.iNumberOfByte : Master-Device로 Response 할 데이터의 개수

사용법

참고 OnReadHoldingRegister 사용법은 "프로그래밍 적용 가이드"에 설명된 코드와 크게 다르지 않아 생략되었습니다. "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

이벤트 OnReadInputRegisters

Master-Device로부터 요청된 Function Code가 Read Input Registers(FC 04)일 경우 발생되는 이벤트입니다.

참고 1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

◉ OnReadInputRegisters(SmartModbusSlave.READINPUTREGISTERS datas)

[인자]

• SmartModbusSlave.READHOLDINGREGISTERS datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳int datas.RequestDatas.iNumberOfRegisters : Master-Device로부터 요청된 Register의 개수
  - ↳int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳byte[] datas.RequestDatas.chDatas : Master-Device로 Response 할 Register 바이트 배열 (Register Word 단위 배열)
  - ↳int datas.RequestDatas.iNumberOfByte : Master-Device로 Response 할 데이터의 개수

사용법

참고 OnReadInputRegisters 사용법은 "프로그래밍 적용 가이드"에 설명된 코드와 크게 다르지 않아 생략되었습니다. "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

이벤트 OnReadInputStatus

Master-Device로부터 요청된 Function Code가 Read Input Status(FC 02)일 경우 발생되는 이벤트입니다.

참고 1. "Function별 Master 요청에 따른 Event 코드 작성 과정" 내용을 참고하시기 바랍니다.
2. "Coil과 Register의 차이점" 내용을 참고하시기 바랍니다.

◉ OnReadInputStatus(SmartModbusSlave.READINPUTSTATUS datas)

[인자]

• SmartModbusSlave.READCOILSTATUS datas : Request 데이터 확인 및 Response 데이터 설정용 인자

- datas.RequestDatas : Master-Device로부터 요청된 Request 데이터가 저장 되어있음
  - ↳ int datas.RequestDatas.iFunctionCode : Master-Device로부터 요청된 FunctionCode(확인할 필요 없음)
  - ↳ int datas.RequestDatas.iNumberOfCoil : Master-Device로부터 요청된 Coil의 개수
  - ↳ int datas.RequestDatas.iSlave : Master-Device로부터 요청된 SlaveAddress
  - ↳ int datas.RequestDatas.iStartAddress : Master-Device로부터 요청된 시작 주소
- datas.ResponseDatas : Master-Device로 Response 할 데이터를 저장받음
  - ↳ bool datas.RequestDatas.bSkipSendResponse : Master-Device로부터 요청된 데이터에 오류가 있을 경우 Master-Device로 잘못된 데이터가 Response되지 않도록 true로 설정해야 함
  - ↳ byte[] datas.RequestDatas.chDatas : Master-Device로 Response 할 Coil 상태 데이터 바이트 배열
  - ↳ int datas.RequestDatas.iNumberOfByte : Master-Device로 Response 할 데이터 바이트의 개수

※ iNumberOfByte = Math.Ceiling(iNumberOfCoil / 8)

사용법

참고 OnReadInputStatus 사용법은 OnReadCoilStatus 사용법과 크게 다르지 않으므로 “OnReadCoilStatus 사용법” 내용을 참고하시기 바랍니다.

6) SmartModbusSlave 예제 사용하기

SmartModbusSlave를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → “SmartModbusSlave”

7. SmartSound

IEC-Series 제품 내부에는 0.8W의 스피커가 내장되어 있으며, 별도로 외부 스테레오 오디오 출력 단자를 지원합니다. SmartSound는 IEC-Series에서 응용 프로그램 개발 시 Sound 관련 기능을 편리하게 제어할 수 있게 한 컴포넌트로 기본적으로 MP3, Wave, WMA 등의 오디오 파일을 쉽게 처리할 수 있습니다. SmartSound를 사용하시면 기기의 음성 안내 프로그램을 쉽게 구현할 수 있습니다.

- 외부 스트레오 오디오 출력 단자 지원
- Sound 제어에 편리한 각종 기능 및 인터페이스 지원
- 오디오 파일 형식(MP3, Wave, WMA 등) 지원

참조 IEC-Series 제품 스피커 관련 사항

※ 터치 소리 조절 방법
"홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 사운드 관련"

※ IEC-Series 제품 스피커 출력 증가 방법
"홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 38. IEC-Series 스피커 출력 증가 방법 안내"

1) IEC-Series 제품에 따른 운영체제별 지원 오디오 파일 형식

IEC-Series에서 지원되는 오디오 파일의 형식은 모델 및 운영체제에 따라 차이가 있습니다. 아래 표를 확인하여 적용하시기 바랍니다.

[표1] IEC1000-Series 운영체제별 지원 오디오 파일 형식

O/S	flac	mov	mp1	mp2	mp3	wav	wma
Pro	X	X	O	O	O	O	O
Std	X	X	X	X	X	O	O

[표2] IEC667-Series 운영체제별 지원 오디오 파일 형식

O/S	flac	mov	mp1	mp2	mp3	wav	wma
Pro	X	O	X	X	O	O	O
Std	X	O	X	X	O	O	O

공통 지원 파일 형식	부분 지원 파일 형식
mp3, wav(wave), wma	mp1(mpeg-1), mp2(mpeg-2), mov

2) 프로그래밍 적용 가이드

STEP-1 SmartSound 사용하기

PlayFilePath 속성으로 재생할 Sound 파일의 경로와 파일명을 설정하며, Volume 속성으로 소리 크기를 설정합니다. 이후 Play() 메소드를 호출하여 설정된 Sound 파일을 시작합니다.

※ 자세한 내용은 "PlayFilePath, Volume 속성"과 "Play() 메소드" 내용을 참고하시기 바랍니다.

```
smartSound1.PlayFilePath = "Flash Disk₩₩TestSound.mp3";  // Sound 파일 지정
smartSound1.Volume = 30000; // 소리 크기 설정
smartSound1.Play();         // 설정한 Sound 파일을 Play
```

3) SmartSound 인터페이스 설명

SmartSound Component Interface		
속성		
PlayFilePath : string	UpDownVolumeStep : uint	Volume : ulong
메소드		
DownStepVolume() : void	IsPlay() : bool	Pause() : void
Play() : bool	Stop() : bool	UpStepVolume() : void

프로퍼티(속성) PlayFilePath

Play 할 Sound 파일의 경로를 문자열로 설정합니다.

- `string` : 음원 파일 경로

C# 사용법

```
smartSound1.PlayFilePath = "Flash Disk₩₩test.mp3"; // SmartSound에 Sound 파일 경로 지정
```

VB 사용법

```
smartSound1.PlayFilePath = "Flash Disk₩₩test.mp3"
```

프로퍼티(속성) UpDownVolumeStep

볼륨의 증감 크기를 설정합니다.

- `uint` : 볼륨 증감 크기 (기본값 : 2500)

C# 사용법

```
smartSound1.UpDownVolumeStep = 3000;  // 볼륨의 증감 크기를 설정
```

C# 사용법

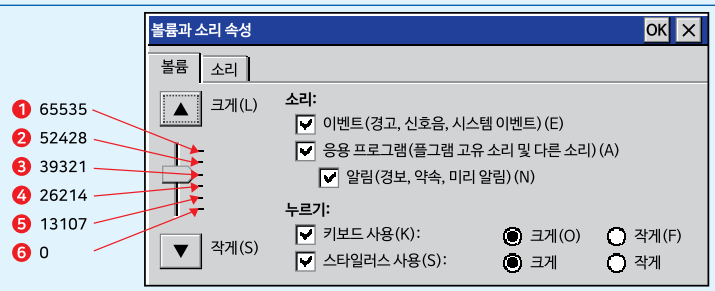
```
smartSound1.UpDownVolumeStep = 3000
```

프로퍼티(속성) Volume

IEC-Series의 오디오 볼륨을 설정하거나 가져옵니다. 볼륨의 최소값은 0이며 최대값은 65535입니다.

- `ulong` : 볼륨 (기본값 : 32767, 최소값 : 0, 최대값 : 65535)

참고 IEC-Series 제품의 "볼륨과 소리 속성"에서 눈금에 따른 볼륨값



C# 사용법

```
smartSound1.Volume += 500;  // 볼륨 500 증가
```

VB 사용법

```
smartSound1.Volume += 500
```

메소드(함수) DownStepVolume, UpStepVolume

IEC-Series의 오디오 볼륨을 UpDownVolumeStep 설정값 단위로 증가 및 감소합니다.

- DownStepVolume() : 볼륨값을 감소시킵니다.
- UpStepVolume() : 볼륨값을 증가시킵니다.

◉ void DownStepVolume()
◉ void UpStepVolume()

C# 사용법

```
// 볼륨 감소
smartSound1.DownStepVolume();
// 볼륨 증가
smartSound1.UpStepVolume();
```

VB 사용법

```
smartSound1.DownStepVolume()
smartSound1.UpStepVolume()
```

메소드(함수) IsPlay

현재 Sound 파일의 Play 상태를 확인합니다.

◉ bool IsPlay()

[리턴값]

- bool : 현재 Sound 파일의 Play 여부
  - true : Play 중
  - false : Play Stop

C# 사용법

```
// Play 상태 확인
if (smartSound1.IsPlay() == true)
{
  MessageBox.Show( " Play 중 " );
}
else
{
  MessageBox.Show( " Play Stop " );
}
```

VB 사용법

```
If smartSound1.IsPlay() = true Then
  MessageBox.Show( " Play 중 " )
Else
  MessageBox.Show( " Play Stop " )
End If
```

메소드(함수) Pause, Play, Stop

Sound 파일의 동작을 제어합니다.

- Pause() : Play를 잠시 중단합니다. Play() 메소드를 호출하면 정지한 곳부터 다시 Play합니다.
- Play() : PlayFilePath 속성에서 지정한 파일을 Play합니다.
- Stop() : Play를 종료합니다. Play() 메소드를 호출하면 다시 처음부터 Play합니다.

◉ void Pause()
◉ bool Play()
◉ bool Stop()

[리턴값]

• bool : 동작 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// 재생
smartSound1.Play();
// 일시 정지
smartSound1.Pause();
// 종료
smartSound1.Stop();
```

VB 사용법

```
smartSound1.Play()
smartSound1.Pause()
smartSound1.Stop()
```

4) SmartSound 예제 사용하기

SmartSound를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartSound"

8. SmartDAC

IEC-Series는 DAC(Digital to Analog Converter) 기능을 옵션 형태의 Board(DAC Block)로 제공하고 있으며, Smart I/O-II, III Board에 장착하여 사용하도록 되어있습니다. SmartDAC는 DAC Block을 편리하게 사용할 수 있도록 하는 컴포넌트입니다.

- 제품별 최대 2채널 지원
- 12Bit(4096) 해상도
- SmartI/O - DAC Block 연동 필수
- 출력 전압 : DC 0 ~ 5V or 0 ~ 10V (스위치 선택)

참조 DAC Block의 하드웨어 관련 정보는 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → Smart I/O - II, III 제품 매뉴얼"을 참조하시기 바랍니다.

주의 SmartDAC 사용 시 주의사항

1. SmartDAC를 사용 시 Port-B의 Pin6, 7을 사용하게 됩니다. SmartIIC도 동일한 Port를 사용하므로 SmartDAC와 SmartIIC는 동시에 사용할 수 없습니다.
2. SmartGPIO와 SmartDAC를 함께 사용하는 경우 프로그램이 정상 동작하지 않을 수 있습니다. 이 경우 SmartGPIO의 속성 중 Port-B의 DIR6, DIR7을 OUTPUT으로 설정하시기 바랍니다.

1) DAC 출력 속도

IEC-Series에서 DAC 출력을 하기 위해서는 SmartI/O-II, III와 DAC Block이 반드시 필요합니다. 아래 표에서 DAC Block 사양과 DAC 출력 속도를 확인하시기 바랍니다.

[표1] DAC Block 사양

채널 수	2EA
분해능	12Bit(4096)
출력 전압	DC 0 ~ 5V or 0 ~ 10V (스위치 선택)

[표2] IEC266-Series / IEC667-Series / IEC1000-Series에서 언어별 DAC 출력 속도

DAC_OutPut() 메소드 성능 테스트						
테스트 방법	A) SmartI/O-II 에 DAC Block을 장착하고 IEC-Series에 연결한다. B) DAC_OutPut() 메소드를 10,000회 반복 호출하여 SamplingTime을 측정한다.					
C# 테스트 코드		VB 테스트 코드		C++ 테스트 코드		

`for (int i = 0; i < 10000; i++)`
`{`
`  smartDAC1.DAC_OutPut(2000);`
`}`
		`For i As Integer = 0 To 10000`
		`  smartDAC1.DAC_OutPut(2000)`
		`Next`
				`for(int i = 0; i < 10000; i++)`
				`{`
				`  m_SmartDAC.SetOutData1(2000);`
				`}`		

DAC_OutPut() 메소드 1회 호출 시 Sampling Time(단위 : Second) 측정 결과(B 결과값을 10,000 으로 나눈 값)						
제품 · 언어	IEC1000-Series CE6.0 (B6버전)		IEC667-Series CE6.0 (B23버전)		IEC266-Series CE5.0 (B15버전)	
	C#, VB	C++	C#, VB	C++	C#, VB	C++
1회 호출 시간	0.000298s	0.000262s	0.000583s	0.000488s	0.000391s	0.000373s

※ 각 채널별 SamplingTime은 동일함

2) 프로그래밍 적용 가이드

STEP-1 DAC 채널 설정 및 출력하기

DAC 출력을 위해 ChannelSelect 속성으로 출력 채널을 설정할 수 있으며, DAC_OutPut() 메소드를 호출해 DAC 전압을 출력시킬 수 있습니다. 또는 DAC_OutPut() 메소드의 오버로드 메소드를 사용해 출력 채널 설정과 DAC 전압 출력을 동시에 하거나, 각 채널(1, 2)의 출력 설정을 동시에 할 수 있습니다.

※ 자세한 내용은 "ChannelSelect 속성", "DAC_OutPut() 메소드"를 참고하시기 바랍니다.

```
private void Output_Channel1()
{
  // DAC 채널1으로 1000에 해당되는 전압을 출력
  smartDAC1.ChannelSelect = SmartX.SmartDAC.DACOUTCHANNEL.DAC_CHANNEL1;
  smartDAC1.DAC_OutPut(1000);
}
private void Output_Channel2()
{
  // DAC 채널2로 4095에 해당되는 전압을 출력
  smartDAC1.DAC_OutPut(SmartX.SmartDAC.DACOUTCHANNEL.DAC_CHANNEL2, 4095);
}
private void Output_AllChannel()
{
  // DAC 채널1으로 0 값, DAC 채널2로 2048 값에 해당되는 전압을 출력
  smartDAC1.DAC_OutPut(0, 2048);
}
```

3) SmartDAC 인터페이스 설명

SmartDAC Component Interface
속성
ChannelSelect : SmartDAC.DACOUTCHANNEL
메소드
DAC_OutPut() : void (+2개 오버로드)

프로퍼티(속성) ChannelSelect

DAC 출력 채널을 설정합니다.

- SmartDAC.DACOUTCHANNEL.DAC_CHANNEL1 : DAC 출력 채널을 1로 설정
- SmartDAC.DACOUTCHANNEL.DAC_CHANNEL2 : DAC 출력 채널을 2로 설정

C# 사용법

```
// DAC 출력 채널을 1로 설정
smartDAC1.ChannelSelect = SmartDAC.DACOUTCHANNEL.DAC_CHANNEL1;
```

VB 사용법

```
smartDAC1.ChannelSelect = SmartDAC.DACOUTCHANNEL.DAC_CHANNEL1
```

메소드(함수) DAC_OutPut

DAC의 출력 전압값을 제어 합니다.

◉ void DAC_OutPut(int iOutData)
◉ void DAC_OutPut(int iOutData_Ch1, int iOutData_Ch2)
◉ void DAC_OutPut(SmartDAC.DACOUTCHANNEL outSelectCh, int iOutData)

[인자]

- int iOutData : 출력할 전압값. 다른 인자를 사용하지 않은 경우 ChannelSelect 속성에서 설정한 채널로 전압을 출력
- SmartDAC.DACOUTCHANNEL outSelectCh : 전압을 출력할 DAC 채널
- int iOutData_Ch1, int iOutData_Ch2 : 각각의 채널 1과 2의 출력 전압을 제어

C# 사용법

```
// DAC 채널1로 1000에 해당되는 전압을 출력
smartDAC1.ChannelSelect = SmartDAC.DACOUTCHANNEL.DAC_CHANNEL1;
smartDAC1.DAC_OutPut(1000);
// DAC 채널2로 4095에 해당되는 전압을 출력
smartDAC1.DAC_OutPut(SmartDAC.DACOUTCHANNEL.DAC_CHANNEL2, 4095);
// DAC 채널1로 0 값, DAC 채널2로 2048 값에 해당되는 전압을 출력
smartDAC1.DAC_OutPut(0, 2048);
```

VB 사용법

```
smartDAC1.ChannelSelect = SmartDAC.DACOUTCHANNEL.DAC_CHANNEL1
smartDAC1.DAC_OutPut(1000)
smartDAC1.DAC_OutPut(SmartDAC.DACOUTCHANNEL.DAC_CHANNEL2, 4095)
smartDAC1.DAC_OutPut(0, 2048)
```

4) SmartDAC 예제 사용하기

SmartDAC를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartDAC"

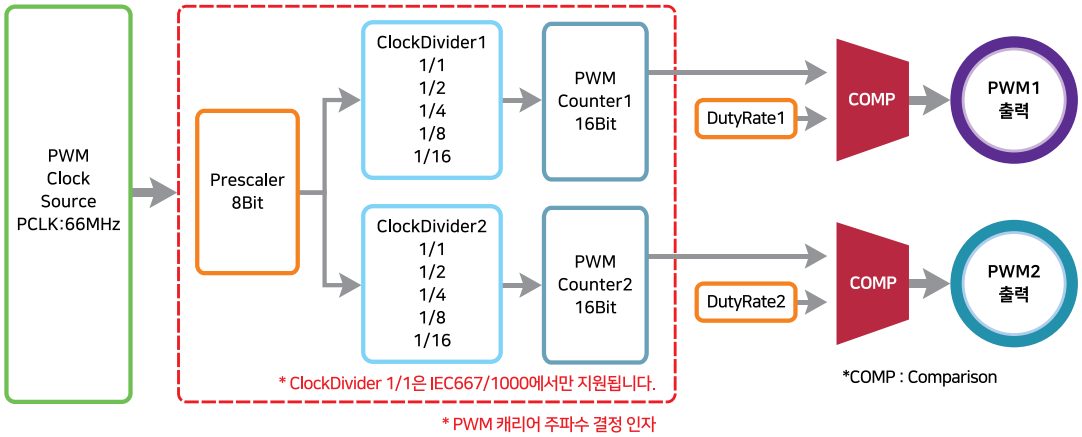
9. SmartPWM

IEC-Series 제품 내부 2채널의 16Bit PWM(Pulse Width Modulation)이 내장되어 있습니다. PWM 출력 단자는 Extension Port- Ⅰ,Ⅱ 커넥터 PWM1, PWM2 Pin을 사용하고 있습니다. 출력 전압 레벨은 0~3.3V입니다. 또한 데드 타임 설정 기능이 있어 전력제어용 PWM으로 사용하실 수 있습니다. SmartPWM을 사용하시면 PWM 제어를 아주 쉽게 처리하실 수 있습니다.

- 2채널 지원
- 최소 0.25Hz ~ 최대 66MHz Carrier Frequency
- Smart I/O – FET Block 연동 지원

참고 콘넥터 핀 정보에 관한 설명은 "Part-VI. 하드웨어 장치 안내"를 참고하시기 바랍니다.

1) Carrier Frequency 와 Duty Rate 관련 설명



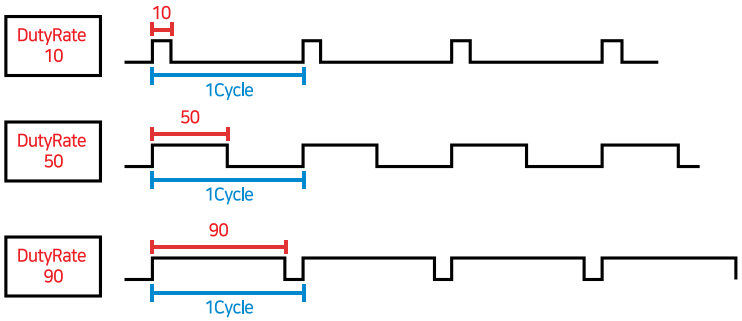
SmartPWM은 크게 Carrier Frequency(출력 주파수)와 Duty Rate(듀티비) 총 2가지의 값을 제어할 수 있습니다. 아래 표를 확인해보시기 바랍니다.

[표] Carrier Frequency와 Duty Rate 설명 및 관련 속성

Carrier Frequency(출력 주파수)	
[SmartPWM 속성값에 따른 Carruer Frequency 출력 과정]	
	
설명	IEC-Series에서 출력되는 PWM 출력 주파수로, SmartPWM의 속성(PreScaler, ClockDivider, PWMCounter) 설정에 따라 PWM Clock Source를 분주한 주파수입니다.
관련 속성	PreScaler, ClockDivider, PWMCounter

Duty Rate(듀티비)

[SmartPWM 속성값에 따른 출력 파형]



설명	PWM Clock Source에서 분주된 펄스(Carrier Frequency)를 SmartPWM의 속성(DutyRate)에서 설정한 값과 비교(Comparison)하여 파형이 출력됩니다.
관련 속성	DutyRate

2) Carrier Frequency 계산방법

Carrier Frequency는 Clock Source(66MHz)가 PreScaler, Clock Divider, PWMCounter 순으로 분주되어 출력됩니다. 따라서 각 속성값을 공식에 대입한다면 Carrier Frequency를 계산할 수 있습니다. 아래 표에서 공식과 예시를 확인 후 프로젝트에 적용해 보시기 바랍니다.

[표] SmartPWM 속성값에 따른 Carrier Frequency 계산 공식

공식	SmartPWM 속성값에 따른 Carrier Frequency 계산 공식

$$\text{Carrier Freq} = \text{Clock Source} \div (\text{PreScaler}+1) \div \text{ClockDivder} \div (\text{PWMCounter}+1)$$

- Carrier Freq : 출력 주파수(Carrier Frequency)
- Clock Source: PWM 원본 주파수(약 66MHz이며, 모든 IEC-Series가 동일함)
- PreScaler : PreScaler 속성값(0~255)
- ClockDivder : ClockDivder1/ ClockDivder2 속성값(1(IEC266의 경우 지원 안 함), 2, 4, 8, 16)
- PWMCounter : PWMCounter1/PWMCounter2 속성값(0~65535)

예시	Carrier Frequency를 최대/최소로 설정하기 위한 SmartPWM 속성값

먼저 IEC-Series의 Clock Source는 66MHz로 모두 같으므로 출력할 수 있는 최대 주파수와 최소 주파수는 IEC-Series와 관계 없이 모두 동일합니다.

- Clock Source : 66

1. 최대 Carrier Frequency 출력을 위한 SmartPWM 속성값은 아래와 같습니다.
- PreScaler : 0
- ClockDivder : 2 (SmartPWM.CLOCKDIVIDER.DIVIDE2)
- PWMCounter : 1

$$\text{Carrier Freq} = 66 \div (0+1) \div 2 \div (1+1) = 16.5\text{MHz}$$

즉, IEC-Series에서 출력 가능한 최대 Carrier Frequency는 16.5MHz입니다.

2. 최소 Carrier Frequency 출력을 위한 SmartPWM 속성값은 아래와 같습니다.
- PreScaler : 255
- ClockDivder : 16 (SmartPWM.CLOCKDIVIDER.DIVIDE16)
- PWMCounter : 65535

$$\text{Carrier Freq} = 66 \div (255+1) \div 16 \div (65535+1) = 0.25\text{Hz}$$

즉, IEC-Series에서 출력 가능한 최소 Carrier Frequency는 0.25Hz입니다.

[결과]
- 최대 Carrier Frequency : 16.5MHz
- 최소 Carrier Frequency : 0.25Hz

권장 실사용 시 PWMCounter는 최소 100 이상으로 설정하시기 바랍니다.

3) 프로그래밍 적용 가이드

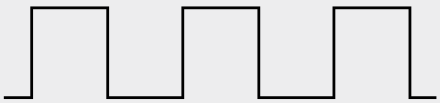
STEP-1 PWM 파형 출력하기

PWM 파형을 출력하려면 사용하는 PWM 채널에 따라 PreScaler, ClockDivider, PWMCounter 속성으로 캐리어 주파수 관련 설정을 해야합니다. 또는 SetCarryFrequency() 메소드로 일괄 설정할 수도 있습니다.
그다음 DutyRate 속성으로 듀티비를 설정하고, Polarity 속성으로 파형의 반전 여부를 설정합니다. 모든 설정이 완료되면 StartPWM() 메소드를 호출해 PWM 파형을 출력시킬 수 있습니다.

※ 자세한 내용은 "PreScaler, ClockDivider, PWMCounter, DutyRate, Polarity 속성", "SetCarryFrequency (), StartPWM() 메소드"를 참고하시기 바랍니다.

```
// 캐리어 주파수 관련 속성을 설정합니다.
smartPWM1.PreScaler = 13; smartPWM1.ClockDivider1 = SmartX.SmartPWM.CLOCKDIVIDER.DIVIDE2;
smartPWM1.PWMCounter1 = 1200;
// 위 속성 설정 코드와 아래 메소드는 동일한 값을 설정합니다.
// smartPWM1.SetCarryFrequency1(SmartPWM.CLOCKDIVIDER.DIVIDE2, 13, 1200);
smartPWM1.DutyRate1 = 50.0;  // 듀티비를 설정합니다.
smartPWM1.Polarity1 = SmartX.SmartPWM.POLARITY.HIGHACTIVE;  // 파형을 반전시키지 않도록 설정합니다.
smartPWM1.StartPWM1();  // PWM 파형 출력을 시작합니다.
```

Carrier Frequency	Period	Duty
1.98kHz	504.13us	50



[그림] 속성 설정에 따른 출력 파형 예시

STEP-2 PWM 파형 출력 중지 및 종료 시 처리사항

StopPWM() 메소드를 호출해 PWM 파형 출력을 중지할 수 있습니다. 프로그램 종료 시 시스템의 안정성을 높이기 위해 Closing 이벤트에서 ReleasePWM() 메소드를 호출해 PWM 관련 레지스터를 초기화합니다.

※ 자세한 내용은 "StopPWM(), ReleasePWM() 메소드"를 참고하시기 바랍니다.

```
private void Stop_PWM()
{
  smartPWM1.StopPWM1();  // PWM 파형 출력을 중지합니다.
}
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartPWM1.StopPWM1();    // 프로그램 종료 시 PWM 파형 출력을 중지 및
  smartPWM1.ReleasePWM();  // PWM 관련 레지스터를 초기화합니다.
}
```

4) SmartPWM 인터페이스 설명

SmartPWM Component Interface		
속성		
ClockDivider1 : SmartPWM.CLOCKDIVIDER	ClockDivider2 : SmartPWM.CLOCKDIVIDER	DeadTime : byte
DutyRate1 : double	DutyRate2 : double	Polarity1 : SmartPWM.POLARITY
Polarity2 : SmartPWM.POLARITY	PreScaler : byte	PWMCounter1 : uint
PWMCounter2 : uint		
메소드		
ReleasePWM() : void	SetCarryFrequency1(SmartPWM.CLOCKDIVIDER ClkDiv, byte btPrescaler, uint uiCounter) : void	SetCarryFrequency2(SmartPWM.CLOCKDIVIDER ClkDiv, byte btPrescaler, uint uiCounter) : void
StartPWM1() : void	StartPWM2() : void	StopPWM1() : void
StopPWM2() : void		

프로퍼티(속성) ClockDivider1, ClockDivider2

PreScaler에서 PWM Clock이 공급되면 이 Clock은 다시 ClockDivder에 의해서 분주됩니다.
ClockDivider은 이 분주값을 설정합니다.

참고 "Carrier Frequency와 Duty Rate 관련 설명"을 참고하시기 바랍니다.

주의 IEC266-Series는 1/1 분주비를 지원하지 않습니다.

- SmartPWM.CLOCKDIVIDER.DIVIDE1 : Clock을 1/1로 분주 (IEC266-Series 미지원)
- SmartPWM.CLOCKDIVIDER.DIVIDE2 : Clock을 1/2로 분주
- SmartPWM.CLOCKDIVIDER.DIVIDE4 : Clock을 1/4로 분주
- SmartPWM.CLOCKDIVIDER.DIVIDE8 : Clock을 1/8로 분주
- SmartPWM.CLOCKDIVIDER.DIVIDE16 : Clock을 1/16로 분주

C# 사용법

```
// 분주비를 1/2로 설정
smartPWM1.ClockDivider1 = SmartPWM.CLOCKDIVIDER.DIVIDE2;
```

VB 사용법

```
smartPWM1.ClockDivider1 = SmartPWM.CLOCKDIVIDER.DIVIDE2
```

프로퍼티(속성) DeadTime

데드 타임 설정 기능이 있어 전력 제어용 PWM으로 사용하실 수 있습니다. 데드 타임 설정값이 0보다 큰 경우 PWM1과 PWM2는 서로 연동되어 동작합니다. 즉 PWM1은 High Side, PWM2는 Low Side로 연동되어 동작됩니다. 데드 타임이 설정되면 PWM1의 제어로 PWM2까지 제어됩니다.

- byte : 데드 타임 설정값
  - 0 : 데드 타임 설정 안 함
  - 1~255 : 데드 타임 설정

C# 사용법

```
// 데드 타임 설정
smartPWM1.DeadTime = 1;
```

VB 사용법

```
smartPWM1.DeadTime = 1
```

프로퍼티(속성) DutyRate1, DutyRate2

PWM 출력의 Duty Rate(듀티비)를 백분율로 설정합니다.

참고 "Carrier Frequency와 Duty Rate 관련 설명"을 참고하시기 바랍니다.

• double : 듀티비 (범위 : 0~100.00)

C# 사용법

```
// PWM1의 Duty Rate를 33.3%로 설정함
smartPWM1.DutyRate1 = 33.3;
```

VB 사용법

```
smartPWM1.DutyRate1 = 33.3
```

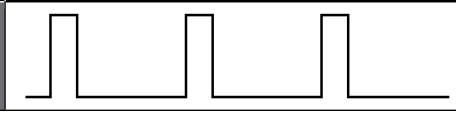
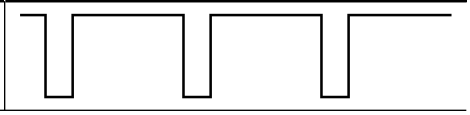
프로퍼티(속성) Polarity1, Polarity2

PWM 출력 파형의 정 • 부 논리를 변경합니다. 즉 출력 파형을 반전하여 PWM 출력을 할 수 있도록 합니다.

• SmartPWM.POLARITY.HIGHACTIVE : PWM 출력 비반전 처리

• SmartPWM.POLARITY.LOWACTIVE : PWM 출력 반전 처리

[표] Polarity 속성값 설정에 따른 출력 파형의 변화 (Duty Rate : 20)

속성값	HIGHACTIVE	LOWACTIVE
출력 파형		

C# 사용법

```
// PWM 출력 반전 처리
smartPWM1.Polarity2 = SmartPWM.POLARITY.LOWACTIVE;
```

VB 사용법

```
smartPWM1.Polarity2 = SmartPWM.POLARITY.LOWACTIVE
```

프로퍼티(속성) PreScaler

PreScaler는 0~255 값을 설정할 수 있으며, PreScaler의 변경으로 PWM의 Carrier Frequency를 조정할 수 있습니다. PWM1과 PWM2는 동일한 PreScaler를 사용합니다.

참고 "Carrier Frequency와 Duty Rate 관련 설명"을 참고하시기 바랍니다.

• byte : PreScaler 값 (범위 : 0~255)

C# 사용법

```
// PreScaler 값을 13으로 적용
smartPWM1.PreScaler = 13;
```

VB 사용법

```
smartPWM1.PreScaler = 13
```

프로퍼티(속성) PWMCounter1, PWMCounter2

PWMCounter는 16Bit(65536)이며 PWM의 캐리어 주파수를 세밀하게 제어할 수 있습니다.

참고 "Carrier Frequency와 Duty Rate 관련 설명"을 참고하시기 바랍니다.

주의 PWMCounter는 최소 100 이상으로 설정하시기 바랍니다.

DutyRate가 0~100의 실수형이므로 PWMCounter는 최소 100 이상 되어야 듀티비를 올바르게 조절 가능합니다.

• uint : PWMCounter 값 (범위 : 0 ~ 65535)

C# 사용법

```
// PWM1의 Counter 값 적용
smartPWM1.PWMCounter1 = 1200;
```

VB 사용법

```
smartPWM1.PWMCounter1 = 1200
```

메소드(함수) ReleasePWM

프로그램 종료 시 PWM관련 레지스터를 초기 상태로 복구합니다. ReleasePWM() 메소드 호출 시 PWM과 관련하여 사용 중인 리소스를 모두 반환하여 시스템의 안정성을 높여줍니다.

◉ void ReleasePWM()

C# 사용법

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartPWM.ReleasePWM();
}
```

VB 사용법

```
Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As System.ComponentModel.
                          CancelEventArgs) Handles MyBase.Closing
  SmartPWM.ReleasePWN()
End Sub
```

메소드(함수) SetCarryFrequency1, SetCarryFrequency2

PWM의 Carrier Frequency관련 인자들을 모두 변경합니다. 즉 PreScaler, ClockDivider, PWMCounter 속성을 일괄 변경합니다.

참고 "PreScaler, ClockDivider, PWMCounter 속성" 내용을 참고하시기 바랍니다.

◉ void SetCarryFrequency1(SmartPWM.CLOCKDIVIDER ClkDiv, byte btPrescaler, uint uiCounter)
◉ void SetCarryFrequency2(SmartPWM.CLOCKDIVIDER ClkDiv, byte btPrescaler, uint uiCounter)

[인자]

• SmartPWM.CLOCKDIVIDER ClkDiv : ClockDivider값
• byte btPrescaler : PreScaler 값 (범위 : 0~255)
• uint uiCounter : PWMCounter 값 (범위 : 0 ~ 65535)

C# 사용법

```
// PreScaler : 33, ClockDivider : 2, PWMCounter : 700
smartPWM1.SetCarryFrequency1(SmartPWM.CLOCKDIVIDER.DIVIDE2, 33, 700);
```

VB 사용법

```
smartPWM1.SetCarryFrequency1(SmartPWM.CLOCKDIVIDER.DIVIDE2, 33, 700)
```

메소드(함수) StartPWM1, StartPWM2

PWM 출력을 시작합니다.

◉ void StartPWM1()
◉ void StartPWM2()

C# 사용법

```
// PWM1의 현재 설정된 상태로 PWM 출력을 시작한다.
smartPWM1.StartPWM1();
```

VB 사용법

```
smartPWM1.StartPWM1()
```

메소드(함수) StopPWM1, StopPWM2

PWM 출력을 중지합니다.

◉ void StopPWM1()
◉ void StopPWM2()

C# 사용법

```
// PWM1의 PWM 출력을 중지한다.
smartPWM1.StopPWM1();
```

VB 사용법

```
smartPWM1.StopPWM1()
```

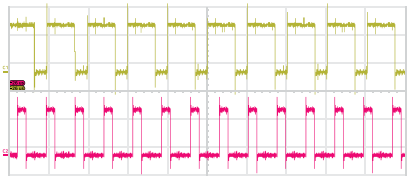
5) SmartPWM 예제 사용하기

SmartPWM을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartPWM"

[출력 결과 파형]

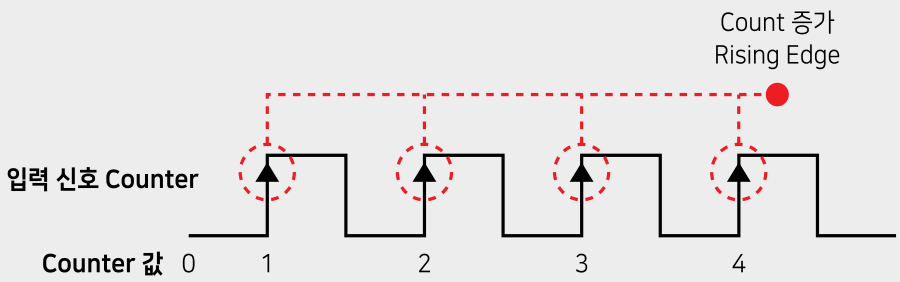


※ 위처럼 SmartPWM 테스트 프로그램에서 설정된 경우 오른쪽 같은 PWM 파형이 출력됩니다.

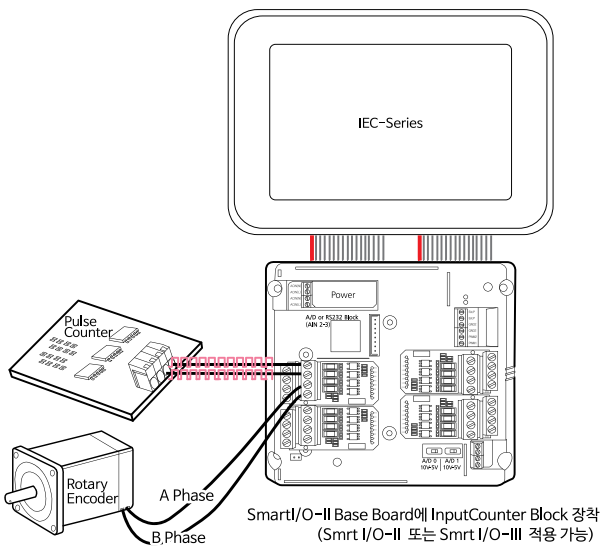
10. SmartInputCounter

SmartInputCounter는 기존 SmartGPIO의 입력 성능을 대폭 개선하여 SmartGPIO의 입력 대비 고속의 입력 처리 기능을 지원하는 컴포넌트입니다. 반드시 InputCounter-Block과 함께 사용해야 하며 Block 당 2개의 InputCounter와 1개의 Rotary Encoder Counter 지원, Counting 데이터 범위가 최대 64Bit, 예외 처리 기능을 제공합니다. SmartImputCounter는 내부에 독립적인 MCU가 탑재되어 높은 주파수의 입력 신호를 처리할 수 있습니다.

- SmartGPIO의 입력 대비 고속의 입력 처리 기능 지원(별도 MCU 처리)
- 기존 IEC-Series SmartGPIO의 Input 성능 대비 약 23배 성능 향상
  - ↳ 기존 SmartGPIO(Input)은 29.48kHz(0.03ms)에 비해 SmartInputCounter는 680kHz(0.00147ms) 속도로 수신 가능(IEC1000-Series 기준)
- 최대 8채널(ch)의 InputCounter 지원
  - ↳ 2개의 InputCounter-Block 장착 가능
- 최대 64Bit Data Width 지원
- Rotary Encoder Counter 지원
- SmartInputCounter의 각종 기능 제공
  - ↳ Initialize(초기화), Run(수신 시작), Pause(수신 정지), Reset(카운팅값 초기화)
- CheckSum Exception 기능 지원으로 데이터 신뢰성 보장
- 입력 신호의 Rising Edge에서 카운팅값 증가



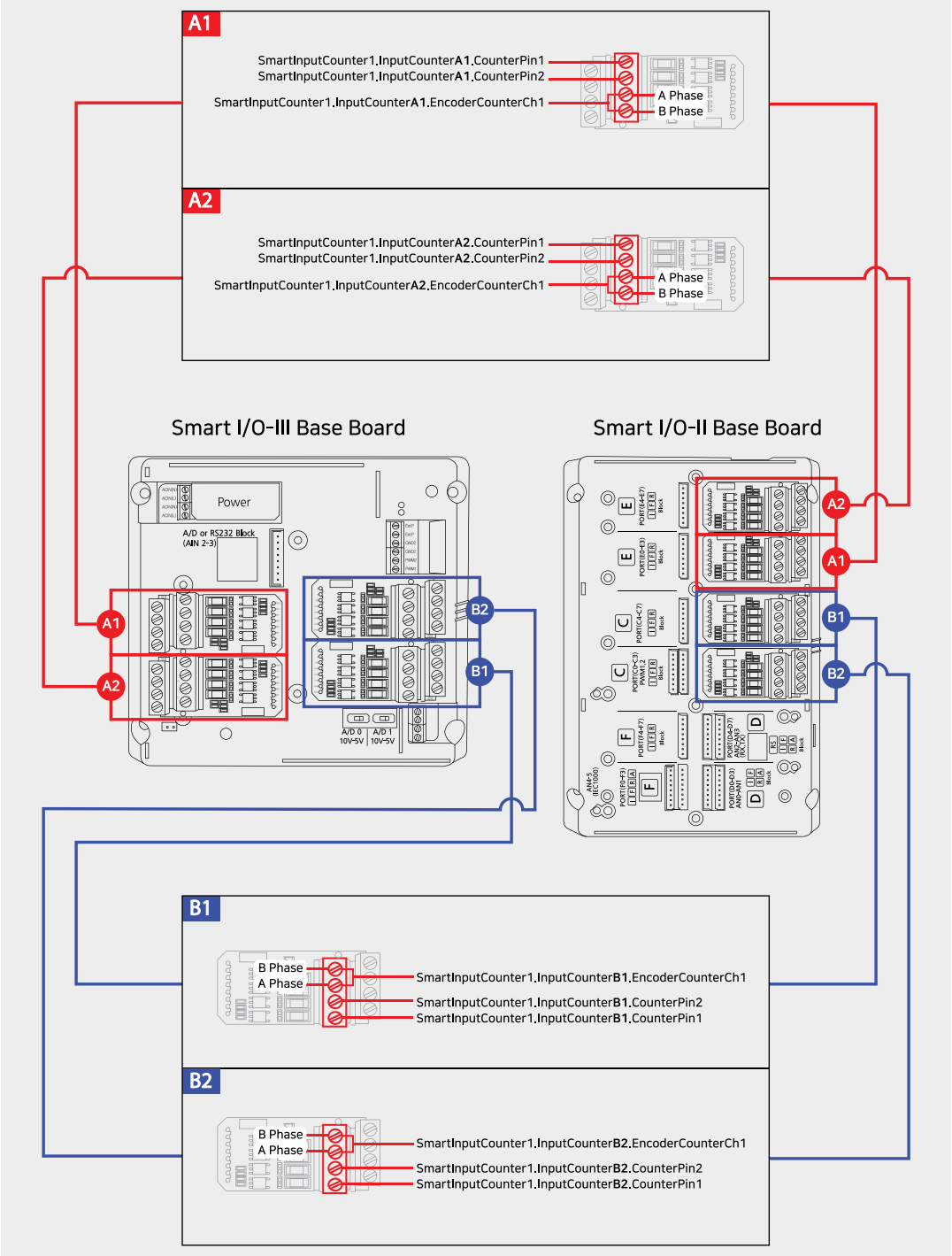
참고 콘넥터 핀 정보에 관한 설명은 "Part-VI. 하드웨어 장치 안내"를 참고하시기 바랍니다.



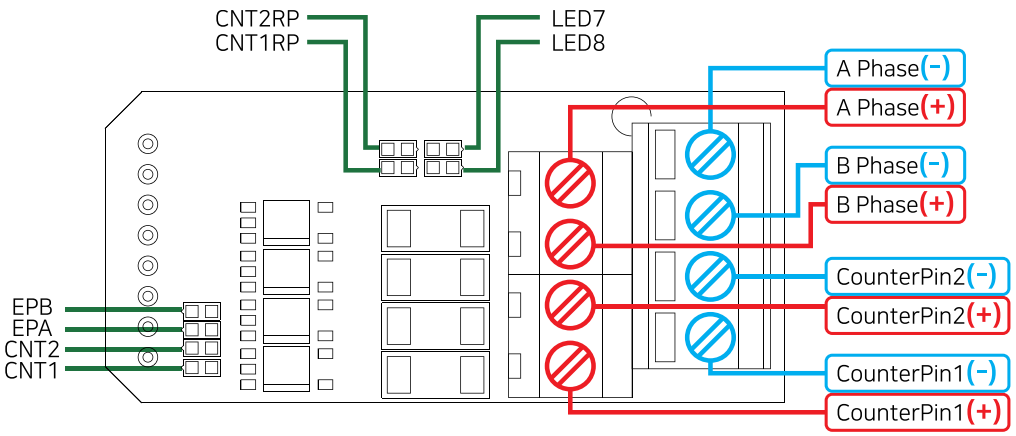
SmartI/O-II Base Board에 InputCounter Block 장착
(Smrt I/O-II 또는 Smrt I/O-III 적용 가능)

1) SmartI/O-II, III 에서 InputCounter-Block 장착 위치 및 포트

SmartI/O-II, III에 InputCounter-Block 장착 시, 반드시 정해진 위치에 장착해야 올바르게 동작합니다.
아래 그림에서 올바른 장착 위치를 확인해 보시기 바랍니다.



2) InputCounter-Block 단자 명칭 및 LED 설명



[그림] InputCounter-Block의 단자 명칭

[표] 입력(통신) 상태에 따른 LED 동작

명칭	설명	LED 상태	
		ON	OFF
EPA	A Phase의 입력 상태	High	Low
EPB	B Phase의 입력 상태		
CNT1	Counter Pin1의 입력 상태		
CNT2	Counter Pin1의 입력 상태		
CNT1RP	Counter Pin1의 Run / Pause 상태	Pause	Run
CNT2RP	Counter Pin2의 Run / Pause 상태		
LED7	데이터 모드가 32Bit / 64Bit 표시	64Bit	32Bit
LED8	InputCounter와 IEC-Series 간의 연결에 따른 I2C 통신 상태	연결된 경우 빠르게 점멸	

3) 프로그래밍 적용 가이드

NO-1 Counter Mode로 동작하기

Counter Mode로 동작할 경우 입력되는 펄스의 상승(Rising Edge) 구간을 Counting 할 수 있으며, 총 4개의 채널을 사용할 수 있습니다. SmartInputCounter를 Counter Mode로 동작시키려면 CounterDataWidth 속성으로 데이터 범위를 설정 해야하며, PortMode 속성을 Used로 설정 후 Initialize() 메소드를 호출해야 합니다. 모든 설정 완료 후 RunCounterPin() 메소드를 호출하면 해당하는 Pin의 Counting을 시작합니다.

※ 자세한 내용은 “CounterDataWidth, PortMode 속성”, “Initialize() 메소드” 내용을 참고하시기 바랍니다.

```
// InputcounterA1 블록 - CounterPin1의 Port Mode를 InputCounter로 설정
smartInputCounter1.InputCounterA1.PortMode = SmartX.SmartInputCounter.PORTMODES.Used;
// InputcounterA1 블록 - CounterPin1의 데이터 범위를 32Bit로 설정
smartInputCounter1.InputCounterA1.CounterDataWidth = SmartX.SmartInputCounter.DATAWIDTH._32BIT;
if (smartInputCounter1.InputCounterA1.Initialize() == true)
{
  // InputcounterA1 블록 - CounterPin1의 Counting 시작
  smartInputCounter1.InputCounterA1.RunCounterPin1();
  // 타이머 시작
```

```
  smartTimer1.Start();
}
else
{
  // InputCounter-Block 미연결 또는 초기화 도중 오류 발생
}

// Timer를 이용해 Input Counter의 Counting 값을 읽음
private void smartTimer1_Tick(object sender, EventArgs e)
{
  // InputcounterA1 블록 - CounterPin1의 Counting 값을 읽음
  ulong ulCount = smartInputCounter1.InputCounterA1.CounterPin1;
}
```

NO-2 Rotary Encoder Mode로 동작하기

SmartInputCounter에서 Rotary Encoder 값을 읽기 위해서는 Roatry Encoder와 InputCounter-Block의 A, B상(Phase)을 결선해야 하며, Rotary Encoder의 회전 방향과 A, B상 결선에 따라 +, - 값으로 Counting됩니다. SmartInputCounter를 Rotary Encoder Mode로 동작시키려면 PortMode 속성을 Used로 설정 후 Initialize() 메소드를 호출해야 합니다. 모든 설정 완료 후 RunCounterPin() 메소드를 호출하면 해당하는 Pin의 Counting을 시작하여 Rotary Encoder 값을 읽을 수 있습니다.

※ 자세한 내용은 "EncoderCounterCh1 속성" 내용을 참고하시기 바랍니다.

```
// InputcounterA1 블록 - CounterPin1의 Port Mode를 InputCounter로 설정
smartInputCounter1.InputCounterA1.PortMode = SmartX.SmartInputCounter.PORTMODES.Used;
if (smartInputCounter1.InputCounterA1.Initialize() == true)
{
  // InputcounterA1 블록 - CounterPin1의 Counting 시작
  smartInputCounter1.InputCounterA1.RunCounterPin1();
  // 타이머 시작
  smartTimer1.Start();
}
else
{
  // InputCounter-Block 미연결 또는 초기화 도중 오류 발생
}

// Timer를 이용해 Input Counter의 Rotary Encoder의 값을 읽음
private void smartTimer1_Tick(object sender, EventArgs e)
{
  // InputcounterA1 블록 - CounterPin1의 Rotary Encoder의 값을 읽음
  long lCount = smartInputCounter1.InputCounterA1.EncoderCounterCh1;
}
```

NO-3 CounterPin 상태 읽기 및 에러 감지하기

SmartInputCounter에서 입력 감지를 시작한 경우, CounterPin의 상태를 읽을 수 있으며, try~catch문을 사용해 Counting 시 발생하는 에러를 감지하여 처리할 수 있습니다. 에러 감지의 경우 NO-1, 2도 적용 가능합니다.

※ 자세한 내용은 "StatusCounterPin1/2 속성", "CheckSumErrorException 이벤트" 내용을 참고하시기 바랍니다.

```
// InputcounterA1 블록 - CounterPin1의 Port Mode를 InputCounter로 설정
smartInputCounter1.InputCounterA1.PortMode = SmartX.SmartInputCounter.PORTMODES.Used;
```

```
if (smartInputCounter1.InputCounterA1.Initialize() == true)
{
  // InputcounterA1 블록 - CounterPin1의 Counting 시작
  smartInputCounter1.InputCounterA1.RunCounterPin1();
  // 타이머 시작
  smartTimer1.Start();
}
else
{
  // InputCounter-Block 미연결 또는 초기화 도중 오류 발생
}

// Timer를 이용해 Input Counter의 CounterPin 상태를 읽음
private void smartTimer1_Tick(object sender, EventArgs e)
{
  try
  {
    // InputcounterA1 블록 - CounterPin1의 상태를 읽음
    if (smartInputCounter1.InputCounterA1.StatusCounterPin1 == true)
    {
      // High
    }
    else
    {
      // Low
    }
    // 나머지 Pin도 동일하게 처리하며 생략되었습니다.
  }
  catch (SmartX.SmartInputCounter.CheckSumErrorException eException)
  {
    // eException 인자를 확인하여 예외 처리 가능
  }
}
```

중요 SmartInputCounter와 SmartGPIO(Output)를 동시에 사용 시 필수사항

1. SmartInputCounter와 SmartGPIO(Output)를 동시에 사용하는 경우에 소스 코드 상에서 Output 설정을 먼저 해야합니다.

```
// **중요** 반드시 SmartInputCounter보다 먼저 설정해 주어야함.
// PORTA1을 출력으로 설정
smartGPIO1.PORTADIR0 = SmartX.SmartGPIO.PORTDIRS.OUTPUT;
smartGPIO1.PORTADIR1 = SmartX.SmartGPIO.PORTDIRS.OUTPUT;
```

2. SmartInputCounter는 다음과 같은 순서로 속성을 설정합니다.

```
// A1 포트 사용
smartInputCounter1.InputCounterA1.PortMode = SmartX.SmartInputCounter.PORTMODES.Used;
// 블록의 초기화
smartInputCounter1.InputCounterA1.Initialize();
// 데이터 범위를 32 또는 64Bit로 설정
smartInputCounter1.InputCounterA1.CounterDataWidth = SmartX.SmartInputCounter.DATAWIDTH._32BIT;
```

4) SmartInputCounter 인터페이스 설명

SmartInputCounter Component Interface		
속성		
CounterDataWidth: SmartInputCounter.DATAWIDTH	CounterPin1: ulong	CounterPin2 : ulong
EncoderCounterCh1 : long	PortMode : SmartInputCounter.PORTMODES	StatusCounterPin1: bool
StatusCounterPin2 : bool		
메소드		
GetFirmwareVersion() : int	Initialize() : bool (+1개 오버로드)	PauseCounterPin1() : void
PauseCounterPin2() : void	ResetCounterPin1() : void	ResetCounterPin2() : void
ResetEncoderCounterCh1() : void	RunCounterPin1() : void	RunCounterPin2() : void
이벤트		
CheckSumErrorException() : SmartInputCounter.CheckSumErrorExcaption (+2개 오버로드)		

프로퍼티(속성) CounterDataWidth

SmartInputCounter의 Counting 범위를 설정합니다.

- SmartInputCounter.DATAWIDTH._32BIT : Counting 범위를 32Bit로 설정(최대 Count 개수 : 4,294,967,295)
- SmartInputCounter.DATAWIDTH._64BIT : Counting 범위를 64Bit로 설정(최대 Count 개수 : 1.8E308)

C# 사용법

```
// InputcounterA1 블록 - CounterPin1의 Counting 범위를 32Bit로 설정
smartInputCounter1.InputCounterA1.CounterDataWidth = SmartInputCounter.DATAWIDTH._32BIT;
```

VB 사용법

```
smartInputCounter1.InputCounterA1.CounterDataWidth = SmartInputCounter.DATAWIDTH._32BIT
```

프로퍼티(속성) CounterPin1, CounterPin2

각 Pin의 현재 Counting 값을 읽습니다.

- CounterPin1 : CounterPin1의 Counting 값을 읽습니다.
- CounterPin2 : CounterPin2의 Counting 값을 읽습니다.
- ulong : 현재 Counting 값

C# 사용법

```
// InputcounterA1 블록 - CounterPin1의 Counting 값을 읽음
ulong ulCount = smartInputCounter1.InputCounterA1.CounterPin1;
```

VB 사용법

```
Dim ulCount As ULong = smartInputCounter1.InputCounterA1.CounterPin1
```

프로퍼티(속성) EncoderCounterCh1

Rotary Encoder의 현재 Counting 값을 읽습니다. 회전 방향과 A, B 상 결선에 따라 +와 – 값으로 카운팅됩니다.

- long : Rotary Encoder의 현재 Counting 값(범위(long) : –2,147,483,648 ~ 2,147,483,647)

C# 사용법

```
// InputcounterA1 블록의 Rotary Encoder의 데이터값을 읽음
long lCount = smartInputCounter1.InputCounterA1.EncoderCounterCh1;
```

VB 사용법

```
Dim lCount As Long = smartInputCounter1.InputCounterA1.EncoderCounterCh1
```

프로퍼티(속성) PortMode

InputCounter-Block의 모드를 설정합니다.

중요 반드시 Initialize() 메소드보다 먼저 설정되어야 합니다.

- SmartInputCounter.PORTMODES.Used : Input Counter 모드로 사용하며, Block 당 2 채널의 Input Counter와 1채널의 Rotary Encoder Counter로 사용
- SmartInputCounter.PORTMODES.NotUsed : GPIO 모드로 사용하며, Input Counter를 사용하지 않음

C# 사용법

```
// SmartInputCounter로 사용하는 경우(SmartInputCounter 용도로 사용)
smartInputCounter1.InputCounterA1.PortMode = SmartInputCounter.PORTMODES.Used;
// SmartInputCounter로 사용안하는 경우(GPIO 용도로 사용)
smartInputCounter1.InputCounterA1.PortMode = SmartInputCounter.PORTMODES.NotUsed;
```

VB 사용법

```
smartInputCounter1.InputCounterA1.PortMode = SmartInputCounter.PORTMODES.Used
smartInputCounter1.InputCounterA1.PortMode = SmartInputCounter.PORTMODES.NotUsed
```

프로퍼티(속성) StatusCounterPin1, StatusCounterPin2

각 CounterPin의 현재 입력 상태를 표시합니다.

- StatusCounterPin1 : CounterPin1의 입력 상태를 표시합니다.
- StatusCounterPin2 : CounterPin2의 입력 상태를 표시합니다.
- bool : 각 CounterPin의 현재 상태
  - true : High(DC 12~24V)
  - false : Low(DC 0V)

C# 사용법

```
if (smartInputCounter1.InputCounterA1.StatusCounterPin1 == true)
{
  // InputcounterA1 블록 - CounterPin1의 입력 상태 : High
}
else
{
  // InputcounterA1 블록 - CounterPin1의 입력 상태 : Low
}
```

VB 사용법

```
If smartInputCounter1.InputCounterA1.StatusCounterPin1 = True Then
  ' InputcounterA1 블록 - CounterPin1의 입력 상태 : High
Else
  ' InputcounterA1 블록 - CounterPin1의 입력 상태 : Low
End If
```

메소드(함수) GetFirmwareVersion

InputCounter-Block 내부 MCU의 Firmware Version을 얻습니다.

◉ int GetFirmwareVersion()

[리턴값]

• int : Firmware Version 값

C# 사용법

```
// InputCounter-Block의 FirmWare 버전을 얻는다.
int iVersion = smartInputCounter1.InputCounterA1.GetFirmwareVersion();
```

VB 사용법

```
Dim iVersion As Inteager = smartInputCounter1.InputCounterA1.GetFirmwareVersion()
```

메소드(함수) Initialize

InputCounter-Block을 초기화하는 메소드로서 SmartInputCounter의 사용 전에 반드시 호출해야 합니다.

중요 반드시 PortMode 속성을 먼저 설정해야 합니다.

◉ bool Initialize()

◉ bool Initialize(int iSpeed)

[인자]

• int iSpeed : Counting 속도. 값이 클수록 느려짐. 인자값 생략 시 최적화된 속도로 동작

[리턴값]

• bool : 연결 상태 및 초기화 성공 여부
  - true : InputCounter-Block 연결 및 정상 초기화
  - false : InputCounter-Block 미연결 또는 초기화 도중 오류 발생

C# 사용법

```
if (smartInputCounter1.InputCounterA1.Initialize() == false)
{
  // InputCounter-Block 미연결 또는 초기화 도중 오류 발생에 따른 코드 작성
}
```

VB 사용법

```
If smartInputCounter1.InputCounterA1.Initialize() = False Then
  ' InputCounter-Block 미연결 또는 초기화 도중 오류 발생
End If
```

메소드(함수) PauseCounterPin1, PauseCounterPin2

InputCounter-Block의 해당하는 CounterPin의 카운팅을 잠시 정지(Pause)하는 기능으로 PauseCounterPin() 메소드 호출 후 RunCounterPin() 메소드를 호출하여 Counting을 재개할 수 있습니다.

• PauseCounterPin1() : CounterPin1의 Counting을 일시 정지합니다.

• PauseCounterPin2() : CounterPin2의 Counting을 일시 정지합니다.

◉ void PauseCounterPin1()

◉ void PauseCounterPin2()

C# 사용법

```
smartInputCounter1.InputCounterA1.PauseCounterPin1();  // A1 블록 - CounterPin1의 카운팅 일시 정지
```

VB 사용법

```
smartInputCounter1.InputCounterA1.PauseCounterPin1()
```

메소드(함수) ResetCounterPin1, ResetCounterPin2

InputCounter-Block의 해당하는 CounterPin의 Counting 값을 초기화(Reset : 0)합니다.

- PauseCounterPin1() : CounterPin1의 Counting 값을 초기화합니다.
- PauseCounterPin2() : CounterPin2의 Counting 값을 초기화합니다.

◉ void ResetCounterPin1()
◉ void ResetCounterPin2()

C# 사용법

```
smartInputCounter1.InputCounterA1.ResetCounterPin1();  // A1 블록-CounterPin1의 Counting 값 초기화
```

VB 사용법

```
smartInputCounter1.InputCounterA1.ResetCounterPin1()
```

메소드(함수) RunCounterPin1, RunCounterPin2

InputCounter-Block의 해당하는 CounterPin의 Counting을 시작(Run)하는 기능으로 PauseCounterPin() 메소드 호출 후 RunCounterPin() 메소드를 호출해 Counting을 재개할 수 있습니다. 또한, RunCounterPin() 메소드를 중복 호출해도 동작 상태를 유지합니다.

- RunCounterPin1() : CounterPin1의 Counting을 시작합니다.
- RunCounterPin2() : CounterPin2의 Counting을 시작합니다.

중요 Initialize() 메소드 호출 후 Counting을 시작해야 합니다.

◉ void RunCounterPin1()
◉ void RunCounterPin2()

C# 사용법

```
// InputcounterA1 블록 - CounterPin1의 사용
smartInputCounter1.InputCounterA1.PortMode = SmartInputCounter.PORTMODES.Used;
// InputcounterA1 블록 - CounterPin1의 초기화
smartInputCounter1.InputCounterA1.Initialize();
// InputcounterA1 블록 - CounterPin1의 Counting 시작
smartInputCounter1.InputCounterA1.RunCounterPin1();
```

VB 사용법

```
smartInputCounter1.InputCounterA1.PortMode = SmartInputCounter.PORTMODES.Used
smartInputCounter1.InputCounterA1.Initialize()
smartInputCounter1.InputCounterA1.RunCounterPin1()
```

이벤트 CheckSumErrorException

장비를 운용하는 전기적인 환경이 좋지 않아 데이터가 전기적인 노이즈에 의해 깨지는 경우를 감지할 수 있도록 내부에서 CheckSum 기능을 지원하여 CheckSum Error가 발생할 경우 발생되는 예외입니다. 만약 CheckSum에서 에러가 발생하는 경우 예외 처리를 하여 에러 메시지를 출력하거나 다시 데이터를 읽어오는 등의 처리를 수행할 수 있습니다.

◉ CheckSumErrorException()
◉ CheckSumErrorException(string message)
◉ CheckSumErrorException(string message, Exception inner)

[인자]

- string message : 에러 메시지
- Exception inner : 현재 예외를 발생시킨 Exception 인스턴스

C# 사용법

```
private void smartTimer1_Tick(object sender, EventArgs e)
{
  try { m_lblCounterPin1.Text = smartInputCounter1.InputCounterA1.CounterPin1.ToString(); }
  catch(SmartInputCounter.CheckSumErrorException exceptions)
  {
    SmartMessageBox.Show(exceptions);  // 에러가 발생하여 메시지 출력 후 다시 읽음
    m_lblCounterPin1.Text = smartInputCounter1.InputCounterA1.CounterPin1.ToString();
  }
}
```

VB 사용법

```
Private Sub smartTimer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
smartTimer1.Tick
  Try : m_lblCounterPin1.Text = smartInputCounter1.InputCounterA1.CounterPin1.ToString()
  Catch exceptions As SmartInputCounter.CheckSumErrorException
    SmartMessageBox.Show(exceptions)
    m_lblCounterPin1.Text = smartInputCounter1.InputCounterA1.CounterPin1.ToString()
  End Try
End Sub
```

5) SmartInputCounter 예제 사용하기

SmartInputCounter를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

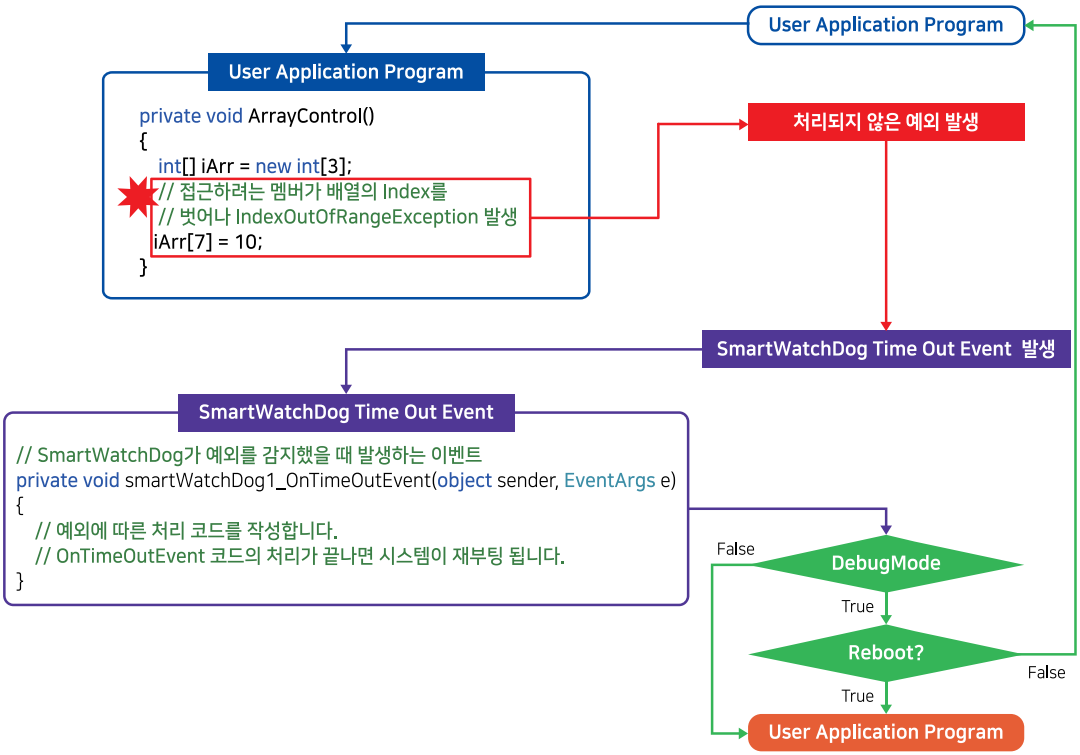
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → “SmartInputCounter”

11. SmartWatchDog

SmartWatchDog은 마이크로 컨트롤러(MCU)에서 말하는 WatchDog과 같은 기능을 하는 컴포넌트며, 장치 응용 프로그램에서 처리되지 않은 예외(Exception) 때문에 프로그램 동작이 멈추는 상황이 발생하면 이를 감지하여 시스템을 다시 시작하도록 처리합니다.

아래 그림과 같이 User Application Program의 동작 중 처리되지 않은 예외가 발생하면 SmartWatchDog에서 이를 감지하여 User Application Program으로 Time-Out 이벤트를 발생하고 시스템을 다시 시작하여 User Application Program을 항상 안정적으로 동작하도록 처리합니다. SmartWatchDog은 Management Code에서 발생하는 예외는 Catch 할 수 있지만, Unmanagement Code(Native Code Error 또는 RunTime Error)에서 발생하는 예외는 Catch 할 수 없습니다.



- 처리되지 않은 예외 감지 기능 지원
- 처리되지 않은 예외 발생 시 자동 재부팅 기능 지원
- 재부팅 지연 시간 설정 기능 지원

참고 IEC-Series에서 WatchDog 기능은 크게 Software(SmartWatchDog)와 Hardware 이렇게 2가지의 방식으로 지원되고 있습니다. 자세한 내용은 "Hardware WatchDog와 SmartWatchDog 설명"내용을 참고하시기 바랍니다.

중요 예외(Exception)란?

예외(Exception)란 프로그램 실행 중 발생한 모든 오류 상태 또는 예기치 못한 동작을 말합니다. 예외는 사용자가 호출한 코드(사용자의 코드나 공유 라이브러리 등)에 오류가 있거나, 운영체제 리소스를 사용할 수 없게 되거나, 공용 언어 런타임에서의 예기치 못한 상황 등에 의해 발생합니다. 사용자가 작성한 장치 응용 프로그램은 이러한 일부 상황으로부터 대부분 복구할 수 있지만 복구할 수 없는 경우(런타임 예외)도 있습니다.

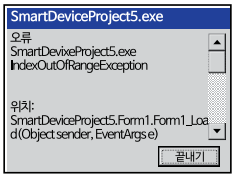
1) 예외 처리 방식

IEC-Series 제품 개발 중 개발자가 예외(Exception) 처리를 못하거나 실수로 인하여 에러는 얼마든지 발생할 수 있으며, 기본적으로는 예외가 발생되지 않도록 코드를 작성해야 하지만, 피치 못하게 예외 처리를 해야 하는 경우 예외 처리 방식은 크게 try~catch문과 WatchDog(Hardware, SmartWatchDog) 2가지가 있습니다. 아래 표를 확인하시기 바랍니다.

[표] 오류 발생 시 예외 처리 방식에 따른 결과

문제	처리되지 않는 런타임 오류가 발생함

코드	`private int GetDataToNumber(string strRawData)`
	`{`
	`  string strNumber = strRawData.Substring(3);`
	`  int iRetValue = Convert.ToInt32(strNumber);`
	`  return iRetValue;`
	`}`
	`private void Test_Click(object sender, EventArgs e)`
	`{`
	`  int iNum = GetDataToNumber( " CMD123456 " );  // 정상 변환 : 123456`
	`  iNum = GetDataToNumber( " CMD12345A " );  // FormatException 발생!`
	`  iNum = GetDataToNumber( " CMD " );  // FormatException 발생!`
	`}`

결과	 실행 중(런타임) 발생되는 오류의 형태

Level-1	예외가 발생되지 않도록 코드를 작성함

코드	`private int GetDataToNumber(string strRawData)`
	`{`
	`  int iRetValue = 0;`
	`  string strNumber = strRawData.Substring(3);`
	`  // 예외 발생 방지를 위해 문자열을 검증하는 코드를 작성`
	`  if (strNumber != " " )`
	`  {`
	`  foreach (char cData in strNumber)`
	`  {`
	`  if (Char.IsNumber(cData) == false){strNumber = " -1 ";}  // 예외 발생 시 -1 값을 리턴`
	`  }`
	`  iRetValue = Convert.ToInt32(strNumber);`
	`  }`
	`  else { iRetValue = -1; }  // 예외 발생 시 -1 값을 리턴`
	`  return iRetValue;`
	`}`
	`private void Test_Click(object sender, EventArgs e)`
	`{`
	`  int iNum = GetDataToNumber( " CMD123456 " );  // 정상 변환. 리턴값 : 123456`
	`  iNum = GetDataToNumber( " CMD12345A " );  // 올바르지 않은 문자열 입력. 리턴값 : -1`
	`  iNum = GetDataToNumber( " CMD " );  // 올바르지 않은 문자열 입력. 리턴값 : -1`
	`  if (iNum == -1) { /* 올바르지 않은 입력에 따른 처리 코드 작성 */ }`
	`}`

결과	에러가 발생하지 않음

Level-2	예외가 발생할 수 있는 곳에 try~catch문으로 예외를 처리하여 프로그램이 계속해서 실행될 수 있도록 함

코드

```
private int GetDataToNumber(string strRawData)
{
  int iRetValue = 0;
  string strNumber = strRawData.Substring(3);
  try { iRetValue = Convert.ToInt32(strNumber); } // 예외 발생 방지를 위해 try~catch 사용
  catch (FormatException) { iRetValue = -1; } // 예외 발생 시 -1 값을 리턴
  return iRetValue;
}
private void Test_Click(object sender, EventArgs e)
{
  int iNum = GetDataToNumber( " CMD123456 " );  // 정상 변환. 리턴값 : 123456
  iNum = GetDataToNumber( " CMD12345A " );  // 올바르지 않은 문자열 입력. 리턴값 : -1
  iNum = GetDataToNumber( " CMD " );  // 올바르지 않은 문자열 입력. 리턴값 : -1
  if (iNum == -1) { /* 올바르지 않은 입력에 따른 처리 코드 작성 */ }
}
```

결과: System Reboot 안 함

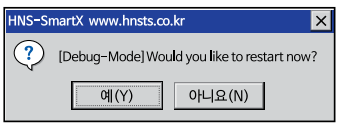
Level-3	다음과 같은 경우는 WatchDog을 이용해 처리함 (Ex. SmartWatchDog) 1. 모든 코드에서 예외 처리가 어려운 경우 2. 예상치 못한 곳에서 예외가 발생하는 경우 3. 시스템 오류가 발생하는 경우

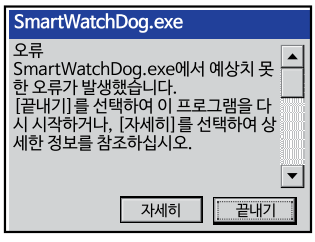
코드

```
private int GetDataToNumber(string strRawData)
{
  string strNumber = strRawData.Substring(3);
  int iRetValue = Convert.ToInt32(strNumber);
  return iRetValue;
}
private void Test_Click(object sender, EventArgs e)
{
  smartWatchDog1.Start(); // SmartWatchDog 기능을 실행
  int iNum = GetDataToNumber( " CMD123456 " );  // 정상 변환 : 123456
  iNum = GetDataToNumber( " CMD12345A " );  // FormatException 발생!
  iNum = GetDataToNumber( " CMD " );  // FormatException 발생!
  // SmartWatchDog이 에러를 감지하여 OnTimeOutEvent를 발생시키고 System Reboot을 진행
}
```

결과

DebugMode = true	DebugMode = false
System Reboot 여부를 확인합니다.	System Reboot을 바로 진행합니다.



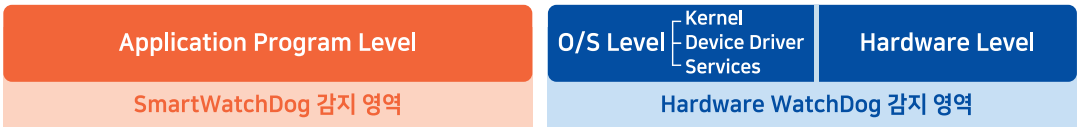


권장 권장 예외 처리 방식

하나의 예외 처리 방식을 적용하는 것보다 Level-1~3의 예외 처리 방식을 조합하여 프로그램을 개발하는 것이 안정적인 시스템 운영 환경을 만들 수 있습니다.

2) Hardware WatchDog와 SmartWatchDog 설명

IEC-Series 제품에서 시스템의 지속적인 운영(동작)과 안정성을 위하여 시스템 감시 기능인 WatchDog 기능을 제공하고 있습니다. 기존의 SmartX Framework에서 SmartWatchDog 기능을 지원하고 있었으며, 부족한 기능을 확장한 Hardware WatchDog이 O/S에 추가되어 더욱 시스템의 안정적인 운영이 가능합니다. 아래 표 내용을 확인하시기 바랍니다.



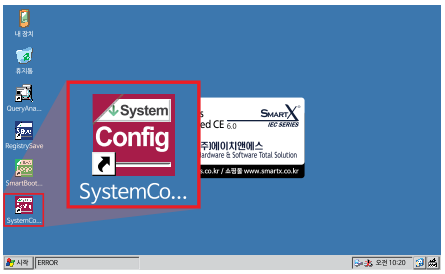
[표1] Hardware WatchDog과 SmartWatchDog의 차이점

구분	Hardware WatchDog	SmartWatchDog
설명	CPU 내부에서 지원되는 WatchDog 기능으로 O/S의 치명적인 오류 발생 시 System Reboot을 진행하여 운영체제(Kernel) 레벨의 오류를 처리할 수 있습니다.	해당 프로세서인 응용 프로그램에서 처리되지 않는 예외가 발생할 경우 System Reboot을 진행하여 응용 프로그램 레벨의 오류를 처리할 수 있습니다.
처리범위	운영체제(Kernel) 레벨	응용 프로그램 레벨
지원제품	- IEC266-Series : 미지원 - IEC667-Series : 빌드 20 이상 - IEC1000-Series : Windows Embedded CE 6.0 빌드 2 이상 Windows Embedded Compact 7 미지원	IEC-Series 전 제품 지원

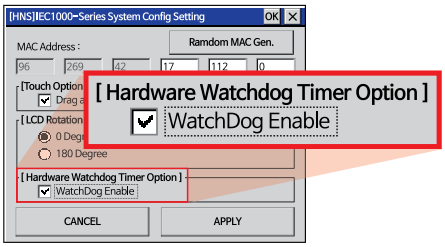
권장 Hardware WatchDog과 SmartWatchDog은 서로 상호 보완적인 기능으로 감시하는 영역과 인지할 수 있는 오류 대상이 달라 어떤 하나를 선택하여 사용하는 것보다는 두 가지 모두 사용할 것을 권장합니다.

[표2] Hardware WatchDog 설정 방법

[STEP-1] 바탕화면의 [SystemConfig] 클릭



[STEP-2] Hardware WatchDog Timer Option에서 [WatchDog Enable] 항목을 체크 후 [APPLY] 클릭



[STEP-3] 바탕화면의 [RegistrySave] 클릭 후 제품 Rebooting

중요 Hardware WatchDog 설정이 변경 될 경우 반드시 제품을 Rebooting(재시작)하셔야 합니다.

3) 프로그래밍 적용 가이드

CASE-1 SmartWatchDog 사용하기

해당 프로세서인 응용 프로그램에서 처리되지 않는 예외가 발생할 경우 System Reboot을 진행하여 응용 프로그램 레벨의 오류를 처리할 수 있습니다.

※ 자세한 내용은 "Hardware WatchDog와 SmartWatchDog 설명"을 참고하시기 바랍니다.

[STEP-1] 기본 설정 및 시작하기

DebugMode 속성으로 예외 발생 시 재부팅을 묻는 메시지의 출력 여부를 설정하고, RebootInterval 속성으로 예외 발생 시 이벤트 발생까지의 지연 시간을 설정합니다. Start() 메소드를 호출하면 Smart WatchDog을 시작할 수 있습니다.

※ 자세한 내용은 "DebugMode, RebootInterval 속성", "Start() 메소드"를 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  smartWatchDog1.DebugMode = false;    // 예외 발생 시 메시지 박스를 출력하지 않도록 설정
  smartWatchDog1.RebootInterval = 10;  // 예외 발생 시 11초 후 OnTimeOutEvent가 발생하도록 설정
  smartWatchDog1.Start();              // SmartWatchDog를 시작
}
```

[STEP-2] 예외 발생 처리하기

SmartWatchDog 시작 후 예외 발생 시 RebootInterval 속성에서 설정한 시간이 지나면 OnTimeOutEvent가 발생됩니다. 이벤트의 코드 진행이 완료되면 IEC-Series는 자동으로 재부팅됩니다.

※ 자세한 내용은 "OnTimeOutEvent 이벤트"를 참고하시기 바랍니다.

```
// 예외 발생 시 호출되며, 예외에 따른 처리 코드를 작성. 코드의 처리가 끝나면 시스템이 재부팅됨
private void smartWatchDog1_OnTimeOutEvent(object sender, EventArgs e)
{
  // SmartTCPClient가 서버와 접속중인 경우 접속을 종료
  if (smartTCPClient1.IsConnect == true) { smartTCPClient1.Close(); }
  smartFile1.Open(); smartFile1.WriteString( " Error " ); smartFile1.Close();  // log 파일 작성
  smartFile2.Close(); smartFile3.Close();  // 현재 Open된 파일을 닫음
  // DB 접속이 되어있는 경우 접속 해제
  if (m_SqlCEDB.State == ConnectionState.Open) { m_SqlCEDB.Close(); }
}
```

CASE-2 Hardware WatchDog 사용하기

SmartWatchDog에서 HardwareWatchDog의 실행 여부를 GetHardwareWatchEnable() 메소드를 호출해 확인할 수 있으며, SetHardwareWatchEnable() 메소드를 사용해 Hardware WatchDog 기능을 끄고 켤 수 있습니다.

※ 자세한 내용은 "Hardware WatchDog와 SmartWatchDog 설명", "GetHardwareWatchEnable(), SetHardware WatchEnable() 메소드"를 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  // Hardware WatchDog이 현재 실행되었는지 확인
  if (smartWatchDog1.GetHardwareWatchEnable() == false)
  {
    smartWatchDog1.SetHardwareWatchEnable(true);  // 실행되지 않은 경우 Hardware WatchDog을 실행
    smartConfigs1.Powers.ReBoot();  // Hardware WatchDog 설정이 변경되어 재부팅
  }
}
```

4) SmartWatchDog 인터페이스 설명

SmartWatchDog Component Interface		
속성		
DebugMode : bool	RebootInterval : long	
메소드		
GetHardwareWatchDogEnable() : bool	SetHardwareWatchDogEnable(bool bEnable) : void	Start() : void
Stop() : void		
이벤트		
OnTimeOutEvent : EventHandler		

프로퍼티(속성) DebugMode

예외가 발생되어 시스템이 다시 시작되기 전 사용자에게 다음과 같은 메시지 상자를 출력하여 시스템을 재시작할지를 결정할 수 있도록 합니다. 프로그램 개발 단계에서 사용하시면 편리합니다.

- bool : 시스템 재시작 확인 여부
  - true : 확인함 (메시지박스 출력됨) / false : 확인 안 함 (메시지박스 출력 안 됨)

C# 사용법

```
// 시스템 재시작 여부를 확인함
smartWatchDog1.DebugMode = true;
```

VB 사용법

```
smartWatchDog1.DebugMode = true
```

프로퍼티(속성) RebootInterval

예외가 발생되고 OnTimeOutEvent 이벤트가 발생되기까지의 지연 시간을 설정합니다. 초 단위로 설정하며, 실제 지연 시간은 설정값 + 1초입니다.

- long : OnTimeOutEvent가 발생되기까지의 지연 시간(단위 : 초)

C# 사용법

```
// 실제 지연 시간은 4초입니다.
smartWatchDog1.RebootInterval = 3;
```

VB 사용법

```
smartWatchDog1.RebootInterval = 3
```

메소드(함수) GetHardwareWatchEnable

현재 Hardware WatchDog 기능이 활성화/비활성화되었는지 확인합니다.

◉ bool GetHardwareWatchEnable()

[리턴값]

- bool : Hardware WatchDog 기능의 활성화/비활성화 상태
  - true : Hardware WatchDog이 활성화 / false : Hardware WatchDog이 비활성화

C# 사용법

```
if(smartWatchDog1.GetHardwareWatchEnable() == false)
{
  // Hardware WatchDog이 비활성화된 경우 활성
  smartWatchDog1.SetHardwareWatchDogEnable(true);
}
```

VB 사용법

```
If smartWatchDog1.GetHardwareWatchEnable() = false Then
  smartWatchDog1.SetHardwareWatchDogEnable(true)
End If
```

메소드(함수) SetHardwareWatchEnable

Hardware WatchDog 기능을 활성화/비활성화합니다. 시스템 재시작 지연 시간은 10 ~ 60초 사이입니다.

참고 "Hardware WatchDog와 SmartWatchDog 설명" 내용을 참고하시기 바랍니다.

중요 IEC-Series 제품에서 Hardware WatchDog 설정이 변경될 경우 반드시 제품을 Rebooting(재시작)하셔야 합니다.

◉ `void SetHardwareWatchEnable(bool bEnable)`

[인자]

- `bool bEnable` : Hardware WatchDog 기능의 활성화/비활성화 여부
  - `true` : Hardware WatchDog을 활성화함 / `false` : Hardware WatchDog을 비활성화함

C# 사용법

```
// Hardware WatchDog 기능을 활성화
smartWatchDog1.SetHardwareWatchEnable(true);
```

VB 사용법

```
smartWatchDog1.SetHardwareWatchEnable(true)
```

메소드(함수) Start, Stop

- Start() : SmartWatchDog이 장치 응용 프로그램의 감시를 시작합니다.
- Stop() : SmartWatchDog이 장치 응용 프로그램의 감시를 중지합니다.

◉ `void Start()`
◉ `void Stop()`

C# 사용법

```
private void Form1_Load(object sender, EventArgs e)
{
  // 프로그램 실행 시 장치 응용 프로그램의 감시를 시작합니다.
  smartWatchDog1.Start();
}
private void Form1_Closing(object sender, CancelEventArgs e)
{
  // 프로그램 종료 시 장치 응용 프로그램의 감시를 중지합니다.
  smartWatchDog1.Stop();
}
```

VB 사용법

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
```

```
  smartWatchDog1.Start()
End Sub

Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As System.ComponentModel.
                          CancelEventArgs) Handles MyBase.Closing
  smartWatchDog1.Stop()
End Sub
```

이벤트 **OnTimeOutEvent**

장치 응용 프로그램에서 처리되지 않은 예외 발생 후 OnTimeOutEvent가 발생하며, 이벤트의 처리가 끝나면 시스템을 재시작하게 됩니다. OnTimeOutEvent에는 오류 발생에 따른 로그 저장 코드 또는 프로그램이 종료될 경우 각종 시스템 리소스 등을 해지하는 코드를 작성합니다. 예를 들면 스레드, 파일 Close, 네트워크 접속 해지, DB (DataBase) Close 등의 코드를 작성합니다.

중요 중단점(브레이크 포인트)을 걸어서 디버깅하는 경우 OnTimeOutEvent가 발생하지 않으므로, "디버깅하지 않고 시작"으로 솔루션 배포하여 테스트하시기 바랍니다.

C# 사용법

```
private void smartWatchDog1_OnTimeOutEvent(object sender, EventArgs e)
{
  // 예외에 따른 처리 코드를 작성. 코드의 처리가 끝나면 시스템이 재부팅됨
}
```

VB 사용법

```
Private Sub smartWatchDog1_OnTimeOutEvent(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles smartWatchDog1.OnTimeOutEvent
  ' 예외에 따른 처리 코드를 작성. 코드의 처리가 끝나면 시스템이 재부팅됨
End Sub
```

5) SmartWatchDog 예제 사용하기

SmartWatchDog를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartWatchDog"

12. SmartVideo

SmartVideo는 IEC667非Lite-Series에서 외부 카메라를 인터페이스 하기 위한 별도의 옵션 보드인 SmartVideoBoard를 간편하게 사용하도록 지원하는 컴포넌트입니다. 따라서 외부 카메라를 인터페이스 하기 위해서는 SmartVideoBoard를 별도로 구입하셔야 합니다. 인터페이스 가능한 카메라는 NTSC(National Television System Committee)의 신호를 출력하는 카메라를 장착하실 수 있습니다.

주의 SmartVideo 사용 시 주의사항

1. IEC667非Lite-Series에서만 지원되며, SmartVideo Board가 반드시 필요합니다.
2. SmartVideo를 사용 시 PortB 6, 7번 Pin을 사용하게 됩니다. SmartIIC도 동일한 Pin을 사용하므로 SmartVideo와 SmartIIC는 동시에 사용할 수 없습니다.
3. SmartVideo에서 동영상 Encoding 중 동시에 화면 캡쳐는 권장하지 않습니다. 스레드가 충돌할 수 있는 위험성이 존재합니다.
4. 개발된 Program에서 CPU Performance를 많이 사용하는 경우, 상/하/좌/우 화면 떨림이 발생할 수 있습니다. (화면의 크기와 비례하여 증가합니다.)

참조 장착 방법 및 하드웨어 관련 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → SmartVideo"를 참조하시기 바랍니다.

[SmartVideo Board]

[SmartVideo Board 사진 및 시스템 연결사 진]

1) 인터페이스 사양

아래 표에서 SmartVideo의 인터페이스 사양을 확인하여 적용하시기 바랍니다.

[표] SmartVideo 인터페이스 사양

입력 영상 신호 방식	NTSC(National Television System Committee)
입력 영상 채널 수	4 채널 / Capture & Preview (한 번에 한 채널만 선택 출력)
영상 속성 제어 기능	밝기(Brightness), 색조(Hue), 명암(Contrast), 채도(Color Saturation)
Preview 최대 해상도	800 x 600 → 해상도 가변
동영상 & 정지 영상 저장 최대 해상도	720 x 480 (640 x 480) → 해상도 가변
동영상 저장 Codec	H.264(10fps, 15fps, 20fps)
정지 영상 저장 Format	BMP, GIF, PNG, JPG
연결 가능한 IEC-Series 제품	※ IEC667非Lite-Series만 연결 가능합니다! ※ IEC667-07 [B1 or B2], IEC667-08[B1 or B2], IEC667-102[B1 or B2], IEC667-104[B1 or B2]
기타 부가 기능	1. Preview(동영상), Capture(정지 영상) 각각 별도 해상도 제어 2. 정지 영상 메모리 Capture 기능
지원 OS	Core - Standard , Core - Professional 지원

2) 프로그래밍 적용 가이드

STEP-1 출력 영상(Preview) 설정 및 영상 출력 시작하기

출력 영상의 밝기, 색조, 명암, 채도, 해상도, 입력 채널, 출력될 컨트롤을 설정 후 영상 출력을 시작합니다.

※ 자세한 내용은 “SmartVideo 인터페이스 설명의 각 속성 및 메소드” 내용을 참고하시기 바랍니다.

```
smartVideo1.BrightnessControl = 100;            // 밝기
smartVideo1.ColorSaturationControl = 200;       // 채도
smartVideo1.ContrastControl = 77;               // 명암
smartVideo1.HueControl = 33;                    // 색조
// Preview Window(Control) 핸들 설정 즉 설정된 Control로 영상 출력 : PictureBox로 설정
smartVideo1.PreviewWindowHandle = pictureBox1.Handle;
smartVideo1.SourceChannel = SmartX.SmartVideo.INPUTCHANNEL.CHANNEL1;  // 영상 입력 채널 1번 선택
// SmartVideo의 동작을 위한 초기화 및 초기 출력 영상 해상도 설정
if (smartVideo1.Initialization(SmartX.SmartVideo.PREVIEWRESOLUTIONS.PREVIEW160X96) == true)
{
  smartVideo1.Play();  // 영상 출력 시작
  // 출력 영상의 해상도 변경
  smartVideo1.SetPreviewResolution(SmartX.SmartVideo.PREVIEWRESOLUTIONS.PREVIEW320X240);
}
else
{
  SmartVideo 보드가 연결되지 않았을 때 처리 코드를 작성
}
```

STEP-2 정지 영상(Capture)의 저장 해상도 설정 및 저장하기

정지 영상의 저장 해상도 설정 후 정지 영상을 이미지로 저장하거나, 메모리로 저장할 수 있습니다.

※ 자세한 내용은 “SmartVideo 인터페이스 설명의 각 속성 및 메소드” 내용을 참고하시기 바랍니다.

```
// Capture 해상도 설정
smartVideo1.SetCaptureResolution(SmartX.SmartVideo.CAPTURERESOLUTIONS.CAPTURE320X240);
// 정지 영상을 PNG 이미지 파일로 저장
smartVideo1.CaptureStillImage( " SD Card₩₩SmartX.png " );
int iWidth = 0, iHeight = 0;
smartVideo1.GetCaptureResolution(ref iWidth, ref iHeight);
byte[] rawRGBData = new byte[(iWidth * iHeight) * 3];  // 정지 영상의 크기만큼 메모리 할당
Bitmap outBmp = new Bitmap(iWidth, iHeight);
smartVideo1.ImageMemoryCapture(rawRGBData);  // 메모리로 정지 영상 얻기
```

STEP-3 동영상 저장하기

현재 출력 중인 동영상을 저장합니다. 동영상은 H.264로 압축되어 저장됩니다.

※ 자세한 내용은 “SmartVideo 인터페이스 설명의 각 속성 및 메소드” 내용을 참고하시기 바랍니다.

```
// 동영상 저장을 위한 관련 파라미터 설정
smartVideo1.SetEncodingParameter1( " SD Card₩₩HNS.264 " , 30, 3000);
smartVideo1.SetEncodingParameter2(60, 15, 20, 0.75f);
smartVideo1.InitEncoding();  // 동영상 저장 초기화
smartVideo1.RunEncoding();   // 동영상 저장 시작
smartVideo1.EndEncoding();   // 동영상 저장 종료
```

3) SmartVideo 인터페이스 설명

SmartVideo Component Interface		
속성		
BrightnessControl: int	ColorSaturationControl: int	ContrastControl: int
HueControl: int	PreviewWindowHandle: IntPtr	SourceChannel: SmartVideo.INPUTCHANNEL
메소드		
CaptureStillImage(string strImgPathName) : bool	EndEncoding() : bool	GetCaptureResolution(ref int pWidth, ref int pHeight) : void
GetPreviewResolution(ref int pWidth, ref int pHeight) : void	ImageMemoryCapture(byte[] pRawByte) : bool	InitEncoding() : bool
Initialization(SmartVideo.PREVIEWRESOLUTIONS ePreviewResolution) : bool	Play() : bool	RunEncoding() : bool
SetCaptureResolution(SmartVideo.CAPTURERESOLUTIONS eCaptureResolution) : bool	SetDefaultControl() : void	SetEncodingParameter1(string strOuFile PathName, uint iframe_rate, uint ibitrate) : void
SetEncodingParameter2(uint igop_num, uint iIntraqp, uint iqpmax, float fgamma) : void	SetPreviewResolution(SmartVideo.PREVIEWRESOLUTIONS ePreviewResolution) : bool	Stop() : bool
StopEncoding() : bool		

프로퍼티(속성) BrightnessControl, ColorSaturationControl, ContrastControl, HueControl

출력 영상의 속성(밝기, 색조, 명암, 채도)값을 변경합니다.

- BrightnessControl : 영상의 밝기를 제어합니다. (기본값 : 128)
- ColorSaturationControl : 영상의 채도를 제어합니다. (기본값 : 128)
- ContrastControl : 영상의 명암을 제어합니다. (기본값 : 128)
- HueControl : 영상의 색조를 제어합니다. (기본값 : 0)
- int : 밝기, 색조, 명암, 채도 (범위 : 0 ~ 255)

C# 사용법

```
smartVideo1.BrightnessControl = 100;        // 밝기
smartVideo1.ColorSaturationControl = 200;   // 채도
smartVideo1.ContrastControl = 77;           // 명암
smartVideo1.HueControl = 33;                // 색조
```

VB 사용법

```
smartVideo1.BrightnessControl = 100
smartVideo1.ColorSaturationControl = 200
smartVideo1.ContrastControl = 77
smartVideo1.HueControl = 33
```

프로퍼티(속성) PreviewWindowHandle

비디오 영상이 출력되는 윈도우의 핸들을 지정하여 지정된 Window, Control 등에 Preview 화면을 출력 하도록 설정합니다.

- IntPtr : 컨트롤의 핸들

C# 사용법

```
// Preview Window(Control) 핸들 설정 즉 설정된 Control로 영상 출력
smartVideo1.PreviewWindowHandle = pictureBox1.Handle;
```

VB 사용법

```
smartVideo1.PreviewWindowHandle = pictureBox1.Handle
```

프로퍼티(속성) SourceChannel

입력 채널을 선택 합니다. SmartVideo Board에는 총 4개의 채널이 있습니다. 이중 출력을 원하는 채널을 선택합니다. (하나의 채널만 출력이 가능하며, 동시 출력을 지원하지 않습니다.)

- SmartVideo.INPUTCHANNEL.CHANNEL1 : 1번 채널
- SmartVideo.INPUTCHANNEL.CHANNEL2 : 2번 채널
- SmartVideo.INPUTCHANNEL.CHANNEL3 : 3번 채널
- SmartVideo.INPUTCHANNEL.CHANNEL4 : 4번 채널

C# 사용법

```
smartVideo1.SourceChannel = SmartVideo.INPUTCHANNEL.CHANNEL1;  // 영상 입력 채널 1번 선택
```

VB 사용법

```
smartVideo1.SourceChannel = SmartVideo.INPUTCHANNEL.CHANNEL1
```

메소드(함수) Initialization

SmartVideo를 사용하기 위하여 초기화 및 초기 영상 출력(Preview) 화면의 해상도를 설정합니다.

주의 초기값 설정은 Form_Load 부분에서 한 번만 하시기 바랍니다.

초기값 설정을 버튼 이벤트 등에 넣어서 두 번 이상 하는 경우 문제가 발생할 수 있습니다.

[C#] 올바른 사용 방법

```
private Form_Load (object sender, EventArgs e)
{
  // ----- 초기값 설정 코드는 생략되었습니다. -----
  if (smartVideo1.Initialization(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW160X96) == true)
  {
    smartVideo1.Play();  // 영상 출력 시작
  }
  else
  {
    // SmartVideo 보드가 연결되지 않았을 때 처리 코드 작성
  }
}
```

◉ bool Initialization(SmartVideo.PREVIEWRESOLUTIONS ePreviewResolution)

[인자]

- SmartVideo.PREVIEWRESOLUTIONS ePreviewResolution : 초기 Preview의 해상도를 설정합니다.

참고 열거형 데이터값은 "SetPreviewResolution() 메소드" 내용을 참고하시기 바랍니다.

[리턴값]

- bool : SmartVideo 연결 여부
  - true : IEC-Series에 SmartVideo가 연결됨
  - false : IEC-Series에 SmartVideo가 연결 안 됨

C# 사용법

```
// SmartVideo의 동작을 위한 초기화 및 초기 영상 출력 화면 해상도 설정
if (smartVideo1.Initialization(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW576X432) == false)
{
```

```
  // SmartVideo 보드가 연결되지 않았을 때 처리 코드 작성
}
else
{
  smartVideo1.Play();  // SmartVideo 보드가 연결되어 영상 출력 시작
}
```

VB 사용법

```
If smartVideo1.Initialization(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW576X432) = False Then
  ' SmartVideo 보드가 연결되지 않았을 때 처리 코드 작성
Else
  smartVideo1.Play()
End If
```

메소드(함수) SetDefaultControl

출력 영상의 속성(밝기, 색조, 명암, 채도)값을 기본값으로 설정합니다.

참고 기본 설정 값

- BrightnessControl : 밝기 128 설정
- ColorSaturationControl : 채도 128 설정
- ContrastControl : 명암 128 설정
- HueControl : 색조 0 설정

※ "BrightnessControl, ColorSaturationControl, ContrastControl, HueControl 속성" 내용을 참고하시기 바랍니다.

◉ void SetDefaultControl()

C# 사용법

```
// 출력 영상의 속성값(밝기, 색조, 명암, 채도)을 기본값으로 설정
smartVideo1.SetDefaultControl();
```

VB 사용법

```
smartVideo1.SetDefaultControl()
```

메소드(함수) SetCaptureResolution

동영상 및 정지 영상의 저장(Capture) 해상도를 설정합니다.

주의 Preview 해상도가 720 X 480 이상인 경우 Capture 해상도는 640 X 480까지 설정 가능합니다.
또한 160X96 크기로 저장된 동영상 파일은 PC에서 Play 되지 않습니다.

◉ bool SetCaptureResolution(SmartVideo.CAPTURERESOLUTIONS eCaptureResolution)

[인자]

- CAPTURERESOLUTIONS eCaptureResolution : Capture 시 가로/세로 해상도값

[표] CAPTURERESOLUTIONS 열거값에 따른 해상도

열거값	해상도	열거값	해상도
CAPTURE160X96	160 X 96	CAPTURE480X288	480 X 288
CAPTURE192X144	192 X 144	CAPTURE560X336	560 X 336
CAPTURE240X144	240 X 144	CAPTURE576X432	576 X 432
CAPTURE256X192	256 X 192	CAPTURE640X384	640 X 384
CAPTURE320X192	320 X 192	CAPTURE640X480	640 X 480
CAPTURE320X240	320 X 240	CAPTURE720X480	720 X 480
CAPTURE448X336	448 X 336	-	-

[리턴값]

- bool : 작업 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// Capture 해상도 설정
smartVideo1.SetCaptureResolution(SmartVideo.CAPTURERESOLUTIONS.CAPTURE320X240);
```

VB 사용법

```
smartVideo1.SetCaptureResolution(SmartVideo.CAPTURERESOLUTIONS.CAPTURE320X240)
```

메소드(함수) SetPreviewResolution

영상 출력(Preview) 해상도를 설정합니다.

주의 Preview 해상도가 720 X 480 이상인 경우 Capture 해상도는 640 X 480까지 설정 가능합니다.

◉ bool SetPreviewResolution(SmartVideo.PREVIEWRESOLUTIONS ePreviewResolution)

[인자]

- SmartVideo.PREVIEWRESOLUTIONS ePreviewResolution : 영상의 가로 X 세로 해상도값 설정

[표] CAPTURERESOLUTIONS 열거값에 따른 해상도

열거값	해상도	열거값	해상도
PREVIEW160X96	160 X 96	PREVIEW512X384	512 X 384
PREVIEW192X144	192 X 144	PREVIEW560X336	560 X 336
PREVIEW240X144	240 X 144	PREVIEW576X432	576 X 432
PREVIEW256X192	256 X 192	PREVIEW640X384	640 X 384
PREVIEW320X192	320 X 192	PREVIEW640X480	640 X 480
PREVIEW320X240	320 X 240	PREVIEW720X480	720 X 480
PREVIEW380X228	380 X 228	PREVIEW800X480	800 X 480
PREVIEW448X336	448 X 336	PREVIEW800X600	800 X 600
PREVIEW480X288	480 X 288	-	-

[리턴값]

- bool : 작업 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// 전체 화면 표시 처리
smartVideo1.SetPreviewResolution(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW800X600);
```

VB 사용법

```
smartVideo1.SetPreviewResolution(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW800X600)
```

메소드(함수) SetEncodingParameter1, SetEncodingParameter2

동영상을 저장하기 위하여 관련된 파라미터 값을 설정합니다. 동영상 Codec은 H.264로 저장됩니다.

주의 동영상의 저장 경로가 SD Card가 아닌 경우 저장이 되지 않으므로, 반드시 SD Card를 사용하시기 바랍니다. 또한, 동영상 저장 중 파일의 크기가 3.7GB 이상이 되는 경우 파일을 새로 생성하여 저장합니다. SD Card의 남은 용량이 60MB 이하인 경우 저장을 중단합니다.

⦿ void SetEncodingParameter1(string strOutFilePathName, uint iframe_rate, uint ibitrate)
⦿ void SetEncodingParameter2(uint igop_num, uint iIntraqp, uint iqpmax, float fgamma)

[인자]

• string strOutFilePathName : 저장될 동영상 파일의 이름 및 경로
• uint iframe_rate : 초당 프레임 수(설정값 : 10, 15, 20(fps))
• uint ibitrate : 비트 레이트(Bit Rate) (입력 범위 : 1 ~ 32767(kbps))

참고 비트 레이트(Bit Rate)란?

비트 레이트는 1초에 해당하는 동영상에 얼마의 비트(Bit) 수를 사용하는지를 의미합니다. 따라서 비트 레이트가 높을수록 동영상은 더 많은 정보(Bit)를 가지게 되므로 화질은 더 좋아지게 됩니다. 하지만 비트의 수는 그만큼 커지게 되므로 용량 또한 더 커지게 됩니다.

• uint igop_num : 프레임 묶음 길이(입력 범위 : 0 ~ 60)

참고 프레임 묶음 길이란?

프레임 간 압축 방식은 프레임 전체 정보가 다 들어있는 I 프레임을 기준으로 그 이후의 프레임은 이전 프레임 대비 변화량만을 기록하는 방식으로 데이터를 압축합니다. 변화량만을 기록하므로 움직임이 적을 경우 아주 적은 데이터만으로 한 프레임을 기록할 수 있습니다.
하지만 그렇다고 무작정 앞 프레임의 변화량을 저장하다가는 만약 데이터 유실에 치명적이고, 특정 위치의 프레임에 임의 접근 시 해당 프레임을 계산해 내는데 시간이 오래 걸리게 되므로 적절한 시간 간격으로 I 프레임을 삽입해 기준점을 다시 갱신해 주어야 합니다. 이렇게 하나의 I 프레임이 있으면 그 이후의 프레임은 특정 I 프레임에 종속적으로 됩니다. 따라서 하나의 I 프레임을 기준으로 다음 I 프레임이 나타날 때까지의 프레임들은 하나의 묶음으로 볼 수 있습니다.

이 묶음은 GOP라고 부르며, 묶음의 길이가 길수록 데이터 압축 효율은 올라가지만, 반응성은 떨어지고 데이터가 손실될 확률이 올라갑니다. 묶음의 길이가 짧을수록 데이터 압축 효율은 줄어들지만 반응성은 좋아지고 데이터 손실 확률이 낮아집니다.

[표] igop_num 인자 설정값에 따른 프레임 묶음

설정 값	설명	설정 값	설명
0	I, P, P, P, …	3	I, P, P, I, P, P, I, …
1	I, I, I, I, …	6	I, P, P, P, P, P, …
2	I, P, I, P, …		

- I(Intra Picture) : 전 후의 화면과는 관계없이 그 화면 내에서 독립적으로 압축하여 얻어지는 화면(Picture). 즉 시간 방향의 움직임 예측을 적용하지 않으면서 화면 내 정보만을 사용해서 압축 처리를 수행합니다.

- P(Predictive Picture) : 화면 간의 순방향 예측 압축에 의해 얻어지는 화면이다. 즉 I Picture 또는 P Picture를 화면(프레임)간 예측에 사용하여 압축 처리를 수행합니다.

• uint iIntraqp : Intra Picture의 양자화(Quantization) 스탭. 즉, 영상의 압축 정도를 설정하며 값이 클수록 압축률이 증가 (입력 범위 : 0 ~ 51)
• uint iqpmax : 최대 양자화 계수. iIntraqp와 같은 값으로 설정
• float fgamma : 감마 보정값. 영상 신호의 입력 레벨에 따라서 0%의 검정에서 100%의 흰색까지 밝기가 변화하도록 직선적으로 실시하는 보정이며 영상데이터의 출력 즉 브라운관이나 액정에서 각각 최적의 감마 보정을 하여 자연스러운 명암을 재현 할 수 있도록 설정 (기본값 : 0.75)

[표] 파라미터와 해상도에 따른 동영상 파일 크기

파라미터 설정값	Bitrate : 700(kbps), GOPNum : 60, InterQP : 51, QPMax : 51		
해상도	Frame Rate-10	Frame Rate-15	Frame Rate-20
192 X 144	4.95MB	4.98MB	4.99MB
320 X 240	4.97MB	4.99MB	4.99MB
640 X 480	4.97MB	5.00MB	5.01MB

C# 사용법

```
// 동영상 저장을 위한 관련 파라미터 설정
smartVideo1.SetEncodingParameter1( " SD Card₩₩HNS.264 " , 30, 3000);
smartVideo1.SetEncodingParameter2(60, 15, 20, 0.75f);
```

VB 사용법

```
smartVideo1.SetEncodingParameter1( " SD Card₩₩HNS.264 " , 30, 3000)
smartVideo1.SetEncodingParameter2(60, 15, 20, 0.75F)
```

메소드(함수) GetCaptureResolution

SetCaptureResolution() 메소드에서 설정된 영상(동영상, 정지 영상) 저장(Capture) 시 사용되는 가로/세로 해상도를 얻습니다.

◉ void GetCaptureResolution(ref int pWidth, ref int pHeight)

[인자]

- ref int pWidth : 가로 해상도
- ref int pHeight : 세로 해상도

C# 사용법

```
int iWidth = 0, iHeight = 0;
// Capture 해상도 설정
smartVideo1.SetCaptureResolution(SmartVideo.CAPTURERESOLUTIONS.CAPTURE800X600);
smartVideo1.GetCaptureResolution(ref iWidth, ref iHeight);  // 설정된 영상 저장 해상도 얻기
```

VB 사용법

```
Dim iWidth As Integer = 0, iHeight As Integer = 0
smartVideo1.SetCaptureResolution(SmartVideo.CAPTURERESOLUTIONS.CAPTURE800X600)
smartVideo1.GetCaptureResolution(iWidth, iHeight)
```

메소드(함수) GetPreviewResolution

SetPreviewResolution() 메소드에서 설정된 영상 출력(Preview) 화면의 가로/세로 해상도를 얻습니다.

◉ void GetPreviewResolution(ref int pWidth, ref int pHeight)

[인자]

- ref int pWidth : 가로 해상도
- ref int pHeight : 세로 해상도

C# 사용법

```
int iWidth = 0, iHeight = 0;
// 영상 출력 해상도 설정
smartVideo1.SetPreviewResolution(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW800X600);
smartVideo1.GetPreviewResolution(ref iWidth, ref iHeight);  // 설정된 영상 출력 해상도 얻기
```

VB 사용법

```
Dim iWidth As Integer = 0, iHeight As Integer = 0
smartVideo1.SetPreviewResolution(SmartVideo.PREVIEWRESOLUTIONS.PREVIEW800X600)
smartVideo1.GetPreviewResolution(iWidth, iHeight)
```

메소드(함수) Play, Stop

SmartVideo의 초기화 후 비디오 영상 출력(Preview)을 시작 및 종료하도록 합니다.

• Play() : 영상 출력을 시작합니다.
• Stop() : 영상 출력을 종료합니다.

◉ bool Play()
◉ bool Stop()

[리턴값]
• bool : 작업 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
smartVideo1.Play();  // 영상 출력 시작
smartVideo1.Stop();  // SmartVideo 종료
```

VB 사용법

```
smartVideo1.Play()
smartVideo1.Stop()
```

메소드(함수) CaptureStillImage

현재 영상의 정지 영상을 이미지 파일로 저장합니다. 저장 형식은 BMP, GIF, PNG, JPG 등으로 저장 가능합니다. 저장 시 적용되는 해상도는 SetCaptureResolution() 메소드에서 설정된 해상도로 저장됩니다.

◉ bool CaptureStillImage(string strImgPathName)

[인자]
• string strImgPathName : 저장될 정지 영상의 경로. 파일 형식은 파일 이름의 확장명에 따라서 적용됨

[리턴값]
• bool : 이미지 저장 성공 여부
  – true : 저장 성공
  – false : 저장 실패

C# 사용법

```
smartVideo1.CaptureStillImage( " SD Card₩₩SmartX.png " );  // 정지 영상을 PNG 이미지 파일로 저장
```

VB 사용법

```
smartVideo1.CaptureStillImage( " SD Card₩₩SmartX.png " )
```

메소드(함수) ImageMemoryCapture

현재 영상의 정지 영상을 메모리로 Capture합니다. 즉, 영상을 Capture하여 이미지 Processing을 할 수 있습니다. 입력 인자를 통하여 정지 영상 이미지의 데이터를 Raw Bmp의 바이트 스트림(stream)으로 얻으며, Capture 시 적용되는 해상도는 SetCaptureResolution() 메소드에서 설정된 해상도로 저장됩니다.

[표] 해상도별 Capture 소요 시간

열거값	소요 시간	열거값	소요 시간
160 X 96	17.2ms	480 X 288	87.7ms
192 X 144	31.8ms	560 X 336	109.3ms
240 X 144	34.7ms	576 X 432	162.3ms
256 X 192	36.2ms	640 X 384	155.7ms
320 X 192	44.5ms	640 X 480	185.8ms
320 X 240	55.5ms	720 X 480	220.5m
448 X 336	91.9ms	-	-

※ 위 측정 시간은 약간의 오차가 있을 수 있습니다.

◉ bool ImageMemoryCapture(byte[] pRawByte)

[인자]

• byte[] pRawByte : 정지 영상을 Capture하여 얻는 메모리 바이트 배열

중요 바이트 배열의 크기 및 멤버 순서

1. 바이트 배열의 크기
한 픽셀은 BGR 3개의 컬러로 이루어져 있습니다. 즉, 1 픽셀은 3Byte(24Bit)가 됩니다. 따라서 바이트 배열 크기 할당 시 전체 픽셀 수의 3배 크기를 할당해야 합니다.
※ 바이트 배열 크기 = 전체 픽셀 수 X 3

2. 바이트 배열의 멤버 순서
한 픽셀은 BGR 3개의 컬러로 이루어져 있으므로, 바이트 배열의 멤버는 B0(Blue), G0(Green), R0(Red), B1(Blue), G1(Green), R1(Red), ..., B-n(Blue), G-n(Green), R-n(Red) 순서로 저장됩니다.

[리턴값]

• bool : 작업 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
int iWidth = 0, iHeight = 0;
smartVideo1.GetCaptureResolution(ref iWidth, ref iHeight);
byte[] rawRGBData = new byte[(iWidth * iHeight) * 3];  // 정지 영상의 크기만큼 메모리 할당
Bitmap outBmp = new Bitmap(iWidth, iHeight);
smartVideo1.ImageMemoryCapture(rawRGBData);  // 메모리로 정지 영상 얻기
```

VB 사용법

```
Dim iWidth As Integer = 0, iHeight As Integer = 0
smartVideo1.GetCaptureResolution(iWidth, iHeight)
Dim rawRGBData((iWidth * iHeight) * 3) As Byte
Dim outBmp As Bitmap = New Bitmap(iWidth, iHeight)
smartVideo1.ImageMemoryCapture(rawRGBData)
```

메소드(함수) EndEncoding, InitEncoding, RunEncoding, StopEncoding

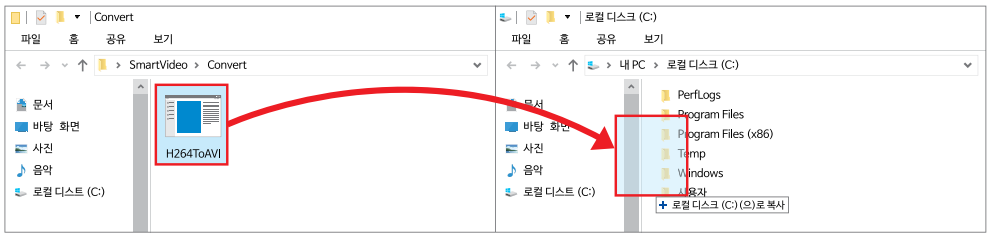
영상(동영상)을 저장하기 위한 초기화 및 시작, 일시 중지, 완료 등의 동작을 제어합니다. 저장된 동영상 파일은 단순히 H.264로 압축된 데이터 입니다. 따라서 PC에서 파일을 Play하기 위해서는 아래와 같이 변환하여 사용하시기 바랍니다. ※ 동영상 저장 시 약간의 시간이 발생할 수 있습니다.

• EndEncoding() : 동영상 저장을 완료(종료)합니다.
• InitEncoding() : 동영상을 저장을 위해 설정을 초기화합니다. 파라미터 값 등이 변경될 경우 호출합니다.
• RunEncoding() : 동영상 저장을 시작합니다.
• StopEncoding() : 동영상 저장을 잠시 중단합니다.

참고 H.264 압축 영상 파일을 AVI 확장자로 변경하는 방법

H.264 압축 영상 파일을 AVI 확장자로 변환하기 위해 변환 툴을 제공하고 있습니다. 변환 툴은 "SmartVideo 예제 폴더 → Convert 폴더 → H264ToAVI.exe"이며 CMD를 사용하여 영상 파일을 변환할 수 있습니다.

[STEP-1] "SmartVideo 예제 폴더 → Convert 폴더 → H264ToAVI.exe" 파일을 C드라이브에 이동



[STEP-2] CMD 실행 후 실행 경로 설정

CMD 실행 후 cd C:₩를 입력합니다.

[STEP-3] H264ToAVI [-f options] [-i input.h264] [-o output.avi] 명령어를 사용하여 영상을 변환

H264ToAVI [-f options] [-i input.h264] [-o output.avi]

- -f : Frame Rate(저장 시 사용했던 값으로 설정. Ex : -f 10, -f 15, -f 20)
- -i : H.264 압축 영상 파일
- -o : AVI 형식으로 변환될 파일

◉ bool EndEncoding()
◉ bool InitEncoding()
◉ bool RunEncoding()
◉ bool StopEncoding()

[리턴값]

- bool : 작업 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// 동영상 저장을 위한 관련 파라미터 설정
smartVideo1.SetEncodingParameter1( " SD Card₩₩test.264 " , 30, 3000);
smartVideo1.SetEncodingParameter2(60, 15, 20, 0.75f);

// 동영상 저장 초기화
smartVideo1.InitEncoding();
// 동영상 저장 시작
smartVideo1.RunEncoding();
```

```
// 동영상 저장 잠시 중지
smartVideo1.StopEncoding();
// 동영상 저장 종료
smartVideo1.EndEncoding();
```

VB 사용법

```
smartVideo1.SetEncodingParameter1( " SD Card₩₩test.264 ", 30, 3000);
smartVideo1.SetEncodingParameter2(60, 15, 20, 0.75f);
smartVideo1.InitEncoding();
smartVideo1.RunEncoding();
smartVideo1.StopEncoding();
smartVideo1.EndEncoding();
```

4) SmartVideo 예제 사용하기

SmartVideo를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

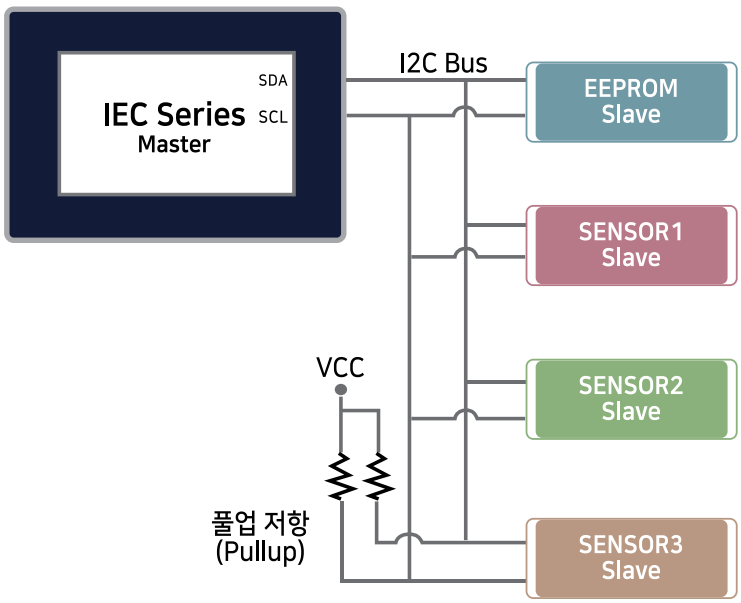
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartVideo"

13. SmartIIC

SmartIIC(I2C)는 IIC(Inter-Integrated Circuit) 통신을 편리하게 사용하도록 만든 컴포넌트입니다. IIC는 2개의 신호선 SDA와 SCL을 사용하고 있으며 외부 디바이스인 경우 같은 GND를 사용하고 있어야 합니다. 2개의 신호선은 풀업(Pull Up)되어 있으며 여러 개의 슬레이브(Slave) 디바이스들과 1:N 통신을 할 수 있습니다.

IEC-Series에서 SmartIIC는 항상 마스터(Master)로 사용됩니다. IIC의 SDA와 SCL은 IEC-Series의 Extension Port에서 Port-B6/SDA이며, Port-B7/SCL입니다. 따라서 IIC를 사용할 경우 Port-B6~7은 GPIO가 아닌 IIC로 사용됩니다.



참고 SmartIIC 사용 시 참고사항

1. SmartIIC를 사용하여 외부의 디바이스와 연동(통신 프로그래밍) 시 문제 및 적용에 어려움이 있는 경우 HNS로 연락주시면 IEC-Series와 디바이스 연동 작업을 지원하도록 하겠습니다. 지원 시 SmartIIC를 기준으로 합니다.
2. 콘넥터 핀 정보에 관한 설명은 "Part-VI. 하드웨어 장치 안내"를 참고하시기 바랍니다.

주의 SmartGPIO와 SmartIIC를 함께 사용하는 경우 프로그램이 정상 동작하지 않을 수 있습니다.
이 경우 SmartGPIO의 속성 중 Port-B의 DIR6, DIR7을 OUTPUT으로 설정하기 바랍니다.

1) 프로그래밍 적용 가이드

STEP-1 기본 설정하기

먼저 I2C 통신을 위해 Initial() 메소드를 호출해 각종 속성 및 Port를 초기화합니다. 이후 통신할 디바이스(메모리, 센서 등)의 I2C 데이터 시트를 참조하여 Speed 속성으로 통신 속도를, ACKWaitInterval 속성으로 ACK 응답 대기 시간을 설정하시기 바랍니다.

※ 자세한 내용은 "Speed, ACKWaitInterval 속성", "Initial() 메소드"를 참고하시기 바랍니다.

```
smartIIC1.Initial();  // I2C를 사용하기 위해 Speed 및 I2C 관련 기능을 초기화합니다.
smartIIC1.Speed = 0;  // I2C 통신 속도 설정
smartIIC1.ACKWaitInterval = 0xFFFF;  // ACK Wait Timeout 값 설정
```

STEP-2 IIC 버스로 특정 디바이스의 데이터 읽기/쓰기 처리

I2C 버스와 데이터 송/수신하기 위해서는 통신할 디바이스(메모리, 센서 등)의 I2C 프로토콜을 알아야 합니다. 따라서 해당 디바이스의 데이터 시트를 참고하여 프로그램을 수정하시기 바랍니다. 본 가이드는 일반적인 I2C 읽기/쓰기 처리를 설명합니다.

[쓰기 처리 기본 과정]

1. 기본적으로 Start Signal과 Stop Signal 사이에 쓰기 명령만 처리되는 경우를 말합니다.
2. Start Signal 전송으로 시작합니다.

```
smartIIC1.Start_Signal();
```

3. 쓰기 처리 시 Slave address는 통상적으로 마지막 Bit(LSB)는 0으로 아래와 같이 두 가지 방식 중 한 가지 방식으로 처리됩니다.

```
smartIIC1.Write_Byte(iSlave_Address);  or  smartIIC1.Write_Byte((byte)(iSlave_Address << 1));
```

4. 다음은 쓰기에 필요한 각종 데이터를 전송합니다.

```
smartIIC1.Write_Byte(Command);
smartIIC1.Write_Byte(Data1);
smartIIC1.Write_Byte(Data2);
```

5. 쓰기 처리가 끝나면 Stop Signal을 전송합니다.

```
smartIIC1.Stop_Signal();
```

[읽기 처리 기본 과정]

1. 기본적으로 Start Signal과 Stop Signal 사이에 읽기 명령만 처리되는 경우를 말합니다.
2. 읽기 명령은 쓰기 명령과 다르게 읽기 처리하기 위해서 필요한 정보를 먼저 쓰기 명령으로 처리를 완료한 후 읽기 처리를 하는 구조로 작성합니다.
3. 읽기 위한 정보를 쓰기 명령으로 처리합니다.

```
smartIIC1.Start_Signal();
smartIIC1.Write_Byte(iSlave_Address);  or  smartIIC1.Write_Byte((byte)(iSlave_Address << 1));
smartIIC1.Write_Byte(Command);
smartIIC1.Write_Byte(Data1);
smartIIC1.Write_Byte(Data2);
smartIIC1.Stop_Signal();  // 다른 읽기 명령 처리 시 경우에 따라서 생략되는 경우도 있습니다.
```

4. 읽기 처리 주요 처리 사항

- 읽기 명령도 Start Signal로 시작해서 Stop Signal로 종료합니다.
- 읽기 처리 시 통상적으로 Slave address의 마지막 Bit(LSB)는 1로 처리합니다. 아래와 같이 두 가지 방식 중 한 가지 방식으로 처리됩니다.
- Read_Byte 처리 시 마지막은 NOACK로 처리합니다.

```
smartIIC1.Start_Signal();
smartIIC1.Write_Byte(iSlave_Address + 1));  or  smartIIC1.Write_Byte((byte)((iSlave_Address << 1) + 1));
iHData = smartIIC1.Read_Byte(SmartX.SmartIIC.ACKSIG.ACK);
iLData = smartIIC1.Read_Byte(SmartX.SmartIIC.ACKSIG.NOACK);
smartIIC1.Stop_Signal();
```

※ 자세한 내용은 “Start_Signal(), Stop_Signal(), Write_Byte(), Read_Byte() 메소드”를 참고하시기 바랍니다.

```
// 지정된 주소 디바이스의 데이터를 읽기 처리. 적용할 디바이스의 Data Sheet를 참조하세요.
private byte ReadData(byte iSlaveAddress, byte iCommand)
{
  smartIIC1.Start_Signal();
  smartIIC1.Write_Byte(iSlaveAddress); smartIIC1.Write_Byte(iCommand);
```

```
  smartIIC1.Stop__Signal();
  smartIIC1.Start_Signal();
  smartIIC1.Write_Byte((byte)(iSlaveAddress | 1));
  byte iData = smartIIC1.Read_Byte(SmartX.SmartIIC.ACKSIG.NOACK);  // 지정된 주소의 데이터를 읽습니다.
  smartIIC1.Stop_Signal();
  return iData;
}
// 지정된 주소 디바이스의 데이터를 쓰기 처리. 적용할 디바이스의 Data Sheet를 참조하세요.
private void WriteData(byte iSlaveAddress, byte iCommand, byte iData)
{
  smartIIC1.Start_Signal();
  smartIIC1.Write_Byte(iSlaveAddress);
  smartIIC1.Write_Byte(iCommand);
  smartIIC1.Write_Byte(iData);
  smartIIC1.Stop_Signal();
}
```

2) SmartIIC 인터페이스 설명

SmartIIC Component Interface		
속성		
ACKWaitInterval : ulong	Speed : long	
메소드		
Initial() : void	Read_Byte(SmartIIC.ACKSIG eACK) : byte	Start_Signal() : void
Stop_Signal() : void	Write_Byte(byte uchData) : bool	

프로퍼티(속성) ACKWaitInterval

IIC(I2C) 통신 중 Read_Byte() 메소드 호출 시 인자값을 SmartIIC.ACKSIG.ACK로 한 경우, 내부적으로 ACK 응답이 수신될 때까지 기다리게 됩니다. 이러한 경우 만약 상대방 디바이스에서 ACK를 보내지 않게 되면 계속 응답을 기다리게 되어 프로그램이 정상적으로 동작하지 않게 되는 문제가 발생합니다. 이러한 문제를 방지하기 위하여 일정 시간 ACK 응답이 없을 경우 ACK를 SKIP할 수 있도록 ACK Wait Timeout하는 기능입니다.

[인자]

• ulong : ACK Wait Timeout 값

C# 사용법

```
smartIIC1.ACKWaitInterval = 0xFFFF;  // ACK Wait Timeout 값 설정
```

VB 사용법

```
smartIIC1.ACKWaitInterval = &HFFFF
```

프로퍼티(속성) Speed

IIC(I2C) 통신 속도와 관련된 설정값입니다. 하나의 단위 신호당 1/1000초로 설정되며, 최대 속도는 90Khz입니다.

[인자]

• long : IIC 통신 속도 (기본값 : 0)

C# 사용법

```
smartIIC1.Speed = 0;  // I2C 통신 속도 설정
```

VB 사용법

```
smartIIC1.Speed = 0
```

메소드(함수) Initial

IIC(I2C) 통신을 사용하기 위해서 Speed 및 Port의 IIC(I2C) 관련 기능을 초기화합니다.

◉ void Initial()

C# 사용법

```
smartIIC1.Initial();
```

VB 사용법

```
smartIIC1.Initial()
```

메소드(함수) Start_Signal, Stop_Signal

IIC(I2C) 버스 상에 있는 마스터(IEC-Series)는 데이터 송/수신을 시작할 때에 시작조건을 발생하고 데이터의 송/수신을 끝낼 때에는 정지조건을 발생시킵니다. 시작 조건을 발생시킨 마스터는 정지조건을 발생할 때까지 버스를 점유하게 되고 이때 다른 마스터는 버스를 사용할 수 없게 됩니다.

SmartIIC의 동작 모드는 마스터 수신 모드와 마스터 송신 모드가 있으며, 각 메소드는 SmartIIC 동작 모드의 시작 및 정지조건 신호를 발생시킵니다.

◉ void Start_Signal() : 마스터 송/수신 모드의 시작조건 신호를 발생시킵니다.

◉ void Stop_Signal() : 마스터 송/수신 모드의 정지조건 신호를 발생시킵니다.

C# 사용법

```
smartIIC1.Start_Signal();  // 시작조건 신호를 발생합니다.
smartIIC1.Stop_Signal();   // 정지조건 신호를 발생합니다.
```

VB 사용법

```
smartIIC1.Start_Signal() : smartIIC1.Stop_Signal()
```

메소드(함수) Read_Byte

IIC(I2C) 버스에서 데이터를 수신합니다. 인자로는 ACK와 NOACK가 있으며, ACK인 경우 ACK가 수신될 때까지 기다립니다. 만약 ACK 응답이 없을 경우 ACKWaitInterval 속성에서 설정된 값에 따라 ACK WaitTimeOut 처리됩니다.

◉ byte Read_Byte(SmartIIC.ACKSIG eACK)

[인자]

- SmartIIC.ACKSIG.ACK : ACK 응답을 기다림
- SmartIIC.ACKSIG.NOACK : ACK 응답을 기다리지 않음

[리턴값]

- byte : 수신받은 데이터

C# 사용법

```
byte retVal = smartIIC1.Read_Byte(SmartIIC.ACKSIG.ACK);  // 바이트 데이터를 읽고 ACK 응답 대기
```

VB 사용법

```
Dim retVal As Byte = smartIIC1.Read_Byte(SmartIIC.ACKSIG.ACK)
```

메소드(함수) Write_Byte

IIC(I2C) 버스에 Byte 데이터를 송신합니다.

◉ bool Write_Byte(byte uchData)

[인자]

• byte uchData : 송신할 Byte 데이터

[리턴값]

• bool : 데이터 송신 성공 여부를 리턴
  – true : 데이터 송신 성공
  – false : 데이터 송신 실패

C# 사용법

```
if (smartIIC1.Write_Byte(0x46) == true)
{
  // 데이터 송신 후 성공 여부 확인
}
```

VB 사용법

```
If smartIIC1.Write_Byte(&H46) = True Then
End If
```

3) SmartIIC 예제 사용하기

SmartIIC를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartIIC"

14. SmartPrint

SmartPrint 컴포넌트는 기존의 임베디드 시스템에서 가장 취약했던 인쇄 관련 기능을 IEC-Series에서 편리하게 프린터 출력할 수 있도록 만들어진 컴포넌트입니다. 기존의 복잡하고 어려운 인쇄 관련 기능을 쉽게 프로그래밍할 수 있도록 했으며, 또한 프린터 드라이버를 별도로 설치하지 않아도 사용할 수 있습니다.

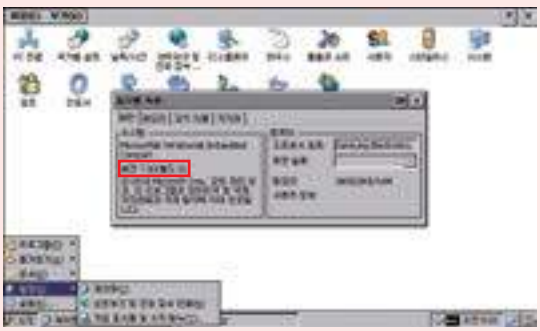
- PCL(Printer Command Language)1~6 지원 (PCL3 최적)
- USB Type 프린터 지원

1) 사양 및 지원사항

SmartPrint는 PCL을 지원하는 프린터에서만 지원되도록 만들어져 있습니다. 현재 호환성이 확인된 프린터는 HP사의 OFFICE JET PRO 8210 (테스트 일자 : 2016년 10월 28일)입니다. 보유하신 프린터와 IEC-Series의 호환성을 검증하고 싶은 경우 HNS로 프린터를 보내주시면 테스트를 진행하여 호환성 여부를 알려드리도록 하겠습니다.

주의 SmartPrint 사용시 O/S 빌드 버전에 따른 주의사항

SmartPrint의 지원 O/S 빌드 버전은 IEC266-Series인 경우 빌드 3이며, IEC667-Series는 빌드 2입니다.
이보다 낮은 빌드 버전에서는 지원이 안됩니다. 이럴 경우 HNS로 제품을 보내주시면 업그레이드 해드리고 있습니다. "시작 버튼 → 설정 → 제어판 → 시스템(시스템 속성) → 일반 탭"에서 O/S 빌드 버전을 확인하시기 바랍니다.



참조 빌드 관련 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 운영 체제 빌드 버전 관련"을 참조하시기 바랍니다.

2) 프로그래밍 적용 가이드

STEP-1 프린터 설정하기

프린터를 사용하기 전 프린터 설정을 진행합니다.

※ 프린터를 사용하기 전 반드시 PrintSetup() 메소드를 호출해야 하며, 자세한 내용은 "PrinterSetup() 메소드" 내용을 참고하시기 바랍니다.

```
// 방법 1. 각 설정을 일괄로 처리
smartPrint1.PrintSetup( " Printer " , SmartX.SmartPrint.PRINTQUALITY.XX, SmartX.SmartPrint.COLORMODE.XX,
SmartX.SmartPrint.PAPERSIZE.XX, SmartX.SmartPrint.ORGDIRECTION.XX, 1);
// 방법 2. 각 속성값 설정 후 PrintSetUP() 호출
smartPrint1.PaperSize = SmartX.SmartPrint.PAPERSIZE.XXX;
```

```
smartPrint1.PaperDirection = SmartX.SmartPrint.ORGDIRECTION.XXX;
smartPrint1.PrintQuality = SmartX.SmartPrint.PRINTQUALITY.XXX;
smartPrint1.COLOR = SmartX.SmartPrint.COLORMODE.XXX;
smartPrint1.PrintSetup();
```

STEP-2 프린터 버퍼에 그리기

프린터 버퍼에 다양한 도형, 문자열, 이미지를 그립니다. 그리기 전 반드시 SetPenStyle(), SetBrushStyle(), SetFontCfg() 메소드를 호출하여 선 스타일, 내부 채움 스타일, 폰트 스타일을 설정해야 합니다.

※ 예제에 사용된 메소드 이외에 다양한 메소드가 있으니 "인터페이스 설명" 내용을 확인해보시기 바랍니다.

```
// 도형의 내부를 투명하게 채운다.
smartPrint1.SetBrushStyle(Color.White, true);
// 도형의 선 색상 및 두께 설정
smartPrint1.SetPenStyle(Color.Red, 3);
// Font 관련 설정
smartPrint1.SetFontCfg(10, Color.Black);
// 프린터 버퍼에 원 그리기
smartPrint1.Circle(300, 300, 50);
// 프린터 버퍼에 선 그리기
smartPrint1.Line(100, 100, 200, 200);
// 리소스에 포함된 이미지 파일을 원본 이미지 크기로 프린터 버퍼에 저장
smartPrint1.ImageDraw(Resource1.SmartX_Logo, 10, 10);
// 문자열을 프린터 버퍼에 저장
smartPrint1.TextOut(200, 70, " SmartPrint " );
```

STEP-3 프린터 버퍼에 그려진 데이터 인쇄하기

프린터 버퍼에 그려진 데이터를 프린터로 전송하여 인쇄를 시작합니다. IsPrinting() 메소드를 호출하여 인쇄 상태를 확인할 수 있습니다.

※ PintOut() 메소드 호출 시 1페이지 단위로 인쇄되며, 새로운 페이지는 NePage() 메소드를 호출하여 생성할 수 있습니다.

```
smartPrint1.PrintOut();  // 프린터 버퍼에 그려진 데이터의 인쇄를 시작
// 현재 프린터의 상태를 주기적으로 체크
private void smartTimer1_Tick(object sender, EventArgs e)
{
  switch (smartPrint1.IsPrinting())
  {
    case SmartX.SmartPrint.PRINTOUTPUTSTATUS.PRINTFAILURE:
      // 인쇄 실패
      smartTimer1.Stop();
      break;
    case SmartX.SmartPrint.PRINTOUTPUTSTATUS.PRINTING:
      // 인쇄 중
      break;
    case SmartX.SmartPrint.PRINTOUTPUTSTATUS.PRINTSTANDBY_PRINTSUCCESS:
      // 인쇄 완료 or 준비 중
      smartTimer1.Stop();
      break;
  }
}
```

3) SmartPrint 인터페이스 설명

SmartPrint Component Interface		
속성		
COLOR : SmartPrint.COLORMODE	PaperDirection : SmartPrint.ORGDIRECTION	PaperSize : SmartPrint.PAPERSIZE
PrintQuality : SmartPrint.PRINTQUALITY		
메소드		
Circle(int iCenX, int iCenY, int iDiameter) : void	DisplayToPrint() : void (+1개 오버로드)	Ellipse(int iCenX, int iCenY, int iWidth, int iHeight) : void
Erase() : void	GetPaperPixel(ref int iWidthPixel, ref int iHeightPixel) : void	GradientFill(int iLeftX, int iTopY, int iWidth, int iHeight, Color startColor, Color endColor, SmartPrint.GRADIENTFILLMODE dwMode) : void
ImageDraw() : void (+3개 오버로드)	IsPrinting() : SmartPrint.PRINTOUTPUTSTATUS	Line(int ifx, int ify, int isx, int isy) : void
NewPage() : void	Polygon(SmartPrint.POINT[] pPoint) : void	Polyline(SmartPrint.POINT[] pPoint) : void
PrintOut() : void	PrintSetup() : bool (+1개 오버로드)	Rectangle(int iLeftX, int iTopY, int iWidth, int iHeight) : void
RoundRect() : void (+1개 오버로드)	SetBrushStyle() : void (+1개 오버로드)	SetFontCfg() : void (+2개 오버로드)
SetPenStyle(Color penColor, int iWidth) : void	SetPixel(int iXPos, int iYPos,Color crColor) : void	TextOut(int iXpos, int iYpos, string strFormat, params object[] args) : void

프로퍼티(속성) Color

인쇄 색상 모드를 설정합니다. 컬러 프린터에서 출력 내용이 컬러로 작업되어 있더라도 이 속성값에 의해 흑백으로 출력할 수 있습니다.

- SmartPrint.COLORMODE.COLOR : 인쇄 색상 모드를 컬러 모드로 설정(기본값)
- SmartPrint.COLORMODE.MONOCHROME : 인쇄 색상 모드를 흑백 모드로 설정

C# 사용법

```
// 인쇄 색상 모드를 컬러 모드로 설정
smartPrint1.COLOR = SmartPrint.COLORMODE.COLOR;
```

VB 사용법

```
smartPrint1.COLOR = SmartPrint.COLORMODE.COLOR
```

프로퍼티(속성) PaperDirection

인쇄할 용지의 방향을 설정합니다.

- SmartPrint.ORGDIRECTION.HORIZON : 용지 방향을 가로 방향으로 설정
- SmartPrint.ORGDIRECTION.VERTICAL : 용지 방향을 세로 방향으로 설정(기본값)

C# 사용법

```
// 용지 방향을 세로 방향으로 설정
smartPrint1.PaperDirection = SmartPrint.ORGDIRECTION.VERTICAL;
```

VB 사용법

```
smartPrint1.PaperDirection = SmartPrint.ORGDIRECTION.VERTICAL
```

프로퍼티(속성) PaperSize

인쇄할 용지의 크기를 설정합니다.

- SmartPrint.PAPERSIZE.A4 : 인쇄 용지를 A4(210 X 297 mm)로 설정(기본값)
- SmartPrint.PAPERSIZE.B5 : 인쇄 용지를 B5(JIS)(182 X 257 mm)로 설정
- SmartPrint.PAPERSIZE.LEGAL : 인쇄 용지를 Legal(8 1/2 X 14 inch)로 설정
- SmartPrint.PAPERSIZE.LETTER : 인쇄 용지를 Letter(8 1/2 X 11 inch)로 설정

C# 사용법

```
// 인쇄 용지를 A4로 설정
smartPrint1.PaperSize = SmartPrint.PAPERSIZE.A4;
```

VB 사용법

```
smartPrint1.PaperSize = SmartPrint.PAPERSIZES.A4
```

프로퍼티(속성) PrintQuality

인쇄 해상도(품질)를 설정합니다. 상대적으로 해상도가 높아지면 인쇄 속도가 느려집니다.

- SmartPrint.PRINTQUALITY.DPI300 : 인쇄 해상도를 300DPI(Dots Per Inch)로 설정(기본값)
- SmartPrint.PRINTQUALITY.DPI600 : 인쇄 해상도를 600DPI(Dots Per Inch)로 설정

C# 사용법

```
// 인쇄 해상도를 600DPI로 설정
smartPrint1.PrintQuality = SmartPrint.PRINTQUALITY.DPI600;
```

VB 사용법

```
smartPrint1.PrintQuality = SmartPrint.PRINTQUALITY.DPI600
```

메소드(함수) PrintSetup

프린터 출력을 위한 프린터 관련 환경 설정 및 초기화를 진행합니다. 인자값을 사용하지 않을 경우 COLOR, PaperDirection, PaperSize, PrintQuality 속성에서 설정된 값으로 프린터를 초기화합니다.

주의 프린터를 사용하기 전 반드시 PrintSetup() 메소드를 호출하시기 바랍니다.

참고 "COLOR, PaperDirection, PaperSize, PrintQuality 속성" 내용을 참고하시기 바랍니다.

◉ bool PrintSetup()
◉ bool PrintSetup(string strPrinterID, SmartPrint.PRINTQUALITYS iQuality, SmartPrint.COLORMODE iColorMode, SmartPrint.PAPERSIZES iPaperSize, SmartPrint.ORGDIRECTION iOrientation, int iCopies)

[인자]

- string strPrinterID : 사용자 지정 프린터 이름
- SmartPrint.PRINTQUALITYS iQuality : 해상도(DPI)
- SmartPrint.COLORMODE iColorMode : 컬러 모드
- SmartPrint.PAPERSIZES iPaperSize : 용지 크기
- SmartPrint.ORGDIRECTION iOrientation : 용지 방향
- int iCopies : 인쇄할 페이지 수

[리턴값]

- bool : 설정 성공 여부를 리턴
  - true : 설정 완료
  - false : 설정 실패

C# 사용법

```
smartPrint1.PrintSetup( " PCL Inkjet " , SmartPrint.PRINTQUALITY.DPI600, SmartPrint.COLORMODE.COLOR,
SmartPrint.PAPERSIZE.A4, SmartPrint.ORGDIRECTION.VERTICAL, 1);
```

VB 사용법

```
smartPrint1.PrintSetup( " PCL Inkjet " , SmartPrint.PRINTQUALITY.DPI600, SmartPrint.COLORMODE.COLOR,
SmartPrint.PAPERSIZE.A4, SmartPrint.ORGDIRECTION.VERTICAL, 1)
```

메소드(함수) GetPaperPixel

현재 PaperSize 속성에서 설정된 용지의 가로, 세로 크기를 Pixel 단위의 값으로 얻습니다. 프린터마다 Pixel 값이 실제와 약간 다를 수 있습니다.

◉ void GetPaperPixel(ref int iWidthPixel, ref int iHeightPixel)

[인자]

- ref int iWidthPixel : 용지의 가로 크기(Pixel)
- ref int iHeightPixel : 용지의 세로 크기(Pixel)

C# 사용법

```
int iWidthPage_Pixel, iHeightPage_Pixel;
smartPrint1.GetPaperPixel(ref iWidthPage_Pixel, ref iHeightPage_Pixel);
```

VB 사용법

```
Dim iWidthPage_Pixel, iHeightPage_Pixel As Integer
smartPrint1.GetPaperPixel(iWidthPage_Pixel, iHeightPage_Pixel)
```

메소드(함수) NewPage

여러 페이지를 출력하는 경우, 각각의 새로운 페이지를 생성할 때 호출합니다. 즉, 새로운 페이지를 만듭니다.

◉ void NewPage()

C# 사용법

```
smartPrint1.TextOut(140, 990, " SmartPrint Page-1 ");  // 페이지에 문자를 출력
smartPrint1.PrintOut();  // 프린터 버퍼에 출력된 데이터를 인쇄
smartPrint1.NewPage();   // 새로운 페이지를 생성
```

VB 사용법

```
smartPrint1.TextOut(140, 990, " SmartPrint Page-1 " )
smartPrint1.PrintOut()
smartPrint1.NewPage()
```

메소드(함수) IsPrinting

프린터의 인쇄 상태를 확인합니다.

◉ SmartPrint.PRINTOUTPUTSTATUS IsPrinting()

[리턴값]

- SmartPrint.PRINTOUTPUTSTATUS : 프린터의 인쇄 상태
  - SmartPrint.PRINTOUTPUTSTATUS.PRINTSTANDBY_PRINTSUCCESS : 인쇄 성공
  - SmartPrint.PRINTOUTPUTSTATUS.PRINTING : 인쇄 중
  - SmartPrint.PRINTOUTPUTSTATUS.PRINTFAILURE : 인쇄 실패

C# 사용법

```
if (smartPrint1.IsPrinting() == SmartPrint.PRINTOUTPUTSTATUS.PRINTSTANDBY_PRINTSUCCESS)
{
  // 현재 프린터가 인쇄 중
}
```

VB 사용법

```
If smartPrint1.IsPrinting() = SmartPrint.PRINTOUTPUTSTATUS.RINTSTANDBY_PRINTSUCCESS Then
   ' 현재 프린터가 인쇄 중
End If
```

메소드(함수) DisplayToPrint

현재 IEC-Series의 LCD 화면에 출력된 상태를 프린터로 출력합니다.

◉ void DisplayToPrint(int iXpos, int iYpos)

◉ void DisplayToPrint(int iXpos, int iYpos, int iWidth, int iHeight)

[인자]

- int iXpos : 출력될 이미지 좌측 상단의 X좌표 (필수)
- int iYpos : 출력될 이미지 좌측 상단의 Y좌표 (필수)
- int iWidth : 출력될 이미지 폭 (옵션 : Scale 조정 시 사용)
- int iHeight : 출력될 이미지 높이 (옵션 : Scale 조정 시 사용)

C# 사용법

```
smartPrint1.DisplayToPrint(100, 300);
```

VB 사용법

```
smartPrint1.DisplayToPrint(100, 300)
```

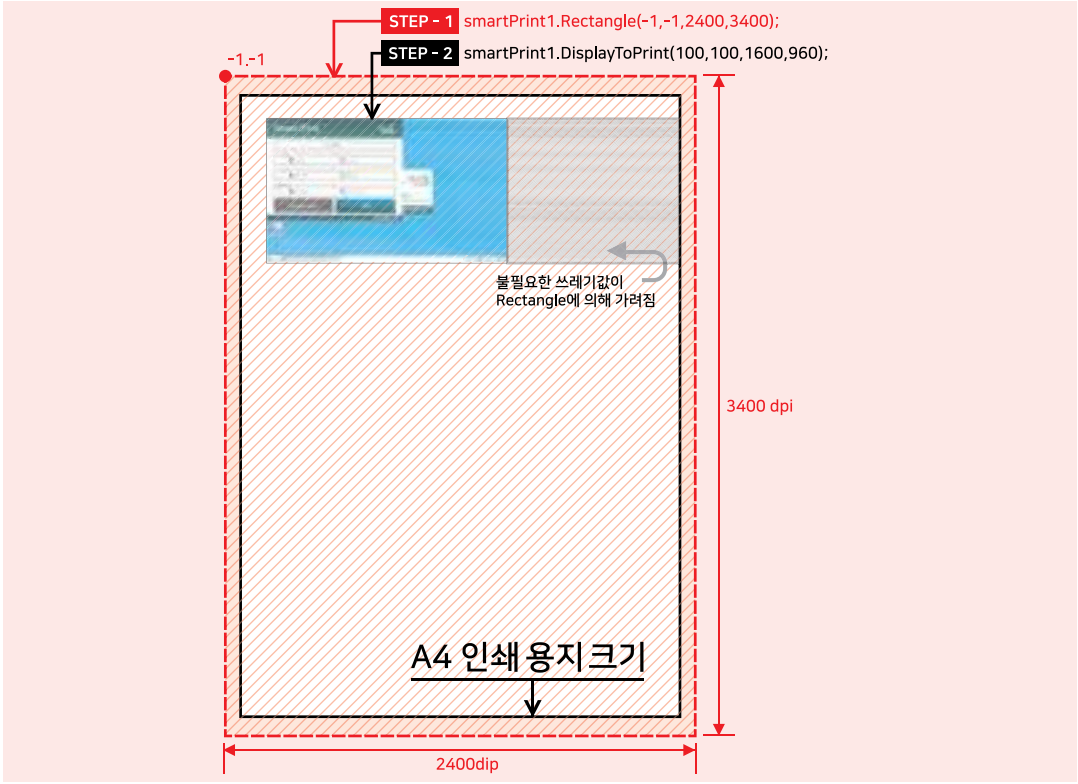
주의 DisplayToPrint() 메소드 사용 시 주의사항

[인쇄 영역] [불필요한 쓰레기값]

DisplayToPrint() 메소드를 사용하는 경우 프린터와의 호환성 문제로 인해 간혹 프린터 내부의 데이터(쓰레기값)이 프린트되는 경우가 있습니다. 이런 경우 Rectangle() 메소드를 호출하여 프린트 영역을 그린 후에 DisplayToPrint() 메소드를 호출하여 프린트하면 되며 설정 방법은 아래와 같습니다.

```
// Rectangle() 메소드를 사용하기 위해 SetPenStyle, SetBrushStyle을 설정합니다.
smartPrint1.SetPenStyle(Color.FromArgb(0, 1, 0), 1);
smartPrint1.SetBrushStyle(Color.White, false);
// iLeftX, iTopY는 -1, -1로 하면 Rectangle의 좌측과 우측 선이 표현되지 않습니다.
// A4 세로 방향 출력이라면 Width는 2400, Height는 3400으로 설정합니다.
// 설정 수치는 Hp OfficeJet Pro 8210 테스트 기준이며 다른 프린터의 경우 변경될 수 있습니다.
smartPrint1.Rectangle(-1, -1, 2400, 3400);
// 현재 화면 버퍼에 그리기 (iXpos : 100, iYpos : 100, Width : 1600, Height : 960)
smartPrint1.DisplayToPrint(100, 100, 1600, 960);
```

아래 그림은 인쇄 영역(A4) 위에 SmartPrint1.Rectangle()로 A4사이즈보다 크게 사각형을 그린 다음 IEC-Series의 현재 화면을 1600 X 960 사이즈로 인쇄한 모습입니다.



메소드(함수) **PrintOut**

프린터 버퍼에 출력된 데이터를 프린터로 전송하여 인쇄를 시작합니다. 1번 호출 시 1페이지 단위로 출력됩니다.

◉ `void PrintOut()`

C# 사용법

```
// 프린터 버퍼에 출력된 데이터를 인쇄합니다.
smartPrint1.PrintOut();
```

VB 사용법

```
smartPrint1.PrintOut()
```

메소드(함수) **SetPenStyle, SetBrushStyle**

도형의 선 및 내부 채움 색상을 설정합니다. 설정 이후 그려지는 모든 도형에 적용됩니다.

- `SetPenStyle()` : 도형의 선 색상 및 두께를 설정합니다.
- `SetBrushStyle()` : 도형의 내부 채움 색상을 설정합니다.

◉ `void SetPenStyle(Color penColor, int iWidth)`
◉ `void SetBrushStyle(Color brushColor)`
◉ `void SetBrushStyle(Color brushColor, bool bTransparent)`

[인자]

- `Color penColor` : 선의 색상
- `int iWidth` : 선의 두께
- `Color brushColor` : 도형 내부를 채울 색상

• bool bTransparent : 도형 내부 투명 처리 여부
  – true : 도형 내부를 투명하게 처리
  – false : 도형 내부를 SetBrushStyleEx()로 설정한 색으로 채움 (기본값)

C# 사용법

```
// 도형의 내부를 투명하게 채운다.
smartPrint1.SetBrushStyle(Color.White, true);
// 도형의 선 색상 및 두께 설정
smartPrint1.SetPenStyle(Color.Red, 3)
```

VB 사용법

```
smartPrint1.SetBrushStyle (Color.White, true)
smartPrint1.SetPenStyle(Color.Red, 3)
```

메소드(함수) SetFontCfg

TextOut() 메소드로 프린터 버퍼에 텍스트를 출력할 경우 폰트 관련 속성값을 설정합니다.

참조 다양한 폰트를 추가하는 방법은 "홈페이지 → 자료실 → Tech Note → 20. IEC-Series에서 다양한 폰트(Fonts 폴더, 트루타입)사용 방법 안내" 내용을 참조하시기 바랍니다.

◉ void SetFontCfg(int iFontSize, Color cFontColor)
◉ void SetFontCfg(string strFontName, int iFontSize, Color cFontColor)
◉ void SetFontCfg(string strFontName, int iFontSize, Color cFontColor, bool bBold, bool bItalic, bool bAntiAlased)

[인자]
• int iFontSize : 출력되는 폰트의 크기
• Color cFontColor : 출력되는 폰트의 색상
• bool bBold : 출력되는 폰트의 굵기
  – true : 굵게
  – false : 표준
• bool bItalic : 출력되는 폰트의 기울임
  – true : 기울임
  – false : 표준
• bool bAntiAlased : 출력되는 폰트를 배경과 글자의 안티에일리어싱(Anti-Aliasing) 효과를 설정
  – true : 안티 에일리어싱 사용
  – false : 안티 에일리어싱 사용 안함

사용법

참고 TextOut() 메소드의 사용법을 참고하시기 바랍니다.

메소드(함수) Erase

프린터 버퍼에 그려진 데이터를 모두 지웁니다. 화면이 지워지면 배경 색상은 흰색으로 됩니다.

◉ void Erase()

C# 사용법

```
// 프린터 버퍼에 그려진 모든 그림을 지운다.
smartPrint1.Erase();
```

VB 사용법

```
smartPrint1.Erase()
```

프로퍼티(속성) ImageDraw

Image

리소스로 포함된 이미지 또는 이미지 파일을 프린터 버퍼에 그린다. (지원 이미지 파일 형식 : BMP, JPG, GIF, PNG)

주의 이미지 파일 출력 시 이미지 파일의 용량 관련 주의사항

이미지 파일의 용량이 클 경우 IEC-Series의 프로그램 메모리 용량에 따라 이미지 출력이 되지 않을 수 있습니다. 따라서 이미지 사용 시 되도록이면 적은 용량의 이미지를 사용하여 프린터 버퍼를 절약하시기 바랍니다. 다만, IEC667/1000 Series 제품을 사용하시면 상대적으로 프린터 버퍼를 확장하여 사용할 수 있습니다.
또한, ImageDraw() 이외의 메소드는 프린터 버퍼를 적게 사용하는 벡터 출력 방식이므로 이미지 출력 이외의 기능은 메모리의 제한이 거의 없습니다.

◉ `void ImageDraw(Image img, int iXpos, int iYpos)`
◉ `void ImageDraw(string strFilePath, int iXpos, int iYpos)`
◉ `void ImageDraw(Image img, int iXpos, int iYpos, int iWidth, int iHeight)`
◉ `void ImageDraw(string strFilePath, int iXpos, int iYpos, int iWidth, int iHeight)`

[인자]

- `string strFilePath` : 이미지 파일의 경로와 파일명 (필수)
- `Image img` : 이미지 객체로 주로 이미지 파일이 리소스로 포함된 경우 사용된다.(필수)
- `int iXpos` : 출력될 이미지 좌측 상단의 X좌표 (필수)
- `int iYpos` : 출력될 이미지 좌측 상단의 Y좌표 (필수)
- `int iWidth` : 출력될 이미지의 수평 길이 (옵션 : Scale 조정 시 사용)
- `int iHeight` : 출력될 이미지의 수직 길이 (옵션 : Scale 조정 시 사용)

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
// " Flash Disk₩₩hns.bmp " 이미지 파일을 읽어온다.
Bitmap img = new Bitmap( " Flash Disk₩₩hns.bmp " );
//지정된 이미지 파일을 불러와 10, 350에 원본 이미지 크기로 프린터 버퍼에 출력합니다.
smartPrint1.ImageDraw(img, 10, 350);
//리소스에 포함된 이미지 파일을 10, 10에 원본 이미지 크기로 프린터 버퍼에 출력합니다.
smartPrint1.ImageDraw(Resource1.Logo, 10, 10);
//리소스에 포함된 이미지를 400, 10에 원본 이미지 크기의 2배로 확대하여 프린터 버퍼에 출력합니다.
smartPrint1.ImageDraw(Resource1.Logo, 400, 10, Resource1.Logo.Width*2, Resource1.Logo.Height*2);
```

VB 사용법

```
Dim img As Bitmap = New Bitmap( " Flash Disk₩₩hns.bmp " )
smartPrint1.ImageDraw(img, 10, 350)
smartPrint1.ImageDraw(Resource1.Logo, 10, 10)
smartPrint1.ImageDraw(Resource1.Logo, 400, 10, Resource1.Logo.Width*2, Resource1.Logo.Height*2)
```

메소드(함수) TextOut

텍스트를 프린터 버퍼에 출력합니다. 폰트 관련 속성 설정은 SetFontCfg로 설정합니다.

◉ `void TextOut(int iXpos, int iYpos, string strFormat, params object[] args)`

[인자]

- `int iXpos` : 출력되는 텍스트 좌측 상단의 X좌표
- `int iYpos` : 출력되는 텍스트 좌측 상단의 Y좌표
- `string strFormat` : 출력될 문자로 형식 문자열을 지원합니다.
- `params object[] args` : 형식 문자열에 대입하는 객체(데이터) 가변 인자

참고 형식 문자열에 관한 자세한 내용은 SmartDraw의 "형식 문자열에 대한 설명"을 참고하시기 바랍니다.

C# 사용법

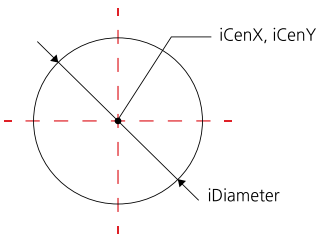
```
smartPrint1.SetFontCfg( " 굴림 " , 20, Color.LightSkyBlue, true, false, true);
smartPrint1.TextOut(10, 70, " HNS / SmartX-SmartPrint/IEC1000 / www.hnsts.co.kr " );
```

VB 사용법

```
smartPrint1.SetFontCfg( " 굴림 " , 20, Color.LightSkyBlue, True, False, True)
smartPrint1.TextOut(10, 70, " HNS / SmartX-SmartPrint/IEC1000 / www.hnsts.co.kr " )
```

메소드(함수) Circle

프린터 버퍼에 원을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며, 내부 채움은 SetBrushStyle() 메소드로 설정합니다.



◉ `void Circle(int iCenX, int iCenY, int iDiameter)`

[인자]

- `int iCenX` : 원 중심의 X좌표
- `int iCenY` : 원 중심의 Y좌표
- `int iDiameter` : 원의 지름

C# 사용법

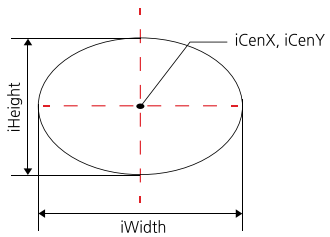
```
// 프린터 버퍼에 원 그리기
smartPrint1.Circle(300, 300, 50);
```

VB 사용법

```
smartPrint1.Circle(300, 300, 50)
```

메소드(함수) Ellipse

프린터 버퍼에 타원을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며 내부 채움은 SetBrushStyle() 메소드로 설정합니다.



◉ `void Ellipse(int iCenX, int iCenY, int iWidth, int iHeight)`

[인자]

- `int iCenX` : 원 중심의 X좌표
- `int iCenY` : 원 중심의 Y좌표

- int iWidth : 타원의 수직 길이
- int iHeight : 타원의 수평 길이

C# 사용법

```
// 프린터 버퍼에 타원 그리기
smartPrint1.Ellipse(300, 300, 150, 70);
```

VB 사용법

```
smartPrint1.Ellipse(300, 300, 150, 70)
```

메소드(함수) GradientFill

프린터 버퍼에 사각형 영역을 지정된 색으로 그라데이션 효과를 주어 그립니다.

◉ void GradientFill(int iLeftX, int iTopY, int iWidth, int iHeight, Color startColor, Color endColor, SmartPrint.GRADIENTFILLMODE dwMode)

[인자]

- int iLeftX : 사각형 좌측 상단의 X좌표
- int iTopY : 사각형 좌측 상단의 Y좌표
- int iWidth : 사각형의 수평 길이
- int iHeight : 사각형의 수직 길이
- Color startColor : 그라데이션의 시작 색상
- Color endColor : 그라데이션의 끝 색상
- SmartPrint.GRADIENTFILLMODE dwMode : 그라데이션 방향
  - SmartPrint.GRADIENTFILLMODE.GRADIENT_FILL_RECT_H : 그라데이션 방향을 수평으로 설정
  - SmartPrint.GRADIENTFILLMODE.GRADIENT_FILL_RECT_V : 그라데이션 방향을 수직으로 설정

[표] GRADIENTFILLMODE 인자값에 따른 표현 예시

GRADIENT_FILL_RECT_H	GRADIENT_FILL_RECT_V
 수평그라데이션	 수직그라데이션

C# 사용법

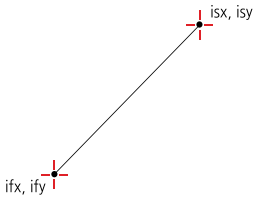
```
// 시작위치 (0,0), 폭 : 800, 높이 : 480, 시작 색상 : LightPink, 끝 색상 : SkyBlue, 수직그라데이션
smartPrint1.GradientFill(0, 0, 800, 480, Color.LightPink, Color.SkyBlue, SmartPrint.
                         GRADIENTFILLMODE.GRADIENT_FILL_RECT_V);
// 시작위치 (0,0), 폭 : 800, 높이 : 480, 시작 색상 : LightPink, 끝 색상 : SkyBlue, 수평그라데이션
smartPrint1.GradientFill(0, 0, 800, 480, Color.LightPink, Color.SkyBlue, SmartPrint.
                         GRADIENTFILLMODE.GRADIENT_FILL_RECT_H);
```

VB 사용법

```
smartPrint1.GradientFill(0, 0, 800, 480, Color.LightPink, Color.SkyBlue, SmartPrint.
                         GRADIENTFILLMODE.GRADIENT_FILL_RECT_V)
smartPrint1.GradientFill(0, 0, 800, 480, Color.LightPink, Color.SkyBlue, SmartPrint.
                         GRADIENTFILLMODE.GRADIENT_FILL_RECT_H)
```

메소드(함수) Line

프린터 버퍼에 선을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정합니다.



◉ void Line(int ifx, int ify, int isx, int isy)

[인자]

- int ifx : 처음 점의 X좌표
- int ify : 처음 점의 Y좌표
- int isx : 두번째 점의 X좌표
- int isy : 두번째 점의 Y좌표

C# 사용법

```
// 프린터 버퍼에 선 그리기
smartPrint1.Line(100, 100, 200, 200);
```

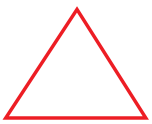
VB 사용법

```
smartPrint1.Line(100, 100, 200, 200)
```

메소드(함수) Polygon, Polyline

Polygon() 메소드와 Polyline() 메소드는 연속된 여러 개의 선을 프린터 버퍼에 그립니다. 선의 색상 및 두께는 SetPen Style() 메소드로 설정하며, Polygon() 메소드의 경우 SetBrushStyle() 메소드로 설정된 색상으로 안쪽이 채워집니다.

[표] Polygon() 메소드와 Polyline() 메소드의 차이점

메소드	Polygon	Polyline
차이점	1. 시작과 끝점이 연결됨 2. 연결된 선의 안쪽이 채워짐	1. 시작과 끝점이 연결되지 않음 2. 연결된 선의 안쪽이 채워지지 않음
출력 결과		

◉ void Polygon(SmartPrint.POINT[] pPoint)

◉ void Polyline(SmartPrint.POINT[] pPoint)

[인자]

- SmartPrint.POINT[] pPoint : 연속된 선을 그리기 위한 여러 점의 좌표 배열

C# 사용법

```
// Polygon으로 별을 그린다.
SmartPrint.POINT[] points = new SmartPrint.POINT[10];
int iSize = 5, iXOffset = 300, iYOffset = 50;
smartPrint1.SetBrushStyle(Color.Yellow);  // 별 안쪽의 채워질 색상을 지정합니다.
smartPrint1.SetPenStyle(Color.Red, 2);    // 별의 외곽 선의 색상 및 두께를 지정합니다.
// 별 모양의 좌표를 지정합니다.
points[0].X = (16 * iSize) + iXOffset; points[0].Y = (28 * iSize) + iYOffset;
points[1].X = (19 * iSize) + iXOffset; points[1].Y = (17 * iSize) + iYOffset;
points[2].X = (30 * iSize) + iXOffset; points[2].Y = (17 * iSize) + iYOffset;
points[3].X = (21 * iSize) + iXOffset; points[3].Y = (11 * iSize) + iYOffset;
points[4].X = (25 * iSize) + iXOffset; points[4].Y = (1 * iSize) + iYOffset;
points[5].X = (16 * iSize) + iXOffset; points[5].Y = (7 * iSize) + iYOffset;
```

```
points[6].X = (7 * iSize) + iXOffset; points[6].Y = (1 * iSize) + iYOffset;
points[7].X = (11 * iSize) + iXOffset; points[7].Y = (11 * iSize) + iYOffset;
points[8].X = (2 * iSize) + iXOffset; points[8].Y = (17 * iSize) + iYOffset;
points[9].X = (13 * iSize) + iXOffset; points[9].Y = (17 * iSize) + iYOffset;
smartPrint1.Polygon(points); // 설정된 데이터를 프린터 버퍼에 출력합니다.(별 모양 그린다.)

// Polyline으로 삼각형을 그린다. Polyline은 시작점과 끝점을 연결하지 않습니다.
SmartPrint.POINT[] points1 = new SmartPrint.POINT[4];
// 삼각형 모양의 좌표를 지정합니다.
points1[0].X = 400; points1[0].Y = 200; points1[1].X = 300; points1[1].Y = 300;
points1[2].X = 500; points1[2].Y = 300; points1[3].X = 400; points1[3].Y = 200;
// 설정된 데이터를 프린터 버퍼에 출력합니다.(삼각형 그림. 삼각형 내부는 투명함)
smartPrint1.Polyline(points1);
```

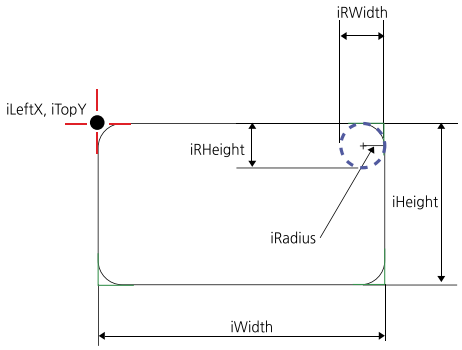
VB 사용법

```
Dim points As SmartPrint.POINT() = New SmartPrint.POINT(10) {}
Dim iSize, iXOffset, iYOffset As Integer
iSize = 5 : iXOffset = 300 : iYOffset = 50
smartPrint1.SetBrushStyle(Color.Yellow) : smartPrint1.SetPenStyle(Color.Red, 2)
points(0).X = (16 * iSize) + iXOffset : points(0).Y = (28 * iSize) + iYOffset
' …중략…
points(9).X = (13 * iSize) + iXOffset : points(9).Y = (17 * iSize) + iYOffset
smartPrint1.Polygon(points)
Dim points1 As SmartPrint.POINT() = New SmartPrint.POINT(4) {}
points1(0).X = 400 : points1(0).Y = 200 : points1(1).X = 300 : points1(1).Y = 300
points1(2).X = 500 : points1(2).Y = 300 : points1(3).X = 400 : points1(3).Y = 200
smartPrint1.Polyline(points1)
```

메소드(함수) Rectangle, RoundRect

내부가 채워진 직사각형을 프린터 버퍼에 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며 내부 채움은 SetBrushStyle() 메소드로 설정합니다.

- Rectangle() : 직사각형을 그립니다.
- RoundRect() : 모서리가 둥근 직사각형을 그립니다.



◉ void Rectangle(int iLeftX, int iTopY, int iWidth, int iHeight)
◉ void RoundRect(int iLeftX, int iTopY, int iWidth, int iHeight, int iRadius)
◉ void RoundRect(int iLeftX, int iTopY, int iWidth, int iHeight, int iRWidth, int iRHeight)

[인자]

- int iLeftX : 출력될 사각형 좌측 상단의 X좌표
- int iTopY : 출력될 사각형 좌측 상단의 Y좌표

- int iWidth : 출력될 사각형의 수평 길이
- int iHeight : 출력될 사각형의 수직 길이
- int iRadius : RoundRect에서 모서리의 라운드의 반지름
- int iRWidth : RoundRect에서 모서리의 라운드의 수평 길이
- int iRHeight : RoundRect에서 모서리의 라운드의 수직 길이

C# 사용법

```
// 프린터 버퍼에 내부가 채워진 사각형을 그립니다.
smartPrint1.Rectangle(10, 10, 300, 200);
// 프린터 버퍼에 내부가 채워진 모서리가 둥근 사각형을 그립니다.
smartPrint1.RoundRect(300, 200, 300, 200, 20);
```

VB 사용법

```
smartPrint1.Rectangle(10, 10, 300, 200)
smartPrint1.RoundRect(300, 200, 300, 200, 20)
```

메소드(함수) SetPixel

지정된 좌표에 색상을 지정하여 프린터 버퍼에 점을 그립니다.

◉ void SetPixel(int iXPos, int iYPos, Color crColor)

[인자]

- int iXPos : 점의 X좌표
- int iYPos : 점의 Y좌표
- Color crColor : 그려질 점의 색상

C# 사용법

```
smartPrint1.SetPixel(100, 200, Color.LightCyan);
```

VB 사용법

```
smartPrint1.SetPixel(100, 200, Color.LightCyan)
```

4) SmartPrint 예제 사용하기

SmartPrint를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

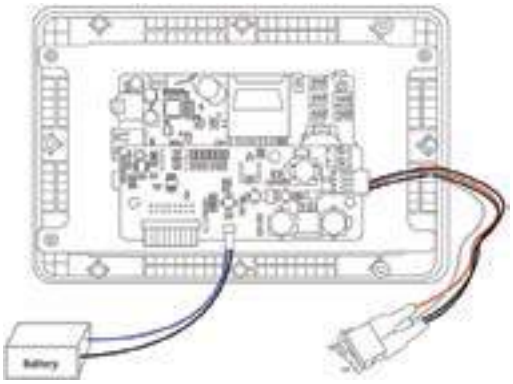
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartPrint"

15. SmartBattery

SmartBattery는 IEC Lite-Series 제품에서 외부 전원으로 리튬이온(Li-ion) 충전지를 장착할 수 있는 별도의 옵션제품인 SmartBattery의 사용을 편리하게 할 수 있도록 만들어진 컴포넌트입니다.

- 외부 전원 연결 시 충전 기능
- 충전 관련 보호 회로 내장
- 충전 상태 및 남은 용량 확인 기능
- 외부 전원 상태 확인 기능
- 충전 속도 선택 Mode 지원(Fast/Normal)
- 시스템 전원 OFF 기능 지원(Power Shutdown)
- 외부의 전원을 사용하지 않는 포터블(Portable) 시스템 구성 용도
- 외부 전원의 공급이 차단될 경우 시스템의 데이터 및 상태를 Backup하기 위한 용도



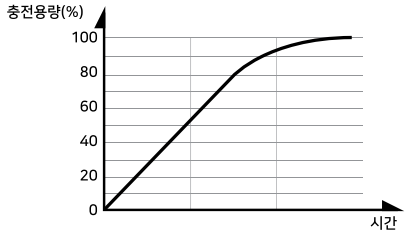
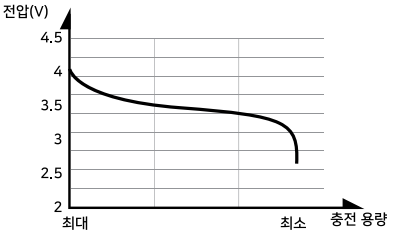
주의 1. 본 기능은 非 Lite-Series에서는 지원되지 않습니다.
2. SmartBattery Board에 장착되는 리튬이온 충전지는 별도입니다.

참조 장착 방법 및 하드웨어 관련 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → SmartBattery"를 참조하시기 바랍니다.

1) 전기적 지원 사양

SmartBattery 사용 시 반드시 별매의 SmartBattery Board와 리튬이온 충전지를 사용해야 하며, 단위 시간당 충전 용량은 최대 1000mAh이며 평균 500mAh입니다. 아래에서 Battery 관련 주의사항과 표를 확인하시기 바랍니다.

[표1] 리튬이온 전지 충방전 특성 곡선(1 CELL 기준)

충전 특성 곡선	방전 특성 곡선
	

※ 충전 시 70%이상의 충전 구간은 배터리의 제조사에 따라 다소 차이가 있습니다.

주의 Battery 관련 주의사항

1. 반드시 리튬이온 충전지를 사용하시기 바랍니다.
2. 검증된 제조 회사의 배터리를 사용하시기 바랍니다. – 화재 및 제품의 파손 등이 발생할 수 있습니다.
3. 전압 7.4V 즉 2CELL을 사용하시기 바라며 500mAh 이상의 충전지를 사용하시기 바랍니다.

[표1] IEC Lite-Series 제품별 소비 전류

IEC Lite-Series	인치	제품명	전압	전류		
				ON	ON(Peak)	* OFF(Light)
IEC266Lite-Series	4.3inch	IEC266Lite-43[B1]/[B2]	5V	430mA	510mA	320mA
	5.6inch	IEC266Lite-56[B1]/[B2]		800mA	900mA	470mA
	7inch	IEC266Lite-07[B1]/[B2]		1000mA	1080mA	520mA
IEC667Lite-Series	5.6inch	IEC667Lite-56[B1]/[B2]	5V	700mA	830mA	360mA
	7inch	IEC667Lite-07[B1]/[B2]		880mA	1000mA	390mA
	8inch	IEC667Lite-08[B1]/[B2]		710mA	840mA	310mA
	10.2inch	IEC667Lite-102[B1]/[B2]		930mA	1020mA	400mA
	10.4inch	IEC667Lite-104[B1]/[B2]		980mA	1080mA	360mA
IEC1000Lite-Series	4.3inch	IEC1000Lite-43[B1]/[B2]	5V	420mA	580mA	310mA
	5.6inch	IEC1000Lite-56[B1]/[B2]		780mA	950mA	440mA
	6.95inch	IEC1000Lite-07N[B1]/[B2]		620mA	770mA	400mA
	7inch	IEC1000Lite-07[B1]/[B2]		980mA	1160mA	450mA
	8inch	IEC1000Lite-08[B1]/[B2]		640mA	800mA	420mA
	10.2inch	IEC1000Lite-102[B1]/[B2]		1050mA	1240mA	500mA
	10.4inch	IEC1000Lite-104[B1]/[B2]		1100mA	1320mA	420mA

* LCD Back Light OFF 상태의 측정값

2) 프로그래밍 적용 가이드

STEP-1 배터리 전압 관련 설정하기

MaxVoltage 속성으로 완충 전압을, MinVoltage 속성으로 IEC-Series가 동작 가능한 최소 전압을 설정합니다.
또한 OffsetVoltage 속성에 배터리에 인가되는 외부 DC 전압과 DC 전원이 해제되었을 때 배터리의 전압의 차이를 설정합니다.

※ 자세한 내용은 "MaxVoltage, MinVoltage, OffsetVoltage 속성"을 참고하시기 바랍니다.

```
private void SetVoltage(double dMax, double dMin, double dOffset)
{
  smartBattery1.MaxVoltage = dMax;  // IEC-Series가 동작 가능한 최소 전압을 설정
  smartBattery1.MinVoltage = dMin;  // 배터리의 최대 충전 전압을 설정
  // 배터리에 인가되는 외부 DC 전압과, 전원이 해제될 경우 배터리의 전압의 차이를 설정
  smartBattery1.OffsetVoltage = dOffset;
}
```

STEP-2 배터리 충전 여부 및 충전량 확인하기

IsExtDCPowerIN() 메소드로 현재 외부 전원이 인가되어 배터리가 충전 중인지 여부를 확인할 수 있습니다. 또한, GetBatteryPercent() 메소드로 현재 배터리의 충전량을 퍼센트로 얻을 수 있으며, GetBatteryVoltage() 메소드로 현재 배터리 충전량을 값(Volt)으로 얻을 수 있습니다.

※ 자세한 내용은 "GetBatteryPercent(), GetBatteryVoltage(), IsExtDCPowerIN() 메소드"를 참고하시기 바랍니다.

```
if (smartBattery1.IsExtDCPowerIN() == true) { /* 현재 외부 전원이 인가되어 배터리 충전 중 */ }
else { /* 현재 외부 전원이 인가되지 않아 배터리 충전 중이 아니며, 처리 코드 작성 */ }
```

```
string strPercent = smartBattery1.GetBatteryPercent().ToString() + "%";
string strVoltage = smartBattery1.GetBatteryVoltage().ToString("0.00") + "V";
```

3) SmartBattery 인터페이스 설명

SmartBattery Component Interface		
속성		
MaxVoltage : double	MinVoltage: double	OffsetVoltage: double
메소드		
GetBatteryPercent() : int	GetBatteryVoltage() : double	IsExtDCPowerIN() : bool
PowerOFF() : void		

프로퍼티(속성) MinVoltage, MaxVoltage

Battery의 완충 전압과 최소 전압을 설정합니다.

- `MinVoltage` : IEC-Series의 동작 가능한 Battery의 최소 전압을 설정합니다. (단위 : Volt, 기본값 : 6.5)
- `MaxVoltage` : Battery의 완충 전압을 설정합니다. (단위 : Volt, 기본값 : 8.2)

주의 SmartBattery Board는 Battery의 전압이 8.2V가 되면 완충으로 인식하도록 설계되어 있습니다. 따라서 MaxVoltage 속성값을 변경하지 마시고 기본값(8.2)으로 사용하시기 바랍니다.

- `double` : Battery의 최소 및 최대 전압 (단위 : Volt)

C# 사용법

```
smartBattery1.MaxVoltage = 8.2;  // 완충 전압 설정 (단위 : Volt)
smartBattery1.MinVoltage = 6.3;  // 최소 전압 설정 (단위 : Volt)
```

C# 사용법

```
smartBattery1.MaxVoltage = 8.2
smartBattery1.MinVoltage = 6.3
```

프로퍼티(속성) OffsetVoltage

Battery를 충전하기 위해 외부 DC 전원을 공급하게 되면 인가되는 전압과 현재 Battery의 전압의 차이(Offset 전압)가 발생하게 되며, 이 전압의 차이를 설정합니다. Offset 전압은 Battery의 용량 및 제조 업체에 따라 달라지며 정확한 사용을 위해서는 다음과 같은 방법으로 설정하시기 바랍니다.

중요 Offset 전압 측정 방법

1. 우선 외부 DC 전원을 연결하지 않은 상태에서 테스터기로 Battery의 양극 전압을 측정합니다.
2. 다음은 외부 DC 전원을 연결한 상태에서 테스터기로 Battery의 양극 전압을 측정합니다.
3. 측정된 두 값의 차이가 Offset 전압값입니다. 본 측정을 보다 정확하게 하기 위해서는 위 측정을 배터리의 전압량의 20%대/50%대/80%대 총 3개의 값을 평균하여 적용하시면 오차를 최소화하실 수 있습니다.

권장 위 측정 및 테스트는 SmartBattery 예제 프로그램을 활용하시길 권장합니다.

- `double` : Offset 전압 (단위 : Volt)

C# 사용법

```
// Offset 전압을 0.23V로 설정
smartBattery1.OffsetVoltage = 0.23;
```

VB 사용법

```
smartBattery1.OffsetVoltage = 0.23
```

메소드(함수) GetBatteryPercent

현재 Battery의 남은 용량의 비율을 얻습니다.

◉ int GetBatteryPercent()

[리턴값]

• int : 현재 Battery의 남은 용량 (단위 : %)

C# 사용법

```
smartProgressBar1.Value = smartBattery1.GetBatteryPercent();  // 현재 Battery의 남은 용량을 표시
```

VB 사용법

```
smartProgressBar1.Value = smartBattery1.GetBatteryPercent()
```

메소드(함수) GetBatteryVoltage

현재 Battery의 전압값을 얻습니다.

◉ double GetBatteryVoltage()

[리턴값]

• double : 현재 Battery의 전압값 (단위 : Volt)

C# 사용법

```
// 현재 Battery 전압 표시
labVoltage.Text = smartBattery1.GetBatteryVoltage().ToString( " 0.000V " );
```

VB 사용법

```
labVoltage.Text = smartBattery1.GetBatteryVoltage().ToString( " 0.000V " )
```

메소드(함수) IsExtDCPowerIN

외부 DC 전원의 공급 상태를 확인합니다. 즉, Battery의 충전을 위하여 외부 DC 전원이 공급되는지 여부를 확인 할 수 있습니다.

◉ bool IsExtDCPowerIN()

[리턴값]

• bool : 외부 DC 전원 공급 여부
  - true : 외부 DC 전원이 공급됨
  - false : 외부 DC 전원이 공급되지 않음 (내부 Battery로 전원 공급됨)

C# 사용법

```
// 현재 외부 DC 전원이 공급되는지 확인
if (smartBattery1.IsExtDCPowerIN() == true)
{
  labPowerStatus.Text = " Battery ";  // 현재 Battery로만 동작 중
}
// 현재 Battery가 충전 중이며 외부 DC 전원이 공급되는 상태
else
{
  labPowerStatus.Text = " External DC Power ";
}
```

VB 사용법

```
If smartBattery1.IsExtDCPowerIN() = True Then
  labPowerStatus.Text = " Battery "
Else
  labPowerStatus.Text = " External DC Power "
End If
```

프로퍼티(속성) PowerOFF

시스템 전원을 OFF(끄기) 합니다. 메소드 호출 후 2초 후에 전원이 OFF 됩니다.

주의
1. 시스템의 전원을 OFF 하기 앞서 중요한 데이터는 BackUp 하시기 바라며, 만약 파일을 쓰고 있다면 파일을 닫아 데이터의 손상을 방지하시기 바랍니다.
2. SmartBattery Board의 J7 점퍼가 OFF 되어 있어야 전원이 OFF 됩니다.

◉ void PowerOFF()

C# 사용법

```
// 시스템을 종료(끄기)합니다.
smartBattery1.PowerOFF();
```

VB 사용법

```
smartBattery1.PowerOFF()
```

4) SmartBattery 예제 사용하기

SmartBattery를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartBattery"

Part-Ⅷ

사용자 인터페이스 컴포넌트

사용자 인터페이스(User Interface) 컴포넌트는 사용자 인터페이스에 적용될 디자이너에 의해 만들어진 요소별 이미지를 간편하게 설정 및 적용만으로도 완성도 높은 사용자 인터페이스를 쉽고 편리하게 개발할 수 있습니다.

사용자 인터페이스 컴포넌트에 이미지를 적용하실 경우 프로그램의 안정적인 동작을 위해
“Part –Ⅳ. 개발 시 유의사항의 1. 이미지 관련 주의사항”을 필독하여 반드시 준수하시기 바랍니다.

Part-Ⅷ. 사용자 인터페이스 컴포넌트

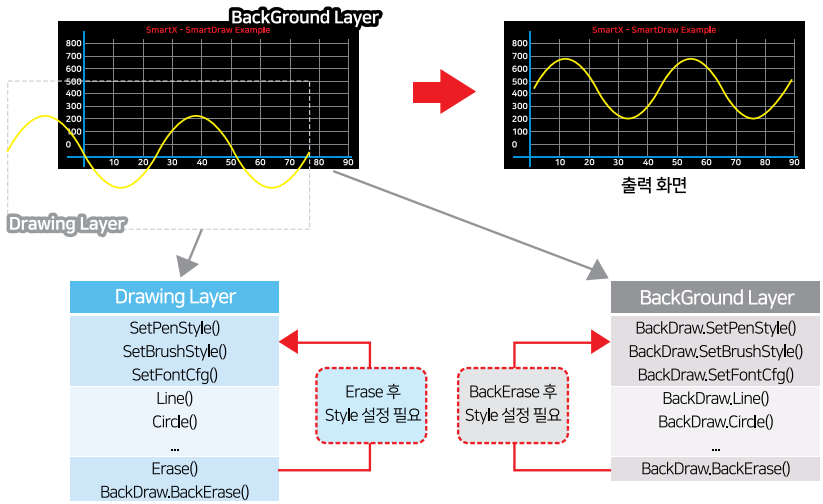
1. SmartDraw

Microsoft 사의 운영체제인 Windows 계열(Windows CE, Windows 10 등)의 그래픽 프로그램 구현 방식은 동일합니다. Windows 계열의 환경에서 그래픽 요소를 직접 그리는 프로그램은 생각보다 구현하기 쉽지 않습니다. Windows가 그래픽을 표현하는 메커니즘(OnPaint 이벤트, GDI, DC, 그래픽 객체 등)을 알고 있어야 하며 이러한 이유로 개발에 앞서 그래픽 처리 관련 서적이나 자료를 참고해야 하는 번거로움이 있습니다. 따라서 처음 그래픽 프로그램을 접하는 개발자는 많은 시간을 Windows의 그래픽 처리 방식을 이해하는데 소비하게 됩니다.

SmartDraw는 이런 불편함을 최소화하기 위해서 개발된 컴포넌트입니다. SmartDraw를 사용하시면 기존의 복잡했던 그래픽 처리를 간소화할 수 있으며, 여러 가지의 편리한 기능이 확장되어 있습니다. 기존의 그래픽 처리 구현에 불편함을 가지고 있었다면, SmartDraw를 프로젝트에 적용해 보시기 바랍니다.

- Windows CE의 그래픽 처리 방식을 이해하지 않아도 편리하게 구현 가능
- 직관적인 사용자 인터페이스
- WM_PAINT 이벤트에 대해 독립적(Form1_Paint, OnPaint 이벤트를 처리할 필요가 없음)
- 더블 버퍼링을 구현할 필요 없음
- 화면 갱신 시 깜박임을 최소화 처리
- 그래픽 처리 성능이 Native 수준에서 처리(성능 향상)
- 기본적인 Chart-Draw 기능 지원
- 애니메이션(Animation) GIF 출력(Decoding) 기능
- 애니메이션 및 기타 그래픽 컨트롤 제작에 편리
- 투명 배경 이미지 출력 기능
- 점, 선, 원, 타원, 둥근 사각형, 사각형, 그라데이션, 다각형, 이미지, 슬라이드 쇼 등의 기능을 쉽게 사용할 수 있도록 지원
- 화면에 그려진 그림을 이미지 파일 형식(BMP, JPG, GIF, PNG)으로 저장 기능 지원

1) BackGround Layer와 Drawing Layer 설명



SmartDraw는 BackGround Layer와 Drawing Layer 이렇게 2개의 Layer를 지원하고 있습니다.
BackGround Layer는 배경 영역으로 화면이 갱신되거나, Erase() 메소드를 호출하여도 화면이 지워지지 않으며 BackEr

ase() 메소드를 호출해야만 지워지는 영역입니다. 따라서 BackGround Layer에는 고정적으로 표현할 그림 요소(도형, 이미지, 텍스트)를 표현합니다.
Drawing Layer는 표현(Drawing) 영역으로 화면이 갱신되거나 Erase() 메소드를 호출하면 화면이 모두 지워지는 영역입니다. 따라서 Drawing Layer에는 자주 갱신되는 그림 요소를 표현합니다.
또한 각 Layer에 그림 요소를 표현하기 전, Erase() 또는 BackErase() 메소드를 호출하여 화면을 초기화하는 경우 반드시 표현에 사용되는 스타일(선, 채움 폰트)을 설정해야 합니다. 만약 스타일을 설정하지 않는다면 올바르게 지워지지 않거나, Draw되지 않을 수 있습니다. 설정한 스타일은 재설정하거나 Erase() / BackErase() 메소드 호출 전까지 설정한 스타일을 유지합니다.

[표] Drawing Layer와 BackGround Layer의 비교

Layer	Drawing Layer	BackGround Layer
호출 위치	SmartDraw	SmartDraw.BackDraw
초기화 방법	Erase() / BackErase()	BackErase()
스타일 설정	SetPenStyle() / SetBrushStyle() / SetFontCfg()	

참고 "BackDraw 속성"과 "Erase(), BackErase(), SetPenStyle(), SetBrushStyle(), SetFontCfg() 메소드" 내용을 참고하시기 바랍니다.

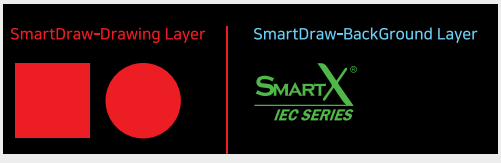
2) 프로그래밍 적용 가이드

CASE-1 그림 요소(도형, 선, 텍스트, 이미지)를 화면에 출력하기

SmartDraw의 메소드를 사용하여 화면에 간단하게 그림 요소(도형, 선, 텍스트, 이미지)를 화면에 출력할 수 있습니다. 그림 요소를 출력할 수 있는 Layer는 BackGround Layer와 Drawing Layer 2가지가 있으며 그림 요소 출력 전 반드시 출력할 그림 요소에 맞는 스타일 설정을 해야 합니다. 만약 스타일을 설정하지 않을 경우 그림 요소가 올바르게 출력되지 않을 수 있습니다.

※ "BackGround Layer와 Drawing Layer 설명"을 참고하시기 바랍니다.

```
// Drawing Layer에 그려질 그림 요소들의 스타일을 설정합니다.
smartDraw1.SetPenStyle(Color.Red, 1);
smartDraw1.SetBrushStyle(Color.Red);
smartDraw1.SetFontCfg("바탕", 20, Color.Red, true, false, true);

// Drawing Layer에 도형 출력하기
smartDraw1.Circle(260, 200, 140);
smartDraw1.Line(350, 20, 350, 300);
smartDraw1.Rectangle(25, 130, 140, 140);
smartDraw1.TextOut(30, 30, "SmartDraw-Drawing Layer");

// BackGround Layer에 그려질 그림 요소들의 스타일을 설정합니다.
smartDraw1.BackDraw.SetFontCfg("바탕", 20, Color.LightSkyBlue, true, false, true);
// BackGround Layer에 텍스트 출력하기
smartDraw1.BackDraw.TextOut(380, 30, "SmartDraw-BackGround Layer");
// BackGround Layer에 리소스에 포함된 이미지 파일 출력하기
smartDraw1.BackDraw.ImageDraw(Resource1.SmartX_Logo, 375, 130);
```



[출력 화면]

CASE-2 Chart 그리기

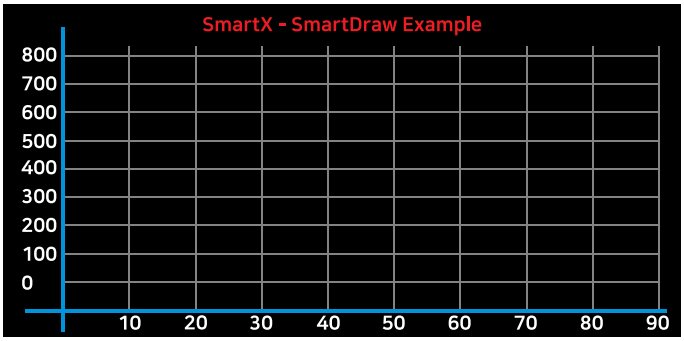
SmartDraw는 시간 진행에 따른 Chart 데이터 변화를 표현하는 Chart를 출력할 수 있습니다.
Chart를 그리기 위해서는 아래 순서대로 진행해야 합니다.

1. Chart의 배경 그리기

Chart가 그려질 때는 화면이 주기적으로 갱신되므로 Chart의 배경은 BackGround Layer에 이미지 또는 직접 도형을 그려 출력시켜야 합니다. 본 가이드에서는 이미지를 사용했으며, 아래 표를 확인하여 적합한 방법을 사용하시기 바랍니다.

[표1] 배경 그리기 방식에 따른 장단점

배경	이미지	도형
장점	도형에 비해 적용이 간편함	이미지에 비해 적용이 어려움
단점	Chart의 배경이 동적으로 변경되는 경우 수정이 어려움	Chart의 배경이 동적으로 변경되는 경우 수정이 용이함



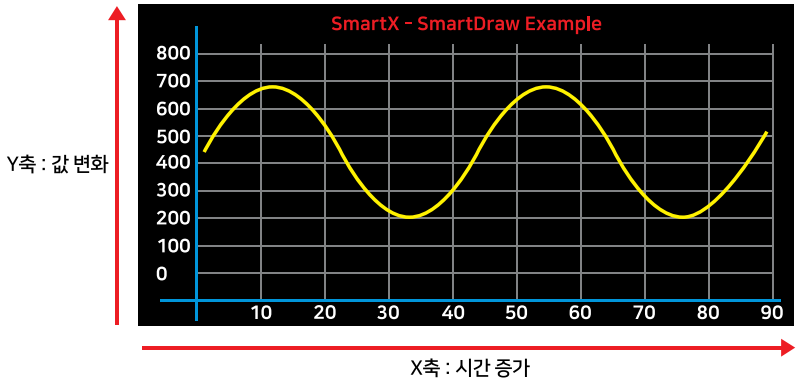
[가이드에서 사용된 배경 이미지]

2. Chart 스타일 설정 및 Chart 그리기

SetChartCfg() 메소드와 ChartChannelPenstyle, ChartDrawStep 속성값을 설정하여 기본적인 Chart 스타일 설정 후 PutData() 메소드로 Chart 데이터를 입력하면 Chart가 그려집니다.

[표2] Chart 그리기 관련 속성 및 메서드와 설명

관련 속성 및 메소드	설명
SetChartCfg()	Chart를 그리기 위한 Chart 관련 파라미터를 설정
ChartChannelPenStyle	Chart에 표현되는 각 채널의 색상 및 두께를 설정
ChartDrawStep	Chart 갱신을 위한 PutData() 메소드의 호출 횟수를 설정
PutData()	Chart를 그리기 위한 Chart 데이터를 입력



[출력 결과]

※ “PutData(), PutDataAllClear(), ChartNowDraw(), SetChartCfg() 메소드”와 “ChartDrawStep, ChartChannelPen Style 속성” 내용을 참고하시기 바랍니다.

```
// BackGround Layer에 Chart 배경 이미지를 출력
smartDraw1.BackDraw.ImageDraw(Resource1.Back,0,0);

// Chart의 원점 : (62, 320), 크기 : 630 X 270, 채널 수 : 1개, 왼쪽 스크롤
smartDraw1.SetChartCfg(62, 320, 630, 270, 3, SmartX.SmartDraw.CHARTREFRESH.LEFTSCROLL, 1);

// Chart의 각 채널의 해당되는 선의 속성을 설정하기 위한 구조체 선언 및 2개 채널에 해당되는
// 각각의 선의 속성 구조체 배열 생성
SmartX.SmartDraw.CHARTPENSTYLE[] chartchPenStyle = new SmartX.SmartDraw.CHARTPENSTYLE[1];

// 채널의 선 색상은 노란색으로 하고 선의 두께는 2로 설정
chartchPenStyle[0].m_chColor = Color.Yellow;
chartchPenStyle[0].m_iPenWidth = 2;

// 채널별 설정된 선의 속성 정보를 적용합니다.
smartDraw1.ChartChannelPenStyle = chartchPenStyle;

// PutData() 함수가 2번 호출될 때마다 화면에 추가된 Chart 데이터를 반영합니다.
smartDraw1.ChartDrawStep = 2;

// Sine 파형을 그리기 위한 변수
private int m_iAg = 0;

// 타이머를 사용해 주기적으로 Chart 데이터를 입력
private void smartTimer1_Tick(object sender, EventArgs e)
{
  // Sine 파형을 그린다.
  int iVal = 150 + (int)(77 * Math.Sin(2 * Math.PI * (m_iAg) / 100));

  // Chart 데이터를 입력합니다.
  smartDraw1.PutData(iVal);
  m_iAg++;
}
```

참조 "홈페이지 → 자료실 → Tech Note → 25. [C#, VB.NET] SmartDraw로 차트의 X축 Scroll(과거 데이터 보기) 기능, X축 커서 기능 구현 예제"를 참조하시기 바랍니다.

3) SmartDraw 인터페이스 설명

SmartDraw Component Interface		
속성		
AnimationGIFFrameInterval : int	BackDraw : SmartDraw.BackImgDraw	BackPitureBox : PictureBox
BackPictureBox1 : SmartInnerForm	ChartChannelPenStyle : SmartDraw.CHARTPENSTYLE[]	ChartDrawStep : int
ImageListIndexIncType: SmartDraw.IMAGELISTINDEXCOUNT	InitVisible : bool	SetBackimage : Bitmap
SetBackImageAutoSize : bool		
메소드		
ActiveViewToBackImage() : void	AddImageList(Image img) : void (+1개 오버로드)	AnimationGIF(string strGifPathName, int iXPos, int iYPos) : void
AnimationGIFPause() : void	AnimationGIFPlay() : void	AnimationGIFStop() : void
BackErase(Color bColor) : void	ChartNowDraw() : void	Circle() : void
Ellipse() : void	Erase() : void	GradientFill(int iLeftX, int iTopY, int iWidth, int iHeight, Color startColor, Color endColor, SmartDraw.GRADIENTFILLMODE dwMode) : void

ImageDraw(Image img, int iXpos, int iYpos) : void (+3개 오버로드)	ImageListClear() : void	ImageListDraw(int iXPos, int iYPos, double dPercentage) : void (+1개 오버로드)
ImageListDrawNext(int iXPos, int iYPos) : void (+1개 오버로드)	Line(int ifx, int ify, int isx, int isy) : void	Polygon(SmartDraw.POINT[] pPoints) : void
PolyLine(SmartDraw.POINT[] pPoints) : void	PutData(params int[] ChDatas) : void	PutDataAllClear() : void
Rectangle(int iLeftX, int iTopY, int iWidth, int iHeight) : void	RotationImage(Bitmap rotateImage, Bitmap backResultImage, int iCenterX, int iCenterY, int iOffsetX, int iOffsetY, double degrees) : void	RotationPoints(SmartDraw.POINT[] inPoints, SmartDraw.POINT centerPoint, double fAngle) : SmartDraw.POINT[]
RoundRect(int iLeftX, int iTopY, int iWidth, int iHeight, int iRadius) : void (+1개 오버로드)	Save(string strFullPathName) : bool	SetBrushStyle() : void
SetChartCfg(int iSetOrgX, int iSetOrgY, int iWidth, int iHeight, int iXIncStep, SmartDraw.CHARTREFRESH ChartRefreshMode, int iDataChannelNum) : void	SetFontCfg(int iFontSize, Color cFontColor) : void (+2개 오버로드)	SetPenStyle(Color penColor, int iWidth) : void (+1개 오버로드)
SetPixel(int iXPos, int iYPos, Color pColor) : void	SlideShowStart(int iInterval, int iXPos, int iYPos) : void	SlideShowStop() : void
TextOut(int iXpos, int iYpos, string strFormat, params object[] args) : void	TransparentImageDraw(Image img, int iXpos, int iYpos, Color cTransparentColor) : void (+3개 오버로드)	

프로퍼티(속성) AnimationGIFFrameInterval

Animation GIF 파일의 화면 출력(Play) 시 각 Frame 이미지의 Interval 시간을 설정합니다.
0인 경우 GIF 파일에서 설정된 값으로 적용합니다.

- int : GIF Frame Interval (단위 : ms)

C# 사용법

```
// 애니메이션 GIF의 Frame 전환 시간을 2초로 설정합니다.
smartDraw1.AnimationGIFFrameInterval = 2000;
```

VB 사용법

```
smartDraw1.AnimationGIFFrameInterval = 2000
```

프로퍼티(속성) BackDraw

BackDraw는 SmartDraw에서 BackGround Layer에 그리기 명령을 수행하기 위한 객체입니다.

중요 BackDraw 속성을 사용하여 BackGround Layer의 스타일 설정 및 그림 요소 그리기 작업을 할 수 있습니다.

참고 "BackGround Layer와 Drawing Layer 설명" 내용을 참고하시기 바랍니다.

C# 사용법

```
// BackGround Layer에 그릴 스타일을 설정합니다.
smartDraw1.BackDraw.SetBrushStyle(Color.LightGreen);
smartDraw1.BackDraw.SetPenStyle(Color.LightSkyBlue, 3);
smartDraw1.BackDraw.SetFontCfg(15, Color.LightSkyBlue);
smartDraw1.BackDraw.Line(100, 100, 200, 200);  // BackGround Layer에 선 그리기
smartDraw1.BackDraw.Circle(300, 300, 50);  // BackGround Layer에 원 그리기
// BackGround Layer에 문자 출력하기
smartDraw1.BackDraw.TextOut(300, 20, "SmartX-SmartDraw BackDraw Test!!!");
```

VB 사용법

```
smartDraw1.BackDraw.SetBrushStyle(Color.LightGreen)
smartDraw1.BackDraw.SetPenStyle(Color.LightSkyBlue, 3)
```

```
smartDraw1.BackDraw.SetFontCfg(15, Color.LightSkyBlue)
smartDraw1.BackDraw.Line(100, 100, 200, 200)
smartDraw1.BackDraw.Circle(300, 300, 50)
smartDraw1.BackDraw.TextOut(300, 20, " SmartX-SmartDraw BackDraw Test!!! " )
```

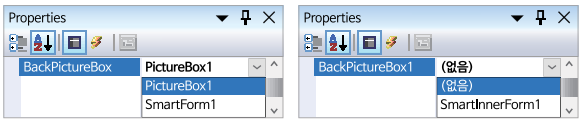
프로퍼티(속성) BackPictureBox, BackPictureBox1

SmartDraw의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartDraw에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법



프로퍼티(속성) ChartChannelPenStyle

SetChartCfg() 메소드에서 iDataChannelNum 인자값에 따라 Chart에 표현되는 여러 개의 선(각 채널의 Chart 데이터)의 색상 및 두께를 채널별로 설정합니다.

- SmartDraw.CHARTPENSTYLE[] : 각 Chart 채널의 색상 및 두께의 정보(사용 채널의 수만큼 생성해야 함)
  - Color m_chColor : 선의 색상
  - int m_iPenWidth : 선의 두께

C# 사용법

```
// Chart의 각 채널의 해당되는 선의 속성을 설정하기 위한 구조체 선언 및 2개 채널에 해당되는
// 각각의 선의 속성 구조체 배열 생성
SmartDraw.CHARTPENSTYLE[] chartchPenStyle = new SmartDraw.CHARTPENSTYLE[2];
// 1 채널의 선 색상은 빨간색, 선의 두께 2로 설정
chartchPenStyle[0].m_chColor = Color.Red; chartchPenStyle[0].m_iPenWidth = 2;
// 2 채널의 선 색상은 노란색, 선의 두께 1로 설정
chartchPenStyle[1].m_chColor = Color.Yellow;  chartchPenStyle[1].m_iPenWidth = 1;
smartDraw1.ChartChannelPenStyle = chartchPenStyle;
// 채널별 설정된 선의 속성 정보를 적용합니다.
```

VB 사용법

```
Dim chartchPenStyle() As SmartDraw.CHARTPENSTYLE = New SmartDraw.CHARTPENSTYLE(1) {}
chartchPenStyle(0).m_chColor = Color.Red  :  chartchPenStyle(0).m_iPenWidth = 2
chartchPenStyle(1).m_chColor = Color.Yellow  :  chartchPenStyle(1).m_iPenWidth = 1
smartDraw1.ChartChannelPenStyle = chartchPenStyle
```

프로퍼티(속성) ChartDrawStep

Chart 데이터 표현 시 화면(Chart) 갱신을 위한 PutData() 메소드 호출 빈도를 설정합니다. 즉, ChartDrawStep 속성 값만큼 PutData() 메소드 호출 시 화면이 갱신됩니다.

참고 Chart 데이터의 빈번한 추가로 화면 갱신이 여러 번 발생하여 성능에 문제가 생기는 경우 ChartDrawStep 속성값을 조절하여 문제를 해결할 수 있습니다.

• int : Chart를 그리는 빈도 (기본값 : 1)

C# 사용법

```
// PutData() 함수가 2번 호출될 때마다 화면에 추가된 Chart 데이터를 반영합니다.
smartDraw1.ChartDrawStep = 2;
smartDraw1.PutData(33);  smartDraw1.PutData(77);   // 화면에 Chart 데이터를 적용
smartDraw1.PutData(100);  smartDraw1.PutData(55);  // 화면에 Chart 데이터를 적용
```

VB 사용법

```
smartDraw1.ChartDrawStep = 2
smartDraw1.PutData(33) : smartDraw1.PutData(77)
smartDraw1.PutData(100) : smartDraw1.PutData(55)
```

프로퍼티(속성) ImageListIndexIncType

SlideShowStart(), ImageListDrawNext() 메소드에서 출력되는 이미지 Index의 카운팅 방식을 설정합니다.

• SmartDraw.IMAGELISTINDEXCOUNT.INCDECREMENT : 증가 후 감소 [출력 순서 : 0,1,2,3,2,1,0,1,2,3,...]
• SmartDraw.IMAGELISTINDEXCOUNT.RINGCOUNT : 증가 후 다시 시작 [출력 순서 : 0,1,2,3,0,1,2,3,0,1,2,3,...]
• SmartDraw.IMAGELISTINDEXCOUNT.INCREMENT : 증가 후 유지 [출력 순서 : 0,1,2,3,3,3,3,3,3,3,...]
• SmartDraw.IMAGELISTINDEXCOUNT.DECREMENT : 감소 후 유지 [출력 순서 : 3,2,1,0,0,0,0,0,0,...]

C# 사용법

```
// 이미지 카운팅 방식을 증가 후 감소하도록 설정
smartDraw1.ImageListIndexIncType = SmartDraw.IMAGELISTINDEXCOUNT.INCDECREMENT;
```

VB 사용법

```
smartDraw1.ImageListIndexIncType = SmartDraw.IMAGELISTINDEXCOUNT.INCDECREMENT
```

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리 할 수 있는 속성입니다. 공통 영역에 위치한 SmartDraw의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

C# 사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) SetBackimage

Masking Image

리소스에 포함된 이미지 또는 이미지 파일을 Background Layer의 이미지로 출력합니다.

• Image : Background Layer에 출력할 이미지

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
Bitmap bkimg = new Bitmap( " Flash Disk₩₩BackImg.jpg " );  // 이미지 파일을 읽어옴
bkimg = new Bitmap(Test1.Resource1.BackImg);  // 이미지 파일이 리소스에 포함된 경우
smartDraw1.SetBackimage = bkimg;  // 읽어온 이미지를 배경 이미지로 적용
```

VB 사용법

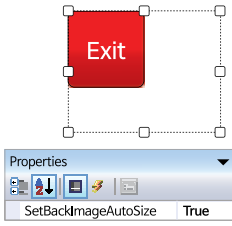
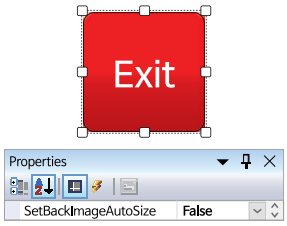
```
Dim bkimg As Bitmap = New Bitmap( " Flash Disk₩₩BackImg.jpg " )
bkimg = New Bitmap(Test1.My.Resources.Resource1.BackImg)
smartDraw1.SetBackimage = bkimg
```

프로퍼티(속성) SetBackImageAutoSize

BackPictureBox와 SetBackImage를 설정한 상태에서 SetBackImageAutoSize를 True로 할 경우 SmartDraw의 크기를 변경해도 BackImage의 크기는 원본사이즈를 유지하고, False로 할 경우 BackImage의 크기는 SmartDraw의 크기에 맞게 변경됩니다.

- bool : BackImage의 원본 크기 유지 여부
  - true : 원본 크기 유지
  - false : SmartDraw 크기에 맞게 변경

사용법

True	False
	

메소드(함수) ActiveViewToBackImage

현재 화면에 보여지는 상태를 BackGround Layer로 복사합니다. 즉 Drawing Layer에 그려진 그림 요소들은 BackGround Layer로 복사됩니다. 복사한 그림 요소는 Erase() 메소드를 호출해도 지워지지 않으며, BackErase() 메소드를 호출하여 지울 수 있습니다.

참고 "BackGround Layer와 Drawing Layer 설명" 내용을 참고하시기 바랍니다.

◉ void ActiveViewToBackImage()

C# 사용법

```
// 도형의 선 및 채울 스타일을 설정합니다.
smartDraw1.SetPenStyle(Color.Red, 3);
smartDraw1.SetBrushStyle(Color.Transparent);
// Drawing Layer에 도형 그리기
smartDraw1.Line(100, 100, 200, 200);
smartDraw1.Circle(300, 300, 50);
smartDraw1.Erase();  // Drawing Layer에 그려진 도형이 모두 지워집니다.
// ---------------- 위 코드와 아래 코드의 동작을 비교해 보시기 바랍니다.--------------------
// 도형의 선 및 채울 스타일을 설정합니다.
smartDraw1.SetPenStyle(Color.Red, 3);
smartDraw1.SetBrushStyle(Color.Transparent);
```

```
// Drawing Layer에 도형 그리기
smartDraw1.Line(100, 100, 200, 200);
smartDraw1.Circle(300, 300, 50);
// 현재 그려진 모든 그림 요소를 배경 Layer로 복사합니다.
smartDraw1.ActiveViewToBackImage();
// Drawing Layer에는 그려진 도형이 없으므로 화면은 변화가 없습니다.
smartDraw1.Erase();
```

VB 사용법

```
smartDraw1.SetPenStyle(Color.Red, 3)
smartDraw1.SetBrushStyle(Color.Transparent)
smartDraw1.Line(100, 100, 200, 200)
smartDraw1.Circle(300, 300, 50)
smartDraw1.Erase()
smartDraw1.SetPenStyle(Color.Red, 3)
smartDraw1.SetBrushStyle(Color.Transparent)
smartDraw1.Line(100, 100, 200, 200)
smartDraw1.Circle(300, 300, 50)
smartDraw1.ActiveViewToBackImage()
smartDraw1.Erase()
```

메소드(함수) AddImageList — Image

이미지 객체를 컬렉션(Collection) 형태로 관리 하도록 하기 위해서 이미지 객체를 추가 합니다.
이미지 컬렉션은 애니메이션 효과 또는 사용자 정의 인터페이스를 구성 하는데 편리합니다.
이미지는 추가(AddImageList)되는 순서대로 이미지 Index가 0부터 1씩 증가하여 적용됩니다.

◉ `void AddImageList(Image img)`
◉ `void AddImageList(string strPathName)`

[인자]

- `Image img` : 내부 컬렉션에 추가할 이미지 객체
- `string strPathName` : 추가할 이미지의 경로

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
smartDraw1.AddImageList(Resource1._0);              // 리소스 이미지를 내부 리스트에 추가합니다.
smartDraw1.AddImageList( " Flash Disk₩₩_0.png " ); // 외부 경로 이미지를 내부 리스트에 추가합니다.
```

VB 사용법

```
smartDraw1.AddImageList(Resource1._0)
smartDraw1.AddImageList( " Flash Disk₩₩_0.png " )
```

메소드(함수) ImageListClear

AddImageList() 메소드에 의해 추가된 이미지 리스트의 모든 이미지를 제거합니다.

◉ `void ImageListClear()`

C# 사용법

```
smartDraw1.ImageListClear();  // 모든 이미지 항목을 제거합니다.
```

VB 사용법

```
smartDraw1.ImageListClear()
```

메소드(함수) ImageListDraw

AddImageList() 메소드에 의해 이미지 리스트에 추가된 이미지를 Drawing Layer에 출력합니다.

참고 "AddImageList() 메소드" 내용을 참고하시기 바랍니다.

◉ void ImageListDraw(int iXPos, int iYPos, double dPercentage)
◉ void ImageListDraw(int iXPos, int iYPos, int iImgIndex, bool bAutoSize)

[인자]

- int iXPos : 출력되는 이미지 좌측 상단의 X좌표
- int iYPos : 출력되는 이미지 좌측 상단의 Y좌표
- double dPercentage : 이미지 리스트에 있는 이미지의 Index를 백분율로 계산하여 출력 (0 ~ 100%)
- int iImgIndex : 이미지 리스트에 추가된 이미지의 Index (0부터 시작)
- bool bAutoSize : SmartDraw 컨트롤의 크기 자동 조절 여부
  - true : SmartDraw 컨트롤의 크기가 이미지에 맞게 조절됨
  - false : SmartDraw 컨트롤의 크기가 이미지 크기와 상관없이 고정됨

C# 사용법

```
// 만약 이미지 리스트에 20개의 이미지가 있는경우 50% 해당되는 Index를 계산하여 출력합니다.
// 50% → Index가 10인 이미지를 Drawing Layer에 출력
smartDraw1.ImageListDraw(10, 10, 50.0);
// Index가 10인 이미지를 Drawing Layer에 출력합니다.
smartDraw1.ImageListDraw(10, 10, 10, true);
```

VB 사용법

```
smartDraw1.ImageListDraw(10, 10, 50.0)
smartDraw1.ImageListDraw(10, 10, 10, true)
```

메소드(함수) ImageListDrawNext

ImageListIndexIncType 속성에서 설정된 방식으로 ImageListDrawNext() 메소드 호출할 때마다 이미지 리스트에서 Index를 증가 및 감소하여 해당하는 이미지를 지정된 좌표에 출력합니다.

참고 "ImageListIndexIncType 속성", "AddImageList() 메소드" 내용을 참고하시기 바랍니다.

◉ void ImageListDrawNext(int iXPos, int iYPos)
◉ void ImageListDrawNext(int iXPos, int iYPos, bool bAutoSize)

[인자]

- int iXPos : 출력되는 이미지 좌측 상단의 X좌표
- int iYPos : 출력되는 이미지 좌측 상단의 Y좌표
- bool bAutoSize : SmartDraw 컨트롤의 크기 자동 조절 여부
  - true : 이미지에 맞게 조절됨
  - false : 이미지 크기와 상관없이 고정됨

C# 사용법

```
smartDraw1.ImageListDrawNext(0, 0, true);  // 다음 이미지를 Drawing Layer에 출력합니다.
```

VB 사용법

```
smartDraw1.ImageListDrawNext(0, 0, true)
```

메소드(함수) SlideShowStart, SlideShowStop

설정된 시간마다 이미지 리스트의 이미지를 ImageListIndexIncType 속성에서 설정된 방식으로 화면에 출력 합니다.

- SlideShowStart() : 자동 이미지 출력을 시작합니다.
- SlideShowStop() : 이미지 출력을 정지합니다.

◉ void SlideShowStart(int iInterval, int iXPos, int iYPos)
◉ void SlideShowStop()

[인자]

- int iInterval : 이미지 출력 간격
- int iXPos : 출력되는 이미지 좌측 상단의 X좌표
- int iYPos : 출력되는 이미지 좌측 상단의 Y좌표

C# 사용법

```
smartDraw1.SlideShowStart(100, 0, 0);  smartDraw1.SlideShowStop();
```

VB 사용법

```
smartDraw1.SlideShowStart(100, 0, 0)  :  smartDraw1.SlideShowStop()
```

메소드(함수) AnimationGIF

Animation GIF 파일을 화면에 출력 합니다.

참고 SmartDraw는 하나의 응용 프로그램에서 하나의 AnimationGIF만 사용 가능합니다.

주의 AnimationGIF() 메소드 사용 시 주의사항

AnimationGIF() 메소드는 호출 시 GIF 파일을 자동으로 Play합니다. 따라서 AnimationGIF() 메소드가 호출된 후 AnimationGIFPause() 메소드나 AnimationGIFStop() 메소드 호출 없이 AnimationGIF Play() 메소드를 호출할 경우 Play가 중복되어 GIF 재생에 문제가 생길 수 있습니다.

올바른 사용 방법	잘못된 사용 방법

// GIF 자동 Play
smartDraw1.AnimationGif(" Flash Disk₩₩12.GIF " , 35, 25);
smartDraw1.AnimationGIFStop(); // Play 종료
smartDraw1.AnimationGIFPlay(); // GIF Play
	// GIF 자동 Play
	smartDraw1.AnimationGif(" Flash Disk₩₩12.GIF " , 35, 25);
	// GIF Play. Play가 중복되어 문제 발생
	smartDraw1.AnimationGIFPlay();

◉ void AnimationGIF(string strGifPathName, int iXPos, int iYPos)

[인자]

- string strGifPathName : 출력할 Animation GIF 파일 경로
- int iXPos : 출력되는 이미지 좌측 상단의 X좌표
- int iYPos : 출력되는 이미지 좌측 상단의 Y좌표

C# 사용법

```
// 35, 25 위치에 GIF Animation 재생
smartDraw1.AnimationGIF( " Flash Disk₩₩12.GIF " , 35, 25);
```

VB 사용법

```
smartDraw1.AnimationGIF( " Flash Disk₩₩12.GIF " , 35, 25)
```

메소드(함수) AnimationGIFPause, AnimationGIFPlay, AnimationGIFStop

Animation GIF 파일의 출력 상태를 제어합니다.

- AnimationGIFPause() : GIF Animation이 Play 상태인 경우 일시 중지합니다.
- AnimationGIFPlay() : GIF Animation을 Play합니다.

• AnimationGIFStop() : GIF Animation을 Stop합니다.

◉ void AnimationGIFPause()
◉ void AnimationGIFPlay()
◉ void AnimationGIFStop()

C# 사용법

```
// GIF Animation 출력
smartDraw1.AnimationGIFPlay();
// GIF Animation 일시 중지
smartDraw1.AnimationGIFPause();
// GIF Animation Play 종료
smartDraw1.AnimationGIFStop();
```

VB 사용법

```
smartDraw1.AnimationGIFPlay()
smartDraw1.AnimationGIFPause()
smartDraw1.AnimationGIFStop()
```

메소드(함수) SetPenStyle, SetBrushStyle

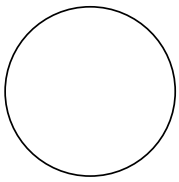
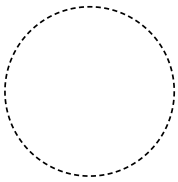
• SetPenStyle() : 도형이 그려질 때 선의 스타일(실선, 파선)과 색상 및 두께를 설정 합니다.
• SetBrushStyle() : 그려진 도형의 내부 색상을 설정합니다.

주의 그리기 관련 메소드 호출 전 반드시 SetPenStyle(), SetBrushStyle(), SetFontCfg() 메소드를 호출하여 스타일을 설정하시기 바랍니다. 설정하지 않을 경우 올바르게 지워지지 않거나, Draw되지 않을 수 있습니다.

참고 BackGround Layer에 도형을 그리는 경우 BackDraw 속성을 사용하여 설정하시기 바랍니다. 자세한 내용은 "BackDraw 속성"을 참고하시기 바랍니다.

참고 선의 스타일을 파선으로 설정하는 방법

SetPenStyle() 메소드의 오버로드 메소드를 이용하여 선, 도형의 테두리를 파선으로 그릴 수 있습니다.
이때 반드시 파선 설정을 먼저 호출한 후 선의 색과 두께를 설정해야 합니다. 또한, 선의 두께는 반드시 1로 설정해야 파선이 적용됩니다.

실선이 적용된 도형	파선이 적용된 도형
	
SmartDraw.PENSTYLES.SOLID	SmartDraw.PENSTYLES.DASH

1. Drawing Layer에 적용하는 방법

```
// PENSTYLES.DASH로 설정하고, 파선의 빈공간의 Color는 Black로 설정합니다.
smartDraw1.SetPenStyle(SmartX.SmartDraw.PENSTYLES.DASH, Color.Black);
// 선의 색을 설정하고, 두께는 반드시 1로 설정합니다.
smartDraw1.SetPenStyle(Color.LightSkyBlue, 1);
```

2. BackGroud Layer에 적용하는 방법

```
// PENSTYLES.DASH로 설정하고, 파선의 빈공간의 Color는 Black로 설정합니다.
smartDraw1.SetPenStyle(SmartX.SmartDraw.PENSTYLES.DASH, Color.Black);
// 선의 색을 설정하고, 두께는 반드시 1로 설정합니다.
smartDraw1.BackDraw.SetPenStyle(Color.Red, 1);
```

◉ void SetPenStyle(SmartDraw.PENSTYLES ePensStyle, Color cPenBackColor)
◉ void SetPenStyle(Color penColor, int iWidth)
◉ void SetBrushStyle(Color brushColor)

[인자]

- SmartDraw.PENSTYLES ePensStyle : 선의 스타일을 설정
  - SmartDraw.PENSTYLES.DASH : 파선
  - SmartDraw.PENSTYLES.SOLID : 실선
- Color cPenBackColor : 파선을 그릴 때 빈 공간에 적용할 색상
- Color penColor : 선의 색상
- int iWidth : 선의 두께
- Color brushColor : 도형 내부를 채울 색상 (Transparent로 설정 시 내부 투명 처리)

C# 사용법

```
// 도형의 선을 실선으로 설정
smartDraw1.SetPenStyle(SmartDraw.PENSTYLES.SOLID, Color.White);
smartDraw1.SetPenStyle(Color.Black, 1);
// Drawing Layer에 실선을 그립니다.
smartDraw1.Line(100, 35, 700, 35);
// 도형의 선을 파선으로 설정
smartDraw1.BackDraw.SetPenStyle(SmartDraw.PENSTYLES.DASH, Color.White);
smartDraw1.BackDraw.SetPenStyle(Color.Black, 1);
// BackGround Layer에 파선을 그립니다.
smartDraw1.BackDraw.Line(100, 75, 700, 75);
smartDraw1.SetPenStyle(SmartDraw.PENSTYLES.DASH, Color.White);  // 도형의 선을 파선으로 설정
smartDraw1.SetPenStyle(Color.Red, 1);  smartDraw1.SetBrushStyle(Color.Transparent);
smartDraw1.RoundRect(500, 150, 200, 100, 30);  // 내부가 투명한 둥근 사각형을 그립니다.
smartDraw1.BackDraw.SetPenStyle(SmartDraw.PENSTYLES.DASH, Color.White);
smartDraw1.BackDraw.SetPenStyle(Color.Red, 1);  smartDraw1.BackDraw.SetBrushStyle(Color.Red);
smartDraw1.BackDraw.Circle(600, 70, 100);  // BackGround Layer에 내부가 빨간색인 원을 그립니다.
```

VB 사용법

```
smartDraw1.SetPenStyle(SmartDraw.PENSTYLES.SOLID, Color.White)
smartDraw1.SetPenStyle(Color.Black, 1) : smartDraw1.Line(100, 35, 700, 35)
smartDraw1.BackDraw.SetPenStyle(SmartDraw.PENSTYLES.DASH, Color.White)
smartDraw1.BackDraw.SetPenStyle(Color.Black, 1)
smartDraw1.BackDraw.Line(100, 75, 700, 75)
smartDraw1.SetPenStyle(SmartDraw.PENSTYLES.DASH, Color.White)
smartDraw1.SetPenStyle(Color.Red, 1)  :  smartDraw1.SetBrushStyle(Color.Transparent)
smartDraw1.RoundRect(500, 150, 200, 100, 30)
smartDraw1.BackDraw.SetPenStyle(SmartDraw.PENSTYLES.DASH, Color.White)
smartDraw1.BackDraw.SetPenStyle(Color.Red, 1) : smartDraw1.BackDraw.SetBrushStyle(Color.Red)
smartDraw1.BackDraw.Circle(600, 70, 100)
```

메소드(함수) SetFontCfg

TextOut() 메소드로 화면에 Text를 출력할 경우 폰트 관련 여러가지 속성 값을 설정 합니다.

주의 그리기 관련 메소드 호출 전 반드시 SetPenStyle(), SetBrushStyle(), SetFontCfg() 메소드를 호출하여 스타일을 설정하시기 바랍니다. 설정하지 않을 경우 올바르게 지워지지 않거나, Draw되지 않을 수 있습니다.

참조 다양한 폰트를 추가하는 방법은 "홈페이지 → 자료실 → Tech Note → 20. IEC-Series에서 다양한 폰트(Fonts 폴더, 트루타입)사용 방법 안내" 내용을 참조하시기 바랍니다.

참고 BackGround Layer에 Text를 출력하는 경우 BackDraw 속성을 사용하여 설정하시기 바랍니다. 자세한 내용은 "BackDraw 속성"을 참고하시기 바랍니다.

◉ `void SetFontCfg(int iFontSize, Color cFontColor)`
◉ `void SetFontCfg(string strFontName, int iFontSize, Color cFontColor)`
◉ `void SetFontCfg(string strFontName, int iFontSize, Color cFontColor, bool bBold, bool bItalic, bool bAntiAlased)`

[인자]

- `int iFontSize` : 출력되는 폰트의 크기
- `Color cFontColor` : 출력되는 폰트의 색상
- `bool bBold` : 출력되는 폰트의 굵기
  - `true` : 굵게
  - `false` : 표준
- `bool bItalic` : 출력되는 폰트의 기울임
  - `true` : 기울임
  - `false` : 표준
- `bool bAntiAlased` : 출력되는 폰트를 배경과 글자의 안티엘리어싱(Anti-Aliasing) 효과를 설정
  - `true` : 안티엘리어싱 사용
  - `false` : 안티엘리어싱 사용 안 함

사용법

참고 TextOut() 메소드의 사용법을 참고하시기 바랍니다.

메소드(함수) TextOut

Drawing Layer 또는 BackGround Layer에 Text를 출력합니다. 폰트 관련 설정은 SetFontCfg() 메소드로 설정합니다.

참고
1. "SetFontCfg() 메소드" 내용을 참고하시기 바랍니다.
2. BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

◉ `void TextOut(int iXpos, int iYpos, string strFormat, params object[] args)`

[인자]

- `int iXpos` : 출력되는 Text 좌측 상단의 X좌표
- `int iYpos` : 출력되는 Text 좌측 상단의 Y좌표
- `string strFormat` : 출력될 문자 (형식 문자열 지원)
- `params object[] args` : 형식 문자열에 대입하는 객체(데이터) 가변 인자

C# 사용법

```
//Drawing Layer에 텍스트를 출력합니다.
smartDraw1.SetFontCfg( " 바탕 " , 20, Color.LightSkyBlue, true, false, true);
smartDraw1.TextOut(10, 70, " HNS / SmartX-SmartDraw / IEC-Series / www.hnsts.co.kr " );
//Background Layer에 텍스트를 출력합니다.
smartDraw1.BackDraw.SetFontCfg(30, Color.LightPink);
smartDraw1.BackDraw.TextOut(10, 110, " 에이치엔에스 / 스마트엑스 / 아이이씨-시리즈 " );
```

VB 사용법

```
smartDraw1.SetFontCfg( " 바탕 " , 20, Color.LightSkyBlue, true, false, true)
smartDraw1.TextOut(10, 70, " HNS / SmartX-SmartDraw / IEC-Series / www.hnsts.co.kr " )
smartDraw1.BackDraw.SetFontCfg(30, Color.LightPink)
smartDraw1.BackDraw.TextOut(10, 110, " 에이치엔에스 / 스마트엑스 / 아이이씨-시리즈 " )
```

참고 형식 문자열에 대하여

형식 문자열 지정 시 반드시 다음과 같은 형태를 해야 하며 중괄호(“{“와 “}”)의 짝이 반드시 맞아야 합니다. 구성 요소에 대한 설명은 아래 내용을 확인해 보시기 바랍니다.

※ 형식 문자열 형태 : “{Index [, Alignment][:Format String]}”

Ex) “{0,10:F2}” : Index = 0 / Alignment = 10 / Format String = F2

1. Index 구성 요소

매개 변수 지정자라고도 하는 필수 Index 구성 요소는 0부터 시작하는 숫자로서, 개체 목록에서 해당하는 항목을 식별합니다. 즉, 매개 변수 지정자가 0인 형식 항목은 목록에 있는 첫 번째 개체의 형식을 지정하고 매개 변수 지정자가 1인 형식 항목은 목록에 있는 두 번째 개체의 형식을 지정하는 식으로 적용됩니다. 동일한 매개 변수 지정자를 지정하여 여러 형식 항목이 개체 목록의 동일한 요소를 참조하도록 할 수 있습니다. 예를 들어, 합성 형식 문자열을 “{0:X}” “{0:E}” “{0:N}”과 같이 지정하여 동일한 숫자값을 16진수, 지수 및 숫자 형식으로 지정할 수 있습니다.

각 형식 항목은 목록의 어떤 개체나 참조할 수 있습니다. 예를 들어, 세 개의 개체가 있을 경우 “{1} {0} {2}”와 같이 복합 형식 문자열을 지정하여 두 번째, 첫 번째 및 세 번째 개체의 형식을 지정할 수 있습니다. 형식 항목에서 참조하지 않는 개체는 무시됩니다. 매개 변수 지정자가 개체 목록 범위를 벗어나는 항목을 지정하면 런타임 예외가 발생합니다.

2. Alignment 구성 요소

선택적인 Alignment 구성 요소는 기본 형식의 필드 너비를 나타내는 부호있는 정수입니다. Alignment 값이 형식이 지정된 문자열보다 작으면 Alignment는 무시되고 형식이 지정된 문자열의 길이가 필드 너비로 사용됩니다. Alignment가 양수이면 필드에서 형식이 지정된 데이터가 오른쪽 맞춤되고 음수이면 왼쪽 맞춤됩니다. 채우기가 필요하면 공백이 사용됩니다. Alignment가 지정되면 앞쪽에 쉼표를 사용해야 합니다.

우측 정렬이 잘되지 않는 경우에는 SetFontCfg() 메소드에서 선택한 Font를 먼저 확인합니다.

Courier New Font의 경우 숫자와 .(소수점)의 가로/세로 사이즈가 동일합니다. 즉, 우측 정렬됩니다. 굴림, Tahoma Font의 경우 숫자와 .(소수점)의 가로/세로 사이즈가 다르므로 우측 정렬이 되지 않습니다. 위의 사항을 고려하여 우측 정렬이 필요하신 경우 가로/세로 사이즈가 동일한 폰트를 선택하시기 바랍니다.

[표] Font 별 Alignment 지정에 따른 결과

Font	Courier New	Tahoma
스타일 설정	`smartDraw1.SetFontCfg( “ Courier New ” , 30, Color .LightBlue, true, false, true);`	`smartDraw1.SetFontCfg( “ Tahoma ” , 30, Color .LightBlue, true, false, true);`

텍스트 출력	`// 고정 소수점 두 번째 자리로 우측 정렬`
	`smartDraw1.TextOut(10, 70, “ {0,10:F2} ” , 999.9);`
	`smartDraw1.TextOut(10, 90, “ {0,10:F2} ” , 9999.9);`
	`smartDraw1.TextOut(10, 110, “ {0,10:F2} ” , 800);`	

출력 결과	999.90 9999.90 800.00	999.90 9999.90 800.00

3. Format String 구성 요소

선택적 Format String 구성 요소는 형식을 지정할 개체 형식에 적절한 형식 문자열입니다. 해당 개체가 숫자값이면 숫자 형식 문자열을, DateTime이면 날짜 및 시간 형식 문자열을, 열거형 값이면 열거형 형식 문자열을 지정합니다. Format String을 지정하지 않으면 숫자, 날짜 및 시간, 또는 열거형 형식에 대해 일반(“G”) 형식 지정자가 사용됩니다. Format String을 지정하는 경우에는 콜론(:)이 필요합니다.

[표1] 숫자 형식 포맷팅

형식지정자	종 류	예제 코드	출력 결과
C / c	통화 Currency	`TextOut(10,10, ” {0:C} ” , 2.5);` `TextOut(10,10, ” {0:C} ” , -2.5);`	3 (3)
D / d	10진법 Decimal	`TextOut(10,10, ” {0:D5} ” , 25);`	00025
E / e	과학적 지수 Scientific	`TextOut(10,10, ” {0:E} ” , 250000);`	2.500000E+005
F / x	고정 소수점 Fixed-point	`TextOut(10,10, ” {0:F2} ” , 25);` `TextOut(10,10, ” {0:F0} ” , 25);`	25.00 25

G / g	일반 General	TextOut(10,10,"{0:G}", 2.5);	2.5
N / n	숫자 Number	TextOut(10,10,"{0:N}", 2500000);	2,500,000.00
P / p	백분율 Percentage	TextOut(10,10,"{0:P}", .2468013); TextOut(10,10,"{0:P1}", .2468013);	24.68% 24.7 %
X / x	16진법 Hexadecimal	TextOut(10,10,"{0:X}", 250); TextOut(10,10,"{0:X}", 0xffff);	FA FFFF

[표2] 날짜 형식 포맷팅

형식지정자	종 류	예제 코드	출력 결과
d	간단한 날짜 패턴	DateTime dt = new DateTime(2021,1,15); TextOut(10,10,"{0:d}", dt);	2021-1-15
D	자세한 날짜 패턴	TextOut(10,10,"{0:D}", dt);	2021년 1월 15일 금요일
f	전체 날짜/시간 패턴 (간단한 시간)	DateTime dt = new DateTime(2021,1,15,16,3,52); TextOut(10,10,"{0:f}", dt);	2021년 1월 15일 금요일 오후 4:03
F	전체 날짜/시간 패턴 (자세한 시간)	DateTime dt = new DateTime(2021,1,15,16,3,52); TextOut(10,10,"{0:F}", dt);	2021년 1월 15일 금요일 오후 4:03:52
g	일반 날짜/시간 패턴 (간단한 시간)	DateTime dt = new DateTime(2021,1,15,16,3,52); TextOut(10,10,"{0:g}", dt);	2021-1-15 오후 4:03
G	일반 날짜/시간 패턴 (자세한 시간)	DateTime dt = new DateTime(2021,1,15,16,3,52); TextOut(10,10,"{0:G}", dt);	2021-1-15 오후 4:03:52
M / m	월 일 패턴	DateTime dt = new DateTime(2021,1,15,16,3,52); TextOut(10,10,"{0:M}", dt);	1월 15일
t	간단한 시간 패턴	DateTime dt = new DateTime(2021,1,15,16,3,52); TextOut(10,10,"{0:t}", dt);	오후 4:03

[표3] 이스케이프 시퀀스

이스케이프 시퀀스	문자 이름	유니코드 코딩
₩'	작은 따옴표	0x0027
₩"	큰 따옴표	0x0022
₩₩	백 슬래시	0x005C
₩0	Null	0x0000
₩a	경고	0x0007
₩b	백 스페이스	0x0008
₩f	폼 피드	0x000C
₩n	줄 바꿈	0x000A

메소드(함수) BackErase

Background Layer 및 Drawing Layer에 그려진 모든 그림 요소를 지웁니다. 화면이 지워지면 배경 색상은 인자값으로 설정한 색상으로 변경됩니다.

참고 "BackGround Layer와 Drawing Layer 설명" 내용을 참고하시기 바랍니다.

주의 Erase(), BackErase() 메소드 호출 후 반드시 SetPenStyle(), SetBrushStyle(), SetFontCfg() 메소드를 호출하여 스타일을 설정하시기 바랍니다. 설정하지 않을 경우 올바르게 지워지지 않거나, Draw되지 않을 수 있습니다.

◉ void BackErase(Color bColor)

[인자]

• Color bColor : 배경 색상으로 칠할 색상

C# 사용법

```
smartDraw1.BackDraw.BackErase(Color.Black);  // BackGround 및 Drawing Layer의 모든 그림 요소 지움
// BackErase 후 스타일을 재설정
smartDraw1.BackDraw.SetBrushStyle(Color.LightGreen);
```

```
smartDraw1.BackDraw.SetPenStyle(Color.LightSkyBlue, 3);
smartDraw1.BackDraw.SetFontCfg(15, Color.LightSkyBlue);
```

VB 사용법

```
smartDraw1.BackDraw.BackErase(Color.Black)
smartDraw1.BackDraw.SetBrushStyle(Color.LightGreen)
smartDraw1.BackDraw.SetPenStyle(Color.LightSkyBlue, 3)
smartDraw1.BackDraw.SetFontCfg(15, Color.LightSkyBlue)
```

메소드(함수) Erase

Drawing Layer에 그려진 그림 요소를 모두 지웁니다. BackGround Layer에 그려진 그림 요소는 지워지지 않으며, 화면이 지워지면 배경 색상은 흰색으로 됩니다.

참고 "BackGround Layer와 Drawing Layer 설명" 내용을 참고하시기 바랍니다.

◉ `void Erase()`

C# 사용법

```
smartDraw1.Erase();  // Drawing Layer에 그려진 그림 요소를 모두 지움
```

VB 사용법

```
smartDraw1.Erase()
```

메소드(함수) SetPixel

Drawing Layer 또는 BackGround Layer의 지정된 좌표에 색상을 지정하여 점을 그립니다.

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

◉ `void SetPixel(int iXPos, int iYPos, Color crColor)`

[인자]

- `int iXPos` : 점의 X좌표
- `int iYPos` : 점의 Y좌표
- `Color crColor` : 그려질 점의 색상

C# 사용법

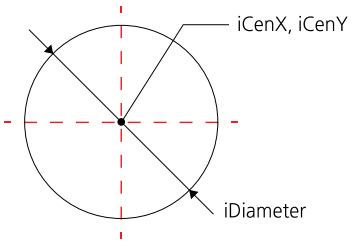
```
// Drawing Layer에 점 그리기
smartDraw1.SetPixel(100, 200, Color.LightCyan);
```

VB 사용법

```
smartDraw1.SetPixel(100, 200, Color.LightCyan)
```

메소드(함수) Circle

Drawing Layer 또는 BackGround Layer에 원을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며, 내부 채움은 SetBrushStyle() 메소드로 설정합니다.



참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

◉ void Circle(int iCenX, int iCenY, int iDiameter)

[인자]

- int iCenX : 원 중심의 X좌표
- int iCenY : 원 중심의 Y좌표
- int iDiameter : 원의 지름

C# 사용법

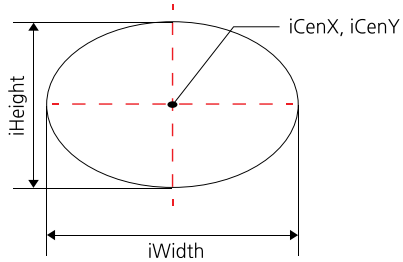
```
// Drawing Layer에 원 그리기
smartDraw1.Circle(300, 300, 50);
```

VB 사용법

```
smartDraw1.Circle(300, 300, 50)
```

메소드(함수) Ellipse

Drawing Layer 또는 BackGround Layer에 타원을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며, 내부 채움은 SetBrushStyle() 메소드로 설정합니다.



참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

◉ void Ellipse(int iCenX, int iCenY, int iWidth, int iHeight)

[인자]

- int iCenX : 원 중심의 X좌표
- int iCenY : 원 중심의 Y좌표
- int iWidth : 타원의 수직 길이
- int iHeight : 타원의 수평 길이

C# 사용법

```
smartDraw1.Ellipse(300, 300, 50);  // Drawing Layer에 원 그리기
```

VB 사용법

```
smartDraw1.Ellipse(300, 300, 50)
```

메소드(함수) GradientFill

Drawing Layer 또는 BackGround Layer에 사각형 영역을 지정된 색으로 그라데이션 효과를 주어 그립니다.

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

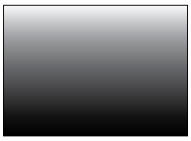
◉ void GradientFill(int iLeftX, int iTopY, int iWidth, int iHeight, Color startColor, Color endColor, SmartDraw.GRADIENTFILLMODE dwMode)

[인자]

- int iLeftX : 사각형 좌측 상단의 X좌표

- int iTopY : 사각형 좌측 상단의 Y좌표
- int iWidth : 사각형의 수평 길이
- int iHeight : 사각형의 수직 길이
- Color startColor : 그라데이션의 시작 색상
- Color endColor : 그라데이션의 끝 색상
- SmartDraw.GRADIENTFILLMODE dwMode : 그라데이션 방향
  - SmartDraw.GRADIENTFILLMODE.GRADIENT_FILL_RECT_H : 그라데이션 방향을 수평으로 설정
  - SmartDraw.GRADIENTFILLMODE.GRADIENT_FILL_RECT_V : 그라데이션 방향을 수직으로 설정

[표] GRADIENTFILLMODE 인자값에 따른 표현 예시

GRADIENT_FILL_RECT_H	GRADIENT_FILL_RECT_V
 수평 그라데이션	 수직 그라데이션

C# 사용법

```
// Drawing Layer에 그라데이션이 적용된 사각형을 그립니다.
// 시작 위치 (0,0), 폭 : 800, 높이 : 480, 시작 색상 : LightPink, 끝 색상 : SkyBlue, 수직 그라데이션
smartDraw1.GradientFill(0, 0, 800, 480, Color.LightPink, Color.SkyBlue, SmartDraw.GRADIENTFILLMODE
.GRADIENT_FILL_RECT_V);
```

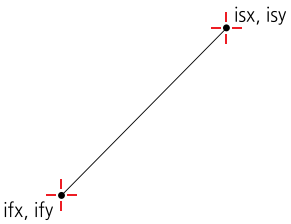
VB 사용법

```
smartDraw1.GradientFill(0, 0, 800, 480, Color.LightPink, Color.SkyBlue, SmartDraw.GRADIENTFILLMODE
.GRADIENT_FILL_RECT_V)
```

메소드(함수) Line

Drawing Layer에 선을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정합니다.

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.



◉ void Line(int ifx, int ify, int isx, int isy)

[인자]

- int ifx : 처음 점의 X좌표
- int ify : 처음 점의 Y좌표
- int isx : 두번째 점의 X좌표
- int isy : 두번째 점의 Y좌표

C# 사용법

```
smartDraw1.Line(100, 100, 200, 200);  // Drawing Layer에 선 그리기
```

VB 사용법

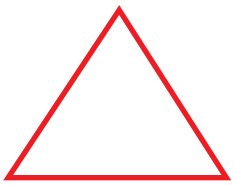
```
smartDraw1.Line(100, 100, 200, 200)
```

메소드(함수) Polygon, Polyline

Polygon() 메소드와 Polyline() 메소드는 연속된 여러 개의 선을 Drawing Layer 또는 BackGround Layer에 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며, Polygon() 메소드의 경우 SetBrushStyle() 메소드로 설정된 색상으로 안쪽이 채워집니다

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

[표] Polygon() 메소드와 Polyline() 메소드의 차이점

메소드	Polygon	Polyline
차이점	1. 시작과 끝점이 연결됨 2. 연결된 선의 안쪽이 채워짐	1. 시작과 끝점이 연결되지 않음 2. 연결된 선의 안쪽이 채워지지 않음
출력 결과		

◉ void Polygon(SmartDraw.POINT[] pPoint)
◉ void Polyline(SmartDraw.POINT[] pPoint)

[인자]

- SmartDraw.POINT[] pPoint : 연속된 선을 그리기 위한 여러 점의 좌표 배열

C# 사용법

```
// Polygon으로 별을 그립니다.
SmartDraw.POINT[] points = new SmartDraw.POINT[10];
int iSize = 5, iXOffset = 300, iYOffset = 50;
// 별 안쪽의 채워질 색상을 지정합니다.
smartDraw1.SetBrushStyle(Color.Yellow);
// 별의 외곽선의 색상 및 두께를 지정합니다.
smartDraw1.SetPenStyle(Color.Red, 2);
// 별 모양의 좌표를 지정합니다.
points[0].X = (16 * iSize) + iXOffset; points[0].Y = (28 * iSize) + iYOffset;
points[1].X = (19 * iSize) + iXOffset; points[1].Y = (17 * iSize) + iYOffset;
points[2].X = (30 * iSize) + iXOffset; points[2].Y = (17 * iSize) + iYOffset;
points[3].X = (21 * iSize) + iXOffset; points[3].Y = (11 * iSize) + iYOffset;
points[4].X = (25 * iSize) + iXOffset; points[4].Y = (1 * iSize) + iYOffset;
points[5].X = (16 * iSize) + iXOffset; points[5].Y = (7 * iSize) + iYOffset;
points[6].X = (7 * iSize)  + iXOffset; points[6].Y = (1 * iSize) + iYOffset;
points[7].X = (11 * iSize) + iXOffset; points[7].Y = (11 * iSize) + iYOffset;
points[8].X = (2 * iSize)  + iXOffset; points[8].Y = (17 * iSize) + iYOffset;
points[9].X = (13 * iSize) + iXOffset; points[9].Y = (17 * iSize) + iYOffset;
// 설정된 그림 요소를 Drawing Layer에 그립니다.(별 모양)
smartDraw1.Polygon(points);
// Polyline으로 삼각형을 그립니다. Polyline은 시작점과 끝점을 연결하지 않습니다.
SmartDraw.POINT[] points1 = new SmartDraw.POINT[4];
// 삼각형 모양의 좌표를 지정합니다.
points1[0].X = 400; points1[0].Y = 200; points1[1].X = 300; points1[1].Y = 300;
points1[2].X = 500; points1[2].Y = 300; points1[3].X = 400; points1[3].Y = 200;
// 설정된 그림 요소를 Drawing Layer에 그립니다.(삼각형, 내부는 투명함)
smartDraw1.Polyline(points1);
```

VB 사용법

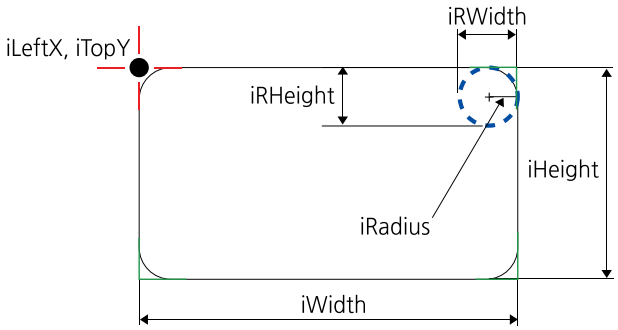
```
Dim points As SmartDraw.POINT() = New SmartDraw.POINT(9) {}
Dim iSize, iXOffset, iYOffset As Integer
iSize = 5 : iXOffset = 300 : iYOffset = 50
smartDraw1.SetBrushStyle(Color.Yellow) : smartDraw1.SetPenStyle(Color.Red, 2)
points(0).X = (16 * iSize) + iXOffset : points(0).Y = (28 * iSize) + iYOffset
points(1).X = (19 * iSize) + iXOffset : points(1).Y = (17 * iSize) + iYOffset
points(2).X = (30 * iSize) + iXOffset : points(2).Y = (17 * iSize) + iYOffset
points(3).X = (21 * iSize) + iXOffset : points(3).Y = (11 * iSize) + iYOffset
points(4).X = (25 * iSize) + iXOffset : points(4).Y = (1 * iSize) + iYOffset
points(5).X = (16 * iSize) + iXOffset : points(5).Y = (7 * iSize) + iYOffset
points(6).X = (7 * iSize)  + iXOffset : points(6).Y = (1 * iSize) + iYOffset
points(7).X = (11 * iSize) + iXOffset : points(7).Y = (11 * iSize) + iYOffset
points(8).X = (2 * iSize)  + iXOffset : points(8).Y = (17 * iSize) + iYOffset
points(9).X = (13 * iSize) + iXOffset : points(9).Y = (17 * iSize) + iYOffset
smartDraw1.Polygon(points)
Dim points1 As SmartDraw.POINT() = New SmartDraw.POINT(3) {}
points1(0).X = 400 : points1(0).Y = 200 : points1(1).X = 300 : points1(1).Y = 300
points1(2).X = 500 : points1(2).Y = 300 : points1(3).X = 400 : points1(3).Y = 200
smartDraw1.Polyline(points1)
```

메소드(함수) Rectangle, RoundRect

Drawing Layer 또는 BackGround Layer에 내부가 채워진 직사각형을 그립니다. 선의 색상 및 두께는 SetPenStyle() 메소드로 설정하며 내부 채움은 SetBrushStyle() 메소드로 설정합니다.

- Rectangle() : 직사각형을 그립니다.
- RoundRect() : 모서리가 둥근 직사각형을 그립니다.

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.



◉ void Rectangle(int iLeftX, int iTopY, int iWidth, int iHeight)
◉ void RoundRect(int iLeftX, int iTopY, int iWidth, int iHeight, int iRadius)
◉ void RoundRect(int iLeftX, int iTopY, int iWidth, int iHeight, int iRWidth, int iRHeight)

[인자]

- int iLeftX : 출력될 직사각형 좌측 상단의 X좌표
- int iTopY : 출력될 직사각형 좌측 상단의 Y좌표
- int iWidth : 출력될 직사각형의 수평 길이
- int iHeight : 출력될 직사각형의 수직 길이
- int iRadius : RoundRect에서 모서리의 라운드의 반지름
- int iRWidth : RoundRect에서 모서리의 라운드의 수평 길이
- int iRHeight : RoundRect에서 모서리의 라운드의 수직 길이

C# 사용법

```
// Drawing Layer에 내부가 채워진 직사각형을 그립니다.
smartDraw1.Rectangle(10, 10, 300, 200);
// Drawing Layer에 내부가 채워진 모서리가 둥근 직사각형을 그립니다.
smartDraw1.RoundRect(300, 200, 300, 200, 20);
```

VB 사용법

```
smartDraw1.Rectangle(10, 10, 300, 200)
smartDraw1.RoundRect(300, 200, 300, 200, 20)
```

메소드(함수) ImageDraw Image

Drawing Layer에 리소스로 포함된 이미지 또는 이미지 파일을 그립니다. (지원 이미지 파일 형식 : BMP, JPG, GIF, PNG)

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

- void ImageDraw(Image img, int iXpos, int iYpos)
- void ImageDraw(string strFilePath, int iXpos, int iYpos)
- void ImageDraw(Image img, int iXpos, int iYpos, int iWidth, int iHeight)
- void ImageDraw(string strFilePath, int iXpos, int iYpos, int iWidth, int iHeight)

[인자]

- string strImgPath : 이미지 파일의 경로와 파일명 (필수)
- Image img : 이미지 객체로 주로 이미지 파일이 리소스로 포함된 경우 사용 (필수)
- int iXpos : 출력될 이미지 좌측 상단의 X좌표 (필수)
- int iYpos : 출력될 이미지 좌측 상단의 Y좌표 (필수)
- int iWidth : 출력될 이미지의 수평 길이 (옵션 : Scale 조정 시 사용)
- int iHeight : 출력될 이미지의 수직 길이 (옵션 : Scale 조정 시 사용)

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
// " Flash Disk₩₩hns.bmp " 이미지 파일을 읽어온 후 이미지 객체 선언 및 생성
Bitmap img = new Bitmap( " Flash Disk₩₩hns.bmp " );
// 지정된 이미지 파일을 10, 350에 원본 이미지 크기로 Drawing Layer에 그립니다.
smartDraw1.ImageDraw(img, 10, 350);
// 리소스에 포함된 이미지 파일을 10, 10에 원본 이미지 크기로 Drawing Layer에 그립니다.
smartDraw1.ImageDraw(Resource1.SmartX_Logo, 10, 10);
```

VB 사용법

```
Dim img As Bitmap = New Bitmap( " Flash Disk₩₩hns.bmp " )
smartDraw1.ImageDraw(img, 10, 350)
smartDraw1.ImageDraw(Resource1.SmartX_Logo, 10, 10)
```

메소드(함수) TransparentImageDraw Masking Image

Drawing Layer 또는 BackGround Layer에 리소스로 포함된 이미지 또는 이미지 파일의 화면에 배경 색상을 지정하여 투명하게 그립니다. (지원 이미지 파일 형식 : BMP, JPG, GIF, PNG)

참고 BackGround Layer에 그릴 수 있으며, "BackDraw 속성" 내용을 참고하시기 바랍니다.

- void TransparentImageDraw(Image img, int iXpos, int iYpos, Color cTransparentColor)
- void TransparentImageDraw(string strImgPath, int iXpos, int iYpos, Color cTransparentColor)
- void TransparentImageDraw(Image img, int iXpos, int iYpos, int iWidth, int iHeight, Color cTransparentColor)
- void TransparentImageDraw(string strImgPath, int iXpos, int iYpos, int iWidth, int iHeight, Color cTransparentColor)

[인자]

- Image img : 이미지 객체로 주로 이미지 파일이 리소스로 포함된 경우 사용 (필수)
- int iXpos : 출력될 이미지 좌측 상단의 X좌표 (필수)
- int iYpos : 출력될 이미지 좌측 상단의 Y좌표 (필수)
- Color cTransparentColor : 출력될 이미지에서 투명하게 처리할 색상
- string strImgPath : 이미지 파일의 경로와 파일명 (필수)
- int iWidth : 출력될 이미지의 수평 길이 (옵션 : Scale 조정 시 사용)
- int iHeight : 출력될 이미지의 수직 길이 (옵션 : Scale 조정 시 사용)

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
// 그라데이션 효과로 배경을 그림
smartDraw1.GradientFill(0, 0, smartDraw1.Width, smartDraw1.Height, Color.LightBlue,
Color.LightPink, SmartDraw.GRADIENTFILLMODE.GRADIENT_FILL_RECT_H);
// 이미지 객체 선언 및 생성. " Flash Disk₩₩hns.png " 이미지 파일을 읽어옵니다.
Bitmap img = new Bitmap( " Flash Disk₩₩hns.png " );
// 지정된 이미지 파일을 불러와 지정된 색상을 투명하게 원본 이미지 크기로 출력합니다.
smartDraw1.TransparentImageDraw(img, 10, 250, Color.White);
// 리소스에 포함된 이미지 파일을 지정된 색상으로 투명하게 원본 이미지 크기로 출력합니다.
smartDraw1.TransparentImageDraw(Resources.SmartX_Logo, 30, 30, Color.White);
```

VB 사용법

```
smartDraw1.GradientFill(0, 0, SmartDraw1.Width, SmartDraw1.Height, Color.LightBlue,
Color.LightPink, SmartDraw.GRADIENTFILLMODE.GRADIENT_FILL_RECT_H)
Dim img As Bitmap = New Bitmap( " Flash Dis₩₩hns.png " )
smartDraw1.TransparentImageDraw(img, 10, 250, Color.White)
smartDraw1.TransparentImageDraw(My.Resources.Resource1.smartx1, 30, 30, Color.White)
```

메소드(함수) RotationImage Masking Image

회전 이미지, 배경 이미지, 회전 이미지의 중심 좌표, Offset 정보를 바탕으로 사용자가 설정한 각도만큼 이미지를 회전시킵니다. 회전 이미지인 rotateImage는 기준점 iCenterX, iCenterY 좌표를 중심으로 회전하며, 회전된 이미지는 backResultImage 인자값에 저장됩니다.

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

특히 회전 이미지는 아래 그림에 "마스킹 처리" 표시된 영역이 마스킹 처리되는 부분이며 배경 이미지 위에 투명 처리되어 표현됩니다. 따라서 회전 이미지 제작 시 아래 STEP 순서로 제작하지 않는 경우 이미지가 자연스럽게 표현되지 않을 수 있습니다.

[회전 이미지 제작 예시]

[STEP-1] 투명 처리할 배경에 마스킹 영역을 만들어 줍니다

마스킹 영역은 회전체가 기준점을 중심으로 360'도 회전하는 경우 포함할 수 있는 크기로 설정하시기 바랍니다.

[STEP-2] 투명 처리할 마스킹 영역에 이미지에 사용되지 않는 색상으로 채웁니다.

이때 마스킹 영역의 색상은 배경이 될 이미지와 유사한 색상으로 하시기 바랍니다.

[STEP-3] 마스킹 영역과 이미지의 경계 부분을 안티엘리어싱(Anti-Aliasing) 효과를 줍니다.

경계 부분을 배경 이미지에 적용되는 색상의 중간 값으로 주는 경우 경계선이 가장 자연스럽게 표현됩니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

참조

SmartDraw를 사용하여 회전체를 그리는 방식은 회전체의 종류에 따라 크게 이미지 또는 선/도형의 두가지 방식으로 나누어 집니다. 회전체가 이미지인 경우 RotationImage() 메소드를 사용하며 선/도형인 경우 RotationPoints() 메소드를 사용합니다.

"홈페이지 → 자료실 → Tech Note → 43. [C#, VB.NET] 이미지 및 도형을 회전하여 (계기판)을 표현하는 방법과 예제" 내용을 참조하시기 바랍니다.

◉ `void RotationImage(Bitmap rotateImage, Bitmap backResultImage, int iCenterX, int iCenterY, int iOffsetX, int iOffsetY, double degrees)`

[인자]

- `Bitmap rotateImage` : 회전체 이미지
- `Bitmap backResultImage` : 배경 이미지. (회전된 이미지가 객체에 저장 됨)
- `int iCenterX` : 회전체 이미지의 중심 기준점 X좌표
- `int iCenterY` : 회전체 이미지의 중심 기준점 Y좌표
- `int iOffsetX` : SmartDraw의 좌측 상단 기준에서 Offset 만큼의 X좌표값
- `int iOffsetY` : SmartDraw의 좌측 상단 기준에서 Offset 만큼의 Y좌표값
- `double degrees` : 회전 이미지를 회전 시킬 각도

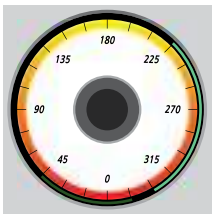
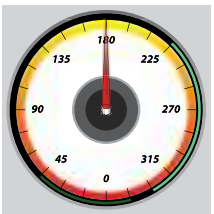
참고 RotationImage() 메소드 사용 시 참고사항

RotationImage() 메소드 사용 시 인자값에 대한 설명입니다.

1. rotateImage, backResultImage 인자값 설명

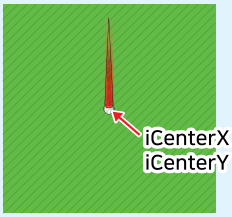
RotationImage() 메소드 호출 시 rotateImage(회전체 이미지)의 iCenterX, iCenterY(중심 기준점)을 기준으로 degrees(회전 각도)만큼 회전 시킨 이미지가 backResultImage(배경 이미지)에 포함되어, 사용자는 backResultImage를 화면에 출력시켜 회전체를 표현할 수 있습니다.

[표] RotationImage() 메소드 호출에 따른 backResultImage 객체의 변화

backResultImage (배경 이미지)	rotateImage (회전체 이미지)	RotationImage() 메소드 호출 후 (rotateImage + backResultImage)
		

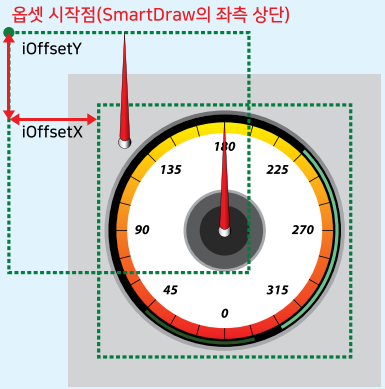
2. iCenterX, iCenterY 인자값 설명

iCenterX, iCenterY 인자값은 각각 rotateImage(회전체 이미지)에서 회전시킬 회전체 중심점의 X좌표와 Y좌표를 의미합니다.



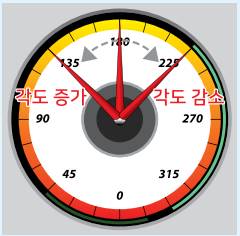
3. iOffsetX, iOffsetY 인자값 설명

iOffsetX, iOffsetY 인자값은 각각 rotateImage(회전체 이미지)가 SmartDraw의 좌측 상단(0, 0)을 기준으로 떨어진 위치(Offset)의 X좌표와 Y좌표를 의미합니다. 즉, Offset 값이 0, 0일 경우 rotateImage는 SmartDraw의 좌측 상단(0, 0)에 위치하게 됩니다. 이때 iOffsetX, iOffsetY 인자값을 조절하여 rotateImage를 backResultImage(배경 이미지)의 중심에 오도록 할 수 있습니다.



4. degrees 인자값 설명

degrees 인자값을 조절하여 rotateImage(회전체 이미지)를 중심점(iCenterX, iCenterY) 기준에서 원하는 각도만큼 회전시킬 수 있습니다.



C# 사용법

```
Bitmap imgRot = new Bitmap(Resource1._2);  // 회전 이미지 객체 생성
Bitmap imgBackResult = new Bitmap(Resource1.bg_2);  // 배경 이미지 객체 생성
// imgRotate.Width / 2는 회전 이미지의 중심 기준점 X 좌표
// imgRotate.Height / 2는 회전 이미지의 중심 기준점 Y 좌표
// iOffsetX : SmartDraw의 좌측 상단 기준에서 옵셋 만큼의 X 좌표값
// iOffsetY : SmartDraw의 좌측 상단 기준에서 옵셋 만큼의 Y 좌표값
// degrees : 회전 이미지를 회전시킬 각도
smartDraw1.RotationImage(imgRot, imgBackResult, imgRot.Width / 2, imgRot.Height / 2, 48, 48, 40);
// Drawing Layer에 회전 이미지가 적용된 이미지 그리기
smartDraw1.ImageDraw(imgBackResult, 0, 0);
```

VB 사용법

```
Dim imgRot As Bitmap = new Bitmap(Resource1._2)
Dim imgBackResult As Bitmap = new Bitmap(Resource1.bg_2)
smartDraw1.RotationImage(imgRot, imgBackResult, imgRot.Width / 2, imgRot.Height / 2, 48, 48, 40)
smartDraw1.ImageDraw(imgBackResult, 0, 0)
```

메소드(함수) RotationPoints

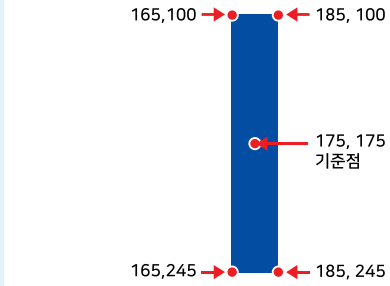
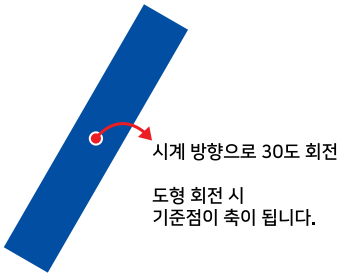
입력된 여러 개의 좌표를 기준점 중심으로 설정된 각도만큼 좌표들의 회전된 좌표를 리턴합니다. 사용자가 그린 회전체(선 또는 도형)는 기준점 centerPoint 좌표를 중심으로 회전합니다.

◉ SmartDraw.POINT[] RotationPoints(SmartDraw.POINT[] inPoints, SmartDraw.POINT centerPoint, double fAngle)

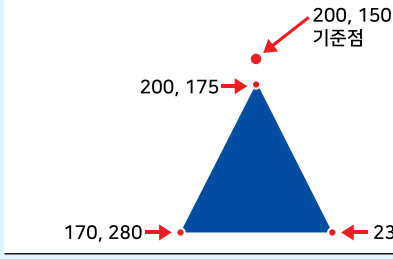
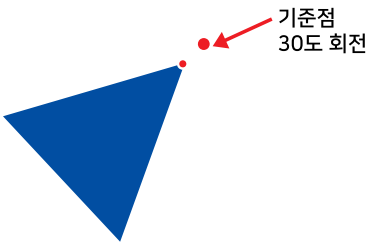
참고 기준점 위치에 따른 회전체의 변화

RotationPoints() 메소드는 사용자가 그린 도형을 회전시킨 좌표를 리턴하는 메소드로, 기준점에 따라 회전하는 결과가 달라지게 됩니다. 아래 표를 확인해보시기 바랍니다.

[표1] 도형 내부에 기준점이 있는 경우

회전시킬 도형	회전된 도형
	

[표2] 도형 외부에 기준점이 있는 경우

회전시킬 도형	회전된 도형
	

[인자]

- SmartDraw.POINT[] inPoints : 선 또는 도형 모양을 그릴수 있는 X, Y좌표가 저장된 배열값
- SmartDraw.POINT centerPoint : 선 또는 도형 모양의 중심축 X, Y좌표
- double fAngle : 선 또는 도형이 회전할 경우 변화되는 각도

[리턴값]

- SmartDraw.POINT[] : 입력 좌표의 회전된 좌표

C# 사용법

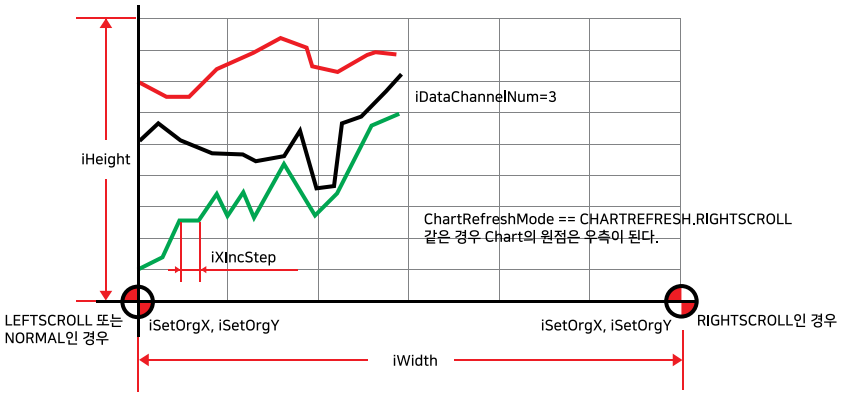
```
SmartDraw.POINT m_NeedleCenter;  // 선 또는 도형의 중심축(좌표) 저장
m_NeedleCenter.X = 175; m_NeedleCenter.Y = 175;
// 선 또는 도형의 좌표를 저장하는 배열 선언 및 생성
SmartDraw.POINT[] m_Needle = new SmartDraw.POINT[5];
// 선 또는 도형의 X, Y 좌표를 설정합니다.
m_Needle[0].X = 165; m_Needle[0].Y = 100; m_Needle[1].X = 185; m_Needle[1].Y = 100;
m_Needle[2].X = 185; m_Needle[2].Y = 245; m_Needle[3].X = 165; m_Needle[3].Y = 245;
m_Needle[4].X = 165; m_Needle[4].Y = 100;
// Drawing Layer에 배경 그리기
smartDraw1.SetBrushStyle(Color.Yellow); smartDraw1.Circle(175, 175, 10);
// Drawing Layer에 회전체 그리기
smartDraw1.SetBrushStyle(Color.Black);
// m_Needle를 사용하여 m_NeedleCenter를 기준으로 선을 90도 회전하는 기능 수행
smartDraw1.Polyline(smartDraw1.RotationPoints(m_Needle, m_NeedleCenter, 90));
```

VB 사용법

```
Dim m_NeedleCenter As SmartDraw.POINT
m_NeedleCenter.X = 175 : m_NeedleCenter.Y = 175
Dim m_Needle As SmartDraw.POINT() = new SmartDraw.POINT(4) {}
m_Needle(0).X = 165 : m_Needle(0).Y = 100 : m_Needle(1).X = 185 : m_Needle(1).Y = 100
m_Needle(2).X = 185 : m_Needle(2).Y = 245 : m_Needle(3).X = 165 : m_Needle(3).Y = 245
m_Needle(4).X = 165 : m_Needle(4).Y = 100
smartDraw1.SetBrushStyle(Color.Yellow) : smartDraw1.Circle(175, 175, 10)
smartDraw1.SetBrushStyle(Color.Black)
smartDraw1.Polyline(smartDraw1.RotationPoints(m_Needle, m_NeedleCenter, 90))
```

메소드(함수) SetChartCfg

SmartDraw의 기능 중 Chart를 그리기 위한 Chart 관련 파라미터를 설정하는 메소드입니다.



참고 ChartRefreshMode 인자값이 CHARTREFRESH.RIGHTSCROLL인 경우 iSetOrgX 인자값은 iWidth 인자값이 더해져 좌측 하단이 아닌 우측 하단이 원점이 됩니다.

◉ `void SetChartCfg(int iSetOrgX, int iSetOrgY, int iWidth, int iHeight, int iXIncStep, SmartDraw.CHARTREFRESH ChartRefreshMode, int iDataChannelNum)`

[인자]

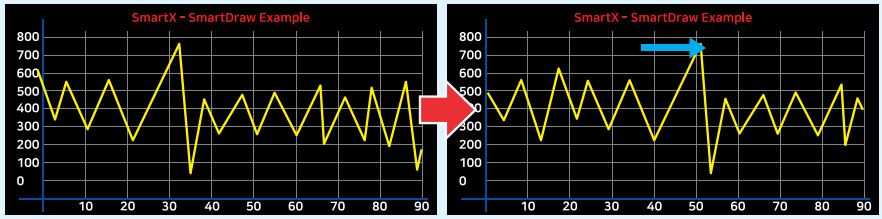
- `int iSetOrgX` : 차트의 X원점을 설정
- `int iSetOrgY` : 차트의 Y원점을 설정
- `int iWidth` : 차트의 폭을 설정
- `int iHeight` : 차트의 높이를 설정
- `int iXIncStep` : PutData() 메소드가 호출되어 한번에 증가되는 X좌표의 증가값 (권장값 : 2 이상)
- `SmartDraw.CHARTREFRESH ChartRefreshMode` : Chart를 그릴 때 화면의 갱신하는 방식 설정
  - `SmartDraw.CHARTREFRESH.LEFTSCROLL` : 처음 좌측부터 그려지며, Chart 데이터가 설정된 폭 이상 그려진 후 새로 Chart 데이터가 추가되면 좌측으로 그래프가 스크롤(이동)됨
  - `SmartDraw.CHARTREFRESH.NORMAL` : 처음 좌측부터 그려지며, Chart 데이터가 설정된 폭 이상 그려진 후 새로 Chart 데이터가 추가되면 차트가 모두 지워지며 다시 처음 좌측부터 그려짐
  - `SmartDraw.CHARTREFRESH.RIGHTSCROLL` : 처음 우측부터 그려지며, Chart의 데이터가 설정된 폭 이상 그려진 후 새로 Chart 데이터가 추가되면 우측으로 그래프가 스크롤(이동)됨
- `int iDataChannelNum` : Chart 데이터 채널의 수

참고 ChartRefreshMode 인자값에 따른 Chart 스크롤

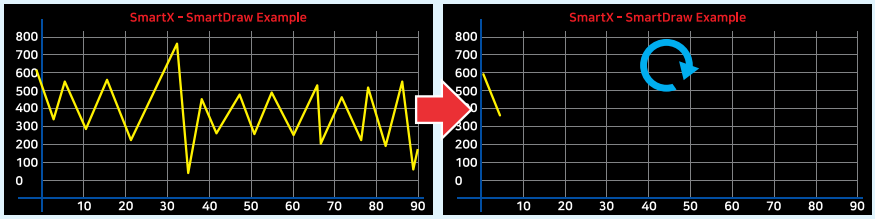
ChartRefreshMode 인자값으로 Chart가 설정된 폭(iWidth)을 넘어가는 경우 갱신 방법을 설정할 수 있습니다. 아래 표를 확인해보시기 바랍니다.

[표] ChartRefreshMode 인자값에 따른 Chart 갱신

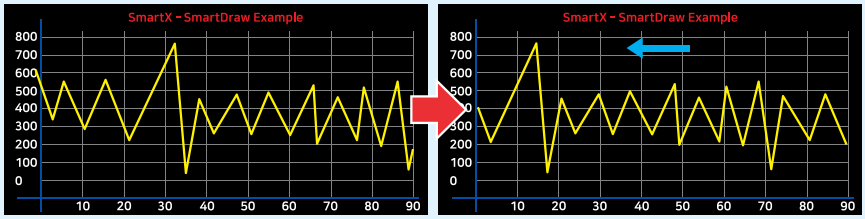
ChartRefreshMode	CHARTREFRESH.LEFTSCROLL
결과	그려진 차트는 왼쪽으로 스크롤되며 추가된 Chart 데이터는 이어서 그려짐



ChartRefreshMode	CHARTREFRESH.NORMAL
결과	그려진 Chart를 지우고 추가된 Chart 데이터는 좌측부터 다시 그려짐



ChartRefreshMode	CHARTREFRESH.RIGHTSCROLL
결과	그려진 차트는 오른쪽으로 스크롤되며 추가된 Chart 데이터는 이어서 그려짐



C# 사용법

```
// Chart의 원점 : (10, 260), 크기 : 630 X 270, X좌표 증가값 : 3, 채널 수 : 3개, 왼쪽 스크롤
smartDraw1.SetChartCfg(10, 260, 630, 270, 3, SmartDraw.CHARTREFRESH.LEFTSCROLL, 3);
```

VB 사용법

```
smartDraw1.SetChartCfg(60, 320, 630, 270, 3, SmartDraw.CHARTREFRESH.LEFTSCROLL, 3)
```

메소드(함수) PutData

Chart를 그리기 위해 Chart 데이터를 입력합니다. 입력 인자는 가변 인자이며 각각의 인자 순으로 채널1은 처음 인자 순으로 대입되어야 합니다. SetChartCfg() 메소드의 iDataChannelNum 인자값이 PutData() 메소드 인자의 개수와 같아야 합니다.

참고 "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

◉ `void PutData(params int[] ChDatas)`

[인자]

• `params int[] ChDatas` : Chart의 각 채널별 정수형 그림 요소 (가변 인자)

C# 사용법

```
// iDataChannelNum = 3인 경우 그래프의 Chart 데이터를 입력합니다.
smartDraw1.PutData(iVal1, iVal2, iVal3);  // Ch1 = iVal1, Ch2 = iVal2, Ch3 = iVal3
```

VB 사용법

```
smartDraw1.PutData(iVal1, iVal2, iVal3)
```

메소드(함수) PutDataAllClear

현재 진행 중인 내부의 모든 Chart 데이터를 삭제합니다. PutData() 메소드는 입력된 Chart 데이터를 내부적으로 일정 수를 저장하고 있습니다. PutDataAllClear() 메소드 호출 시 PutData() 메소드가 저장하고 있는 Chart 데이터를 모두 삭제합니다.

참고 "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

◉ `void PutDataAllClear()`

C# 사용법

```
// 입력된 Chart 데이터를 모두 삭제
smartDraw1.PutDataAllClear();
```

VB 사용법

```
smartDraw1.PutDataAllClear()
```

메소드(함수) ChartNowDraw

ChartDrawStep 속성값에 상관없이 즉시 Chart를 다시 그리게 합니다.

◉ `void ChartNowDraw()`

C# 사용법

```
// 내부에 저장되어 있는 Chart 데이터들로 Chart를 새로 그리게 합니다.
smartDraw1.ChartNowDraw();
```

VB 사용법

```
smartDraw1.ChartNowDraw()
```

메소드(함수) **Save**

화면(BackGround Layer + Drawing Layer)에 출력된 상태를 지정된 이미지 파일 형식으로 저장합니다. (BMP, JPG, GIF, PNG 형식 지원)

중요 Save() 메소드 호출 시 strFullPathName 인자값은 반드시 확장자를 포함해야 합니다.

◉ `void Save(string strFullPathName)`

[인자]

• `string strFullPathName` : 저장할 이미지 파일의 경로와 확장자

C# 사용법

```
smartDraw1.Save( " Flash Disk₩₩SmartDrawTest.PNG " );  // 현재 화면의 출력된 이미지 파일로 저장
```

VB 사용법

```
smartDraw1.Save( " Flash Disk₩₩SmartDrawTest.PNG " )
```

4) SmartDraw 예제 사용하기

SmartDraw를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

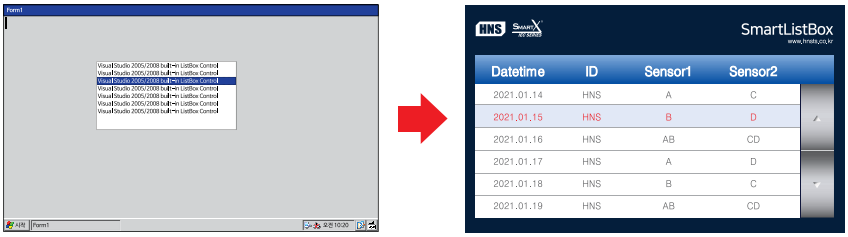
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartDraw", "SmartDraw2"

2. SmartListBox

SmartListBox는 .NET Compact Framework에서 지원되는 ListBox 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 ListBox의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

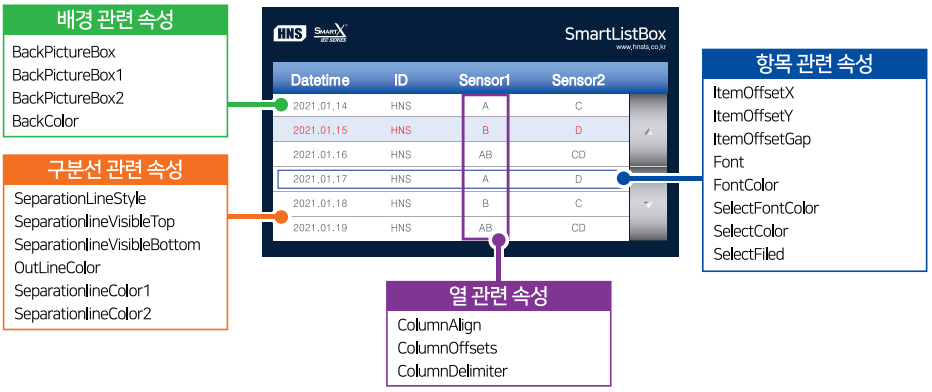
- 배경 투명 처리 효과 및 스킨 처리
- 리스트 터치 Move(Scroll) 처리
- 다양한 스타일을 적용할 수 있도록 색상 및 크기 변경 기능
- 항목 추가 방향 설정 기능
- 컬럼 구분 출력 기능



[기존 ListBox] [SmartListBox]

참조 "홈페이지 → 자료실 → Tech Note → 32. SmartListBox에서 컬럼 사용 및 정렬 방법 안내"을 참조하시기 바랍니다.

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

STEP-1 배경 설정하기

배경을 컬러로 설정하는 경우 BackColor 속성을 사용하여 설정하면 되며, 이미지로 설정하는 경우 BackPictureBox 관련 속성을 이용해 배경이 되는 컨트롤(PictureBox, SmartForm, SmartInnerForm, SmartGroupBox)의 이미지를 투영하여 배경을 설정할 수 있습니다. BackPictureBox 설정 시 BackColor는 적용되지 않습니다. 본 가이드는 Back PictureBox 적용을 기준으로 안내합니다.

※ 자세한 내용은 " BackColor, BackPictureBox 속성" 내용을 참고하시기 바랍니다.

[배경 컨트롤에 적용될 이미지]

STEP-2 구분선 설정하기

구분선의 경우 SeparationLineStyle, SeparatorlineVisibleTop, SeparatorlineVisibleBottom, SeparationlineColor1, SeparationlineColor2, OutLineColor 속성을 설정해 구분선을 다양하게 디자인할 수 있습니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
SeparationLineStyle	Fixed3D	SeparationlineColor1	DarkGray
SeparatorlineVisibleTop	False	SeparationlineColor2	Silver
SeparatorlineVisibleBottom	False	OutLineColor	Black

STEP-3 항목 설정하기

항목의 경우 ItemOffsetX, ItemOffsetY, ItemOffsetGap, Font, FontColor, SelectFontColor, SelectColor, SelectFiled 속성을 설정해 항목을 다양하게 디자인할 수 있습니다.

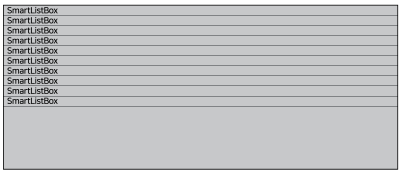
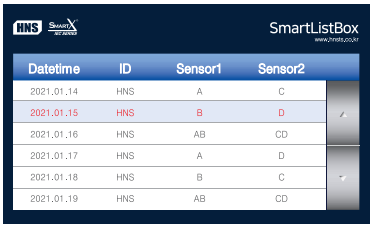
[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
ItemOffsetX	0	FontColor	Black
ItemOffsetY	0	SelectFontColor	DarkRed
ItemOffsetGap	20	SelectColor	LightGray
Font	Arial, 14pt, style=Bold	SelectFiled	True

[표] 관련 속성 설명

디자인 요소	관련 속성
배경	BackColor, BackPictureBox
구분선	SeparationLineStyle, SeparatorlineVisibleTop, SeparatorlineVisibleBottom, SeparationlineColor1, SeparationlineColor2, OutLineColor
항목	ItemOffsetX, ItemOffsetY, ItemOffsetGap, Font, FontColor, SelectFontColor, SelectColor, SelectFiled

※ 자세한 내용은 위 표의 속성 내용을 참고하시기 바랍니다.

디자인 및 배경 적용 전	디자인 및 배경 적용 후
	

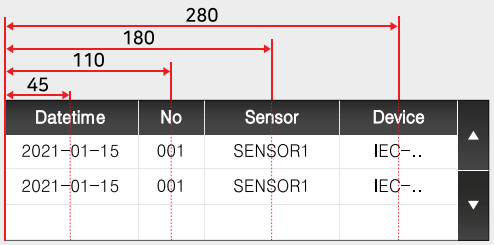
STEP-4 컬럼 설정하기

SmartListBox는 컬럼을 구분하여 항목을 표시할 수 있습니다. ColumnAlign 속성에서 정렬 기준을, ColumnOffsets 속성에서 컬럼의 거리를, ColumnDelimiter 속성에서 컬럼 구분자를 설정할 수 있습니다. 본 가이드는 중앙으로 정렬하고 컬럼의 수가 4개인 예제를 설명합니다.

※ 자세한 내용은 "ColumnAlign, ColumnOffsets, ColumnDelimiter 속성" 내용을 참고하시기 바랍니다.

```
// 컬럼기준 중앙 정렬. 구분자로 구분된 데이터들을 중앙 정렬 시 사용
smartListBox1.ColumnAlign = SmartX.SmartListBox.COLUMNALIGNS.CENTER;
// 컬럼의 Position을 배열로 저장. 여기서 사용할 컬럼 개수가 4개이므로 배열 크기는 4
int[] iColPos = new int[4];
// 각 컬럼의 Position을 초기화 (기준점 : SmartListBox의 가장 좌측)
iColPos[0] = 45;   // 기준점에서 첫번째 컬럼의 중앙점까지의 거리
iColPos[1] = 110;  // 기준점에서 두번째 컬럼의 중앙점까지의 거리
iColPos[2] = 180;  // 기준점에서 세번째 컬럼의 중앙점까지의 거리
iColPos[3] = 280;  // 기준점에서 네번째 컬럼의 중앙점까지의 거리
smartListBox1.ColumnOffsets = iColPos;  // ColumnOffsets에 컬럼의 Position 값을 대입
smartListBox1.ColumnDelimiter = ';';  // ColumnDelimiter 를 ';'로 설정
// 각 데이터 사이에 ';'를 삽입하여 컬럼을 구분해 데이터를 추가
smartListBox1.AddItem("2021-01-15;001;SENSOR1;IEC-..");
```

[출력 결과]



3) SmartListBox 인터페이스 설명

SmartListBox Component Interface		
속성		
BackColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
BackPictureBox2 : SmartGroupBox	ColumnAlign : SmartListBox.COLUMNALIGNS	ColumnDelimiter : char
ColumnOffsets : int[]	Font : Font	FontColor : Color
InitVisible : bool	ItemAddOrder : SmartListBox.ITEMADDORDERS	ItemCount : int
ItemOffsetGap : int	ItemOffsetX : int	ItemOffsetY : int
Items : List<string>	MouseMoveEventSpace : int	MouseMoveSpace : int
OutLineColor : Color	SelectColor : Color	SelectFilled : bool
SelectFontColor : Color	SelectItemIndex : int	SeperationlineColor1 : Color
SeperationlineColor2 : Color	SeperationLineStyle: SmartListBox.SEPARATIONLINETYPES	SeperationlineVisibleBottom : bool
SeperationlineVisibleTop : bool	ViewRemainCount : int	ViewStartItemIndex : int
메소드		
AddItem() : void (+1개 오버로드)	ClearAll() : void	ModifyItem(int iItemIndex, string strNewValue) : void
RemoveItem(int iItemIndex) : void	ScrollDown : void (+1개 오버로드)	ScrollUp : void (+1개 오버로드)
이벤트		
OnUpdateEvent : EventHandler	SelectedIndexChanged : EventHandler	

프로퍼티(속성) BackColor

SmartListBox의 배경 색상을 설정합니다.

• Color : SmartListBox의 배경 색상

C# 사용법

```
smartListBox1.BackColor = Color.Blue;  // SmartListBox의 배경 색상을 파란색으로 설정
```

VB 사용법

```
smartListBox1.BackColor = Color.Blue
```

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartListBox의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartListBox에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartListBox의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) ColumnAlign, ColumnOffsets, ColumnDelimiter

smartListBox의 컬럼별로 구분하여 정렬하기 위해 사용됩니다. 각 속성들은 항상 함께 사용해야 합니다.

• ColumnAlign : SmartListBox 아이템의 정렬 방법(Left, Center, Right)을 설정합니다.

• int[] ColumnOffsets : 각 컬럼에 정렬될 항목의 기준선 위치를 설정합니다. 기준선(정렬)의 위치는 ColumnAlign 속성에 따라 달라집니다.

• char ColumnDelimiter : smartListBox1.AddItem()을 통해 SmartListBox에 입력된 문자열을 컬럼별로 구분하기 위한 구분자를 설정합니다.

주의
1. 각 정렬 방법별로 시작 Position이 다르므로 유의 바랍니다. Left는 컬럼의 좌측, Center는 컬럼의 중앙, Right는 컬럼의 우측이 시작 Position이 됩니다.
2. 한번 선택한 정렬은 중간에 변경하지 못하며, Form_Load에서 최초 한 번만 설정합니다.

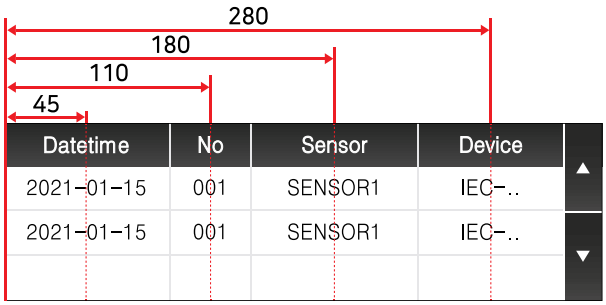
권장 ColumnDelimiter로 지정 가능한 문자는 SmartListBox에 추가되는 실제 데이터와 중복되지 않는 특수 문자의 사용을 권장합니다.

- SmartListBox.COLUMNALIGNS.CENTER : 각 컬럼의 중앙을 기준으로 아이템을 정렬
- SmartListBox.COLUMNALIGNS.LEFT : 각 컬럼의 좌측을 기준으로 아이템을 정렬
- SmartListBox.COLUMNALIGNS.RIGHT : 각 컬럼의 우측을 기준으로 아이템을 정렬

[표] ColumnAlign 및 ColumnOffsets, ColumnDelimiter 속성값에 따른 컬럼 정렬

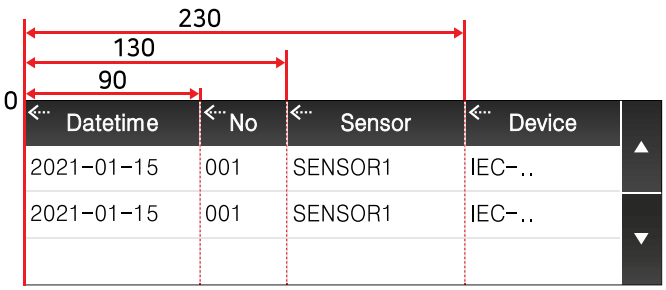
NO-1	CENTER

코드	`// 컬럼 기준 중앙 정렬. 구분자로 구분된 데이터들을 중앙 정렬 시 사용`
	`smartListBox1.ColumnAlign = SmartX.SmartListBox.COLUMNALIGNS.CENTER;`
	`// 컬럼의 Position을 배열로 저장. 여기서 사용할 컬럼 개수가 4개이므로 배열 크기는 4`
	`int[] iColPos = new int[4];`
	`// 각 컬럼의 Position을 초기화 (기준점 : SmartListBox의 가장 좌측)`
	`iColPos[0] = 45;  // 기준점에서 첫번째 컬럼의 중앙점까지의 거리`
	`iColPos[1] = 110;  // 기준점에서 두번째 컬럼의 중앙점까지의 거리`
	`iColPos[2] = 180;  // 기준점에서 세번째 컬럼의 중앙점까지의 거리`
	`iColPos[3] = 280;  // 기준점에서 네번째 컬럼의 중앙점까지의 거리`
	`smartListBox1.ColumnOffsets = iColPos;  // ColumnOffsets에 컬럼의 Position 값을 대입`
	`smartListBox1.ColumnDelimiter = ';';  // ColumnDelimiter를 ';'로 설정`
	`// 각 데이터 사이에 ';'를 삽입하여 컬럼을 구분해 데이터를 추가`
	`smartListBox1.AddItem("2021-01-15;001;SENSOR1;IEC-..");`

출력 결과	

NO-2	LEFT

코드	`// 컬럼 기준 좌측 정렬. 구분자로 구분된 데이터들을 좌측 정렬 시 사용`
	`smartListBox1.ColumnAlign = SmartX.SmartListBox.COLUMNALIGNS.LEFT;`
	`// 컬럼의 Position을 배열로 저장. 여기서 사용할 컬럼 개수가 4개이므로 배열 크기는 4`
	`int[] iColPos = new int[4];`
	`// 각 컬럼의 Position을 초기화 (기준점 : SmartListBox의 가장 좌측)`
	`iColPos[0] = 0;  // 기준점에서 첫번째 컬럼의 시작점까지의 거리`
	`iColPos[1] = 90;  // 기준점에서 두번째 컬럼의 시작점까지의 거리`
	`iColPos[2] = 130;  // 기준점에서 세번째 컬럼의 시작점까지의 거리`
	`iColPos[3] = 230;  // 기준점에서 네번째 컬럼의 시작점까지의 거리`
	`smartListBox1.ColumnOffsets = iColPos;  // ColumnOffsets에 컬럼의 Position 값을 대입`
	`smartListBox1.ColumnDelimiter = ';';  // ColumnDelimiter를 ';'로 설정`
	`// 각 데이터 사이에 ';'를 삽입하여 컬럼을 구분해 데이터를 추가`
	`smartListBox1.AddItem("2021-01-15;001;SENSOR1;IEC-..");`

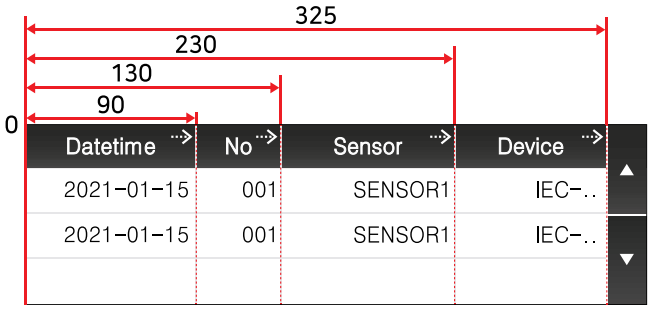
출력 결과	

NO-3	RIGHT

코드

```
// 컬럼 기준 우측 정렬. 구분자로 구분된 데이터들을 우측 정렬 시 사용
smartListBox1.ColumnAlign = SmartX.SmartListBox.COLUMNALIGNS.RIGHT;
// 컬럼의 Position을 배열로 저장. 여기서 사용할 컬럼 개수가 4개이므로 배열 크기는 4
int[] iColPos = new int[4];
// 각 컬럼의 Position을 초기화 (기준점 : SmartListBox의 가장 좌측)
iColPos[0] = 90;   // 기준점에서 첫번째 컬럼의 끝점까지의 거리
iColPos[1] = 130;  // 기준점에서 두번째 컬럼의 끝점까지의 거리
iColPos[2] = 230;  // 기준점에서 세번째 컬럼의 끝점까지의 거리
iColPos[3] = 325;  // 기준점에서 네번째 컬럼의 끝점까지의 거리
smartListBox1.ColumnOffsets = iColPos;  // ColumnOffsets에 컬럼의 Position 값을 대입
smartListBox1.ColumnDelimiter = ';';  // ColumnDelimiter를 ';'로 설정
// 각 데이터 사이에 ';'를 삽입하여 컬럼을 구분해 데이터를 추가한다.
smartListBox1.AddItem("2021-01-15;001;SENSOR1;IEC-..");
```

출력 결과



C# 사용법

```
// 컬럼 기준 중앙 정렬
smartListBox1.ColumnAlign = SmartListBox.COLUMNALIGNS.CENTER;
// 컬럼의 Position 설정
int[] iColPos = new int[4];
iColPos[0] = 45;
iColPos[1] = 110;
iColPos[2] = 180;
iColPos[3] = 280;
smartListBox1.ColumnOffsets = iColPos;
// 각 컬럼을 구분할 구분자
smartListBox1.ColumnDelimiter = ';';
// 데이터 추가
smartListBox1.AddItem("2021-01-15;001;SENSOR1;IEC-..");
```

VB 사용법

```
smartListBox1.ColumnAlign = SmartListBox.COLUMNALIGNS.CENTER
Dim iColPos() As Integer = New Integer(4) {}
iColPos[0] = 45
iColPos[1] = 110
iColPos[2] = 180
iColPos[3] = 280
smartListBox1.ColumnOffsets = iColPos
smartListBox1.ColumnDelimiter = ';'
smartListBox1.AddItem("2021-01-15;001;SENSOR1;IEC-..")
```

프로퍼티(속성) Font

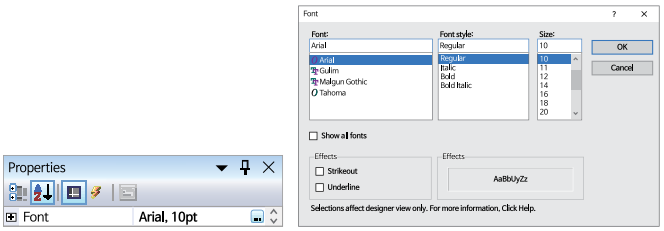
SmartListBox에 표시되는 폰트를 설정합니다.

Font

2021.01.14	HNS	A	C
2021.01.15	HNS	B	D
2021.01.16	HNS	AB	CD
2021.01.17	HNS	A	D
2021.01.18	HNS	B	C
2021.01.19	HNS	AB	CD

• Font : 표시될 항목의 폰트

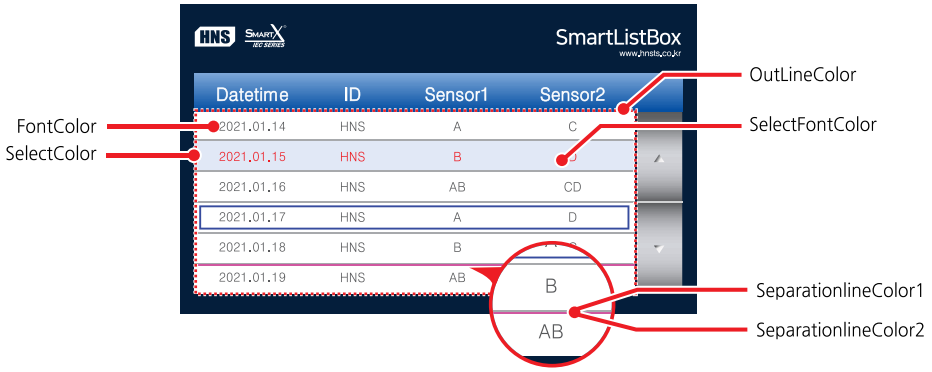
사용법



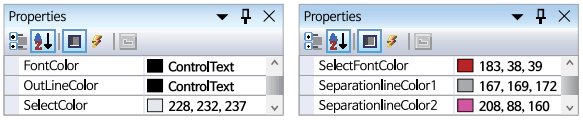
프로퍼티(속성) FontColor, OutLineColor, SelectColor, SelectFontColor, SeparationlineColor1, SeparationlineColor2

SmartListBox 각 요소의 색상을 변경합니다.

• FontColor : 리스트 항목의 글자 색상을 설정합니다.
• OutLineColor : SmartListBox 외곽선 색상을 설정합니다.
• SelectColor : 선택된 항목의 색상을 설정합니다.
• SelectFontColor : 선택된 항목의 글자 색상을 설정합니다.
• SeparationlineColor1 : 항목의 구분선(첫 번째 선)의 색상을 설정합니다.
• SeparationlineColor2 : SeparationLineStyle 속성값이 Fixed3D인 경우 항목의 두번째 구분선의 색상을 설정합니다.



사용법

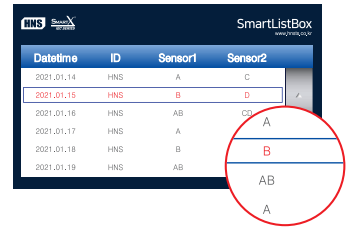
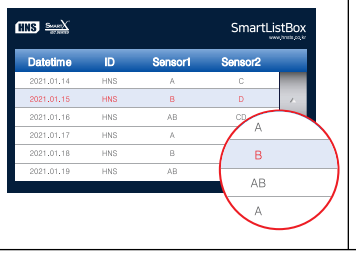
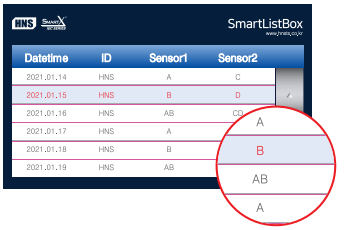


프로퍼티(속성) SeparationLineStyle

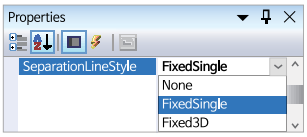
SmartListBox의 항목 구분선 Style을 설정합니다.

- SmartListBox.SEPARATIONLINETYPES.None : 항목 구분선 없음
- SmartListBox.SEPARATIONLINETYPES.FixedSingle : 2차원 선
- SmartListBox.SEPARATIONLINETYPES.Fixed3D : 3차원 선

[표] SeparationLineStyle 속성값에 따른 구분선 Style

None	FixedSingle	Fixed3D
		

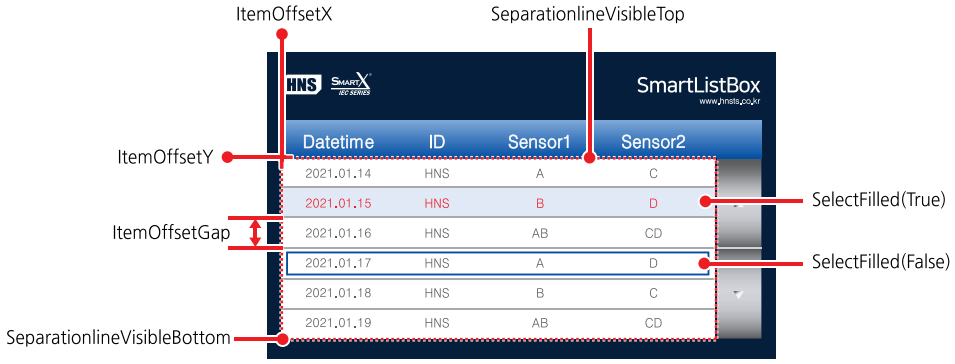
사용법



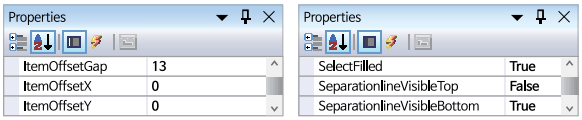
프로퍼티(속성) ItemOffsetGap, ItemOffsetX, ItemOffsetY, SelectFilled, SeparationlineVisibleTop, SeparationlineVisibleBottom

SmartListBox 각 요소의 표시되는 위치 및 형태를 변경합니다.

- ItemOffsetGap : 각 항목의 높이를 변경합니다.
- ItemOffsetX : 항목이 표시되는 위치의 가로 Offset 값을 설정합니다.
- ItemOffsetY : 항목이 표시되는 위치의 세로 Offset 값을 설정합니다.
- SelectFilled : 선택된 항목의 색 채움 여부를 설정합니다. (True : 채움, False : 비움)
- SeparationlineVisibleTop : 시작 항목의 상단 구분선 표시 여부를 설정합니다. (True : 표시, False : 숨김)
- SeparationlineVisibleBottom : 끝 항목의 하단 구분선 표시 여부를 설정합니다. (True : 표시, False : 숨김)



사용법

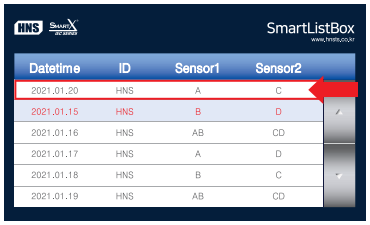
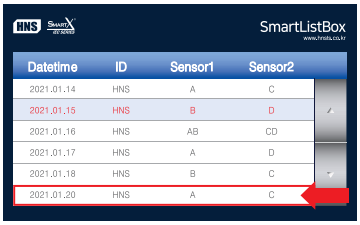


프로퍼티(속성) ItemAddOrder

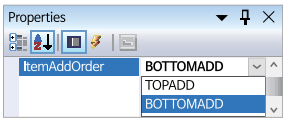
리스트에 항목이 추가되는 위치(방향)를 설정합니다.

- SmartListBox.ITEMADDORDERS.TOPADD : 리스트의 첫번째에 항목을 새로 추가
- SmartListBox.ITEMADDORDERS.BOTTOMADD : 리스트의 마지막에 항목을 새로 추가

[표] ItemAddOrder 속성값에 따른 아이템 추가 위치

TOPADD	BOTTOMADD
	
새로 추가한 항목이 항상 상단에 추가되어 추가된 항목을 항상 식별 가능함	화면에 표현 가능한 항목보다 전체 항목의 수가 많을 경우 새로 추가한 항목을 볼 수 없음

사용법



프로퍼티(속성) ItemCount

SmartListBox에 추가된 항목의 수를 얻습니다.

- int : 현재 리스트에 추가된 항목의 수 (없을 경우 -1)

C# 사용법

```
// 현재 리스트에 추가된 항목의 수를 저장
int iCount = smartListBox1.ItemCount;
```

VB 사용법

```
Dim iCount As Integer = smartListBox1.ItemCount
```

프로퍼티(속성) Items

SmartListBox에 추가된 항목을 얻습니다.

- List<string> : 현재 리스트에 추가된 항목의 리스트

C# 사용법

```
// 리스트의 첫번째 항목을 얻습니다
string strColumnData = smartListBox1.Items[1];
```

VB 사용법

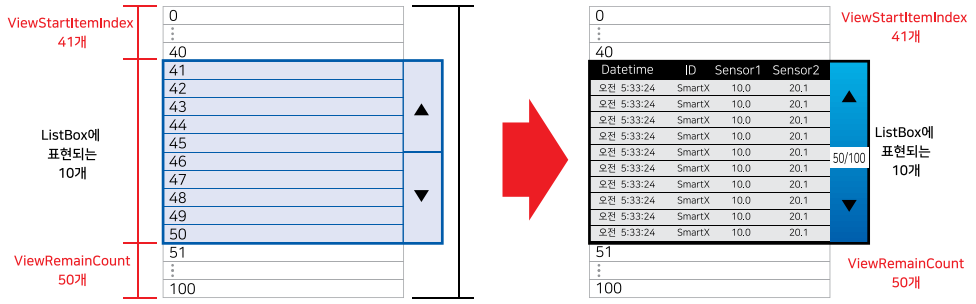
```
Dim strColumnData As String = smartListBox1.Items(1)
```

프로퍼티(속성) ViewStartItemIndex, ViewRemainCount

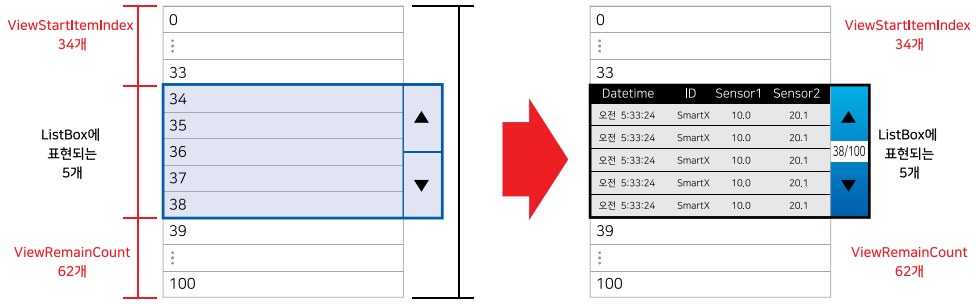
- ViewStartItemIndex : 화면에 보여지는 리스트의 첫 항목(최상단)의 Index를 얻습니다.
- ViewRemainCount : 화면에 보여지지 않는 리스트 아래쪽에 남아있는 항목의 수를 얻습니다.

참고 리스트의 터치 이동(Scroll) 처리에 따라서 값의 오차가 생길 수 있습니다.

100개의 데이터 ListBox 10개씩 출력 경우



100개의 데이터 ListBox 5개씩 출력 경우



• int : 화면에 보여지는 최상단 항목의 Index 또는 화면에 보이지 않는 아래쪽 항목의 수

C# 사용법

```
int iViewStartIndex = smartListBox1.ViewStartItemIndex;  // 현재 보여지는 최상단 항목의 Index 저장
int iViewRemainCnt = smartListBox1.ViewRemainCount;  // 화면에 보이지 않는 아래쪽 항목의 수를 저장
```

VB 사용법

```
Dim iViewStartIndex As Integer = smartListBox1.ViewStartItemIndex
Dim iViewRemainCnt As Integer = smartListBox1.ViewRemainCount
```

프로퍼티(속성) SelectItemIndex

SmartListBox에서 선택된(클릭한) 항목의 Index를 얻거나 설정한 Index의 항목을 선택합니다.

• int : 현재 리스트에서 선택한 항목의 Index를 얻거나 항목을 선택하기 위해 설정할 Index (선택된 항목이 없거나 설정하지 않았을 경우 -1)

사용법

참고 "SelectedIndexChanged 이벤트" 내용을 참고하시기 바랍니다.

프로퍼티(속성) MouseMoveEventSpace, MouseMoveSpace

터치 이동 처리 시 리스트의 이동(Scroll)과 관련된 속성값을 설정합니다.

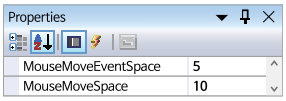
• MouseMoveEventSpace : 터치 이동 이벤트가 발생되는 이동 거리를 설정합니다. 값이 커지면 터치 이동값이 커져야 되므로 감도가 떨어집니다.

• MouseMoveSpace : 터치 이동 이벤트가 발생되고 리스트가 이동(Scroll)되는 거리를 설정합니다. 값이 커질수록 터치 이동 시 리스트의 이동거리가 커집니다.

중요 본 기능은 SystemConfig(바탕화면)에서 Touch Option의 Drag and Drop 항목을 Enable 처리하셔야 동작되는 기능으로 제품의 O/S 버전에 따라 지원이 안 될 수 있습니다.

참조 Drag & Drop 관련하여 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → Drag & Drop 관련"을 참조하시기 바랍니다.

사용법



메소드(함수) AddItem

리스트에 항목을 추가합니다.

참조 리스트에 한번에 많은 항목을 추가하는 경우 bUpdateFlag 인자값을 False로 하시면 빠른 추가가 가능합니다.
자세한 내용은 "홈페이지 → 자료실 → Tech Note → 72. 즉시 처리 방식과 일괄 처리 방식의 비교 및 관련된 SmartX Framework 컴포넌트 안내" 내용을 참조하시기 바랍니다.

중요 bUpdateFlag 인자값을 False로 하여 항목을 추가한 경우 반드시 Refresh() 메소드를 호출하여 화면을 갱신하시기 바랍니다.

◉ `void AddItem(string strItem)`
◉ `void AddItem(string strItem, bool bUpdateFlag)`

[인자]

- `string strItem` : 추가할 항목
- `bool bUpdateFlag` : 추가된 항목의 리스트박스에서 표시를 갱신할 것인지 설정
  - `true` : 리스트에 항목 추가 직후 화면을 갱신함 (기본값)
  - `false` : 리스트에 항목을 추가해도 화면을 갱신하지 않음

C# 사용법

```
smartListBox1.AddItem( " ItemAdd " );
```

VB 사용법

```
smartListBox1.AddItem( " ItemAdd " )
```

메소드(함수) ModifyItem

리스트에 추가된 특정 항목의 값(문자열)을 변경합니다.

◉ `void ModifyItem(int iItemIndex, string strNewValue)`

[인자]

- `int iItemIndex` : 변경할 항목의 Index
- `string strNewValue` : 변경될 값(새로운 문자열의 값)

C# 사용법

```
if (smartListBox1.SelectItemIndex != -1)
{
  smartListBox1.ModifyItem(smartListBox1.SelectItemIndex, " Item Modified " );
}
```

VB 사용법

```
If smartListBox1.SelectItemIndex <> -1 Then
 smartListBox1.ModifyItem(smartListBox1.SelectItemIndex, " Item Modified " )
```

```
End If
```

메소드(함수) RemoveItem, ClearAll

- RemoveItem() : 특정 항목을 삭제합니다.
- ClearAll() : 전체 항목을 삭제합니다.

◉ void RemoveItem(int iItemIndex)
◉ void ClearAll()

[인자]

- int iItemIndex : 삭제할 항목의 Index

C# 사용법

```
smartListBox1.RemoveItem(3);  // Index 3인 항목을 제거
smartListBox2.ClearAll();
```

VB 사용법

```
smartListBox1.RemoveItem(3)
smartListBox2.ClearAll()
```

메소드(함수) ScrollDown, ScrollUp

- ScrollDown() : 항목들을 리스트 아래쪽으로(하나의 항목 단위) 스크롤 합니다. (기본 STEP:1)
- ScrollUp() : 항목들을 리스트 위쪽으로(하나의 항목 단위) 스크롤 합니다. (기본 STEP:1)

◉ void ScrollDown()
◉ void ScrollDown(int iStep)
◉ void ScrollUp()
◉ void ScrollUp(int iStep)

[인자]

- int iStep : 이동할 거리(단위 : Pixel)

C# 사용법

```
smartListBox1.ScrollUp();    // 하나의 항목 단위로 위로 이동(Scroll)합니다.
smartListBox1.ScrollDown();  // 하나의 항목 단위로 아래로 이동(Scroll)합니다.
```

VB 사용법

```
smartListBox1.ScrollUp()
smartListBox1.ScrollDown()
```

이벤트 OnUpdateEvent

리스트 항목의 변화(추가, 갱신, 삭제, 리스트의 이동 등)가 생길 경우 발생되는 이벤트입니다.

C# 사용법

```
private void smartListBox1_OnUpdateEvent(object sender, EventArgs e)
{
  int iTemp = smartListBox1.ViewRemainCount;
  labView.Text = iTemp.ToString() + " / " + smartListBox1.ItemCount.ToString();
}
```

VB 사용법

```
Private Sub smartListBox1_OnUpdateEvent(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartListBox1.OnUpdateEvent
Dim iTemp As Integer = smartListBox1.ViewRemainCount
```

```
  labView.Text = iTemp.ToString() + " / " + smartListBox1.ItemCount.ToString()
End Sub
```

이벤트 | **SelectedIndexChanged**

SmartListBox Item을 클릭하여 SelectItemIndex 속성값이 변하거나, SelectItemIndex 속성값을 변경하는 경우 발생하는 이벤트입니다.

참고 "SelectItemIndex 속성" 내용을 참고하시기 바랍니다.

C# 사용법

```
private void smartListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  if (smartListBox1.SelectItemIndex != -1)
  {
    // 현재 선택된 Index 표시
    smartLabel2.Text = smartListBox1.SelectItemIndex.ToString();
  }
}
```

VB 사용법

```
Private Sub smartListBox1_SelectedIndexChanged(sender As Object, ByVal e As EventArgs)
  If smartListBox1.SelectItemIndex <> -1 Then
    smartLabel2.Text = smartListBox1.SelectItemIndex.ToString()
  End If
End Sub
```

4) SmartListBox 예제 사용하기

SmartListBox를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

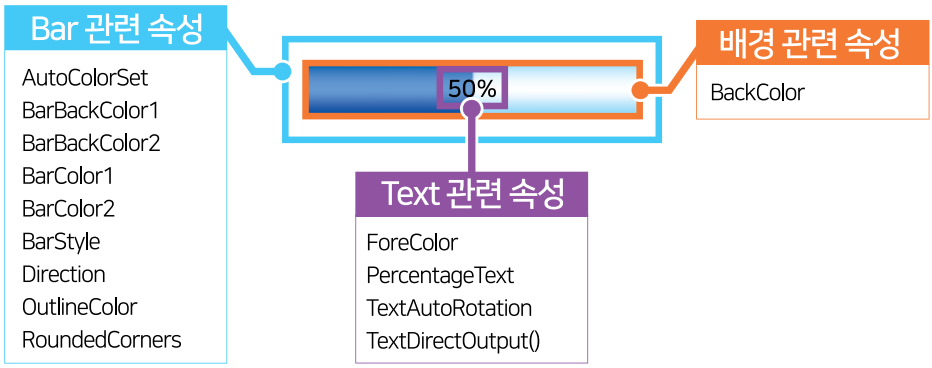
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartListBox"

3. SmartProgressBar

SmartProgressBar는 .NET Compact Framework에서 지원되는 ProgressBar 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 ProgressBar의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 가로/세로 방향의 ProgressBar 지원
- 다양한 스타일을 적용할 수 있도록 디자인 요소 변경 기능
- Percent(%) Text 표시 지원
- 값 변동 시 깜빡거림 최소화

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

STEP-1 배경 속성 설정하기

SmartProgressBar의 배경 색상을 반드시 BackColor 속성을 사용해 설정합니다. 만약 BackColor 설정이 안 되어있을 경우 깨짐 현상이 발생합니다. 색상은 최대한 배경 색상과 유사한 색상으로 설정하시기 바랍니다.
(단, RGB값으로 설정해야 하며, Web.Control값은 적용이 안됩니다.)

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackColor	Silver

※ 자세한 내용은 "BackColor 속성" 내용을 참고하시기 바랍니다.

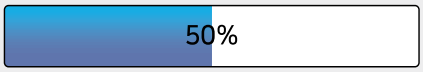
STEP-2 Bar 관련 속성 설정하기

BarStyle 속성으로 스타일 설정 후 Direction 속성으로 Bar의 방향을 설정합니다. Bar의 배경 색상은 BarBackColor1, BarBackColor2 속성으로 설정하며, Bar Gage 색상은 BarColor1, BarColor2 속성으로 설정합니다. 또한, AutoColor Set 속성값을 Ture로 설정하여 BarColor1 속성만 바꾼다면 다른 속성값을 자동으로 변경되도록 설정할 수 있습니다. OutlineColor, RoundedCorners 속성으로 Bar의 테두리 색상 및 모서리 부분의 Round 처리를 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
AutoColorSet	False	BarStyle	Normal1
BarBackColor1	AliceBlue	Direction	Horizontal
BarBackColor2	White	OutlineColor	Black
BarColor1	DodgerBlue	RoundedCorners	True
BarColor2	CornflowerBlue	-	-

※ 자세한 내용은 "AutoColorSet, BarBackColor1, BarBackColor2, BarColor1, BarColor2, BarStyle, Direction, OutlineColor, RoundedCorners 속성" 내용을 참고하시기 바랍니다.



[그림] 적용된 모습

STEP-3 Text 관련 속성 설정하기

PercentageText 속성값을 True로 하여 Text를 표시하며, Direction 속성값이 Vertical인 경우 TextAutoRotation 속성으로 Text의 방향을 자동으로 세로 방향으로 변경할 수 있습니다. ForeColor 속성으로 Text의 색상을 설정하며, TextDirectOutput() 메소드를 호출해 Percentage(%)가 아닌 사용자 지정 Text를 출력할 수 있습니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
PercentageText	True	ForeColor	Black
TextAutoRotation	True	-	-

※ 자세한 내용은 "PercentageText, TextAutoRotation, ForeColor 속성"과 "TextDirectOutput 메소드" 내용을 참고하시기 바랍니다.

```
// PercentageText 속성값이 True일 때, 출력될 값을 직접 입력
smartProgressBar1.TextDirectOutput( " 진행 중… " );
```

[그림] 적용된 모습

3) SmartProgressBar 인터페이스 설명

SmartProgressBar Component Interface		
속성		
AutoColorSet : bool	BackColor : Color	BarBackColor1 : Color
BarBackColor2 : Color	BarColor1 : Color	BarColor2 : Color
BarStyle : SmartProgressBar.BARTYPE	Direction : SmartProgressBar.DIR	ForeColor : Color
InitVisible : bool	Maximum : int	Minimum : int
OutlineColor : Color	PercentageText : bool	RoundedCorners : bool
TextAutoRotation : bool	Value : int	
메소드		
TextDirectOutput(string strMsg) : void		

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을

방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartProgressBar의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

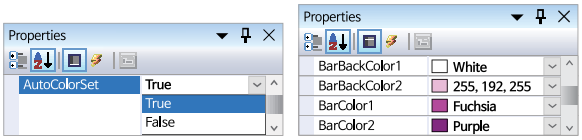
참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) AutoColorSet

SmartProgressBar 관련 색상 속성의 Color 값을 자동으로 변경되도록 설정합니다. 즉, BarColor1 값을 Custom 팔레트에서 선택해 변경하면 다른 속성(BarColor2, BarBackColor1, BarBackColor2)의 Color 값 또한 명도차가 있는 색상으로 자동 변경됩니다.

- `bool` : BarColor1 값이 변경되면 자동으로 다른 속성(BarColor2, BarBackColor1, BarBackColor2)도 변경
  - `true` : 자동으로 변경함
  - `false` : 자동으로 변경 안 함

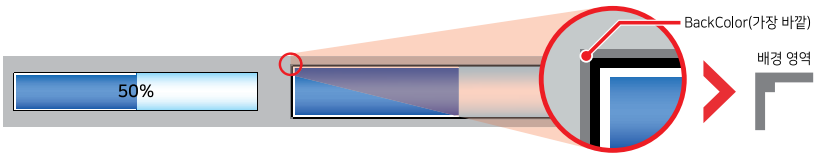
사용법



프로퍼티(속성) BackColor

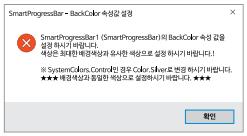
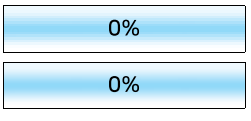
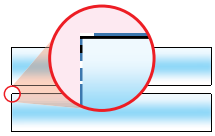
SmartProgressBar의 다음과 같이 해당되는 영역의 색상 설정합니다.

중요 반드시 BackColor 속성을 설정하시기 바랍니다.



주의 BackColor 설정 시 주의사항

BackColor 설정이 안 되어있을 경우 Visual Studio에서 폼을 열 때 마다 아래와 같이 알림창이 뜨며, 프로젝트 실행 시 깨짐 현상이 발생할 수 있습니다. 따라서 반드시 BackColor 속성을 설정하시기 바라며, BackColor 색상은 최대한 배경 색상과 유사한 색상으로 설정하시기 바랍니다. (단, Web.Control 값은 적용이 안됩니다.) 만약 사용되는 색상이 SystemColors.Control인 경우 Color.Silver로 변경하시기 바랍니다.

BackColor 속성값 설정 알림창	BackColor 적용	BackColor 미적용
		

C# 사용법

```
smartProgressBar1.BackColor = Color.Gray;
```

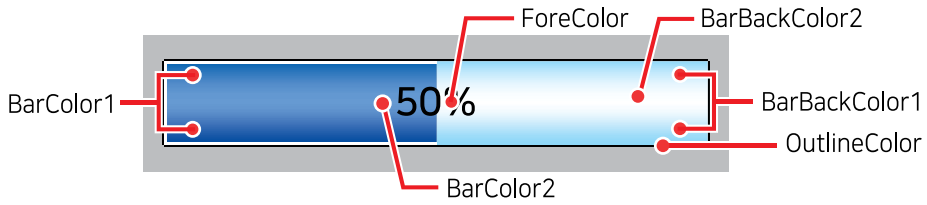
VB 사용법

```
smartProgressBar1.BackColor = Color.Gray
```

프로퍼티(속성) BarBackColor1, BarBackColor2, BarColor1, BarColor2, ForeColor, OutlineColor

SmartProgressBar 관련 색상 등을 설정합니다.

- BarBackColor1 : Bar의 배경 색상
- BarBackColor2 : Bar의 배경 색상 (BarStyle이 그라데이션일 경우 그라데이션의 두 번째 색상)
- BarColor1 : Bar Gage의 색상
- BarColor2 : Bar Gage의 색상 (BarStyle이 그라데이션일 경우 그라데이션의 두 번째 색상)
- ForeColor : SmartProgressBar의 Text 색상
- OutlineColor : SmartProgressBar의 프레임 색상



C# 사용법

```
smartProgressBar1.BarBackColor1 = Color.White;
smartProgressBar1.BarBackColor2 = Color.SkyBlue;
smartProgressBar1.BarColor1 = Color.Blue;
smartProgressBar1.BarColor2 = Color.SkyBlue;
smartProgressBar1.ForeColor = Color.Black;
smartProgressBar1.OutlineColor = Color.Black;
```

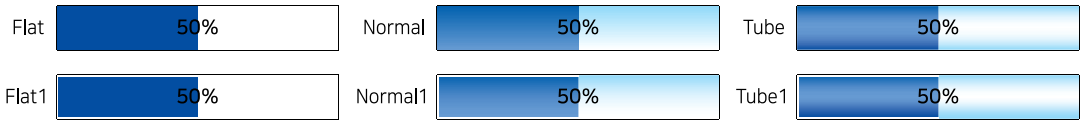
VB 사용법

```
smartProgressBar1.BarBackColor1 = Color.White
smartProgressBar1.BarBackColor2 = Color.SkyBlue
smartProgressBar1.BarColor1 = Color.Blue
smartProgressBar1.BarColor2 = Color.SkyBlue
smartProgressBar1.ForeColor = Color.Black
smartProgressBar1.OutlineColor = Color.Black
```

프로퍼티(속성) BarStyle

SmartProgressBar의 Bar 스타일(Flat, Nomal, Tube)을 설정합니다.

- SmartProgressBar.BARTYPE.Flat : 단색 Bar (Bar Gage가 프레임과 여백 없음)
- SmartProgressBar.BARTYPE.Flat1 : 단색 Bar (Bar Gage가 프레임과 여백 있음)
- SmartProgressBar.BARTYPE.Nomal : 세로형 그라데이션 Bar (Bar Gage가 프레임과 여백 없음)
- SmartProgressBar.BARTYPE.Nomal1 : 세로형 그라데이션 Bar (Bar Gage가 프레임과 여백 있음)
- SmartProgressBar.BARTYPE.Tube : 중앙형 그라데이션 Bar (Bar Gage가 프레임과 여백 없음)
- SmartProgressBar.BARTYPE.Tube1 : 중앙형 그라데이션 Bar (Bar Gage가 프레임과 여백 있음)



C# 사용법

```
smartProgressBar1.BarStyle = SmartProgressBar.BARTYPE.Normal;
```

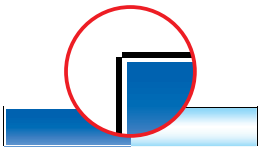
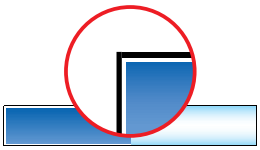
VB 사용법

```
smartProgressBar1.BarStyle = SmartProgressBar.BARTYPE.Normal
```

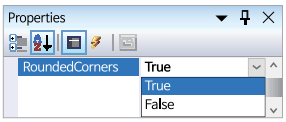
프로퍼티(속성) RoundedCorners

프레임 모서리 부분의 Round 처리를 설정합니다.

- bool : 모서리 Round 처리 유무
  - true : Round 처리 함
  - false : Round 처리 안 함

True	False
	

사용법

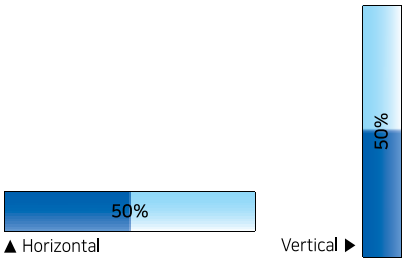


프로퍼티(속성) Direction

SmartProgressBar의 가로/세로 방향을 설정합니다.

- SmartProgressBar.DIR.Horizontal : 가로 방향
- SmartProgressBar.DIR.Vertical : 세로 방향

참고 세로 방향일 경우 SmartProgressBar에 출력되는 Text 방향 설정은 TextAutoRotation 속성을 참고하시기 바랍니다.



C# 사용법

```
smartProgressBar1.Direction = SmartProgressBar.DIR.Vertical;  // 세로 방향의 ProgressBar
```

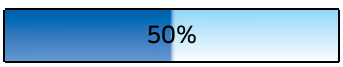
VB 사용법

```
smartProgressBar1.Direction = SmartProgressBar.DIR.Vertical
```

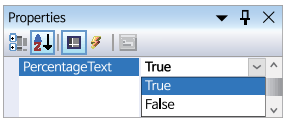
프로퍼티(속성) PercentageText

SmartProgressBar의 중앙에 Percent(%) 값의 Text 표시 상태를 설정합니다.

- bool : Text 표시 유무
  - true : Text 표시함
  - false : Text 표시 안 함

True	False
	

사용법



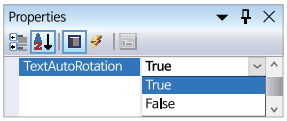
프로퍼티(속성) TextAutoRotation

SmartProgressBar의 방향이 세로인 경우 Percent Text 또는 TextDirectOutput() 메소드에 의해서 출력되는 Text의 방향을 자동으로 세로 방향으로 변경할 것인지 설정합니다.

- `bool` : 세로 방향인 경우 자동으로 Text의 출력을 세로 방향으로 변경
  - `true` : 자동으로 변경함
  - `false` : 자동으로 변경 안 함(항상 가로 방향 출력)

True	False
	

사용법



프로퍼티(속성) Maximum, Minimum

SmartProgressBar의 Percent(%)의 최대값과 최소값을 설정합니다.

- `int` : Percent(%)의 최대값과 최소값

C# 사용법

```
// 최대값 설정
smartProgressBar1.Maximum = 100;
// 최소값 설정
smartProgressBar1.Minimum = 0;
```

VB 사용법

```
smartProgressBar1.Maximum = 100
smartProgressBar1.Minimum = 0
```

프로퍼티(속성) Value

SmartProgressBar의 Bar Gage 값을 설정합니다. Minimum, Maximum 값에 준하여 설정하시면 됩니다.

- `int` : Bar Gage 값

C# 사용법

```
// Bar Gage 값을 설정합니다.
smartProgressBar1.Value = 50;
```

VB 사용법

```
smartProgressBar1.Value = 50
```

메소드(함수) `TextDirectOutput`

SmartProgressBar의 중앙에 나타나는 Text를 입력합니다.

중요 PercentageText 속성값을 True로 설정하시기 바랍니다.

◉ `void TextDirectOutput(string strMsg)`

[인자]

• `string strMsg` : 사용자 지정 Text

C# 사용법

```
// 문자값을 출력하기 위해 텍스트를 보이도록 설정합니다.
smartProgressBar1.PercentageText = true;
// 출력될 값을 직접 입력합니다.
smartProgressBar1.TextDirectOutput( " 3.77Hz " );
```

VB 사용법

```
smartProgressBar1.PercentageText = true
smartProgressBar1.TextDirectOutput( " 3.77Hz " )
```

4) SmartProgressBar 예제 사용하기

SmartProgressBar를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartProgressBar"

4. SmartKeyboard

SmartKeyboard는 Windows CE에 포함된 가상 키보드(InputPanel)의 단점을 보완한 가상 키보드 패널입니다.
다음과 같은 특수한 기능을 SmartKeyboard에서 지원하고 있습니다.
가상 키보드와 관련하여 SmartKeyboard와 SmartKeyPad 두 가지 컴포넌트를 지원하고 있습니다. 선택 가이드를 참고하여 적용하시기 바랍니다.

- ■ 사용자 인터페이스에 맞는 디자인적 요소 변경 기능
  - └, 색상 및 배경 이미지 설정기능
- ■ 키 이미지 설정 기능
- ■ 다양한 디자인 스타일 기본 제공
- ■ 기본적인 숫자만을 입력하기 위한 Numerical KeyPad 기능
- ■ 컨트롤 속성에서 Text 속성이 있는 경우 모두 적용 가능. 즉, Label 컨트롤에도 적용 가능 (기존의 가상 키보드는 TextBox에서만 입력 가능)
- ■ 한글 입력 지원

[일반 키보드]

[숫자 입력 패드]

1) SmartKeyPad 와 SmartKeyboard 의 선택 가이드

IEC-Series에서 키 입력을 지원하는 컴포넌트는 SmartKeyPad와 SmartKeyboard가 있습니다. 아래 표를 확인하여 적합한 컴포넌트를 프로젝트에 적용하시기 바랍니다.

[표] SmartKeyPad와 SmartKeyboard 비교

	SmartKeyPad	SmartKeyboard
형태		
Layout과 키 구성	모든 다양한 형태로 구성 가능	미리 구성된 Layout만 사용가능 (NORMAL, NUMERICAL)
키 정의	항상 각 키의 정의 필요	키 정의 필요없이 바로 사용 가능 NORMAL Layout 형태 키 정의와 NUMERICAL Layout 형태의 키 정의만 사용 가능
디자인 적용 범위	모든 디자인 적용 가능	제한적으로 변경 적용 가능
지원 컨트롤	컨트롤 속성에서 Text 속성이 있는 경우 모두 적용 가능 (SmartLabel, Label 지원)	

입력 처리 위치	1. TextBox의 경우 - TextBox의 KeyDown, KeyUp, KeyPress 이벤트 - SmartKeyPad의 OnKeyButtonDown, OnKeyButtonUp 이벤트 2. SmartLabel 또는 Label의 경우 - OnLabelKeyDown, OnKeyButtonUp, OnKeyButtonDown 이벤트	1. TextBox의 경우 - TextBox의 KeyDown, KeyUp, KeyPress 이벤트 2. SmartLabel 또는 Label의 경우 - SmartKeyboard의 OnLabelKeyPress 이벤트

2) 프로그래밍 적용 가이드

NO-1 SmartKeyboard 사용하기

SmartKeyboard를 사용하기 위해서는 우선 입력 대상(타겟)을 설정해야 합니다. SmartKeyboard는 Text 속성이 있는 모든 컨트롤에 입력이 가능합니다. 즉, SmartLabel과 Label에 입력이 가능합니다. 타겟 설정이 완료되면 Show() 메소드를 사용해 SmartKeboard를 화면에 출력합니다. 이때 인자값을 이용해 위치, 크기, 타겟, 타입(Normal, Numerical)을 설정 가능합니다. 또한 특정 키 입력 시 동작을 처리하고 싶은 경우 OnLabelKeyPress 이벤트를 사용해 인자값을 확인하여 동작을 처리할 수 있습니다.

※ 자세한 내용은 "TargetInputObject, KeyboardType 속성", "Show() 메소드", "OnLabelKeyPress 이벤트" 내용을 참고하시기 바랍니다.

```
// SmartKeyboard1을 Normal 모드로 (50, 70) 위치에 450X180 사이즈로 출력하고
// 입력 대상을 SmartLabel1로 설정
smartKeyboard1.Show(50, 70, 450, 180, smartLabel1, SmartX.SmartKeyboard.KEYBOARDTYPES.NORMAL);
// 입력 대상이 SmartLabel 또는 Label일 경우 키 입력 시 발생하는 이벤트
private void smartKeyboard1_OnLabelKeyPress(object sender, KeyPressEventArgs e)
{
  // 현재 입력된 타겟이 SmartLabel1인 경우
  if (sender == smartLabel1)
  {
    // 입력된 키가 Enter인 경우
    if (e.KeyChar == (char)Keys.Enter)
    {
      // 입력을 무시하고 SmartKeyboard1을 닫음
      e.Handled = true;
      smartKeyboard1.Hide();
    }
  }
}
```

3) SmartKeyboard 인터페이스 설명

SmartKeyboard Component Interface		
속성		
ControlKeyDisable : bool	DesignMinimize : bool	FontColor : Color
HanYoungKeyDisable : bool	Heigth : int	KeyboardType : SmartKeyboard.KEYBOARDTYPES
KeyFillColor : Color	KeyOutLineColor : Color	KeyOutLineWidth : int
KeyPressImage1 : Image	KeyPressImage2 : Image	KeyPressImage3 : Image
KeyUpImage1 : Image	KeyUpImage2 : Image	KeyUpImage3 : Image

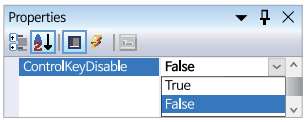
Left : int	LeftMargin : int	PressFontColor : Color
PressKeyFillColor : Color	PressKeyOutLineColor : Color	RoundedCorners : bool
SetBackimage : Image	TabKeyDisable : bool	TargetInputObject : Control
ThemeStyle : SmartKeyboard.KEYBOARDTHEMSTYLE	Top : int	TopMargin : int
Visible : bool	Width : int	
메소드		
HanYoungKeyToggle() : bool	Hide() : void	LabelSelect(int iStart, int iLength) : void
LabelClear() : void	LableSetTextChange(string strText) : void	SetHangulfont(bool bHangul) : void
SetKeyTextFontName(string strFontName) : void	Show() : void (+3개 오버로드)	
이벤트		
OnLabelKeyPress : EventHandler	OnHanYoungKeyChange : EventHandler	OnXKeyClick : EventHandler

프로퍼티(속성) ControlKeyDisable

가상 키보드에서 Alt, Ctrl 키를 활성화 및 비활성화 해주는 속성입니다.

- bool : Alt, Ctrl 키 활성화 여부
  - true : 비활성화
  - false : 활성화(기본값)

사용법

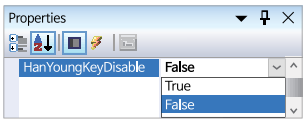


프로퍼티(속성) HanYoungKeyDisable

SmartKeyboard의 한/영 키를 활성화 및 비활성화 해주는 속성입니다.

- bool : 한/영 키 활성화 여부
  - true : 비활성화
  - false : 활성화(기본값)

사용법

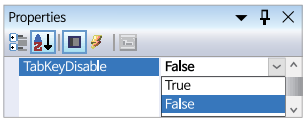


프로퍼티(속성) TabKeyDisable

SmartKeyboard의 Tab 키를 활성화 및 비활성화합니다.

- bool : Tab 키 활성화 여부
  - true : 비활성화
  - false : 활성화

사용법



프로퍼티(속성) DesignMinimize

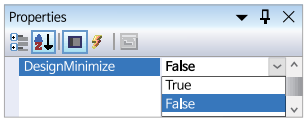
폼 디자인 모드에서 SmartKeyboard의 크기를 최소화시켜 폼의 다른 디자인을 원활하게 할 수 있도록 하는 속성입니다. (장치의 RunTime 모드에서는 무시됩니다.)

[표] DesignMinimize 속성값에 따른 디자인 모드에서 SmartKeyboard 표시

DesingMinimize = false	DesingMinimize = true
	

- bool : SmartKeyboard 최소화 여부
  - true : 최소화
  - false : 전체 사이즈(기본값)

사용법 속성값의 변경은 반드시 폼 디자인 모드의 속성창에서 변경하시기 바랍니다.

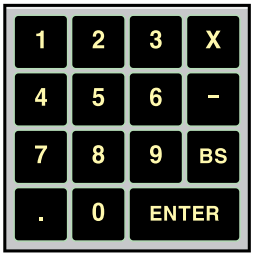


프로퍼티(속성) KeyboardType

SmartKeyboard의 모드를 설정합니다.

- SmartKeyboard.KEYBOARDTYPES.NORMAL : 일반적인 키보드

- SmartKeyboard.KEYBOARDTYPES.NUMERICAL : 숫자 입력 패드(키패드)



C# 사용법

```
smartKeyboard1.KeyboardType = SmartKeyboard.KEYBOARDTYPES.NUMERICAL;
```

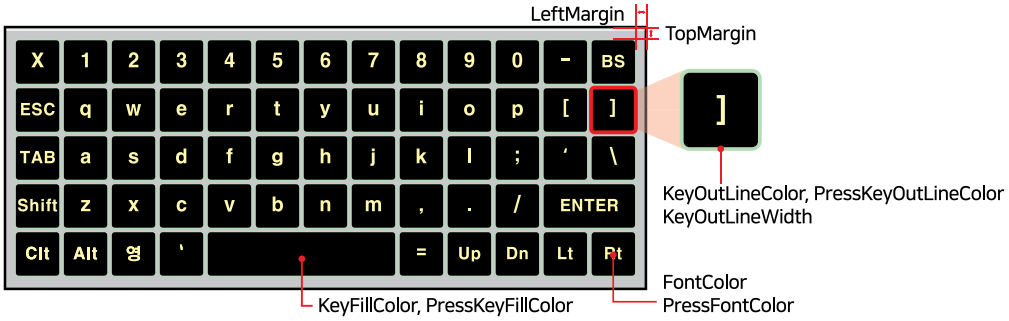
VB 사용법

```
smartKeyboard1.KeyboardType = SmartKeyboard.KEYBOARDTYPES.NUMERICAL
```

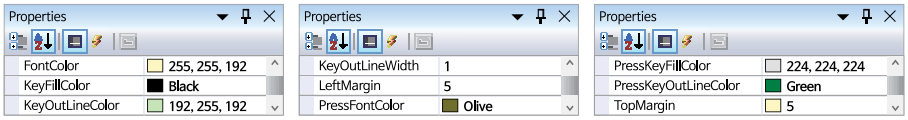
프로퍼티(속성) FontColor, KeyFillColor, KeyOutLineColor, KeyOutLineWidth, LeftMargin, PressFontColor, PressKeyFillColor, PressKeyOutLineColor, TopMargin

SmartKeyboard의 디자인 관련 속성을 변경할 수 있습니다.

- `FontColor` : Key Button Text 색상을 설정합니다.
- `KeyFillColor` : Key Button 색상을 설정합니다.
- `KeyOutLineColor` : Key Button 테두리의 색상을 설정합니다.
- `KeyOutLineWidth` : Key Button 테두리의 두께를 설정합니다.
- `LeftMargin` : SmartKeyboard의 Key Button과 배경의 좌측 거리를 설정합니다.
- `PressFontColor` : Key Button이 눌렸을 때의 Text 색상을 설정합니다.
- `PressKeyFillColor` : Key Button이 눌렸을 때의 색상을 설정합니다.
- `PressKeyOutLineColor` : Key Button이 눌렸을 때의 테두리 색상을 설정합니다.
- `TopMargin` : SmartKeyboard의 Key Button과 배경의 상단 거리를 설정합니다.



사용법

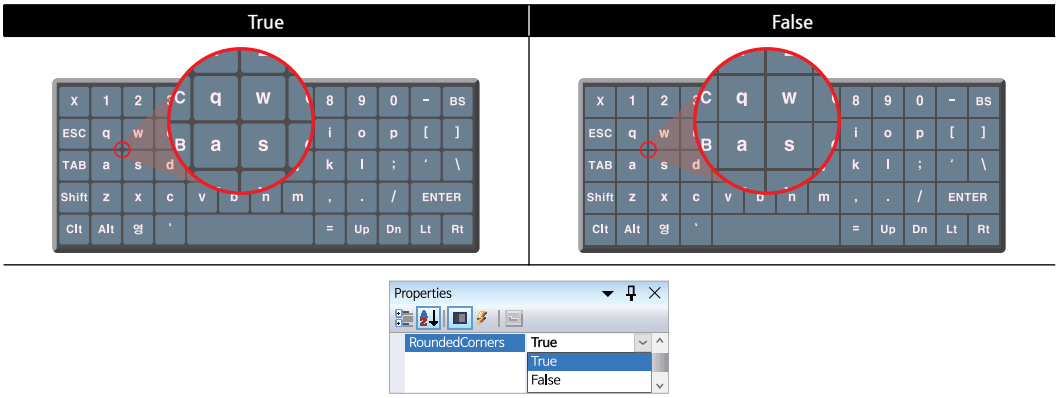


프로퍼티(속성) RoundedCorners

ThemeStyle 속성값이 STANDARD 계열인 경우 모서리 부분을 라운드 처리할 것인지 설정합니다.

- `bool` : SmartKeyboard 테두리 라운드 처리 여부
  - `true` : 라운드 처리 (기본값)
  - `false` : 직각 처리

사용법



프로퍼티(속성) ThemeStyle

SmartKeyboard의 스타일을 설정합니다.

• SmartKeyboard.KEYBOARDTHEMESTYLE.CUSTOMIZED_LITE : 키 Button의 이미지는 기존을 유지하며, 속성값 (Font Color, PressFontColor, KeyFillColor, PressKeyFillColor, KeyOutLineColor, PressKeyOutLineColor)을 이용해 색상을 변경

• SmartKeyboard.KEYBOARDTHEMESTYLE.CUSTOMIZED_SKIN : 키 Button의 이미지를 속성값(KeyUpImage1, KeyUp Image2, KeyUpImage3, KeyPressImage1, KeyPressImage2, KeyPressImage3)을 이용해 사용자 이미지로 설정

• SmartKeyboard.KEYBOARDTHEMESTYLE.STANDARD1 ~ NAVYBLUE_SKIN : 미리 지정된 스타일을 사용

참고 ThemeStyle 속성값에 따른 키보드 스타일 이미지


CUSTOMIZED_LITE


CUSTOMIZED_SKIN


STANDARD1


STANDARD2


STANDARD3


STANDARD4

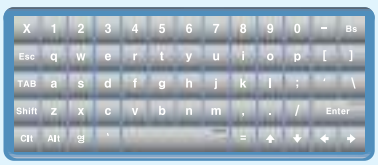

STANDARD5


STANDARD6


STANDARD7


STANDARD8

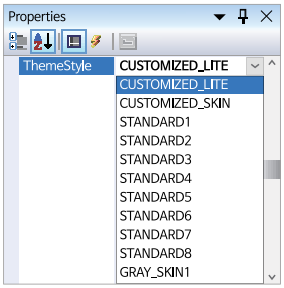

GRAY_SKIN1


GRAY_SKIN2

DARKVIOLET_SKIN

NAVYBLUE_SKIN

사용법



프로퍼티(속성) `KeyPressImage1, KeyPressImage2, KeyPressImage3` `KeyUpImage1, KeyUpImage2, KeyUpImage3` Masking Image

ThemeStyle 속성값이 CUSTOMIZED_SKIN인 경우 키 Button의 이미지를 각각 설정하여 사용합니다.

- `KeyPressImage1` : 눌려진 상태의 기본 키(1칸)
- `KeyPressImage2` : 눌려진 상태의 Enter 키(2칸)
- `KeyPressImage3` : 눌려진 상태의 Space 키(4칸)
- `KeyUpImage1` : 눌려지지 않은 상태의 기본 키(1칸)
- `KeyUpImage2` : 눌려지지 않은 상태의 Enter 키(2칸)
- `KeyUpImage3` : 눌려지지 않은 상태의 Space 키(4칸)

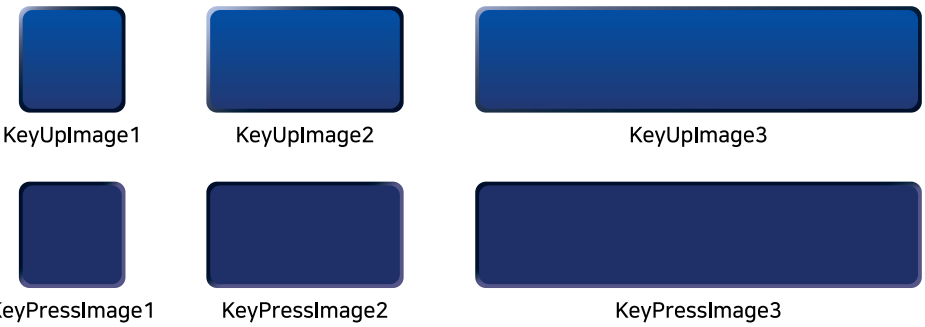
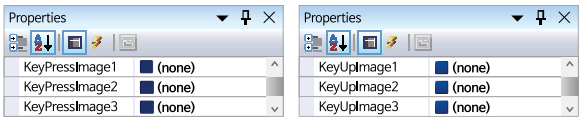


Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 “투명(Masking) 처리 영역 제작 가이드”에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 “이미지 제작 가이드”와 “투명(Masking) 처리 영역 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) SetBackimage Image

SmartKeyboard의 배경 이미지를 설정합니다.

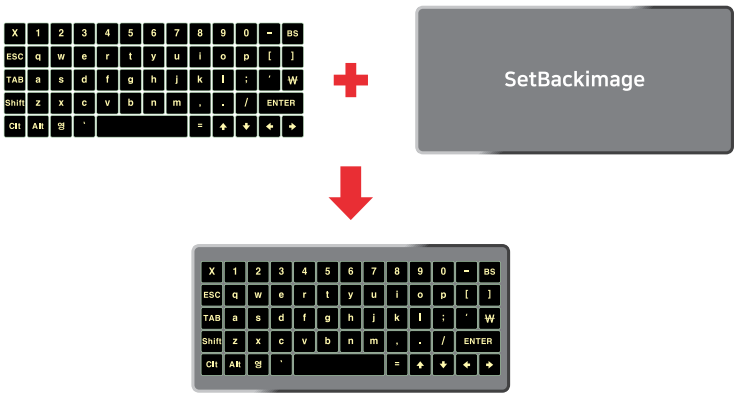


Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법

프로퍼티(속성) Left, Top

SmartKeyboard 키보드의 위치를 설정합니다.

- `Left` : SmartKeyboard가 위치할 X 좌표를 설정합니다. (기준값 : 폼 좌측 상단(0,0))
- `Top` : SmartKeyboard가 위치할 Y 좌표를 설정합니다. (기준값 : 폼 좌측 상단(0,0))
- `int` : SmartKeyboard가 위치할 좌표 (단위 : Pixel)

C# 사용법

```
smartKeyboard1.Left = 300;  // 왼쪽으로부터 300픽셀
smartKeyboard1.Top = 200;   // 위로부터 200픽셀
```

VB 사용법

```
smartKeyboard1.Left = 300 : smartKeyboard1.Top = 200
```

프로퍼티(속성) TargetInputObject

SmartKeyboard의 입력 대상 컨트롤을 설정합니다. TextBox 뿐만 아니라 Label 등 다양한 컨트롤도 설정할 수 있습니다.

주의 TargetInputObject 속성이 설정되지 않은 경우 키 입력이 올바르게 되지 않을 수 있습니다.

- `Control` : SmartKeyboard의 입력 대상 컨트롤

C# 사용법

```
smartKeyboard1.TargetInputObject = textbox1; smartKeyboard1.TargetInputObject = smartlabel1;
```

VB 사용법

```
smartKeyboard1.TargetInputObject = textbox1; smartKeyboard1.TargetInputObject = smartlabel1
```

프로퍼티(속성) Visible

SmartKeyboard의 화면 표시 여부를 설정합니다

권장 Show() 메소드 사용을 권장합니다.

- bool : SmartKeyboard 화면 표시 여부
  - true : 표시
  - false : 숨김

C# 사용법

```
// SmartKeyboard를 표시합니다.
smartKeyboard1.Visible = true;
```

VB 사용법

```
smartKeyboard1.Visible = True
```

메소드(함수) Show

SmartKeyboard를 화면에 출력합니다. Show() 메소드는 Visible 속성과 다르게 인자를 사용하여 출력할 수 있습니다.

참고 "Left, Top, Height, Width, KeyboardType, SetBackimage, TargetInputObject 속성" 내용을 참고하시기 바랍니다.

◉ void Show()
◉ void Show(int iLeft, int iTop, Control cTargetInputObject)
◉ void Show(int iLeft, int iTop, int iWidth, int iHeight, Control cTargetInputObject)
◉ void Show(int iLeft, int iTop, int iWidth, int iHeight, Control cTargetInputObject, SmartKeyboard.KEYBOARDTYPES eKeyboardType)
◉ void Show(int iLeft, int iTop, int iWidth, int iHeight, Control cTargetInputObject, SmartKeyboard.KEYBOARDTYPES eKeyboardType, Image iBackimage)

[인자]

- int iLeft : SmartKeyboard의 X 좌표(Form의 좌측 상단(0, 0) 기준)
- int iTop : SmartKeyboard의 Y 좌표(Form의 좌측 상단(0, 0) 기준)
- int iWidth : SmartKeyboard의 너비
- int iHeight : SmartKeyboard의 높이
- Control cTargetInputObject : SmartKeyboard로 입력할 컨트롤
- SmartKeyboard.KEYBOARDTYPES eKeyboardType : SmartKeyBoard의 모드를 설정
- Image iBackimage : SmartKeyboard의 배경 이미지

C# 사용법

```
smartKeyboard1.Show();
smartKeyboard1.Show(24, 60, labID);
smartKeyboard1.Show(24, 60, 753, 310, labID);
smartKeyboard1.Show(24, 60, 753, 310, labID, SmartKeyboard.KEYBOARDTYPES.NORMAL);
```

VB 사용법

```
smartKeyboard1.Show()
smartKeyboard1.Show(24, 60, labID)
smartKeyboard1.Show(24, 60, 753, 310, labID)
smartKeyboard1.Show(24, 60, 753, 310, labID, SmartKeyboard.KEYBOARDTYPES.NORMAL)
```

메소드(함수) Hide

SmartKeyboard를 화면상에서 보이지 않도록 합니다.

◉ void Hide()

C# 사용법

```
smartKeyBoard1.Hide();  // SmartKeyboard 숨김
```

VB 사용법

```
smartKeyboard1.Hide()
```

메소드(함수) HanYoungKeyToggle

SmartKeyboard의 한/영 상태를 변경(Toggle)시킵니다.

주의 HanYoungKeyToggle() 메소드 사용 시 주의사항

1. HanYoungKeyToggle() 메소드는 반드시 Show() 메소드보다 먼저 호출되어야 합니다.

올바른 사용 예	잘못된 사용 예

smartKeyboard1.HanYoungKeyToggle();
// Show() 메소드 전 호출되어 올바르게 적용됨
smartKeyboard1.Show();
	smartKeyboard1.Show();
	// Show() 메소드 다음 호출되어 적용 안 됨
	smartKeyboard1.HanYoungKeyToggle();

2. HanYoungKeyToggle() 메소드와 Show() 메소드 사이에서 MessageBox 또는 SmartMessageBox를 호출하지 마시기 바랍니다.

올바른 사용 예	잘못된 사용 예

smartKeyboard1.HanYoungKeyToggle();
// 코드사이에 아무런 코드가 없어 올바르게 호출됨
smartKeyboard1.Show();
	smartKeyboard1.HanYoungKeyToggle();
	// 코드 중간에 MessageBox가 들어가서
	// SmartKeyboard가 올바르게 보이지 않음
	MessageBox.Show(" Message ");
	smartKeyboard1.Show();

◉ void HanYoungKeyToggle()

C# 사용법

```
smartKeyboard1.HanYoungKeyToggle();
smartKeyboard1.Show();
```

VB 사용법

```
smartKeyboard1.HanYoungKeyToggle()
smartKeyboard1.Show()
```

메소드(함수) SethangulFont

한/영 키의 표현 방법을 설정

◉ void SetHangulFont(bool bHangul)

[인자]

- bool : 한/영 키 표현 방법
  - true : 한글 (한/영)
  - false : 영어 (H/E)

C# 사용법

```
smartKeyboard1.SethangulFont(true);  // 한/영 키의 텍스트를 한글(영/한)로 표현
```

VB 사용법

```
smartKeyboard1.SethangulFont(true)
```

메소드(함수) SetKeyTextFontName

SmartKeyboard의 키 Button이 한글일 경우 표시할 폰트를 설정합니다.

중요 반드시 해당 폰트가 IEC-Series에 설치되어 있어야 합니다. 또한 영문 폰트를 적용 시 한글, 영문, 숫자의 폰트가 적용되지 않으니 반드시 한글 폰트를 사용하시기 바랍니다.

참조 다양한 폰트를 추가하는 방법은 "홈페이지 → 자료실 → Tech Note → 20. IEC-Series에서 다양한 폰트(Fonts 폴더, 트루타입)사용 방법 안내" 내용을 참조하시기 바랍니다.

참고 OS가 영문(Optimum)인 경우

OS 버전이 Optimum인 경우, 한글 IME가 없으므로 해당 폰트가 표시는 되지만 키보드 입력 시 영문으로 입력됩니다.

◉ `void SetKeyTextFontName(string strFontName)`

[인자]

- `string strFontName` : 한글 폰트

C# 사용법

```
private void Form1_Load(object sender, EventArgs e)
{
  smartKeyboard1.SetKeyTextFontName( " gulim " );
}
```

VB 사용법

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
  smartKeyboard1.SetKeyTextFontName( " gulim " )
End Sub
```

메소드(함수) LabelSelect

TextBox의 Select() 메서드와 동일한 기능의 메서드로, 컨트롤 객체의 키 입력 위치를 지정합니다.

참고 LableSetTextChange() 메소드를 사용하여 컨트롤 객체의 Text를 수정한 경우, LabelSelect() 메소드를 이용하여 다음 키 입력 위치가 맨 앞으로 이동하는 현상을 개선할 수 있습니다.

◉ `void LabelSelect(int iStart, int iLength)`

[인자]

- `int iStart` : 선택 영역에 있는 첫 문자의 문자 인덱스(0부터 시작)
- `int iLength` : 선택 영역의 길이(문자 수)

C# 사용법

```
private void smartKeyboard1_OnLabelKeyPress(object sender, KeyPressEventArgs e)
{
  if (e.KeyChar == (char)Keys.Space)
  {
    // Space 입력을 무시
    e.Handled = true;
    // Space 입력 시 SmartLabel2의 Text를 변경
    smartKeyboard1.LableSetTextChange(smartLabel2.Text + " ABC " );
  }
  // 다음 입력 위치를 Text 끝으로 설정
  smartKeyboard1.LableSelect((smartLabel2.Text).Length, 0);
}
```

VB 사용법

```
Private Sub smartKeyboard1_OnLableKeyPress(ByVal sender As Object, ByVal e As KeyPress EventArgs)
Handles smartKeyboard1.OnLableKeyPress
  If e.KeyChar = CChar(ChrW(Keys.Space)) Then
    e.Handled = True
    smartKeyboard1.LableSetTextChange(smartLabel2.Text & " ABC " )
  End If
   smartKeyboard1.LabelSelect((smartLabel2.Text).Length, 0)
End Sub
```

메소드(함수) LableClear

TargetInputObject로 설정된 컨트롤이 SmartLabel인 경우, SmartKeyboard가 저장하고 있는 SmartLabel의 Text 정보를 초기화합니다.

중요 SmartLable의 Text를 초기화할 경우 SmartLabel의 Text를 빈문자열로 변경만 한다면 SmartLable의 Text는 순간 지워지지만 버퍼에 저장된 Text가 지워지지 않아 SmartKeyboard를 통해 Text 입력 시 기존의 Text에 새로 입력한 Text가 추가됩니다. 따라서 반드시 LableClear 메소드를 호출해 버퍼에 저장된 Text를 지우는 과정이 필요합니다.

◉ void LableClear()

C# 사용법

```
// SmartKeyboard의 Target을 SmartLabel로 설정
smartKeyboard1.TargetInputObject = smartLable1;
// SmartKeyboard 내부의 SmartLabel Text 정보를 초기화
smartLable1.Text = " ";
// SmartLabel Text를 초기화
smartKeyboard1.LableClear();
```

SmartLabel의 텍스트가 지워진 후 SmartKeyboard를 통해 Text 입력 시 지워진 Text에 새로 입력한 Text가 추가됨

LabelClear 함수 호출 즉시 Target으로 지정한 SmartLabel의 텍스트 지워짐

VB 사용법

```
smartKeyboard1.TargetInputObject = smartLabel1
smartKeyboard1.LableClear()
smartLabel1.Text = " "
```

메소드(함수) LableSetTextChange

SmartKeyboard의 TargetInputObject로 설정된 컨트롤이 SmartLabel 또는 Lable인 경우, 컨트롤 객체의 Text를 변경합니다.

중요 SmartLable의 Text를 수정할 경우 SmartLabel의 Text를 수정할 Text로만 변경만 한다면 SmartLable의 Text는 순간 변경되지만 버퍼에 저장된 Text는 수정되지 않아 SmartKeyboard를 통해 Text 입력 시 기존의 Text에 새로 입력한 Text가 추가됩니다. 따라서 반드시 LabelSetTextChange 메소드를 호출해 버퍼에 저장된 Text를 변경하는 과정이 필요합니다.

참고 LableSetTextChange() 메소드를 사용하여 컨트롤 객체의 Text를 수정한 경우, 다음 키 입력 위치가 맨 앞으로 이동하게 됩니다. 이때 LableSelect() 메소드를 이용하면 현상을 개선할 수있습니다. "LableSelect() 메소드" 내용을 참고하시기 바랍니다.

◉ void LableClear()

C# 사용법

```
// SmartKeyboard1의 Target을 SmartLabel1로 설정
smartKeyboard1.TargetInputObject = smartLabel1;
// SmartLabel1의 Text를 SmartX Framework로 수정
smartLabel1.Text = " SmartX Framework ";
// SmartKeyboard1 내부의 SmartLabel Text 정보를 수정
smartKeyboard1.LabelSetTextChange( " SmartX Framework " );
// SmartLabel1의 다음 키 입력 위치를 맨 끝으로 이동
smartKeyboard1.LabelSelect((smartLabel1.Text).Length, 0);
```

SmartLabel의 Text가 수정된 후 SmartKeyboard를 통해 Text 입력 시 수정 전의 Text에 새로 입력한 Text가 앞에 추가됨

LabelClear 함수 호출 즉시 Target으로 지정한 SmartLabel의 텍스트가 수정되며 다음 키 입력 시 Text의 가장 끝에 추가됨

VB 사용법

```
smartKeyboard1.TargetInputObject = smartLabel1
smartLabel1.Text = " SmartX Framework "
smartKeyboard1.LabelSetTextChange( " SmartX Framework " )
smartKeyboard1.LabelSelect((smartLabel1.Text).Length, 0)
```

이벤트 OnLabelKeyPress

SmartKeyboard의 TargetInputObject로 설정된 컨트롤이 SmartLable 또는 Label인 경우, 키 Button을 클릭(KeyPress)했을 때 발생하는 이벤트입니다. 인자값 Sender를 통해 선택된 컨트롤을 확인할 수 있습니다.

◉ smartKeyboard1_OnLableKeyPress(object sender, KeyPressEventArgs e)

[인자]

- object sender : 이벤트가 발생한 대상 컨트롤
- KeyPressEventArgs e : 컨트롤에 입력된 키값 처리를 위한 인자
  - bool Handled : 컨트롤에 입력된 키값 적용 여부 (true로 설정 시 입력 무시)
  - char KeyChar : 컨트롤에 입력된 키값

C# 사용법

```
private void smartKeyboard1_OnLableKeyPress(object sender, KeyPressEventArgs e)
{
  // smartLabel1 선택된 경우
  if (sender == smartLable1)
  {
    if (e.KeyChar == '.')
    {
      // "." 키가 입력되면 무시
      e.Handled = true;
      // 공백 문자로 강제 입력 처리
      smartLable1.Text += "  ";
    }
  }
}
```

VB 사용법

```
Private Sub smartKeyboard1_OnLableKeyPress(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles smartKeyboard1.OnLableKeyPress
  If sender Is smartLable1 Then
 If e.KeyChar = "." Then
    e.Handled = True
    smartLable1.Text += "  "
  End If
End Sub
```

참고 설정된 TargetInputObject 컨트롤에 따른 Key Event 처리 방법

CASE-1 TargetInputObject로 설정된 컨트롤이 SmartLable 또는 Label인 경우

[설정 방법]

TextBox내 이벤트인 OnLabelKeyPress 이벤트를 사용합니다.

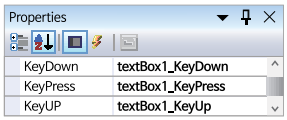
[사용 방법]

※ "OnLabelKeyPress 이벤트" 내용을 참고하시기 바랍니다.

CASE-2 TargetInputObject로 설정된 컨트롤이 TextBox인 경우

[설정 방법]

TextBox 컨트롤에서 지원하는 이벤트인 KeyPress, KeyDown, KeyUp 이벤트를 사용합니다.



※ 이벤트 발생 순서는 KeyDown → KeyPress → KeyUp의 순서로 발생합니다.

[사용 방법]

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
  // 엔터 키가 클릭된 경우
  if (e.KeyChar == (char)Keys.Enter)
  {
    // 키 입력을 무시
    e.Handled = true;
  }
}
```

이벤트 OnHanYoungKeyChange

SmartKeyboard의 한/영 키를 클릭할 때 또는 HanYoungKeyToggle() 메소드가 호출될 때마다 발생되는 이벤트입니다.

참고 프로그램 시작 시 SmartKeyboard의 자판 상태는 항상 영어입니다. OnHanYoungKeyChange 이벤트 코드에서 사용자가 지정한 Flag를 체크하게 되면 현재 자판의 상태가 한글인지 영어인지 확인이 가능합니다. 이 이벤트를 잘 응용하면 Show() 메서드 호출 시 원하는 자판의 상태(한글/영어)로 불러올 수 있습니다.

※ "HanYoungKeyToggle() 메소드" 내용을 참고하시기 바랍니다.

C# 사용법

```
// 프로그램 시작 시 자판의 상태는 항상 영어이므로 false(영어)
// 현재 SmartKeyboard 자판 상태를 저장할 Flag
private bool HanYoungKeyFlag = false;
// SmartKeyboard의 한/영 키 클릭 또는 HanYoungKeyToggle() 메소드 호출 시 발생되는 이벤트
private void smartKeyboard1_OnHanYoungKeyChange(object sender, EventArgs e)
{
  // 만약 HanYoungKeyFlag가 한글이면 영어로
  if (HanYoungKeyFlag == true)
  {
    // 플래그 체크 : 영어
    HanYoungKeyFlag = false;
  }
```

```
  else
  {
    // 만약 HanYoungKeyFlag가 영어면 한글로
    // 플래그 체크 : 한글
    HanYoungKeyFlag = true;
  }
}
// SmartLabel 클릭 시 SmartKeyboard를 Show
private void smartLabel1_Click(object sender, EventArgs e)
{
  // smartKeyboard1.Hide() 실행 전의 마지막 자판 상태가 영어라면
  if (HanYoungKeyFlag == false)
  {
    // 플래그 체크
    HanYoungKeyFlag = true;
    // 영어 -> 한글 전환(Toggle) 반대의 경우도 가능합니다.
    smartKeyboard1.HanYoungKeyToggle();
  }
  // SmartKeyboard를 출력
  smartKeyboard1.Show();
}
```

VB 사용법

```
Private HanYoungKeyFlag As Boolean = False
Private Sub smartKeyboard1_OnHanYoungKeyChange(ByVal sender As System.Object, ByVal e As aEventArgs)
  If HanYoungKeyFlag = True Then
    HanYoungKeyFlag = False
  Else
    HanYoungKeyFlag = True
  End If
End Sub
Private Sub smartLabel1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
smartLabel1.Click
  If HanYoungKeyFlag = False Then
    HanYoungKeyFlag = True
    smartKeyboard1.HanYoungKeyToggle()
  End If
End Sub
```

이벤트 OnXKeyClick

SmartKeyboard 좌측 상단의 "X" 키가 눌려지면 발생되는 이벤트입니다.

참고 "X" 키 클릭 시 SmartKeyboard가 닫히는 현상을 방지할 수 있습니다.

C# 사용법

```
private void smartKeyboard1_OnXKeyClick(object sender, EventArgs e)
{
  // X 키 클릭에 따른 처리 코드를 작성
  // X 키 클릭 시 smartKeyboard1창 닫히지 않음
  // 생략하는 경우에도 " X " 키 눌렀을 때 실행할 코드가 없기 때문에 닫히지 않습니다.
  smartKeyboard1.Visible = ture;
}
```

VB 사용법

```
Private Sub smartKeyboard1_OnXKeyClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartKeyboard1.OnXKeyClick
  ' X 키 클릭에 따른 처리 코드를 작성
  smartKeyboard1.Visible = Ture
End Sub
```

4) SmartKeyboard 예제 사용하기

SmartKeyboard를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartKeyboard"

5. SmartKeyPad

SmartKeyPad는 SmartKeyBoard와 달리 모든 형태의 가상 키보드를 구현할 수 있도록 키보드 레이아웃, 관련 이미지, 키의 위치와 개수, 전체 사이즈 등을 변경하여 필요한 키의 요소의 조합으로 키보드를 구성할 수 있도록 지원하는 컴포넌트입니다.

- 모든 형태의 키보드 구현 가능
- 필요한 키만을 정의하는 커스텀 기능
- 키 입력 처리 관련 이벤트 제공
- 이미지 설정 및 투명 처리 지원으로 모든 디자인 적용 가능
- 키 레이어 설정으로 한/영 및 기타 기호 등의 처리 가능
- 컨트롤 속성에서 Text 속성이 있는 경우 모두 적용 가능

1) SmartKeyPad 와 SmartKeyboard 의 선택 가이드

IEC-Series에서 키 입력을 지원하는 컴포넌트는 SmartKeyPad와 SmartKeyboard가 있습니다. 아래 표를 확인하여 적합한 컴포넌트를 프로젝트에 적용하시기 바랍니다.

[표] SmartKeyPad와 SmartKeyboard 비교

	SmartKeyPad	SmartKeyboard
형태		
Layout과 키 구성	모든 다양한 형태로 구성 가능	미리 구성된 Layout만 사용 가능 (NORMAL, NUMERICAL)
키 정의	항상 각 키의 정의 필요	키 정의 필요없이 바로 사용 가능 NORMAL Layout 형태 키 정의와 NUMERICAL Layout 형태의 키 정의만 사용 가능
디자인 적용 범위	모든 디자인 적용 가능	제한적으로 변경 적용 가능
지원 컨트롤	컨트롤 속성에서 Text 속성이 있는 경우 모두 적용 가능 (SmartLabel, Label 지원)	
입력 처리 위치	1. TextBox의 경우 - TextBox의 KeyDown, KeyUp, KeyPress 이벤트 - SmartKeyPad의 OnKeyButtonDown, OnKeyButtonUp 이벤트 2. SmartLabel 또는 Label의 경우 - OnLabelKeyDown, OnKeyButtonUp, OnKeyButtonDown 이벤트	1. TextBox의 경우 - TextBox의 KeyDown, KeyUp, KeyPress 이벤트 2. SmartLabel 또는 Label의 경우 - SmartKeyboard의 OnLabelKeyPress 이벤트

2) KeyCode, CKeyText 상수값 정의

SmartKeyPad는 내부에 미리 정의된 KeyCode와 CKeyText 상수값을 제공하고 있습니다. 사용자는 SetKeyInfoAdd() 메소드에서 KeyCode와 CKeyText 상수값으로 키 버튼 텍스트를 표현하거나, 입력된 키를 구분하는데 편리하게 이용할 수 있습니다. 아래에서 정의된 상수값을 확인해 보시기 바랍니다.

참고 "SetKeyInfoAdd() 메소드" 내용을 참고하시기 바랍니다.

2-1) KeyCode 열거값

KeyCode는 특정 키의 처리를 할 때 각 키의 식별을 위한 열거형(enum)의 일련 번호(키 코드)를 의미하며, SetKeyInfoAdd() 메소드의 [`SmartKeyPad.KeyCode eKey`] 인자로 사용되어 각 키를 정의할 수 있습니다.

[표] KeyCode 열거값

0	1	2	3	4	5	6	7	8	9
None	_1	_2	_3	_4	_5	_6	_7	_8	_9
10	11	12	13	14	15	16	17	18	19
_0	Hyphen	BackSpace	_q	_w	_e	_r	_t	_y	_u
20	21	22	23	24	25	26	27	28	29
_i	_o	_p	Left Bracket	Right Bracket	_a	_s	_d	_f	_g
30	31	32	33	34	35	36	37	38	39
_h	_j	_k	_l	Semicolon	Apostrophe	WonSign	_z	_x	_c
40	41	42	43	44	45	46	47	48	49
_v	_b	_n	_m	Comma	Period	Slash	Enter	Alt	한영
50	51	52	53	54	55	56	57	58	59
Single Quotation	Ctrl	Shift	Esc	Tab	SpaceBar	EqualSign	UP	DOWN	LEFT
60	61	62	63	64	65	66	67	68	69
RIGHT	Exclamation Point	AtSign	Sharp	Dollar Sign	Percent Sign	Circumflex	Ampersand	Asterisk	LeftParenthesis
70	71	72	73	74	75	76	77	78	79
Right Parenthesis	Under Score	Q	W	E	R	T	Y	U	I
80	81	82	83	84	85	86	87	88	89
O	P	LeftBrace	Right Brace	A	S	D	F	G	H
90	91	92	93	94	95	96	97	98	99
J	K	L	Colon	Quotation Mark	VerticalBar	Z	X	C	V
100	101	102	103	104	105	106	107	108	109
B	N	M	LessThan Sign	Greater ThanSign	Question Mark	Tilde	PlusSign	ㅂ	ㅈ
110	111	112	113	114	115	116	117	118	119
ㄷ	ㄱ	ㅅ	ㅛ	ㅕ	ㅑ	ㅐ	ㅔ	ㅁ	ㄴ
120	121	122	123	124	125	126	127	128	129
ㅇ	ㄹ	ㅎ	ㅗ	ㅓ	ㅏ	ㅣ	ㅋ	ㅌ	ㅊ
130	131	132	133	134	135	136	137	138	139
ㅍ	ㅠ	ㅜ	ㅡ	ㅃ	ㅉ	ㅃ	ㄲ	ㅆ	ㅒ
140	-	-	-	-	-	-	-	-	-
ㅖ	–	–	–	–	–	–	–	–	–

2-2) CKeyText 상수값

KeyText는 특정 키의 처리를 할 때 화면상 표시되는 키값(Text)을 의미하며, SetKeyInfoAdd() 메소드의 [`string keyText`] 인자로 사용되어 각 키의 버튼에 표시되는 텍스트를 설정할 수 있습니다.

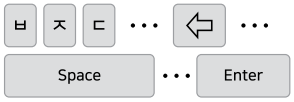
[표] CKeyText 상수값

"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"	"0"
_1	_2	_3	_4	_5	_6	_7	_8	_9	_0
"-"	"←"	"q"	"w"	"e"	"r"	"t"	"y"	"u"	"i"
Hyphen	Back Space	_q	_w	_e	_r	_t	_y	_u	_i
"o"	"p"	"{"	"}"	"a"	"s"	"d"	"f"	"g"	"h"
_o	_p	Left Bracket	Right Bracket	_a	_s	_d	_f	_g	_h
"j"	"k"	"l"	";"	"'"	"₩"	"z"	"x"	"c"	"v"
_j	_k	_l	Semicolon	Apostrophe	WonSign	_z	_x	_c	_v
"b"	"n"	"m"	","	"."	"/"	"Enter"	"Alt"	"한/영"	"`"
_b	_n	_m	Comma	Period	Slash	Enter	Alt	한영	Single Quotation
"Ctrl"	"Shift"	"Esc"	"Tab"	"Space"	"="	"▲"	"▼"	"◀"	"▶"
Ctrl	Shift	Esc	Tab	Space	EqualSign	UP	DOWN	LEFT	RIGHT
"!"	"@"	"#"	"$"	"%"	"^"	"&"	"*"	"("	")"
Exclamation Point	AtSign	Sharp	Dollar Sign	Percent Sign	Circumflex	Ampersand	Asterisk	LeftParen thesis	RightParen thesis
"_"	"Q"	"W"	"E"	"R"	"T"	"Y"	"U"	"I"	"O"
UnderScore	Q	W	E	R	T	Y	U	I	O
"P"	"{"	"}"	"A"	"S"	"D"	"F"	"G"	"H"	"J"
P	LeftBrace	Right Brace	A	S	D	F	G	H	J
"K"	"L"	":"	"""	"|"	"Z"	"X"	"C"	"V"	"B"
K	L	Colon	Quotation Mark	Vertical Bar	Z	X	C	V	B
"N"	"M"	"<"	">"	"?"	"~"	"+"	"ㅂ"	"ㅈ"	"ㄷ"
N	M	Less ThanSign	Greater ThanSign	Question Mark	Tilde	PlusSign	ㅂ	ㅈ	ㄷ
"ㄱ"	"ㅅ"	"ㅛ"	"ㅕ"	"ㅑ"	"ㅒ"	"ㅖ"	"ㅁ"	"ㄴ"	"ㅇ"
ㄱ	ㅅ	ㅛ	ㅕ	ㅑ	ㅒ	ㅖ	ㅁ	ㄴ	ㅇ
"ㄹ"	"ㅎ"	"ㅗ"	"ㅓ"	"ㅏ"	"ㅣ"	"ㅋ"	"ㅌ"	"ㅊ"	"ㅍ"
ㄹ	ㅎ	ㅗ	ㅓ	ㅏ	ㅣ	ㅋ	ㅌ	ㅊ	ㅍ
"ㅠ"	"ㅜ"	"ㅡ"	"ㅃ"	"ㅉ"	"ㄸ"	"ㄲ"	"ㅆ"	"ㅐ"	"ㅔ"
ㅠ	ㅜ	ㅡ	ㅃ	ㅉ	ㄸ	ㄲ	ㅆ	ㅐ	ㅔ

3) 프로그래밍 적용 가이드

SmartKeyPad 디자인 시 각 키 버튼 이미지를 공통으로 사용할 것인지, 또는 각각 다르게 사용할 것인지에 따라 제작해야 하는 이미지의 개수 및 크기가 달라지게 됩니다. 또한 키 버튼 이미지를 각각 사용하는 경우 반드시 키 텍스트를 이미지에 구성해야 합니다. 아래 표에서 키 텍스트 구성 방식에 따른 장단점과 CASE별 적용 방법을 확인하여 프로젝트에 적용하시기 바랍니다.

[표] 키 텍스트 구성 방식 비교

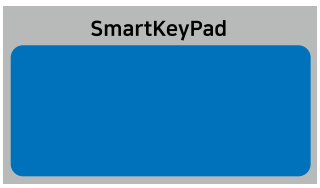
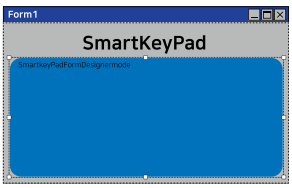
키 텍스트 구성 방식	텍스트 형식 - 최소 이미지 사용	이미지 형식 - 모든 키 이미지 사용
KeyTextVisible	True	False
출력 예시		
사용 이미지		
설명	KeyDownImageCommon, keyUpImageCommon 속성에 적용할 이미지를 제작	각 키 버튼별로 텍스트가 포함된 UP/DOWN 이미지를 제작
필요한 이미지 개수	UpImage 개수 : 4개 DownImage 개수 : 4개 **※ 총 이미지 개수 : 4 X 2 = 8개**	UpImage 개수 : 33개 DownImage 개수 : 33개 **※ 총 이미지 개수 : 33 X 2 = 66개**
장점	1. 사용되는 이미지 개수가 적음 2. 키 버튼 사이즈별로 공통의 UP/DOWN 이미지 사용 가능	1. 별도의 폰트 파일이 필요 없음 2. 글자를 깔끔하게 표현 가능 3. 한글 표시가 원활함
단점	1. 실제 표현되는 글꼴이 다를 수 있음 2. 한글(다국어) 표현 시 별도의 글꼴이 필요함	1. 사용되는 이미지 개수가 많음 2. 각 키 버튼별로 UP/DOWN 이미지를 제작 해야함

STEP-1 KeyBoard(KeyPad) 배경 설정하기

SmartKeyPad는 배경에 색상을 설정해 사용할 수 있으며, 또는 이미지를 사용해 설정할 수 있습니다.
배경에 색상을 사용하는 경우, BackColor 속성으로 배경의 색상을 설정할 수 있습니다. 단, 이때 BackPictureBox 속성을 적용하게 되면 색상이 적용되지 않습니다. 배경에 이미지를 설정해서 사용하는 방법은 Form 이미지를 투영해 사용하는 방법과 SmartKeyPad 배경을 직접 설정하는 방법이 있습니다.

[CASE-1] Form 이미지를 사용하는 경우

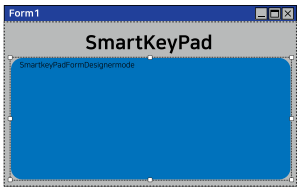
BackPictureBox 속성을 사용해 배경 Form에 설정된 이미지를 투영시킬 수 있습니다.

Form 배경 이미지	SmartKeyPad 적용 예시
	

[CASE-2] 배경을 직접 설정하는 경우

SmartKeyPad의 배경 이미지는 KeyPadBackImage 속성으로 적용할 수 있으며, 반드시 마스킹 처리가 필요합니다.
배경 이미지의 (0, 0) 좌표가 마스킹 컬러이며, BackPictureBox 속성을 적용해 사용하시기 바랍니다.

Form 배경 이미지	SmartKeyPad 배경 이미지
	



[SmartKeyPad 적용 예시]

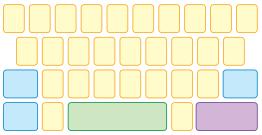
STEP-2 키 버튼 이미지 설정하기

SmartKeyPad는 키 버튼 이미지를 공용으로 사용하거나, 각 키마다 개별의 이미지를 사용할 수 있습니다.
키 버튼 이미지를 공용으로 사용하는 경우, 공통의 이미지를 각 키 버튼에 사용하기 때문에 이미지 제작 시 키 버튼에 텍스트를 추가하여 제작하시기 바라며 KeyTextVisible 속성값을 반드시 Ture로 설정해야 합니다.
키 버튼 이미지를 각 키마다 사용하는 경우, 이미지 제작 시 키 버튼에 텍스트를 추가하여 제작하시기 바라며 KeyText Visible 속성값을 반드시 False로 설정해야 합니다.

※ 자세한 내용은 "KeyTextVisible 속성" 내용을 참고하시기 바랍니다.

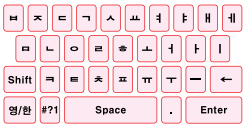
[CASE-1] 키 버튼 이미지를 공용으로 사용하는 경우

구성하려는 키 패드의 Layout에 따라 사용하는 키 버튼 이미지를 제작합니다. 이때 공용으로 사용하는 버튼을 구분하여 각 버튼 별로 UpImage, DownImage 총 2개를 제작합니다. 본 가이드에서 공통으로 사용되는 키 버튼은 4개로 제작해야 할 이미지의 개수는 "4 X 2 = 8" 총 8개를 제작해야 합니다. 또한, 각 이미지는 키 버튼이 공용으로 사용하므로 이미지에 텍스트가 포함되어 있으면 안됩니다.

구성하려는 Layout	이미지에 키 텍스트가 없는 키 버튼 이미지
	 [UpImage]
공용으로 사용되는 키 버튼	 [DownImage]
	

[CASE-2] 키 버튼 이미지를 각 키마다 사용하는 경우

구성하려는 키 패드의 Layout에 따라 사용하는 키 버튼 이미지를 제작합니다. 이때 모든 키 버튼의 이미지를 제작하며 각 버튼 별로 UpImage, DownImage 총 2개를 제작합니다. 본 가이드에서 공통으로 사용되는 키 버튼은 33개로, 제작해야 할 이미지의 개수는 "33 X 2 = 66" 총 66개를 제작해야합니다. 또한, 각 이미지에 텍스트를 포함하여 제작하시기 바랍니다.

구성하려는 Layout	모든 키 버튼 제작 필요
	
키 버튼 UpImage	키 버튼 DownImage
	

STEP-3 키 버튼 정의 및 생성하기

STEP-2에서 제작한 이미지를 사용해 키 버튼 정의 및 생성을 진행합니다.
키 버튼 이미지를 공용으로 사용하는 경우, 버튼 생성 시 SetKeyInfoAdd() 메소드의 [SmartKeyPad.KeyCode eKey], [string KeyText] 인자값에 미리 정의된 KeyCode와 CKeyText 상수값으로 설정할 수 있으며, KeyText 인자를 생략 시 KeyCode에 따른 텍스트가 자동으로 출력됩니다.
키 버튼 이미지를 각 키마다 사용하는 경우, 버튼 생성 시 SetKeyInfoAdd() 메소드의 [SmartKeyPad.KeyCode eKey] 인자값을 각 키 버튼 이미지에 맞게 설정하시기 바랍니다.

※ 자세한 내용은 "KeyCode, CKeyText 상수값 정의", "SetKeyInfoAdd() 메소드" 내용을 참고하시기 바랍니다.

[CASE-1] 공통의 키 버튼 이미지를 SmartKeyPad에 적용하는 경우

프로젝트의 리소스에 이미지를 추가한 후, Form_Load 이벤트에서 SetKeyInfoAdd() 메소드를 사용해 키 버튼을 생성합니다. 키 버튼의 텍스트를 출력하기 위해 KeyTextVisible 속성값은 반드시 true로 설정하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  // 키 버튼 이미지에 텍스트가 포함되지 않아 true로 설정
  smartKeyPad1.KeyTextVisible = true;
  // 레이어 0번에 문자를 추가 (iLayer, left, top, KeyUp, KeyPress, KeyCode)
  smartKeyPad1.SetKeyInfoAdd(0, 7, 8, Resource1.type_1_KeyUp, Resource1.type_1_KeyPress,
                             SmartX.SmartKeyPad.KeyCode.ㅂ);
  // …중략…
  smartKeyPad1.SetKeyInfoAdd(413, 209, Resource1.type_3_KeyUp, Resource1.type_3_KeyPress,
                             SmartX.SmartKeyPad.KeyCode.Enter);
}
```

[CASE-2] 각각의 키 버튼 이미지를 SmartKeyPad에 적용하는 경우

프로젝트의 리소스에 이미지를 추가한 후, Form_Load 이벤트에서 SetKeyInfoAdd() 메소드를 사용해 키 버튼을 생성합니다. KeyCode 인자값을 키 버튼 이미지에 맞는 상수값으로 추가해야하며, 키 버튼 텍스트를 출력하지 않기 위해 KeyTextVisible 속성값은 반드시 false로 설정하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  // 키 버튼 이미지에 텍스트가 포함되어 false로 설정
  smartKeyPad1.KeyTextVisible = false;
  // 레이어 0번에 문자를 추가 (iLayer, left, top, KeyUp, KeyPress, KeyCode)
  smartKeyPad1.SetKeyInfoAdd(0, 7, 8, Resource1.ㅂ_KeyUp, Resource1.ㅂ_KeyPress,
                             SmartX.SmartKeyPad.KeyCode.ㅂ);
  // …중략…
  smartKeyPad1.SetKeyInfoAdd(413, 209, Resource1.Enter_KeyUp, Resource1.Enter_KeyPress,
                             SmartX.SmartKeyPad.KeyCode.Enter);
}
```

STEP-4 SmartKeyPad 사용하기

SmartKeyPad를 사용하기 위해서는 우선 입력 대상(타겟)을 설정해야 합니다. SmartKeyPad는 Text 속성이 있는 모든 컨트롤에 입력이 가능합니다. 즉, SmartLabel과 Label에 입력이 가능합니다. TargetInputObject 속성을 사용해 타겟 설정이 완료되면 Visible 속성을 사용해 SmartKeyPad를 화면에 출력합니다. LayerChange() 메소드를 사용해 원하는 레이어로 변경이 가능하며, 또한 특정 키 입력 시 동작을 처리하고 싶은 경우 OnLabelKeyDown, OnKeyButtonDown, OnKeyButtonUp 이벤트를 사용해 인자값을 확인하여 동작을 처리할 수 있습니다.

※ 자세한 내용은 "TargetInputObject, Visible 속성", "LayerChange() 메소드", "OnLabelKeyDown, OnKeyButtonDown, OnKeyButtonUp 이벤트" 내용을 참고하시기 바랍니다.

```
smartKeyPad1.TargetInputObject = smartLabel1;  // 입력 대상을 SmartLabel1로 설정
smartKeyPad1.LayerChange(0);  // 레이어를 0번으로 변경
smartKeyPad1.Visible = true;  // SmartKeyPad를 화면에 출력

// 입력 대상이 SmartLabel 또는 Label일 경우 키 입력 시 발생하는 이벤트
private void smartKeyPad1_OnLabelKeyDown(object sender, KeyPressEventArgs e)
{
  // 입력된 키가 Enter인 경우, 입력을 무시하고 SmartKeyboard1을 닫는다.
  if (e.KeyChar == (char)Keys.Enter)
  {
    e.Handled = true;
    smartKeyPad1.Visible = false;
  }
}
```

4) SmartKeyPad 인터페이스 설명

SmartKeyPad Component Interface		
속성		
BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm	BackPictureBox2 : SmartGroupBox
keyDownImageCommon : Image	KeyPadBackImage : Image	KeyTextColor : Color
KeyTextFont : Font	KeyTextLeftOffset : int	KeyTextTopOffset : int
KeyTextVisible : bool	keyUpImageCommon : Image	TargetInputObject : Control
TransparentKeyImage : bool	Visble : bool	
메소드		
KeyInput() : void	LabelClear() : void	LabelSelect(int iStart, int iLength) : void
LableSetTextChange(string strText) : void	LayerChange(int iLayer) : void	SetKeyInfoAdd(int left, int top, SmartKeyPad.KeyCode eKey) : void (+7개 오버로드)
이벤트		
OnLabelKeyDown : KeyPressEventHandler	OnKeyButtonDown : SmartKeyPad.KeyEventHandler	OnKeyButtonUp : SmartKeyPad.KeyEventHandler

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartKeyPad의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartKeyPad에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

프로퍼티(속성) KeyPadBackImage

Masking Image

SmartKeyPad의 배경 이미지를 설정합니다. 자동으로 투명 효과가 적용됩니다. 마스킹 컬러는 (0, 0) 좌표의 색상입니다. 또한, BackPictureBox 속성을 적용해 사용하시기 바랍니다.

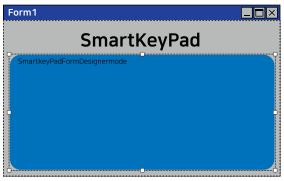
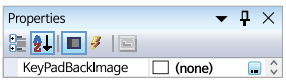
Form 배경 이미지	SmartKeyPad 배경 이미지	적용 예시
		

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) keyDownImageCommon,keyUpImageCommon

Masking Image

SmartKeyPad의 키 버튼의 크기가 모두 동일한 경우 사용하는 속성입니다.

- keyDownImageCommon : SmartKeyPad의 DownImage를 공통으로 적용합니다.
- keyUpImageCommon : SmartKeyPad의 UpImage를 공통으로 적용합니다.

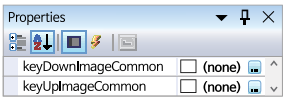
참고 "TransparentKeyImage 속성" 내용을 참고하시기 바랍니다.

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법 속성창에서 설정하시면 편리하게 설정할 수 있습니다.



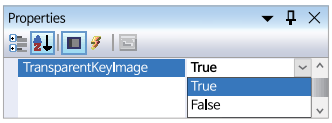
프로퍼티(속성) TransparentKeyImage

각 키 버튼 이미지의 투명 처리 여부를 설정합니다.

- bool : 키 버튼의 투명 처리 여부
  - true : 투명 처리
  - false : 투명 처리 안 함

사용되는 Key Image	
True(투명 처리)	False(투명 처리 안 함)
투명 처리를 위한 마스킹 영역	배경 이미지와 같은 색상일 경우

사용법



프로퍼티(속성) KeyTextColor, KeyTextFont, KeyTextLeftOffset, KeyTextTopOffset

키패드의 키 컬러, 폰트, 옵셋값 설정으로 KeyTextVisible 속성의 설정값이 True일 때 적용 가능합니다. 모든 키가 공통적으로 적용됩니다.

- KeyTextColor : 키 텍스트의 색상 설정
- KeyTextFont : 키 텍스트의 폰트 설정
- KeyTextLeftOffset : 키 텍스트의 X축 위치 설정
- KeyTextTopOffset : 키 텍스트의 Y축 위치 설정

참고 "KeyTextVisible 속성" 내용을 참고하시기 바랍니다.

- Color : 키 텍스트의 색상
- Font : 키 텍스트의 폰트
- int : 키 텍스트의 X좌표값 (기준점 : 각 키 버튼의 X축 중앙 / 기본값 : 0)
- int : 키 텍스트의 Y좌표값 (기준점 : 각 키 버튼의 Y축 중앙 / 기본값 : 0)

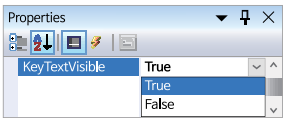
사용법

프로퍼티(속성) KeyTextVisible

키패드의 키 텍스트를 보이거나 숨깁니다. 키 텍스트를 보이게 처리하는 경우 SetKeyInfoAdd의 KeyText 또는 KeyCode에 의해 키 Text가 출력됩니다.

- bool : 키 텍스트 숨김 여부
  - true : 보임
  - false : 숨김

사용법



프로퍼티(속성) Visible

SmartKeyPad의 화면 표시 여부를 설정합니다.

권장 Show() 메소드 사용을 권장합니다.

- `bool` : SmartKeyPad 화면 표시 여부
  - `true` : 표시
  - `false` : 숨김

C# 사용법

```
smartKeyPad1.Visible = true;  // SmartKeyPad를 표시합니다.
```

VB 사용법

```
smartKeyPad1.Visible = True
```

프로퍼티(속성) TargetInputObject

SmartKeyPad의 입력 대상 컨트롤을 설정합니다. TextBox 뿐만 아니라 Label 등 다양한 컨트롤도 설정할 수 있습니다.

주의 TargetInputObject 속성이 설정되지 않은 경우 키 입력이 올바르게 되지 않을 수 있습니다.

- `Control` : SmartKeyPad의 입력 대상 컨트롤

C# 사용법

```
smartKeyPad1.TargetInputObject = textbox1; smartKeyPad1.TargetInputObject = smartlabel1;
```

VB 사용법

```
smartKeyPad1.TargetInputObject = textbox1 : smartKeyPad1.TargetInputObject = smartlabel1
```

메소드(함수) KeyInput

사용자가 원하는 KeyCode를 타겟 컨트롤에 직접 입력합니다.

참고 "KeyCode, CKeyText 상수값 정의" 내용을 참고하시기 바랍니다.

◉ `void KeyInput(SmartKeyPad.KeyCode eKeyName)`

C# 사용법

```
smartKeyPad1.KeyInput(SmartKeyPad.KeyCode._0);  // KeyCode 0 타겟 컨트롤에 입력합니다.
```

VB 사용법

```
smartKeyPad1.KeyInput(SmartKeyPad.KeyCode._0)
```

메소드(함수) SetKeyInfoAdd

Masking Image

레이어별 구성될 키의 정보를 추가하는 메소드. 즉, 키보드 구성 시 필요한 각각의 키 버튼을 만듭니다.

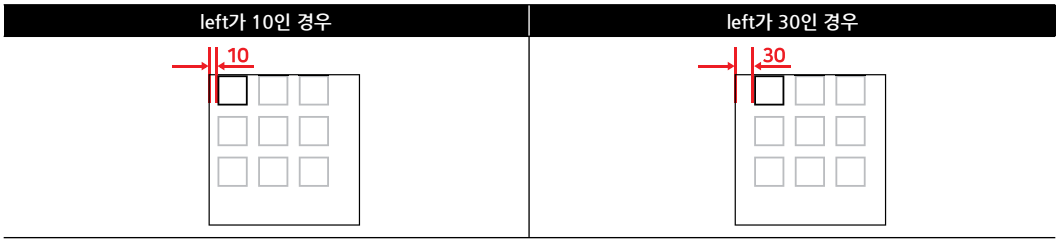
참고 "KeyCode, CKeyText 상수값 정의", "LayerChange() 메소드" 내용을 참고하시기 바랍니다.

◉ `void SetKeyInfoAdd(int left, int top, SmartKeyPad.KeyCode eKey)`
◉ `void SetKeyInfoAdd(int iLayer, int left, int top, SmartKeyPad.KeyCode eKey)`
◉ `void SetKeyInfoAdd(int left, int top, string keyText, SmartKeyPad.KeyCode eKey)`

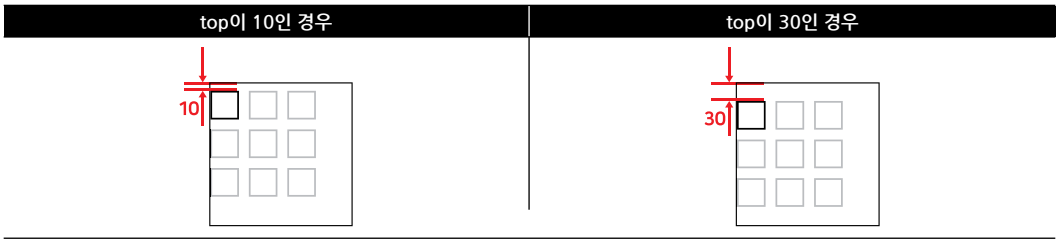
- void SetKeyInfoAdd(int left, int top, Image keyUp, Image keyPress, SmartKeyPad.KeyCode eKey)
- void SetKeyInfoAdd(int iLayer, int left, int top, string keyText, SmartKeyPad.KeyCode eKey)
- void SetKeyInfoAdd(int left, int top, Image keyUp, Image keyPress, string keyText, SmartKeyPad.KeyCode eKey)
- void SetKeyInfoAdd(int iLayer, int left, int top, Image keyUp, Image keyPress, SmartKeyPad.KeyCode eKey)
- void SetKeyInfoAdd(int iLayer, int left, int top, Image keyUp, Image keyPress, string keyText, SmartKeyPad.KeyCode eKey)

[인자]

- int iLayer : 구성할 키가 포함될 레이어 ※ "LayerChange() 메소드" 내용을 참고하시기 바랍니다.
- int left : 각 키 버튼의 X좌표값(기준점 : SmartKeyPad 좌측 상단)



- int top : 각 키 버튼의 Y좌표값(기준점 : SmartKeyPad 좌측 상단)



- SmartKeyPad.KeyCode eKey : 각 키 버튼 입력 시 발생되는 키 코드를 설정
  ※ "KeyCode, CKeyText 상수값 정의" 내용을 참고하시기 바랍니다.
- Image keyUp : 각 키 버튼의 눌려지지 않은 이미지를 설정
- Image keyPress : 각 키 버튼의 눌려진 이미지를 설정
- string keyText : 각 키 버튼의 출력되는 Text를 설정 (생략 시 키 텍스트는 eKey 인자값에 따라 설정됨)
  – 내부에 정의된 CkeyText 상수값으로 설정하거나 사용자가 원하는 Text를 출력시킬 수 있습니다.
  ※ "KeyCode, CKeyText 상수값 정의" 내용을 참고하시기 바랍니다

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

참고 SetKeyInfoAdd() 메소드의 keyText 인자 유무에 따른 사용 방법

keyText 인자는 KeyTextVisible 속성값이 True일 때 키 버튼에 보여지는 텍스트를 설정하는 인자로 SetKeyInfoAdd() 메소드는 KeyText 인자 유무로 구분할 수 있습니다. 아래 CASE의 예시로 정의되는 키 버튼은 모두 "1"을 입력하는 버튼이며, 보여지는 키 버튼의 텍스트만 다르게 보여집니다.

※ 버튼 텍스트 표현 방법의 장단점은 "3) 프로그래밍 적용 가이드 – [표] 키 텍스트 구성 방식 비교" 내용을 참고하시기 바랍니다.

CASE-1 KeyText 인자를 사용하지 않는 경우

KeyText 인자를 사용하지 않는다면, eKey 인자로 설정한 KeyCode 열거값에 Mapping된 CKeyText가 자동으로 버튼 텍스트로 설정됩니다. 또는 KeyTextVisible 속성값을 False로 하여 버튼 이미지에 텍스트를 포함하여 사용하는 방법이 있습니다.

[관련 메소드 오버로드 형식]

- SetKeyInfoAdd(int left, int top, SmartKeyPad.KeyCode eKey)
- SetKeyInfoAdd(int iLayer, int left, int top, SmartKeyPad.KeyCode eKey)
- SetKeyInfoAdd(int left, int top, Image keyUp, Image keyPress, SmartKeyPad.KeyCode eKey)
- SetKeyInfoAdd(int iLayer, int left, int top, Image keyUp, Image keyPress, SmartKeyPad.KeyCode eKey)

CASE-1-1 키 버튼 이미지에 텍스트를 포함해 사용

KeyTextVisible 속성을 False로 하여 이미지로 키 버튼 텍스트를 대체합니다.

```
// 이미지에 키 버튼 텍스트가 포함된 경우 사용 방법
smartKeyPad1.KeyTextVisible = false;
smartKeyPad1.SetKeyInfoAdd(0, 0, Resources1.OneUp, Resources1.OneDown, SmartX.SmartKeyPad.
                           KeyCode._1);
```

[출력 결과]

CASE-1-2 KeyCode 열거값으로 텍스트를 자동 Mapping

eKey 인자로 설정된 KeyCode 열거값에 따라 자동으로 Mapping되어 키 버튼 텍스트가 설정됩니다.

```
// 이미지에 키 버튼 텍스트가 포함되지 않은 경우 사용 방법
smartKeyPad1.KeyTextVisible = true;
// 키 이미지 리소스 생략 시 KeyUpImageCommon, KeyDownImageCommon 속성에서 설정된 이미지 사용
smartKeyPad1.SetKeyInfoAdd(0, 0, SmartX.SmartKeyPad.KeyCode._1);
```

[출력 결과]

CASE-2 KeyText 인자를 사용하는 경우

KeyText 인자를 사용한다면, 사용자가 직접 KeyText 인자를 설정해 키 버튼 텍스트를 변경할 수 있습니다. 또는, 미리 정의된 CKeyText 상수를 사용해 버튼 텍스트를 구성할 수 있습니다.

[관련 메소드 오버로드 형식]

- SetKeyInfoAdd(int left, int top, string keyText, SmartX.SmartKeyPad.KeyCode eKey)
- SetKeyInfoAdd(int iLayer, int left, int top, string keyText, SmartKeyPad.KeyCode eKey)
- SetKeyInfoAdd(int left, int top, Image keyUp, Image keyPress, string keyText, SmartKeyPad.KeyCode eKey)
- SetKeyInfoAdd(int iLayer, int left, int top, Image keyUp, Image keyPress, string keyText, SmartKeyPad.KeyCode eKey)

CASE-2-1 사용자가 텍스트를 직접 정의

사용자가 원하는 문자로 키 버튼 텍스트를 설정할 수 있습니다. 키 입력 시 eKey 인자로 설정한 KeyCode가 입력됩니다.

```
// 이미지에 키 버튼 텍스트가 포함되지 않은 경우 사용 방법
smartKeyPad1.KeyTextVisible = true;
// 사용자가 원하는 텍스트를 설정
smartKeyPad1.SetKeyInfoAdd(0, 0, "일", SmartX.SmartKeyPad.KeyCode._1);
```

[출력 결과]

CASE-2-2 미리 정의된 CKeyText 상수값으로 텍스트를 설정

미리 정의된 CKeyText 상수값으로 키 버튼 텍스트를 설정할 수 있습니다. 키 입력 시 eKey 인자로 설정한 KeyCode가 입력됩니다.

```
// 이미지에 키 버튼 텍스트가 포함되지 않은 경우 사용 방법
smartKeyPad1.KeyTextVisible = true;
// 미리 정의된 CKeyText 상수로 텍스트를 설정
// 키 이미지 리소스 생략 시 KeyUpImageCommon, KeyDownImageCommon 속성에서 설정된 이미지 사용
smartKeyPad1.smartKeyPad1.SetKeyInfoAdd(0, 0, SmartX.SmartKeyPad.CKeyText._1, SmartX.
                                        SmartKeyPad.KeyCode._1);
```

[출력 결과]

C# 사용법

```
// 아래의 코드는 동일한 "1" 키의 생성 방법을 인자 형식에 따른 모든 방법을 설명합니다.
// left : 0, top : 0, keyText : 1, eKey : _1
smartKeyPad1.SetKeyInfoAdd(0, 0, SmartKeyPad.CKeyText._1, SmartKeyPad.KeyCode._1);
// iLayer : 0, left : 0, top : 0, keyText : 1, eKey : _1
smartKeyPad1.SetKeyInfoAdd(0, 0, 0, SmartKeyPad.CKeyText._1, SmartKeyPad.KeyCode._1);
// left : 0, top : 0, keyUp : Resource1._1, keyPress : Resource1._1_on, keyText : 1,eKey :_1
smartKeyPad1.SetKeyInfoAdd(0, 0, Resource1._1, Resource1._1_on, SmartKeyPad.CKeyText._1,
                           SmartKeyPad.KeyCode._1);
// iLayer : 0, left : 0, top : 0, keyUp : Resource1._1, keyPress : Resource1._1_on,
// keyText : 1, eKey : _1
smartKeyPad1.SetKeyInfoAdd(0, 0, 0, Resource1._1, Resource1._1_on, SmartKeyPad.CKeyText._1,
                           SmartKeyPad.KeyCode._1);
// left : 0, top : 0, eKey: _1
smartKeyPad1.SetKeyInfoAdd(0, 0, SmartKeyPad.KeyCode._1);
// iLayer : 0, left : 0, top : 0, eKey : _1
smartKeyPad1.SetKeyInfoAdd(0, 0, 0, SmartKeyPad.KeyCode._1);
// left : 0, top : 0, keyUp : Resource1._1, keyPress : Resource1._1_on, eKey : _1
smartKeyPad1.SetKeyInfoAdd(0, 0, Resource1._1, Resource1._1_on, SmartKeyPad.KeyCode._1);
// iLayer : 0, left : 0, top : 0, keyUp : Resource1._1, keyPress : Resource1._1_on,eKey : _1
smartKeyPad1.SetKeyInfoAdd(0, 0, 0, Resource1._1, Resource1._1_on, SmartKeyPad.KeyCode._1);
```

VB 사용법

```
smartKeyPad1.SetKeyInfoAdd(0, 0, SmartKeyPad.KeyCode._1)
' 중략
smartKeyPad1.SetKeyInfoAdd(0, 0, 0, Resource1._1, Resource1._1_on, SmartKeyPad.CKeyText._1,
                           SmartKeyPad.KeyCode._1)
```

메소드(함수) LayerChange

SmartKeyPad의 레이어를 변경합니다.

참고 "SetKeyInfoAdd() 메소드" 내용을 참고하시기 바랍니다.

◉ void LayerChange(int iLayer)

중요 레이어의 개념

SmartKeyPad는 레이어 개념을 가지고 있으며, 자판을 대문자, 소문자, 한글, 특수기호 등 사용자가 원하는 개수와 배열로 레이어를 생성할 수 있습니다. 즉, Shift 또는 한/영 키 입력에 따른 자판 배열의 변경을 처리하기 위해서 필요한 개념입니다. 아래 표에서 레이어 표현 예시를 확인하시기 바랍니다.

[표] 레이어 표현 예시

0번 레이어 : 한글 키보드	1번 레이어 : 한글 된소리 키보드
	
2번 레이어 : 특수 기호 키보드	**3번 레이어 : 영 • 소문자 키보드**
	

[인자]

• int iLayer : 키보드의 레이어를 설정합니다. 한 번 설정 후 다음부터 설정된 값으로 적용됩니다.

C# 사용법

```
smartKeyPad1.LayerChange(1);  // SmartKeyPad의 Layer를 1번으로 변경
```

VB 사용법

```
smartKeyPad1.LayerChange(1)
```

메소드(함수) **LabelSelect**

TextBox의 Select() 메소드와 동일한 기능의 메서드로, 컨트롤 객체의 키 입력 위치를 지정합니다.
LableSetTextChange() 메소드를 사용하여 컨트롤 객체의 Text를 수정한 경우, LabelSelect() 메소드를 이용하여 다음 키 입력 위치가 맨 앞으로 이동하는 현상을 개선할 수 있습니다.

◉ void LabelSelect(int iStart, int iLength)

[인자]

• int iStart : 선택 영역에 있는 첫 문자의 문자 인덱스(0부터 시작)

• int iLength : 선택 영역의 길이(문자 수)

C# 사용법

```
private void smartKeyPad1_OnLabelKeyDown(object sender, KeyPressEventArgs e)
{
  if (e.KeyChar == (char)Keys.Space)
  {
    e.Handled = true;  // Space 입력을 무시
    smartKeyPad1.LableSetTextChange(smartLabel2.Text + " ABC " );  // SmartLabel2의 Text를 변경
  }
  smartKeyPad1.LabelSelect((smartLabel2.Text).Length, 0);  // 다음 키 입력 위치를 Text 끝으로 설정
}
```

VB 사용법

```
Private Sub smartKeyPad1_OnLabelKeyDown(ByVal sender As Object, ByVal e As KeyPressEventArgs)
Handles smartKeyPad1.OnLabelKeyDown
```

```
  If e.KeyChar = CChar(ChrW(Keys.Space)) Then
    e.Handled = True
    smartKeyPad1.LableSetTextChange(smartLabel2.Text & " ABC " )
  End If
  smartKeyPad1.LabelSelect((smartLabel2.Text).Length, 0)
End Sub
```

메소드(함수) LabelClear

TargetInputObject로 설정된 컨트롤이 SmartLabel인 경우, SmartKeyPad가 저장하고 있는 SmartLabel의 Text 정보를 초기화합니다.

중요 LabelClear() 메소드 호출 시 Target SmartLabel의 Text는 지워지지 않으며, 메소드 호출 후 SmartKeyPad의 키 Button 클릭 시 Target SmartLabel의 Text가 지워집니다.

◉ `void LableClear()`

C# 사용법

```
// SmartKeyPad의 Target을 SmartLabel로 설정
smartKeyPad1.TargetInputObject = smartLabel1;
// SmartKeyboard 내부의 SmartLabel Text 정보를 초기화
smartKeyPad1.LableClear();
// SmartLabel Text를 초기화
smartLabel1.Text = " ";
```

VB 사용법

```
smartKeyPad1.TargetInputObject = smartLabel1
smartKeyPad1.LableClear()
smartLabel1.Text = " "
```

메소드(함수) LableSetTextChange

SmartKeyPad의 TargetInputObject로 설정된 컨트롤이 SmartLabel 또는 Label인 경우, 컨트롤 객체의 Text를 변경합니다.

중요 OnLabelKeyDown 이벤트 코드 내에서 특정 키 입력을 대신하는 용도로 사용하시기 바랍니다.

참고 LableSetTextChange() 메소드를 사용하여 컨트롤 객체의 Text를 수정한 경우, 다음 키 입력 위치가 맨 앞으로 이동하게 됩니다. 이때 LabelSelect() 메소드를 이용하면 현상을 개선할 수 있습니다. "LabelSelect() 메소드" 내용을 참고하시기 바랍니다.

◉ `void LableSetTextChange(string strText)`

[인자]

• `string strText` : 변경할 Text 내용

C# 사용법

```
private void smartKeyPad1_OnLabelKeyDown(object sender, KeyPressEventArgs e)
{
  // smartKeyPad에서 Enter 키가 입력되면 무시하고 컨트롤 객체의 Text 수정
  if (e.KeyChar == 13)
  {
    e.Handled = true;
    smartKeyPad1.LableSetTextChange(smartLabel1.Text + " Press " );
  }
}
```

VB 사용법

```
Private Sub smartKeyPad1_OnLabelKeyDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles smartKeyPad1.OnLabelKeyDown
  If Convert.ToInt32(e.KeyChar) = 13 Then
    e.Handled = True
    smartKeyPad1.LableSetTextChange(smartLabel1.Text + " Press " )
  End If
End Sub
```

이벤트 OnKeyButtonDown, OnKeyButtonUp

- OnKeyButtonDown : 키패드의 키를 누르는 경우 발생하는 이벤트입니다.
- OnKeyButtonUp : 키패드의 키를 누른 후(눌렀다 뗀 경우) 경우 발생하는 이벤트입니다.

◉ OnKeyButtonDown(string strKeyText, int iLayer, object sender)
◉ OnKeyButtonUp(string strKeyText, int iLayer, object sender)

[인자]

- string strKeyText : 이벤트가 발생한 키의 KeyText
- int iLayer : 이벤트가 발생한 키가 존재하는 레이어
- object sender : 이벤트가 발생한 대상 컨트롤

C# 사용법

```
// 키패드의 키를 누르는 경우 발생하는 이벤트
private void smartKeyPad1_OnKeyButtonDown(string strKeyText, int iLayer, object sender)
{
  // 현재 레이어가 레이어 0번이고
  // 키 입력을 처리하는 대상 컨트롤 객체가 smartLabel1이라면
  if ((sender == smartLabel1) & (iLayer == 0))
  {
    // 만약 Shift 키가 눌려지면
    if (strKeyText == " Shift " )
    {
      // 레이어 1로 화면 전환
      smartKeyPad1.LayerChange(1);
    }
  }
}
// 키패드의 키를 누르는 경우 발생하는 이벤트
private void smartKeyPad1_OnKeyButtonUp(string strKeyText, int iLayer, object sender)
{
  // 처리는 OnKeyButtonDown과 동일합니다.
}
```

VB 사용법

```
Private Sub smartKeyPad1_OnKeyButtonDown(ByVal strKeyText As String, ByVal iLayer As Integer,
                                         ByVal sender As Object)
  If (sender = smartLabel1) And (iLayer = 0) Then
    If strKeyText = " Shift " Then
      smartKeyPad1.LayerChange(1)
    End If
  End If
End Sub
Private Sub smartKeyPad1_OnKeyButtonUp(ByVal strKeyText As System.String, ByVal iLayer As
```

```
  System.Int32, ByVal sender As System.Object) Handles smartKeyPad1.OnKeyButtonDown
  ' 처리는 OnKeyButtonDown과 동일합니다.
End Sub
```

이벤트 OnLabelKeyDown

SmartKeyPad의 TargetInputObject로 설정된 컨트롤이 SmartLabel 또는 Label인 경우, 키 Button을 클릭(KeyPress) 했을 때 발생하는 이벤트입니다. 인자값 Sender를 통해 선택된 컨트롤을 확인할 수 있습니다.

◉ OnLabelKeyDown(object sender, KeyPressEventArgs e)

[인자]

- object sender : 이벤트가 발생한 대상 컨트롤
- KeyPressEventArgs e : 컨트롤에 입력된 키값 처리를 위한 인자
  - bool Handled : 컨트롤에 입력된 키 값 적용 여부 (true로 설정 시 입력 무시)
  - char KeyChar : 컨트롤에 입력된 키 값

C# 사용법

```
private void smartKeyPad1_OnLabelKeyDown(object sender, KeyPressEventArgs e)
{
  // smartLabel1 선택된 경우
  if (sender == smartLabel1)
  {
    // '.' 입력 시 입력을 무시
    if (e.KeyChar == '.')
    {
      e.Handled = true;  // '.' 키가 입력되면 무시
      smartLabel1.Text += "  ";  // 공백 문자로 강제 입력 처리
    }
  }
}
```

VB 사용법

```
Private Sub smartKeyPad1_OnLabelKeyDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.KeyPressEventArgs) Handles smartKeyboard1.OnLabelKeyDown
  If sender Is smartLabel1 Then
    If e.KeyChar = "." Then
      e.Handled = True
      smartLabel1.Text += "  "
    End If
  End If
End Sub
```

참고 설정된 TargetInputObject 컨트롤에 따른 Key Event 처리 방법

CASE-1 TargetInputObject로 설정된 컨트롤이 SmartLabel 또는 Label인 경우

[설정 방법]

SmartKeyPad내 이벤트인 OnLabelKeyDown, OnKeyButtonUp, OnKeyButtonDown 이벤트를 사용합니다.

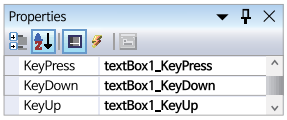
[사용 방법]

※"OnLabelKeyDown, OnKeyButtonUp, OnKeyButtonDown 이벤트" 내용을 참고하시기 바랍니다.

CASE-2 TargetInputObject로 설정된 컨트롤이 TextBox인 경우

[설정 방법]

TextBox내 이벤트인 KeyPress, KeyDown, KeyUp 이벤트를 사용합니다.



※ 이벤트 발생 순서는 KeyDown → KeyPress → KeyUp의 순서로 발생합니다.

[사용 방법]

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e)
{
  // 엔터 키가 클릭된 경우
  if (e.KeyChar == (char)Keys.Enter)
  {
    // 키 입력을 무시한다.
    e.Handled = true;
    textBox1.Text += " EnterKey_TextboxPress " ;
  }
}
```

5) SmartKeyPad 예제 사용하기

SmartKeyPad를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

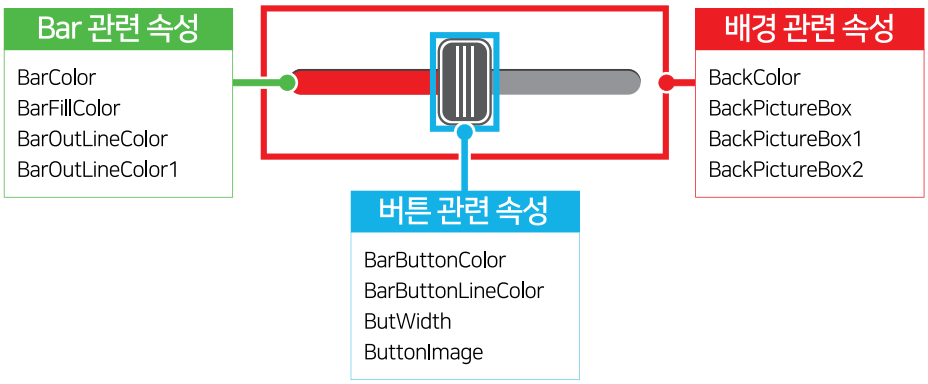
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartKeyPad"

6. SmartTrackBar

SmartTrackBar은 .NET Compact Framework에서 지원되는 TrackBar 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 TrackBar의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 배경 투명 처리 효과 기능
  - SmartForm, SmartInnerForm, SmartGroupBox 연동
- 컨트롤 Full 리사이징 및 버튼 크기 조절
- 다양한 스타일을 적용할 수 있도록 디자인 요소 변경 기능
- 일부 디자인 요소 이미지 적용 기능

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartTrackBar 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartTrackBar에서 이미지는 배경 컴포넌트와 TrackBar 버튼에 사용됩니다. TrackBar 버튼의 경우 마스킹 처리를 해야 합니다.

STEP-1 배경 속성 설정하기

SmartTrackBar의 배경 색상을 BackColor 속성을 사용해 색상으로 설정하는 방법과, BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있습니다.

[CASE-1] 색상으로 설정하는 경우

배경 컴포넌트의 배경 색상과 동일하게 설정하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackColor	Cyan

※ 자세한 내용은 “BackColor 속성” 내용을 참고하시기 바랍니다.

[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackPictureBox	SmartForm

※ 자세한 내용은 "BackPictureBox 속성" 내용을 참고하시기 바랍니다.

Image가 적용된 폼　　BackPictureBox 적용 전

STEP-2 TrackBar의 버튼 관련 속성 설정하기

TrackBar의 버튼은 버튼 이미지를 사용하지 않고 디자인하는 방법과 사용자 이미지를 사용하여 디자인하는 방법이 있습니다.

[CASE-1] 이미지를 사용하지 않을 경우

BarButtonColor, BarButtonLineColor, ButWidth 속성으로 TrackBar 버튼의 색상, 선 색상, 폭을 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
BarButtonColor	Black	ButWidth	50
BarButtonLineColor	White	-	-

※ 자세한 내용은 "BarButtonColor, BarButtonLineColor, ButWidth 속성"과 내용을 참고하시기 바랍니다.

[그림] 적용된 모습

[CASE-2] 이미지를 사용할 경우

마스킹 처리가 된 버튼 이미지를 ButtonImage 속성을 사용하여 출력합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
ButtonImage	

※ 자세한 내용은 "ButtonImage 속성" 내용을 참고하시기 바랍니다.

[그림] 적용된 모습

STEP-3 TrackBar의 Bar 관련 속성 설정하기

Bar의 가로/세로 방향은 Orientation 속성을 사용해 설정할 수 있으며, BarColor 속성으로 배경 색상을, BarFillColor 속성으로 Bar Gage의 배경 색상을 설정합니다. BarOutLineColor 속성과 BarOutLineColor1() 메소드로 Bar의 테두리 색상을 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
BarColor	LightGray	BarOutLineColor	DimGray
BarFillColor	Red	Orientation	Horizontal

※ 자세한 내용은 "Orientation, BarColor, BarFillColor, BarOutLineColor 속성"과 "BarOutLineColor1() 메소드" 내용을 참고하시기 바랍니다.

```
smartTrackBar1.BarOutLineColor1(Color.Blue);  // Bar 테두리의 하단 색상을 Blue로 변경
```

STEP-4 SmartTrackBar 관련 Step 설정하기

Step 속성으로 TrackBar의 전체 Step을 설정하며, SmallChangeStep 속성으로 버튼을 클릭했을 때 이동하는 Step을 설정합니다. Bar 버튼 영역을 제외한 구간 클릭 시 이동할 Step을 MoveOnClick 속성에서 Small Step, LargeStep, Click 중에 선택합니다. 만약 LargeStep으로 설정 시 LargeChangeStep 속성을 설정해야 합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
Step	20	LargeChangeStep	4
SmallChangeStep	2	MoveOnClick	LargeStep

※ 자세한 내용은 "Step, SmallChangeStep, LargeChangeStep, MoveOnClick 속성" 내용을 참고하시기 바랍니다.

3) SmartTrackBar 인터페이스 설명

SmartTrackBar Component Interface		
속성		
BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm	BackPictureBox2 : SmartGroupBox
BarButtonColor : Color	BarButtonLineColor : Color	BarColor : Color
BarFillColor : Color	BarOutLineColor : Color	ButtonImage : Image
ButWidth : int	InitVisible : bool	InitVisible : bool
LargeChangeStep : int	MoveOnClick : SmartTrackBar.MOVECLICKTYPE	Orientation : SmartTrackBar.ORIENTATIONTYPE
SmallChangeStep : int	Step : int	Value : double
메소드		
BarOutLineColor1(Color cColor) : void		
이벤트		
ValueChanged : EventHandler		

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartTrackBar의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 부드럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartTrackBar의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartTrackBar에 의해 배경이 가려지는 현상을 방지할 수 있습니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

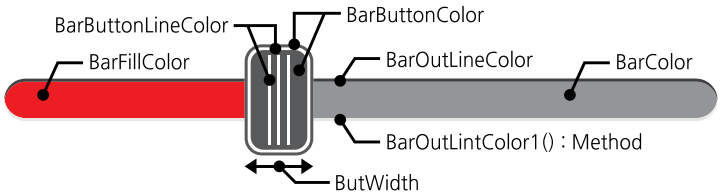
주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

프로퍼티(속성) BarButtonColor, BarButtonLineColor, BarColor, BarFillColor, BarOutLineColor, ButWidth

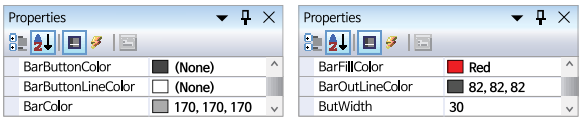
SmartTrackBar 관련 모양 및 색상 등을 설정합니다.

- `BarButtonColor` : 버튼 외부의 테두리 선과 버튼 내부 색상
- `BarButtonLineColor` : 버튼 내부의 테두리와 세 줄의 선 색상
- `BarColor` : TrackBar의 Bar 색상
- `BarFillColor` : TrackBar의 Value에 따라 채워지는 Bar 색상
- `BarOutLineColor` : TrackBar의 Bar 상단 테두리 색상
- `ButWidth` : TrackBar의 버튼 폭



참고 "BarOutLineColor1()" 메소드 설명을 참고하시기 바랍니다.

사용법



프로퍼티(속성) ButtonImage

Masking Image

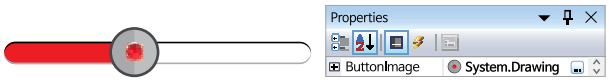
SmartTrackBar의 버튼 Image를 설정합니다.

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

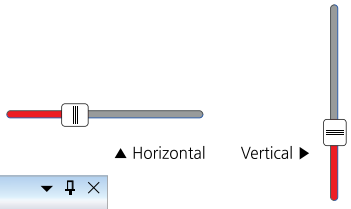
사용법



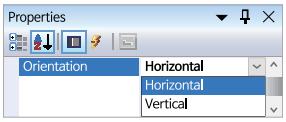
프로퍼티(속성) Orientation

SmartTrackBar의 가로, 세로 방향을 설정합니다.

- ORIENTATIONTYPE.Horizontal : 가로 방향
- ORIENTATIONTYPE.Vertical : 세로 방향



사용법



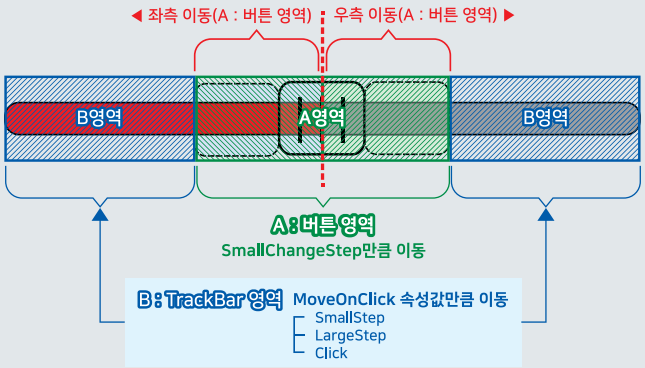
프로퍼티(속성) LargeChangeStep, SmallChangeStep, Step

SmartTrackBar관련 Step 값을 설정합니다.

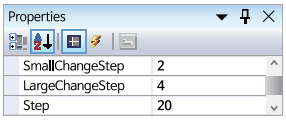
- LargeChangeStep : TrackBar의 버튼 영역이 아닌 다른 영역을 클릭할 경우 이동하는 Step
  (단, MoveOnClick 속성을 LargeStep으로 설정되어 있어야 함)
- SmallChangeStep : TrackBar의 버튼을 클릭할 경우 이동하는 Step
- Step : TrackBar의 전체 Step

중요 SmartTrackBar의 클릭 영역에 따른 버튼 이동 방식

버튼 이동 방식은 영역에 따라 달라지며 버튼 영역, TrackBar 영역으로 나뉘어집니다. 버튼 영역은 SmallChange Step만큼 이동하며, TrackBar 영역은 MoveOnClick 속성 설정값(SmallStep, LargeStep,Click)만큼 이동합니다.



사용법



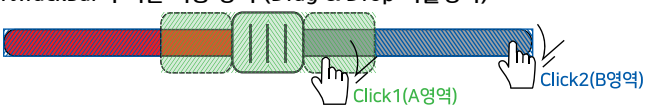
프로퍼티(속성) MoveOnClick

SmartTrackBar의 Bar 버튼 영역을 제외한 구간을 클릭 시 이동할 Step을 설정합니다.
(Bar 버튼을 클릭한다면 SmallChangeStep 값만큼 이동)

- SmartTrackBar.MOVECLICKTYPE.SmallStep : SmallChangeStep 값만큼 이동
- SmartTrackBar.MOVECLICKTYPE.LargeStep : LargeChangeStep 값만큼 이동
- SmartTrackBar.MOVECLICKTYPE.Click : 클릭한 위치에서 가장 가까운 Step으로 이동

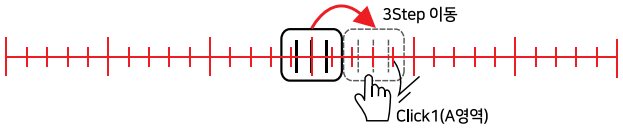
[표] MoveOnClick 속성값에 따른 SmartTrackBar의 버튼 이동 방식 (Drag & Drop 비활성화)

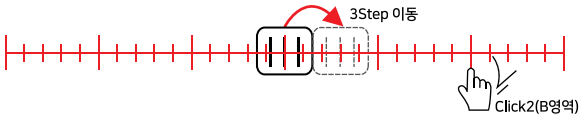
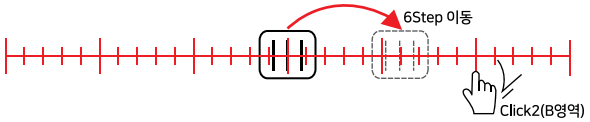
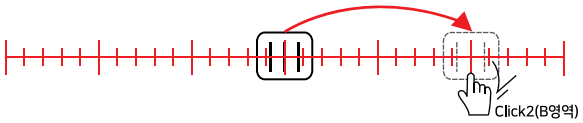
Step	30
SmallChangeStep	3
LargeChangeStep	6



Click 1 (버튼 영역 클릭)

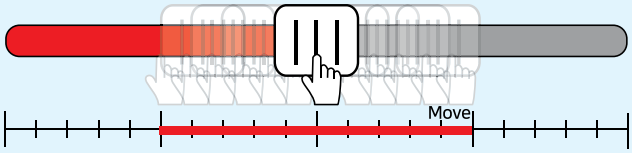
MoveOnClick 속성값에 상관없이 버튼 영역 클릭 시 설정된 SmallChangeStep만큼 이동합니다.



MoveOnClick 속성값	Click 2 (TrackBar 영역 클릭)
SmallStep	설정된 SmallChangeStep만큼 이동하며, A영역과 B영역 클릭 시 이동량이 항상 같습니다. 
LargeStep	설정된 LargeChangeStep만큼 이동합니다. 
Click	클릭한 위치에서 가장 가까운 Step으로 이동합니다. 

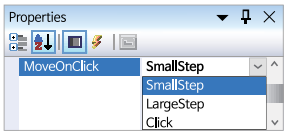
참고 IEC-Series 제품에서 Drag & Drop을 활성화한 경우 SmartTrackBar 이동 방식

Bar 버튼을 클릭한 상태로 움직인다면 SmallChangeStep만큼 이동합니다. 만약 Bar 버튼 영역을 제외한 구간을 클릭한다면 MoveOnClick 속성 설정값만큼 이동합니다.



참조 Drag & Drop 관련하여 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → Drag & Drop 관련"을 참조하시기 바랍니다.

사용법



프로퍼티(속성) Value

SmartTrackBar의 Bar 버튼 위치값을 읽거나 설정합니다.

- `double` : Bar 버튼 위치값으로 소수점값으로 설정 가능 (범위 : 0~100)

C# 사용법

```
smartTrackBar1.Value = 77.33;  // Bar 버튼 77.33% 위치로 이동
```

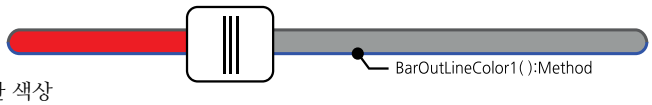
VB 사용법

```
smartTrackBar1.Value = 77.33
```

메소드(함수) BarOutLineColor1

TrackBar의 Bar 테두리 하단 색상 설정합니다. (기본값 : 흰색)

◉ void BarOutLineColor1(Color cColor)



[인자]

Color cColor : TrackBar의 Bar 테두리 하단 색상

C# 사용법

```
smartTrackBar1.BarOutLineColor1(Color.Blue);  // Bar 테두리의 하단 색상을 Blue로 변경
```

VB 사용법

```
smartTrackBar1.BarOutLineColor1(Color.Blue)
```

이벤트 ValueChanged

Value 속성이 변경될 경우 이벤트를 발생합니다.

C# 사용법

```
private void smartTrackBar1_ValueChanged(object sender, EventArgs e)
{
  labX1.Text = smartTrackBar1.Value.ToString( " 0 " );
}
```

VB 사용법

```
Private Sub smartTrackBar1_ValueChanged(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartTrackBar1.ValueChanged
  labX1.Text = smartTrackBar1.Value.ToString( " 0 " )
End Sub
```

4) SmartTrackBar 예제 사용하기

SmartTrackBar를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartTrackBar"

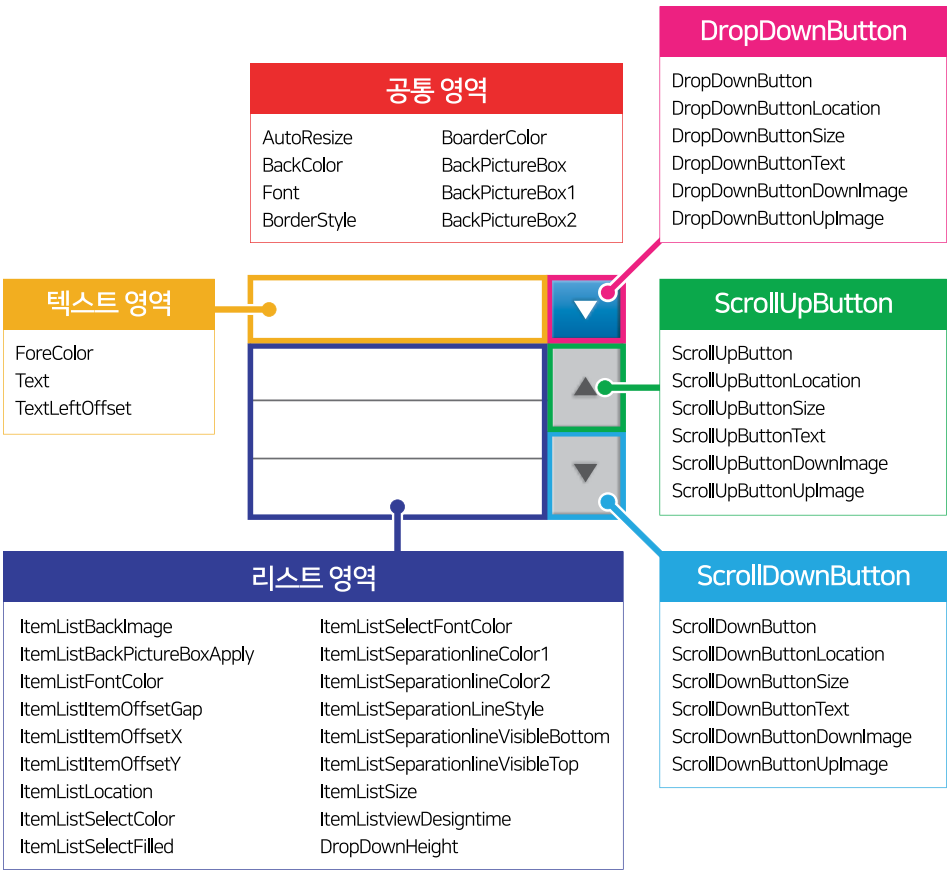
7. SmartComboBox

SmartComboBox는 .NET Compact Framework에서 지원되는 ComboBox 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 ComboBox의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 배경 투명 처리 효과 및 스킨 처리
- 컨트롤 Full 리사이징 및 버튼 크기 조절
- 다양한 스타일을 적용할 수 있도록 모든 디자인 요소 색상 및 형태 변경 기능

참고 ItemList는 SmartListBox와 동일한 인터페이스(속성, 메소드)를 제공합니다. "SmartListBox 컴포넌트" 내용을 참고하시기 바랍니다.

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartComboBox 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartComboBox에서 이미지는 버튼들(DropDownButton, ScrollDownButton, ScrollUpButton)과 리스트(ItemList) 배경에 사용됩니다.

STEP-1 크기 설정하기

SmartComboBox의 크기를 설정하기 전 각종 디자인 요소의 위치 및 크기가 자동으로 맞춰지도록 AutoResize 속성을 True로 설정한 후 SmartComboBox를 원하는 크기로 설정합니다. 크기 설정이 완료되면 각종 디자인 요소를 원하는 크기 및 위치로 설정하기 위해 AutoResize 속성을 반드시 False로 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
AutoResize	True → False	Size	173, 41

※ 자세한 내용은 위 표의 관련 속성 내용을 참고하시기 바랍니다.

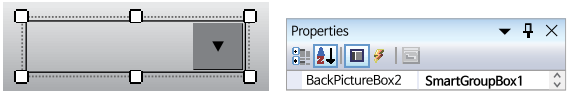
STEP-2 배경 설정하기

SmartComboBox의 배경은 BackColor 속성을 사용해 색상으로 설정하는 방법과, BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있습니다.

※ 자세한 내용은 "BackColor, BackPictureBox 속성" 내용을 참고하시기 바랍니다.

[CASE-1] 색상으로 설정하는 경우

[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우



STEP-3 테두리 스타일 설정하기

SmartComboBox는 None, FixedSingle, FixedSingle1, FixedSingle2 총 4가지의 테두리 스타일을 지원하며, BorderStyle 속성에서 스타일을 변경할 수 있습니다. 또한, BorderColor 속성에서 테두리의 색상을 설정할 수 있습니다.

[표] 관련 속성의 속성값 예시

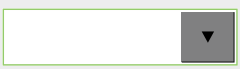
관련 속성	속성값	관련 속성	속성값
BorderStyle	FixedSingle1	BorderColor	LimeGreen

※ 자세한 내용은 위 표의 관련 속성 내용을 참고하시기 바랍니다.

[표] BorderStyle 속성값에 따른 테두리 스타일

None	FixedSingle	FixedSingle1	FixedSingle2
IEC266-Series IEC667-Series IEC1000-Series	IEC266-Series IEC667-Series IEC1000-Series	IEC266-Series IEC667-Series IEC1000-Series	IEC266-Series IEC667-Series IEC1000-Series

※ BorderColor = Red



[출력 화면]

STEP-4 텍스트 영역 설정하기

SmartComboBox의 텍스트 영역의 크기는 STEP-1에서 설정할 수 있습니다. 출력되는 텍스트는 Font 속성으로 폰트를 설정할 수 있으며, ForeColor로 색상을, Text 속성으로 초기에 출력되는 텍스트를 코드상에서 임의로 변경할 수 있습니다. 또한, TextLeftOffset 속성으로 텍스트가 출력되는 X축의 좌표를 설정할 수 있습니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
Font	Arial, 10pt	ForeColor	Black
Text	IEC-Series	TextLeftOffset	3

※ 자세한 내용은 위 표의 관련 속성 내용을 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  smartComboBox1.Text = " IEC-Series ";  // 프로그램 배포 시 SmartComboBox의 텍스트를 설정
}
```

[출력 화면]

STEP-5 DropDownButton 디자인하기

DropDownButton은 DropDownButtonLocation 속성으로 위치를, DropDownButtonSize 속성으로 크기를 설정할 수 있습니다. 그리고 DropDownButton은 이미지를 적용할 수 있으며 이미지를 사용하는 경우 STEP-2에서 BackPictureBox를 적용해 배경 이미지를 투영했다면, 버튼 이미지에는 반드시 마스킹 처리를 해야 합니다. 또한 DropDownButtonText 속성으로 버튼에 표시되는 텍스트를 설정할 수 있습니다. 이때 버튼에 적용한 이미지에 문자가 표현되어있다면 텍스트가 겹치는 현상을 방지하기 위해 DropDownButtonText 속성값을 “”(공백)으로 설정하셔야 합니다. 버튼 이미지를 사용하지 않는 경우 DropDownButton 속성에 접근 후 ButtonColor 속성을 이용해 버튼의 색상을 변경할 수 있습니다.

※ 자세한 내용은 아래 CASE에서 사용한 속성 내용을 참고하시기 바랍니다.

[CASE-1] 이미지를 사용하지 않는 경우

1. DropDownButtonSize 속성으로 버튼 크기를 조절합니다.
2. DropDownButtonLocation 속성으로 버튼 위치를 조절합니다.
3. DropDownButton 속성 내부의 ButtonColor 속성에서 버튼의 색상을 설정합니다.

주의 디자이너 속성에서 DropDownButton 색상 설정 시 주의사항

디자이너 속성에서 DropDownButton의 색상 설정 시 프로젝트를 닫으면 색상 값이 초기화되므로, 디자이너에서는 적용되는 색상만 확인하고 실제 색상 적용은 코드상에서 하시기 바랍니다.

[C#] 예시 코드

```
private void Form1_Load(object sender, EventArgs e)
{
  // 프로그램 배포 시 DropDownButton의 색상을 설정
  smartComboBox1.DropDownButton.ButtonColor = Color.YellowGreen;
}
```

4. DropDownButtonText 속성에서 버튼에 표현할 텍스트를 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
DropDownButtonSize	43, 38	ButtonColor	YellowGreen
DropDownButtonLocation	128, 1	DropDownButtonText	Show

[출력 화면]

[CASE-2] 이미지를 사용하는 경우

1. DropDownButtonDownImage, DropDownButtonUpImage 적용하기

※ 본 가이드에서는 BackPictureBox를 적용하여 사용할 버튼 이미지에 마스킹 처리를 하였습니다.

참고 DropDownButton에 이미지를 적용하는 경우, 버튼의 크기는 사용한 이미지의 크기에 맞게 자동으로 조절되어 DropDownButtonSize 속성을 조절할 필요가 없습니다.

2. DropDownButtonLocation 속성으로 버튼 위치를 조절합니다.
3. DropDownButtonText 속성에서 버튼에 표현할 텍스트를 설정합니다.

※ 본 가이드에서는 버튼 이미지에 텍스트가 표현되어 공백으로 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
DropDownButtonDownImage		DropDownButtonLocation	128, 1
DropDownButtonUpImage		DropDownButtonText	" "(공백)

[출력 화면]

STEP-6 리스트 영역 디자인하기

SmartComboBox의 리스트 영역을 디자인하기 전 ItemListviewDesignTime 속성을 True로 하여 디자이너 화면에서 디자인을 할 수 있도록 합니다. 리스트가 화면상에 보이면 다양한 속성을 사용해 리스트를 디자인할 수 있습니다. 리스트 영역도 배경에 이미지를 적용할 수 있으며, 속성 설정 순서는 다음과 같습니다.

※ 배경에 이미지를 사용하는 경우 ItemListBackImage 속성을 사용해 이미지를 설정합니다.
1. ItemListSeparationLineStyle 속성으로 리스트 영역의 테두리 스타일을 설정합니다.
2. ItemListSeparationlineColor 속성으로 테두리의 색상을 설정합니다.
※ ItemListSeparationLineStyle 속성값이 Fixed3D인 경우 ItemListSeparationlineColor2 속성으로 2번째 테두리의 색상을 설정합니다.
3. ItemListSeparationlineVisibleTop/Bottom 속성으로 리스트의 위/아래 테두리의 숨김 여부를 설정합니다.
4. DropDownHeight 속성으로 출력되는 리스트가 보여질 영역의 크기를 설정합니다.
5. ItemListSize 속성으로 리스트의 크기를 설정합니다.
6. ItemListLocation 속성으로 리스트의 출력 위치를 설정합니다.
7. ItemListItemOffsetGap 속성으로 리스트 각 항목의 높이를 설정합니다.
8. ItemListItemOffsetX/Y 속성으로 리스트 항목이 표시되는 위치를 설정합니다.
9. ItemListFontColor 속성으로 리스트 항목의 텍스트 색상을 설정합니다.
10. ItemListSelectFontColor 속성으로 선택한 리스트 항목의 텍스트 색상을 설정합니다.
11. ItemListSelectFilled 속성으로 선택한 리스트 항목의 색 채움 여부를 설정합니다.
12. ItemListSelectColor 속성으로 선택한 리스트 항목이 채워지는 색상을 설정합니다.
※ ItemListSelectFilled 속성이 False인 경우, ItemListSelectColor 속성에서 설정한 색상으로 선택한 리스트 항목의 테두리만 표시됩니다.

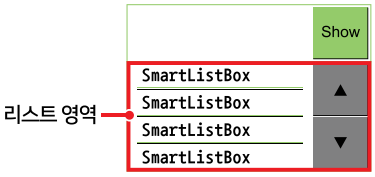
※ 자세한 내용은 " ItemListviewDesignTime 속성"과 다음 CASE의 관련 속성의 내용을 참고하시기 바랍니다

[CASE-1] 이미지를 사용하지 않는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
ItemListSeparationLineStyle	Fixed3D	ItemListItemOffsetGap	10

ItemListSeparationlineColor1	LimeGreen	ItemListItemOffsetX	3
ItemListSeparationlineColor2	Black	ItemListItemOffsetY	0
ItemListSeparationlineVisibleBottom	False	ItemListFontColor	Black
ItemListSeparationlineVisibleTop	False	ItemListSelectFilled	True
DropDownHeight	95	ItemListSelectColor	LimeGreen
ItemListSize	130, 93	ItemListSelectFontColor	White
ItemListLocation	2, 41	-	-

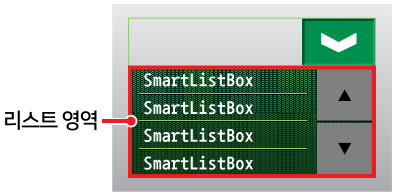


[출력 화면]

[CASE-2] 이미지를 사용하는 경우

[표] 관련 속성의 속성값 예시

ItemListBackImage(배경 이미지)			
관련 속성	속성값	관련 속성	속성값
ItemListSeparationLineStyle	Fixed3D	ItemListItemOffsetGap	11
ItemListSeparationlineColor1	LimeGreen	ItemListItemOffsetX	3
ItemListSeparationlineColor2	Black	ItemListItemOffsetY	0
ItemListSeparationlineVisibleBottom	False	ItemListFontColor	White
ItemListSeparationlineVisibleTop	False	ItemListSelectFilled	False
DropDownHeight	95	ItemListSelectColor	LimeGreen
ItemListSize	168, 95	ItemListSelectFontColor	Black
ItemListLocation	1, 33	-	-



[출력 화면]

STEP-7 ScrollUp/DownButton 디자인하기

ScrollUp/DownButton을 디자인하는 방법은 DropDownButton을 디자인하는 방법과 크게 다르지 않으므로 STEP-5 내용을 참고하시기 바랍니다. 단, ScrollUp/DownButton은 리스트 영역에 있으므로 디자인 전 ListviewDesignTime 속성을 True로 하여 디자이너 화면에서 디자인을 할 수 있도록 합니다.

※ 자세한 내용은 표의 속성 내용을 참고하시기 바랍니다.

[CASE-1] 이미지를 사용하지 않는 경우

1. ScrollDownButtonSize, ScrollUpButtonSize 속성으로 버튼 크기를 조절합니다.
2. ScrollDownButtonLocation, ScrollUpButtonLocation 속성으로 버튼 위치를 조절합니다.
3. ScrollDownButton, ScrollUpButton 속성 내부의 ButtonColor 속성에서 버튼의 색상을 설정합니다.

주의 디자이너 속성에서 ScrollDown/UpButton 색상 설정 시 주의사항

디자이너 속성에서 ScrollDown/UpButton의 색상 설정 시 프로젝트를 닫으면 색상값이 초기화되므로, 디자이너에서는 적용되는 색상만 확인하고 실제 색상 적용은 코드상에서 하시기 바랍니다.

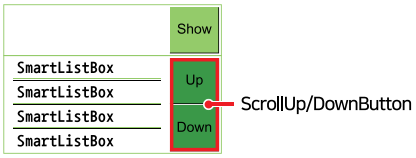
[C#] 예시 코드

```
private void Form1_Load(object sender, EventArgs e)
{
  // 프로그램 배포 시 ScrollDownButton의 색상을 설정
  smartComboBox1.ScrollDownButton.ButtonColor = Color.DarkOrange;
  // 프로그램 배포 시 ScrollUpButton의 색상을 설정
  smartComboBox1.ScrollUpButton.ButtonColor = Color.DarkOrange;
}
```

4. DropDownButtonText 속성에서 버튼에 표현할 텍스트를 설정합니다.

[표] 관련 속성의 속성값 예시

ScrollDownButton 속성	속성값	ScrollUpButton 속성	속성값
ScrollDownButtonSize	41, 46	ScrollUpButtonSize	41, 45
ScrollDownButtonLocation	130, 87	ScrollUpButtonLocation	130, 42
ScrollDownButtonText	Down	ScrollUpButtonText	Up
ButtonColor	ForestGreen	ButtonColor	ForestGreen



[출력 화면]

[CASE-2] 이미지를 사용하는 경우

1. ScrollDown/UpButtonDownImage 속성으로 눌린 이미지를, ScrollDown/UpButtonUpImage 속성으로 눌리지 않은 이미지를 설정합니다.

※ 본 가이드에서는 리스트에 BackPictureBox를 적용하지 않아 버튼 이미지에 마스킹 처리를 하지 않았습니다.

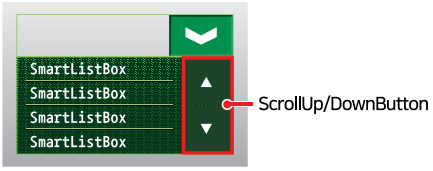
참고 ScrollDown/UpButton에 이미지를 적용하는 경우, 버튼의 크기는 사용한 이미지의 크기에 맞게 자동으로 조절되어 ScrollDown/UpButtonSize 속성을 조절할 필요가 없습니다.

2. ScrollDownButtonLocation, ScrollUpButtonLocation 속성으로 버튼 위치를 조절합니다.
3. ScrollDownButtonText, ScrollUpButtonText 속성에서 버튼에 표현할 텍스트를 설정합니다.

※ 본 가이드에서는 버튼 이미지에 텍스트가 표현되어 공백으로 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
ScrollDownButtonDownImage		ScrollUpButtonDownImage	
ScrollDownButtonUpImage		ScrollUpButtonUpImage	
ScrollDownButtonSize	51, 49 (자동 설정)	ScrollUpButtonSize	51, 47 (자동 설정)
ScrollDownButtonLocation	119, 81	ScrollUpButtonLocation	119, 33
ScrollDownButtonText	“”(공백)	ScrollUpButtonText	“”(공백)



[출력 화면]

3) SmartComboBox 인터페이스 설명

SmartComboBox Component Interface		
속성		
AutoResize : bool	BackColor : Color	BackPictureBox : PictureBox
BackPictureBox1 : SmartInnerForm	BackPictureBox2 : SmartGroupBox	BoarderColor : Color
BorderStyle : SmartComboBox.BOARDERSTYLE	DropDownButton : SmartButton	ScrollDownButton : SmartButton
ScrollUpButton : SmartButton	DropDownButtonDownImage : Image	DropDownButtonLocation : Point
DropDownButtonSize : Size	DropDownButtonText : string	DropDownButtonUpImage : Image
DropDownHeight : int	Font : Font	ForeColor : Color
InitVisible : bool	ItemList : SmartListBox	ItemListBackImage : Image
ItemListBackPictureBoxApply : bool	ItemListFontColor : Color	ItemListItemOffsetGap : int
ItemListItemOffsetX : int	ItemListItemOffsetY : int	ItemListLocation : Point
ItemListSelectColor : Color	ItemListSelectFilled : bool	ItemListSelectFontColor : Color
ItemListSelectItemIndex : int	ItemListSeparationlineColor1 : Color	ItemListSeparationlineColor2 : Color
ItemListSeparationLineStyle : SmartListBox.SEPARATIONLINETYPES	ItemListSeparationlineVisibleBottom : bool	ItemListSeparationlineVisibleTop : bool
ItemListSize : Size	ItemListviewDesigntime : bool	ScrollDownButtonDownImage : Image
ScrollDownButtonLocation : Point	ScrollDownButtonSize : Size	ScrollDownButtonText : string
ScrollDownButtonUpImage : Image	ScrollUpButtonDownImage : Image	ScrollUpButtonLocation : Point
ScrollUpButtonSize : Size	ScrollUpButtonText : string	ScrollUpButtonUpImage : Image
Text : string	TextLeftOffset : int	
이벤트		
SelectedIndexChanged : EventHandler		

프로퍼티(속성) AutoResize

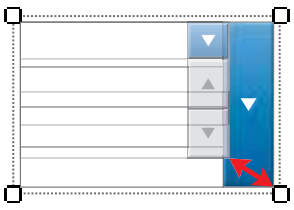
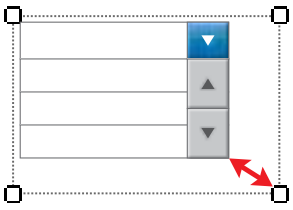
SmartComboBox의 크기 변경 시 자동 리사이징(Resizing) 여부를 설정합니다. 즉, SmartComboBox의 크기를 변경하면 리스트와 버튼들의 위치 및 크기가 자동으로 변경됩니다.

참고 AutoResize 속성 사용 방법

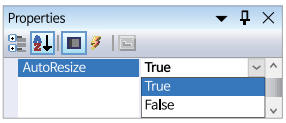
AutoResize 속성을 사용하면 SmartComboBox의 레이아웃 디자인을 좀 더 편리하게 할 수 있습니다.

※ 자세한 디자인 적용 방법은 "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

- bool : 자동 리사이징 여부
  - true : 적용
  - false : 미적용

true	false
	

사용법



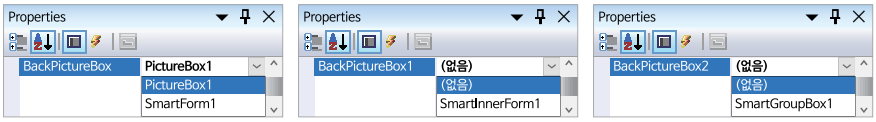
프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartComboBox의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartComboBox에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

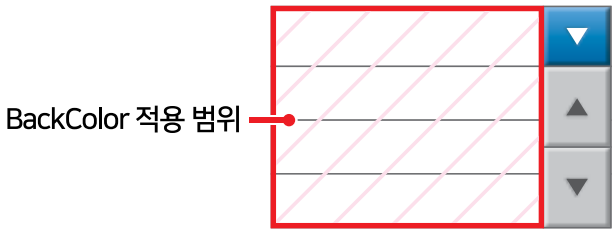
주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 Back Color 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법



프로퍼티(속성) BackColor

SmartComboBox의 배경 색상을 설정합니다.

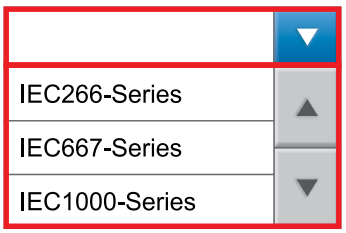


사용법

프로퍼티(속성) BorderColor

SmartComboBox의 테두리 색상을 설정합니다.

참고 BorderStyle 속성에서 지정된 테두리 스타일에 따라 적용되는 범위가 다르며, 자세한 내용은 “BorderStyle 속성” 내용을 참고하시기 바랍니다.



사용법

프로퍼티(속성) BorderStyle

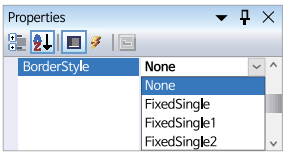
SmartComboBox의 테두리 스타일을 설정합니다.

- SmartComboBox.BOARDERSTYLE.None : 테두리 없음
- SmartComboBox.BOARDERSTYLE.FixedSingle : 텍스트 부분에만 적용
- SmartComboBox.BOARDERSTYLE.FixedSingle1 : 외곽선 및 텍스트 부분과 리스트 부분 사이에 적용
- SmartComboBox.BOARDERSTYLE.FixedSingle2 : 외곽선에만 적용

[표] BorderStyle 속성값에 따른 테두리 스타일

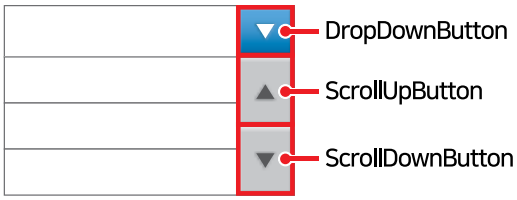
None	FixedSingle	FixedSingle1	FixedSingle2
IEC266-Series IEC667-Series IEC1000-Series	IEC266-Series IEC667-Series IEC1000-Series	IEC266-Series IEC667-Series IEC1000-Series	IEC266-Series IEC667-Series IEC1000-Series

사용법



프로퍼티(속성) DropDownButton, ScrollUpButton, ScrollDownButton

SmartUpDown에 사용된 버튼들(DropDownButton, ScrollUpButton, ScrollDownButton)의 속성에 접근할 수 있습니다. 디자인과 관련된 주요 속성은 SmartComboBox에서 직접 접근할 수 있게 설계되어 있으며, 그 외의 속성에 접근하는데 사용됩니다.

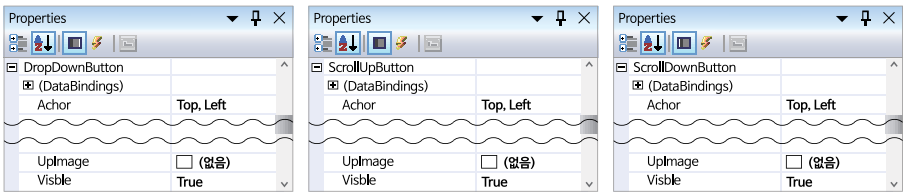


[표] SmartComboBox에서 직접 접근 가능한 버튼들의 속성

버튼	DropDownButton	ScrollUpButton	ScrollDownButton
속성	DropDownButtonLocation	ScrollUpButtonLocation	ScrollDownButtonLocation
	DropDownButtonSize	ScrollUpButtonSize	ScrollDownButtonSize
	DropDownButtonText	ScrollUpButtonText	ScrollDownButtonText
	DropDownButtonDownImage	ScrollUpButtonDownImage	ScrollDownButtonDownImage
	DropDownButtonUpImage	ScrollUpButtonUpImage	ScrollDownButtonUpImage

참고 각 속성의 자세한 내용은 위 표에 표시된 속성의 내용을 참고하시기 바랍니다.

사용법



프로퍼티(속성) DropDownButtonDownImage, DropDownButtonUpImage

Masking Image

SmartComboBox의 리스트 출력 버튼(DropDownButton)의 Up/Down 이미지를 설정합니다.

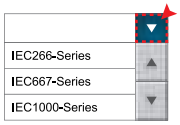
- DropDownButtonDownImage : DropDownButton이 Down된 이미지
- DropDownButtonUpImage : DropDownButton이 Up된 이미지

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

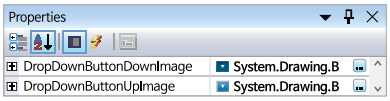
사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

[표] 속성에 따른 이미지 적용 모습

DropDownButtonDownImage	DropDownButtonUpImage
	
DropDownButton이 눌려져 리스트가 출력됨	DropDownButton이 눌리지 않아 리스트가 출력되지 않음

사용법

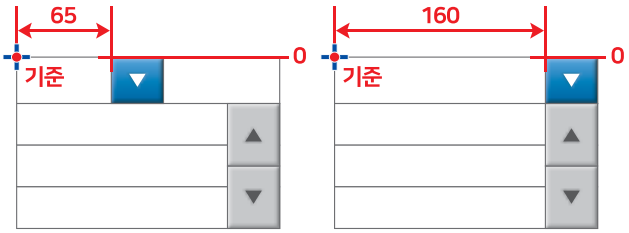


프로퍼티(속성) DropDownButtonLocation

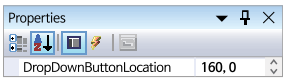
리스트 출력 버튼(DropDownButton)의 위치를 설정합니다. (기준점 : SmartComboBox의 좌측 상단)

중요 AutoResize 속성값을 False로 설정 후 사용하시기 바랍니다. 속성값이 True인 경우 위치 설정이 올바르게 적용되지 않을 수 있습니다.

참고 "AutoResize 속성" 내용을 참고하시기 바랍니다.



사용법

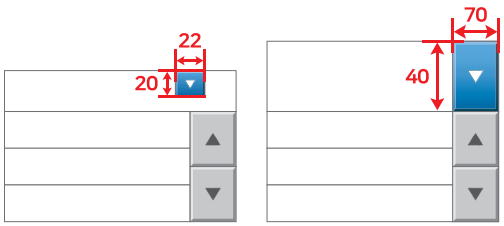


프로퍼티(속성) DropDownButtonSize

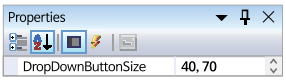
리스트 출력 버튼(DropDownButton)의 크기를 설정합니다

중요 AutoResize 속성값을 False로 설정 후 사용하시기 바랍니다. 속성값이 True인 경우 크기 설정이 올바르게 적용되지 않을 수 있습니다.

참고 "AutoResize 속성" 내용을 참고하시기 바랍니다.



사용법



프로퍼티(속성) DropDownButtonText

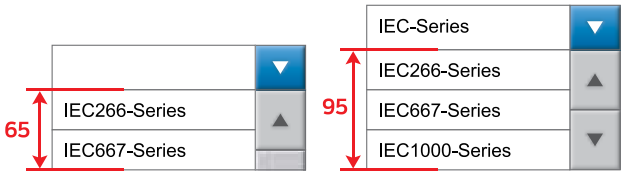
리스트 출력 버튼(DropDownButton)에 표시할 텍스트를 설정합니다.

참고 DropDownButtonDownImage, DropDownButtonUpImage 속성에 적용한 이미지에 텍스트가 포함되어 있는 경우, 텍스트가 겹치는 현상을 방지하기 위해 DropDownButtonText 속성값을 ""(공백)으로 설정하시기 바랍니다.

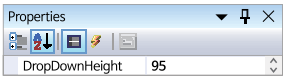
사용법

프로퍼티(속성) DropDownHeight

리스트 출력 버튼(DropDownButton) 클릭 시 하단으로 밀려 내려오는 리스트 영역의 높이를 설정합니다.

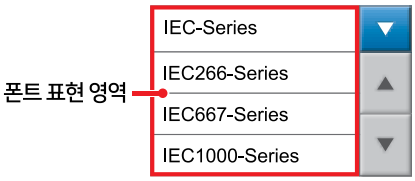


사용법



프로퍼티(속성) Font

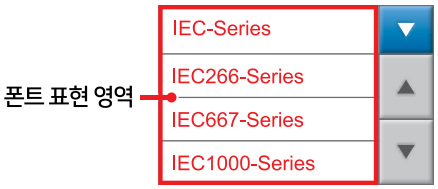
SmartComboBox에 표현되는 폰트를 설정합니다.



사용법

프로퍼티(속성) ForeColor

SmartComboBox에 표현되는 폰트의 색상을 설정합니다.



사용법

프로퍼티(속성) `InitVisible`

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리 할 수 있는 속성입니다. 공통 영역에 위치한 SmartComboBox의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

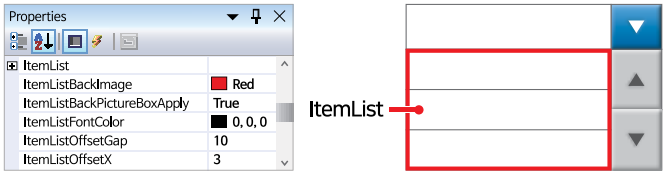
참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) `ItemList`

SmartComboBox의 리스트 부분은 SmartListBox의 기능을 적용하여 다양한 디자인 작업을 할 수 있도록 하였습니다. 일부 속성은 SmartComboBox에 배치하여 디자인 속성창에서 직접 접근이 가능하도록 하였으며, 직접 접근을 지원하지 않는 속성은 코드상에서 ItemList 속성을 사용해 접근이 가능합니다. SmartComboBox에서 직접 접근이 가능한 속성은 아래표와 같습니다.

참고 디자이너 속성창에서 보이지 않는 SmartListBox의 속성은 코드상에서 ItemList 속성에 접근해 사용 가능합니다. 즉, SmartComBoBox1.ItemList.XXX와 같이 접근이 가능합니다. ItemList 속성을 이용해 사용하는 속성에 대한 자세한 설명은 "SmartListBox 컴포넌트" 내용을 참고하시기 바랍니다.

표] SmartComboBox에서 직접 접근이 가능한 SmartListBox 속성

속성	ItemListFontColor, ItemListItemOffsetGap, ItemListItemOffsetX, ItemListItemOffsetY, ItemListLocation, ItemListSelectColor, ItemListSelectFilled, ItemListSelectFontColor, ItemListSelectItemIndex, ItemListSeperationlineColor1, ItemListSeperationlineColor2, ItemListSeperationLineStyle, ItemListSeperationlineVisibleBottom, ItemListSeperationlineVisibleTop, ItemlistSize
디자이너 속성창 및 표현 영역	

C# 사용법

```
smartComboBox1.ItemList.AddItem( " IEC-Series " );
```

VB 사용법

```
smartComboBox1.ItemList.AddItem( " IEC-Series " )
```

프로퍼티(속성) `ItemListBackImage` Image

리스트 영역의 배경 이미지를 설정합니다.

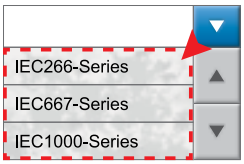
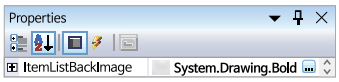


Image 이미지 사용 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 “이미지 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

사용법

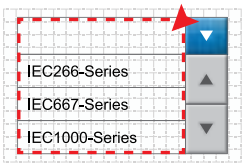
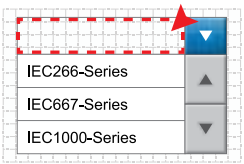


프로퍼티(속성) ItemListBackPictureBoxApply

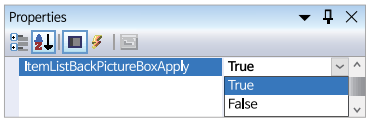
BackPictureBox 속성을 설정하여 SmartComboBox에 투명 처리를 적용한 경우, 리스트 영역의 투명처리 여부를 설정합니다. 즉, 리스트 영역을 아이템 영역과 다르게 디자인할 수 있습니다.

참고 ItemListBackPictureBoxApply 속성값을 False로 하는 경우 리스트 영역은 BackColor 속성으로 설정된 색상이 보여지며, ItemListBackImage 속성을 사용해 이미지가 적용되어 있다면 이미지가 보여집니다.

[표] ItemListBackPictureBoxApply 속성값에 따른 투명 처리 방법

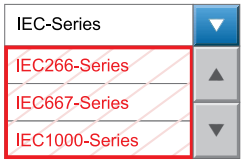
True	False
	
텍스트 부분과 리스트 부분 모두 투명 처리	텍스트 부분만 투명 처리

사용법



프로퍼티(속성) ItemListFontColor

리스트 영역에 출력되는 텍스트의 색상을 설정합니다.

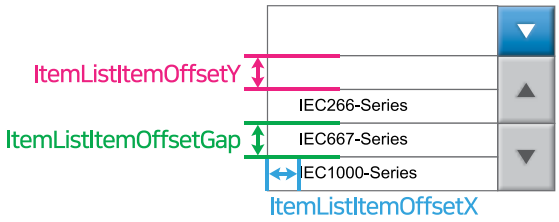


사용법

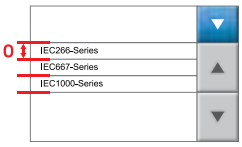
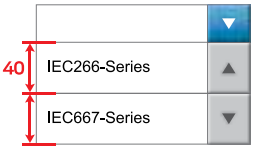
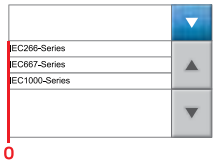
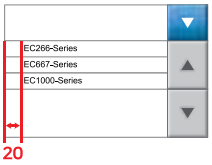
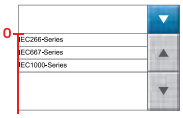
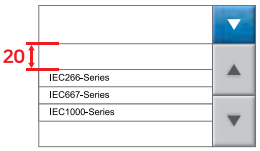
프로퍼티(속성) ItemListItemOffsetGap, ItemListItemOffsetX, ItemListItemOffsetY

- ItemListItemOffsetGap : 리스트에 추가된 각 항목의 높이를 설정합니다. (기본값 : 0(폰트 크기에 맞춤))
- ItemListItemOffsetX : 리스트의 X축 시작 위치를 설정합니다.(기본값 : 0)

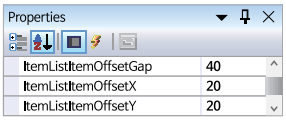
• ItemListItemOffsetY : 리스트의 Y축 시작 위치를 설정합니다.(기본값 : 0)



[표] 각 속성값에 따른 리스트 출력

ItemListItemOffsetGap = 0 (기본값)	ItemListItemOffsetGap = 40
	
ItemListItemOffsetX = 0 (기본값)	**ItemListItemOffsetX = 20**
	
ItemListItemOffsetY = 0 (기본값)	**ItemListItemOffsetY = 20**
	

사용법

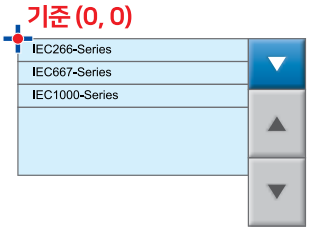
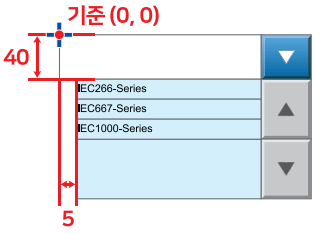


프로퍼티(속성) ItemListLocation

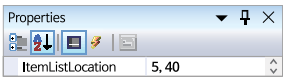
리스트의 출력 위치를 설정합니다.

참고 AutoResize 속성값이 True인 경우 위치가 자동으로 조절됩니다. 자세한 내용은 “AutoResize 속성” 내용을 참고하시기 바랍니다.

[표] ItemListLocation 속성값에 따른 리스트 출력 위치

ItemListLocation = 0, 0	ItemListLocation = 5, 40
	

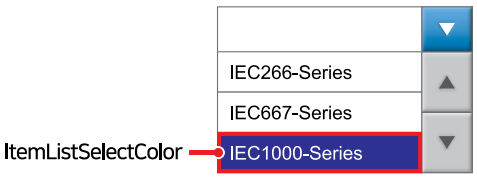
사용법



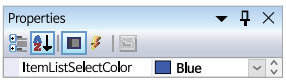
프로퍼티(속성) ItemListSelectColor

리스트에서 아이템 클릭 시 표시되는 색상을 설정합니다.

참고 ItemListSelectFilled 속성값이 False인 경우 클릭한 항목의 외곽선만 표시되며, 색상은 ItemListSelectColor 속성에서 설정한 색상이 표시됩니다. "ItemListSelectFilled 속성" 내용을 참고하시기 바랍니다.



사용법

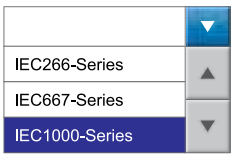
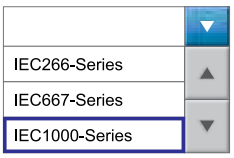


프로퍼티(속성) ItemListSelectFilled

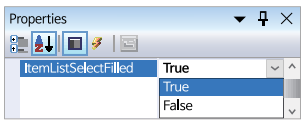
리스트에서 클릭한 아이템의 색 채움 여부를 설정합니다.

참고 ItemListSelectFilled 속성값이 False인 경우 클릭한 항목의 외곽선만 표시되며, 색상은 ItemListSelectColor 속성에서 설정한 색상이 표시됩니다. "ItemListSelectColor 속성" 내용을 참고하시기 바랍니다.

[표] ItemListSelectFilled 속성값에 따른 출력 화면

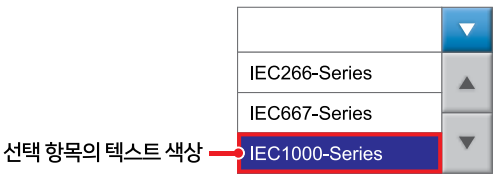
True	False
	

사용법



프로퍼티(속성) ItemListSelectFontColor

리스트에서 클릭한 아이템의 텍스트 색상을 설정합니다.



사용법

프로퍼티(속성) ItemListSelectItemIndex

리스트에서 클릭된 아이템의 인덱스를 얻거나, 임의로 인덱스를 지정하여 리스트에 추가된 아이템을 선택합니다.

참고 ItemListSelectItemIndex 속성값 설정 시 SelectedIndexChanged 이벤트가 발생합니다. 자세한 내용은 "SelectedIndexChanged 이벤트" 내용을 참고하시기 바랍니다.

- `int` : 리스트에서 선택되거나 선택할 아이템의 인덱스(선택 아이템이 없는 경우 : -1)

C# 사용법

```
// smartComboBox1에 추가된 아이템 중 임의로 3번째(인덱스 = 2) 아이템을 선택
smartComboBox1.ItemListSelectItemIndex = 2;

// ItemListSelectItemIndex 속성값이 변경되어 이벤트 발생
private void smartComboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  SmartMessageBox.Show(smartComboBox1.ItemListSelectItemIndex.ToString());
}
```

VB 사용법

```
smartComboBox1.ItemListSelectItemIndex = 2

Private Sub smartComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles smartComboBox1.SelectedIndexChanged
  SmartMessageBox.Show(smartComboBox1.ItemListSelectItemIndex.ToString())
End Sub
```

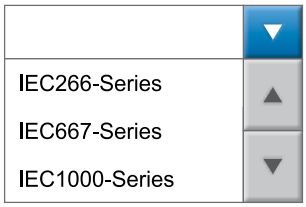
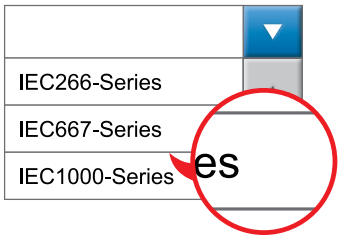
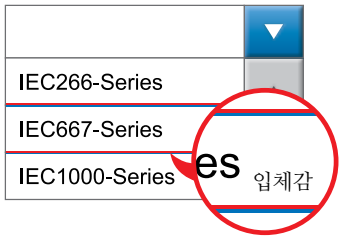
프로퍼티(속성) ItemListSeparationLineStyle

리스트에 추가된 아이템의 구분선 스타일을 설정합니다.

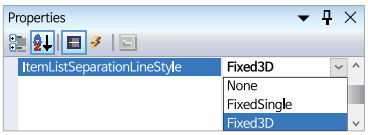
참고 구분선의 색상은 ItemListSeparationlineColor1, ItemListSeparationlineColor2 속성으로 설정하며, 자세한 내용은 "ItemListSeparationlineColor1, ItemListSeparationlineColor2 속성" 내용을 참고하시기 바랍니다.

- `SmartComboBox.ItemListSeparationLineStyle.None` : 리스트 구분선 없음
- `SmartComboBox.ItemListSeparationLineStyle.FixedSingle` : 리스트 구분선을 한 줄로 표시
- `SmartComboBox.ItemListSeparationLineStyle.Fixed3D` : 리스트 구분선을 두 줄로 표시

[표] ItemListSeparationLineStyle 속성값에 따른 리스트 출력 화면

None (선 없음)	FixedSingle (한 줄)	Fixed3D (두 줄)
		

사용법



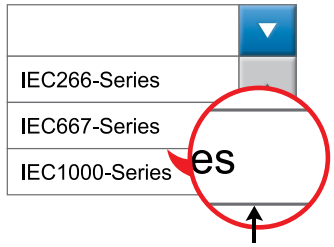
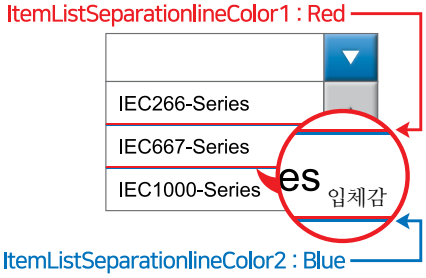
프로퍼티(속성) ItemListSeparationlineColor1, ItemListSeparationlineColor2

- `ItemListSeparationlineColor1` : 리스트의 첫번째 구분선 색상을 설정합니다.

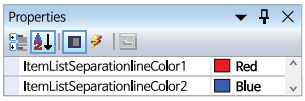
• ItemListSeparationlineColor2 : ItemListSeparationLineStyle 속성값이 Fixed3D인 경우 리스트의 두번째 구분선 색상을 설정합니다.

참고 "ItemListSeparationLineStyle 속성" 내용을 참고하시기 바랍니다.

[표] ItemListSeparationLineColor 1, 2 속성값에 따른 색상 적용 예시

FixedSingle	FixedSingle3D
	

사용법

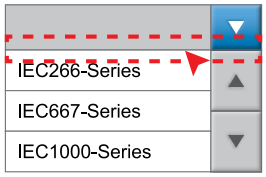
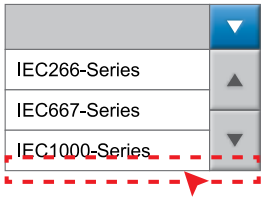
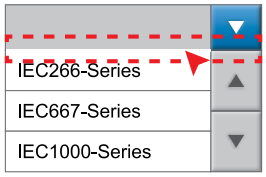
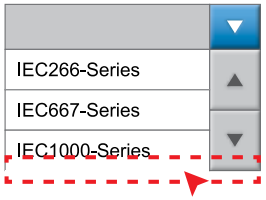


프로퍼티(속성) ItemListSeparationlineVisibleBottom, ItemListSeparationlineVisibleTop

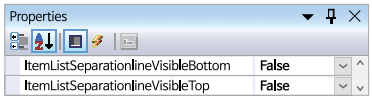
• ItemListSeparationlineVisibleBottom : 리스트의 하단 구분선 표시 여부를 설정합니다. (기본값 : True)

• ItemListSeparationlineVisibleTop : 리스트의 상단 구분선 표시 여부를 설정합니다. (기본값 : True)

[표] 속성값에 따른 출력 예시

속성	ItemListSeparationlineVisibleTop	ItemListSeparationlineVisibleBottom
True		
False		

사용법

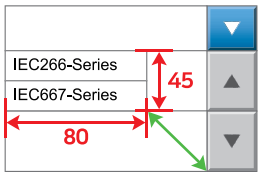
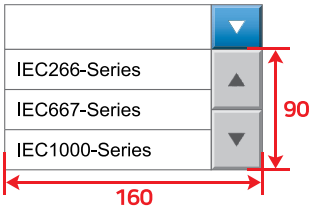


프로퍼티(속성) ItemListSize

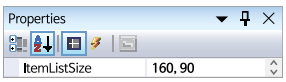
리스트의 크기를 설정합니다.

참고 AutoResize 속성값이 True인 경우 크기가 자동으로 조절됩니다. 따라서 리스트 크기 설정 시 AutoResize 속성값을 False로 설정하시기 바랍니다. 자세한 내용은 "AutoResize 속성" 내용을 참고하시기 바랍니다.

[표] ItemListSize 속성값 변경에 따른 리스트 크기 변화

ItemListSize = 80, 45	ItemListSize = 160, 90
	

사용법

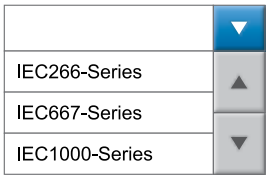


프로퍼티(속성) ItemListviewDesigntime

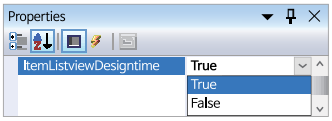
Visual Studio의 디자인 모드에서 리스트 영역의 표시 여부를 설정합니다. 리스트와 ScrollDownButton, ScrollUpButton의 세밀한 디자인이 필요한 경우 True로 설정하여 편리하게 디자인하실 수 있습니다.

참고 ItemListviewDesigntime 속성을 True로 설정 후 IEC-Series에 배포 시 False 형태로 배포됩니다.

[표] ItemListviewDesigntime 속성값에 따른 디자인 모드에서 출력 화면

True	False
	

사용법



프로퍼티(속성) ScrollDownButtonDownImage, ScrollDownButtonUpImage

Masking Image

ScrollDownButton의 Down/Up 이미지를 설정합니다.

- ScrollDownButtonDownImage : ScrollDownButton이 Down된 이미지
- ScrollDownButtonUpImage : ScrollDownButton이 Up된 이미지

중요 ItemListBackPictureBoxApply 속성을 True로 설정하여 배경 이미지를 투영시키는 경우, 이미지 마스킹 처리가 필요하며, 반대로 False인 경우 마스킹 처리를 하지마시기 바랍니다.

[표] 속성에 따른 이미지 적용 화면

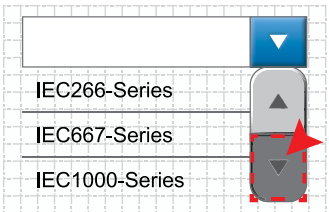
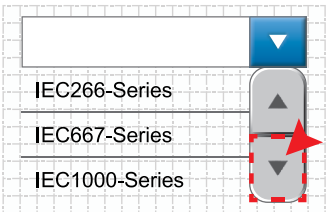
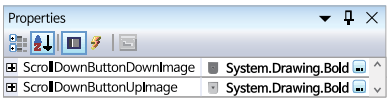
ScrollDownButtonDownImage	ScrollDownButtonUpImage
	

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) ScrollUpButtonDownImage, ScrollUpButtonUpImage

Masking Image

ScrollUpButton의 Down/Up 이미지를 설정합니다.

- ScrollUpButtonDownImage : ScrollUpButton이 Down된 이미지
- ScrollUpButtonUpImage : ScrollUpButton이 Up된 이미지

중요 ItemListBackPictureBoxApply 속성을 True로 설정하여 배경 이미지를 투영시키는 경우, 이미지 마스킹 처리가 필요하며, 반대로 False인 경우 마스킹 처리를 하지 마시기 바랍니다.

[표] 속성에 따른 이미지 적용 화면

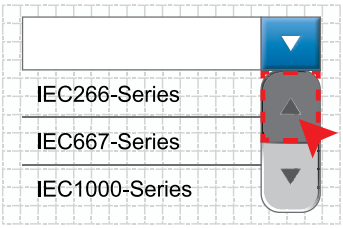
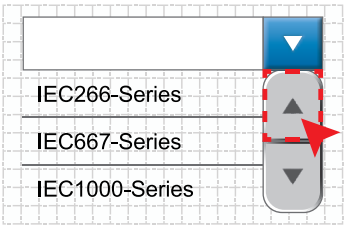
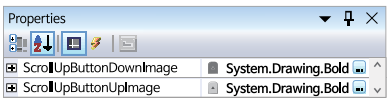
ScrollUpButtonDownImage	ScrollUpButtonUpImage
	

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) ScrollDownButtonLocation, ScrollUpButtonLocation

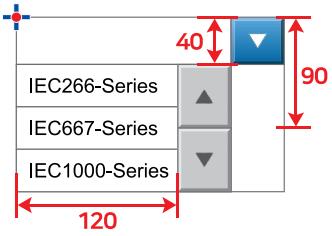
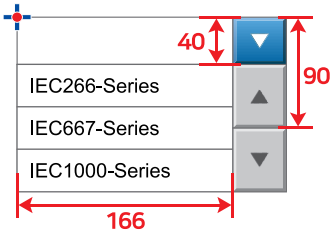
ScrollDownButton과 ScrollUpButton의 위치를 설정합니다.(기준점 : SmartComboBox의 좌측 상단)

- ScrollDownButtonLocation : ScrollDownButton의 위치를 설정합니다.
- ScrollUpButtonLocation : ScrollUpButton의 위치를 설정합니다.

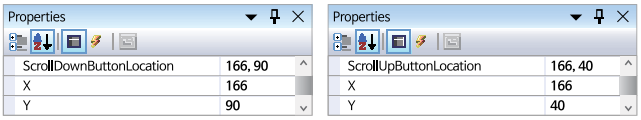
참고 AutoResize 속성값이 True인 경우 위치가 자동으로 조절됩니다. 따라서 디자인 시 AutoResize 속성값을 False로 설정하시기 바랍니다.

※ 자세한 내용은 "AutoResize 속성" 내용을 참고하시기 바랍니다.

[표] 속성값에 따른 각 버튼의 위치 변화

ScrollUpButtonLocation = 120, 40 ScrollDownButtonLocation = 120, 90	ScrollUpButtonLocation = 166, 40 ScrollDownButtonLocation = 166, 90
	

사용법



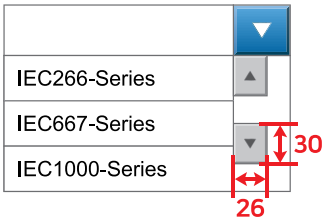
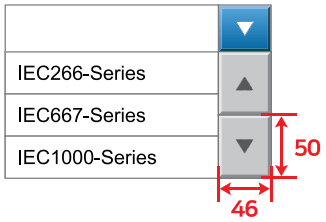
프로퍼티(속성) ScrollDownButtonSize, ScrollUpButtonSize

ScrollDownButton과 ScrollUpButton의 크기를 설정합니다.

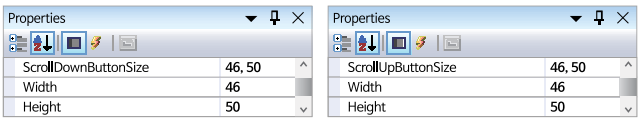
- `ScrollDownButtonSize` : ScrollDownButton의 크기를 설정합니다.
- `ScrollUpButtonSize` : ScrollUpButton의 크기를 설정합니다.

참고 AutoResize 속성값이 True인 경우 크기가 자동으로 조절됩니다. 따라서 디자인 시 AutoResize 속성값을 False로 설정하시기 바랍니다. 자세한 내용은 "AutoResize 속성" 내용을 참고하시기 바랍니다.

[표] 속성값에 따른 각 버튼의 크기 변화

ScrollUpButtonSize = 26, 30 ScrollDownButtonSize = 26, 30	ScrollUpButtonSize = 46, 50 ScrollDownButtonSize = 46, 50
	

사용법

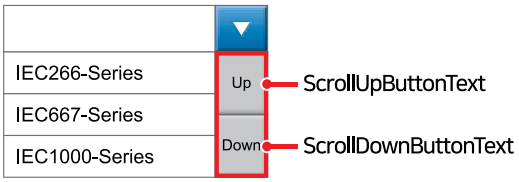


프로퍼티(속성) ScrollDownButtonText, ScrollUpButtonText

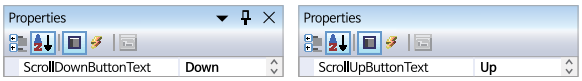
ScrollDownButton과 ScrollUpButton에 표시되는 텍스트를 설정합니다.

- `ScrollDownButtonText` : ScrollDownButton에 표시되는 텍스트를 설정합니다.
- `ScrollUpButtonText` : ScrollUpButton에 표시되는 텍스트를 설정합니다.

참고 ScrollDownButton과 ScrollUpButton에 적용된 이미지에 텍스트가 포함되어있는 경우, 텍스트가 겹치는 현상을 방지하기 위해 ScrollDownButtonText, ScrollUpButtonText 속성값을 “”(공백)으로 설정하시기 바랍니다.

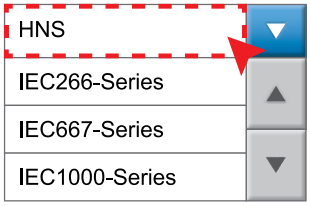


사용법



프로퍼티(속성) Text

SmartComboBox의 텍스트 부분에 표시되는 텍스트를 설정하거나 읽어옵니다.



C# 사용법

```
smartComboBox1.Text = “HNS”;
```

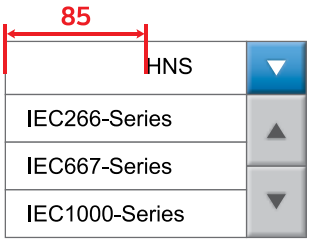
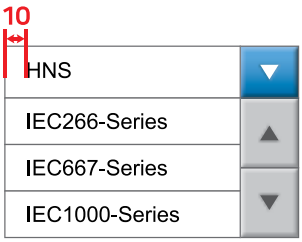
VB 사용법

```
smartComboBox1.Text = “HNS”
```

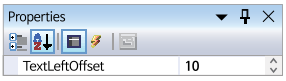
프로퍼티(속성) TextLeftOffset

텍스트 부분에 표시되는 텍스트의 표시 위치를 설정합니다. (기준점 : SmartComboBox의 좌측 구분선)

[표] TextLeftOffset 속성값에 따른 텍스트 표시 위치

TextLeftOffset = 85	TextLeftOffset = 10
	

사용법



이벤트 SelectedIndexChanged

리스트에서 아이템이 클릭되거나, 사용자가 임의로 ItemListSelectItemIndex 속성값 변경 시 발생하는 이벤트입니다.

참고 "ItemListSelectItemIndex 속성" 내용을 참고하시기 바랍니다.

C# 사용법

```
// ItemListSelectItemIndex 속성값이 변경되는 경우 발생하는 이벤트
private void smartComboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
  // 선택된 Index를 출력
  SmartMessageBox.Show(smartComboBox1.ItemListSelectItemIndex.ToString());
}
```

VB 사용법

```
Private Sub smartComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles smartComboBox1.SelectedIndexChanged
  SmartMessageBox.Show(smartComboBox1.ItemListSelectItemIndex.ToString())
End Sub
```

4) SmartComboBox 예제 사용하기

SmartComboBox를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

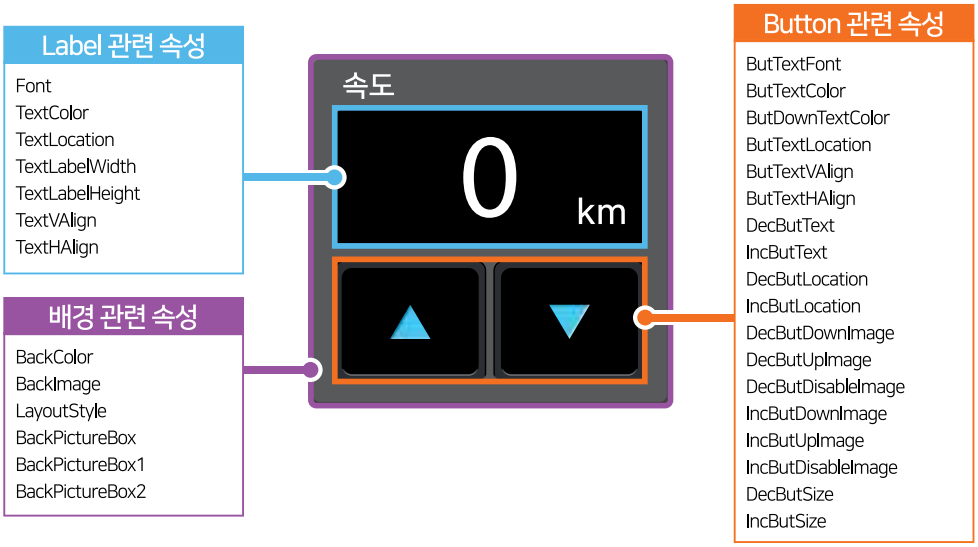
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartComboBox"

8. SmartUpDown

SmartUpDown은 .NET Compact Framework에서 지원되는 NumericUpDown 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 UpDown의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다

- 다양한 형태의 사용자 인터페이스 디자인 이미지 적용 기능
- 버튼 및 값 표시 위치를 변경할 수 있어 각 요소별 디자인 변경 지원
- 값 표시는 숫자 형식과 문자열 배열도 적용 가능
- 버튼의 기능은 SmartButton의 기능을 적용하여 반복 입력, 안전 모드 등의 특수 기능 지원
- 배경 이미지를 투명하게 처리할 수 있는 속성을 제공하여 이미지 작업을 편리하게 처리
- 기본 디자인 모드 지원 및 Layout 설정 변경 기능 지원
- 반복 입력 시 입력 가속도 설정 기능 지원
- 다양한 형태에서 원하는 크기로 조절 가능하며 크기 변경에 따라서 각 요소가 자동 정렬

1) 디자인 요소별 속성 명칭



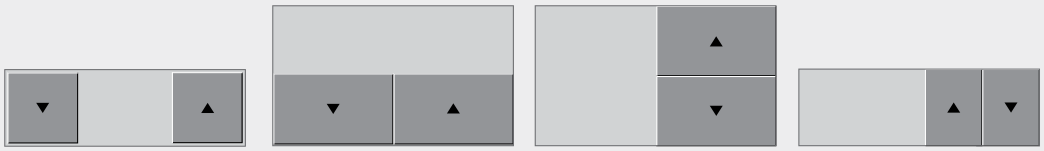
2) 프로그래밍 적용 가이드

STEP-1 Layout 설정하기

SmartUpDown의 Layout(Button의 위치)은 총 4가지를 지원하며 LayoutStyle 속성을 변경하여 다양한 형태를 표현할 수 있습니다. 또한 Label의 위치도 TextLocation 속성으로 자유롭게 변경할 수 있습니다.

※ 자세한 내용은 "LayoutStyle, TextLocation 속성" 내용을 참고하시기 바랍니다.

[LayoutStyle 적용에 따른 SmartUpDown 표현 형태]



중요 LayoutStyle 속성값 적용 방법 및 과정

1. SmartUpDown 컨트롤러의 폼 디자이너 추가 시 최초 LayoutStyle 속성값은 사용하고자 하는 Layout 형태에 따라서 BOTTOM, RIGHT1, RIGHT2 중에서 선택하여 적용 후 전체 크기와 위치를 조절합니다.
2. 크기와 위치 조절 후 Layout Style 속성값을 CUSTOM으로 설정하여 추가적인 디자인 요소들을 변경합니다.

STEP-2 디자인 요소 설정하기

SmartUpDown은 배경, Button, Label에 다양한 디자인을 적용할 수 있습니다. 특히 Button과 Label의 경우 Layout에 따라 위치를 조절해 주셔야 합니다.

1. 배경의 경우 BackColor 속성을 사용하여 배경 색상을 변경할 수 있으며, BackImage 속성을 이용해 배경 이미지를 개별로 설정하거나, BackPictureBox 관련 속성을 이용해 배경이 되는 컨트롤(PictureBox, SmartForm, SmartInnerForm, SmartGroupBox)의 이미지를 투영하여 배경을 설정할 수 있습니다. BackImage 속성과 BackPictureBox 속성은 서로 중첩되지 않으며, BackPictureBox 속성을 설정하면 BackImage가 적용되지 않습니다.

2. Button의 경우 이미지를 사용해 디자인을 적용할 수 있으며, 배경을 BackPictureBox로 설정한 경우 Button 이미지에 마스킹 처리를 해야합니다. 또한, Button 이미지에 Text가 포함된 경우 Inc, Dec Button 각각의 Text를 지워야 합니다. 이미지 적용 후 DecButLocation, IncButLocation 속성을 이용해 Button의 위치를 조절할 수 있습니다.

3. Label의 경우 SmartUpDown의 Layout에 따라 TextLocation 속성을 사용해 위치를, Font 속성으로 폰트를 변경할 수 있습니다.

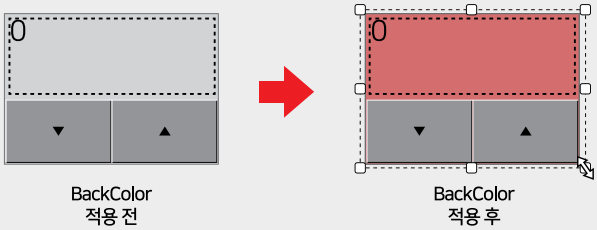
[표] 디자인 요소에 따른 관련 속성

디자인 요소	관련 속성
배경	BackColor, BackImage, BackPictureBox
Button	DecButDownImage, DecButUpImage, DecButDisableImage, IncButDownImage, IncButUpImage, IncButDisableImage, ButDownTextColor, ButTextColor, ButTextFont, ButTextLocation, ButTextHAlign, ButTextVAlign, DecButLocation, IncButLocation, DecButSize, IncButSize, DecButText, IncButText
Label	Font, TextHAlign, TextLabelHeight, TextLabelWidth, TextVAlign, TextLocation

※ 자세한 내용은 위 표의 속성 내용을 참고하시기 바랍니다.

1. BackColor 속성을 사용하는 경우

SmartUpDown의 BackColor 속성을 변경하면 SmartUpDown의 배경 색상이 변경됩니다.

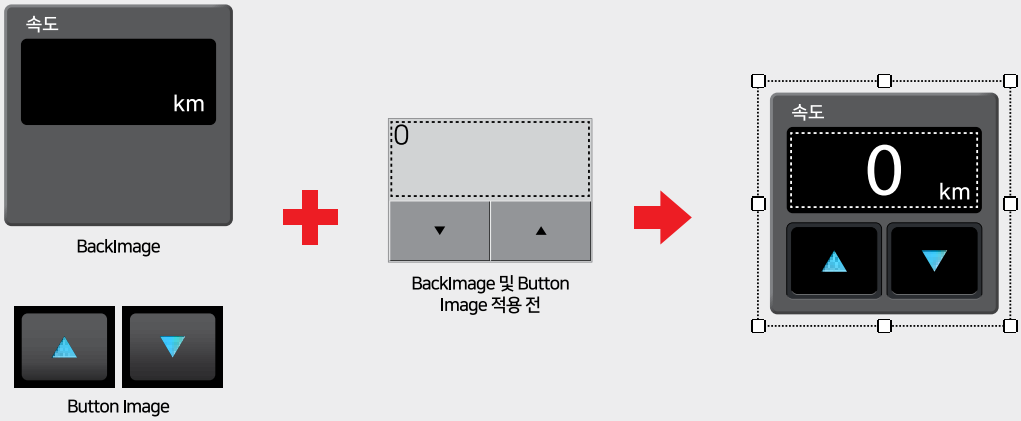


BackColor 적용 전 BackColor 적용 후

참고 점선으로 표시된 영역은 Label Text에 정렬되는 영역의 기준을 폼 디자이너에서만 확인할 수 있는 가상의 영역입니다. 해당 기능은 TextLabelBoundaryLine 속성으로 활성화하거나 비활성화할 수 있습니다. 자세한 설명은 "TextLabelBoundaryLine 속성" 내용을 참고하시기 바랍니다.

2. BackImage 속성을 사용하는 경우

SmartUpDown의 BackImage를 적용하면 SmartUpDown의 배경을 이미지로 설정할 수 있습니다. Button 이미지는 마스킹 처리를 하면 안 됩니다.



BackImage

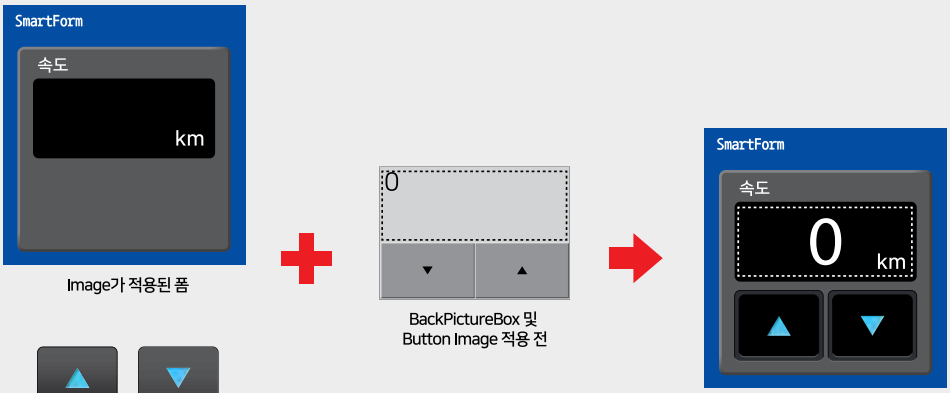
BackImage 및 Button Image 적용 전

Button Image

3. BackPictureBox 속성을 사용하는 경우

이미지가 적용된 PictureBox, SmartForm, SmartInnerForm, SmartGroupBox 컨트롤 위에 SmartUpDown을 위치시킨 후 BackPictureBox 속성을 해당 컨트롤로 설정하면 배경 이미지가 SmartUpDown에 투영됩니다.

Button 이미지는 마스킹 처리를 하셔야 합니다.



Image가 적용된 폼

BackPictureBox 및 Button Image 적용 전

마스킹 처리된 Button Image

3) SmartUpDown 인터페이스 설명

SmartUpDown Component Interface		
속성		
BackColor : Color	BackImage : Image	BackPictureBox : PictureBox
BackPictureBox1 : SmartInnerForm	BackPictureBox2 : SmartGroupBox	ButDownTextColor : Color
ButTextColor : Color	ButTextFont : Font	ButTextHAlign : SmartButton.TextHorAlign
ButTextLocation : Point	ButTextVAlign : SmartButton.TextVerAlign	DecButDisableImage : Image
DecButDownImage : Image	DecButLocation : Point	DecButSize : Size
DecButText : string	DecButUpImage : Image	fMaxValue : decimal
fMinValue : decimal	Font : Font	FormatString : string
fStepValue : decimal	fValue : decimal	IncButDisableImage : Image
IncButDownImage : Image	IncButLocation : Point	IncButSize : Size
IncButText : string	IncButUpImage : Image	InitVisible : bool
LabelTextArray : string[]	LayoutStyle : SmartUpDown.LAYOUTSTYPES	MaxValue : long
MinValue : long	RepeatInterval : int	RepeatIntervalAccelerate : SmartButton.RepeatAccelerate[]
SafeInterval : int	SpecialFunction: SmartButton.SPECIALFUNC	StepValue : int
TextColor : Color	TextHAlign : SmartUpDown.TextHorAlign	TextLabelHeight : int
TextLabelWidth : int	TextLocation : Point	TextVAlign : SmartUpDown.TextVerAlign
TxtValue : string	Value : long	TextLabelBoundaryLine : bool
이벤트		
OnDecButClick : EventHandler	OnDecButMouseUp : EventHandler	OnIncButMouseUp : EventHandler
OnIncButClick : EventHandler		

프로퍼티(속성) BackColor

SmartUpDown의 배경 색상을 설정합니다.

- Color : SmartUpDown의 배경 색상

C# 사용법

```
// SmartUpdown의 배경 색상을 파란색으로 설정
smartUpDown1.BackColor = Color.Blue;
```

VB 사용법

```
smartUpDown1.BackColor = Color.Blue
```

프로퍼티(속성) BackImage Image

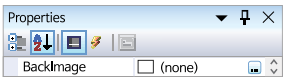
SmartUpDown의 배경 이미지를 설정합니다.

주의 BackPictureBox, BackPictureBox1, BackPictureBox2 속성이 설정된 경우 적용되지 않습니다.

Image 이미지 사용 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 “이미지 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartButton의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartUpDown에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 BackColor 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

프로퍼티(속성) InitVisible

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartUpDown의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) LayoutStyle

SmartUpDown의 Layout을 편리하게 변경할 수 있습니다. 속성값은 다음과 같으며 자세한 설명은 아래 표를 확인하시기 바랍니다.

중요 LayoutStyle 속성값 권장 사용 방법

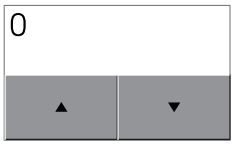
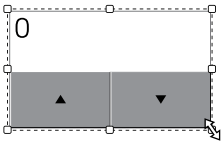
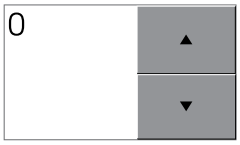
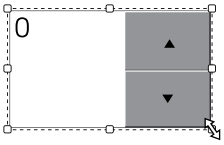
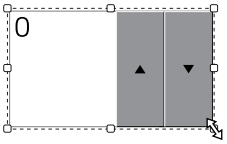
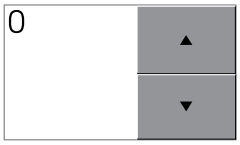
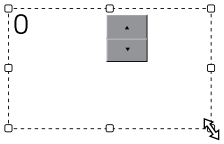
LayoutStyle 속성값을 BOTTOM, RIGHT1, RIGHT2으로 설정 후 크기나 위치를 조절하는 경우 변경한 디자인이 적용되지 않기 때문에 반드시 다음과 같은 과정으로 사용하시기 바랍니다.

STEP-1	SmartUpDown 컨트롤러의 폼 디자이너 추가 시 최초 LayoutStyle 속성값은 사용하고자 하는 Layout 형태에 따라서 BOTTOM, RIGHT1, RIGHT2 중에서 선택하여 적용 후 전체 크기와 위치를 조절합니다.
STEP-2	크기와 위치 조절 후 Layout Style 속성값을 CUSTOM으로 설정하여 추가적인 디자인 요소들을 변경합니다.

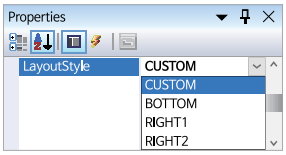
참고 "DecButLocation, IncButLocation 속성" 내용을 참고하시기 바랍니다.

- SmartUpDown.LAYOUTSTYPES.BOTTOM
- SmartUpDown.LAYOUTSTYPES.RIGHT1
- SmartUpDown.LAYOUTSTYPES.RIGHT2
- SmartUpDown.LAYOUTSTYPES.CUSTOM

[표] LAYOUTSTYPES에 따른 SmartUpDown의 Layout 변화

속성	버튼위치 및 형태	마우스로 Size 조정 시	특징 및 설명
BOTTOM			사이즈 조정 시 전체 사이즈는 비율에 맞게 조정되며, 증가, 감소 Button은 하단으로 정렬됨
RIGHT1			사이즈 조정 시 전체 사이즈는 비율에 맞게 조정되며, 증가, 감소 Button이 우측의 상/하 형태로 정렬됨
RIGHT2			사이즈 조정 시 전체 사이즈는 비율에 맞게 조정되며, 증가, 감소 Button이 우측의 좌/우 형태로 정렬됨
CUSTOM			사이즈 조정 시 전체 사이즈는 고정되며, 증가, 감소 Button은 우측의 상/하 형태로 정렬됨 ※ 모든 LayoutStyle 적용 시 디자인 설정이 완료된 후 사용됨

사용법



프로퍼티(속성) ButDownTextColor, ButTextColor, TextColor

- ButDownTextColor : SmartUpDown의 증가, 감소 Button의 눌러진 상태의 Text 색상을 설정합니다.
- ButTextColor : SmartUpDown의 증가, 감소 Button Text 색상을 설정합니다.
- TextColor : SmartUpDown의 Label Text 색상을 설정합니다.
- Color : Label 또는 증가, 감소 Button Text의 Color를 설정

C# 사용법

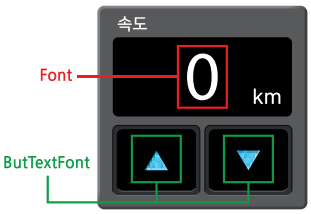
```
// 속성창 및 코드 입력하여 변경 가능합니다.
smartUpDown1.TextColor = Color.LightYellow;
smartUpDown1.ButTextColor = Color.Blue;
smartUpDown1.ButDownTextColor = Color.DarkGray;
```

VB 사용법

```
smartUpDown1.TextColor = Color.LightYellow
smartUpDown1.ButTextColor = Color.Blue
smartUpDown1.ButDownTextColor = Color.DarkGray
```

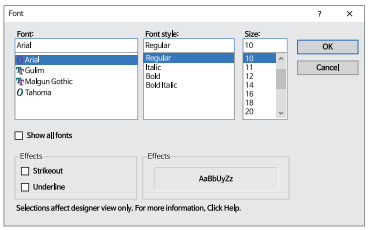
프로퍼티(속성) ButTextFont, Font

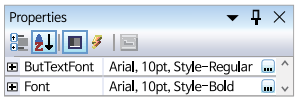
- ButTextFont : SmartUpDown의 증가, 감소 Button의 폰트를 설정합니다.
- Font : SmartUpDown의 Label 영역에 표시되는 폰트를 설정합니다.



- Font : Label 또는 Button에 표시할 폰트

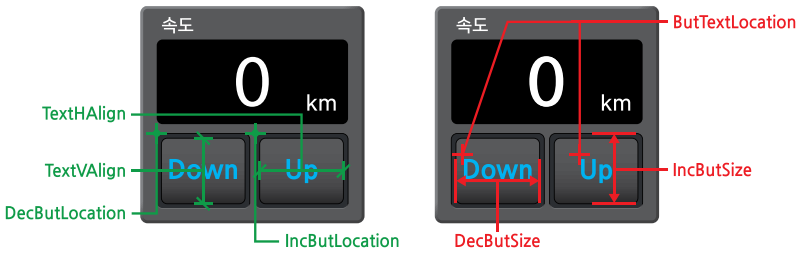
사용법



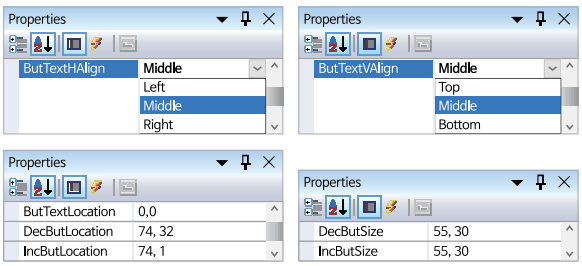


프로퍼티(속성) ButTextHAlign, ButTextVAlign, ButTextLocation, DecButLocation, IncButLocation, DecButSize, IncButSize

- ButTextHAlign : 증가, 감소 Button Text의 수평 정렬을 설정합니다.
- ButTextVAlign : 증가, 감소 Button Text의 수직 정렬을 설정합니다.
- ButTextLocation : 증가, 감소 Button Text의 위치를 설정합니다.
- DecButLocation : 감소 Button의 위치를 설정합니다.
- IncButLocation : 증가 Button의 위치를 설정합니다.
- DecButSize : 감소 Button의 크기를 설정합니다.
- IncButSize : 증가 Button의 크기를 설정합니다.



사용법



프로퍼티(속성) DecButDownImage, DecButUpImage, DecButDisableImage, IncButDownImage, IncButUpImage, IncButDisableImage

Masking Image

각각의 버튼 상태 변화에 따른 이미지를 설정합니다.

- DecButDownImage : 감소 버튼이 눌러진 상태의 이미지를 설정합니다.
- DecButUpImage : 감소 버튼이 눌러지지 않은 상태의 이미지를 설정합니다.
- DecButDisableImage : 감소 버튼 Enable 속성값이 False인 상태의 이미지를 설정합니다.
- IncButDownImage : 증가 버튼이 눌러진 상태의 이미지를 설정합니다.
- IncButUpImage : 증가 버튼이 눌러지지 않은 상태의 이미지를 설정합니다.
- IncButDisableImage : 증가 버튼 Enable 속성값이 False인 상태의 이미지를 설정합니다.

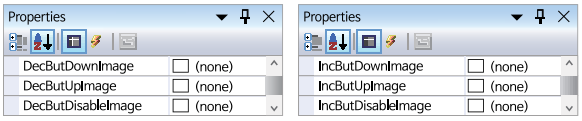
DecButDownImage	DecButUpImage	DecButDisableImage
		
IncButDownImage	**IncButUpImage**	**IncButDisableImage**
		

Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 “투명(Masking) 처리 영역 제작 가이드”에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 “이미지 제작 가이드”와 “투명(Masking) 처리 영역 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

사용법



프로퍼티(속성) DecButText, IncButText

증가, 감소 Button Text 설정 값을 가져오거나 설정합니다.

- DecButText : 감소 Button의 Text를 가져오거나 설정합니다.
- IncButText : 증가 Button의 Text를 가져오거나 설정합니다.

참고 Button 이미지에 Text가 포함된 경우 DecButText, IncButText를 삭제하시기 바랍니다.

- string : 증가, 감소 Button Text

C# 사용법

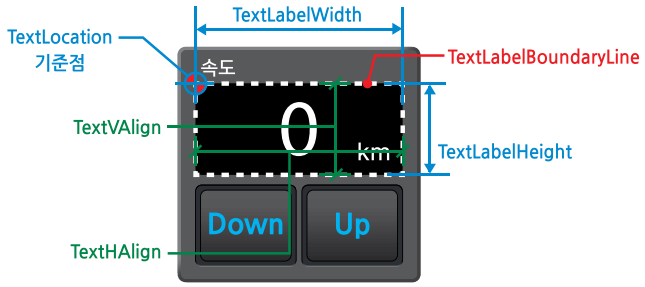
```
// 속성창 및 코드 입력하여 변경 가능합니다.
smartUpDown1.IncButText = " "; smartUpDown1.DecButText = " ";
```

VB 사용법

```
smartUpDown1.IncButText = " "  smartUpDown1.DecButText = " "
```

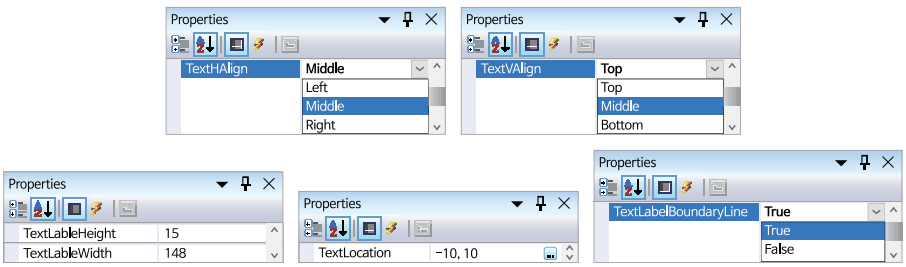
프로퍼티(속성) TextHAlign, TextVAlign, TextLabelHeight, TextLabelWidth, TextLocation, TextLabelBoundaryLine

- TextHAlign : Label Text의 수평 정렬을 설정합니다.
- TextVAlign : Label의 수직 정렬을 설정합니다.
- TextLabelHeight : TextVAlign 정렬을 위한 Label의 가상 높이를 설정합니다. (TextLocation 기준)
- TextLabelWidth : TextHAlign 정렬을 위한 Label의 가상 폭을 설정합니다.(TextLocation 기준)
- TextLocation : Label Text의 기준 위치를 설정합니다.(좌측 상단)
- TextLabelBoundatyLine : Label Text에 정렬되는 영역의 기준을 폼 디자이너에서 편리하게 확인할 수 있도록 가상의 영역을 폼 디자이너에서만 표시합니다. TextLocation, TextLabelWidth, TextLabelHeight 속성에 따른 사각형 영역을 표시합니다. (True : 폼 디자이너에서 영역 표시, False : 폼 디자이너에서 영역 표시 안 함) ※ 프로그램 실행 시 자동으로 False로 변경되어 제거됩니다.



- SmartUpDown.TextHorAlign : Label Text의 수평 정렬
  - SmartUpDown.TextHorAlign.Left : 좌측 정렬
  - SmartUpDown.TextHorAlign.Middle : 중앙 정렬
  - SmartUpDown.TextHorAlign.Right : 우측 정렬
- SmartUpDown.TextVerAlign : Label Text의 수직 정렬
  - SmartUpDown.TextVerAlign.Top : 상단 정렬
  - SmartUpDown.TextVerAlign.Middle : 중단 정렬
  - SmartUpDown.TextVerAlign.Bottom : 하단 정렬
- int : 가상 Label의 폭 및 높이

사용법



프로퍼티(속성) fMaxValue, fMinValue, fStepValue, MaxValue, MinValue, StepValue

SmartUpDown에서 표현할 수 있는 최대값과 최소값 그리고 증가, 감소값을 설정합니다.

- fMaxValue : 최대값 설정(실수형 : decimal)
- fMinValue : 최소값 설정(실수형 : decimal)
- fStepValue : 증가, 감소 Button 클릭시 증가, 감소할 값 설정(실수형 : decimal)
- MaxValue : 최대값 설정(정수형 : long)
- MinValue : 최소값 설정(정수형 : long)
- StepValue : 증가, 감소 Button 클릭시 증가, 감소할 값 설정(정수형 : int)

- decimal : SmartUpDown의 실수형 값
- long : SmartUpDown의 정수형 값
- int : 정수형 값의 증가, 감소값

C# 사용법

```
smartUpDown1.fMaxValue = 100.0m; smartUpDown1.fMinValue = 10.0m; smartUpDown1.fStepValue = 2.0m;
smartUpDown1.MaxValue = 150; smartUpDown1.MinValue = 10; smartUpDown1.StepValue = 2;
```

VB 사용법

```
smartUpDown1.fMaxValue = 100.0D : smartUpDown1.fMinValue = 10.0D : smartUpDown1.fStepValue = 2.0D
smartUpDown1.MaxValue = 150 : smartUpDown1.MinValue = 10 : smartUpDown1.StepValue = 2
```

프로퍼티(속성) TxtValue

SmartUpDown의 Label Text를 문자 형식으로 설정합니다. 즉, 수치형 데이터뿐만 아니라 문자열도 적용이 가능합니다.

- string : Label Text에 표현할 문자열

C# 사용법

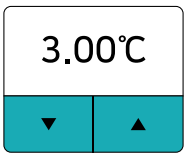
```
// 수치형 데이터만 처리되는 것이 아닌 문자열도 적용이 가능합니다.
smartUpDown1.TxtValue = " 문자열 적용 "; smartUpDown1.TxtValue = " 33.678 ";
```

VB 사용법

```
smartUpDown1.TxtValue = " 문자열 적용 "  smartUpDown1.TxtValue = " 33.678 "
```

프로퍼티(속성) FormatString

SmartUpDown의 Label 값이 fValue(실수형)로 설정된 경우 표현 형식을 설정합니다. 즉, 실수 표현 시 소수점 자릿수를 설정합니다.



- string : 실수형 데이터의 표현 형식 (기본값 : "0.00")

C# 사용법

```
// 실수형 데이터의 표현 형식을 설정합니다.
smartUpDown1.FormatString = " 0.00℃ ";  // 결과 1.50℃
smartUpDown1.FormatString = " 000.0% ";  // 결과 001.5%
```

VB 사용법

```
smartUpDown1.FormatString = " 0.00℃ "  : smartUpDown1.FormatString = " 000.0% "
```

프로퍼티(속성) LabelTextArray

SmartUpDown 컴포넌트의 특징 중 하나인 수치형(정수형, 실수형) 데이터 표현뿐만 아니라 문자열 형태의 데이터를 적용할 수 있도록 하였습니다. 모드 선택과 같은 형태의 선택 메뉴 및 비선형적인 수치 데이터를 표현할 수 있습니다.

- string[] : 증가, 감소 Button 클릭 시 표현할 문자열 배열

C# 사용법

```
string[] strMenu = new string[7];  // 각각의 메뉴 문자열을 배열로 설정합니다.
strMenu[0] = " Mode-A "; strMenu[1] = " Mode-B "; strMenu[2] = " Mode-C "; strMenu[3] = " Mode-D ";
```

```
strMenu[4] = " Mode-E " ; strMenu[5] = " Mode-F " ; strMenu[6] = " Mode-G " ;
smartUpDown1.LabelTextArray = strMenu;
// 증가 및 감소 Button을 누를때마다 선택 항목이 변경됨
```

VB 사용법

```
Dim strMenu() As String = New String(6) {}
strMenu(0) = " Mode-A " : strMenu(1) = " Mode-B " : strMenu(2) = " Mode-C " : strMenu(3) = " Mode-D "
strMenu(4) = " Mode-E " : strMenu(5) = " Mode-F " : strMenu(6) = " Mode-G "
smartUpDown1.LabelTextArray = strMenu
```

프로퍼티(속성) RepeatInterval, SafeInterval

SpecialFunction 속성을 설정한 경우 각 특수 기능(SAFE, REPT)에서 각각의 시간 간격(Interval)을 설정합니다.

- RepeatInterval : SpecialFunction이 REPT인 경우 반복 입력 시간 간격을 설정합니다.
- SafeInterval : SpecialFunction이 SAFE인 경우 입력 유지 시간을 설정합니다.

참고 SpecialFunction 속성값이 SAFE_REPT인 경우 두 속성을 모두 사용하며, "SpecialFunction 속성" 내용을 참고하시기 바랍니다.

- int : 시간 간격 (단위 : ms)

C# 사용법

```
// REPT 기능 사용 설정하고 반복 입력 간격 시간을 1초(1000ms) 설정
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.REPT; smartUpDown1.RepeatInterval = 1000;
// SAFE 기능 사용 설정하고 Button의 입력 유지 시간을 2초(2000ms) 설정
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE; smartUpDown1.SafeInterval = 2000;
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE_REPT;  // SAFE + REPT 기능 사용 설정
// 반복 입력 간격 시간 : 1초(1000ms), Button의 입력 유지 시간 : 2초(2000ms)
smartUpDown1.RepeatInterval = 1000; smartUpDown1.SafeInterval = 2000;
```

VB 사용법

```
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.REPT : smartUpDown1.RepeatInterval = 1000
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE : smartUpDown1.SafeInterval = 2000
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.SAFE_REPT
smartUpDown1.RepeatInterval = 1000 : smartUpDown1.SafeInterval = 2000
```

프로퍼티(속성) RepeatIntervalAccelerate

SpecialFunction 속성을 REPT 또는 SAVE_REPT로 설정하여 Button 반복 입력 기능을 사용하는 경우, 반복 입력 횟수에 따라 입력 시간 간격(Interval)을 증가, 감소할 수 있습니다. 즉, Button 입력 속도를 가속시킬 수 있습니다.

- SmartButton.RepeatAccelerate[] : Button 반복 입력 횟수 및 입력 속도 배열
  - int iClickCount : 반복 입력 횟수
  - int iInterval : 입력 속도

C# 사용법

```
// 버튼을 계속 누르고 있을 경우 클릭(Click) 이벤트를 일정 간격으로 발생시킨다.
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.REPT;
// RepeatAccelerate 타입의 배열변수 repAcc 선언 및 생성. 3단계 가속
SmartButton.RepeatAccelerate[] repAcc = new SmartButton.RepeatAccelerate[3];
repAcc[0].iClickCount = 10; repAcc[0].iInterval = 300;  // 처음 10회 입력 클릭 간 Interval 300
repAcc[1].iClickCount = 10; repAcc[1].iInterval = 100;  // 그 다음 10회 입력 클릭 간 Interval 100
repAcc[2].iClickCount = 10; repAcc[2].iInterval = 10;   // 이후 입력 되는 클릭 간 Interval 10
smartUpDown1.RepeatIntervalAccelerate = repAcc;
```

VB 사용법

```
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.REPT
Dim repAcc As SmartButton.RepeatAccelerate() = New SmartButton.RepeatAccelerate(2) {}
repAcc(0).iClickCount = 10 : repAcc(0).iInterval = 300
repAcc(1).iClickCount = 10 : repAcc(1).iInterval = 100
repAcc(2).iClickCount = 10 : repAcc(2).iInterval = 10
smartUpDown1.RepeatIntervalAccelerate = repAcc
```

프로퍼티(속성) SpecialFunction

SmartButton의 SpecialFunction 속성과 동일한 기능을 하며, 증가, 감소 Button의 특수 기능을 설정합니다.

참고 "RepeatInterval, SafeInterval 속성" 내용을 참고하시기 바랍니다.

- SmartButton.SPECIALFUNC.NONE : 특수 기능을 사용하지 않습니다. (기본값)
- SmartButton.SPECIALFUNC.REPT : Button을 계속 누르고 있을 경우 클릭(Click) 이벤트를 일정 간격(RepeatInterval 또는 RepeatIntervalAccelerate 속성값)으로 발생시킵니다.
- SmartButton.SPECIALFUNC.SAFE : Button 클릭 시, 즉 터치 입력에서 체터링(Chattering)을 방지해야 하거나 혹은 장비 동작에 있어 신중하게 버튼의 입력을 해야 하는 경우 사용하는 기능입니다. Button 클릭 시 바로 입력되는 것이 아니라 설정된 시간 간격(SafeInterval 속성값) 이상 Button 클릭을 유지해야만 클릭 이벤트가 발생됩니다.
- SmartButton.SPECIALFUNC.SAFE_REPT : REPT 기능과 SAFE 기능을 모두 사용합니다.

C# 사용법

```
// 반복 기능 사용
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.REPT;
```

VB 사용법

```
smartUpDown1.SpecialFunction = SmartButton.SPECIALFUNC.REPT
```

이벤트 OnDecButClick, OnIncButClick

증가, 감소 Button이 클릭된 경우 발생되는 이벤트입니다.

- OnDecButClick : 감소 Button이 클릭된 경우 발생되는 이벤트
- OnIncButClick : 증가 Button이 클릭된 경우 발생되는 이벤트

C# 사용법

```
private void smartUpDown1_OnDecButClick(object sender, EventArgs e)
{
  // 증가 Button이 눌려진 경우 처리해야 할 코드를 입력하세요.
}
private void smartUpDown1_OnIncButClick(object sender, EventArgs e)
{
  // 감소 Button이 눌려진 경우 처리해야 할 코드를 입력하세요.
}
```

VB 사용법

```
Private Sub smartUpDown1_OnDecButClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartUpDown1.OnDecButClick
  ' 증가 버튼이 눌려진 경우 처리해야 할 코드를 입력하세요.
End Sub
Private Sub smartUpDown1_OnIncButClick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartUpDown1.OnIncButClick
  ' 감소 버튼이 눌려진 경우 처리해야 할 코드를 입력하세요.
```

```
End Sub
```

이벤트 OnDecButMouseUp, OnIncButMouseUp

증가, 감소 Button이 눌렀다 떼질 때 발생되는 이벤트입니다.

- OnDecButMouseUp : 감소 Button이 눌렀다 떼질 때 발생되는 이벤트
- OnIncButMouseUp : 증가 Button이 눌렀다 떼질 때 발생되는 이벤트

C# 사용법

```
private void smartUpDown1_OnDecButMouseUp(object sender, EventArgs e)
{
  // 감소 Button이 눌렀다 떼진 경우 처리해야 할 코드를 입력하세요.
}
private void smartUpDown1_OnIncButMouseUp(object sender, EventArgs e)
{
  // 증가 Button이 눌렀다 떼진 경우 처리해야 할 코드를 입력하세요.
}
```

VB 사용법

```
Private Sub smartUpDown1_OnDecButMouseUp(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartUpDown1.OnDecButMouseUp
  ' 감소 Button이 눌렀다 떼진 경우 처리해야 할 코드를 입력하세요.
End Sub
Private Sub smartUpDown1_OnIncButMouseUp(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartUpDown1.OnIncButMouseUp
  ' 증가 Button이 눌렀다 떼진 경우 처리해야 할 코드를 입력하세요.
End Sub
```

4) SmartUpDown 예제 사용하기

SmartUpDown을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

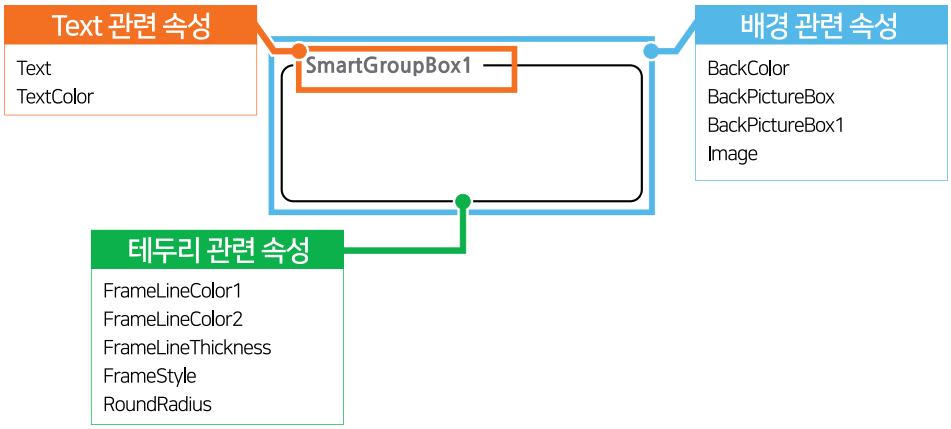
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartUpDown"

9. SmartGroupBox

SmartGroupBox은 .NET Compact Framework에서 지원되는 GroupBox 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GroupBox의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 배경 투명 처리 효과 기능(PictureBox, SmartForm, SmartInnerForm 연동) 및 스킨 처리
- Panel 컨트롤과 같은 컨테이너 컨트롤 기능
- 다양한 스타일을 적용할 수 있도록 색상 및 스타일 변경 기능
- 배경 이미지 설정 기능 및 SmartX의 사용자 인터페이스 컴포넌트와 연동 기능

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartGroupBox 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartGroupBox에서 이미지는 배경 컴포넌트에 사용됩니다.

STEP-1 배경 속성 설정하기

SmartGroupBox의 배경 색상을 BackColor 속성을 사용해 색상으로 설정하는 방법, BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있으며, Image 속성으로 사용자 이미지를 사용하는 방법이 있습니다.

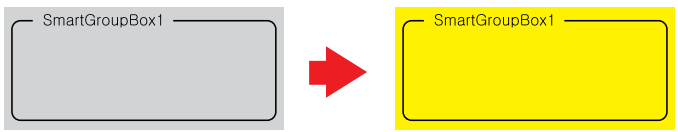
[CASE-1] 색상으로 설정하는 경우

배경 컴포넌트의 배경 색상과 동일하게 설정하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackColor	Yellow

※ 자세한 내용은 “BackColor 속성” 내용을 참고하시기 바랍니다.



[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackPictureBox	SmartForm

※ 자세한 내용은 "BackPictureBox 속성" 내용을 참고하시기 바랍니다.

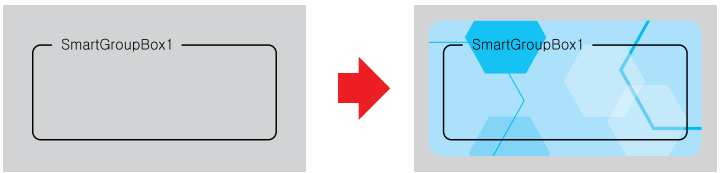
Image가 적용된 폼　　BackPictureBox 적용 전

[CASE-3] 사용자 이미지를 사용하는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
Image	

※ 자세한 내용은 "Image 속성" 내용을 참고하시기 바랍니다.



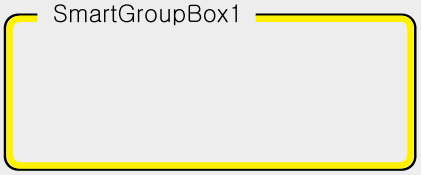
STEP-2 테두리 관련 속성 설정하기

테두리 스타일을 FrameStyle 속성으로 설정하며, FrameLineColor1, FrameLineThickness 속성으로 테두리의 색상 및 두께를 설정합니다. 만약 스타일 중 3D인 경우에 FrameLineColor2 속성도 같이 설정하며, RoundRectangle인 경우에는 RoundRadius 속성으로 모서리 부분의 라운드값을 설정합니다

[표] 관련 속성의 속성값 예시

관련 속성	속성값
FrameStyle	RoundRectangle3D
FrameLineColor1	Black
FrameLineColor2	Yellow
FrameLineThickness	5
RoundRadius	15

※ 자세한 내용은 "FrameStyle, FrameLineColor1, FrameLineColor2, FrameLineThickness, RoundRadius 속성" 내용을 참고하시기 바랍니다.



[그림] 적용된 모습

STEP-3 Text 관련 속성 설정하기

SmartGroupBox에 표시되는 Text를 Text 속성으로 설정하며, TextColor 속성으로 색상을 설정합니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
Text	SmartGroupBoxTest
TextColor	Black

※ 자세한 내용은 "Text, TextColor 속성" 내용을 참고하시기 바랍니다.

3) SmartGroupBox 인터페이스 설명

SmartGroupBox Component Interface		
속성		
BackColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
FrameLineColor1 : Color	FrameLineColor2 : Color	FrameLineThickness : int
FrameStyle : SmartGroupBox.FRAMESTYLES	Image : Image	InitVisible : bool
RoundRadius : int	Text : string	TextColor : Color

프로퍼티(속성) `InitVisible`

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리할 수 있는 속성입니다. 공통 영역에 위치한 SmartGroupBox의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

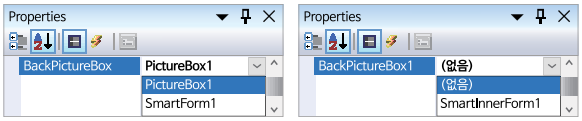
프로퍼티(속성) `BackPictureBox, BackPictureBox1`

SmartGroupBox의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartGroupBox에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 BackColor 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법



프로퍼티(속성) Image

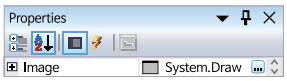
Image

SmartGroupBox의 배경 이미지를 지정합니다.

Image 이미지 사용 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 "이미지 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

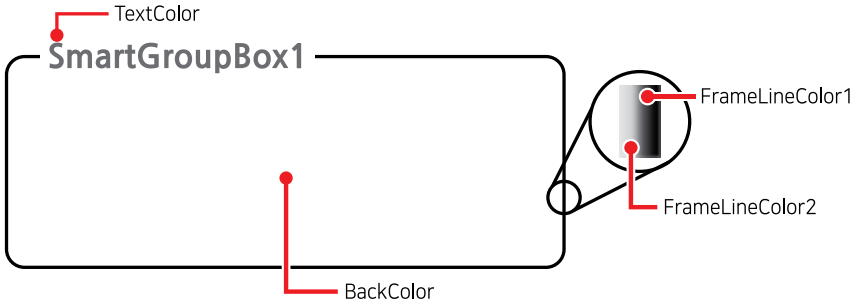
사용법



프로퍼티(속성) BackColor, FrameLineColor1, FrameLineColor2, TextColor

SmartGroupBox 관련 모양 및 색상 등을 설정합니다. (디자인 관련 요소)

- `BackColor` : SmartGroupBox의 배경 색상
- `FrameLineColor1` : 프레임의 첫 번째 선(외부)의 색상
- `FrameLineColor2` : 프레임의 두 번째 선(내부)의 색상
  (FrameStyle 속성값이 Rectangle3D, RoundRectangle3D인 경우에만 적용)
- `TextColor` : SmartGroupBox의 Font 색상



C# 사용법

```
smartGroupBox1.BackColor = Color.White;
smartGroupBox1.FrameLineColor1 = Color.Black;
smartGroupBox1.FrameLineColor2 = Color.DarkGray;
smartGroupBox1.TextColor = Color.DarkGray;
```

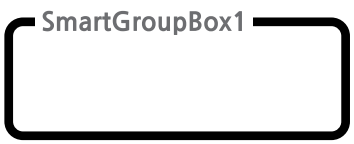
VB 사용법

```
smartGroupBox1.BackColor = Color.White
smartGroupBox1.FrameLineColor1 = Color.Black
smartGroupBox1.FrameLineColor2 = Color.DarkGray
smartGroupBox1.TextColor = Color.DarkGray
```

프로퍼티(속성) FrameLineThickness

프레임 선의 두께를 설정합니다.
(단, FrameStyle 속성값이 Rectangle3D, RoundRectangle3D인 경우 프레임의 두 번째 선(내부)에 적용됩니다.)

- int : 프레임 선의 두께 (기본값 : 1)

FrameLineThickness : 1 (기본값)	FrameLineThickness : 5
	

C# 사용법

```
smartGroupBox1.FrameLineThickness = 5;
```

VB 사용법

```
smartGroupBox1.FrameLineThickness = 5
```

프로퍼티(속성) FrameStyle

SmartGroupBox의 프레임 모양을 설정합니다.

- SmartGroupBox.FRAMESTYLES.None : 프레임 없음
- SmartGroupBox.FRAMESTYLES.Rectangle : 단선으로 사각형 모양
- SmartGroupBox.FRAMESTYLES.Rectangle3D : 입체감 있는 사각형 모양
- SmartGroupBox.FRAMESTYLES.RoundRectangle : 단선으로 모서리가 둥근 사각형 모양 (기본값)
- SmartGroupBox.FRAMESTYLES.RoundRectangle3D : 입체감 있는 모서리가 둥근 사각형 모양

C# 사용법

```
smartGroupBox1.FrameStyle = SmartGroupBox.FRAMESTYLES.None;  // 프레임 없음
```

VB 사용법

```
smartGroupBox1.FrameStyle = SmartGroupBox.FRAMESTYLES.None
```

프로퍼티(속성) RoundRadius

FrameStyle 속성값이 RoundRectangle, RoundRectangle3D인 경우 모서리 부분의 라운드 값을 설정합니다.

- int : 모서리 부분의 라운드값 (기본값 : 5)

RoundRadius : 5 (기본값)	RoundRadius : 15
	

C# 사용법

```
smartGroupBox1.RoundRadius = 15;
```

VB 사용법

```
smartGroupBox1.RoundRadius = 15;
```

프로퍼티(속성) **Text**

SmartGroupBox에 표시되는 문자열을 설정합니다.

C# 사용법

```
smartGroupBox1.Text = " SmartGroupBox1 Test " ;
```

VB 사용법

```
smartGroupBox1.Text = " SmartGroupBox1 Test "
```

4) SmartGroupBox 예제 사용하기

SmartGroupBox를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

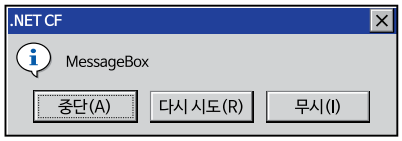
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartGroupBox"

10. SmartMessageBox

SmartMessageBox는 .NET Compact Framework에서 지원되는 MessageBox 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 GUI 필수 요소인 MessageBox의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.
기존의 MessageBox는 마우스 입력 중심으로 설계되어 만들어져 IEC-Series처럼 터치 기반의 제품에서는 여러 가지 불편한 점이 있으며, 이를 개선하기 위해 SmartMessageBox를 지원하여 터치 인터페이스에 적합하도록 버튼 위치 및 비율을 크게 하고 불필요한 OK 버튼 위치를 변경하여 접근성을 개선하였습니다. 그 밖에 세련되지 않은 디자인, 버튼의 텍스트가 고정인 점 등의 불편한 사항을 개선하였습니다.

- Windows 8.1 스타일의 세련된 디자인 적용
- MessageBox의 색상을 테마 별로 쉽게 변경
- MessageBox의 크기 변경 기능
- Button Text 및 아이콘 이미지 변경 기능
- 기존 MessageBox 인터페이스와 동일하여 코드 수정을 최소화
- 터치 인터페이스에 적합하도록 버튼 위치 및 크기 변경

[표] .NET Compact Framework의 MessageBox와 SmartMessageBox의 차이

MessageBox	SmartMessageBox
	
1. 변경할 수 있는 항목이 제한적(사이즈, 버튼 텍스트, 아이콘, 색상 등 변경 불가) 2. Blocking 방식만 지원 3. 사용자가 임의로 Close 불가능 4. 메시지 창을 띄운 후 Caption, Message, Button의 텍스트를 변경 불가	1. 사용자가 원하는 크기, 버튼 텍스트, 아이콘, 색상 등을 설정 가능 2. Blocking, None-Blocking 방식 모두 지원 3. 사용자가 임의로 Close 가능 4. 메시지 창을 띄운 후 Caption, Message, Button의 텍스트를 변경 가능 (None-Blocking 방식인 경우)

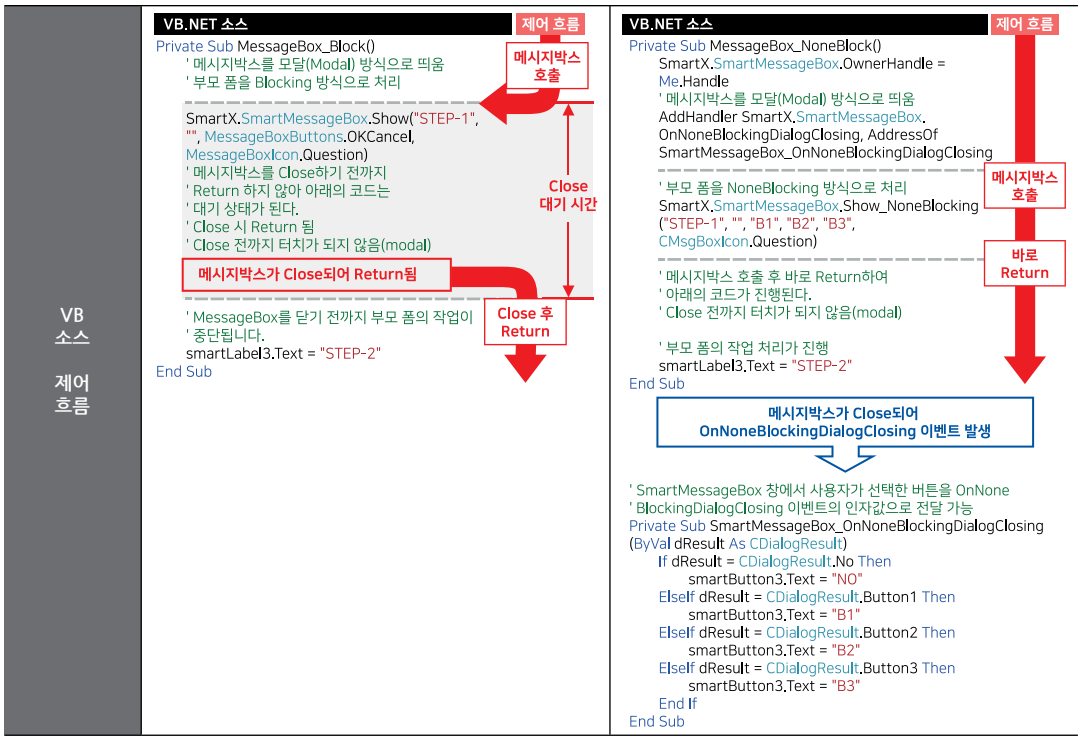
1) 디자인 요소별 속성 명칭

2) Blocking 방식 호출 VS None-Blocking 방식 호출

일반적으로 MessageBox는 Modal + Blocking 처리 방식으로 동작하지만 SmartMessageBox에서는 별도로 Modal + NoneBlocking 형태의 MessageBox를 출력할 수 있는 기능을 추가로 지원하고 있습니다.

[표] (Modal + Blocking 방식) VS (Modal + NoneBlocking 방식)

<table>
<tr><th>방식</th><th>Modal + Blocking 방식</th><th>Modal + NoneBlockingDialog 방식</th></tr>
<tr><td>메소드</td><td>Show(), Show_Ex()</td><td>Show_NoneBlocking()</td></tr>
<tr><td>설명</td><td>Show() 또는 Show_Ex() 메소드 호출 시 SmartMessageBox는 Modal 방식으로 띄워지고 배경이 되는 폼(부모 폼)은 Blocking 방식으로 처리되어 MessageBox가 띄워져 있는 동안 부모폼의 작업 처리가 진행되지 않습니다.</td><td>ShowDialogNoneBlock() 메소드 호출 시 SmartMessageBox는 Modal 방식으로 띄워지고 배경이 되는 폼(부모 폼)은 None-Blocking 방식으로 처리되어 MessageBox가 띄워져 있는 동안 부모 폼의 작업 처리가 진행됩니다.</td></tr>
<tr><td>특징</td><td>1. Modal 방식
2. 배경(부모 폼)의 작업 처리 중지</td><td>1. Modal 방식
2. 배경(부모 폼)의 작업 처리 진행
3. 메시지박스가 띄워져 있는 상태에서 캡션, 메시지, 버튼 텍스트 변경 가능</td></tr>
<tr><td>특징</td><td> 
[부모 폼(배경)의 작업 처리 중지]</td><td> 
[부모 폼(배경)의 작업 처리 진행]</td></tr>
<tr><td>C# 소스
제어 흐름</td><td>C# 소스 / 제어 흐름
private void MessageBox_Block()
{
// 메시지박스를 모달(Modal) 방식으로 띄움
// 부모 폼을 Blocking 방식으로 처리
SmartX.SmartMessageBox.Show("STEP-1","",MessageBoxButtons.OKCancel, MessageBoxIcon.Question);
// 메시지박스를 Close하기 전까지
// Return하지 않아 아래의 코드는
// 대기 상태가 된다.
// Close 시 Return 됨
// Close 전까지 터치가 되지 않음(modal)
메시지박스가 Close되어 Return됨
// MessageBox를 닫기 전까지 부모 폼의
// 작업이 중단됩니다.
smartLabel3.Text = "STEP-2";
}
메시지박스 호출
Close 대기 시간
Close 후 Return</td><td>C# 소스 / 제어 흐름
private void MessageBox_NoneBlock()
{
SmartX.SmartMessageBox.OwnerHandle = this.Handle;
// 메시지박스를 모달(Modal) 방식으로 띄움
SmartX.SmartMessageBox.OnNoneBlockingDialogClosing += new SmartMessageBox.DialogCloseEventHandler(SmartMessageBox_OnNoneBlockingDialogClosing);
// 부모 폼을 NoneBlocking 방식으로 처리
SmartX.SmartMessageBox.Show_NoneBlocking("STEP-1", "", "Yes", "No", "", CMsgBoxIcon.Question);
// 메시지박스 호출 후 바로 Return하여
// 아래의 코드가 진행된다.
// Close 전까지 터치가 되지 않음(modal)
// 부모 폼의 작업 처리가 진행
smartLabel3.Text = "STEP-2";
}
메시지박스 호출
바로 Return
메시지박스가 Close되어 OnNoneBlockingDialogClosing 이벤트 발생
// SmartMessageBox 창에서 사용자가 선택한 버튼을
// OnNoneBlockingDialogClosing 이벤트의
// 인자값으로 전달 가능
void smartMessageBox_OnNoneBlockingDialogClosing(CDialogResult dResult)
{
if (dResult == CDialogResult.No)
{
smartButton3.Text = "NO"; // 선택 안 함
}
else if (dResult == CDialogResult.Button1)
{
smartButton3.Text = "B1"; // 1번 버튼 선택
}
else if (dResult == CDialogResult.Button2)
{
smartButton3.Text = "B2"; // 2번 버튼 선택
}
else if (dResult == CDialogResult.Button3)
{
smartButton3.Text = "B3"; // 3번 버튼 선택
}
}</td></tr>
</table>



3) 프로그래밍 적용 가이드

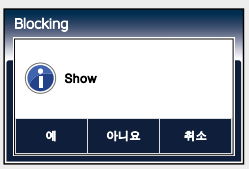
SmartMessageBox는 기본적인 메시지박스로 사용하는 방법과, 디자인적인 요소를 변경하여 사용하는 방법, None-Blocking 방식으로 사용하는 방법이 있습니다. 아래 CASE를 확인하여 적용하시기 바랍니다.

CASE-1 기본적인 메시지 박스 사용하기

SmartMessageBox는 .NET의 MessageBox와 동일한 사용법의 Show() 메소드를 호출해 화면에 출력시킬 수 있습니다. Show() 메소드 또한 DialogResult를 리턴하여 선택한 버튼에 따른 처리를 할 수 있습니다.

※ 자세한 내용은 "Show() 메소드"를 참고하시기 바랍니다.

```
// SmartMessageBox 버튼 선택 결과를 저장할 변수 선언 및 SmartMessageBox를 Blocking 방식으로 출력
DialogResult dResult = SmartX.SmartMessageBox.Show( " Show " , " Blocking " , MessageBoxButtons.
                                                    YesNoCancel, MessageBoxIcon.Asterisk);
// 클릭된 버튼 확인
if (dResult == DialogResult.Yes)
{
  // SmartMessageBox의 Yes 버튼이 클릭됨
}
```



[출력 결과]

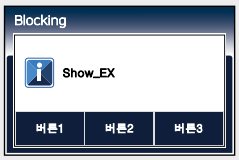
CASE-2 디자인적인 요소를 변경하여 사용하기

SmartMessageBox는 다양한 디자인 요소를 변경하여 출력할 수 있습니다. Size() 메소드로 크기, ThemeColor() 메소드로 테마, ICONimage() 메소드로 아이콘 이미지를 변경할 수 있으며, SetButtonFont(), SetCaptionFont(), SetMessageFont() 메소드를 사용해 각각 버튼, 캡션, 메시지에 출력되는 텍스트 폰트를 변경할 수 있습니다. 모든 설정값들은 한번만 설정해도 무방합니다.
디자인 요소 변경 후 Show_Ex() 메소드를 호출해 변경된 디자인이 적용된 SmartMessageBox를 화면에 출력시킬 수 있습니다. 버튼 텍스트는 Show_Ex() 메소드의 인자값을 이용해 변경할 수 있으며, 사용자 아이콘 이미지는 인자값을 Customize로 설정하여 적용할 수 있습니다. 또한, Show_Ex() 메소드는 CDialogResult를 리턴하여 선택한 버튼에 따른 처리를 할 수 있습니다.

[사용자 아이콘 이미지]

※ 자세한 내용은 " Size(), ThemeColor(), SetButtonFont(), SetCaptionFont(), SetMessageFont(), ICONimage(), Show_Ex() 메소드"를 참고하시기 바랍니다.

```
SmartX.SmartMessageBox.Size(300, 150); SmartX.SmartMessageBox.ThemeColor(SmartX.THEMECOLOR.HNS);
// 크기 및 테마 설정
Font Bfont = new Font( " 굴림 " , 10, FontStyle.Bold);
// 버튼, 캡션, 메시지에 출력할 텍스트의 폰트 설정
SmartX.SmartMessageBox.SetButtonFont(Bfont); SmartX.SmartMessageBox.SetCaptionFont(Bfont);
SmartX.SmartMessageBox.SetMessageFont(Bfont);
SmartX.SmartMessageBox.ICONImage(Resource1. TEST);  // 사용자 아이콘 이미지 설정
// SmartMessageBox 버튼 선택 결과를 저장할 변수 선언 및 SmartMessageBox를 Blocking 방식으로 출력
CDialogResult cdResult = SmartX.SmartMessageBox.Show_Ex( " Show_Ex " , " Blocking " , " 버튼1 " , " 버튼2 " ,
                                                  " 버튼3 " , SmartX.CMsgBoxIcon.Customize);
if (cdResult == SmartX.CDialogResult.Button1)
{
  // SmartMessageBox의 버튼1이 클릭됨
}
```



[출력 결과]

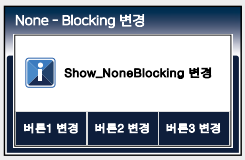
CASE-3 SmartMessageBox를 None-Blocking 방식으로 사용하기

Show_NoneBlocking() 메소드로 SmartMessageBox를 None-Blocking 방식으로 출력시킬 수 있습니다.
이때 Show_NoneBlocking() 메소드 호출 전 OwnerHandle 속성을 설정해야 배경 폼(부모 폼)의 작업 처리가 진행됩니다. Show_NoneBlocking() 메소드는 Show_Ex() 메소드와 같은 방법으로 디자인적인 요소를 변경하여 화면에 출력시킬 수 있습니다. 또한 메시지박스가 출력된 상태에서 SetButsText_NoneBlocking() 메소드로 버튼, SetCaption_NoneBlocking() 메소드로 캡션, SetMessage_NoneBlocking() 메소드로 메시지의 텍스트를 변경할 수 있는 기능을 제공합니다. 버튼 선택 결과는 Show(), Show_Ex() 메소드와는 달리 OnNoneBlockingDialogClosing 이벤트로 확인할 수 있습니다.

※ 자세한 내용은 "Blocking 방식 호출 VS None-Blocking 방식 호출", "Show_NoneBlocking(), SetButsText_NoneBlocking(), SetCaption_NoneBlocking(), SetMessage_NoneBlocking() 메소드"를 참고하시기 바랍니다.

```
private void Message_NoneBlock()
{
  // 디자인 요소 설정은 CASE-2와 동일, 활성화된 배경 폼의 핸들을 SmartMessageBox로 넘김
  SmartX.SmartMessageBox.OwnerHandle = this.Handle;
```

```
  // 이벤트에 SmartMessageBox가 닫힐 때 발생하는 이벤트 핸들러 추가
  SmartX.SmartMessageBox.OnNoneBlockingDialogClosing += new SmartX.SmartMessageBox.
                                DialogCloseEventHandler (SmartMessageBox_OnNoneBlockingDialogClosing);
  // SmartMessageBox를 NoneBlocking 방식으로 호출
 SmartX.SmartMessageBox.Show_NoneBlocking( " Show_NoneBlocking " , " None-Blocking " , " 버튼1 " , " 버튼2 " ,
                                        " 버튼3 " , SmartX.CMsgBoxIcon.Customize);
  // 버튼, 캡션 메시지의 텍스트를 변경
  SmartX.SmartMessageBox.SetButsText_NoneBlocking( " 버튼1 변경 " , " 버튼2 변경 " , " 버튼3 변경 " );
  SmartX.SmartMessageBox.SetCaption_NoneBlocking( " None-Blocking 변경 " );
  SmartX.SmartMessageBox.SetMessage_NoneBlocking( " Show_NoneBlocking 변경 " );
}

// SmartMessageBox가 닫힐 때 발생하는 이벤트
void SmartMessageBox_OnNoneBlockingDialogClosing(SmartX.CDialogResult dResult)
{
  if (dResult == SmartX.CDialogResult.No)
  { /* 선택 안 함. 즉, Close_NoneBlocking() 메소드가 호출됨 */ }
  else if (dResult == SmartX.CDialogResult.Button3)
  { /* 버튼3이 선택됨에 따른 코드 작성 */ }
}
```



[출력 결과]

4) SmartMessageBox 인터페이스 설명

SmartMessageBox Component Interface		
속성		
OwnerHandle : static IntPtr	TopMost : static bool	
메소드		
Close_NoneBlocking() : static void	ICONImage(Image imgICON) : static void	SetButsText_NoneBlocking(string strBut1Text, string strBut2Text, string strBut3Text) : static void
SetButtonFont(Font buttonFont) : static void	SetCaption_NoneBlocking(string strCaption) : static void	SetCaptionFont(Font captionFont) : static void
SetMessage_NoneBlocking(string strMessage) : static void	SetMessageFont(Font messageFont) : static void	Show(string text) : static DialogResult (+3개 오버로드)
Show_Ex(string text, string caption, string but1Text, CMsgBoxIcon MsgBoxIcon) : static CDialogResult (+2개 오버로드)	Show_NoneBlocking(string text, string caption, string but1Text, string but2Text, string but3Text, CMsgBoxIcon MsgBoxIcon) : static void	Size(int iWidth, int iHeight) : static void
ThemeColor(THEMECOLOR tColor) : static void		
이벤트		
OnNoneBlockingDialogClosing : SmartMessageBox.DialogCloseEventHandler		

중요 SmartMessageBox는 Static(정적) 속성과 메소드로 구성되어 인스턴싱 없이 사용합니다.

정적 프로퍼티(속성) OwnerHandle

Show_NoneBlocking() 메소드를 호출하여 SmartMessageBox가 None-Blocking으로 출력되었을 때, 부모 폼의 처리를 계속하기 위한 속성입니다.

참고 "Show_NoneBlocking() 메소드" 내용을 참고하시기 바랍니다.

- `static IntPtr` : SmartMessageBox를 출력한 컨트롤의 핸들

C# 사용법

```
SmartMessageBox.OwnerHandle = this;  // 출력한 컨트롤의 핸들을 넘김
```

VB 사용법

```
SmartMessageBox.OwnerHandle = Me
```

정적 프로퍼티(속성) TopMost

SmartMessageBox의 프로그램상 Z-Order 배치를 제일 상단으로 설정합니다. Form(SmartForm)에 의해 SmartMessageBox가 가려지는 것을 방지할 수 있습니다.

- `static bool` : SmartMessageBox의 Z-Order 배치 최상단 여부
  - `true` : Z-Order를 최상단으로 설정
  - `false` : Z-Order를 최상단으로 설정하지 않음

C# 사용법

```
SmartMessageBox.TopMost = true;  // SmartMessageBox의 Z-Order를 최상단으로 설정
```

VB 사용법

```
SmartMessageBox.TopMost = true
```

정적 메소드(함수) Size

SmartMessageBox의 사이즈를 설정합니다.

중요 Show(), Show_Ex() 메소드 호출 전 설정하시기 바랍니다.

◉ `static void Size(int iWidth, int iHeight)`

[인자]

- `int iWidth` : 가로 길이 (최소값 : 50)
- `int iHeight` : 세로 길이 (최소값 : 30)

[표] 인자값에 따른 출력 예시

SmartMessageBox.Size(250, 150)	SmartMessageBox.Size(150, 250)
	

C# 사용법

```
SmartMessageBox.Size(300, 500);  // Width = 300, Height = 500으로 설정
```

VB 사용법

```
SmartMessageBox.Size(300, 500)
```

정적 메소드(함수) ThemeColor

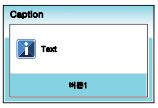
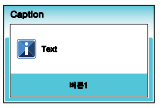
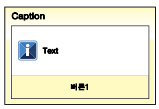
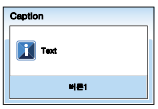
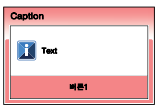
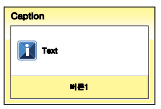
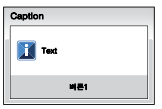
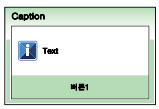
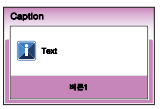
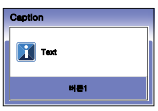
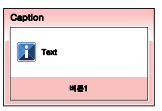
SmartMessageBox의 테마(색상)을 설정합니다. 총 54가지의 테마를 지원합니다.

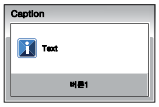
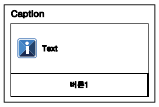
◉ `static void ThemeColor(THEMECOLOR tColor)`

[인자]

• `THEMECOLOR tColor` : SmartMessageBox 테마 (기본값 : HNS)

[표] SmartMessageBox 지원 테마

HNS	AQUAMARINE	BLACK	BLUE	CYAN
				
DARKBLUE	**DARKGRAY**	**DARKGREEN**	**DARKOLIVER**	**DARKPURPLE**
				
DARKSLATEGRAY	**DEEPBLUE**	**DEEPNAVY**	**DEEPPURPLE**	**FIREBRICK**
				
FUCHSIA	**GRAY**	**GREEN**	**INDIGO**	**KHAKI**
				
LEMONCHIFFON	**LIGHTBLUE**	**LIGHTCORAL**	**LIGHTGOLDENROD**	**LIGHTGRAY**
				
LIGHTGREEN	**LIGHTPURPLE**	**LIGHTSKYBLUE**	**LIME**	**MAGENTA**
				
MAROON	**MEDIUMPURPLE**	**MEDIUMSEAGREEN**	**MIDNIGHTBLUE**	**NAVY**
				
NEUTRALGRAY	**OLIVE**	**ORCHID**	**PALEGREEN**	**PALEPINK**
				
PALETURQUOISE	**PINEGREEN**	**PINK**	**PLUSH**	**PURPLE**

RED	ROYALBLUE	SILVER	SKYBLUE	TEAL
				
TURQUOISE	**VIOLET**	**WHITE**	**YELLOW**	**-**
				-

C# 사용법

```
SmartMessageBox.ThemeColor(THEMECOLOR.HNS);  // 디자인 테마를 HNS로 설정합니다.
```

VB 사용법

```
SmartMessageBox.ThemeColor(THEMECOLOR.HNS)
```

정적 메소드(함수) ICONimage

Image

SmartMessageBox에 표현되는 아이콘을 사용자 이미지로 설정합니다.

중요

1. Show_Ex() 메소드 호출 시 적용되며, 반드시 Show_Ex() 메소드 호출 전에 설정되어야 합니다.
2. Show_Ex() 메소드의 CMsgBoxIcon 인자값을 반드시 Customize로 해야 적용됩니다.
3. 아이콘 이미지의 사이즈는 반드시 32 X 32 사이즈로 제작되어야 합니다.

[표] 사용자 아이콘 출력 예시

기존 아이콘 이미지	사용자 아이콘 이미지
	

◉ `static void ICONImage(Image imgICON)`

[인자]

- `Image imgICON` : 아이콘 이미지

Image 이미지 사용 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 “이미지 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
SmartMessageBox.ICONImage(Resource1.TEST);
SmartMessageBox.Show_Ex(“123”, “123”, “but1”, “but2”, “but3”, CMsgBoxIcon.Customize);
```

VB 사용법

```
SmartMessageBox.ICONImage(SmartMessageBoxEx.My.Resources.Resource1.TEST)
SmartMessageBox.Show_Ex(“123”, “123”, “but1”, “but2”, “but3”, CMsgBoxIcon.Customize)
```

정적 메소드(함수) SetButtonFont, SetCaptionFont, SetMessageFont

- `SetButtonFont()` : Button의 폰트를 설정합니다.
- `SetCaptionFont()` : Caption의 폰트를 설정합니다.
- `SetMessageFont()` : Message의 폰트를 설정합니다.

◉ static void SetButtonFont(Font buttonFont)
◉ static void SetCaptionFont(Font captionFont)
◉ static void SetMessageFont(Font messageFont)

[인자]

• Font buttonFont, captionFont, messageFont : Button, Caption, Message의 폰트

C# 사용법

```
Font font = new Font( " 바탕 " , 12, FontStyle.Regular);
SmartMessageBox.SetButtonFont(font);   // Button 폰트 설정
SmartMessageBox.SetCaptionFont(font);  // Caption 폰트 설정
SmartMessageBox.SetMessageFont(font);  // Message 폰트 설정
```

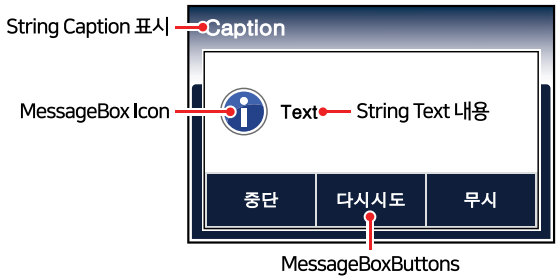
VB 사용법

```
Dim font As New Font( " 바탕 " , 12, FontStyle.Regular)
SmartMessageBox.SetButtonFont(font) : SmartMessageBox.SetCaptionFont(font)
SmartMessageBox.SetMessageFont(font)
```

정적 메소드(함수) Show

SmartMessageBox를 출력합니다.

참고 기존의 MessageBox와 동일한 기능으로, 기존 MessageBox 코드를 SmartMessageBox로 변경하여 바로 적용이 가능합니다. "Blocking 방식 호출 VS None-Blocking 방식 호출", "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

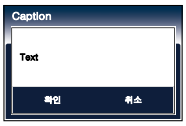


◉ static DialogResult Show(string text)
◉ static DialogResult Show(string text, string caption)
◉ static DialogResult Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIconicon icon)
◉ static DialogResult Show(string text, string caption, MessageBoxButtons buttons, MessageBoxIconicon, icon, MessageBoxDefaultButton dummy)

[인자]

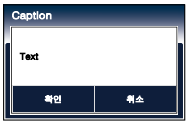
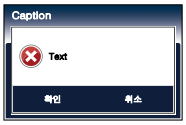
• string text : Message 영역에 표시할 텍스트
• string caption : Caption 영역에 표시할 텍스트
• MessageBoxButtons buttons : 표시할 버튼을 설정

[표1] 인자값에 따른 메시지 박스 버튼 출력

OK	OKCancel	AbortRetryIgnore
		
YesNoCancel	**YesNo**	**RetryCancel**
		

• MessageBoxIcon icon : 아이콘 설정

[표2] 인자값에 따른 아이콘 출력

None	Hand	Question
		
Exclamation	**Asterisk**	**-**
		-

• MessageBoxDefaultButton dummy : 기본 선택 버튼 설정(지원 안 함)

[리턴값]

• DialogResult : 메시지박스 버튼 입력 결과 ※ 인자값으로 설정한 버튼 중 클릭한 버튼이 리턴됩니다.

C# 사용법

```
DialogResult dResult = SmartMessageBox.Show( " Text " , " Caption " , MessageBoxButtons.OKCancel,
MessageBoxIcon.Asterisk);  // SmartMessageBox 출력 후 클릭된 버튼을 저장
```

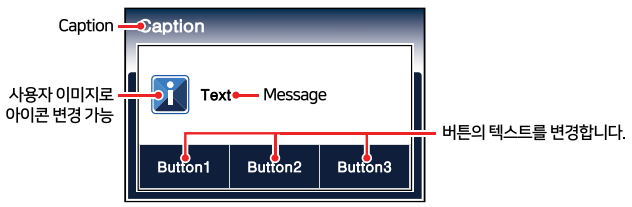
VB 사용법

```
Dim dResult As DialogResult = SmartMessageBox.Show( " Text " , " Caption " , MessageBoxButtons.OK,
MessageBoxIcon.Asterisk)
```

정적 메소드(함수) Show_Ex

Show() 메소드의 기능을 확장한 메소드로, Button 텍스트 변경 기능, ICONimage 속성에서 설정한 사용자 아이콘 출력 기능을 제공합니다.

참고 "Blocking 방식 호출 VS None-Blocking 방식 호출", "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

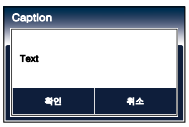
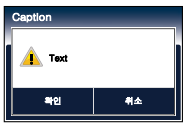
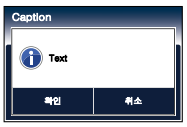


◉ static CDialogResult Show_Ex(string text, string caption, string but1Text, CMsgBoxIcon MsgBoxIcon)
◉ static CDialogResult Show_Ex(string text, string caption, string but1Text, string but2Text, CMsgBoxIcon MsgBoxIcon)
◉ static CDialogResult Show_Ex(string text, string caption, string but1Text, string but2Text, string but3Text, CMsgBoxIcon MsgBoxIcon)

[인자]

- string text : Message 영역에 표시할 텍스트
- string caption : Caption 영역에 표시할 텍스트
- string but1Text, but2Text, but3Text : 각 버튼에 해당하는 텍스트 설정. but2Text, but3Text 인자값은 생략이 가능하며, 생략 시 버튼이 출력되지 않습니다.
- CMsgBoxIcon MsgBoxIcon : 아이콘 설정. 사용자 아이콘

[표] 인자값에 따른 아이콘 출력

None	Hand	Question
		
Exclamation	**Asterisk**	**Customize (예시)**
		 ICONImage() 에서 설정된 이미지

[리턴값]

- CDialogResult : 버튼 클릭 결과

열거값	설명	열거값	설명
Button1	버튼1이 클릭됨	Button3	버튼3이 클릭됨
Button2	버튼2가 클릭됨	No	클릭된 버튼이 없음

C# 사용법

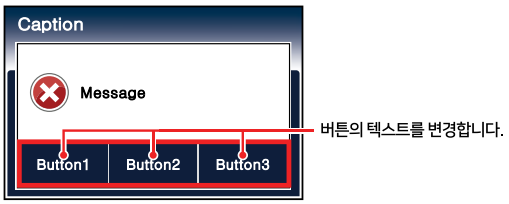
```
CDialogResult cdResult = SmartMessageBox.Show_Ex( " Text " , " Caption " , " But1 " , " But2 " , " But3 " , CMsgBoxIcon.Customize);  // SmartMessageBox 출력 후 클릭된 버튼을 저장
```

VB 사용법

```
Dim cdResult As CDialogResult = SmartMessageBox.Show_Ex( " Text " , " Caption " , " But1 " , " But2 " , " But3 " , CMsgBoxIcon.Customize)
```

정적 메소드(함수) SetButsText_NoneBlocking

Show_NoneBlocking() 메소드를 호출하여 SmartMessageBox가 None-Blocking으로 화면에 출력된 상태에서 Smart MessageBox의 각 버튼 텍스트를 변경합니다.



참고 "Show_NoneBlocking() 메소드" 내용을 참고하시기 바랍니다.

◉ `static void SetButsText_NoneBlocking(string strBut1Text, string strBut2Text, string strBut3Text)`

[인자]

- `string strBut1Text` : 변경할 버튼1 텍스트
- `string strBut2Text` : 변경할 버튼2 텍스트
- `string strBut3Text` : 변경할 버튼3 텍스트

※ 출력된 SmartMessageBox의 버튼 수에 따라 인자값을 설정하시기 바랍니다.

C# 사용법

```
// SmartMessageBox.SetButsText_NoneBlocking("버튼1", "", ""); // 버튼을 1개만 쓰는 경우
// SmartMessageBox.SetButsText_NoneBlocking("버튼1", "버튼2", ""); // 버튼을 2개만 쓰는 경우
SmartMessageBox.SetButsText_NoneBlocking("버튼1", "버튼2", "버튼3"); // 버튼을 3개 쓰는 경우
```

VB 사용법

```
SmartMessageBox.SetButsText_NoneBlocking("버튼1", "", "")
SmartMessageBox.SetButsText_NoneBlocking("버튼1", "버튼2", "")
SmartMessageBox.SetButsText_NoneBlocking("버튼1", "버튼2", "버튼3")
```

정적 메소드(함수) SetCaption_NoneBlocking

Show_NoneBlocking() 메소드를 호출하여 SmartMessageBox가 None-Blocking으로 화면에 출력된 상태에서 Smart MessageBox의 Caption 영역의 텍스트를 변경합니다.

참고 "Show_NoneBlocking() 메소드" 내용을 참고하시기 바랍니다.

◉ `static void SetCaption_NoneBlocking(string strCaption)`

[인자]

- `string strCaption` : 변경할 Caption 영역의 텍스트

C# 사용법

```
SmartMessageBox.SetCaption_NoneBlocking("캡션의 텍스트 설정");  // 캡션의 텍스트를 변경
```

VB 사용법

```
SmartMessageBox.SetCaption_NoneBlocking("캡션의 텍스트 설정")
```

정적 메소드(함수) SetMessage_NoneBlocking

Show_NoneBlocking() 메소드를 호출하여 SmartMessageBox가 None-Blocking으로 화면에 출력된 상태에서 Smart MessageBox의 Message 영역의 텍스트를 변경합니다.

참고 "Show_NoneBlocking() 메소드" 내용을 참고하시기 바랍니다.

◉ `static void SetMessage_NoneBlocking(string strMessage)`

[인자]

• `string strMessage` : 변경할 Message 영역의 텍스트

C# 사용법

```
SmartMessageBox.SetMessage_NoneBlocking( " 메시지의 텍스트를 설정 " );  // 메시지의 텍스트를 변경
```

VB 사용법

```
SmartMessageBox.SetMessage_NoneBlocking( " 메시지의 텍스트를 설정 " );
```

정적 메소드(함수) Close_NoneBlocking

Show_NoneBlocking() 메소드를 호출하여 SmartMessageBox가 None-Blocking으로 출력되었을 때, 출력된 Smart MessageBox를 닫습니다.

참고 "Show_NoneBlocking() 메소드" 내용을 참고하시기 바랍니다.

◉ `static void Close_NoneBlocking()`

C# 사용법

```
SmartMessageBox.Close_NoneBlocking();  // SmartMessageBox를 Closing하는 메서드
```

VB 사용법

```
SmartMessageBox.Close_NoneBlocking()
```

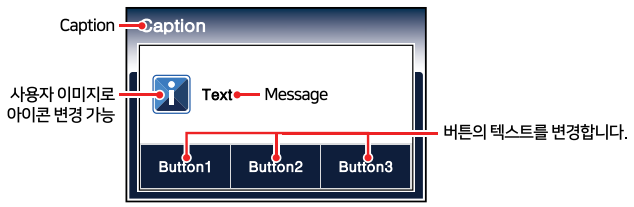
정적 메소드(함수) Show_NoneBlocking

SmartMessageBox를 None-Blocking 방식으로 출력합니다. 즉, SmartMessageBox 창을 Modal 방식으로 띄우고 부모 폼(배경이 되는 폼)은 None-Blocking 방식으로 처리하여 SmartMessageBox 창이 띄워져 있는 동안에도 부모 폼의 작업 처리가 진행되게 합니다.

중요 Show_NoneBlocking() 메소드로 SmartMessageBox 창을 띄운 경우 Show() 또는 Show_Ex() 메소드와 달리 MessageBox의 버튼 선택 결과는 OnNoneBlockingDialogClosingEvent를 통해서만 결과를 확인할 수 있습니다.
자세한 내용은 "OnNoneBlockingDialogClosingEvent" 내용을 참고하시기 바랍니다.

참고 "Blocking 방식 호출 VS None-Blocking 방식 호출", "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

참조 None-Blocking 방식의 메시지박스 활용 방법 관련하여 "홈페이지 → 자료실 → Tech Note → 77. [C#, VB.NET] SmartMessageBox를 활용하여 일정 시간 후 닫히는 메시지박스 구현 방법"을 참조하시기 바랍니다.



◉ `static void Show_NoneBlocking(string text, string caption, string but1Text, string but2Text, string but3Text, CMsgBoxIcon MsgBoxIcon)`

[인자]

• `string text` : Message 영역에 표시할 텍스트
• `string caption` : Caption 영역에 표시할 텍스트
• `string but1Text, but2Text, but3Text` : 각 버튼에 해당하는 텍스트 설정. but2Text, but3Text 인자값은 생략이 가능하며, 생략 시 버튼이 출력되지 않습니다.

• CMsgBoxIcon MsgBoxIcon : 아이콘 설정. 사용자 아이콘

[표] 인자값에 따른 아이콘 출력

None	Hand	Question
		
Exclamation	**Asterisk**	**Customize (예시)**
		 ICONImage() 에서 설정된 이미지

C# 사용법

```
// 활성화된 윈도우 창의 핸들을 SmartMessageBox로 넘김
SmartMessageBox.OwnerHandle = this.Handle;
// Closing 이벤트 추가
SmartMessageBox.OnNoneBlockingDialogClosing += new SmartMessageBox.DialogCloseEventHandler
(SmartMessageBox_OnNoneBlockingDialogClosing);
// SmartMessageBox를 NoneBlocking 방식으로 호출
SmartMessageBox.Show_NoneBlocking( " STEP-1 " , " " , " B1 " , " B2 " , " B3 " , CMsgBoxIcon.Question);
smartLabel3.Text = " STEP-2 ";  // ← 소스 코드 계속 진행
```

VB 사용법

```
SmartMessageBox.OwnerHandle = Me.Handle
AddHandler SmartMessageBox.OnNoneBlockingDialogClosing, AddressOf SmartMessageBox_
                          OnNoneBlockingDialogClosing
SmartMessageBox.Show_NoneBlocking( " STEP-1 " , " " , " B1 " , " B2 " , " B3 " , CMsgBoxIcon.Question)
smartLabel3.Text = " STEP-2 "
```

정적 이벤트 OnNoneBlockingDialogClosing

Show_NoneBlocking() 메서드로 호출된 SmartMessageBox 창이 닫힐 때 발생하는 이벤트로, 사용자가 선택한 버튼을 인자값으로 받습니다.

참고 자세한 사용법은 "프로그래밍 적용 가이드" 내용을 참고하시기 바랍니다.

◉ static event SmartMessageBox.DialogCloseEventHandler OnNoneBlockingDialogClosing(CDialogResult dResult)

[인자]

• CDialogResult dResult : 사용자가 선택한 버튼

• CDialogResult : 버튼 클릭 결과

열거값	설명	열거값	설명
Button1	버튼1이 클릭됨	Button3	버튼3이 클릭됨
Button2	버튼2가 클릭됨	No	클릭된 버튼이 없음

C# 사용법

```
// NoneBlocking 방식의 SmartMessageBox가 닫힐 때 발생하는 메서드, 인자는 사용자가 선택한 버튼
void SmartMessageBox_OnNoneBlockingDialogClosing(CDialogResult dResult)
{
  // 선택 안 함. 즉, Close_NoneBlocking() 메소드가 호출됨
```

```
  if (dResult == CDialogResult.No)
  {
    smartButton3.Text = "NO";
  }
  // 3번 버튼 선택
  else if (dResult == CDialogResult.Button3)
  {
    smartButton3.Text = "B3";
  }
}
```

VB 사용법

```
Private Sub SmartMessageBox_OnNoneBlockingDialogClosing(ByVal dResult As CDialogResult)
  If dResult = CDialogResult.No Then
    smartButton3.Text = "NO"
  ElseIf dResult = CDialogResult.Button3 Then
    smartButton3.Text = "B3"
  End If
End Sub
```

5) SmartMessageBox 예제 사용하기

SmartMessageBox를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartMessageBox"

11. SmartSeparatorLine

SmartSeparatorLine은 선 또는 구분선을 폼 디자이너에서 직접 그리는 기능을 합니다. 일반적으로 구분선은 화면에서 사용자 Control들의 구역을 구분하기 위하여 사용되며, 기본 스타일은 입체감이 있는 선을 그려줍니다. 또한, 색상 및 선의 두께를 변경할 수 있으며 가로/세로 방향의 선을 표현할 수 있습니다.

- 입체감 있는 선 지원
- 선의 색상 및 두께 변경 기능
- 수평/수직의 선 표현

1) SmartSeparatorLine 인터페이스 설명

SmartSeparatorLine Component Interface		
속성		
InitVisible : bool	Line1Color : Color	Line1Width : float
Line2Color : Color	Line2Visible : bool	Line2Width : float
LineDirection : SmartSeperatorLine.DIR		

프로퍼티(속성) InitVisible

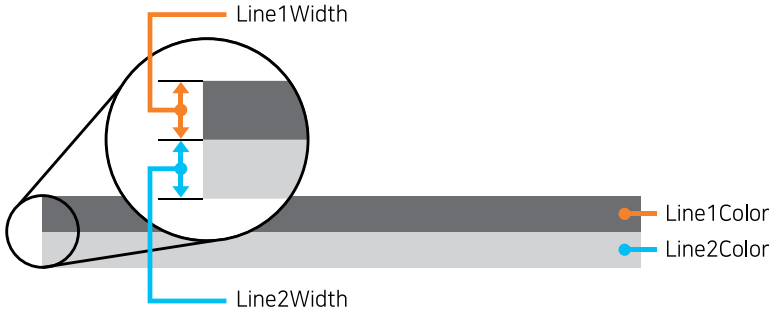
SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리 할 수 있는 속성입니다. 공통 영역에 위치한 SmartSeparatorLine의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

사용법

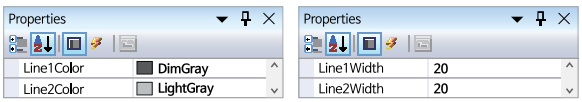
참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) Line1Color, Line2Color, Line1Width, Line2Width

Line1, Line2의 색상과 두께를 설정합니다.



사용법

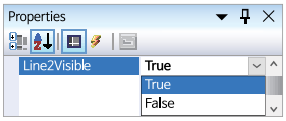


프로퍼티(속성) `Line2Visible`

Line2가 폼 화면에 표시되는지 여부를 설정합니다.

- `bool` : Line2 표시 여부
  - `true` : 표시
  - `false` : 표시 안 함

사용법

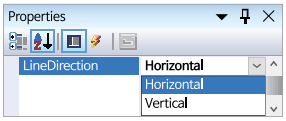


프로퍼티(속성) `LineDirection`

Line(선)의 상태를 수평, 수직으로 설정합니다.

- `SmartSeparatorLine.DIR.Horizontal` : 수평 선
- `SmartSeparatorLine.DIR.Vertical` : 수직 선

사용법



2) SmartSeparatorLine 예제 사용하기

SmartSeparatorLine은 예제를 따로 제공하지 않으며, 아래를 참고하시기 바랍니다.

[SmartSeparatorLine이 적용되기 전 모습]

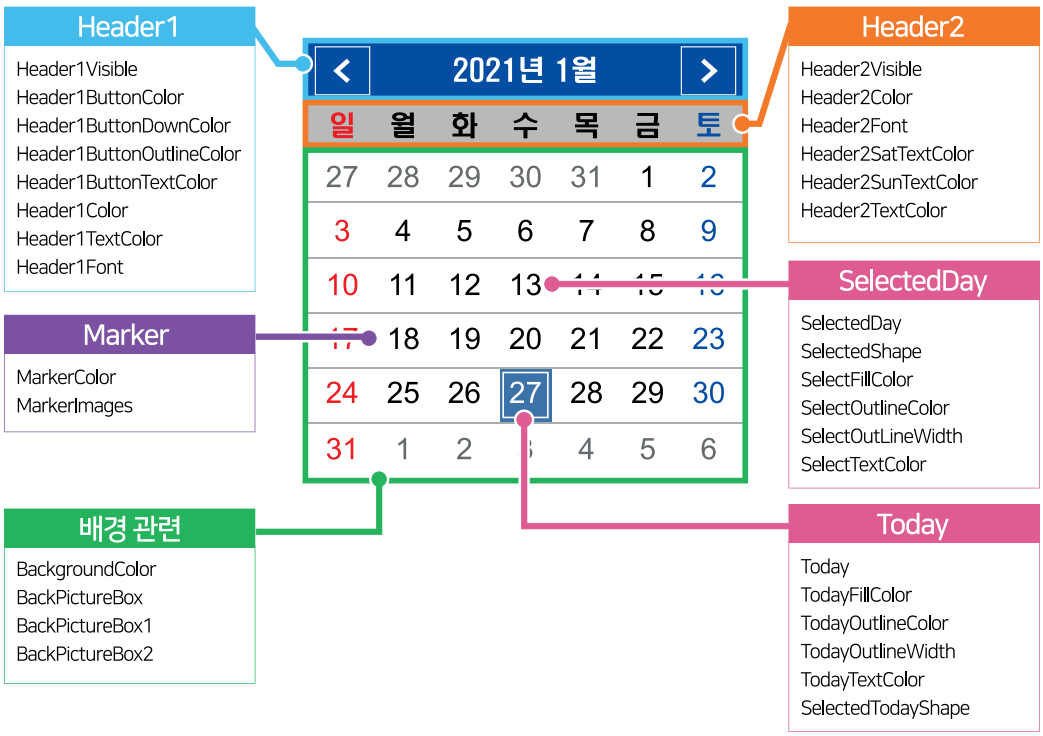
[SmartSeparatorLine이 적용된 모습]

12. SmartMonthCalendar

SmartMonthCalendar는 .NET Compact Framework에서 지원되는 MonthCalendar 컴포넌트에 디자인적인 요소를 추가하였으며 그 밖에 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 MonthCalendar의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- 다양한 형태의 디자인 적용 기능
- 각 디자인적 요소별 색상 변경 기능
- 다양한 디자인 효과 설정 기능
- Calendar의 크기를 자유롭게 변경
- 모든 디자인적 요소의 이미지 사용 및 투명 처리 기능
- 편리한 이벤트 기능 지원
- 일 Marker 기능 지원
  - ↳ 일정 관련 처리 기능

1) 디자인 요소별 속성 명칭



2) 프로그래밍 적용 가이드

SmartMonthCalendar 디자인 시 이미지를 사용하는 경우에 따라 디자인 방법에 약간의 차이가 있습니다. 따라서 아래 STEP 중 일부는 이미지 사용 여부에 따라 CASE가 나눠지며, SmartMonthCalendar에서 이미지는 배경 컴포넌트와 달 이동 버튼인 SmartButton Up/Down, Marker에 사용됩니다. 버튼과 Marker의 경우 마스킹 처리를 해야 합니다.

STEP-1 배경 속성 설정하기

SmartMonthCalendar의 Header 영역을 제외한 배경 색상을 BackgroundColor 속성을 사용해 색상으로 설정하는 방법과, BackPictureBox 속성을 사용해 배경 컨트롤의 이미지를 투영시키는 방법이 있습니다.

※ 자세한 내용은 "BackColor, BackPictureBox 속성" 내용을 참고하시기 바랍니다.

[CASE-1] 색상으로 설정하는 경우

[표] 관련 속성의 속성값 예시

관련 속성	속성값
BackgroundColor	Cyan

[CASE-2] 배경 컨트롤 이미지를 투영시키는 경우

여기서는 적용된 이미지가 Header 영역 부분까지 디자인을 했으므로 Header의 Visible을 False로 설정하시기 바랍니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값	관련 속성	속성값
BackPictureBox	SmartForm	Header2Visible	False
Header1Visible	False	-	-

Image가 적용된 폼

BackPictureBox Image 적용 및 Header 영역 False 설정 전

STEP-2 Header 관련 속성 설정하기

Header 경우 1과 2로 나누어져 있으며, Header1Visible와 Header2Visible 속성을 사용해 Header 영역을 사용하는 방법과 사용하지 않는 방법이 있습니다.

[CASE-1] Header 영역을 사용하는 경우

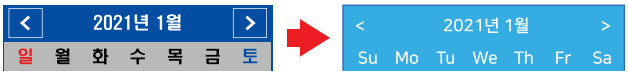
Header1Visible와 Header2Visible 설정이 True로 되어있어야 디자인 적용이 되며, Header1은 이동 버튼 디자인과 Header1 영역의 배경 및 Text 색상, Font를 설정할 수 있고, Header2는 Header2 영역의 배경 색상과 Font, 토요일, 일요일, 평일의 Text 색상, Text의 언어 설정을 할 수 있습니다.

※ 자세한 내용은 "Header1,2 관련 속성" 내용을 참고하시기 바랍니다.

[표] 관련 속성의 속성값 예시

Header1 속성	설정값	Header2 속성	설정값
Header1Visible	True	Header2Visible	True
Header1ButtonColor	DeepSkyBlue	Header2Font	Tahoma, 9pt, style=Bold

Header1ButtonDownColor	DeepSkyBlue	Header2Color	DeepSkyBlue
Header1ButtonOutlineColor	DeepSkyBlue	Header2SatTextColor	White
Header1ButtonTextColor	White	Header2SunTextColor	White
Header1Color	DeepSkyBlue	Header2TextColor	White
Header1TextColor	White	HeaderDayOfWeekLanguag	English
Header1Font	Tahoma, 9pt, style=Bold	-	-



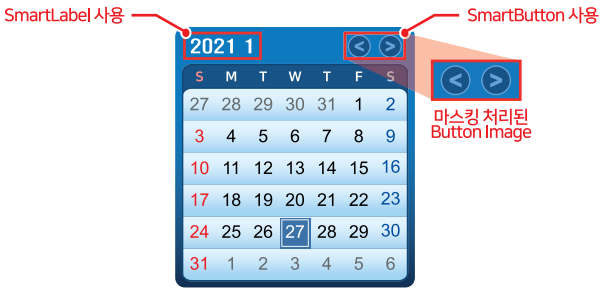
[CASE-2] Header 영역을 사용하지 않는 경우

STEP-1에서 설정한 이미지 디자인에 따라 Header1Visible와 Header2Visible 설정을 False로 설정하며, 현재 연도와 달을 나타낼 SmartLabel과 달 이동 버튼으로 쓰일 SmartButton이 필요합니다. NowYearMonth(), BackMonth(), NextMonth() 메소드를 호출하여 달력의 달 이동과 현재 연도, 달을 출력할 수 있습니다.

※ 자세한 내용은 "SmartButton, SmartLebel 인터페이스 설명"과 "NowYearMonth(), BackMonth(), NextMonth() 메소드" 내용을 참고하시기 바랍니다.

[표] 관련 속성의 속성값 예시

Header1 속성	설정값	Header2 속성	설정값
Header1Visible	False	Header2Visible	False




```
// Back 버튼 클릭 시
private void btn_Back_Click(object sender, EventArgs e)
{
  // Back 메소드 호출(이전 달로 이동)
  smartMonthCalendar1.BackMonth();
  // 현재 연도와 달을 출력
  lbl_Year.Text = smartMonthCalendar1.NowYearMonth.iYear.ToString();
  lbl_Month.Text = smartMonthCalendar1.NowYearMonth.iMonth.ToString();
}

// Next 버튼 클릭 시
private void btn_Next_Click(object sender, EventArgs e)
{
  // Next 메소드 호출 (다음 달로 이동)
  smartMonthCalendar1.NextMonth();
  // 현재 연도와 달을 출력
  lbl_Year.Text = smartMonthCalendar1.NowYearMonth.iYear.ToString();
  lbl_Month.Text = smartMonthCalendar1.NowYearMonth.iMonth.ToString();
}
```

STEP-3 선(외곽선, 구분선) 속성 설정하기

선의 경우 외곽선과 구분선으로 나누어져 있으며, 외곽선은 BorderColor, BorderLineWidth 속성으로 색상 및 두께를 설정하며, 구분선은 SeparateHorLineVisible, SeparateVerLineVisible 속성으로 날짜 영역의 각 행간 및 열간의 구분선을 보일지 말지의 여부와 SeparateLineColor 속성으로 색상을 설정합니다.

[CASE-1] 선(외곽선, 구분선) 속성을 사용하는 경우

[표] 관련 속성의 속성값 예시

외곽선 속성	설정값	구분선 속성	설정값
BorderColor	DeepSkyBlue	SeparateHorLineVisible	True
BorderLineWidth	4	SeparateVerLineVisible	False
-	-	SeparateLineColor	DeepSkyBlue

참고 결과 이미지는 STEP-6의 CASE-1을 참고하시기 바랍니다.

[CASE-2] 선(외곽선, 구분선) 속성을 사용하지 않는 경우

STEP-1에서 설정한 이미지 디자인에 따라 외곽선의 BorderColor 속성을 Transparent로 설정하며, SeparateHorLineVisible와 SeparateVerLineVisible 설정을 False로 설정합니다.

[표] 관련 속성의 속성값 예시

외곽선 속성	설정값	구분선 속성	설정값
BorderColor	Transparent	SeparateHorLineVisible	False
-	-	SeparateVerLineVisible	False

STEP-4 Day 관련 속성 설정하기

Day의 경우 날짜 영역의 일별(평일, 주말) Text 색상, Font를 설정할 수 있습니다. DayTextColor, DayTextSatColor, DayTextSunColor 속성으로 현재 달의 평일과 주말의 Text 색상과 DayTextColor1 속성으로 지난 달과 다음 달의 Text 색상을 설정할 수 있습니다.

[표] 관련 속성의 속성값 예시

관련 속성	속성값
Font	Arial, 10pt, Regular
DayTextColor	Black
DayTextColor1	Gray
DayTextSatColor	Blue
DayTextSunColor	Red

참고 결과 이미지는 STEP-6의 CASE-1과 CASE-2를 참고하시기 바랍니다.

STEP-5 Today, SelectedDay 관련 속성 설정하기

Today의 경우 오늘 일자를 가지며, 오늘 일자의 배경 및 Text 색상과 외곽선의 색상, 두께, 모양을 설정할 수 있습니다.

SelectedDay의 경우 선택된 일자를 가지며, 선택된 일자의 배경 및 Text 색상과 외곽선의 색상, 두께, 모양을 설정할 수 있습니다.

[표] 관련 속성의 속성값 예시

외곽선 속성	설정값	구분선 속성	설정값
TodayFillColor	DeepSkyBlue	SelectedShape	Rectangle
TodayOutlineColor	DeepSkyBlue	SelectedFillColor	Transparent
TodayOutlineWidth	1	SelectedOutlineColor	DeepSkyBlue
TodayTextColor	White	SelectedOutlineWidth	2
SelectedTodayShape	Rectangle	SelectedTextColor	Black

참고 결과 이미지는 STEP-6의 CASE-1과 CASE-2를 참고하시기 바랍니다.

STEP-6 Marker 관련 속성 설정하기

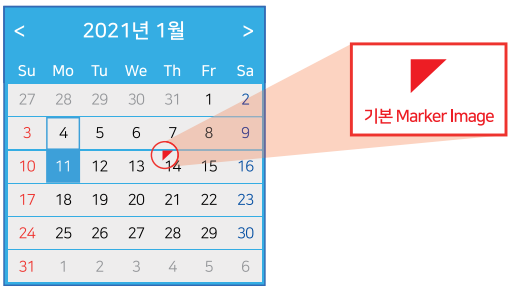
Marker는 MarkerImages 속성과 MarkerInfoAdd() 메소드를 사용해 Maker의 기본 이미지를 사용하는 방법과 사용자 이미지를 사용하는 방법이 있습니다.

※ 자세한 내용은 "MarkerImages 속성"과 "MarkerInfoAdd() 메소드" 내용을 참고하시기 바랍니다.

[CASE-1] Marker 기본 이미지를 사용하는 경우

[표] 관련 속성의 속성값 예시

외곽선 속성	설정값
MarkerColor	Red
MarkerSize	10
MarkerOffsetX	0
MarkerOffsetY	0
MarkerPosition	LeftTop



[CASE-2] Marker 사용자 이미지를 사용하는 경우

MarkerImage 속성에 이미지 추가하여 MarkerOffsetX, MarkerOffsetY 속성으로 Marker의 위치를 설정할 수 있습니다. MarkerInfoAdd() 메소드로 Marker할 일자와 Marker 이미지 Index를 설정하여 호출합니다.
(추가된 이미지의 Index는 1부터 시작됩니다.)

[표] 관련 속성의 속성값 예시

관련 속성	속성값
MarkerOffsetX	0
MarkerOffsetY	0


```
// 제작한 Marker 이미지 추가
smartMonthCalendar1.MarkerImages.Add(Properties.Resources.logo);
// Marker할 일자와 추가된 Marker Index 설정
smartMonthCalendar1.MarkerInfoAdd(2021, 1, 14, 1);
```

3) SmartMonthCalendar 인터페이스 설명

SmartMonthCalendar Component Interface		
속성		
BackgroundColor : Color	BackPictureBox : PictureBox	BackPictureBox1 : SmartInnerForm
BackPictureBox2 : SmartGroupBox	BorderColor : Color	BorderLineWidth : int
DayTextColor : Color	DayTextColor1 : Color	DayTextColorSat : Color
DayTextColorSun : Color	Header1ButtonColor : Color	Header1ButtonDownColor : Color
Header1ButtonOutlineColor : Color	Header1ButtonTextColor : Color	Header1Color : Color
Header1Font : Font	Header1TextColor : Color	Header1Visible : bool
Header2Color : Color	Header2Font : Font	Header2SatTextColor : Color
Header2SunTextColor : Color	Header2TextColor : Color	Header2Visible : bool
HeaderButtonHeightOffset : int	HeaderButtonLeftOffset : int	HeaderButtonTopOffset : int
HeaderButtonWidthOffset : int	HeaderDayOfWeekLanguag : SmartMonthCalendar.HeaderLanguage	InitVisible : bool
MarkerColor : Color	MarkerImages : List<Image>	MarkerOffsetX : int
MarkerOffsetY : int	MarkerPosition : SmartMonthCalendar.MarkerPositions	MarkerSize : int
NowYearMonth : SmartMonthCalendar.MONTHYEAR	SelectedDay : int	SelectedShape : SmartMonthCalendar.SelectShape
SelectedTodayShape : SmartMonthCalendar.SelectShape	SelectFillColor : Color	SelectOutlineColor : Color
SelectOutLineWidth : int	SelectTextColor : Color	SeparateHorLineVisible : bool
SeparateLineColor : Color	SeparateVerLineVisible : bool	TextOffsetX : int
TextOffsetY : int	Today : DateTime	TodayFillColor : Color
TodayOutlineColor : Color	TodayOutlineWidth : int	TodayTextColor : Color
메소드		
BackMonth() : void	NextMonth() : void	MarkerInfoAdd() : void (+1개 오버로드)
MarkerInfoFind() : int	MarkerInfoRemove() : void	MarkerInfoRemoveAll() : void
MarkerInfosUpdate() : void		
이벤트		
OnMonthChange : SmartMonthCalendar.MonthChangeHandler	OnSelectedDayChange : SmartMonthCalendar.SelectedDayChangeHandler	

프로퍼티(속성) `InitVisible`

SmartForm을 이용한 MDI 조합 구성 시 공통 영역이 있을 경우 해당 사용자 인터페이스 컴포넌트가 사라지는 현상을 방지하며 화면 전환 시 좀 더 부드럽게 처리 할 수 있는 속성입니다. 공통 영역에 위치한 SmartMonthCalendar의 InitVisible 속성값을 False로 설정하시면 화면 전환 시 좀 더 자연스럽게 처리됩니다.

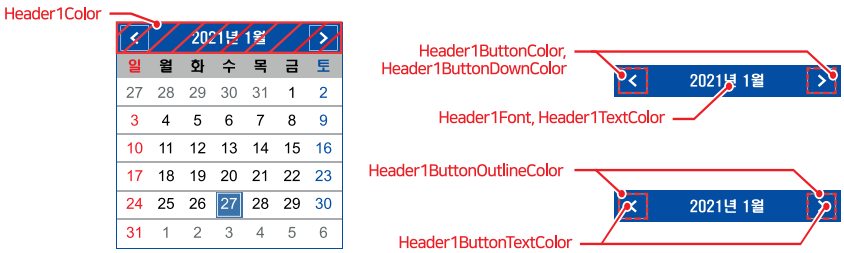
사용법

참고 자세한 사용법은 SmartForm의 "공통 영역이 있는 MDI 구성 시 InitVisible 속성값 설정"을 참고하시기 바랍니다.

프로퍼티(속성) `Header1ButtonColor, Header1ButtonDownColor, Header1ButtonOutlineColor, Header1ButtonTextColor, Header1Color, Header1Font, Header1TextColor`

- `Header1ButtonColor` : 헤더1(Header1)영역 이동 버튼의 눌러지지 않은 상태의 색상을 설정합니다.
- `Header1ButtonDownColor` : 헤더1(Header1)영역 이동 버튼의 눌러진 상태의 색상을 설정합니다.
- `Header1ButtonOutlineColor` : 헤더1(Header1)영역 이동 버튼의 외곽선 색상을 설정합니다.
- `Header1ButtonTextColor` : 헤더1(Header1)영역 이동 버튼의 Text 색상을 설정합니다.
- `Header1Color` : 헤더1(Header1)영역의 배경 색상을 설정합니다.

- Header1Font : 헤더1(Header1)영역의 Font 스타일을 설정합니다.
- Header1TextColor : 헤더1(Header1)영역의 연도와 월의 Text 색상을 설정합니다.



C# 사용법

```
// 이동 버튼의 OFF 색상
smartMonthCalendar1.Header1ButtonColor = Color.Navy;
// 이동 버튼의 ON 색상
smartMonthCalendar1.Header1ButtonDownColor = Color.Gray;
// 이동 버튼의 외곽선(OutLine) 색상
smartMonthCalendar1.Header1ButtonOutlineColor = Color.Gray;
// 이동 버튼의 Text 색상
smartMonthCalendar1.Header1ButtonTextColor = Color.Yellow;
// 헤더1의 배경 색상
smartMonthCalendar1.Header1Color = Color.LightBlue;
// 헤더1의 연도와 월의 Text 색상
smartMonthCalendar1.Header1TextColor = Color.Black;
// 헤더1의 폰트 설정
Font fo = new Font( " 굴림 " , 9, FontStyle.Bold);
smartMonthCalendar1.Header1Font = fo;
```

VB 사용법

```
smartMonthCalendar1.Header1ButtonColor = Color.Navy
smartMonthCalendar1.Header1ButtonDownColor = Color.Gray
smartMonthCalendar1.Header1ButtonOutlineColor = Color.Gray
smartMonthCalendar1.Header1ButtonTextColor = Color.Yellow
smartMonthCalendar1.Header1Color = Color.LightBlue
smartMonthCalendar1.Header1TextColor = Color.Black
Dim fo As New Font( " 굴림 " , 9, FontStyle.Bold)
smartMonthCalendar1.Header1Font = fo
```

프로퍼티(속성) Header1Visible

헤더1(Header1)영역을 폼 화면에 표시되는지 여부를 설정합니다.

- bool : 헤더1(Header1)영역을 폼 화면에 표시 유무
  - true : 표시함
  - false : 표시 안 함

True (보이기)	False (숨기기)
	

C# 사용법

```
smartMonthCalendar1.Header1Visible = true;  // 보이기
```

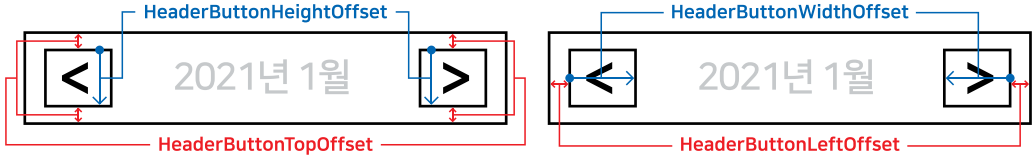
VB 사용법

```
smartMonthCalendar1.Header1Visible = True
```

프로퍼티(속성) HeaderButtonHeightOffset, HeaderButtonLeftOffset, HeaderButtonTopOffset, HeaderButtonWidthOffset

헤더1(Header) 영역의 버튼 크기 및 여백을 설정합니다.

- HeaderButtonHeightOffset : 버튼의 높이를 설정합니다.
- HeaderButtonWidthOffset : 버튼의 폭을 설정합니다.
- HeaderButtonTopOffset : 버튼의 상하 여백을 설정합니다.
- HeaderButtonLeftOffset : 버튼의 좌우 여백을 설정합니다.



C# 사용법

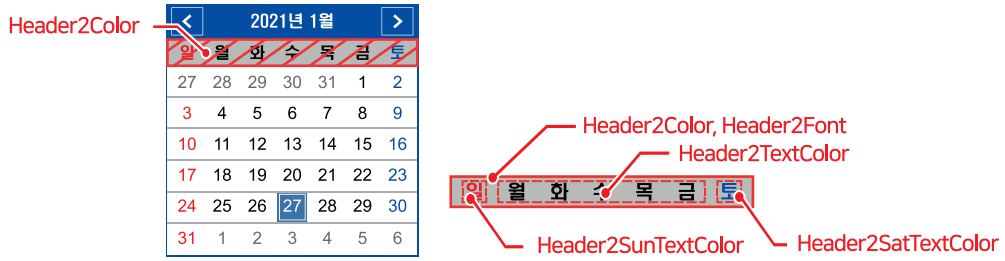
```
// 버튼의 높이를 설정
smartMonthCalendar1.HeaderButtonHeightOffset = 10;
// 버튼의 폭을 설정
smartMonthCalendar1.HeaderButtonWidthOffset = 40;
// 버튼의 상하 여백을 설정
smartMonthCalendar1.HeaderButtonTopOffset = 5;
// 버튼의 좌우 여백을 설정
smartMonthCalendar1.HeaderButtonLeftOffset = 20;
```

VB 사용법

```
smartMonthCalendar1.HeaderButtonHeightOffset = 10
smartMonthCalendar1.HeaderButtonLeftOffset = 20
smartMonthCalendar1.HeaderButtonTopOffset = 5
smartMonthCalendar1.HeaderButtonWidthOffset = 40
```

프로퍼티(속성) Header2Color, Header2Font, Header2SatTextColor, Header2SunTextColor, Header2TextColor

- Header2Color : 헤더2(Header2) 영역의 배경 색상을 설정합니다.
- Header2Font : 헤더2(Header2) 영역의 Font 스타일을 설정합니다.
- Header2SatTextColor : 헤더2(Header2) 영역의 토요일 Text 색상을 설정합니다.
- Header2SunTextColor : 헤더2(Header2) 영역의 일요일 Text 색상을 설정합니다.
- Header2TextColor : 헤더2(Header2) 영역의 평일 Text 색상을 설정합니다.



C# 사용법

```
smartMonthCalendar1.Header2Color = Color.Red;  // 헤더2 영역의 배경 색상
Font fo = new Font("Arial", 9, FontStyle.Italic);  // (일,월,화,수,목,금,토) Font 스타일 설정
smartMonthCalendar1.Header2Font = fo;
smartMonthCalendar1.Header2SatTextColor = Color.Red;  // 토요일 Text 색상
smartMonthCalendar1.Header2SunTextColor = Color.Red;  // 일요일 Text 색상
smartMonthCalendar1.Header2TextColor = Color.Red;  // 평일(월~금) Text 색상
```

VB 사용법

```
smartMonthCalendar1.Header2Color = Color.Red
Dim fo As New Font("Arial", 9, FontStyle.Italic)
smartMonthCalendar1.Header2Font = fo
smartMonthCalendar1.Header2SatTextColor = Color.Red
smartMonthCalendar1.Header2SunTextColor = Color.Red
smartMonthCalendar1.Header2TextColor = Color.Red
```

프로퍼티(속성) Header2Visible

헤더2(Header2) 영역을 폼 화면에 표시되는지 여부를 설정합니다.

- bool : 헤더2(Header2) 영역을 폼 화면에 표시 유무
  - true : 표시함
  - false : 표시 안 함

True (보이기)	False (숨기기)
	

C# 사용법

```
smartMonthCalendar1.Header2Visible = true;  // 보이기
```

VB 사용법

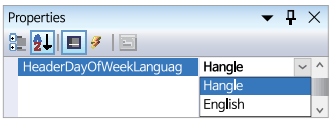
```
smartMonthCalendar1.Header2Visible = True
```

프로퍼티(속성) HeaderDayOfWeekLanguag

헤더2(Header2) 영역에 표시되는 언어(한글,영어)를 설정합니다.

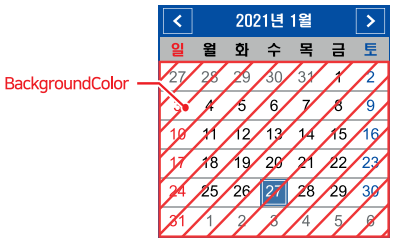
- SmartMonthCalendar.HeaderLanguage.Hangle : 한글
- SmartMonthCalendar.HeaderLanguage.English : 영어

C# 사용법



프로퍼티(속성) BackgroundColor

SmartMonthCalendar의 전체 배경색을 설정합니다.



C# 사용법

```
smartMonthCalendar1.BackgroundColor = Color.White;
```

VB 사용법

```
smartMonthCalendar1.BackgroundColor = Color.White
```

프로퍼티(속성) BackPictureBox, BackPictureBox1, BackPictureBox2

SmartMonthCalendar의 투명 효과 처리를 위해 해당 배경 컴포넌트를 설정합니다. SmartMonthCalendar에 의해 배경이 가려지는 현상을 방지할 수 있는 기능을 제공합니다.

속성		해당 배경 컴포넌트
BackPictureBox	→	PictureBox SmartForm
BackPictureBox1	→	SmartInnerForm
BackPictureBox2	→	SmartGroupBox

주의 배경 컴포넌트의 배경 이미지가 설정되어 있어야 투명 효과가 적용됩니다. 만약 배경 이미지가 없고 BackColor 속성만 설정되어 있을 경우 BackPictureBox 속성을 통한 투명 효과 처리가 적용되지 않습니다.

사용법

프로퍼티(속성) BorderColor, BorderLineWidth

SmartMonthCalendar의 외곽선을 표시하는 경우 외곽선의 색상 및 두께를 설정합니다.

- BorderColor : 외곽선의 색상을 설정합니다.
- BorderLineWidth : 외곽선의 두께를 설정합니다.

C# 사용법

```
smartMonthCalendar1.BorderColor = Color.FromArgb(26, 67, 143);
smartMonthCalendar1.BorderLineWidth = 5;
```

VB 사용법

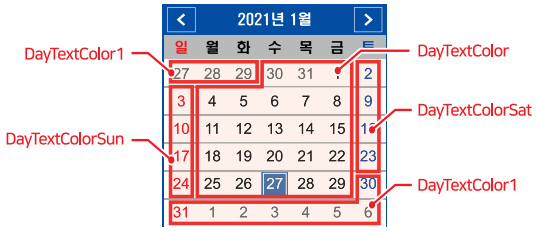
```
smartMonthCalendar1.BorderColor = Color.FromArgb(26, 67, 143)
smartMonthCalendar1.BorderLineWidth = 5
```

프로퍼티(속성) DayTextColor, DayTextColor1, DayTextColorSat, DayTextColorSun

일별(평일, 주말) Text의 색상을 설정합니다.

- DayTextColor : 일별 Text 색상

- DayTextColor1 : 지난 달, 다음 달 일별 Text 색상
- DayTextColorSat : 토요일 Text 색상
- DayTextColorSun : 일요일 Text 색상



C# 사용법

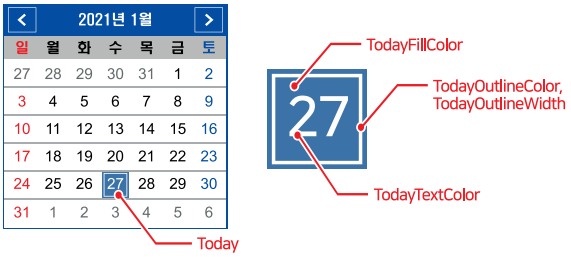
```
smartMonthCalendar1.DayTextColor = Color.Black;  // 일별 Text 색상
smartMonthCalendar1.DayTextColor1 = Color.Gray;  // 지난 달, 다음 달 일별 Text 색상
smartMonthCalendar1.DayTextColorSat = Color.FromArgb(0, 30, 177);  // 토요일 Text 색상
smartMonthCalendar1.DayTextColorSun = Color.FromArgb(177, 0, 0);  // 일요일 Text 색상
```

VB 사용법

```
smartMonthCalendar1.DayTextColor = Color.Black : smartMonthCalendar1.DayTextColor1 = Color.Gray
smartMonthCalendar1.DayTextColorSat = Color.FromArgb(0, 30, 177)
smartMonthCalendar1.DayTextColorSun = Color.FromArgb(177, 0, 0)
```

프로퍼티(속성) Today, TodayFillColor, TodayOutlineColor, TodayOutlineWidth, TodayTextColor

- Today : 오늘 일자를 가져오거나 설정합니다.
- TodayFillColor : 오늘 일자의 배경 색상을 설정합니다.
- TodayOutlineColor : 오늘 일자의 외곽선 색상을 설정합니다.
- TodayOutlineWidth : 오늘 일자의 외곽선 두께를 설정합니다.
- TodayTextColor : 오늘 일자의 Text 색상을 설정합니다.



C# 사용법

```
smartLabel1.Text = smartMonthCalendar1.Today.ToString();  // 오늘 일자를 출력
smartMonthCalendar1.TodayFillColor = Color.FromArgb(58, 109, 204);  // 오늘 일자의 배경 색상
smartMonthCalendar1.TodayOutlineColor = Color.White;  // 오늘 일자의 외곽선(OutLine) 색상
smartMonthCalendar1.TodayOutlineWidth = 1;  // 오늘 일자의 외곽선(OutLine) 두께
smartMonthCalendar1.TodayTextColor = Color.White;  // 오늘 일자의 Text 색상
```

VB 사용법

```
smartLabel1.Text = smartMonthCalendar1.Today.ToString()
smartMonthCalendar1.TodayFillColor = Color.FromArgb(58, 109, 204)
smartMonthCalendar1.TodayOutlineColor = Color.White
smartMonthCalendar1.TodayOutlineWidth = 1
smartMonthCalendar1.TodayTextColor = Color.White
smartMonthCalendar1.TodayOutlineColor = Color.White
```

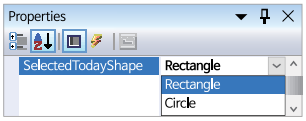
```
smartMonthCalendar1.TodayOutlineWidth = 1
smartMonthCalendar1.TodayTextColor = Color.White
```

프로퍼티(속성) SelectedTodayShape

오늘 일자의 외곽선 모양을 설정합니다.

- SmartMonthCalendar.SelectShape.Rectangle : 사각형 모양
- SmartMonthCalendar.SelectShape.Circle : 원형 모양

사용법



프로퍼티(속성) NowYearMonth

달력에 현재 연도와 월을 표시하며, 특정 연도와 월로 변경하여 설정합니다.

- SmartMonthCalendar.MONTHYEAR : MONTHYEAR 객체
  - int iYear : 연도
  - int iMonth : 월

C# 사용법

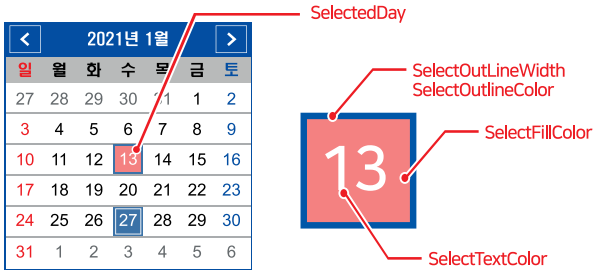
```
// 연, 월을 설정
SmartMonthCalendar.MONTHYEAR monYear = new SmartMonthCalendar.MONTHYEAR();
monYear.iYear = 2021; monYear.iMonth = 1; smartMonthCalendar1.NowYearMonth = monYear;
```

VB 사용법

```
Dim monYear As SmartMonthCalendar.MONTHYEAR = new SmartMonthCalendar.MONTHYEAR()
monYear.iYear = 2021 : monYear.iMonth = 1
smartMonthCalendar1.NowYearMonth = monYear
```

프로퍼티(속성) SelectedDay, SelectFillColor, SelectOutlineColor, SelectOutLineWidth, SelectTextColor

- SelectedDay : 선택된 일자를 가져오거나 설정합니다.
- SelectFillColor : 선택된 일자의 배경 색상을 설정합니다.
- SelectOutlineColor : 선택된 일자의 외곽선 색상을 설정합니다.
- SelectOutLineWidth : 선택된 일자의 외곽선 두께를 설정합니다.
- SelectTextColor : 선택된 일자의 Text 색상을 설정합니다.



C# 사용법

```
smartMonthCalendar1.SelectedDay = 13;  // 현재 보이는 캘린더의 13일을 선택된 모습으로 표시
smartMonthCalendar1.SelectFillColor = Color.Pink;  // 선택된 일자의 배경 색상
smartMonthCalendar1.SelectOutlineColor = Color.FromArgb(58, 109, 204);  // 외곽선(OutLine) 색상
```

```
smartMonthCalendar1.SelectOutLineWidth = 2;  // 외곽선(OutLine) 두께 설정
smartMonthCalendar1.SelectTextColor = Color.FromArgb(58, 109, 204);  // 선택된 일자의 Text 색상
```

VB 사용법

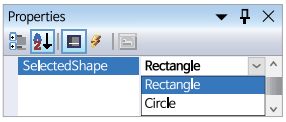
```
smartMonthCalendar1.SelectedDay = 13
smartMonthCalendar1.SelectFillColor = Color.Pink
smartMonthCalendar1.SelectOutlineColor = Color.FromArgb(58, 109, 204)
smartMonthCalendar1.SelectOutLineWidth = 2
smartMonthCalendar1.SelectTextColor = Color.FromArgb(58, 109, 204)
```

프로퍼티(속성) SelectedShape

선택된 일자의 외곽선 모양을 설정합니다.

- SmartMonthCalendar.SelectShape.Rectangle : 사각형 모양
- SmartMonthCalendar.SelectShape.Circle : 원형 모양

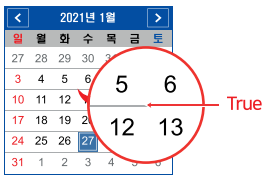
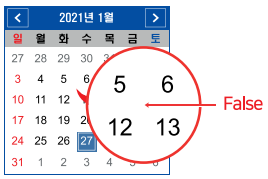
사용법



프로퍼티(속성) SeparateHorLineVisible

날짜 영역의 행간 구분선(가로선)을 보일지 말지의 여부를 설정합니다.

- bool : 행간 구분선(가로선) 표시 유무
  - true : 표시함
  - false : 표시 안 함

True (보이기)	False (숨기기)
	

C# 사용법

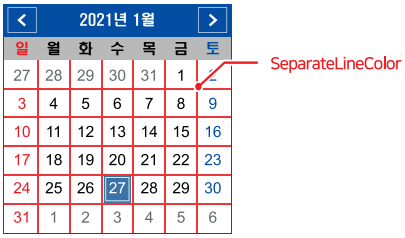
```
smartMonthCalendar1.SeparateHorLineVisible = true;  // 행간 구분선 보이기
```

VB 사용법

```
smartMonthCalendar1.SeparateHorLineVisible = True
```

프로퍼티(속성) SeparateLineColor

날짜 영역의 구분선 색상을 설정합니다.



C# 사용법

```
smartMonthCalendar1.SeparateLineColor = Color.Red;
```

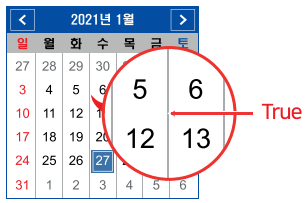
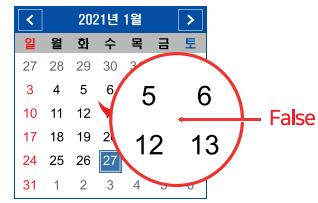
VB 사용법

```
smartMonthCalendar1.SeparateLineColor = Color.Red
```

프로퍼티(속성) SeparateVerLineVisible

날짜 영역의 열간 구분선(세로선)을 보일지 말지의 여부를 설정합니다.

- bool : 열간 구분선(세로선) 표시 유무
  - true : 표시함
  - false : 표시 안 함

True (보이기)	False (숨기기)
	

C# 사용법

```
smartMonthCalendar1.SeparateVerLineVisible = true;  // 열간 구분선 보이기
```

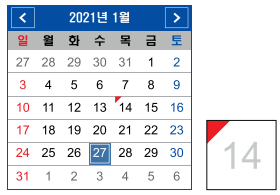
VB 사용법

```
smartMonthCalendar1.SeparateVerLineVisible = True
```

프로퍼티(속성) MarkerSize

일정 관리 Marker의 크기를 설정합니다.

- int : 설정할 Marker 크기 (기본값 : 10)

MarkerSize = 10	MarkerSize = 20
	

C# 사용법

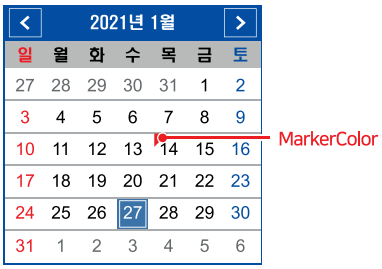
```
// Marker 크기를 20으로 설정
smartMonthCalendar1.MarkerSize = 20;
```

VB 사용법

```
smartMonthCalendar1.MarkerSize = 20
```

프로퍼티(속성) MarkerColor

일정관리 Marker의 색상을 설정합니다.



C# 사용법

```
smartMonthCalendar1.MarkerColor = Color.Red;  // Marker의 색상을 Red로 설정
```

VB 사용법

```
smartMonthCalendar1.MarkerColor = Color.Red
```

프로퍼티(속성) MarkerImages

Masking Image

일정 관리 Marker의 이미지를 설정합니다.

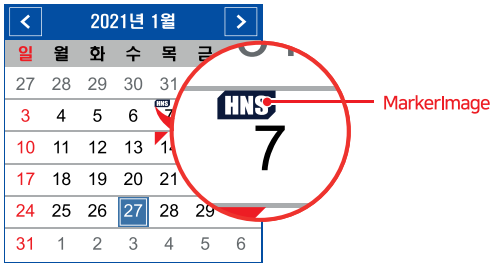


Image Masking 이미지 사용 및 마스킹 처리 시 주의사항

사용되는 이미지가 "이미지 제작 가이드"에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있으며, 마스킹 처리 시 "투명(Masking) 처리 영역 제작 가이드"에 맞게 처리되지 않은 경우, 마스킹 처리가 올바르게 되지 않을 수 있습니다.

※ 반드시 "이미지 제작 가이드"와 "투명(Masking) 처리 영역 제작 가이드"를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
// Marker 설정
SmartMonthCalendar.MARKERDAYINFOS markinfo1 = new SmartMonthCalendar.MARKERDAYINFOS();
smartMonthCalendar1.MarkerImages.Add(Resource1._1);
markinfo1.iYear = 2021; markinfo1.iMonth = 1; markinfo1.iDay = 7; markinfo1.iImageIndex = 1;
smartMonthCalendar1.MarkerInfoAdd(markinfo1.iYear, markinfo1.iMonth, markinfo1.iDay,
markinfo1.iImageIndex);
smartMonthCalendar1.MarkerInfosUpdate();
```

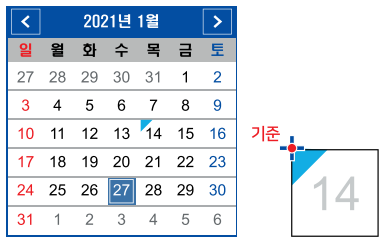
VB 사용법

```
Dim markinfo1 As New SmartMonthCalendar.MARKERDAYINFOS
smartMonthCalendar1.MarkerImages.Add(My.Resources.Resource1._1)
markinfo1.iYear = 2021 : markinfo1.iMonth = 1 : markinfo1.iDay = 7 : markinfo1.iImageIndex = 1
smartMonthCalendar1.MarkerInfoAdd(markinfo1.iYear, markinfo1.iMonth, markinfo1.iDay,
markinfo1.iImageIndex)
smartMonthCalendar1.MarkerInfosUpdate()
```

프로퍼티(속성) MarkerOffsetX, MarkerOffsetY

마킹이 표시되는 해당 셀의 좌측 상단 기준으로부터 X, Y만큼 떨어진 거리에 Marker를 표시합니다.

- MarkerOffsetX : Marker의 X 좌표를 설정합니다.
- MarkerOffsetY : Marker의 Y 좌표를 설정합니다.
- int : Marker의 X, Y 좌표값

좌표값이 0인 경우	좌표값이 10인 경우
	

C# 사용법

```
// 마킹이 표시된 셀의 좌측 상단 기준으로 (10, 10)에 위치 시킴
smartMonthCalendar1.MarkerOffsetX = 10;
smartMonthCalendar1.MarkerOffsetY = 10;
```

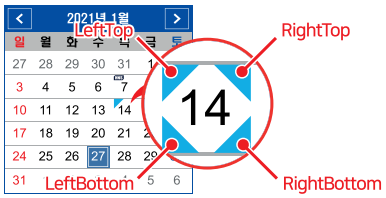
VB 사용법

```
smartMonthCalendar1.MarkerOffsetX = 10
smartMonthCalendar1.MarkerOffsetY = 10
```

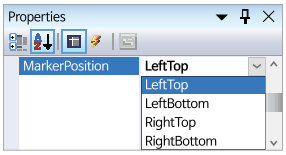
프로퍼티(속성) MarkerPosition

Marker의 위치를 설정합니다.

- SmartMonthCalendar.MarkerPositions.LeftBottom : 셀의 좌측 하단에 위치
- SmartMonthCalendar.MarkerPositions.LeftTop : 셀의 좌측 상단에 위치
- SmartMonthCalendar.MarkerPositions.RightBottom : 셀의 우측 하단에 위치
- SmartMonthCalendar.MarkerPositions.RightTop : 셀의 우측 상단에 위치



C# 사용법



메소드(함수) BackMonth, NextMonth

- BackMonth() : 현재 달 기준으로 지난 달 화면을 출력합니다.
- NextMonth() : 현재 달 기준으로 다음 달 화면을 출력합니다.

◉ void BackMonth()
◉ void NextMonth()

C# 사용법

```
// 현재 달 기준으로 지난 달
smartMonthCalendar1.BackMonth();
// 현재 달 기준으로 다음 달
smartMonthCalendar1.NextMonth();
```

VB 사용법

```
smartMonthCalendar1.BackMonth()
smartMonthCalendar1.NextMonth()
```

메소드(함수) MarkerInfoAdd

일정 관리를 하기 위해 사용될 Marker 관련 정보를 추가합니다.

◉ void MarkerInfoAdd(SmartMonthCalendar.MARKERDAYINFOS infoAdd)

◉ void MarkerInfoAdd(int iYear, int iMonth, int iDay, int iMarkerImageIndex)

[인자]

- SmartMonthCalendar.MARKERDAYINFOS infoAdd : MARKERDAYINFOS 객체
  - int iYear : 연도
  - int iMonth : 월
  - int iDay : 일자
  - int iImageIndex : Marker 이미지 Index
- int iYear : 연도
- int iMonth : 월
- int iDay : 일자
- int iMarkerImageIndex : Marker 이미지 Index

참고

달력에 이미지로 마킹을 원하는 경우 Marker 이미지 Index 값을 MarkerImages 속성에 추가된 순서로 설정해야 합니다. Index는 1부터 시작합니다.

[Marker 이미지 Index 값]

- -1 : Marker 정보가 없는 경우
- 0 : 기본 Marker 이미지 사용할(◤) 경우
- 1 : 추가된 첫번째 Marker 이미지
- n : 추가된 n번째 Marker 이미지(순번대로 증가)

C# 사용법

```
// 2021년 1월 5일에 Marker 추가. iMarkerImageIndex가 0인 경우 기본 마킹 추가
smartMonthCalendar1.MarkerInfoAdd(2021, 1, 5, 0);
smartMonthCalendar1.MarkerInfosUpdate();
```

VB 사용법

```
smartMonthCalendar1.MarkerInfoAdd(2021, 1, 5, 0)
smartMonthCalendar1.MarkerInfosUpdate()
```

메소드(함수) MarkerInfoFind

특정 연도-월-일자에 해당하는 Marker 이미지 Index 값(iDayMarkerIndex)을 찾아 리턴합니다.

◉ int MarkerInfoFind(int iYear, int iMonth, int iDay)

[인자]

- int iYear : 연도
- int iMonth : 월
- int iDay : 일자

[리턴값]

- int : 특정 연도-월-일자에 해당하는 Marker 이미지 Index 값(iDayMarkerIndex)을 int형으로 표현

[표] 날짜에 따른 Marker 이미지 Index 리턴값(iDayMarkerIndex)

날짜 선택	iYear	iMonth	iDay	리턴값(iDayMarkerIndex)
2021/01/05	2021	1	5	-1 (Marker 사용 안 함)
2021/01/06	2021	1	6	0 (Marker 기본 이미지 사용)
2021/01/18	2021	1	18	1 (Marker 첫번째 이미지 사용)

C# 사용법

```
int m_iYear1 = 2021, m_ iMonth1 = 1, m_iDay1 = 8;
// 상기 그림에서 1월 8일은 마킹이 없으므로 -1을 반환
smartLabel1.Text = smartMonthCalendar1.MarkerInfoFind(m_iYear1, m_iMonth1, m_iDay1).ToString();
```

VB 사용법

```
Dim m_iYear1 As Integer = 2021, m_iMonth1 As Integer = 1, m_iDay1 As Integer = 8
smartLabel1.Text = smartMonthCalendar1.MarkerInfoFind(m_iYear1, m_iMonth1,m_iDay1).ToString()
```

메소드(함수) MarkerInfoRemove

MarkerInfoAdd() 메서드에 의해 추가된 Marker의 정보(연도, 월, 일자)를 삭제합니다.

◉ bool MarkerInfoRemove(int iYear, int iMonth, int iDay)

[인자]

- int iYear : 연도
- int iMonth : 월
- int iDay : 일자

[리턴값]

- bool : 삭제 성공 유무
  - true : 삭제 성공
  - false : 삭제 실패 (마킹이 없는 경우)

C# 사용법

```
int m_iYear1 = 2021, m_ iMonth1 = 1, m_iDay1 = 2;
// 사용자가 지정한 특정 일자에 마킹된 마커 또는 마커 정보를 삭제
smartMonthCalendar1.MarkerInfoRemove(m_iYear1, m_iMonth1, m_iDay1);
smartMonthCalendar1.MarkerInfosUpdate();
```

VB 사용법

```
Dim m_iYear1 As Integer = 2021, m_iMonth1 As Integer = 1, m_iDay1 As Integer = 2
smartMonthCalendar1.MarkerInfoRemove(m_iYear1, m_iMonth1, m_iDay1)
```

```
smartMonthCalendar1.MarkerInfosUpdate()
```

메소드(함수) MarkerInfoRemoveAll

모든 Marker의 정보를 삭제합니다.

◉ void MarkerInfoRemoveAll()

C# 사용법

```
// 마킹된 마커 또는 마커 정보를 전부 삭제
smartMonthCalendar1.MarkerInfoRemoveAll(); smartMonthCalendar1.MarkerInfosUpdate();
```

VB 사용법

```
smartMonthCalendar1.MarkerInfoRemoveAll() : smartMonthCalendar1.MarkerInfosUpdate()
```

메소드(함수) MarkerInfosUpdate

Marker의 정보를 즉시 갱신하여 달력의 상태를 변경합니다. 즉, 화면을 갱신합니다.

◉ void MarkerInfosUpdate()

C# 사용법

```
// CASE-1. 마커 추가 시 즉시 변경 안 돼 화면 갱신 시 적용됨
smartMonthCalendar1.MarkerInfoAdd(2021,1,5,0);
// CASE-2. 설정한 날의 마커 제거 시 즉시 변경 안 돼 화면 갱신 시 적용됨
smartMonthCalendar1.MarkerInfoRemove(2021, 1, 5);
// CASE-3. 모든 마커 제거 시 즉시 변경 안 돼 화면 갱신 시 적용됨
smartMonthCalendar1.MarkerInfoRemoveAll();
// 마커 관련 정보를 즉시 업데이트하는 해당 코드를 사용함으로 각 CASE의 수정사항을 화면에 즉시 표시
smartMonthCalendar1.MarkerInfosUpdate();
```

VB 사용법

```
smartMonthCalendar1.MarkerInfoAdd(2021,1,5,0)
' smartMonthCalendar1.MarkerInfoRemove(m_iYear1, m_iMonth1, m_iDay1)
' smartMonthCalendar1.MarkerInfoRemoveAll()
smartMonthCalendar1.MarkerInfosUpdate()
```

이벤트 OnMonthChange

달력에서 월(이전달/다음달)을 변경하는 경우 발생하는 이벤트입니다.

[인자]

- int iYear : 사용자가 월을 변경할 때 해당 연도
- int iMonth : 사용자가 월을 변경할 때 변경된 월

C# 사용법

```
private void smartMonthCalendar1_OnMonthChange(int iYear, int iMonth)
{
  MessageBox.Show( " 현재 연도는 " + iYear + " 현재 월은 " + iMonth);
}
```

VB 사용법

```
Private Sub smartMonthCalendar1_OnMonthChange(ByVal iYear As System.Int32, ByVal iMonth As
```

```
System.Int32) Handles smartMonthCalendar1.OnMonthChange
  MessageBox.Show( " 현재 연도는 " + iYear + " 현재 월은 " + iMonth)
End Sub
```

이벤트 **OnSelectedDayChange**

달력에서 특정 일자를 선택하는 경우 발생하는 이벤트입니다.

[인자]

- int iYear : 사용자가 특정 일자를 선택할 때 해당하는 연도
- int iMonth : 사용자가 특정 일자를 선택할 때 해당하는 월
- int iDay : 사용자가 특정 일자를 선택할 때 해당하는 일자
- int iDayMarkerIndex : 사용자가 특정 일자를 선택할 때 해당하는 Marker 이미지 Index

C# 사용법

```
// 달력에서 특정 날짜(2021/01/04)를 선택하는 경우 발생하는 이벤트
private void smartMonthCalendar1_OnSelectedDayChange(int iYear, int iMonth, int iDay , int
iDayMarkerIndex)
{
  MessageBox.Show( " 2021년 1월 4일에서 1월 8일 선택하면 iYear는 2021, iMonth는 1, iDay는 8,
  iDayMarkerIndex는 -1입니다. " );
}
```

VB 사용법

```
Private Sub smartMonthCalendar1_OnSelectedDayChange(ByVal iYear As System.Int32, ByVal iMonth As
System.Int32, ByVal iDay As System.Int32) Handles smartMonthCalendar1.OnSelectedDayChange
  MessageBox.Show( " 2021년 1월 4일에서 1월 8일 선택하면 iYear는 2021, iMonth는 1, iDay는 8,
  iDayMarkerIndex는 -1입니다. " )
End Sub
```

4) SmartMonthCalendar 예제 사용하기

SmartMonthCalendar를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartMonthCalendar"

Part-IX

사용자 편의 컴포넌트

사용자 편의(Useful) 컴포넌트는 장치 응용 프로그램 개발 시 유용하게 사용할 수 있는
각종 기능을 지원하는 컴포넌트들을 구성하였습니다.

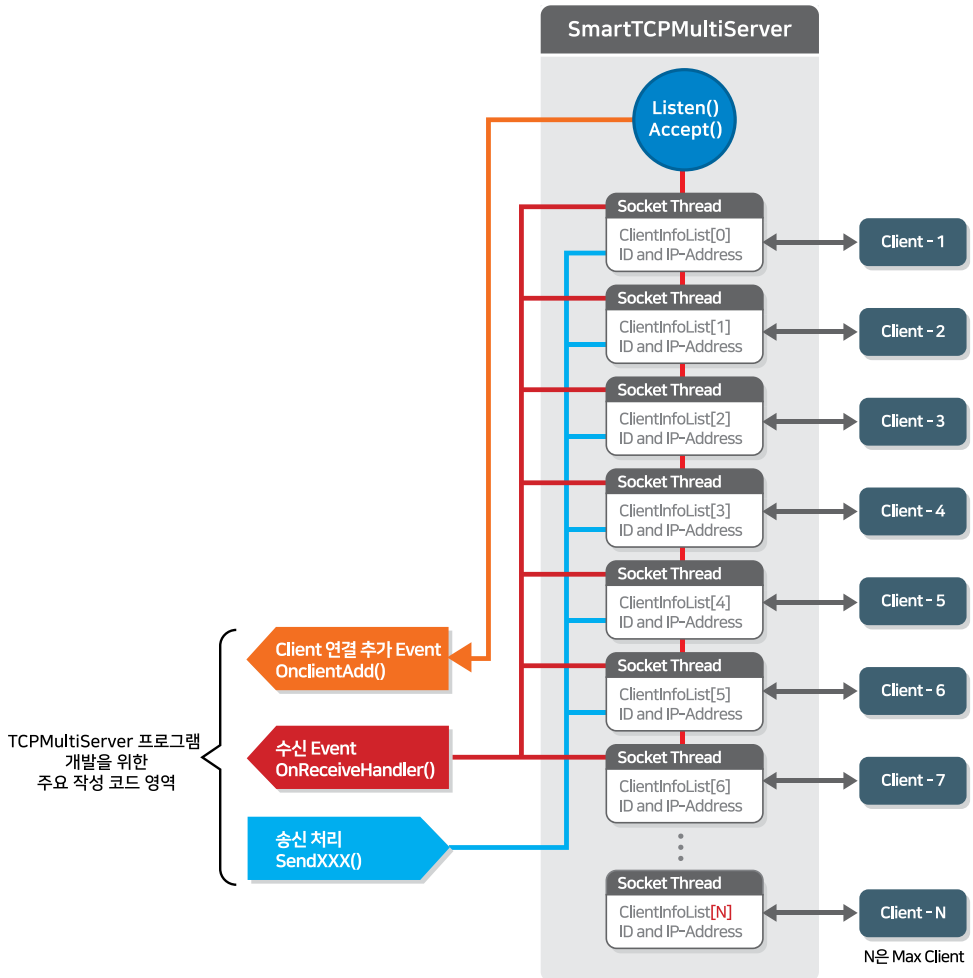
1. SmartTCPMultiServer
2. SmartTCPClient
3. SmartConfigs
4. SmartRemote
5. SmartTimer
6. SmartFile
7. SmartUpdate
8. SmartLock
9. SmartFileSetting
10. SmartThread
11. SmartFTP
12. SmartScreenSaver
13. SmartPlayer
14. SmartLaunch

Part-IX. 사용자 편의 컴포넌트

1. SmartTCPMultiServer

SmartTCPMultiServer는 다중 접속 TCP Socket 서버 프로그램을 쉽게 개발할 수 있도록 만들어진 컴포넌트입니다. 다음과 같은 특징을 가지고 있습니다.

- 다중 접속 서버 기능
- 접속 리스트 정보 제공 및 각각의 Socket 접근 및 제어
- 수신 데이터 이벤트 처리 방식
- 접속 Client 관리를 위한 ID 설정 및 IP-Address 연동 처리
- 동시 접속 제한 기능
- 다중 접속에 따른 각각의 스레드 처리
- ASCII, UNICODE, BYTE 등의 편리한 데이터 송/수신 처리
- 각종 편리한 Socket 관련 기능 지원
- 비동기 Socket 기능 지원



1) 프로그래밍 적용 가이드

SmartTCPMultiServer는 다중 접속 처리를 위하여 모든 송/수신 처리 시 Client의 접속에 따른 정보를 관리하며, 정보의 구성은 각각의 Client의 IP-Address와 ID를 매핑하는 작업부터 시작합니다. 관리되는 Client 접속 정보를 통하여 연결된 Client와 데이터를 송/수신 처리하게 되며, 이때 송/수신 대상을 구별하여 처리할 수 있도록 모든 송신 처리 함수에서 대상 IP-Address 또는 ID를 입력하여 전송할 수 있도록 하였습니다. 수신 처리 역시 수신 이벤트 정보에 Client의 IP-Address와 ID 정보가 함께 수신되도록 하였습니다. 송신 처리 방식의 경우 동기 처리 방식과 비동기 처리 방식이 있으며, 본 가이드는 동기 처리 방식을 기준으로 설명합니다.

권장 SmartTCPMultiServer의 권장 송/수신 처리 모델

송신 처리 : 동기와 비동기 처리를 모두 지원하지만 동기 방식으로 사용할 것을 권장합니다.
수신 처리 : 비동기 처리(이벤트 모델) 방식만 지원합니다.

STEP-1 TCP Server 관련 설정 및 시작하기

SmartTCPMultiServer의 동작 관련 속성을 설정합니다. MaxClient 속성으로 최대 연결 Client 개수를, MaxReceiveBufferSize 속성으로 최대 수신 데이터 크기를, Port 속성으로 TCP Port 번호를, ReceiveTimeOut 속성으로 응답 시간을, SetBlocking() 메소드로 송신 처리 방식을 설정합니다. 설정이 완료되면 Start() 메소드로 SmartTCPMultiServer를 시작하고 Client의 연결을 기다립니다.

※ 자세한 내용은 "MaxClient, MaxReceiveBufferSize, Port, ReceiveTimeOut 속성", "SetBlocking(), Start() 메소드"를 참고하시기 바랍니다.

```
smartTCPMultiServer1.MaxClient = 100;  // 최대 연결 Client 개수를 100개로 설정
smartTCPMultiServer1.MaxReceiveBufferSize = 1024;  // 한 번에 수신 가능한 데이터의 최대 크기 설정
smartTCPMultiServer1.Port = 9012;  // TCP Port 번호를 9012로 설정
smartTCPMultiServer1.ReceiveTimeout = 1000;  // Client 데이터 수신에 따른 응답 시간을 1000ms로 설정
smartTCPMultiServer1.SetBlocking(true);  // 데이터 수신 처리 방식을 동기 방식으로 설정
// 현재 SmartTCPMultiServer가 시작되지 않은 경우 시작
if (smartTCPMultiServer1.IsStart == false) {  smartTCPMultiServer1.Start();  }
```

STEP-2 Client 접속 요청에 따른 Event 처리 방법

Client의 접속 요청이 발생할 때 마다 OnClientAdd 이벤트가 발생합니다. 이때 addClientInfo 인자로 전달되는 IP-Address로 요청이 발생한 Client를 구분할 수 있으며, 추가로 Client의 IP-Address와 ID의 맵핑 처리도 함께하여 Client의 식별을 편리하게 할 수 있도록 처리합니다.

※ 자세한 내용은 "OnClientAdd 이벤트"를 참고하시기 바랍니다.

```
// Client가 SmartTCPMultiServer에 Connect될 때마다 발생하는 이벤트
private void smartTCPMultiServer1_OnClientAdd(SmartX.SmartTCPMultiServer.CLIENTSINFOS addClientInfo)
{
  // 특정 Client IP Address를 특정 ID로 매핑
  switch (addClientInfo.strIPAddress)
  {
    case "192.168.1.3":
      addClientInfo.strID = "Device-01";  // 192.168.1.3 IP를 Client ID : Device-01로 매핑함
      break;
    case "192.168.1.5":
      addClientInfo.strID = "Device-02";  // 192.168.1.5 IP를 Client ID : Device-02로 매핑함
      break;
  }
}
```

STEP-3 Client와 Server 간 송신 처리하기

ClientInfoList 속성으로 현재 연결된 Client의 정보를 얻을 수 있으며, 특정 Client로 데이터를 전송하기 위해서 Client의 IP-Address또는 ID를 입력하여 대상 Client로 송신 처리를 할 수 있습니다. 데이터 송신 메소드는 크게 Byte, ASCII, Unicode 3가지 방식으로 나눌수 있습니다.

※ 자세한 내용은 "ClientInfoList 속성", "SendByteByClientID(), SendByteByClientIP(), SendStringASCIIID(), SendStringASCIIIP(), SendStringUnicodeID(), SendStringUnicodeIP() 메소드"를 참고하시기 바랍니다.

```
// 연결된 Client가 있는 경우
if (smartTCPMultiServer1.ClientInfoList != null)
{
  byte[] sendData = new byte[5];  // 전송할 데이터
  sendData[0] = 0x33; sendData[1] = 0x45; sendData[2] = 0x71;
  sendData[3] = 0x52; sendData[4] = 0x02;
  // 대상 Client의 IP-Address로 전송하기                                    IP-Address 기준
  if (smartTCPMultiServer1.SendByteByClientIP("192.168.1.3", sendData) == false)
  {
    // 전송 실패 : "192.168.1.3"과 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
  }
  // 또는 대상 Client의 ID로 전송하기                                       ID 기준
  if (smartTCPMultiServer1. SendByteByClientID("Device-01", sendData) == false)
  {
    // 전송 실패 : "Device-01"과 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
  }
}
```

STEP-4 Client와 Server 간 수신 Event 처리하기

OnReceiveHandler 이벤트는 연결된 Client의 데이터 수신 시 발생되는 이벤트로 수신한 Client를 구별할 수 있도록 인자값을 사용해 수신 데이터와 Client의 IP-Address와 ID 정보를 함께 제공합니다.

※ 자세한 내용은 "OnReceiveHandler 이벤트"를 참고하시기 바랍니다.

```
// Client로 부터 데이터 수신 시 발생하는 이벤트
private void smartTCPMultiServer1_OnReceiveHandler(SmartX.SmartTCPMultiServer.
                                                   CHandleClinet.READINFO datas)
{
  // 수신 데이터 변환 처리
  string strReceiveData = SmartX.SmartTCPMultiServer.
                          ConvertAsciiByteToString(datas.receiveDatas);
  // 수신된 Client 구분을 IP-Address로 하는 경우                             IP-Address 기준
  switch (datas.strIPAddress)
  {
    case "192.168.1.3":
      byte[] sendData = new byte[3];
      sendData[0] = 0x77;  sendData[1] = 0x53;  sendData[2] = 0x02;
      // 연결된 IP 주소로 데이터 전송 방법
      if(smartTCPMultiServer1.SendByteByClientIP("192.168.1.3", sendData) == false)
      {
        // 전송 실패 : "192.168.1.3"과 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
      }
      break;
    case "192.168.1.5":
      strReceiveData = "수신 완료" + strReceiveData;
```

```
    // 연결된 IP 주소로 데이터 전송 방법
    if (smartTCPMultiServer1.SendStringASCIIIP("192.168.1.5", strReceiveData) == false)
    {
      // 전송 실패 : "192.168.1.5"과 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
    }
    break;
  }
  // 또는 수신된 Client 구분을 ID로 하는 경우
  switch (datas.strID)
  {
    case "Device-01":
      byte[] sendData = new byte[3];
      sendData[0] = 0x77;  sendData[1] = 0x53;  sendData[2] = 0x02;
      // 연결된 ID로 데이터 전송 방법
      if(smartTCPMultiServer1.SendByteByClientID("Device-01", sendData) == false)
      {
        // 전송 실패 : "Device-01"과 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
      }
      break;
    case "Device-02":
      strReceiveData = "수신 완료" + strReceiveData;
      // 연결된 ID로 데이터 전송 방법
      if (smartTCPMultiServer1.SendStringASCIIID("Device-02", strReceiveData) == false)
      {
        // 전송 실패 : "Device-02"과 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
      }
      break;
  }
}
```

ID 기준

STEP-5 현재 연결된 Client 정보 확인하기

ClientInfoList 속성으로 현재 서버와 연결된 Client의 IP-Address와 ID를 얻을 수 있으며, 연결 상태에 따라서 자동으로 갱신되며 비정상적으로 연결이 종료되는 경우 갱신 지연이 발생할 수 있습니다.

※ 자세한 내용은 "ClientInfoList 속성"을 참고하시기 바랍니다.

```
// 현재 연결된 Client가 존재하는 경우
if (smartTCPMultiServer1.ClientInfoList != null)
{
  string strIPAddress, strID;
  // ClientInfoList 속성을 통해 연결된 Client들의 정보를 확인
  // ClientInfoList 속성은 Client 연결 상태에 따라서 자동으로 속성값이 갱신됨
  for (int i = 0; i < smartTCPMultiServer1.ClientInfoList.Length; i++)
  {
    strIPAddress = "[IP]:" + smartTCPMultiServer1.ClientInfoList[i].strIPAddress;
    strID = "[ID]:" + smartTCPMultiServer1.ClientInfoList[i].strID;
    // 현재 연결된 Client 정보를 출력
    smartListBox1.AddItem(strIPAddress + strID);
  }
}
```

STEP-6 선택된 Client 연결 강제 종료 방법

현재 연결된 Client를 IP-Address 또는 ID로 구분하여 연결을 종료할 수 있습니다.

※ 자세한 내용은 "CloseClientIP(), CloseClientID() 메소드"를 참고하시기 바랍니다.

```
// 지정된 Client와 연결 해제 처리
smartTCPMultiServer1.CloseClientIP( " 192.168.1.3 " );
smartTCPMultiServer1.CloseClientID( " Device-02 " );
```

2) SmartTCPMultiServer 인터페이스 설명

SmartTCPMultiServer Component Interface		
속성		
ClientInfoList : SmartTCPMultiServer.CLIENTSINFOS[]	Clients : SmartTCPMultiServer.CHandleClinet[]	IsStart : bool
MaxClient : int	MaxReceiveBufferSize : int	Port : int
ReceiveTimeout : int		
메소드		
CloseClientID(string strID) : bool	CloseClientIP(string strIPAddress): bool	ConvertAsciiByteToString(byte[] byte Data): static string
ConvertUnicodeByteToString(byte[] byteData) : static string	ConvertUTF8ByteToString(byte[] byte Data) : static string	GetCheckSum16Gen(byte[] rawData) : static ushort (+3개 오버로드)
GetCheckSum8Gen(byte[] rawData) : static byte (+3개 오버로드)	GetCRC16Gen(byte[] rawData) : static ushort (+3개 오버로드)	GetCRC32Gen(byte[] rawData) : static uint (+3개 오버로드)
SendByteByClientID(string strID, byte[] sendData) : bool	SendByteByClientIP(string strIPAddress, byte[] sendData) : bool	SendStringASCIIID(string strID, string strSend) : bool
SendStringASCIIIP(string strIPAddress, string strSend) : bool	SendStringUnicodeID(string strID, string strSend) : bool	SendStringUnicodeIP(string strIPAddre ss, string strSend) : bool
SetBlocking(bool bBlocking) : void	SetLocalIPAddress(string strLocalIPAddr ess): void	SetMaxCnnectionReturnMessage(string strMaxconnectionMsg, SmartTCPMulti Server.RETURNMSGCODE eReturnMsg Code) : void
Start() : void	Stop() : void	
이벤트		
OnClientAdd : SmartTCPMultiServer. ClientAddHandler	OnReceiveHandler : SmartTCPMultiSer ver.ReceiveHandler	OnSent : SmartTCPMultiServer.CHandle Clinet.SendHandler

프로퍼티(속성) ClientInfoList

SmartTCPMultiServer와 연결된 Client 리스트의 ID, IP-Address 등의 정보를 얻습니다.

- SmartTCPMultiServer.CLIENTSINFOS[] : 연결된 Client 정보
  - SmartTCPMultiServer.CLIENTSINFOS.strID : OnClientAdd Event에서 Mapping된 문자열 ID
  - SmartTCPMultiServer.CLIENTSINFOS.strIPAddress : IP-Address

C# 사용법

```
// 리스트박스에 [i]번째의 Client를 추가. ID로 표시
smartListBox1.AddItem(smartTCPMultiServer1.ClientInfoList[i].strID);
// 리스트박스에 [i]번째의 Client를 추가. IP-Address로 표시
smartListBox1.AddItem(smartTCPMultiServer1.ClientInfoList[i].IPAddress);
```

VB 사용법

```
smartListBox1.AddItem(SmartTCPMultiServer1.ClientInfoList(i).strID)
smartListBox1.AddItem(smartTCPMultiServer1.ClientInfoList(i).IPAddress)
```

프로퍼티(속성) Clients

SmartTCPMultiServer에서 각각의 Client를 직접 접근할 수 있습니다.
- Clients.ID : 서버와 연결된 Client의 ID
- Clients.IPAddress : 서버와 연결된 Client의 IP-Address
- Clients.IsConnect : 서버에 연결된 Client가 현재 접속되었는지 확인
- Clients.ReceiveTimeout : 서버에서 Client에 응답할 수 있는 시간 (단위 : ms)
- Clients.TcpClient : 각각의 Client와 연결된 Socket Class를 제공 (MSDN 참고)

C# 사용법

```
// 리스트박스에 0번째의 Client를 추가, ID로 표시
smartListBox1.AddItem(smartTCPMultiServer1.Clients[0].ID);
```

VB 사용법

```
smartListBox1.AddItem(smartTCPMultiServer1.Clients(0).ID)
```

프로퍼티(속성) IsStart

SmartTCPMultiServer의 시작 여부를 확인합니다.

[리턴값]

• bool : SmartTCPMultiServer 시작 여부
  - true : 시작됨
  - false : 시작 안 됨

C# 사용법

```
if (smartTCPMultiServer1.IsStart == true)
{
  smartTCPMultiServer1.Stop();  // SmartTCPMultiServer가 시작된 경우 동작을 중지
}
```

VB 사용법

```
If smartTCPMultiServer1.IsStart = True Then
  smartTCPMultiServer1.Stop();
End If
```

프로퍼티(속성) MaxClient

SmartTCPMultiServer에 연결 가능한 최대 Client 개수를 설정합니다. 지정한 개수 이상의 Client가 접속을 시도하는 경우 SetMaxCnnectionReturnMessage() 메소드에서 설정한 문자열이 Client로 전송되며, 연결이 강제로 해제됩니다.

참고 "SetMaxCnnectionReturnMessage() 메소드" 내용을 참고하시기 바랍니다.

• int : 연결 가능한 최대 Client 수

C# 사용법

```
smartTCPMultiServer1.MaxClient = 5;  // 연결 가능한 최대 Client 수를 5개로 설정
```

VB 사용법

```
smartTCPMultiServer1.MaxClient = 5
```

프로퍼티(속성) MaxReceiveBufferSize

한번에 수신 가능한 수신 버퍼 사이즈를 설정합니다. 설정된 크기 이상의 데이터가 수신되는 경우, 수신된 데이터 크기

를 MaxReceiveBufferSize로 나눈 횟수만큼 OnReceiveHandler 이벤트가 발생합니다.
※ Ex) 수신 데이터 크기 : 1500Byte, MaxReceiveBufferSize = 1000 → OnReceiveHandler 이벤트 2회 발생
• int : 한 번에 수신 가능한 수신 버퍼 사이즈 (단위 : Byte)

C# 사용법

```
smartTCPMultiServer1.MaxReceiveBufferSize = 1000;  // 버퍼 사이즈를 1000Byte로 설정
```

VB 사용법

```
smartTCPMultiServer1.MaxReceiveBufferSize = 1000
```

 **프로퍼티(속성)** **Port**

TCP/IP 통신에 사용할 Port 번호를 설정합니다.

중요 Port 번호는 반드시 Server와 Client가 일치해야 합니다.

• int : TCP/IP Port 번호

C# 사용법

```
// Port 번호를 3377번으로 설정
smartTCPMultiServer1.Port = 3377;
```

VB 사용법

```
smartTCPMultiServer1.Port = 3377
```

 **프로퍼티(속성)** **ReceiveTimeout**

SmartTCPMultiServer에서 Client로부터 데이터 수신에 따른 응답 대기 시간을 설정합니다.

참고 ReceiveTimeout 속성값을 너무 크게 설정하는 경우 수신 데이터 처리 시간에 지연이 발생할 수 있습니다. 따라서 충분한 테스트를 거쳐 값을 설정하시기 바랍니다.

• int : 데이터 수신에 따른 응답 시간 (단위 : ms)

C# 사용법

```
// ReceiveTimeout을 3초로 설정
smartTCPMultiServer1.ReceiveTimeout = 3000;
```

VB 사용법

```
smartTCPMultiServer1.ReceiveTimeout = 3000
```

 **메소드(함수)** **Start, Stop**

• Start() : SmartTCPMultiServer를 시작합니다.
• Stop() : SmartTCPMultiServer를 중지합니다.

◉ void Start()
◉ void Stop()

C# 사용법

```
smartTCPMultiServer1.Start();  // smartTCPMultiServer를 시작
smartTCPMultiServer1.Stop();   // smartTCPMultiServer의 동작을 중지
```

VB 사용법

```
smartTCPMultiServer1.Start()
smartTCPMultiServer1.Stop()
```

메소드(함수) SetLocalIPAddress

IEC非Lite-Series 제품군에서 유선랜과 무선랜을 동시에 사용하는 경우, SmartTCPMultiServer의 통신 회선을 설정합니다.

◉ void SetLocalIPAddress(string strLocalIPAddress)

[인자]

• string strLocalIPAddress : 유선/무선 IP 중에 SmartTCPMultiServer로 사용할 IP-Address

C# 사용법

```
// 유선 IP 주소가 192.168.0.10이고 무선 IP 주소가 192.168.1.20일 때 사용자가
// smartTCPMultiServer1으로 유선 IP 주소를 사용하기 위해 인자값으로 192.168.0.10를 입력
smartTCPMultiServer1.SetLocalIPAddress( " 192.168.0.10 " );
```

VB 사용법

```
smartTCPMultiServer1.SetLocalIPAddress( " 192.168.0.10 " )
```

메소드(함수) SetMaxCnnectionReturnMessage

SmartTCPMultiServer에 MaxClient 속성으로 설정한 개수 이상의 Client가 접속을 시도하는 경우 해당 Client로 전송할 메시지를 설정합니다.

참고 "MaxClient 속성" 내용을 참고하시기 바랍니다.

◉ void SetMaxCnnectionReturnMessage(string strMaxconnectionMsg,
SmartTCPMultiServer.RETURNMSGCODE eReturnMsgCode)

[인자]

• string strMaxconnectionMsg : 초과 접속된 Client에게 전송할 메시지
• SmartTCPMultiServer.RETURNMSGCODE eReturnMsgCode : 메시지의 인코딩 방식
  - SmartTCPMultiServer.RETURNMSGCODE.ASCII : ASCII
  - SmartTCPMultiServer.RETURNMSGCODE.UNICODE : UNICODE

C# 사용법

```
// 초과 접속한 Client에게 전송할 메시지를 설정
smartTCPMultiServer1.SetMaxCnnectionReturnMessage( " Over Connection!!! " , SmartTCPMultiServer
                                                  .RETURNMSGCODE.UNICODE);
```

VB 사용법

```
smartTCPMultiServer1.SetMaxCnnectionReturnMessage( " Over Connection!!! " , SmartTCPMultiServer
                                                  .RETURNMSGCODE.UNICODE)
```

메소드(함수) SetBlocking

SmartTCPMultiServer의 데이터 송신 처리 방식(동기, 비동기)을 설정합니다. SmartTCPMultiServer에서 동기, 비동기 처리 방식은 각 Blocking, NoneBlocking 처리 방식을 의미합니다.

권장 SmartTCPMultiServer의 권장 송/수신 처리 모델

송신 처리 : 동기와 비동기 처리를 모두 지원하지만, 동기 방식으로 사용할 것을 권장합니다.
수신 처리 : 비동기 처리(이벤트 모델) 방식만 지원합니다.

참고 SmartTCPMultiServer에서 데이터 송신 처리 방식

SmartTCPMultiServer는 수신 처리의 경우 비동기 처리(이벤트 모델)만을 지원하며, 송신 처리 시 동기, 비동기 처리를 지원합니다. 즉, 송신의 결과를 메소드의 리턴값에서 확인하는 것이 아닌 이벤트에서 확인해야 합니다.

[표] 관련 메소드 및 이벤트

관련 메소드	관련 이벤트
SendByteByClientID(), SendByteByClientIP(), SendStringASCIIID(), SendStringASCIIIP(), SendStringUnicodeID(), SendStringUnicodeIP()	OnSent

※ 위 표의 메소드와 이벤트를 참고하시기 바랍니다.

◉ `void SetBlocking(bool bBlocking)`

[인자]

- `bool bBlocking` : 데이터 송신 처리 방식
  - `true` : 동기-Blocking
  - `false` : 비동기-NoneBlocking

C# 사용법

```
// 송신 처리 방식을 동기 처리 방식으로 설정
smartTCPMultiServer1.SetBlocking(true);
```

VB 사용법

```
smartTCPMultiServer1.SetBlocking(True)
```

메소드(함수) CloseClientID, CloseClientIP

SmartTCPMultiServer와 연결된 Client의 연결을 해제합니다.

- `CloseClientID()` : 연결된 Client를 ID로 구분하여 연결을 해제합니다.
- `CloseClientIP()` : 연결된 Client를 IP-Address로 구분하여 연결을 해제합니다.

◉ `bool CloseClientID(string strID)`

◉ `bool CloseClientIP(string strIPAddress)`

[인자]

- `string strID` : 접속을 해제할 Client의 ID
- `string strIPAddress` : 접속을 해제할 Client의 IP-Address

[리턴값]

- `bool` : 해제 성공 여부
  - `true` : 성공
  - `false` : 실패

C# 사용법

```
// ID가 Device-01인 Client의 연결을 종료
smartTCPMultiServer1.CloseClientID( " Device-01 " );
// IP-Address가 192.168.1.10인 Client의 연결을 종료
smartTCPMultiServer1.CloseClientIP( " 192.168.1.10 " );
```

VB 사용법

```
smartTCPMultiServer1.CloseClientID( " Device-01 " )
smartTCPMultiServer1.CloseClientIP( " 192.168.1.10 " )
```

메소드(함수) SendByteByClientID, SendByteByClientIP

SmartTCPMultiServer와 연결된 Client에 바이트 배열 데이터를 전송합니다.

- `SendByteByClientID()` : 연결된 Client를 ID로 구분하여 바이트 배열 데이터를 전송합니다.
- `SendByteByClientIP()` : 연결된 Client를 IP-Address로 구분하여 바이트 배열 데이터를 전송합니다.

참고 SetBlocking() 메소드의 인자값을 false로 하여 데이터 송신 처리를 비동기 방식으로 하는 경우, 데이터 송신 메소드 호출 즉시 false를 리턴하며, 실제 데이터 송신 처리 결과는 OnSent 이벤트가 발생하여 확인할 수 있습니다.
"SetBlocking() 메소드", "OnSent 이벤트" 내용을 참고하시기 바랍니다.

◉ bool SendByteByClientID(string strID, byte[] sendData)
◉ bool SendByteByClientIP(string strIPAddress, byte[] sendData)

[인자]
- string strID : 데이터를 수신할 Client의 ID
- string strIPAddress : 데이터를 수신할 Client의 IP-Address
- byte[] sendData : Client로 전송할 바이트 배열형 데이터

[리턴값]
- bool : 데이터 전송 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// sendData라는 이름의 바이트 배열을 선언하고 배열 초기화
byte[] sendData = new byte[3];
sendData[0] = 0x33;  sendData[1] = 0x45;  sendData[2] = 0x71;
// ID가 Device-01인 Client에게 바이트 배열형 데이터 sendData를 전송
smartTCPMultiServer1.SendByteByClientID( " Device-01 " , sendData);
// IP-Address가 192.168.1.100인 Client에게 바이트 배열형 데이터 sendData를 전송
smartTCPMultiServer1.SendByteByClientIP( " 192.168.1.100 " , sendData);
```

VB 사용법

```
Dim sendData As Byte() = new Byte(2) {}
sendData(0) = &H33  :  sendData(1) = &H45  :  sendData(2) = &H71
smartTCPMultiServer1.SendByteByClientID( " Device-01 " , sendData)
smartTCPMultiServer1.SendByteByClientIP( " 192.168.1.100 " , sendData)
```

메소드(함수) SendStringASCIIID, SendStringASCIIIP

SmartTCPMultiServer와 연결된 Client에 String 데이터를 ASCII 문자로 전송합니다.
- SendStringASCIIID() : 연결된 Client를 ID로 구분하여 ASCII 문자를 전송합니다.
- SendStringASCIIIP() : 연결된 Client를 IP-Address로 구분하여 ASCII 문자를 전송합니다.

참고 SetBlocking() 메소드의 인자값을 false로 하여 데이터 송신 처리를 비동기 방식으로 하는 경우, 데이터 송신 메소드 호출 즉시 false를 리턴하며, 실제 데이터 송신 처리 결과는 OnSent 이벤트가 발생하여 확인할 수 있습니다.
"SetBlocking() 메소드", "OnSent 이벤트" 내용을 참고하시기 바랍니다.

◉ bool SendStringASCIIID(string strID, string strSend)
◉ bool SendStringASCIIIP(string strIPAddress, string strSend)

[인자]
- string strID : 데이터를 수신할 Client의 ID
- string strIPAddress : 데이터를 수신할 Client의 IP-Address
- string strSend : Client로 전송할 string형의 데이터

[리턴값]
- bool : 데이터 전송 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// sendStrData라는 이름의 String 변수를 선언하고 변수값을 초기화
string sendStrData = " HNS, Hardware And Software Total Solution! ";
// ClientID가 Device-01인 Client에게 string형의 strSend를 ASCII 문자 형태로 전송
smartTCPMultiServer1.SendStringASCIIID( " Device-01 ", sendStrData);
// ClientIP가 192.168.1.100인 Client에게 string형의 strSend를 ASCII 문자 형태로 전송
smartTCPMultiServer1.SendStringASCIIIP( " 192.168.1.100 ", sendStrData);
```

VB 사용법

```
Dim sendStrData As String = " HNS, Hardware And Software Total Solution! "
smartTCPMultiServer1.SendStringASCIIID( " Device-01 ", sendStrData)
smartTCPMultiServer1.SendStringASCIIIP( " 192.168.1.100 ", sendStrData)
```

메소드(함수) SendStringUnicodeID, SendStringUnicodeIP

SmartTCPMultiServer와 연결된 Client에 String 데이터를 Unicode 문자로 전송합니다.

- SendStringUnicodeID() : 연결된 Client를 ID로 구분하여 Unicode 문자를 전송합니다.
- SendStringUnicodeIP() : 연결된 Client를 IP-Address로 구분하여 Unicode 문자를 전송합니다.

참고 SetBlocking() 메소드의 인자값을 false로 하여 데이터 송신 처리를 비동기 방식으로 하는 경우, 데이터 송신 메소드 호출 즉시 false를 리턴하며, 실제 데이터 송신 처리 결과는 OnSent 이벤트가 발생하여 확인할 수 있습니다.
"SetBlocking() 메소드", "OnSent 이벤트" 내용을 참고하시기 바랍니다.

◉ bool SendStringUnicodeID(string strID, string strSend)
◉ bool SendStringUnicodeIP(string strIPAddress, string strSend)

[인자]

- string strID : 데이터를 수신할 Client의 ID
- string strIPAddress : 데이터를 수신할 Client의 IP-Address
- string strSend : Client로 전송할 string형의 데이터

[리턴값]

- bool : 데이터 전송 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// sendStrData라는 이름의 String 변수를 선언하고 변수값을 초기화
string sendStrData = " HNS, Hardware And Software Total Solution! ";
// ClientID가 Device-01인 Client에게 string형의 strSend를 Unicode 문자 형태로 전송
smartTCPMultiServer1.SendStringUnicodeID( " Device-01 ", sendStrData);
// ClientIP가 192.168.1.100인 Client에게 string형의 strSend를 Unicode 문자 형태로 전송
smartTCPMultiServer1.SendStringUnicodeIP( " 192.168.1.100 ", sendStrData);
```

VB 사용법

```
Dim sendStrData As String = " HNS, Hardware And Software Total Solution! "
smartTCPMultiServer1.SendStringUnicodeID( " Device-01 ", sendStrData);
smartTCPMultiServer1.SendStringUnicodeIP( " 192.168.1.100 ", sendStrData);
```

정적 메소드(함수) ConvertAsciiByteToString, ConvertUnicodeByteToString, ConvertUTF8ByteToString

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

바이트 배열 데이터를 ASCII, Unicode, UTF-8 문자로 변환합니다.

- ConvertAsciiByteToString() : 바이트 배열 데이터를 ASCII 문자로 변환합니다.
- ConvertUnicodeByteToString() : 바이트 배열 데이터를 Unicode 문자로 변환합니다.
- ConvertUTF8ByteToString() : 바이트 배열 데이터를 UTF-8 문자로 변환합니다.

참고 Client로부터 수신되는 바이트 배열 형태의 데이터를 편리하게 변환할 수 있습니다.

◉ static string ConvertAsciiByteToString(byte[] byteData)
◉ static string ConvertUnicodeByteToString(byte[] byteData)
◉ static string ConvertUTF8ByteToString(byte[] byteData)

[인자]

- byte[] byteData : 바이트 배열 데이터

[리턴값]

- static string : 변환된 문자열

C# 사용법

```
// Client로부터 데이터 수신 시 발생하는 이벤트
private void smartTCPMultiServer1_OnReceiveHandler(SmartTCPMultiServer.CHandleClinet.READINFO
datas)
{
  string strRecData = null;
  // 수신된 바이트 배열 데이터를 ASCII 문자열로 변환함
  strRecData = SmartTCPMultiServer.ConvertAsciiByteToString(datas.receiveDatas);
  // 수신된 바이트 배열 데이터를 UniCode 문자열로 변환함
  strRecData = SmartTCPMultiServer.ConvertUnicodeByteToString(datas.receiveDatas);
  // 수신된 바이트 배열 데이터를 UTF-8 문자열로 변환함
  strRecData = SmartTCPMultiServer.ConvertUTF8ByteToString(datas.receiveDatas);
}
```

VB 사용법

```
Private Sub smartTCPMultiServer1_OnReceiveHandler(ByVal datas As SmartTCPMultiServer.
CHandleClinet.READINFO) Handles smartTCPMultiServer1.OnReceiveHandler
  Dim strRecData As String = Nothing
  strRecData = SmartTCPMultiServer.ConvertAsciiByteToString(datas.receiveDatas)
  strRecData = SmartTCPMultiServer.ConvertUnicodeByteToString(datas.receiveDatas)
  strRecData = SmartTCPMultiServer.ConvertUTF8ByteToString(datas.receiveDatas)
End Sub
```

정적 메소드(함수) GetCheckSum8Gen, GetCheckSum16Gen, GetCRC16Gen, GetCRC32Gen

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

Byte[] 데이터에 해당하는 ErrorCheck Code를 계산합니다.

- GetCheckSum8Gen() : 바이트 배열 데이터의 CheckSum8 값을 얻습니다.
- GetCheckSum16Gen() : 바이트 배열 데이터의 CheckSum16 값을 얻습니다.
- GetCRC16Gen() : 바이트 배열 데이터의 CRC16 값을 얻습니다.
- GetCRC32Gen() : 바이트 배열 데이터의 CRC32 값을 얻습니다.

◉ static byte GetCheckSum8Gen(byte[] rawData)
◉ static byte[] GetCheckSum8Gen(byte[] rawData, SmartTCPMultiServer.CODETYPES eCode)
◉ static byte GetCheckSum8Gen(byte[] rawData, int iStart, int iLength)
◉ static byte[] GetCheckSum8Gen(byte[] rawData, int iStart, int iLength,
SmartTCPMultiServer.CODETYPES eCode)
◉ static ushort GetCheckSum16Gen(byte[] rawData)

⦿ static byte[] GetCheckSum16Gen(byte[] rawData, SmartTCPMultiServer.CODETYPES eCode)
⦿ static ushort GetCheckSum16Gen(byte[] rawData, int iStart, int iLength)
⦿ static byte[] GetCheckSum16Gen(byte[] rawData, int iStart, int iLength,
SmartTCPMultiServer.CODETYPES eCode)
⦿ static ushort GetCRC16Gen(byte[] rawData)
⦿ static byte[] GetCRC16Gen(byte[] rawData, SmartTCPMultiServer.CODETYPES eCode)
⦿ static ushort GetCRC16Gen(byte[] rawData, int iStart, int iLength)
⦿ static byte[] GetCRC16Gen(byte[] rawData, int iStart, int iLength,
SmartTCPMultiServer.CODETYPES eCode)
⦿ static uint GetCRC32Gen(byte[] rawData)
⦿ static byte[] GetCRC32Gen(byte[] rawData, SmartTCPMultiServer.CODETYPES eCode)
⦿ static uint GetCRC32Gen(byte[] rawData, int iStart, int iLength)
⦿ static byte[] GetCRC32Gen(byte[] rawData, int iStart, int iLength,
SmartTCPMultiServer.CODETYPES eCode)

[인자]
• byte[] rawData : 바이트 배열 형식의 원본 데이터
• int iStart : 데이터의 시작 지점
• int iLength : 데이터의 길이
• SmartTCPMultiServer.CODETYPES eCode : 데이터의 타입이 ASCII 또는 Unicode인지 설정

[리턴값]
– GetCheckSum8Gen() 메소드 리턴값
• byte : 데이터를 CheckSum 8Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터
– GetCheckSum16Gen() 메소드 리턴값
• ushort : 데이터를 CheckSum 16Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터
– GetCRC16Gen() 메소드 리턴값
• ushort : 데이터를 CRC 16Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터
– GetCRC32Gen() 메소드 리턴값
• uint : 데이터를 CRC 32Bit로 계산한 결과값
• byte[] : eCode 인자값에 따른 ASCII 또는 Unicode 데이터

[표] 데이터에 따른 ErrorCheckCode 계산 결과

testByte 배열	testByte[0]	testByte[1]	testByte[2]	testByte[3]	testByte[4]
데이터 (10진수)	65	66	67	68	69

ErrorCheck	Binary		ASCII	Unicode
	10진수	16진수	Chr (Binary 코드의 16진수 각 문자)	Chr (Binary 코드의 16진수 각 문자)
CheckSum8	79	0x4F	34-46	34-00-46-00
CheckSum16	335	0x14F	30-31-34-46	30-00-31-00-34-00-46-00
CRC16	3920	0xF50	30-46-35-30	30-00-46-00-35-00-30-00
CRC32	1926437589	0x72D31AD5	37-32-44-33-31-41-44-35	37-00-32-00-44-00-33-00-31-00-41-00-44-00-35-00

C# 사용법

```
byte[] testByte = new byte[5];
testByte[0] = 65; testByte[1] = 66; testByte[2] = 67; testByte[3] = 68; testByte[4] = 69;
// 실행 결과 : 10진수(Binary 코드)
ushort result = SmartTCPMultiServer.GetCRC16Gen(testByte);
```

VB 사용법

```
Dim testByte(4) As Byte = New Byte() {}
testByte(0) = 65 : testByte(1) = 66 : testByte(2) = 67 : testByte(3) = 68 : testByte(4) = 69
Dim result As UShort = SmartTCPMultiServer.GetCRC16Gen(testByte)
```

이벤트 OnClientAdd

SmartTCPMultiServer에 Client가 연결(Connect)될 때마다 발생하는 이벤트입니다. 인자값으로 연결된 Client의 IP-Address를 확인할 수 있으며, ID를 맵핑할 수 있습니다.

◉ OnClientAdd(SmartTCPMultiServer.CLIENTSINFOS addClientInfo)

[인자]

- SmartTCPMultiServer.CLIENTSINFOS addClientInfo : Client 정보
  - string addClientInfo.strID : IP-Address와 맵핑할 특정 ID
  - string addClientInfo.strIPAddress : IP-Address (변경 불가)

C# 사용법

```
// Client가 SmartTcpMultiServer에 Connect될 때마다 발생하는 이벤트
private void smartTCPMultiServer1_OnClientAdd(SmartTCPMultiServer.CLIENTSINFOS addClientInfo)
{
  // 특정 Client IP Address를 특정 ID로 맵핑
  switch (addClientInfo.strIPAddress)
  {
    // 현재 192.168.1.5 Client가 연결된 상태인 경우
    // 192.168.1.5번 Client의 ID를 Device-01 PC로 맵핑함
    case "192.168.1.5":
      addClientInfo.strID = "Device-01";
      break;
    case "192.168.1.6":
      addClientInfo.strID = "Device-02";
      break;
  }
}
```

VB 사용법

```
Private Sub smartTCPMultiServer1_OnClientAdd(ByVal addClientInfo As SmartTCPMultiServer.
CLIENTSINFOS) Handles smartTCPMultiServer1.OnClientAdd
  Select Case addClientInfo.strIPAddress
    Case "192.168.1.5"
      addClientInfo.strID = "Device-01"
    Case "192.168.1.6"
      addClientInfo.strID = "Device-02"
  End Select
End Sub
```

이벤트 OnReceiveHandler

SmartMultiServer가 Client로부터 데이터 수신 시 발생하는 이벤트입니다. 인자값에서 수신받은 데이터와, 송신한 Client의 ID 또는 IP-Address를 확인할 수 있습니다.

◉ OnReceiveHandler(SmartTCPMultiServer.CHandleClinet.READINFO datas)

[인자]

- SmartTCPMultiServer.CHandleClinet.READINFO datas : 수신 정보

- byte[] datas.receiveDatas : 수신 데이터(바이트 배열)
- string datas.strID : 송신한 Client의 ID
- string datas.strIPAddress : 송신한 Client의 IP-Address

C# 사용법

```
private void smartTCPMultiServer1_OnReceiveHandler(SmartTCPMultiServer.CHandleClinet.READINFO
datas)
{
  string strRecvData;
  // 송신한 Client의 ID를 확인
  if(datas.strID == " Device-01 " )
  {
    // 수신된 바이트 배열 데이터를 Unicode 문자로 변환함
    strRecData = SmartTCPMultiServer.ConvertByteToUnicode(datas.receiveDatas);
  }

  // 송신한 Client의 IP-Address를 확인
  if(datas.strIPAddress == " 192.168.1.6 " )
  {
    // 수신된 바이트 배열 데이터를 ASCII 문자로 변환함
    strRecData = SmartTCPMultiServer.ConvertByteToASCII(datas.receiveDatas);
  }
  lblRecv.Text = strRecData;
}
```

VB 사용법

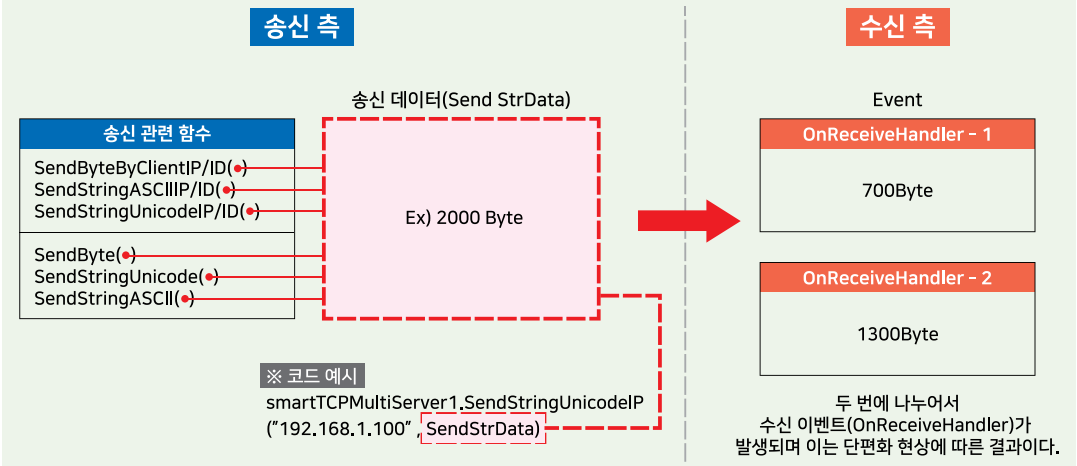
```
Private Sub smartTCPMultiServer1_OnReceiveHandler(ByVal datas As SmartTCPMultiServer.CHandle
                                                  Clinet.READINFO)
  Dim strRecData As String
  If datas.strID = " Device-01 " Then
    strRecData = SmartTCPMultiServer.ConvertByteToUnicode(datas.receiveDatas)
  End If

  If datas.strID = " 192.168.1.6 " Then
    strRecData = SmartTCPMultiServer.ConvertByteToUnicode(datas.receiveDatas)
  End If
  lblRecv.Text = strRecData
End Sub
```

참조 대량의 데이터 연속 송/수신 시 OnReceiveHandler 이벤트에서 데이터 단편화(Fragmentation) 발생 원인과 해결 방법

[발생 원인]

TCP/IP 통신에서 대량의 데이터를 연속으로 송/수신하는 경우 수신 측 이벤트(OnReceiveHandler)에서 데이터의 단편화(Fragmentation)현상이 발생합니다.

아래 그림에서 송신 측에서 보낸 데이터(2000Byte)가 수신 측의 수신 이벤트(OnReceiveHandler)가 두 번으로 나뉘어서 발생하며 700Byte와 1300Byte로 데이터 단편화가 발생합니다.



[해결 방법]

위의 문제를 해결하기 위해서는 수신된 데이터의 패킷을 Frame화 할 수 있는 별도의 처리가 필요합니다.

※ "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 73. [C#, VB.NET] TCP/IP 통신에서 송/수신 시 수신 측 이벤트에서 데이터 단편화(Fragmentation)발생 원인과 해결 방법" 내용을 참조하시기 바랍니다.

이벤트 OnSent

SetBlockint() 메소드의 인자값을 false로 설정하여 데이터 송신을 비동기 방식으로 처리하는 경우, SmartTCP MultiServer에서 Client로 데이터 전송이 완료되면 발생하는 이벤트입니다.

참고 "SetBlocking() 메소드" 내용을 참고하시기 바랍니다.

◉ OnSent(SmartTCPMultiServer.CLIENTSINFOS addClientInfo, int iBytesSent)

[인자]

- SmartTCPMultiServer.CLIENTSINFOS addClientInfo : Client 정보
  - string addClientInfo.strID : IP-Address와 맵핑할 특정 ID
  - string addClientInfo.strIPAddress : IP-Address (변경 불가)
- int iBytesSent : Client로 전송한 데이터의 길이 (단위 : Byte)

C# 사용법

```
// NONEBlocking 경우에서 서버 → Client로 데이터 전송 시 발생하는 이벤트
private void smartTCPMultiServer1_OnSent(SmartTCPMultiServer.CLIENTSINFOS ClientInfo, int
iBytesSent)
{
  // Client의 IP로 확인
  switch (ClientInfo.strIPAddress)
  {
    case "192.168.1.5":
      if (iBytesSent == m_strSendData.Length)
      {
        // 데이터 전송 성공 처리
      }
      else
      {
        // 데이터 전송 실패 처리
      }
```

```
        break;
        // 중략
    }
}
```

VB 사용법

```
Private Sub smartTCPMultiServer1_OnSent(ByVal ClientInfo As SmartTCPMultiServer.CLIENTSINFOS,
                                        ByVal iBytesSent As Integer)
  Select Case ClientInfo.strIPAddress
    Case " 192.168.1.5 "
      If iBytesSent = m_strSendData.Length Then
        ' 데이터 전송 성공 처리
      Else
        ' 데이터 전송 실패 처리
      End If
  End Select
End Sub
```

3) SmartTCPMultiServer 예제 사용하기

SmartTCPMultiServer를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

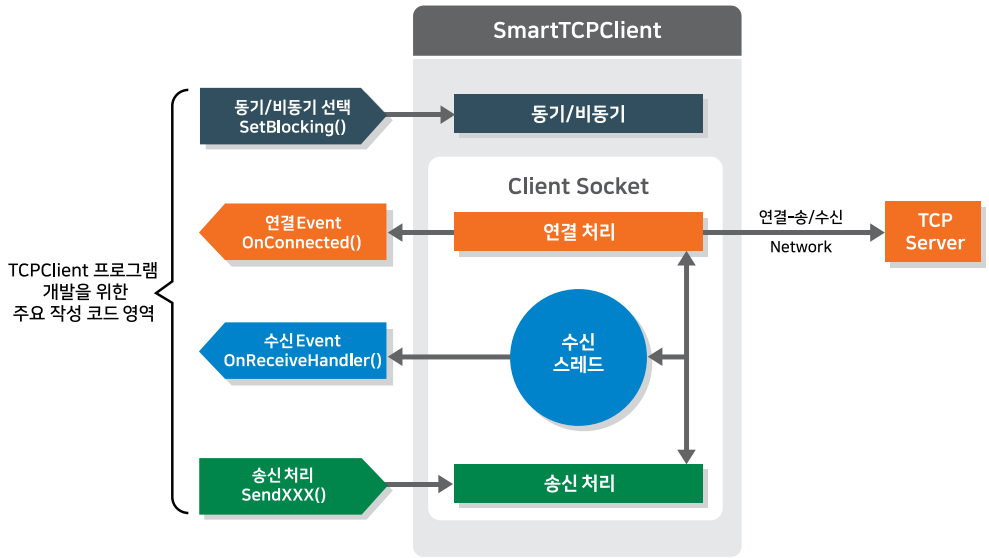
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartTCPMultiServer"

2. SmartTCPClient

SmartTCPClient는 TCP Client Socket 구현을 간편하게 처리할 수 있도록 개발되었으며, 이벤트 방식의 수신 처리로 편리하게 적용하실 수 있도록 만들어진 컴포넌트입니다.

- 동기/비동기 처리 지원
  - 비동기 방식은 적용하기 편리한 이벤트 방식으로 지원
- 수신 데이터 감지 스레드 처리
  - 수신 처리 방식은 이벤트 처리 방식을 권장
- 동기/비동기 혼용하여 사용 가능
  - 서버 접속(Connect) 기능만 비동기로 처리하여 접속 지연에 따른 문제점 해결
- ASCII, Unicode, Byte 등의 편리한 데이터 송/수신 처리
- 각종 편리한 Socket 관련 기능 지원



참조 TCP/IP Network Client 측 통신 프로그램 작성 시 발생될 수 있는 여러 가지 예외 처리 및 프로토콜에 의한 송/수신 시 필요한 처리를 별도로 작성할 수 있도록 만들어진 Wrapping Class를 제공하고 있습니다.
자세한 내용은 "홈페이지 → 자료실 → TechNote → 81. [C#] SmartTCPClient-Wrapping Class를 이용한 TCPClient 프로그램 구현 방법"을 참조하시기 바랍니다.

1) 프로그래밍 적용 가이드

SmartTCPClient는 연결 및 송신 처리 방식의 경우 동기 처리 방식과 비동기 처리 방식이 있으며, 본 가이드는 아래 권장 방식의 처리 모델에 따른 방식으로 설명합니다.

권장 SmartTCPClient의 권장 연결, 송/수신 처리 모델

연결 처리 : 동기와 비동기 처리를 모두 지원하지만, 비동기 방식으로 사용할 것을 권장합니다.
송신 처리 : 동기와 비동기 처리를 모두 지원하지만, 동기 방식으로 사용할 것을 권장합니다.
수신 처리 : 비동기 처리(이벤트 모델) 방식만 지원합니다.

STEP-1 TCP Client 관련 설정하기

SmartTCPClient의 동작 관련 속성을 설정합니다. ServerIPAddress 속성으로 접속할 서버의 IP-Address, MaxReceiveBufferSize 속성으로 최대 수신 데이터 크기, Port 속성으로 TCP Port 번호, ReceiveTimeout 속성으로 응답 시간을 설정합니다.

※ 자세한 내용은 "ServerIPAddress, MaxReceiveBufferSize, Port, ReceiveTimeout 속성"을 참고하시기 바랍니다.

```
smartTCPClient1.ServerIPAddress = "192.168.0.5"; // 연결할 서버의 IP-Address를 설정
smartTCPClient1.MaxReceiveBufferSize = 1024; // 한 번에 수신 가능한 데이터의 최대 크기 설정
smartTCPClient1.Port = 9012; // TCP Port 번호를 9012로 설정
smartTCPClient1.ReceiveTimeout = 1000; // 서버 데이터 수신에 따른 응답 시간을 1000ms로 설정
```

STEP-2 Server로 연결 요청하기

SmartTCPClient에서 연결 처리 방식은 동기-Blocking 방식과, 비동기-NoneBlocking 방식을 지원하며, 연결 요청 결과가 지연되는 경우 코드가 멈추는(Block) 현상을 방지하기 위해 비동기 방식을 사용하기를 권장합니다.

[CASE-1] 비동기 방식(이벤트 처리 방식)으로 연결 요청하기

연결 처리를 비동기 방식으로 하기 위해 SetBlocking() 메소드의 인자값을 false로 하여 호출합니다. Connect() 메소드로 Server와의 연결을 요청합니다. 연결 요청이 완료되면 OnConnected 이벤트가 발생하며, 인자값을 사용해 연결 결과를 확인할 수 있습니다.

※ 자세한 내용은 "SetBlocking(), Connect() 메소드", "OnConnected 이벤트"를 참고하시기 바랍니다.

```
private void Connect_NoneBlocking()
{
  // 데이터 송신 처리 방식을 비동기 방식으로 설정
  smartTCPClient1.SetBlocking(false);
  // 현재 SmartTCPClient가 서버와 연결되지 않은 경우
  if (smartTCPClient1.IsConnect == false)
  {
    // 서버와의 연결을 시도
    smartTCPClient1.Connect();
  }
}

// 연결 처리 방식이 비동기 방식인 경우 연결 요청이 완료되면 발생하는 이벤트
private void smartTCPClient1_OnConnected(bool bConnected)
{
  // 인자값으로 실제 연결 결과를 확인
  if (bConnected == true)
  {
    // 서버와의 연결이 성공함
  }
  else
  {
    // 서버와의 연결이 실패함
    // 연결 재시도 또는 실패에 따른 코드를 작성
  }
}
```

[CASE-2] 동기 방식으로 연결 요청하기

연결 처리를 동기 방식으로 하기 위해 SetBlocking() 메소드의 인자값을 true로 하여 호출합니다. Connect() 메소드로 Server와의 연결을 요청합니다. Connect() 메소드 호출 시 인자값으로 Connect Timeout 값을 설정하여, 연결 문제에 따른 프로그램의 Blocking 현상을 방지할 수 있습니다. 또한, 연결 요청 결과는 리턴값으로 확인할 수 있습니다.

※ 자세한 내용은 "SetBlocking(), Connect() 메소드"를 참고하시기 바랍니다.

```
private void Connect_Blocking()
{
  // 데이터 송신 처리 방식을 동기 방식으로 설정
  smartTCPClient1.SetBlocking(true);
  // 현재 SmartTCPClient가 서버와 연결되지 않은 경우
  if (smartTCPClient1.IsConnect == false)
  {
    // 서버와의 연결을 시도합니다. 5초간 서버의 응답을 대기합니다.
    if (smartTCPClient1.Connect(5000) == true)
    {
      // 서버와의 연결이 성공함
    }
    else
    {
      // 서버와의 연결이 실패함
      // 연결 재시도 또는 실패에 따른 코드를 작성
    }
  }
}
```

STEP-3 Client와 Server 간 송신 처리하기

서버로 데이터를 전송하기 위한 메소드는 데이터 타입에 따라 Byte, ASCII, Unicode 3가지로 나눌 수 있으며, 각 메소드에서 false를 리턴하여 데이터 전송에 실패한 경우, 서버와 연결이 끊어졌거나 네트워크의 불안정한 상태로 볼 수 있습니다. 이럴 경우 Close() 메소드를 호출해 서버와의 연결을 해제 후 재연결을 요청해야합니다. 또한, 데이터 전송 처리 방식은 동기 방식으로 설정하시길 권장합니다.

※ 자세한 내용은 "SendByte(), SendStringASCII(), SendStringUnicode(), Close() 메소드"를 참고하시기 바랍니다.

```
private void SendtoServer()
{
  // 서버와 연결에 성공한 경우
  if (smartTCPClient1.IsConnect == true)
  {
    // 데이터 전송 방식을 동기로 설정
    smartTCPClient1.SetBlocking(true);
    // 전송할 데이터
    byte[] sendData = new byte[5];
    sendData[0] = 0x33;
    sendData[1] = 0x45;
    sendData[2] = 0x71;
    sendData[3] = 0x52;
    sendData[4] = 0x02;
    // 서버로 데이터 전송
    if (smartTCPClient1.SendByte(sendData) == false)
    {
      // 전송 실패 : 서버와 연결이 끊어진 경우 또는 연결되어 있지 않은 경우
      // 서버와의 연결을 해제 후 STEP-2을 참고하여 재연결하도록 합니다.
      smartTCPClient1.Close();
      // 서버와 연결 재시도
    }
  }
}
```

STEP-4 Client와 Server 간 수신 Event 처리하기

OnReceiveHandler 이벤트는 연결된 Server의 데이터 수신 시 발생되는 이벤트로, 인자값을 사용해 수신 데이터를 확인할 수 있습니다.

※ 자세한 내용은 "OnReceiveHandler 이벤트"를 참고하시기 바랍니다.

```
// Server로부터 데이터 수신 시 발생하는 이벤트
private void smartTCPClient1_OnReceiveHandler(byte[] datas)
{
  // 수신받은 데이터가 있는 경우
  if (datas.Length != 0)
  {
    // 수신 데이터 변환 처리
    string strReceiveData = SmartX.SmartTCPClient.ConvertAsciiByteToString(datas);
  }
  else
  {
    // 수신받은 데이터가 없다면 Server로 잘못된 데이터가 수신 됨을 알림
    smartTCPClient1.SendStringASCII( " Wrong Data!! " );
  }
}
```

STEP-5 Server와 연결 종료하기

Close() 메소드를 호출해 현재 연결된 Server와 연결을 종료할 수 있습니다.

※ 자세한 내용은 "Close() 메소드"를 참고하시기 바랍니다.

```
// Server와 연결 해제 처리
smartTCPClient.Close();
```

2) SmartTCPClient 인터페이스 설명

SmartTCPClient Component Interface		
속성		
InnerSocket : Socket	IsConnect : bool	MaxReceiveBufferSize : int
Port : int	ReceiveTimeout : int	ServerIPAddress : string
메소드		
Close() : void	Connect() : bool (+1개 오버로드)	ConvertAsciiByteToString(byte[] byteData): static string
ConvertUnicodeByteToString(byte[] byteData) : static string	ConvertUTF8ByteToString(byte[] byteData) : static string	SendByte(byte[] sendData) : bool
SendStringASCII(string strSend): bool	SendStringUnicode(string strSend) : bool	SetBlocking(bool bBlocking) : void
SetLocalIPAddress(string strLocalIPAddress): void		
이벤트		
OnConnected: SmartTCPClient.ConnectHandler	OnReceiveHandler : SmartTCPClient.ReceiveHandler	OnSent : SmartTCPClient.SendHandler

프로퍼티(속성) InnerSocket

SmartTCPClient에서 InnerSocket Class를 직접 접근할 수 있도록 InnerSocket 속성/메서드를 제공합니다.

참조 자세한 내용은 MSDN에서 소켓 클래스(InnerSocket Class)를 참조하시기 바랍니다.

C# 사용법

```
byte[] sendByteData = { 65, 66, 67 };  // sendByteData라는 이름의 바이트 배열을 선언 및 초기화
smartTCPClient1.InnerSocket.Send(sendByteData);  // sendByteData를 연결된 Socket에 전송
```

VB 사용법

```
Dim sendByteData As Byte() = {65, 66, 67}
smartTCPClient1.InnerSocket.Send(sendByteData)
```

프로퍼티(속성) IsConnect

SmartTCPClient가 서버와 연결되었는지 확인합니다.

주의 IsConnect 속성으로 서버 연결 확인 시 주의사항

SmartTCPClient가 서버와 연결된 경우, 즉 Connect() 메소드의 리턴값이 True일 때 IsConnect 속성값은 True가 됩니다. 이후 네트워크 상의 접속이 끊어져도(물리적인 연결이 끊기거나, 네트워크 연결이 불안정한 경우) IsConnect 속성값은 True를 유지합니다. 만약 서버에서 연결 해제 요청을 하여 올바르게 연결이 해제되었거나, SmartTCPClient의 Close() 메소드가 호출되면 IsConnect 속성은 False로 설정됩니다. 즉, IsConnect 속성은 네트워크 연결 상태를 체크할 수 없습니다.

참고 서버와 실제 연결 확인 방법

1. 서버와 실제로 연결이 되었는지 확인하기 위해서는 IsConnect 속성이 아닌, 데이터를 실제로 주고받는 과정이 정상적으로 이루어지는지 여부를 확인해야 합니다. 즉, 프로그램을 Request-Response 구조로 설계하면 통신 실패 여부를 확인해 서버와 실제 연결이 정상적으로 되었는지를 확인할 수 있습니다.
2. Ping Test를 이용하여 서버와 중간 네트워크 장비의 연결 상태를 확인합니다. 자세한 내용은 "SmartConfigs. IPSettings.Ping() 메소드"를 참고하시기 바랍니다.

- bool : 서버 연결 여부
  - true : 연결됨
  - false : 연결 안 됨

C# 사용법

```
// 현재 서버와 연결되지 않은 경우 서버와 연결을 시도
if (smartTCPClient1.IsConnect == false)
{
  smartTCPClient1.Connect();
}
```

VB 사용법

```
If smartTCPClient1.IsConnect = False Then
  smartTCPClient1.Connect()
End If
```

프로퍼티(속성) MaxReceiveBufferSize

한 번에 수신 가능한 수신 버퍼 사이즈를 설정합니다. 설정된 크기 이상의 데이터가 수신되는 경우, 수신된 데이터 크기를 MaxReceiveBufferSize로 나눈 횟수만큼 OnReceiveHandler 이벤트가 발생합니다.
※ Ex) 수신 데이터 크기 : 1500Byte, MaxReceiveBufferSize = 1000 → OnReceiveHandler 이벤트 2회 발생

- int : 한 번에 수신 가능한 수신 버퍼 사이즈 (단위 : Byte)

C# 사용법

```
smartTCPClient1.MaxReceiveBufferSize = 1000;  // 한 번에 수신 가능한 버퍼 사이즈를 1000Byte로 설정
```

VB 사용법

```
smartTCPClient1.MaxReceiveBufferSize = 1000
```

프로퍼티(속성) Port

TCP/IP 통신에 사용할 Port 번호를 설정합니다.

참고 Port 번호는 반드시 Server와 Client가 일치해야 합니다.

• int : TCP/IP Port 번호

C# 사용법

```
smartTCPClient1.Port = 3377;  // Port 번호를 3377번으로 설정
```

VB 사용법

```
smartTCPClient1.Port = 3377
```

프로퍼티(속성) ReceiveTimeout

SmartTCPClient에서 서버로부터 데이터 수신에 따른 응답 대기 시간을 설정합니다.

참고 ReceiveTimeout 속성값을 너무 크게 설정하는 경우 수신 데이터 처리 시간에 지연이 발생할 수 있습니다. 따라서 충분한 테스트를 거쳐 값을 설정하시기 바랍니다.

• int : 데이터 수신에 따른 응답 시간 (단위 : ms)

C# 사용법

```
smartTCPClient1.ReceiveTimeout = 3000;  // ReceiveTimeout을 3초로 설정
```

VB 사용법

```
smartTCPClient1.ReceiveTimeout = 3000
```

프로퍼티(속성) ServerIPAddress

접속할 서버의 IP-Address를 설정합니다.

• string : 서버의 IP-Address

C# 사용법

```
smartTCPClient1.ServerIPAddress = "192.168.1.100"; // 접속할 서버 IP 주소(192.168.1.100) 설정
```

VB 사용법

```
smartTCPClient1.ServerIPAddress = "192.168.1.100"
```

메소드(함수) SetLocalIPAddress

IEC非Lite-Series 제품군에서 유선랜과 무선랜을 동시에 사용하는 경우, SmartTCPClient의 통신 회선을 설정합니다.

◉ void SetLocalIPAddress(string strLocalIPAddress)

[인자]

• string strLocalIPAddress : 유선/무선 IP 중에 SmartTCPClient로 사용할 IP-Address

C# 사용법

```
// 유선 IP 주소가 192.168.0.10이고 무선 IP 주소가 192.168.1.20일때 사용자가 smartTCPClient1로
// 유선 IP 주소를 사용하기 위해 인자값으로 192.168.0.10를 입력함
smartTCPClient1.SetLocalIPAddress("192.168.0.10");
```

VB 사용법

```
smartTCPClient1.ServerIPAddress = "192.168.0.10"
```

메소드(함수) SetBlocking

SmartTCPClient의 연결 방식 및 데이터 송신 처리 방식(동기, 비동기)을 설정합니다. SmartTCPClient에서 동기, 비동기 처리 방식은 각 Blocking, NoneBlocking 처리 방식을 의미합니다.

권장 SmartTCPClient의 권장 연결, 송/수신 처리 모델

연결 처리 : 동기와 비동기 처리를 모두 지원하지만, 비동기 방식으로 사용할 것을 권장합니다.
송신 처리 : 동기와 비동기 처리를 모두 지원하지만, 동기 방식으로 사용할 것을 권장합니다.
수신 처리 : 비동기 처리(이벤트 모델) 방식만 지원합니다.

참고 SmartTCPClient에서 Server 연결 및 데이터 송신 처리 방식

SmartTCPClient는 수신 처리의 경우 비동기 처리(이벤트 모델)만을 지원하며, 연결 처리와 송신 처리 시 동기, 비동기 처리를 지원합니다. 즉, 비동기 처리로 설정하는 경우 연결 및 송신의 결과를 메소드의 리턴값에서 확인하는 것이 아닌 이벤트에서 확인해야 합니다.

[표] 관련 메소드 및 이벤트

관련 메소드	관련 이벤트
Connect(), SendByte(), SendStringASCII(), SendStringUnicode()	OnConnected, OnSent

※ 위 표의 메소드와 이벤트를 참고하시기 바랍니다.

◉ `void SetBlocking(bool bBlocking)`

[인자]

- `bool bBlocking` : 데이터 송신 처리 방식
  - `true` : 동기-Blocking
  - `false` : 비동기-NoneBlocking

C# 사용법

```
// SmartTCPClient의 송신 처리 방식을 동기 처리 방식으로 설정
smartTCPClient1.SetBlocking(true);
```

VB 사용법

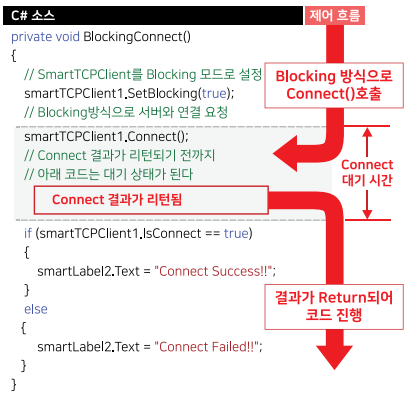
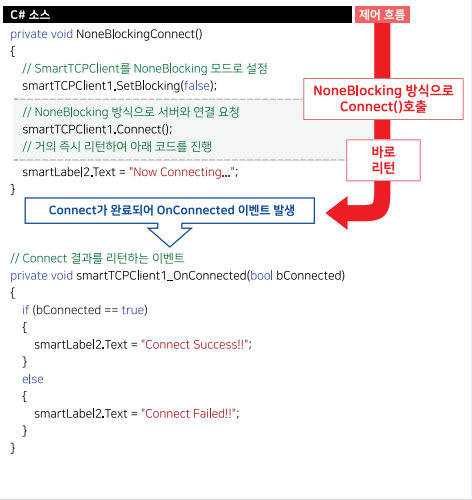
```
smartTCPClient1.SetBlocking(True)
```

참고 동기 방식과 비동기 방식의 차이

1. 동기 처리와 비동기 처리 비교

방식	동기 방식	비동기 방식
호출 방식	Blocking	None-Blocking
특징	실행 결과를 호출한 함수가 직접 리턴합니다.	실행 결과를 Event 함수가 호출되어 리턴합니다.
장점	1. None-Blocking 방식보다 프로그램 구조가 간단합니다. 2. 디버깅 작업이 간편합니다.	1. 응답이 오래 걸리는 작업(연결-Connect, 수신-Receive)의 처리를 기다리지 않고 바로 리턴하여 CPU 자원을 제대로 활용할 수 있습니다. 2. 프로그램의 사용자 인터페이스의 응답성이 좋아집니다.
단점	1. CPU 사용 없이 시간이 오래 걸리는 작업을(연결-Connect, 수신-Receive)할 경우 CPU 자원을 제대로 활용할 수 없습니다. 2. 프로그램 동작이 비효율적으로 운영됩니다.	1. 응답이 오래 걸리는 작업(연결-Connect, 수신-Receive)의 처리를 기다리지 않고 바로 리턴하여 CPU 자원을 제대로 활용할 수 있습니다. 2. 프로그램의 사용자 인터페이스의 응답성이 좋아집니다.

2. SmartTCPClient에서 Blocking 방식과 NoneBlocking 방식

방식	동기/Blocking	비동기/NoneBlocking
설정 방법	SetBlocking() 메서드의 인자값을 true로 설정합니다. -SetBlocking(true)	SetBlocking() 메서드의 인자값을 false로 설정합니다. -SetBlocking(false)
연결	Connect() 메서드 호출 시 작업이 완료될 때까지 프로세스를 멈추도록 하며, 인자값을 사용하여 일정 시간 동안만 대기하도록 설정할 수 있습니다. - Connect() - Connect(iConnectTime)	Connect() 메서드 호출 즉시 리턴하여 다른 작업을 수행할 수 있도록 하며, Connect 동작 완료값은 작업이 완료되면 OnConnected 이벤트가 발생하여 확인할 수 있습니다. - Connect() - OnConnected 이벤트
데이터 전송	Send 관련 메서드 호출 시 작업이 완료될 때까지 프로세스는 중단됩니다. - SendStringASCII(), SendStringUnicode(), SendByte()	Send 관련 메서드 호출 즉시 리턴하여 다른 작업을 수행할 수 있도록 하며, Send 동작 완료값은 작업이 완료되면 OnSent 이벤트가 발생하여 확인할 수 있습니다. - SendStringASCII(), SendStringUnicode(), SendByte() - OnSent 이벤트
데이터 수신	비동기/NoneBlocking 방식으로 데이터 수신이 발생하면 OnReceiveHandler 이벤트가 발생하여 데이터를 확인할 수 있습니다. - OnReceiveHandler 이벤트	
제어 흐름		

메소드(함수) Connect

서버에 연결 요청을 전송합니다.

참고

서버와 클라이언트가 동기 통신 방식일 때 SetBlocking(true)로 연결하는 경우 어떠한 문제(네트워크가 불안정한 경우, 서버 프로그램이 시작하지 않은 경우, 네트워크 선이 빠진 경우 등)로 인해 연결이 지연되었을 때 경우 프로그램이 연결 대기로 인해 다른 작업을 수행하지 못하는 현상을 개선할 수 있습니다.

자세한 내용은 “Socket 통신에서 Blocking 방식과 NoneBlocking 방식의 차이”, “SetBlocking() 메소드” 내용을 참고하시기 바랍니다.

◉ `bool Connect()`
◉ `bool Connect(int iTimeout)`

[인자]

• `int iTimeout` : 연결 요청 제한 시간 (단위 : ms)

[리턴값]

• `bool` : 연결 성공 여부
  - `true` : 성공
  - `false` : 실패

[표] 통신 방식에 따른 Connect() 메소드 동작

메소드	통신 방식	반드시 설정할 내용 SetBlocking()	메소드 호출 시 동작
Connect()	동기 방식	SetBlocking (true)	연결 요청이 만료될 때까지 호출 시점에서 Blocking되며, 연결 성공 여부를 리턴값으로 확인 ※ 만료 시간은 네트워크 상태에 따라 상이함 ※ 서버와의 연결 성공 시 대기하지 않고 리턴
	비동기 방식	SetBlocking (false)	서버와의 연결 요청을 시도하고 바로 false를 리턴되며 실제 연결 확인 여부는 OnConnected 이벤트에서 bConnected 인자값으로 확인
Connect (int iTimeout)	동기 방식	SetBlocking (true)	설정 시간(iTimeout)만큼 호출 시점에서 Blocking되며, 연결 성공 여부를 리턴값으로 확인 ※ 서버와의 연결 성공 시 대기하지 않고 리턴
	비동기 방식	지원 안 함	지원 안 함

C# 사용법

```
// 동기 통신 방식으로 사용 시 반드시 SetBlocking(true)로 설정
smartTCPClient1.SetBlocking(true);
// CASE-1. 일정 시간 동안 서버와의 연결 요청을 시도하되 연결 실패 시까지 Blocking되어 있음
smartTCPClient1.Connect();
// CASE-2. 최대 설정 시간(3초)만큼 서버와의 연결을 시도하되 연결 성공 시 바로 리턴
smartTCPClient1.Connect(3000);
// ------------------------------------------------------------------
// 비동기 통신 방식으로 사용 시 반드시 SetBlocking(false)로 설정
// 연결 확인은 OnConnected 이벤트에서 확인 가능
smartTCPClient1.SetBlocking(false);
smartTCPClient1.Connect();

// 비동기 통신 방식일 때 서버와의 실제 연결 결과를 확인
private void smartTCPClient1_OnConnected(bool bConnected)
{
  if (bConnected == true)
  {
    // 서버와 연결 성공 시 처리 코드 작성
  }
}
```

VB 사용법

```
smartTCPClient1.SetBlocking(True)
' smartTCPClient1.Connect()
smartTCPClient1.Connect(3000)
smartTCPClient1.SetBlocking(False)
smartTCPClient1.Connect()

Private Sub smartTCPClient1_OnConnected(ByVal bConnected As Boolean) Handles smartTCPClient1.
OnConnected
  If bConnected = True Then
    ' 서버와 연결 성공 시 처리 코드 작성
  End If
End Sub
```

메소드(함수) Close

서버와의 연결을 해제합니다.

◉ void Close()

C# 사용법

```
smartTCPClient1.Close();  // 서버와의 연결을 해제
```

VB 사용법

```
smartTCPClient1.Close()
```

메소드(함수) SendByte

SmartTCPClient와 연결된 서버에 바이트 배열 데이터를 전송합니다.

참고 SetBlocking() 메소드의 인자값을 false로 하여 데이터 송신 처리를 비동기 방식으로 하는 경우, 데이터 송신 메소드 호출 즉시 false를 리턴하며, 실제 데이터 송신 처리 결과는 OnSent 이벤트가 발생하여 확인할 수 있습니다. "SetBlocking() 메소드", "OnSent 이벤트" 내용을 참고하시기 바랍니다.

◉ bool SendByte(byte[] sendData)

[인자]

• byte[] sendData : 서버로 전송할 바이트 배열형 데이터

[리턴값]

• bool : 데이터 전송 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
byte[] sendData = new byte[3];  // sendData라는 이름의 바이트 배열을 선언하고 배열 초기화
sendData[0] = 0x33;  sendData[1] = 0x45;  sendData[2] = 0x71;
smartTCPClient1.SendByte(sendData);  // 서버로 바이트 배열형 데이터 sendData를 전송
```

VB 사용법

```
Dim sendData As Byte() = new Byte(2) {}
sendData(0) = &H33  :  sendData(1) = &H45  :  sendData(2) = &H71
smartTCPClient1.SendByte(sendData)
```

메소드(함수) SendStringASCII

SmartTCPClient와 연결된 서버에 string형 데이터를 ASCII 인코딩 방식으로 전송합니다.

참고 SetBlocking() 메소드의 인자값을 false로 하여 데이터 송신 처리를 비동기 방식으로 하는 경우, 데이터 송신 메소드 호출 즉시 false를 리턴하며, 실제 데이터 송신 처리 결과는 OnSent 이벤트가 발생하여 확인할 수 있습니다. "SetBlocking() 메소드", "OnSent 이벤트" 내용을 참고하시기 바랍니다.

◉ bool SendStringASCII(string strSend)

[인자]

• string strSend : 서버로 전송할 string형 데이터

[리턴값]

• bool : 데이터 전송 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// sendStrData라는 이름의 String 변수를 선언하고 변수값을 초기화
string sendStrData = " HNS, Hardware And Software Total Solution! ";
// 서버에 string형태의 sendStrData를 ASCII 문자 형태로 전송
smartTCPClient1.SendStringASCII(sendStrData);
```

VB 사용법

```
Dim sendStrData As String = " HNS, Hardware And Software Total Solution! "
smartTCPClient1.SendStringASCII(sendStrData)
```

메소드(함수) SendStringUnicode

SmartTCPClient와 연결된 서버에 string형 데이터를 Unicode 인코딩 방식으로 전송합니다.

참고 SetBlocking() 메소드의 인자값을 false로 하여 데이터 송신 처리를 비동기 방식으로 하는 경우, 데이터 송신 메소드 호출 즉시 false를 리턴하며, 실제 데이터 송신 처리 결과는 OnSent 이벤트가 발생하여 확인할 수 있습니다.
"SetBlocking() 메소드", "OnSent 이벤트" 내용을 참고하시기 바랍니다.

◉ bool SendStringUnicode(string strSend)

[인자]

• string strSend : 서버로 전송할 string형 데이터

[리턴값]

• bool : 데이터 전송 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// sendStrData라는 이름의 String 변수를 선언하고 변수값을 초기화
string sendStrData = " HNS, Hardware And Software Total Solution! ";
// 서버에 string형태의 sendStrData를 Unicode 문자 형태로 전송
smartTCPClient1.SendStringUnicode(sendStrData);
```

VB 사용법

```
Dim sendStrData As String = " HNS, Hardware And Software Total Solution! "
smartTCPClient1.SendStringUnicode(sendStrData)
```

정적 메소드(함수) ConvertAsciiByteToString, ConvertUnicodeByteToString, ConvertUTF8ByteToString

※ Static(정적) 메소드로 인스턴싱 없이 사용합니다. ※

Byte[] 데이터를 ASCII, Unicode, UTF-8 문자로 변환합니다.

• ConvertAsciiByteToString() : 바이트 배열 데이터를 ASCII 문자로 변환합니다.
• ConvertUnicodeByteToString() : 바이트 배열 데이터를 Unicode 문자로 변환합니다.
• ConvertUTF8ByteToString() : 바이트 배열 데이터를 UTF-8 문자로 변환합니다.

참고 서버로부터 수신되는 바이트 배열 형태의 데이터를 편리하게 변환할 수 있습니다.

◉ static string ConvertAsciiByteToString(byte[] byteData)
◉ static string ConvertUnicodeByteToString(byte[] byteData)
◉ static string ConvertUTF8ByteToString(byte[] byteData)

[인자]

• byte[] byteData : 바이트 배열 데이터

[리턴값]

• static string : 변환된 문자열

C# 사용법

```
// 서버로부터 데이터 수신 시 발생하는 이벤트
private void smartTCPClient1_OnReceiveHandler(byte[] datas)
{
```

```
  string strRecData = null;
  // 수신된 바이트 배열 데이터를 ASCII 문자열로 변환함
  strRecData = SmartTCPClient.ConvertAsciiByteToString(datas);
  // 수신된 바이트 배열 데이터를 UniCode 문자열로 변환함
  strRecData = SmartTCPClient.ConvertUnicodeByteToString(datas);
  // 수신된 바이트 배열 데이터를 UTF-8 문자열로 변환함
  strRecData = SmartTCPClient.ConvertUTF8ByteToString(datas);
}
```

VB 사용법

```
Private Sub smartTCPClient1_OnReceiveHandler(ByVal datas As Byte()) Handles
smartTCPClient1.OnReceiveHandler
  Dim strRecData As String = Nothing
  strRecData = SmartTCPClient.ConvertAsciiByteToString(datas)
  strRecData = SmartTCPClient.ConvertUnicodeByteToString(datas)
  strRecData = SmartTCPClient.ConvertUTF8ByteToString(datas)
End Sub
```

이벤트 OnConnected

SetBlocking(false)로 설정하여 SmartTCPClient를 비동기 통신 방식으로 설정한 경우, Connect() 메소드 호출 시 발생하는 이벤트입니다. SmartTCPClient와 서버의 실제 연결 결과를 인자값을 이용해 확인할 수 있습니다.

참고 "Socket 통신에서 동기 방식과 비동기 방식의 차이", "SetBlocking(), Connect() 메소드" 내용을 참고하시기 바랍니다.

◉ OnConnected(bool bConnected)

[인자]

- bool : 연결 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// 비동기 통신 방식일 때 서버와 실제 연결 결과를 확인, SetBlocking(false)
private void smartTCPClient1_OnConnected(bool bConnected)
{
  if (bConnected == true)
  {
    // 서버와 연결 성공 시 처리 코드 작성
  }
}
```

VB 사용법

```
Private Sub smartTCPClient1_OnConnected(ByVal bConnected As Boolean) Handles smartTCPClient1.
OnConnected
  If bConnected = True Then
    ' 서버와 연결 성공 시 처리 코드 작성
  End If
End Sub
```

이벤트 OnReceiveHandler

SmartTCPClient가 서버로부터 데이터 수신 시 발생하는 이벤트입니다. 인자값에서 수신받은 데이터를 확인할 수 있습니다.

◉ OnReceiveHandler(byte[] datas)

[인자]

• byte[] datas : 수신 데이터

C# 사용법

```
private void smartTCPClient1_OnReceiveHandler(byte[] datas)
{
  // 수신된 바이트 배열 데이터를 UniCode 문자로 변환하여 smartLabel1에 출력
  if (datas.Length != 0)
  {
    smartLabel1.Text = SmartTCPClient.ConvertByteToUnicode(datas);
  }
}
```

VB 사용법

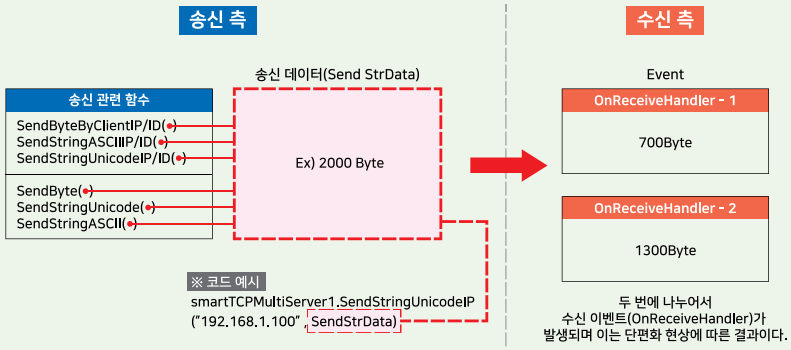
```
Private Sub smartTCPClient1_OnReceiveHandler(ByVal datas As Byte())
  If datas.Length <> 0 Then
    smartLabel1.Text = SmartTCPClient.ConvertByteToUnicode(datas)
  End If
End Sub
```

참조 대량의 데이터 연속 송/수신 시 OnReceiveHandler 이벤트에서 데이터 단편화(Fragmentation) 발생 원인과 해결 방법

[발생 원인]

TCP/IP 통신에서 대량의 데이터를 연속으로 송/수신하는 경우 수신 측 이벤트(OnReceiveHandler)에서 데이터의 단편화(Fragmentation)현상이 발생합니다.

다음 그림에서 송신 측에서 보낸 데이터(2000Byte)가 수신 측의 수신 이벤트(OnReceiveHandler)가 두 번으로 나뉘어서 발생하며 700Byte와 1300Byte로 데이터 단편화가 발생합니다.



[해결 방법]

위의 문제를 해결하기 위해서는 수신된 데이터의 패킷을 프레임화 할 수 있는 별도의 처리가 필요합니다.

※ "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 73. [C#, VB.NET] TCP/IP 통신에서 송/수신 시 수신 측 이벤트에서 데이터 단편화(Fragmentation)발생 원인과 해결 방법"내용을 참조하시기 바랍니다.

이벤트 OnSent

SetBlocking(false)로 설정하여 비동기 통신 방식으로 설정하는 경우, SmartTCPClient에서 서버로 데이터 전송이 완료되면 발생하는 이벤트입니다.

참고 "Socket 통신에서 동기 방식과 비동기 방식의 차이", "SetBlocking(), SendByte(), SendString ASCII(), SendStringUnicode() 메소드" 내용을 참고하시기 바랍니다.

◉ OnSent(int iBytesSent)

[인자]

• int iBytesSent : 서버로 전송한 데이터의 길이 (단위 : Byte)

C# 사용법

```
// 통신 처리 방식이 NONE-Blocking 경우에서 클라이언트 → 서버로 데이터 전송 시 발생하는 이벤트
private void smartTCPClient1_OnSent(int iBytesSent)
{
  // iBytesSent와 전송한 데이터의 길이가 같은 경우 데이터 전송 성공 카운트 증가
  if (iBytesSent == txtSendData.Text.Length)
  {
    m_SendOK.Text = (++Server_SendOK).ToString();
  }
  // iBytesSent 인자값이 -1인 경우, 데이터 전송 실패 카운트 증가
  else if (iBytesSent == -1)
  {
    m_SendNG.Text = (++Server_SendNG).ToString();
  }
}
```

VB 사용법

```
Private Sub smartTCPClient1_OnSent(ByVal iBytesSent As System.Int32) Handles smartTCPClient1.
OnSent
  If iBytesSent = txtSendData.Text.Length Then
    m_Sever_SendOK.Text = (++Server_SendOK).ToString()
  ElseIf iBytesSent = -1 Then
    m_Sever_SendNG.Text = (++Server_SendNG).ToString()
  End If
End Sub
```

3) SmartTCPClient 예제 사용하기

SmartTCPClient를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

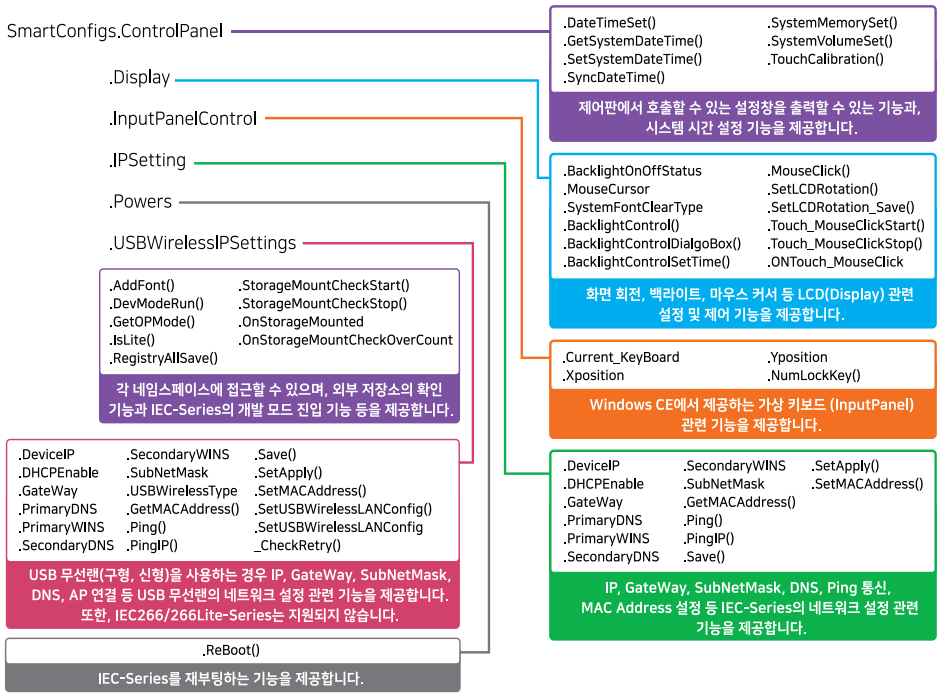
홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → “SmartTCPClient”, “SmartTCPClient_NoneBlocking”

3. SmartConfigs

SmartConfigs는 IEC-Series의 환경 설정 및 특수 기능을 제어할 수 있는 컴포넌트입니다.

- IEC-Series의 LAN Device(유선/무선)의 IP/MAC Address 설정 기능
  ※ IEC266의 경우 무선랜 IP를 변경하는 기능은 제공하지 않습니다.
- Display(LCD) 관련 설정 기능
  - ㄴ Backlight On, Off 제어 및 LCD의 회전(Rotation) 기능
  - ㄴ 마우스 커서 제어 기능 (시스템상에서 전역적 제어)
  - ㄴ 클리어 타입 폰트 적용 기능
  - ㄴ 화면 터치 또는 마우스 클릭을 O/S 레벨에서(Event Hooking) 전역적 Click Event 발생 기능
- 시스템 폰트 관련 설정 기능
- IEC-Series의 Reboot 기능
- 제어판 기능 호출
  - ㄴ Windows CE의 제어판에 있는 일부 기능을 호출하는 기능
  - ㄴ 인터넷 타임 서버를 통한 날짜 시간 동기화 기능
- 가상 키보드(InputPanel) 기능 제어
  - ㄴ InputPanel의 종류(LARGE, SMALL) 및 위치 설정 기능
- 외부저장장치(SD Card, USB Memory)의 연결 확인 기능
- RunTime Mode에서 Development Mode 전환 기능

1) SmartConfigs 네임스페이스 설명



SmartConfigs는 IEC-Series의 환경설정을 위해 다양한 기능을 제공하고 있으며, 각 기능을 네임스페이스로 분류하였습니다. 제공하는 네임스페이스는 총 7개이며 네임스페이스별 기능은 위 그림과 같습니다.

※ 자세한 내용은 "인터페이스 설명"에서 각 네임스페이스를 참고하시기 바랍니다.

2) 프로그래밍 적용 가이드

CASE-1 제어판의 시스템 설정창 출력하기

SmartConfigs는 제어판에서 접근할 수 있는 각종 시스템 설정창을 코드상에서 접근할 수 있습니다.

※ 자세한 내용은 "ControlPanel.DateTimeSet(), ControlPanel.SystemMemorySet(), ControlPanel.SystemVolumeSet(), ControlPanel.TouchCalibration() 메소드"를 참고하시기 바랍니다.

```
smartConfigs1.ControlPanel.DateTimeSet();  // 시스템 날짜 및 시간 설정창을 출력합니다.
smartConfigs1.ControlPanel.SystemMemorySet();  // 시스템 메모리 설정창을 출력합니다.
smartConfigs1.ControlPanel.SystemVolumeSet();  // 시스템 볼륨과 소리 설정창을 출력합니다.
smartConfigs1.ControlPanel.TouchCalibration(); // 터치 스크린 보정(Calibration)창을 출력합니다.
```

CASE-2 타임 서버를 사용하여 시스템 시간 설정하기

SmartConfigs는 시스템 시간을 읽거나 설정할 수 있으며, 네트워크에 연결된 경우 타임 서버에서 현재 시간을 동기화 할 수 있습니다. 타임 서버는 미리 지정된 타임 서버를 사용하거나, 사용자가 직접 지정할 수 있습니다.

※ 자세한 내용은 "ControlPanel.GetSystemDateTime, ControlPanel.SetSystemDateTime, ControlPanel.SyncDateTime 속성"을 참고하시기 바랍니다.

```
// InterNet Time Server DateTime 동기화, 시스템에서 제공하는 타임 서버 사용
smartConfigs1.ControlPanel.SyncDateTime();
// 또는 사용자가 원하는 타임 서버를 직접 입력하는 방식
smartConfigs1.ControlPanel.SyncDateTime( " nist1-macon.macon.ga.us " );
```

CASE-3 화면 회전하기

SmartConfigs는 IEC-Series의 화면(LCD)을 원하는 방향으로 회전시킬 수 있습니다.

※ 자세한 내용은 "Display.SetLCDRotation(), Display.SetLCDRotation_Save() 메소드"를 참고하시기 바랍니다.

```
smartConfigs1.Display.SetLCDRotation(SmartX.CDisplay.ROT.ROT90);       // LCD 90도 회전하기
smartConfigs1.Display.SetLCDRotation_Save(SmartX.CDisplay.ROT.ROT90);  // LCD 90도 회전하고 저장
```

CASE-4 백라이트 제어하기

SmartConfigs는 IEC-Series의 화면(LCD) 백라이트(BackLight)를 제어할 수 있습니다.

※ 자세한 내용은 "Display.BacklightOnOffStatus 속성", "Display.BacklightControl(), Display.BacklightControlDialogBox(), Display.BacklightControlSetTime() 메소드"를 참고하시기 바랍니다.

```
if (smartConfigs1.Display.BacklightOnOffStatus == SmartX.CDisplay.OnOff.ON)
{
  // LCD BackLight가 켜져있는 경우 LCD Backlight OFF하기
  smartConfigs1.Display.BacklightControl(SmartX.CDisplay.OnOff.OFF);
}
smartConfigs1.Display.BacklightControlDialogBox(); // 제어판의 조명 설정창 출력하기
// LCD Backlight OFF 시간 설정하기. 0초가 입력되면 Backlight ON과 동일함
smartConfigs1. Display.BacklightControlSetTime(30);
```

CASE-5 마우스 커서 제어하기

SmartConfigs는 IEC-Series의 마우스 커서를 제어할 수 있습니다.

※ 자세한 내용은 "Display.MouseCursor 속성", "Display.MouseClick(), Display.Touch_MouseClickStart(), Display.Touch_MouseClickStop() 메소드", "ONTouch_MouseClick 이벤트"를 참고하시기 바랍니다.

```
smartConfigs1.Display.MouseCursor = SmartX.CDisplay.OnOff.OFF;  // 마우스 커서를 숨김
smartConfigs1.Display.MouseClick(100, 100);  // (100, 100) 좌표를 강제 클릭
smartConfigs1.Display.Touch_MouseClickStart();  // 사용자 터치 또는 마우스 클릭 감지 시작
// Touch_MouseClickStart() 메소드 호출 후 터치 시 발생하는 이벤트
private void smartConfigs1_ONTouch_MouseClick(SmartX.SmartConfigs.MOUSESTATECODE eMouse,
                                              int iXPos, int iYPos)
{
  // 마우스 다운 상태인 경우 사용자 터치 또는 마우스 클릭 감지 중지
  if (eMouse == SmartX.SmartConfigs.MOUSESTATECODE.MOUSE_DOWN)
  {
    smartConfigs1.Display.Touch_MouseClickStop();
  }
}
```

CASE-6 IEC-Series의 네트워크 주소 설정하기(유선 LAN)

SmartConfigs는 IEC-Series의 IP, GateWay, SubNetMask, DNS, WINS 주소를 설정할 수 있습니다.
이중 IP, GateWay, SubNetMask는 유동, 고정 설정을 지원합니다.

※ 자세한 내용은 "IPSettings.DHCPEnable, IPSettings.DeviceIP, IPSettings.GateWay, IPSettings.SubNetMask, IPSettings.PrimaryDNS, IPSettings.SecondaryDNS, IPSettings.PrimaryWINS, IPSettings.SecondaryWINS 속성", "IPSettings.Save(), IPSettings.SetApply() 메소드"를 참고하시기 바랍니다.

```
smartConfigs1.IPSettings.DHCPEnable = 0;  // IP를 고정 IP로 사용
// 아래 IP 주소는 예시입니다.
smartConfigs1.IPSettings.DeviceIP = "192.168.0.1";      // IEC-Series의 IP를 설정
smartConfigs1.IPSettings.GateWay = "192.168.1.1";       // IEC-Series의 GateWay를 설정
smartConfigs1.IPSettings.SubNetMask = "255.255.255.0"; // IEC-Series의 SubNetMask를 설정
smartConfigs1.IPSettings.PrimaryDNS = "8.8.8.8";        // IEC-Series의 PrimaryDNS를 설정
smartConfigs1.IPSettings.SecondaryDNS = "8.8.4.4";      // IEC-Series의 SecondaryDNS를 설정
smartConfigs1.IPSettings.PrimaryWINS = "127.0.0.1";     // IEC-Series의 PrimaryWINS를 설정
smartConfigs1.IPSettings.SecondaryWINS = "127.0.0.2";   // IEC-Series의 SecondaryWINS를 설정
// 시스템을 다시 시작할 경우 설정된 IP 관련 주소를 유지하기 위해 레지스트리에 저장
smartConfigs1.IPSettings.Save();
// 변경된 IP 주소값을 바로 적용하기 위해 Network Driver를 Rebind 처리
smartConfigs1.IPSettings.SetApply();
```

CASE-7 MAC Address 설정하기(유선 LAN)

SmartConfigs는 IEC-Series의 MAC Address를 읽거나, 설정할 수 있습니다. IEC-Series를 네트워크 망에 연결 시 반드시 설정하시기 바랍니다.

※ 자세한 내용은 "IPSettings.GetMACAddress(), IPSettings.SetMACAddress() 메소드"를 참고하시기 바랍니다.

```
private void SetMAC()
{
  byte[] baSetMac = new byte[6];  // MAC Address 설정을 위한 바이트 배열
  // 앞 3자리는 고정
  baSetMac[0] = 0x60; baSetMac[1] = 0xDB; baSetMac[2] = 0x2A;
  // 뒤 3자리는 변경
  baSetMac[3] = 0x03; baSetMac[4] = 0xFB; baSetMac[5] = 0xA2;
  // MAC Address 정상 설정 시 제품 재부팅
  if (smartConfigs1.IPSettings.SetMACAddress(baSetMac) == true)
  {
    smartConfigs1.Powers.Reboot();
```

```
  }
}

private void GetMAC()
{
  // IEC-Series에 설정된 MAC Address를 읽음
  byte[] baGetMac = smartConfigs1.IPSettings.GetMACAddress();
  // 읽어온 MAC Address를 SmartListBox에 출력
  for (int i = 0; i < baGetMac.Length; i++)
  {
    smartListBox1.AddItem(baGetMac[i].ToString());
  }
}
```

CASE-8 USB 무선랜의 네트워크 주소 설정하기

SmartConfigs는 USB 무선랜(구형, 신형)의 IP, GateWay, SubNetMask, DNS, WINS 주소를 설정할 수 있습니다. 이중 IP, GateWay, SubNetMask는 유동, 고정 설정을 지원하며, 설정 전 반드시 사용하는 USB 무선랜의 타입(구형, 신형)을 설정해야 합니다.

※ 자세한 내용은 "USBWirelessIPSettings.USBWirelessType, USBWirelessIPSettings.DHCPEnable, USBWirelessIPSettings.DeviceIP, USBWirelessIPSettings.GateWay, USBWirelessIPSettings.SubNetMask, USBWirelessIPSettings.PrimaryDNS, USBWirelessIPSettings.SecondaryDNS, USBWirelessIPSettings.PrimaryWINS, USBWirelessIPSettings.SecondaryWINS 속성", " USBWirelessIPSettings.Save(), USBWirelessIPSettings.SetApply() 메소드"를 참고하시기 바랍니다.

```
// 신형 USB 무선랜 사용 중
smartConfigs1.USBWirelessIPSettings.USBWirelessType = SmartX.CWUSBIPSetting.USBWLANTYPE.RT2870;
smartConfigs1.USBWirelessIPSettings.DHCPEnable = 0;  // IP를 고정 IP로 사용
// 아래 IP 주소는 예시입니다.
smartConfigs1.USBWirelessIPSettings.DeviceIP = " 192.168.0.1 ";      // 무선랜의 IP를 설정
smartConfigs1.USBWirelessIPSettings.GateWay = " 192.168.1.1 ";       // 무선랜의 GateWay를 설정
smartConfigs1.USBWirelessIPSettings.SubNetMask = " 255.255.255.0 "; // 무선랜의 SubNetMask를 설정
smartConfigs1.USBWirelessIPSettings.PrimaryDNS = " 8.8.8.8 ";        // 무선랜의 PrimaryDNS를 설정
smartConfigs1.USBWirelessIPSettings.SecondaryDNS = " 8.8.4.4 ";      // 무선랜의 SecondaryDNS를 설정
smartConfigs1.USBWirelessIPSettings.PrimaryWINS = " 127.0.0.1 ";     // 무선랜의 PrimaryWINS를 설정
smartConfigs1.USBWirelessIPSettings.SecondaryWINS = " 127.0.0.2 ";   // 무선랜의 SecondaryWINS를 설정
// 시스템을 다시 시작할 경우 설정된 IP 관련 주소를 유지하기 위해 레지스트리에 저장
smartConfigs1.USBWirelessIPSettings.Save();
// 변경된 IP 주소값을 바로 적용하기 위해 Network Driver를 Rebind 처리
smartConfigs1.USBWirelessIPSettings.SetApply();
```

CASE-9 AP에 연결하기

SmartConfigs는 USB 무선랜을 AP에 연결할 수 있습니다. USB 무선랜 타입(신형, 구형)을 구분하며, AP가 암호화된 경우에도 연결할 수 있습니다.

※ 자세한 내용은 "USBWirelessIPSettings.SetUSBWirelessLANConfig() 메소드"를 참고하시기 바랍니다.

```
// 신형 USB 무선랜 사용 시 AP가 개방된 경우
smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig(SmartX.CIPSetting.USBWLANTYPE.RT2870,
                                                            " testAP_FREE " );
// 신형 USB 무선랜 사용 시 AP가 암호화된 경우
smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig(SmartX.CIPSetting.USBWLANTYPE.RT2870,
                                                            " testAP " , " 12345678 " );
```

CASE-10 Ping 사용하기

SmartConfigs는 IEC-Series가 네트워크를 통해 특정한 호스트에 도달 가능한지 체크하기 위해 Ping 통신 기능을 제공합니다. 도메인 이름이나 IP 주소를 사용할 수 있습니다.

※ 자세한 내용은 "IPSettings.Ping(), IPSettings.PingIP() 메소드"를 참고하시기 바랍니다.

```
SmartX.CIPSetting.PINGRESULTINFO pingResultInfo = new SmartX.CIPSetting.PINGRESULTINFO();
// 도메인 이름으로 Ping 통신 시도
if (smartConfigs1.IPSettings.Ping( " google.co.kr " , ref pingResultInfo) == true)
{
  // Ping 통신이 성공한 경우
}
// IP 주소로 Ping 통신 시도
if (smartConfigs1.IPSettings.PingIP( " 192.168.1.1 " , ref pingResultInfo) == true)
{
  // Ping 통신이 성공한 경우
}
```

3) SmartConfigs 인터페이스 설명

SmartConfigs Component Interface		
속성		
ControlPanel : CControlPanel	Display : CDisplay	InputPanelControl : InputPanelCon
IPSettings : CIPSetting	Powers : CPower	USBWirelessIPSettings : CWUSBIPSetting
메소드		
AddFont() : static void	DevModeRun() : void	GetOPMode() : SmartForm.RUNMODE
IsLite() : bool	RegistryAllSave() : void	StorageMountCheckStart(int iCheckInterval, int iCheckLimitCount, string strStoragePath) : void
StorageMountCheckStop() : void		
이벤트		
OnStorageMounted : EventHandler	OnStorageMountCheckOverCount : EventHandler	

정적 메소드(함수) AddFont

폰트를 Fonts 폴더에 등록하지 않고 동적으로 등록하여 적용 및 사용 가능하게 합니다.

중요 AddFont() 메소드의 호출은 프로그램 실행 시 가장 먼저 호출되어야 하는 메소드입니다. 따라서 반드시 사용법의 내용과 같이 폼의 생성자 또는 program.cs에서 해당 메소드를 호출하시기 바랍니다.

◉ `static void AddFont(string strFontPathname)`

[인자]

• `string strFontPathname` : 폰트가 저장된 경로

C# 사용법

```
public Form1()
{
  // 반드시 Program.cs 또는 폼 생성자에서 메서드를 호출
  SmartConfigs.AddFont( " Flash Disk₩₩TestFonts₩₩gulim.ttc " );
  InitializeComponent();
}
```

VB 사용법

```
Public Sub New()
 SmartConfigs.AddFont( " Flash Disk₩₩TestFonts₩₩gulim.ttc " )
 InitializeComponent()
End Sub
```

메소드(함수) DevModeRun

Windows Shell(바탕화면)을 로딩하여 실행시킵니다. 즉, IEC-Series가 RunTime 모드인 경우 Developer 모드로 강제 전환합니다.

◉ void DevModeRun()

C# 사용법

```
smartConfigs1.DevModeRun();
```

VB 사용법

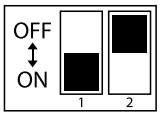
```
smartConfigs1.DevModeRun()
```

메소드(함수) GetOPMode

IEC-Series의 동작 모드를 확인합니다.

참조 Dip 스위치 위치 및 IEC-Series 동작 모드 확인법

Dip 스위치 1번이 ON이라면 RunTime 모드이고 OFF라면 Developer 모드입니다.

Dip 스위치 위치	1번 스위치 동작
	

※ 딥(Dip) 스위치에 관련한 자세한 내용은 "홈페이지(www.hnsts.co.kr)→ 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 동작 모드 관련"을 참조하시기 바랍니다.

◉ SmartForm.RUNMODE GetOPMode()

[리턴값]

- SmartForm.RUNMODE : IEC-Series의 동작 모드
  - SmartForm.RUNMODE.RUN : RUN 모드
  - SmartForm.RUNMODE.DEVELOPER : DEVELOPER 모드

C# 사용법

```
if (smartConfigs1.GetOPMode() == SmartForm.RUNMODE.DEVELOPER)
{
  // 현재 DEVELOPER 모드임
}
else
{
  // 현재 RUN 모드임
}
```

VB 사용법

```
If smartConfigs1.GetOPMode() = SmartForm.RUNMODE.DEVELOPER Then
  ' 현재 DEVELOPER 모드임
```

```
Else
  ' 현재 RUN 모드임
End If
```

메소드(함수) IsLite

현재 제품이 IECLite-Series인지 확인합니다.

◉ bool IsLite()

[리턴값]

- bool : IECLites-Series 여부
  - true : 현재 제품이 IECLite-Series
  - false : 현재 제품이 IEC非Lite-Series

C# 사용법

```
if(smartConfigs1.IsLite() == true)
{
  // 현재 제품이 IECLite-Series임
}
```

VB 사용법

```
If smartConfigs1.IsLite() = True Then
   ' 현재 제품이 IECLite-Series임
End If
```

메소드(함수) RegistryAllSave

IEC-Series의 레지스트리를 저장합니다.

참고 IEC-Series의 설정값을 변경하고 RegistryAllSave() 메소드를 호출하지 않을 경우, 재부팅 시 설정 정보가 초기화 됩니다.

주의 RegistryAllSave() 메소드를 자주 호출할 경우 Flash Disk의 데이터가 손상될 수 있습니다.

◉ void RegistryAllSave()

C# 사용법

```
smartConfigs1.RegistryAllSave();
```

VB 사용법

```
smartConfigs1.RegistryAllSave()
```

메소드(함수) StorageMountCheckStart

외부저장장치(SD Card, USB Memory) 연결 확인 및 특정 경로에 있는 File에 접근할 수 있는지 확인을 시작합니다.

주의 외부저장장치의 파일 접근 가능 여부를 확인할 때 아래와 같은 이유로 SmartMemory의 EvtExternalStorageAttached 이벤트를 사용하시는 것을 권장합니다.
1. 해당 함수 사용 시 최초로 액세스할 때 외부저장장치에 파일이 없을 경우 접근 가능 여부가 확인되지 않습니다.
2. SmartMemory의 EvtExternalStorageAttached 이벤트에 비해 CPU 리소스를 상대적으로 더 많이 사용합니다.
사용 방법은 " Part-Ⅶ. 하드웨어 장치 제어 컴포넌트 → 4. SmartMemory → EvtExternalStorageAttached 이벤트"의 사용법을 참고하시기 바랍니다.

참고 RunTime 모드에서 외부 저장 장치 접근 시 참고사항

IEC-Series가 RunTime 모드로 설정되어 프로그램이 구동될 경우, O/S가 외부 저장 장치를 인식하는데까지 어느 정도 시간이 소요됩니다. 따라서 프로그램 실행 직후 외부 저장 장치에 접근하게 되면 I/O Exception이 발생할 수 있습니다. 이러한 경우 StorageMountCheckStart() 메소드를 사용하면 외부저장장치의 인식이 완료되면 OnStorageMounted 이벤트가 발생하여 안전하게 외부저장장치에 접근할 수 있습니다.

참고 USB Memory의 경우 SD CARD에 비해 정상 연결(Mount)되는 시간이 차이가 있을 수 있으니 정상 연결 확인이 안 되는 경우 Mount 체크 시간 간격과 체크 횟수를 조절하시기 바랍니다.

◉ `void StorageMountCheckStart(int iCheckInterval, int iCheckLimitCount, string strStoragePath)`

[인자]

- `int iCheckInterval` : 외부저장장치 확인 간격(Interval) (단위 : ms)
- `int iCheckLimitCount` : 외부저장장치 확인 횟수
- `string strStoragePath` : 접근 가능 여부를 확인할 외부저장장치의 특정 경로

C# 사용법

```
// StorageMountCheck 시작
smartConfigs1.StorageMountCheckStart(1000, 5, " SD Card₩₩Run₩₩1.txt " );
smartConfigs1.StorageMountCheckStart(1000, 5, " 하드 디스크₩₩Run₩₩1.txt " );
```

VB 사용법

```
smartConfigs1.StorageMountCheckStart(1000, 5, " SD Card₩₩Run₩₩1.txt " )
smartConfigs1.StorageMountCheckStart(1000, 5, " 하드 디스크₩₩Run₩₩1.txt " )
```

메소드(함수) StorageMountCheckStop

외부저장장치 연결 확인을 정지시킵니다.

주의 반드시 StorageMountCheckStart() 메소드가 호출된 경우에만 호출하시기 바랍니다.

◉ `void StorageMountCheckStop()`

C# 사용법

```
smartConfigs1.StorageMountCheckStop();
```

VB 사용법

```
smartConfigs1.StorageMountCheckStop()
```

이벤트 OnStorageMounted, OnStorageMountCheckOverCount

- `OnStorageMounted` : StorageMountCheckStart() 메소드 호출 후 strStoragePath 인자값으로 설정된 외부저장장치의 파일에 접근 가능한 경우 발생하는 이벤트입니다.
- `OnStorageMountCheckOverCount` : StorageMountCheckStart() 메소드의 iCheckLimitCount 인자값으로 설정한 횟수동안 strStoragePath 인자값으로 설정된 외부저장장치의 파일에 접근하지 못한 경우 발생하는 이벤트입니다.

참고 "StorageMountCheckStart() 메소드" 내용을 참고하시기 바랍니다.

C# 사용법

```
// 외부저장장치의 특정 파일에 접근 가능한 경우 발생하는 이벤트
private void smartConfigs1_OnStorageMounted(object sender, EventArgs e)
{
  smartConfigs1.StorageMountCheckStop();
  MessageBox.Show( " Mounted " );
}
```

VB 사용법

```
Private Sub smartConfigs1_OnStorageMounted(ByVal sender As Object, ByVal e As EventArgs)
  smartConfigs1.StorageMountCheckStop()
  MessageBox.Show( " Mounted " )
End Sub
```

ControlPanel Component Interface		
메소드		
DateTimeSet() : bool (+1개 오버로드)	GetSystemDateTime() : DateTime	SetSystemDateTime(DateTime setDate Time): void
SyncDateTime() : bool (+1개 오버로드)	SystemMemorySet() : void (+1개 오버로드)	SystemVolumeSet() : bool (+1개 오버로드)
TouchCalibration() : bool (+1개 오버로드)		

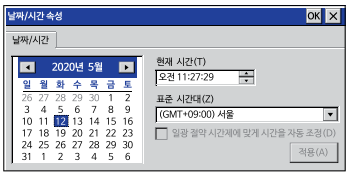
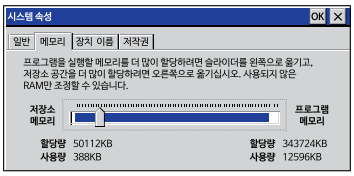
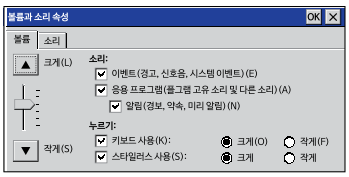
메소드(함수) `ControlPanel.DateTimeSet, ControlPanel.SystemMemorySet, ControlPanel.SystemVolumeSet, ControlPanel.TouchCalibration`

- `ControlPanel.DateTimeSet()` : 시스템 날짜 및 시간 설정창을 출력합니다.
- `ControlPanel.SystemMemorySet()` : 시스템 메모리 설정창을 출력합니다.
- `ControlPanel.SystemVolumeSet()` : 시스템 볼륨과 소리 설정창을 출력합니다.
- `ControlPanel.TouchCalibration()` : 터치 스크린 보정(Calibration) 화면을 출력합니다.

참고 "설정창을 모달 방식으로 띄울 경우 주의사항" 내용을 참고하시기 바랍니다.

참조 시스템 볼륨 관련하여 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 사운드 관련"을 참조하시기 바랍니다.

[표] 메소드 호출에 따른 출력 설정창

ControlPanel.DateTimeSet()	ControlPanel.SystemMemorySet()
	
ControlPanel.SystemVolumeSet()	**ControlPanel.TouchCalibration()**
	

- `void ControlPanel.DateTimeSet()`
- `void ControlPanel.DateTimeSet(int iTimeOutMilliseconds)`
- `void ControlPanel.SystemMemorySet()`
- `void ControlPanel.SystemMemorySet(int iTimeOutMilliseconds)`
- `void ControlPanel.SystemVolumeSet()`
- `void ControlPanel.SystemVolumeSet(int iTimeOutMilliseconds)`
- `void ControlPanel.TouchCalibration()`
- `void ControlPanel.TouchCalibration(int iTimeOutMilliseconds)`

[인자]

• int iTimeOutMilliseconds : 모달 방식으로 띄우는 경우 하단 코드의 대기 시간 (단위 : ms)
※ 0으로 설정할 경우 무기한 대기합니다.

C# 사용법

```
smartConfigs1.ControlPanel.DateTimeSet();       // 시스템 날짜 및 시간 설정창을 출력
smartConfigs1.ControlPanel.SystemMemorySet();   // 시스템 메모리 설정창을 출력
smartConfigs1.ControlPanel.SystemVolumeSet();   // 시스템 볼륨과 소리 설정창을 출력
smartConfigs1.ControlPanel.TouchCalibration();  // 터치 스크린 보정(Calibration) 화면을 출력
```

VB 사용법

```
smartConfigs1.ControlPanel.DateTimeSet()
smartConfigs1.ControlPanel.SystemMemorySet()
smartConfigs1.ControlPanel.SystemVolumeSet()
smartConfigs1.ControlPanel.TouchCalibration()
```

주의 설정창을 모달 방식으로 띄울 경우 주의사항

모달창을 띄우는 경우, 모달창의 부모 폼이 지정되어야 모달 기능이 정상 동작합니다. 하지만, SmartConfigs를 사용해 설정창을 모달 방식으로 띄우는 경우 부모 폼을 지정할 수 없습니다. 이러한 문제로 인해 설정창이 띄워진 상태에서 설정창 이외의 영역(예를 들어 버튼)을 터치하게 되면 설정창이 닫힐때 터치한 부분(버튼)이 실행되는 문제점이 있습니다. 따라서 모달 방식의 출력은 권장하지 않으며 아래 표에서 메소드에 따른 대안책을 확인하시기 바랍니다.

[표] Modal 방식으로 사용해야 하는 경우 대안 제시

메소드	Modal 방식으로 사용이 필요할 때 대안
ControlPanel.DateTimeSet(0)	HNS 홈페이지의 "자료실 → Tech Note → 15. [C#, VB.NET] 날짜/시간 설정 예제" 내용을 참조하여 사용하시기 바랍니다.
ControlPanel.SystemMemorySet(0)	SmartMemory를 참고하여 사용하시기 바랍니다.
ControlPanel.SystemVolumeSet(0)	SmartSound를 참고하여 사용하시기 바랍니다.
ControlPanel.TouchCalibration(0)	실행될 때 전체 화면으로 표시되기 때문에 이동이 불가하여 해당 문제점이 발생하지 않습니다.
Display.BacklightControlDialogBox(0)	SmartConfigs의 Display 네임 스페이스를 참고하여 사용하시기 바랍니다.

메소드(함수) `ControlPanel.GetSystemDateTime, ControlPanel.SetSystemDateTime, ControlPanel.SyncDateTime`

• ControlPanel.GetSystemDateTime() : 현재 시스템의 날짜 및 시간을 얻습니다.
• ControlPanel.SetSystemDateTime() : 현재 시스템의 날짜 및 시간을 설정합니다.
• ControlPanel.SyncDateTime() : 미리 지정된 타임 서버에서 시스템 날짜 및 시간을 동기화합니다.

참고 미리 지정된 타임 서버 리스트

SyncDateTime() 메소드 호출 시 인자값을 지정하지 않으면 미리 지정된 타임 서버 리스트의 순서대로 동기화를 시도합니다.

[미리 지정된 타임 서버 리스트]
1. time.windows.com
2. time.nist.gov
3. time.kriss.re.kr
4. time.google.com
5. ntp.postech.ac.kr

◉ DateTime ControlPanel.GetSystemDateTime()
◉ void ControlPanel.SetSystemDateTime(DateTime setDateTime)
◉ bool ControlPanel.SyncDateTime()
◉ bool ControlPanel.SyncDateTime(string strTimeserver)

[인자]
• DateTime setDateTime : 설정할 시스템의 날짜 및 시간
• string strTimeserver : 타임 서버 주소

[리턴값]
• DateTime : 현재 시스템의 날짜 및 시간
• bool : 설정할 시스템의 날짜 및 시간
  – true : 동기화 성공
  – false : 동기화 실패

C# 사용법

```
// InterNet Time Server DateTime 동기화, 시스템에서 제공하는 타임 서버 사용
smartConfigs1.ControlPanel.SyncDateTime();
// 또는 사용자가 원하는 타임 서버를 직접 입력하는 방식
smartConfigs1.ControlPanel.SyncDateTime( " nist1-macon.macon.ga.us " );
// DataTime의 시간을 설정하는 방법. 2021년 1월 15일 3시 7분 33초로 설정
DateTime date1 = new DateTime(2021, 1, 15, 3, 7, 33);
smartConfigs1.ControlPanel.SetSystemDateTime(date1);  // 시스템 시간을 설정
date1 = smartConfigs1.ControlPanel.GetSystemDateTime();  // 시스템 날짜 및 시간 얻기
```

VB 사용법

```
smartConfigs1.ControlPanel.SyncDateTime()
smartConfigs1.ControlPanel.SyncDateTime( " nist1-macon.macon.ga.us " )
Dim date1 As DateTime = New DateTime(2021, 1, 15, 3, 7, 33)
smartConfigs1.ControlPanel.SetSystemDateTime(date1)
date1 = smartConfigs1.ControlPanel.GetSystemDateTime()
```

CDisplay Component Interface		
속성		
BacklightOnOffStatus: CDisplay.OnOff	MouseCursor : CDisplay.OnOff	SystemFontClearType : bool
메소드		
BacklightControl(CDisplay.OnOff status) : void	BacklightControlDialogBox() : void (+1개 오버로드)	BacklightControlSetTime (int iTime) : void
MouseClick(int iXpos, int iYpos) : void	SetLCDRotation(CDisplay.ROT rotation) : void	SetLCDRotation_Save(CDisplay.ROT rotation) : void
Touch_MouseClickStart() : bool	Touch_MouseClickStop() : void	
이벤트		
ONTouch_MouseClick : CDisplay.Touch_MouseClick		

프로퍼티(속성) Display.BacklightOnOffStatus

IEC-Series의 LCD BackLight의 On/Off 상태를 확인합니다.

- CDisplay.OnOff : LCD Backlight의 상태(On/Off)
  - CDisplay.OnOff.OFF : LCD Backlight Off
  - CDisplay.OnOff.ON : LCD Backlight On

C# 사용법

```
if (smartConfigs1.Display.BacklightOnOffStatus == CDisplay.OnOff.ON)
{
  // LCD BackLight가 켜져있는 경우 실행할 코드
}
```

VB 사용법

```
If smartConfigs1.Display.BacklightOnOffStatus = CDisplay.OnOff.ON Then
  ' LCD BackLight가 켜져있는 경우 실행할 코드
End If
```

프로퍼티(속성) Display.MouseCursor

마우스 커서의 표시 여부를 설정합니다.

- CDisplay.OnOff : 마우스 커서의 표시 여부
  - CDisplay.OnOff.ON : 표시
  - CDisplay.OnOff.OFF : 숨김

C# 사용법

```
smartConfigs1.Display.MouseCursor = CDisplay.OnOff.OFF;  // 마우스 커서를 숨김
```

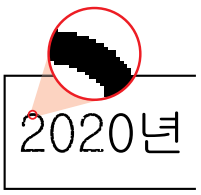
VB 사용법

```
smartConfigs1.Display.MouseCursor = CDisplay.OnOff.OFF
```

프로퍼티(속성) Display.SystemFontClearType

시스템상에 표현되는 모든 텍스트에 클리어 타입 폰트 적용 여부를 설정합니다.

- bool : 클리어 타입 폰트 적용 여부
  - true : 적용
  - false : 미적용 (기본값)

true	false
	

C# 사용법

```
// 클리어 타입 폰트 적용
smartConfigs1.Display.SystemFontClearType = true;
```

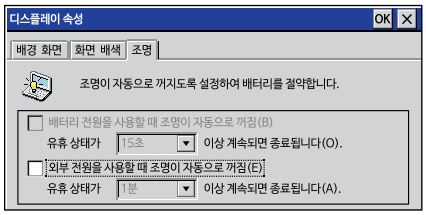
VB 사용법

```
smartConfigs1.Display.SystemFontClearType = True
```

메소드(함수) Display.BacklightControl, Display.BacklightControlDialogBox, Display.BacklightControlSetTime

- Display.BacklightControl() : LCD Backlight의 상태(On/Off)를 제어합니다.
- Display.BacklightControlDialogBox() : LCD Backlight의 상태(On/Off) 제어 설정창을 출력합니다.
- Display.BacklightControlSetTime() : LCD Backlight가 Off되는 시간을 설정합니다.

참고 "설정창을 모달 방식으로 띄울 경우 주의사항" 내용을 참고하시기 바랍니다.



◉ void Display.BacklightControl(CDisplay.OnOff status)
◉ void Display.BacklightControlDialogBox()
◉ void Display.BacklightControlDialogBox(int iTimeOutMilliseconds)
◉ void Display.BacklightControlSetTime(int iTime)

[인자]

- CDisplay.OnOff status : LCD Backlight의 상태(On/Off)
  - CDisplay.OnOff.OFF : LCD Backlight Off
  - CDisplay.OnOff.ON : LCD Backlight On
- int iTime : LCD Backlight가 Off되는 시간 (단위 : 초)
- int iTimeOutMilliseconds : 모달 방식으로 띄우는 경우 하단 코드의 대기 시간 (단위 : ms)

※ 0으로 설정할 경우 무기한 대기합니다.

C# 사용법

```
smartConfigs1.Display.BacklightControl(CDisplay.OnOff.OFF);  // LCD Backlight OFF
smartConfigs1.Display.BacklightControlDialogBox();  // 제어판의 조명 설정창 출력
smartConfigs1.Display.BacklightControlSetTime(30);  // LCD Backlight OFF 시간 설정
```

VB 사용법

```
smartConfigs1.Display.BacklightControl(CDisplay.OnOff.OFF)
smartConfigs1.Display.BacklightControlDialogBox()
smartConfigs1.Display.BacklightControlSetTime(30)
```

메소드(함수) Display.MouseClick, Display.Touch_MouseClickStart, Display.Touch_MouseClickStop

- Display.MouseClick() : 화면상의 특정 좌표(X, Y)를 클릭합니다.
- Display.Touch_MouseClickStart() : O/S 레벨에서 지원하는 기능으로, 화면의 모든 영역에서 터치 또는 마우스 클릭 감지를 시작합니다.(OnTouch.MouseClick Event 참고)
- Display.Touch_MouseClickStop() : 터치 또는 마우스 클릭 감지를 중지합니다.

주의 Touch_MouseClickStart(), Touch_MouseClickStop() 메소드는 IEC667/1000-Series에서만 지원됩니다. (IEC266-Series 미지원)

◉ void Display.MouseClick(int iXpos, int iYpos)
◉ bool Display.Touch_MouseClickStart()
◉ bool Display.Touch_MouseClickStop()

[인자]

- int iXpos : 화면상의 X좌표 (기준점 : 화면 좌측 상단)
- int iYpos : 화면상의 Y좌표 (기준점 : 화면 좌측 상단)

[리턴값]

• bool : 터치 또는 마우스 클릭 감지 시작/중지 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
smartConfigs1.Display.MouseClick(100, 100);  // (100, 100) 좌표를 강제 클릭
smartConfigs1.Display.Touch_MouseClickStart();  // 사용자 터치 또는 마우스 클릭 감지 시작
smartConfigs1.Display.Touch_MouseClickStop();  // 사용자 터치 또는 마우스 클릭 감지 중지
```

VB 사용법

```
smartConfigs1.Display.MouseClick(100, 100)
smartConfigs1.Display.Touch_MouseClickStart()
smartConfigs1.Display.Touch_MouseClickStop()
```

메소드(함수) Display.SetLCDRotation, Display.SetLCDRotation_Save

• Display.SetLCDRotation() : LCD를 회전시킵니다. (재부팅 시 회전 상태 초기화)
• Display.SetLCDRotation_Save() : LCD를 회전시킵니다. (재부팅 시 회전 상태 유지)

◉ void Display.SetLCDRotation(CDisplay.ROT rotation)
◉ void Display.SetLCDRotation_Save(CDisplay.ROT rotation)

[인자]

• CDisplay.ROT rotation : LCD 회전 각도
  – CDisplay.ROT.ROT0 : 0도 (기본값)
  – CDisplay.ROT.ROT90 : 90도
  – CDisplay.ROT.ROT180 : 180도
  – CDisplay.ROT.ROT270 : 270도

C# 사용법

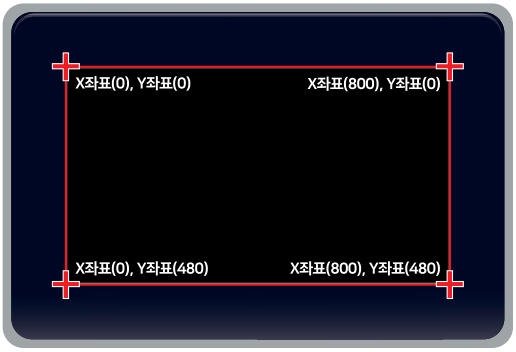
```
smartConfigs1.Display.SetLCDRotation(CDisplay.ROT.ROT90);  // LCD 90도 회전하기
smartConfigs1.Display.SetLCDRotation_Save(CDisplay.ROT.ROT90);  // LCD 90도 회전하고 저장
```

VB 사용법

```
smartConfigs1.Display.SetLCDRotation(CDisplay.ROT.ROT90)
smartConfigs1.Display.SetLCDRotation_Save(CDisplay.ROT.ROT90)
```

이벤트 ONTouch_MouseClick

Display.Touch_MouseClickStart() 메소드 호출 후 사용자가 터치 또는 마우스 클릭한 경우 발생하는 이벤트로 화면상의 사용자 터치 또는 마우스 클릭한 지점의 X좌표와 Y좌표를 인자값으로 전달받습니다.



※ LCD : IEC-07N/07/102 Series 기준입니다.

주의 1. 본 기능은 IEC667/1000에서만 지원됩니다. (IEC266에서 지원 안 함)
2. 이벤트 내의 코드가 길어질 경우 시스템이 느려질 수 있습니다.

◉ void CDisplay.Touch_MouseClick ONTouch_MouseClick(SmartConfigs.MOUSESTATECODE eMouse, int iXPos, int iYPos)

[인자]

- SmartConfigs.MOUSESTATECODE eMouse : 마우스 업/다운 상태
  - MOUSESTATECODE.MOUSE_DOWN : 마우스 다운
  - MOUSESTATECODE.MOUSE_UP : 마우스 업
- int iXpos : 화면상의 X좌표 (기준점 : 화면 좌측 상단)
- int iYpos : 화면상의 Y좌표 (기준점 : 화면 좌측 상단)

C# 사용법

```
private void smartConfigs1_ONTouch_MouseClick(SmartConfigs.MOUSESTATECODE eMouse, int iXPos, int iYPos)
{
  if (eMouse == SmartConfigs.MOUSESTATECODE.MOUSE_DOWN)
  {
    // 마우스 다운 상태일 때 처리 코드
  }
}
```

VB 사용법

```
Private Sub smartConfigs1_ONTouch_MouseClick(ByVal eMouse As SmartConfigs.MOUSESTATECODE, ByVal iXPos As System.Int32, ByVal iYPos As System.Int32) Handles smartConfigs1.ONTouch_MouseClick
  If eMouse = SmartConfigs.MOUSESTATECODE.MOUSE_DOWN Then
    ' 마우스 다운 상태일 때 처리 코드
  End If
End Sub
```

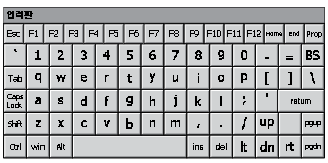
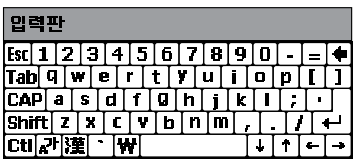
InputPanelCon Component Interface		
속성		
Current_KeyBoard : InputPanelCon.KEYBOARDMODE	XPosition : int	YPosition : int
메소드		
NumLockKey() : void		

프로퍼티(속성) InputPanelControl.Current_KeyBoard

사용할 시스템 가상 키보드(InputPanel)의 타입을 설정합니다.

권장 시스템 가상 키보드의 단점을 보완한 SmartKeyboard 또는 SmartKeyPad 사용을 권장합니다.

- InputPanelControl.KEYBOARDMODE : 시스템 가상 키보드의 타입
  - InputPanelControl.KEYBOARDMODE.LARGE : 키 버튼이 큰 타입
  - InputPanelControl.KEYBOARDMODE.SMALL : 키 버튼이 작은 타입

InputPanelControl.KEYBOARDMODE.LARGE	InputPanelControl.KEYBOARDMODE.SMALL
	

C# 사용법

```
// 큰 타입의 가상 키보드를 사용
smartConfigs1.InputPanelControl.Current_KeyBoard = InputPanelControl.KEYBOARDMODE.LARGE;
```

VB 사용법

```
smartConfigs1.InputPanelControl.Current_KeyBoard = InputPanelControl.KEYBOARDMODE.LARGE
```

프로퍼티(속성) InputPanelControl.XPosition, InputPanelControl.YPosition

활성화된 시스템 가상 키보드(InputPanel)의 위치를 설정합니다.

- InputPanelControl.XPosition : 가상 키보드의 X좌표를 설정합니다.
- InputPanelControl.YPosition : 가상 키보드의 Y좌표를 설정합니다.
- int : 시스템 가상 키보드의 위치 (기준점 : LCD 좌측 상단)

권장 시스템 가상 키보드의 단점을 보완한 SmartKeyboard 또는 SmartKeyPad 사용을 권장합니다.

C# 사용법

```
// 가상 키보드(InputPanel)의 X좌표를 설정
smartConfigs1.InputPanelControl.XPosition = 30;
// 가상 키보드(InputPanel)의 Y좌표를 설정
smartConfigs1.InputPanelControl.YPosition = 70;
```

VB 사용법

```
smartConfigs1.InputPanelControl.XPosition = 30
smartConfigs1.InputPanelControl.YPosition = 70
```

메소드(함수) InputPanelControl.NumLockKey

장치에 USB 키보드 연결 시 NumLock의 상태를 On/Off로 토글합니다. USB 키보드 NumLock의 초기 상태는 Off이며, 프로그램상에서 현재 NumLock 상태를 알고자 하는 경우에는 프로그램상에서 Flag를 만들어서 처리해야 합니다.

◉ void InputPanelControl.NumLockKey()

C# 사용법

```
// NumLock의 상태 토글
smartConfigs1.InputPanelControl.NumLockKey();
```

VB 사용법

```
smartConfigs1.InputPanelControl.NumLockKey()
```

CIPSetting Component Interface		
속성		
DeviceIP : string	DHCPEnable : int	GateWay : string
PrimaryDNS : string	PrimaryWINS : string	SecondaryDNS : string
SecondaryWINS : string	SubNetMask : string	
메소드		
GetMACAddress() : byte[]	Ping(string strIPAddress, ref CIPSetting.PINGRESULTINFO pingInfos) : bool	PingIP(string strIPAddress, ref CIPSetting.PINGRESULTINFO pingInfos) : bool
Save() : void	SetApply() : void	SetMACAddress(byte[] baMACAddress) : bool

프로퍼티(속성) `IPSettings.DeviceIP, IPSettings.GateWay, IPSettings.SubNetMask, IPSettings.PrimaryDNS, IPSettings.SecondaryDNS, IPSettings.PrimaryWINS, IPSettings.SecondaryWINS`

IEC-Series의 네트워크 관련 설정값을 설정하거나 확인합니다.(유선 LAN)

- `IPSettings.DeviceIP` : IEC-Series의 IP 주소를 설정하거나 확인합니다.
- `IPSettings.GateWay` : IEC-Series의 GateWay 주소를 설정하거나 확인합니다.
- `IPSettings.SubNetMask` : IEC-Series의 SubNetMask 주소를 설정하거나 확인합니다.
- `IPSettings.PrimaryDNS` : IEC-Series의 PrimaryDNS 주소를 설정하거나 확인합니다.
- `IPSettings.SecondaryDNS` : IEC-Series의 SecondaryDNS 주소를 설정하거나 확인합니다.
- `IPSettings.PrimaryWINS` : IEC-Series의 PrimaryWINS 주소를 설정하거나 확인합니다.
- `IPSettings.SecondaryWINS` : IEC-Series의 SecondaryWINS 주소를 설정하거나 확인합니다.

주의 IEC-Series의 IP 주소를 임의로 변경할 경우 주의사항

IP 주소체계는 1Byte(8Bit) 크기의 Octet 4개가 모여 총 4Byte(32Bit) 크기로 구성됩니다. 따라서 각 Octet에는 0~255 까지의 범위를 갖게 되며, IP 주소 변경 시 해당 범위로 설정하지 않는다면 IP가 올바르게 설정되지 않으며, 네트워크 드라이버가 비활성화 되는 문제가 발생합니다. 아래 표에서 올바른 사용법을 확인하시기 바랍니다.

192	.	168	.	0	.	1
Octet1	구분자	Octet2	구분자	Octet3	구분자	Octet4
8Bit		8Bit		8Bit		8Bit

[IP 주소의 구성 및 각 Octet의 크기]

[표] 올바른 IP 주소 설정과 잘못된 IP 주소 설정 비교

올바른 IP 주소 설정		잘못된 IP 주소 설정	
IP 주소	설명	IP 주소	설명
192.168.0.1	올바른 IP 주소	256.32.11.0	Octet1값이 범위를 벗어남
127.0.0.1		0	형식에 맞지 않음
164.0.0.1		0.0.0.0	올바른 IP가 아님

[문제 발생 시 해결 방법]

방법-1. 각 속성의 IP 주소를 위 표를 참고하여 올바르게 설정합니다

방법-2. 제품을 초기화합니다.

참조 제품 초기화 방법은 "홈페이지 → 커뮤니티 → 자주하는 질문 → 10. IEC-Series 초기화 방법 및 부팅 시간 안내" 내용을 참조하시기 바랍니다.

중요 IEC-Series의 네트워크 설정값을 임의로 변경 시 중요사항

복수 개의 IEC-Series가 네트워크(공유기, 허브 등)에 연결되는 경우, 반드시 각 IEC-Series의 MAC Address를 서로 겹치지 않게 설정하시기 바랍니다.(제품별 MAC 주소는 제품 커버에 부착되어 있습니다.)

참고 IEC-Series의 네트워크 설정값을 임의로 변경하려면 IPSettings.DHCPEnable 속성값이 0(Static)이어야 합니다. "IPSettings.DHCPEnable 속성" 내용을 참고하시기 바랍니다.

- `string` : 네트워크 설정값 (IPv4 체계)

C# 사용법

```
smartConfigs1.IPSettings.DHCPEnable = 0;  // IP를 고정 IP로 사용
// 아래 IP 주소는 예시
smartConfigs1.IPSettings.DeviceIP = "192.168.0.1";      // IEC-Series의 IP를 설정
smartConfigs1.IPSettings.GateWay = "192.168.1.1";       // IEC-Series의 GateWay를 설정
smartConfigs1.IPSettings.SubNetMask = "255.255.255.0"; // IEC-Series의 SubNetMask를 설정
smartConfigs1.IPSettings.PrimaryDNS = "8.8.8.8";        // IEC-Series의 PrimaryDNS를 설정
smartConfigs1.IPSettings.SecondaryDNS = "8.8.4.4";      // IEC-Series의 SecondaryDNS를 설정
smartConfigs1.IPSettings.PrimaryWINS = "127.0.0.1";     // IEC-Series의 PrimaryWINS를 설정
smartConfigs1.IPSettings.SecondaryWINS = "127.0.0.2";   // IEC-Series의 SecondaryWINS를 설정
```

VB 사용법

```
smartConfigs1.IPSettings.DHCPEnable = 0
smartConfigs1.IPSettings.DeviceIP = “192.168.0.1”
‘중략
smartConfigs1.IPSettings.SecondaryWINS = “127.0.0.2”
```

프로퍼티(속성) IPSettings.DHCPEnable

IEC-Series의 IP 할당 방식을 고정 또는 동적으로 설정합니다.

- int : IP 할당 방식
  - 0 : 고정 IP 사용 (Static)
  - 1 : 동적 IP 사용 (DHCP)

C# 사용법

```
smartConfigs1.IPSettings.DHCPEnable = 1;  // IP를 동적으로 할당받도록 설정
```

VB 사용법

```
smartConfigs1.IPSettings.DHCPEnable = 1
```

메소드(함수) IPSettings.GetMACAddress, IPSettings.SetMACAddress

- IPSettings.GetMACAddress() : 현재 IEC-Series에 설정된 MAC Address를 읽어옵니다.
- IPSettings.SetMACAddress() : MAC Address를 설정합니다. (재부팅 시 적용)

참고 MAC Address 앞 세자리 “60:DB:2A”는 HNS의 고유 번호로 고정되어 있습니다.

참조 SmartMAC Address Setting 프로그램을 사용하면 편리하게 MAC Address를 등록하실 수 있습니다.
“홈페이지 → 자료실 → 제품관련 → SmartMAC Address Setting”을 참조하시기 바랍니다.

◉ byte[] IPSettings.GetMACAddress()
◉ bool IPSettings.SetMACAddress(byte[] baMACAddress)

[인자]

- byte[] baMACAddress : MAC Address로 설정할 16진수 바이트 배열

[리턴값]

- byte[] baMACAddress : IEC-Series에 설정된 10진수 바이트 배열의 MAC Address
- bool : MAC Address 정상 설정 여부
  - true : 성공
  - false : 실패

C# 사용법

```
private void SetMAC()
{
  byte[] baSetMac = new byte[6];  // MAC Address 설정을 위한 바이트 배열
  baSetMac[0] = 0x60; baSetMac[1] = 0xDB; baSetMac[2] = 0x2A;  // 앞 3자리는 고정
  baSetMac[3] = 0x03; baSetMac[4] = 0xFB; baSetMac[5] = 0xA2;  // 뒤 3자리는 제품 커버 뒷면에 부착됨
  // MAC Address를 설정 후 정상 설정 시 제품 재부팅
  if (smartConfigs1.IPSettings.SetMACAddress(baSetMac) == true)
  { smartConfigs1.Powers.Reboot(); }
}
private void GetMAC()
{
  byte[] baGetMac = smartConfigs1.IPSettings.GetMACAddress();  // 설정된 MAC Address를 읽음
  // 읽어온 MAC Address를 SmartListBox에 출력
```

```
  for (int i = 0; i < baGetMac.Length; i++)
  {
    smartListBox1.AddItem(baGetMac[i].ToString());
  }
}
```

VB 사용법

```
Private Sub SetMAC()
  Dim baSetMac As Byte() = New Byte(5) {}
  baSetMac(0) = &H60 : baSetMac(1) = &HDB : baSetMac(2) = &H2A
  baSetMac(3) = &H3 : baSetMac(4) = &HFB : baSetMac(5) = &HA2
  If smartConfigs1.IPSettings.SetMACAddress(baSetMac) = True Then
    smartConfigs1.Powers.Reboot()
  End If
End Sub
Private Sub GetMAC()
  Dim baGetMac As Byte() = smartConfigs1.IPSettings.GetMACAddress()
  For i As Integer = 0 To baGetMac.Length  -  1
    smartListBox1.AddItem(baGetMac(i).ToString())
  Next
End Sub
```

메소드(함수) IPSettings.Ping, IPSettings.PingIP

• IPSettings.Ping() : IEC-Series가 네트워크를 통해 특정한 호스트(도메인 이름 또는 IP 주소)에 도달 가능 여부를 확인합니다.

• IPSettings.PingIP() : IEC-Series가 네트워크를 통해 특정한 호스트(IP 주소)에 도달 가능 여부를 확인합니다.

참고 Ping() 메소드와 PingIP() 메소드의 차이

Ping()	PingIP()
– 인자로 도메인 이름 또는 IP 주소를 사용 가능 – 도메인 네임 변환 작업으로 PingIP() 메소드에 비해 응답 속도가 느림	– 인자로 IP 주소만 사용 가능 – 도메인 네임 변환 작업 없이 다이렉트로 목적지 서버에 도달하므로 Ping() 메소드에 비해 응답 속도가 빠름

◉ bool IPSettings.Ping(string strIPAddress, ref CIPSetting.PINGRESULTINFO pingInfos)

◉ bool IPSettings.PingIP(string strIPAddress, ref CIPSetting.PINGRESULTINFO pingInfos)

[인자]

• string strIPAddress : 도메인 이름 또는 IP 주소

주의 IPSettings.PingIP() 메소드의 경우 반드시 IP 주소만을 입력해야 합니다.

• ref CIPSetting.PINGRESULTINFO pingInfos : Ping 통신의 결과를 저장받는 구조체
  – pingInfos.dwRoundTripTime : 데이터를 보내고 응답까지 소요되는 시간
  – pingInfos.iPacketByte : 패킷 사이즈
  – pingInfos.iTTL : 패킷이 통과할 수 있는 라우터(홉)의 개수

[리턴값]

• bool : Ping 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
CIPSetting.PINGRESULTINFO pingResultInfo = new CIPSetting.PINGRESULTINFO();
// 도메인 이름으로 Ping 통신 시도 및 성공에 따른 처리 작성
if (smartConfigs1.IPSettings.Ping( " google.co.kr " , ref pingResultInfo) == true) { }
```

```
// IP 주소로 Ping 통신 시도 및 성공에 따른 처리 작성
if (smartConfigs1.IPSettings.PingIP( " 192.168.1.1 " , ref pingResultInfo) == true) { }
```

VB 사용법

```
Dim pingResultInfo As CIPSetting.PINGRESULTINFO = New CIPSetting.PINGRESULTINFO()
If smartConfigs1.IPSettings.Ping( " google.co.kr " , pingResultInfo) = True Then
  ' Ping 통신이 성공한 경우
End If
If smartConfigs1.IPSettings.PingIP( " 192.168.1.1 " , pingResultInfo) = True Then
  ' Ping 통신이 성공한 경우
End If
```

메소드(함수) IPSettings.Save, IPSettings.SetApply

- IPSettings.Save() : IP 주소 설정값을 레지스트리에 저장합니다.
- IPSettings.SetApply() : Network Driver를 ReBind(셋팅값을 즉시 적용)합니다.

중요 IP 주소 관련 속성값을 변경한 경우 반드시 Save() 또는 SetApply() 메소드를 호출하시기 바랍니다.

◉ void IPSettings.Save()
◉ void IPSettings.SetApply()

C# 사용법

```
// 레지스트리에 저장. 시스템을 다시 시작 시 설정값 유지
smartConfigs1.IPSettings.Save();
// IP 주소값을 적용하기 위해 Network Driver를 Rebind 처리
smartConfigs1.IPSettings.SetApply();
```

VB 사용법

```
smartConfigs1.IPSettings.Save()
smartConfigs1.IPSettings.SetApply()
```

Power Component Interface		
메소드		
ReBoot() : void		

메소드(함수) Powers.ReBoot

IEC-Series를 재부팅(Rebooting)합니다.

◉ void Powers.ReBoot()

C# 사용법

```
smartConfigs1.Powers.ReBoot();
```

VB 사용법

```
smartConfigs1.Powers.ReBoot()
```

CWUSBIPSetting Component Interface		
속성		
DeviceIP : string	DHCPEnable : int	GateWay : string
PrimaryDNS : string	PrimaryWINS : string	SecondaryDNS : string
SecondaryWINS : string	SubNetMask : string	USBWirelessType : CWUSBIPSetting.USBWLANTYPE

메소드		
Save() : void	SetApply() : void	SetUSBWirelessLANConfig(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName) : void (+1개 오버로드)
SetUSBWirelessANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE) eUSBLANType, string strAPNetworkName, string strAPIPAddress : CWUSBIPSetting.APCONNECTSTATUS(+7개 오버로드)		

프로퍼티(속성) USBWirelessIPSettings.USBWirelessType

USB 무선랜(구형/신형)의 타입(구형/신형)을 설정합니다.

중요 USB 무선랜 관련 속성 설정 및 확인 전 반드시 USBWirelessIPSettings.USBWirelessType 속성값을 설정해야 합니다.

구형 USB 무선랜	신형 USB 무선랜
 RT2501	 RT2870

- CWUSBIPSetting.USBWLANTYPE : 사용 중인 USB 무선랜의 타입
  - CWUSBIPSetting.USBWLANTYPE.RT2870 : 신형 USB 무선랜
  - CWUSBIPSetting.USBWLANTYPE.RT2501 : 구형 USB 무선랜

C# 사용법

```
// 신형 USB 무선랜 사용 중
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870;
```

VB 사용법

```
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870
```

프로퍼티(속성) USBWirelessIPSettings.DeviceIP, USBWirelessIPSettings.GateWay, USBWirelessIPSettings.SubNetMask, USBWirelessIPSettings.PrimaryDNS, USBWirelessIPSettings.SecondaryDNS, USBWirelessIPSettings.PrimaryWINS, USBWirelessIPSettings.SecondaryWINS

USB 무선랜(구형/신형)의 네트워크 관련 설정값을 설정하거나 확인합니다.

- USBWirelessIPSettings.DeviceIP : USB 무선랜의 IP 주소를 설정하거나 확인합니다.
- USBWirelessIPSettings.GateWay : USB 무선랜의 GateWay 주소를 설정하거나 확인합니다.
- USBWirelessIPSettings.SubNetMask : USB 무선랜의 SunNetMask 주소를 설정하거나 확인합니다.
- USBWirelessIPSettings.PrimaryDNS : USB 무선랜의 PrimaryDNS 주소를 설정하거나 확인합니다.
- USBWirelessIPSettings.SecondaryDNS : USB 무선랜의 SecondaryDNS 주소를 설정하거나 확인합니다.
- USBWirelessIPSettings.PrimaryWINS : USB 무선랜의 PrimaryWINS 주소를 설정하거나 확인합니다.
- USBWirelessIPSettings.SecondaryWINS : USB 무선랜의 SecondaryWINS 주소를 설정하거나 확인합니다.

주의 IEC-Series의 IP 주소를 임의로 변경할 경우 주의사항

IP 주소체계는 1Byte(8Bit) 크기의 Octet 4개가 모여 총 4Byte(32Bit) 크기로 구성됩니다. 따라서 각 Octet에는 0~255까지의 범위를 갖게 되며, IP 주소 변경 시 해당 범위로 설정하지 않는다면 IP가 올바르게 설정되지 않으며, 네트워크 드라이버가 비활성화되는 문제가 발생합니다. 아래 표에서 올바른 사용법을 확인하시기 바랍니다.

192	.	168	.	0	.	1
Octet1	구분자	Octet2	구분자	Octet3	구분자	Octet4
8Bit		8Bit		8Bit		8Bit

[IP 주소의 구성 및 각 Octet의 크기]

[표] 올바른 IP 주소 설정과 잘못된 IP 주소 설정 비교

올바른 IP 주소 설정		잘못된 IP 주소 설정	
IP 주소	설명	IP 주소	설명
168.0.1	올바른 IP 주소	256.32.11.0	Octet1 값이 범위를 벗어남
127.0.0.1		0	형식에 맞지 않음
164.0.0.1		0.0.0.0	올바른 IP가 아님

[문제 발생 시 해결 방법]
방법-1. 각 속성의 IP 주소를 위 표를 참고하여 올바르게 설정합니다
방법-2. 제품을 초기화합니다.

참조 제품 초기화 방법은 "홈페이지 → 커뮤니티 → 자주하는 질문 → 10. IEC-Series 초기화 방법 및 부팅 시간 안내" 내용을 참조하시기 바랍니다.

주의 IEC266/266Lite - Series는 지원되지 않습니다.

중요 USB 무선랜의 네트워크 설정값을 임의로 변경 시 중요사항

USB 무선랜 관련 속성 설정 및 확인 전 반드시 USBWirelessIPSettings.USBWirelessType 속성값을 설정해야 합니다.

참고 무선랜(구형/신형)의 네트워크 설정값을 임의로 변경하려면 USBWirelessIPSettings.DHCPEnable 속성값이 0(Static)이어야 합니다. "USBWirelessIPSettings.DHCPEnable 속성" 내용을 참고하시기 바랍니다.

• `string` : 네트워크 주소 (IPv4 체계)

C# 사용법

```
// 신형 USB 무선랜 사용 중
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870;
smartConfigs1.USBWirelessIPSettings.DHCPEnable = 0;// IP를 고정 IP로 사용
// 아래 IP 주소는 예시
smartConfigs1.USBWirelessIPSettings.DeviceIP = "192.168.0.1";      // 무선랜의 IP를 설정
smartConfigs1.USBWirelessIPSettings.GateWay = "192.168.1.1";       // 무선랜의 GateWay를 설정
smartConfigs1.USBWirelessIPSettings.SubNetMask = "255.255.255.0"; // 무선랜의 SubNetMask를 설정
smartConfigs1.USBWirelessIPSettings.PrimaryDNS = "8.8.8.8";      // 무선랜의 PrimaryDNS를 설정
smartConfigs1.USBWirelessIPSettings.SecondaryDNS = "8.8.4.4";    // 무선랜의 SecondaryDNS를 설정
smartConfigs1.USBWirelessIPSettings.PrimaryWINS = "127.0.0.1";   // 무선랜의 PrimaryWINS를 설정
smartConfigs1.USBWirelessIPSettings.SecondaryWINS = "127.0.0.2"; // 무선랜의 SecondaryWINS를 설정
```

VB 사용법

```
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870
smartConfigs1.USBWirelessIPSettings.DHCPEnable = 0
smartConfigs1.USBWirelessIPSettings.DeviceIP = "192.168.0.1"
' 중략
smartConfigs1.USBWirelessIPSettings.SecondaryWINS = "127.0.0.2"
```

프로퍼티(속성) USBWirelessIPSettings.DHCPEnable

USB 무선랜(구형/신형)의 IP 할당 방식을 고정 또는 동적으로 설정합니다.

• `int` : IP 할당 방식
  - 0 : 고정 IP 사용 (Static) / - 1 : 동적 IP 사용 (DHCP)

C# 사용법

```
// 신형 USB 무선랜 사용 중
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870;
```

```
smartConfigs1.USBWirelessIPSettings.DHCPEnable = 1;  // IP를 동적으로 할당받도록 설정
```

VB 사용법

```
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870
smartConfigs1.USBWirelessIPSettings.DHCPEnable = 1
```

메소드(함수) USBWirelessIPSettings.Save, USBWirelessIPSettings.SetApply

- USBWirelessIPSettings.Save() : USB 무선랜의 IP 주소 설정값을 레지스트리에 저장합니다.
- USBWirelessIPSettings.SetApply() : USB 무선랜의 Network Driver를 ReBind(즉시 적용)합니다.

중요

1. USB 무선랜 관련 속성 설정 및 확인 전 반드시 USBWirelessIPSettings.USBWirelessType 속성값을 설정해야 합니다.
2. USB 무선랜의 IP 주소 관련 속성값을 변경한 경우 반드시 Save() 또는 SetApply() 메소드를 호출하시기 바랍니다.

◉ void USBWirelessIPSettings.Save()
◉ void USBWirelessIPSettings.SetApply()

C# 사용법

```
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870;
// 레지스트리에 저장하기 시스템을 다시 시작할 경우 설정된 IP 관련 주소 유지됨
smartConfigs1.USBWirelessIPSettings.Save();
// 변경된 IP 주소값을 적용하기 위해 Network Driver를 Rebind 처리
smartConfigs1.USBWirelessIPSettings.SetApply();
```

VB 사용법

```
smartConfigs1.USBWirelessIPSettings.USBWirelessType = CWUSBIPSetting.USBWLANTYPE.RT2870
smartConfigs1.USBWirelessIPSettings.Save()
smartConfigs1.USBWirelessIPSettings.SetApply()
```

메소드(함수) USBWirelessIPSettings.SetUSBWirelessLANConfig

USB 무선랜을 AP에 연결합니다. USB 무선랜의 타입에 따라 인자값이 다르며 사용하는 타입에 따라 인자값을 적용하여 사용하시기 바랍니다.

권장 USBWirelessIPSettings.SetUSBWirelessLANConfig_CheckRetry() 메소드 사용을 권장합니다.

주의 해당 메소드는 IEC266/266Lite – Series에서 지원되지 않습니다.

구형 USB 무선랜		신형 USB 무선랜
	RT2501	

◉ void SetUSBWirelessLANConfig(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName)
◉ void SetUSBWirelessLANConfig(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPNetworkKey)

[인자]

- CWUSBIPSetting.USBWLANTYPE eUSBWLANType : 사용하는 USB 무선랜 타입
  - CWUSBIPSetting.USBWLANTYPE.RT2870 : 신형 USB 무선랜
  - CWUSBIPSetting.USBWLANTYPE.RT2501 : 구형 USB 무선랜
- string strAPNetworkName : AP의 SSID
- string strAPNetworkKey : AP가 암호화된 경우 AP에 설정된 네트워크 키

C# 사용법

```
// 신형 USB 무선랜 사용 시 AP가 개방된 경우
smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig(CIPSetting.USBWLANTYPE.RT2870,
" testAP_FREE " );
// 신형 USB 무선랜 사용 시 AP가 암호화된 경우
smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig(CIPSetting.USBWLANTYPE.RT2870,
" testAP " , " 12345678 " );
```

VB 사용법

```
smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig(CIPSetting.USBWLANTYPE.RT2870,
" testAP_FREE " )
smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig(CIPSetting.USBWLANTYPE.RT2870,
" testAP " , " 12345678 " )
```

메소드(함수) **USBWirelessIPSettings.SetUSBWirelessLANConfig_CheckRetry**

USB 무선랜을 AP에 연결합니다. USB 무선랜의 타입에 따라 인자값이 다르며 사용하는 타입에 따라 인자값을 적용하여 사용하시기 바랍니다. SetUSBWirelessLANConfig와 같은 기능을 하지만 SetUSBWirelessLANConfig_CheckRetry() 함수는 AP와 연결이 되었는지 상태를 리턴하며 인자값에 따라 무선랜과 AP의 연결 재시도 처리를 하실 수 있습니다.

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPIPAddress);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPIPAddress, int iFailCheckCount);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPNetworkKey, string strAPIPAddress);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPIPAddress, int iRetryCount, int iRetryInterval);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPNetworkKey, string strAPIPAddress, int iFailCheckCount);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPIPAddress, int iRetryCount, int iFailCheckCount, int iRetryInterval);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPNetworkKey, string strAPIPAddress, int iRetryCount, int iRetryInterval);

◉ CWUSBIPSetting.APCONNECTSTATUS SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.USBWLANTYPE eUSBWLANType, string strAPNetworkName, string strAPNetworkKey, string strAPIPAddress, int iRetryCount, int iFailCheckCount, int iRetryInterval);

[인자]

- CWUSBIPSetting.USBWLANTYPE eUSBWLANType : 사용하는 USB 무선랜 타입
  - CWUSBIPSetting.USBWLANTYPE.RT2870 : 신형 USB 무선랜
  - CWUSBIPSetting.USBWLANTYPE.RT2501 : 구형 USB 무선랜
- string strAPNetworkName : AP의 SSID
- string strAPNetworkKey : AP가 암호화된 경우 AP에 설정된 네트워크 키
- string strAPIPAddress : AP의 할당된 IP 주소
- int iRetryCount : AP와 연결 실패 시 재시도 횟수(인자가 생략될 경우: 1회)
- int iFailCheckCount : AP와 연결 실패 시 연결 확인 횟수 – 한 번의 연결 실패 이후 연속적으로 몇 번의 연결 실패 시 리턴을 할지 설정(인자가 생략될 경우 : 3회)

• int iRetryInterval : AP와 연결 실패 시 재시도 간격 시간 – 단위 ms: 1000 → 1초 (인자가 생략될 경우: 700ms(0.7초))

[리턴값]

CWUSBIPSetting.APCONNECTSTATUS : AP와 무선랜의 연결 상태

– APCONNECTSTATUS.DISCONNECT : AP와 연결 안 됨

– APCONNECTSTATUS.CONNECTED : AP와 연결됨

C# 사용법

```
if (smartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.
USBWLANTYPE.RT2870, "HNS", "192.168.0.1", 1, 3, 1000) == CWUSBIPSetting.APCONNECTSTATUS
.CONNECTED)
{
  smartLabel1.Text = "AP Connection";
}
else
{
  smartLabel1.Text = "AP Disconnection";
}
```

VB 사용법

```
If SmartConfigs1.USBWirelessIPSettings.SetUSBWirelessLANConfig_CheckRetry(CWUSBIPSetting.
USBWLANTYPE.RT2870, "HNS", "192.168.0.1", 1, 3, 1000) = CWUSBIPSetting.APCONNECTSTATUS
.CONNECTED Then
  SmartLabel1.Text = "AP Connection"
Else
  SmartLabel1.Text = "AP Disconnection"
End If
```

4) SmartConfigs 예제 사용하기

SmartConfigs를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartConfigs"

4. SmartRemote

SmartRemote는 기존에 PC에서 사용하던 원격 제어(원격 데스크톱 연결) 솔루션을 IEC-Series에서 지원 가능하도록 HNS에서 개발한 솔루션입니다. PC에서는 SmartRemote-PC 프로그램을 사용해 IEC-Series와 연결할 수 있으며, 네트워크 또는 USB Windows Mobile Device Center를 통해 연결됩니다. IEC-Series 제품이 현장에서 무인으로 동작되는 경우 원격지에서 IEC-Series의 동작 상태 및 제어를 할 수 있어 현장에 직접 방문하지 않고 기기의 상태를 점검할 수 있도록 활용할 수 있습니다. 또한, SmartRemote는 컴포넌트로 구성 되어있어 개발자가 프로그램에 편리하게 적용하여 원격지원 시스템 운영을 하실 수 있습니다.

- 접속 계정에 따른 제어/모니터링 모드 설정 기능
- Network(유/무선 LAN) 뿐만 아니라 USB Windows Mobile Device Center로도 연결 지원
- IEC1000XGA - I 프로그램 개발 시, 개발 PC와 Windows Mobile Device Center로 연결되어 LCD 없이도 프로그램을 개발할 수 있도록 지원
- 별도 프로그램 구성 없이 개발 중인 프로젝트(응용 프로그램)에 쉽게 적용
- 1대의 원격지 컴퓨터에서 여러 대의 IEC-Series에 동시 접속/제어

중요 SmartRemote 사용을 위해서는 반드시 SmartRemotePC 프로그램을 같이 사용해야 합니다.

주의 SmartRemote 사용 시 주의사항

1. SmartRemote는 실시간 화면 갱신 및 제어 응답을 요구하는 시스템에서는 권장하지 않습니다.
2. 본 SmartRemote 솔루션은 다른 상업용 솔루션에 비해 성능(안전성, 호환성 등)에 많은 제한이 있으며, 이러한 부분을 고려하여 적용하시기 바랍니다.
3. Server와 연결 해제 후 재연결 시도 시 SmartRemotePC의 Connect를 바로 실행하지 말고, 약 2분 이상 Delay를 준 후 Connect를 시도하시기 바랍니다. Delay를 줘도 연결되지 않는 경우 Server의 SmartRemotePC 프로그램을 재시작하거나 IEC 제품의 재부팅이 필요합니다.
4. SmartRemote에서 90도 또는 270도로 화면이 회전된 모습의 캡쳐를 지원하지 않습니다. Visual Studio2008에서 지원하는 Visual Studio Remote Tools 원격 이미지 캡쳐 프로그램을 이용하시기 바랍니다.

1) SmartRemote 연결 방법

SmartRemote 연결 방법은 크게 유/무선 LAN으로 연결하는 방법과, Windows Mobile Device Center로 연결하는 방법의 총 2가지로 분류할 수 있으며, 세부적으로는 총 4가지로 분류할 수 있습니다. 아래 CASE를 확인하시기 바랍니다.

참조 SmartRemote 연결이 잘 안 되는 경우 참조 자료

1. LAN으로 연결하는 경우
"홈페이지 → 커뮤니티 → 자주하는 질문 → 32. 네트워크가 연결이 되지 않을 때 점검사항"

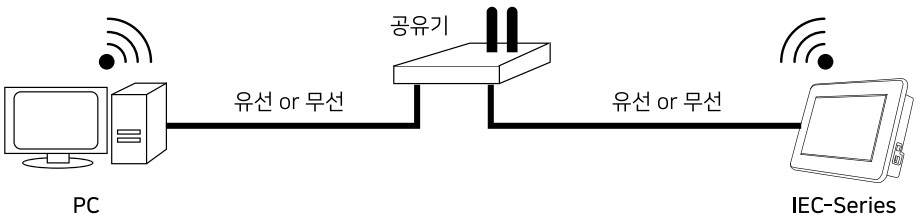
2. Windows Mobile Device Center로 연결하는 경우
"홈페이지 → 커뮤니티 → 자주하는 질문 → 28. Windows Mobile Device Center 연결이 안되는 경우 체크사항"

CASE-1 PC와 IEC-Series를 LAN 케이블을 이용하여 직접 연결(내부 네트워크)

참고 연결이 잘 안 되는 경우 확인사항

1. LAN 케이블이 크로스 케이블인지 확인합니다. PC와 IEC-Series를 LAN 케이블을 이용해 직접 연결하는 경우 LAN 케이블은 반드시 크로스 케이블이어야 합니다.
2. LAN 케이블이 단선되지는 않았는지, 올바르게 연결되었는지 확인합니다.(Ping 테스트 확인)

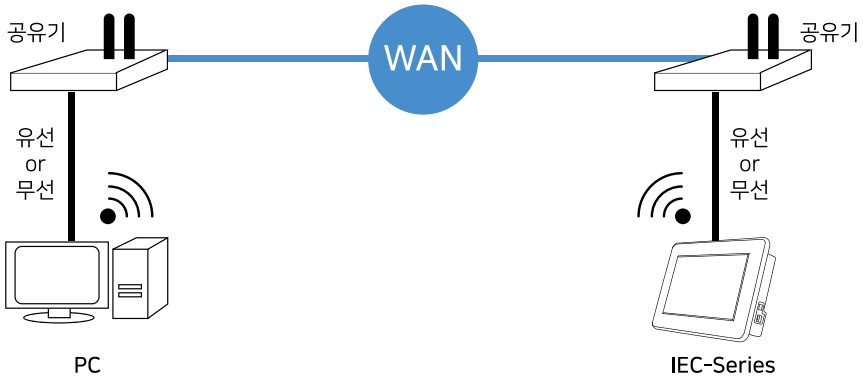
CASE-2 PC와 IEC-Series를 중간 장치(공유기)를 통하여 연결(내부 네트워크)



참고 연결이 잘 안 되는 경우 확인사항

1. LAN 케이블이 단선되지는 않았는지, 올바르게 연결되었는지 확인합니다.(Ping 테스트 확인)
2. IEC-Series의 MAC Address가 올바르게 설정되었는지 확인합니다.
3. IEC-Series의 IP가 유동인지 고정인지 확인하고, IP가 올바르게 설정되었는지 확인합니다.
4. 공유기의 설정이 올바른지 확인합니다.

CASE-3 PC와 IEC-Series를 WAN을 통하여 연결(외부 네트워크)



※ WAN을 통하여 PC와 IEC-Series를 연결하는 경우 연결/갱신 속도가 느릴 수 있습니다.

참고 연결이 잘 안 되는 경우 확인사항

1. LAN 케이블이 단선되지는 않았는지, 올바르게 연결되었는지 확인합니다.(Ping 테스트 확인)
2. IEC-Series의 MAC Address가 올바르게 설정되었는지 확인합니다.
3. IEC-Series의 IP가 유동인지 고정인지 확인하고, IP가 올바르게 설정되었는지 확인합니다.
4. 공유기의 설정이 올바른지 확인합니다.

CASE-4 PC와 IEC-Series를 Windows Mobile Device Center(USB)를 이용해 직접 연결

참고 연결이 잘 안 되는 경우 확인사항

1. Windows Mobile Device Center가 올바르게 연결되었는지 확인합니다.

2) 프로그래밍 적용 가이드

STEP-1 SmartRemote 시작하기

SmartRemote는 간단한 속성값만 설정하면 바로 사용하실 수 있습니다. AdminPassword, GuestPassword 속성으로 비밀번호를 설정하고, TCPPortNo 속성으로 통신 포트를 설정합니다. RefreshIntervalTime 속성으로 화면 갱신 간격을, ErrorMsgBoxShow 속성으로 에러 발생 시 출력 여부를 설정할 수 있습니다.
모든 속성 설정이 완료되면 Start() 메소드로 SmartRemote를 시작할 수 있으며, Stop() 메소드로 중지할 수 있습니다.

※ 자세한 내용은 “AdminPassword, GuestPassword, TCPPortNo, RefreshIntervalTime, ErrorMsgBoxShow 속성”, “Start(), Stop() 메소드”를 참고하시기 바랍니다.

```
// Admin 계정 Password 설정(반드시 5자리 숫자)
smartRemote1.AdminPassword = “ 12345 ”
// Guest 계정 Password 설정(반드시 5자리 숫자)
smartRemote1.GuestPassword = “ 54321 ”

// TCP 통신 Port 설정
smartRemote1.TCPPortNo = 7070;
// 화면 갱신 지연 시간 설정 (단위 : ms)
smartRemote1.RefreshIntervalTime = 1000;
// SmartRemote 사용 중 Error 발생 시 MessageBox 표시 여부를 설정
smartRemote1.ErrorMsgBoxShow = false;

// SmartRemote 시작
smartRemote1.Start();
// SmartRemote 중지
smartRemote1.Stop();
```

참고 SmartRemotePC프로그램 사용법은 “5) SmartRemotePC 프로그램 사용하기” 내용을 참고하시기 바랍니다.

3) SmartRemote 인터페이스 설명

SmartRemote Component Interface		
속성		
AdminPassword : string	ErrorMsgBoxShow : bool	GuestPassword : string
RefreshIntervalTime : int	TCPPortNo : int	
메소드		
IsConnect() : bool	Start() : void	Stop() : void
이벤트		
OnErrorMessage : string		

프로퍼티(속성) AdminPassword, GuestPassword

SmartRemote의 접속 계정 비밀번호를 설정합니다.

• AdminPassword : 관리자 계정 비밀번호를 설정합니다. 관리자 계정은 원격 제어 시 마우스와 키보드 입력을 할 수 있습니다.

• GuestPassword : 모니터링 계정 비밀번호를 설정합니다. 모니터링 계정은 원격 제어 시 마우스와 키보드 입력이 불가합니다.

중요 비밀번호의 입력 형식은 숫자이며 반드시 5자리여야 합니다.

• string : 비밀번호

C# 사용법

```
smartRemote1.AdminPassword = " 12345 "; // Admin 계정 Password 설정(반드시 5자리 숫자)
smartRemote1.GuestPassword = " 54321 "; // Guest 계정 Password 설정(반드시 5자리 숫자)
```

VB 사용법

```
smartRemote1.AdminPassword = " 12345 " : smartRemote1.GuestPassword = " 54321 "
```

프로퍼티(속성) ErrorMsgBoxShow

SmartRemote Server Module에서 Error 발생 시 MessageBox 출력 여부를 설정합니다.

• bool : MessageBox 출력 여부
  – true : MessageBox 출력
  – false : MessageBox 출력 안 함

C# 사용법

```
smartRemote1.ErrorMsgBoxShow = true; // Server에서 Error 발생 시 MessageBox 출력 여부를 설정
```

VB 사용법

```
smartRemote1.ErrorMsgBoxShow = True
```

프로퍼티(속성) RefreshIntervalTime

SmartRemote 접속 시 화면 갱신을 위한 데이터 전송 주기 시간을 설정합니다.

• int : 화면 갱신 시간 (단위 : ms, 기본값 : 1000)

C# 사용법

```
smartRemote1.RefreshIntervalTime = 1000; // 화면 갱신 지연 시간 설정, 1000ms = 1초
```

VB 사용법

```
smartRemote1.RefreshIntervalTime = 1000
```

프로퍼티(속성) TCPPortNo

SmartRemote의 네트워크 연결 시 사용될 Port 번호를 설정합니다.

- int : TCP Port 번호

C# 사용법

```
smartRemote1.TCPPortNo = 7070;  // TCP 통신 Port 설정
```

VB 사용법

```
smartRemote1.TCPPortNo = 7070
```

메소드(함수) IsConnect

SmartRemote와 원격 서버의 접속 상태를 확인합니다.

◉ bool IsConnect()

[리턴값]

- bool : 원격 접속 여부
  - true : 원격 접속된 상태
  - false : 원격 접속 안된 상태

C# 사용법

```
if (smartRemote1.IsConnect() == true)
{
  // 서버와 연결된 상태일 때 처리 코드 작성
}
```

VB 사용법

```
If smartRemote1.IsConnect() = True Then
End If
```

메소드(함수) Start, Stop

- Start() : SmartRemote를 시작합니다.
- Stop() : SmartRemote를 종료합니다.

◉ void Start()

◉ void Stop()

C# 사용법

```
smartRemote1.Start();  // SmartRemote 시작
smartRemote1.Stop();   // SmartRemote 종료
```

VB 사용법

```
smartRemote1.Start() : smartRemote1.Stop()
```

이벤트 OnErrorMessage

SmartRemote 동작 중 에러 발생 시 발생하는 이벤트입니다.

◉ OnErrorMessage(string strErrorMsg)

[인자]

- string strErrorMsg : 에러 메시지

C# 사용법

```
private void smartRemote1_OnErrorMessage(string strErrorMsg)
{
  // smartListBox에 현재 시각과 에러 메시지를 추가
  smartListBox1.AddItem( " [ " + DateTime.Now.ToString() + " ] " + strErrorMsg);
}
```

VB 사용법

```
Private Sub smartRemote1_OnErrorMessage(ByVal strErrorMsg As System.String) Handles
smartRemote1.OnErrorMessage
  smartListBox1.AddItem( " [ " + DateTime.Now.ToString() + " ] " + strErrorMsg)
End Sub
```

4) SmartRemote 예제 사용하기

SmartRemote를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartRemote"

5) SmartRemotePC 프로그램 사용하기

SmartRemotePC 프로그램은 PC에서 실행되는 프로그램으로 IEC-Series에서 동작하는 SmartRemote Server와 연결하여 제어 및 화면을 모니터링합니다. 아래 내용은 SmartRemotePC 프로그램의 사용 방법입니다.

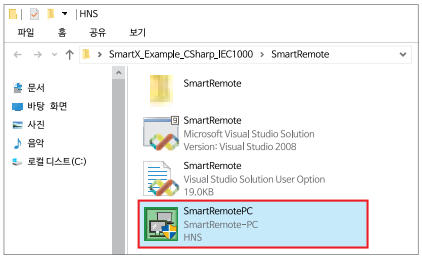
- ■ 다양한 연결 방식 지원
  - ↳ 유선 LAN(Ethernet), 무선 LAN(Option USB Type), USB(Windows Mobile Device Center)
- ■ 자주 사용하는 IP Address 등록 기능 지원
- ■ HNS Logo와 회사 정보 숨김 기능 지원

참고 IEC266/266Lite-Series의 경우 하드웨어 스펙상 원활한 연결이 힘들 수 있습니다.

5-1) SmartRemotePC 위치 및 실행

[프로그램 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제파일 다운로드 → "SmartRemote"
(별도 설치 없이 바로 실행 사용 가능)



주의 SmartRemotePC 프로그램은 반드시 관리자 권한으로 실행해야 합니다.

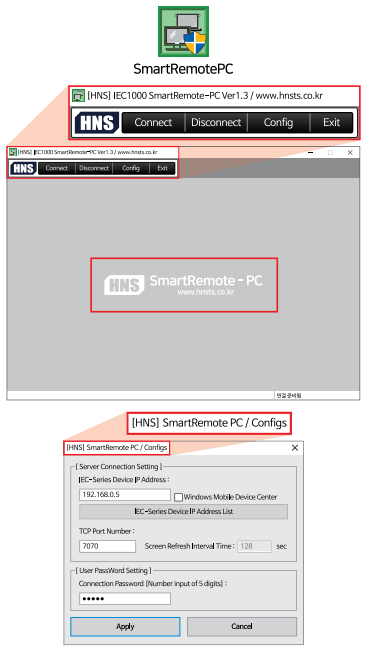
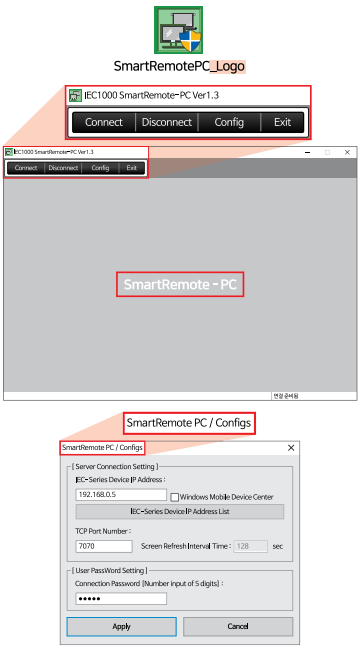
참고 SmartRemotePC 프로그램에서 HNS 로고 제거하기

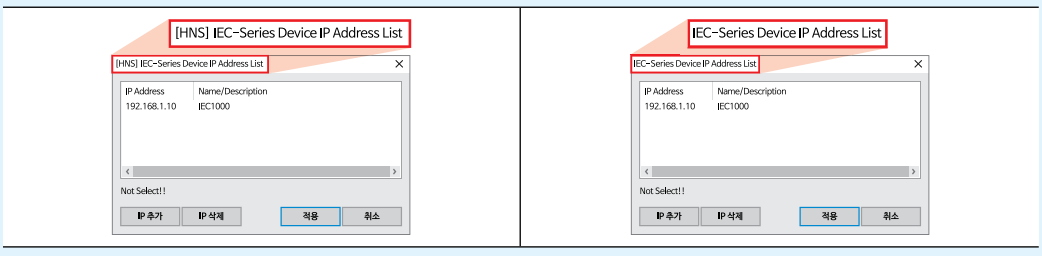
SmartRemotePC 프로그램에는 HNS의 로고와 홈페이지 주소가 표시되어 있습니다.
로고가 불필요하신 경우 SmartRemotePC 프로그램의 파일명을 변경하면 로고를 제거할 수 있습니다.

[표] SmartRemotePC 파일명 변경예시에 따른 HNS 로고 및 홈페이지 주소 표시 여부

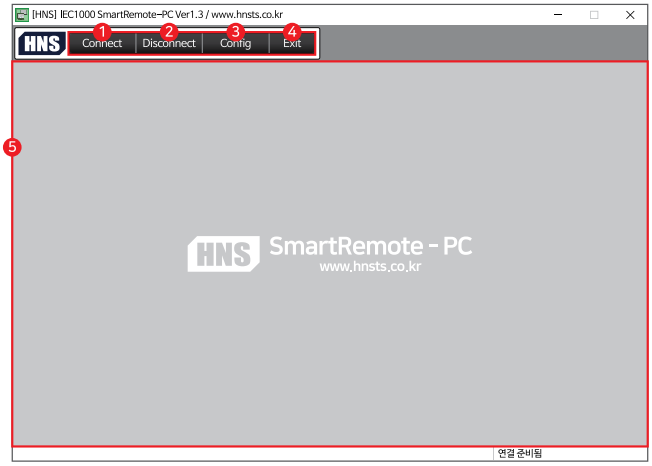
파일명 변경 예시	HNS 로고 및 홈페이지 주소 표시 여부
SmartRemotePC	표시됨
smartremotepc	표시 안 됨
SmartRemotePC_Logo	표시 안 됨
SmartRemote	표시 안 됨
RemoteView	표시 안 됨

※ 파일명은 "SmartRemotePC"가 기준으로, 파일명이 다른 경우 HNS 로고가 제거됩니다. (대소문자 구분)

[파일명 변경 전]	[파일명 변경 후]
	

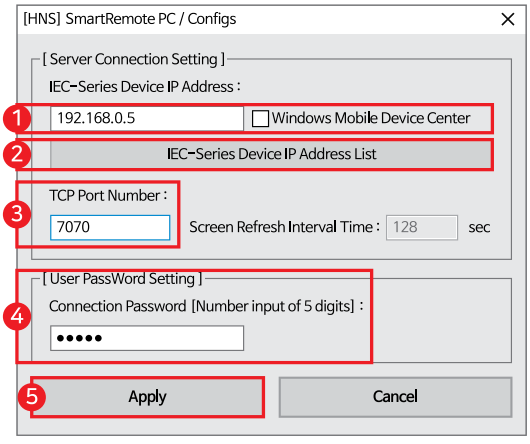


5-2) UI 설명



1. [Config]에서 설정된 정보에 따라 IEC-Series로 연결을 요청합니다.
2. 원격 연결이 되어있을 경우, 원격 연결을 해제합니다.
3. 원격으로 연결할 IEC-Series의 정보를 설정합니다. 반드시 [Connect] 버튼 클릭 전에 설정합니다.
4. 프로그램을 종료합니다.
5. IEC-Series와 연결 성공 시 IEC-Series의 화면이 출력됩니다.

5-3) 연결할 IEC-Series 정보 설정

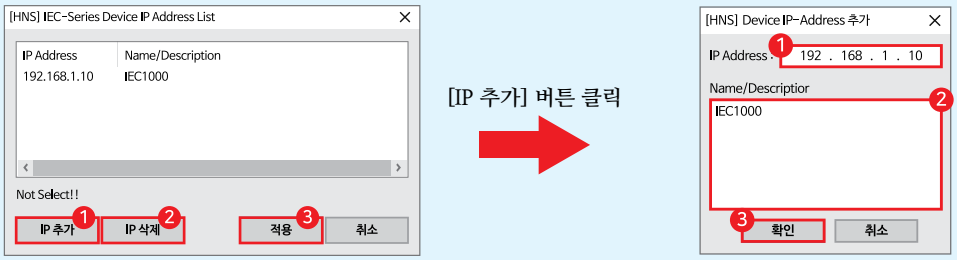


1. 연결할 IEC-Series의 IP 정보를 입력합니다. USB(Windows Mobile Device Center)를 사용하여 연결하는 경우 Windows Mobile Device Center 체크 박스를 선택합니다.
2. 자주 사용하는 IEC-Series의 IP 주소를 저장할 수 있습니다.
3. TCP Port 번호를 설정합니다. 만약 Mobile Device Center 연결 시 Port 번호를 변경하시면 시스템을 다시 시작하시기 바랍니다.
4. IEC-Series에서 SmartRemote 속성 AdminPassword, GuestPassword 값으로 설정한 값을 입력합니다. 반드시 5자리의 숫자를 입력해야 하며, 속성값으로 설정한 값과 다를 경우 접속이 되지 않습니다.

참고 SmartRemote의 "AdminPassword, GuestPassword 속성"을 참고하시기 바랍니다.

5. 설정 사항을 저장합니다.

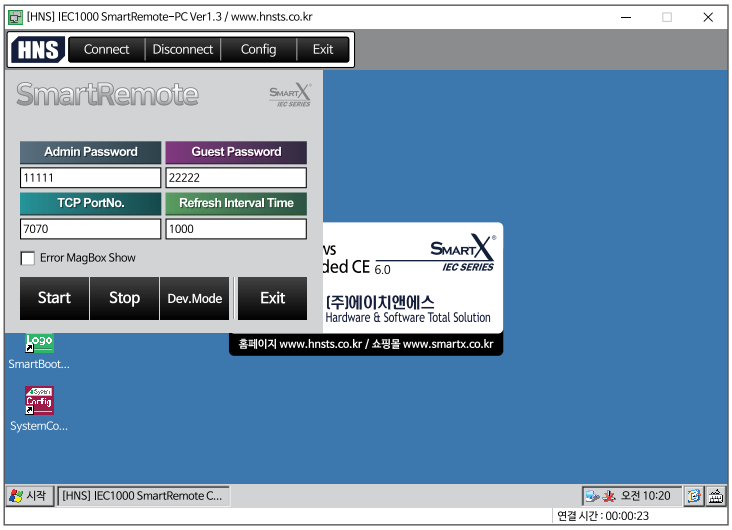
참고 IEC-Series 원격 접속 IP 주소 관리



1. IP 리스트에 IP를 추가합니다.
2. 선택한 IP를 목록에서 삭제합니다.
3. 선택한 IP를 설정값으로 적용합니다.

1. 추가할 IP를 입력합니다.
2. IP의 설명을 입력합니다.
3. IP를 리스트에 추가합니다.

5-4) IEC-Series 와 연결 성공된 화면



참고 SmartRemotePC를 사용하여 장비와 연결 후 제어 및 화면이 이상하게 출력되는 경우 확인사항

1. 공유기를 사용하는 경우, 공유기 사용없이 다이렉트로 연결하여 테스트가 필요합니다.
2. 무선랜 사용 시 무선랜의 연결 상태가 정상인지 체크해야 합니다.
3. PC와 IEC-Series의 시스템 상태가 정상인지 체크해야 합니다.

5. SmartTimer

.NET Compact Framework에서 지원되는 Timer 컴포넌트에서 내부에 고 정밀 Timer Counter를 내장하여 프로세서의 독립적인 Counter를 할 수 있도록 하였습니다. 아래의 특징과 같은 기능을 확장하여 프로그램 개발 시 SmartTimer의 활용도를 높여 편리하게 적용할 수 있도록 만들어진 컴포넌트입니다.

- Counter 오차 발생하지 않도록 내부 Timer Counter 내장
- Up / Down Timer Counter 모드 지원
- 고 정밀 1/1000sec Timer Counter 표현
- 현재 Timer Counter 값의 편리한 형식(Format) 출력 지원
- 편리한 Timer 시간 설정 변경 기능
- 여러 개의 Interval 설정에 따른 이벤트 발생 및 식별 기능
- 일시 중지(Pause) 기능
- Finish 이벤트 기능

[표] 다음과 같은 형식의 Timer를 표시할 경우 처리 방법 비교

표현 형식	00(분):00(초):000(밀리초) (Up Counter)	
처리 방법	Timer 사용 시	SmartTimer 사용 시
구현 방법	Timer Tick 이벤트에서 1초씩 증가하여 구현	SmartTimer의 내부 고 정밀 Timer Counter를 사용해 구현

예제 코드	`private int m_iSec = 0;  // 초를 표현할 변수`
	`private int m_iMin = 0;  // 분을 표현할 변수`
	`// Timer 초기 설정 및 시작`
	`private void StartTimer()`
	`{`
	`  // Tick 이벤트 발생 간격을 1초로 설정`
	`  timer1.Interval = 1000;`
	`  // Timer 시작`
	`  timer1.Enabled = true;`
	`}`
	`// 1초마다 발생하는 Tick 이벤트`
	`private void timer1_Tick(object sender, Event Args e)`
	`{`
	`  // 60초가 되면 분을 증가`
	`  if (++m_iSec == 60)`
	`  {`
	`  m_iSec = 0;`
	`  m_iMin++;`
	`  }`
	`  // 라벨에 현재 시간을 표현`
	`  // Timer의 경우 밀리초 표현이 안 됨`
	`  labelTimer.Text = m_iMin.ToString( " 00 " ) +`
	`" : " + m_iSec.ToString( " 00 " );`
	`}`
		`// SmartTimer 초기 설정 및 시작`
		`private void StartTimer()`
		`{`
		`  // SmartTimer를 Up Counter로 설정`
		`  smartTimer1.StartTime = 0;`
		`  smartTimer1.EndTime = 1000000000000;`
		`  // Tick 이벤트 발생 간격을 1초로 설정`
		`  smartTimer1.Interval = 1000;`
		`  // SmartTimer 시작`
		`  smartTimer1.Start();`
		`}`
		`// 1초마다 발생하는 Tick 이벤트`
		`private void smartTimer1_Tick(object sender, Event Args e)`
		`{`
		`  // 라벨에 현재 시간을 표현`
		`  // SmartTimer의 경우 밀리초 표현이 가능`
		`  labelSmartTimer.Text = smartTimer1.GetNowTim eFormatString(SmartX.SmartTimer.FORMATTYPENO. M_S_MS);`
		`}`

비교 항목	1. CPU 가용 상태에 따라서 오차 발생 2. 1/1000초 표현 불가	1. 오차 발생하지 않음 2. 1/1000초 표현 가능

1) 프로그래밍 적용 가이드

CASE-1 Timer 모드로 사용하기

[STEP-1] 기본 설정 및 시작하기

SmartTimer는 Up Timer와 Down Timer 총 두 가지의 동작 모드를 지원합니다. StartTime 속성값이 EndTime 속성값보다 작으면 시간 측정 시 Up Timer로 동작하며, 반대로 StartTime 속성값이 EndTime 속성값보다 크면 Down Timer로 동작합니다. 동작 모드 설정이 완료되면 Interval 속성값으로 Tick 이벤트 발생 간격을 설정합니다. 모든 설정이 완료되면 Start() 메소드를 호출해 SmartTimer를 시작할 수 있으며, Stop() 메소드를 호출해 SmartTimer를 종료할 수 있습니다.

※ 자세한 내용은 “StartTime, EndTime, Interval 속성”, Start(), Stop() 메소드”를 참고하시기 바랍니다.

```
smartTimer1.StartTime = 0; smartTimer1.EndTime = 60000;  // Up Timer로 사용하기
smartTimer1.StartTime = 60000; smartTimer1.EndTime = 0;  // Down Timer로 사용하기
smartTimer1.Interval = 500;  // Tick 이벤트 발생 간격 설정하기
smartTimer1.Start();  // SmartTimer 시작
```

[STEP-2] Tick 이벤트 사용하기

SmartTimer 시작 후 Interval 속성에서 설정한 시간마다 Tick 이벤트가 발생합니다. Tick 이벤트 코드 내에서 경과 시간을 NowFormatTime, NowFormatTimes, NowIncMillisecond, NowMillisecond 속성과 GetNowTimeFormatString() 메소드를 사용해 확인할 수 있습니다. 또한, STEP-1에서 설정한 동작 모드가 Up Timer인 경우 StartTime부터 증가한 시간을 얻으며, Down Timer인 경우 StartTime에서 감소한 시간을 얻습니다.

※ 자세한 내용은 “NowFormatTime, NowFormatTimes, NowIncMillisecond, NowMillisecond 속성”, “GetNowTimeFormatString() 메소드”, “Tick 이벤트”를 참고하시기 바랍니다.

```
// 주기적으로 처리해야하는 코드를 작성
private void smartTimer1_Tick(object sender, EventArgs e)
{
  // 경과된 시간을 ms로 표시
  lblNowms.Text = smartTimer1.NowMillisecond.ToString();
  // 경과된 시간을 자릿수 형식으로 표시
  lblNowFormat.Text = smartTimer1.GetNowTimeFormatString(SmartX.SmartTimer.FORMATTYPENO.H_M_S_MS);
}
```

CASE-2 특정 구간의 작업 소요 시간 확인하기

SmartTimer를 사용해 특정 구간의 작업 소요 시간을 확인할 수 있습니다. StartWatch() 메소드로 작업 시작 구간을 설정하고, StopWatch() 메소드로 작업 종료 구간을 설정합니다. StopWatch() 메소드 호출 후 StopWatchElaspedMicrosecond 속성을 확인하여 작업 소요 시간을 확인할 수 있습니다.

※ 자세한 내용은 “StopWatchElaspedMicrosecond 속성”, “StartWatch(), StopWatch() 메소드”를 참고하시기 바랍니다.

```
smartTimer1.StartWatch();  // 작업 시작 구간 설정
// 100개의 라인을 그리는 시간을 체크
for(int i=0; i < 100 ; i++)
{
  smartDraw1.Line(i + 50, i + 50, i + 100, i + 100);
}
smartTimer1.StopWatch(); // 작업 종료 구간 설정
smartLabel1.Text = smartTimer1.StopWatchElapsedMicrosecond.ToString();  // 작업 소요 시간 확인
```

2) SmartTimer 인터페이스 설명

SmartTimer Component Interface		
속성		
EndTime : long	Interval : int	IntervalSeries : int[]
IsStart : bool	NowFormatTime : long	NowFormatTimes : SmartTimer.TIMESFORMAT
NowIncMillisecond : long	NowMillisecond : long	StartTime : long
StopWatchElaspedMicrosecond : double		
메소드		
GetNowTimeFormatString(SmartTimer.FORMATTYPENO formatNo) : string	GetTimeFormatString(long dwTotMilliSecond) : SmartTimer.TIMESFORMAT (+1개 오버로드)	Pause() : void
SetNowMillisecondDec(SmartTimer.SETNOWTIMES setField, long dwMinMillisecond) : bool (+1개 오버로드)	SetNowMillisecondInc(SmartTimer.SETNOWTIMES setField, long dwMaxMilliSecond) : bool (+1개 오버로드)	Start() : void
StartWatch() : void	Stop() : void	StopWatch() : void
이벤트		
OnFinishTick : EventHandler	OnIntervalSeriesTick : SmartTimer.EventHandler	Tick : EventHandler

프로퍼티(속성) StartTime, EndTime

- `StartTime` : SmartTimer의 시작 시간을 설정합니다.
- `EndTime` : SmartTimer의 종료 시간을 설정합니다.

중요 SmartTimer 동작 모드 설정

StartTime 속성값이 EndTime 속성값보다 작으면 시간 측정 시 Up Timer로 동작하며, 반대로 StartTime 속성값이 EndTime 속성값보다 크면 Down Timer로 동작합니다.

[표] 속성값에 따른 SmartTimer 동작 모드

속성값	StartTime < EndTime	StartTime > EndTime
동작 모드	Up Timer	Down Timer
동작 예시	StartTime = 0 EndTime = 100 0에서 100까지 증가함	StartTime = 100 EndTime = 0 100에서 0까지 감소함

참고 최대 설정 시간은 Long형 타입의 크기인 2의 63승(9,223,372,036,854,780,000)이며, 시간으로 환산 시 292471208년(2억 9천만년)에 이릅니다.

- `long` : SmartTimer의 시작 또는 종료 시간 (단위 : ms)

C# 사용법

```
// Down-Timer Setting. 3분 (180000 / 1000 / 60 = 3)
smartTimer1.StartTime = 180000; smartTimer1.EndTime = 0;
// Up-Timer Setting
smartTimer1.StartTime = 0; smartTimer1.EndTime = 180000;
```

VB 사용법

```
smartTimer1.StartTime = 180000 : smartTimer1.EndTime = 0
smartTimer1.StartTime = 0 : smartTimer1.EndTime = 180000
```

프로퍼티(속성) Interval

Tick 이벤트 발생 간격을 설정합니다.

참고 "Tick 이벤트" 내용을 참고하시기 바랍니다.

- `int` : Tick 이벤트 발생 간격 (단위 : ms)

C# 사용법

```
smartTimer1.Interval = 300;  // 300ms가 경과될 때마다 Tick 이벤트를 발생시킴
```

VB 사용법

```
smartTimer1.Interval = 300
```

프로퍼티(속성) IntervalSeries

OnIntervalSeriesTick 이벤트 발생 간격을 설정합니다. Interval 속성 및 Tick 이벤트와 달리 배열을 사용해 발생 간격의 가변이 가능합니다.

참고 "OnIntervalSeriesTick 이벤트" 내용을 참고하시기 바랍니다.

- `int[]` : Tick 이벤트 발생 간격 (단위 : ms)

C# 사용법

```
// 다음의 패턴으로 smartTimer1_OnIntervalSeriesTick 이벤트가 발생되도록 설정
// 500ms → 3000ms → 1000ms → 500ms → 3000ms → 1000ms…(반복)
int[] iIntervalArray = new int[3];
iIntervalArray[0] = 500; iIntervalArray[1] = 3000; iIntervalArray[2] = 1000;
smartTimer1.IntervalSeries = iIntervalArray;
```

VB 사용법

```
Dim iIntervalArray() As Integer = New Integer(2) {}
iIntervalArray(0) = 500 : iIntervalArray(1) = 3000 : iIntervalArray(2) = 1000
smartTimer1.IntervalSeries = iIntervalArray
```

프로퍼티(속성) IsStart

SmartTimer의 동작 상태를 확인합니다.

- `bool` : SmartTimer 동작 상태
  - `true` : 시작됨
  - `false` : 시작되지 않음

C# 사용법

```
if (smartTimer1.IsStart == true
{
  // SmartTimer가 동작 중일 때 처리 코드를 작성
}
```

VB 사용법

```
If smartTimer1.IsStart = True Then
  ' SmartTimer가 동작 중일 때 처리 코드를 작성
End If
```

프로퍼티(속성) NowFormatTime, NowFormatTimes, NowIncMillisecond, NowMillisecond

Start() 메소드 호출 후 경과된 시간을 얻습니다.

- `NowFormatTime` : 자릿수를 가지고 있는 형식의 값을 얻습니다. (예 : 121000 -> 1분 21초 0밀리초)
- `NowFormatTimes` : 경과된 시, 분, 초, 밀리초를 구조체 형식으로 얻습니다.

- NowIncMillisecond : SmartTimer의 동작 모드와 관계없이 경과된 시간을 얻습니다.
- NowMillisecond : 경과된 시간을 ms로 얻습니다. (예 : 121000 -> 2분 1초 0밀리초)

참고 동작 모드에 따른 경과 시간 형식

SmartTimer의 동작 모드에 따라 시간 형식에 차이가 있으므로, 아래 표를 참고하시기 바랍니다.

[표] SmartTimer 동작 모드에 따른 속성값 예시

동작 모드	Up Timer	Down Timer
StartTime	0	3600000 (ms)
EndTime	3600000 (ms)	0
경과 시간	135487 (2분 15초 487밀리초)	
NowFormatTime	215487 (2분 15초 487밀리초)	5744513 (57분 44초 513밀리초)
NowFormatTimes	[SmartTimer.TIMESFORMAT] - dwHour = 0 - dwMinute = 2 - dwSec = 15 - dwMillisecond = 487	[SmartTimer.TIMESFORMAT] - dwHour = 0 - dwMinute = 57 - dwSec = 44 - dwMillisecond = 513
NowIncMillisecond	135487	
NowMillisecond	135487	3464513 (3600000 - 135487)

- long : 누적된 시간 (단위 : ms)
- SmartTimer.TIMESFORMAT : 누적된 시간이 저장된 구조체
  - long dwHour : 시간
  - long dwMinute : 분
  - long dwSec : 초
  - long dwMillisecond : 밀리초

C# 사용법

```
// 경과된 시간을 ms로 표시
lblNowms.Text = smartTimer1.NowMillisecond.ToString();
lblNowIncms.Text = smartTimer1.NowIncMillisecond.ToString();

// 경과된 시간을 자릿수 형식으로 표시
lblNowFormat.Text = smartTimer1.NowFormatTime.ToString();

// 경과된 시간을 구조체로 얻은 후 각 라벨에 표시
SmartTimer.TIMESFORMAT sFormat = smartTimer1.NowFormatTimes;
lblNowFormatH.Text = sFormat.dwHour.ToString( " 00 " );
lblNowFormatM.Text = sFormat.dwMinute.ToString( " 00 " );
lblNowFormatS.Text = sFormat.dwSec.ToString( " 00 " );
lblNowFormatms.Text = sFormat.dwMillisecond.ToString( " 00 " );
```

VB 사용법

```
lblNowms.Text = smartTimer1.NowMillisecond.ToString()
lblNowIncms.Text = smartTimer1.NowIncMillisecond.ToString()
lblNowFormat.Text = smartTimer1.NowFormatTime.ToString()
Dim sFormat As SmartTimer.TIMESFORMAT = smartTimer1.NowFormatTimes
lblNowFormatH.Text = sFormat.dwHour.ToString( " 00 " )
lblNowFormatM.Text = sFormat.dwMinute.ToString( " 00 " )
lblNowFormatS.Text = sFormat.dwSec.ToString( " 00 " )
lblNowFormatms.Text = sFormat.dwMillisecond.ToString( " 00 " )
```

프로퍼티(속성) StopWatchElaspedMicrosecond

StartWatch() 메소드 호출 후 StopWatch() 메소드를 호출하기까지의 시간을 얻습니다. (단위 : 초)

중요 반드시 StartWatch(), StopWatch() 메소드와 함께 사용해야 하며, StopWatch() 메소드 호출 후 StopWatchElaspedMicrosecond 속성을 확인하시기 바랍니다.

참고 "StartWatch(), StopWatch() 메소드" 내용을 참고하시기 바랍니다.

• double : 경과 시간 (단위 : 초)

C# 사용법

```
smartTimer1.StartWatch();
// 100개의 라인을 그리는 시간을 체크
for(int i=0; i < 100 ; i++)
{
  smartDraw1.Line(i + 50, i + 50, i + 100, i + 100);
}
smartTimer1.StopWatch();
smartLabel1.Text = smartTimer1.StopWatchElapsedMicrosecond.ToString();
```

VB 사용법

```
smartTimer1.StartWatch()
For i As Integer = 0 To 100
  smartDraw1.Line(i + 50, i + 50, i + 100, i + 100)
Next
smartTimer1.StopWatch()
smartLabel1.Text = smartTimer1.StopWatchElapsedMicrosecond.ToString()
```

메소드(함수) Start, Stop, Pause

SmartTimer의 동작 상태를 제어합니다.

• Start() : SmartTimer를 시작합니다.
• Stop() : SmartTimer를 중지합니다.
• Pause() : SmartTimer를 일시 중지합니다.

◉ void Start()
◉ void Stop()
◉ void Pause()

C# 사용법

```
smartTimer1.Start(); smartTimer1.Stop(); smartTimer1.Pause();
```

VB 사용법

```
smartTimer1.Start() : smartTimer1.Stop() : smartTimer1.Pause()
```

메소드(함수) StartWatch, StopWatch

특정 구간의 경과 시간을 체크합니다.

• StartWatch() : 체크할 구간의 시작을 설정합니다.
• StopWatch() : 체크할 구간의 끝을 설정합니다.

◉ void StartWatch()
◉ void StopWatch()

사용법

참고 StopWatchElaspedMicrosecond 속성의 사용법을 참고하시기 바랍니다.

메소드(함수) GetNowTimeFormatString

Start() 메소드 호출 후 경과된 시간을 표시 형식을 지정하여 얻습니다.

참고 "NowMillisecond 속성" 내용을 참고하시기 바랍니다.

◉ string GetNowTimeFormatString(SmartTimer.FORMATTYPENO formatNo)

[인자]

• SmartTimer.FORMATTYPENO formatNo : 경과 시간의 표현 방식

속성값	설명	속성값	설명
H_M	시간:분	M_S	분:초
H_M_S	시간:분:초	M_S_MS	분:초:밀리초
H_M_S_MS	시간:분:초:밀리초	-	-

[리턴값]

• string : 인자로 설정한 형식의 시간

[표] SmartTimer 동작 모드 및 인자값에 따른 리턴값 예시

속성값	Up Timer	Down Timer
StartTime	0	3600000 (ms)
EndTime	3600000 (ms)	0
경과 시간	135487 (2분 15초 487밀리초)	
H_M	00:02	00:57
H_M_S	00:02:15	00:57:44
H_M_S_MS	00:02:15:487	00:57:44:513
M_S	02:15	57:44
M_S_MS	02:15:487	57:44:513

C# 사용법

```
string strNowTime = smartTimer1.GetNowTimeFormatString(SmartTimer.FORMATTYPENO.H_M_S_MS);
```

VB 사용법

```
Dim strNowTime As String = smartTimer1.GetNowTimeFormatString(SmartTimer.FORMATTYPENO.H_M_S_MS)
```

메소드(함수) GetTimeFormatString

변환할 시간을 인자로 받아 표현 형식을 지정하여 문자열로 리턴합니다.

◉ SmartTimer.TIMEFORMAT GetTimeFormatString(long dwTotMillisecond)

◉ sting GetTimeFormatString(long dwTotMillisecond, SmartTimer.FORMATTYPENO formatNo)

[인자]

• long dwTotMillisecond : 변환할 시간값 (단위 : ms)

• SmartTimer.FORMATTYPENO formatNo : 경과 시간의 표현 방식

속성값	설명	속성값	설명
H_M	시간:분	M_S	분:초
H_M_S	시간:분:초	M_S_MS	분:초:밀리초
H_M_S_MS	시간:분:초:밀리초	-	-

[리턴값]

• SmartTimer.TIMESFORMAT : 누적된 시간이 저장된 구조체

- long dwHour : 시간
- long dwMinute : 분
- long dwSec : 초
- long dwMillisecond : 밀리초

• string : formatNo 인자로 설정한 형식의 문자열

참고 "GetNowTimeFormatString() 메소드" 내용을 참고하시기 바랍니다.

C# 사용법

```
// 135487ms를 변환 → " 00:02:15:487 "
lblTime.Text = smartTimer1.GetTimeFormatString(135487, SmartTimer.FORMATTYPENO.H_M_S_MS);
```

VB 사용법

```
lblTime.Text = smartTimer1.GetTimeFormatString(135487, SmartTimer.FORMATTYPENO.H_M_S_MS)
```

메소드(함수) SetNowMillisecondDec, SetNowMillisecondInc

SmartTimer의 StartTime과 EndTime, NowMilliSecond 속성값을 시간 단위(시, 분, 초)로 증가, 감소합니다.

• SetNowMillisecondDec() : 인자값에 따라 각 속성값을 감소시킵니다.
• SetNowMillisecondInc() : 인자값에 따라 각 속성값을 증가시킵니다.

주의 반드시 SmartTimer가 중지된 상태에서 호출하시기 바랍니다.

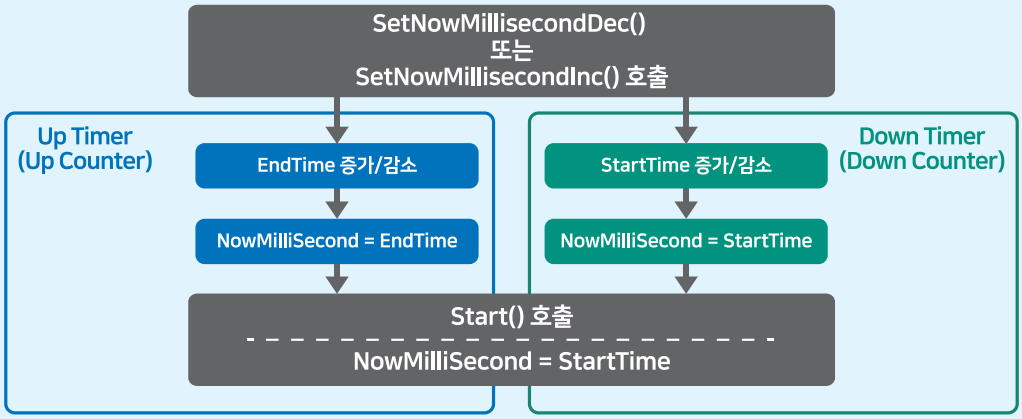
참고 StartTime, EndTime, NowMilliSecond 속성과의 관계

SmartTimer는 Up/Down Timer 두 가지 동작 모드를 설정할 수 있습니다.
동작 모드가 Up Timer인 경우, 각 메소드 호출 시 EndTime 속성이 증감되며, NowMilliSecond 속성값은 End Time 속성값과 동일하게 설정됩니다.
동작 모드가 Down Timer인 경우, 각 메소드 호출 시 StartTime 속성이 증감되며, NowMilliSecond 속성값은 StartTime 속성값과 동일하게 설정됩니다. 아래 표에서 동작을 확인하시기 바랍니다.

[표] SmartTimer 동작 모드에 따른 메소드 동작

동작 모드	Up Timer(Up Counter)	Down Timer(Down Counter)
속성 관계	StartTime 〈 EndTime Ex) StartTime = 0, EndTime = 100	StartTime 〉 EndTime Ex) StartTime = 100, EndTime = 0
SetNowMillisecondDec(), SetNowMillisecondInc() 메소드 호출 시 속성값 변화	EndTime 속성이 증가/감소되며, NowMilliSecond 속성값은 EndTime 속성값으로 설정됨. 이후 Start() 메소드 호출 시 NowMilliSecond 값은 StartTime 속성값으로 변경됨	StartTime 속성이 증가/감소되며, NowMilliSecond 속성값은 StartTime 속성값으로 설정됨

※ "StartTime, EndTime 속성", "C#, VB 사용법" 내용을 참고하시기 바랍니다.




```
bool SetNowMillisecondDec(SmartTimer.SETNOWTIMES setField, long dwMinMillisecond)
bool SetNowMillisecondDec(SmartTimer.SETNOWTIMES setField, long dwMinMillisecond, int iStep)
bool SetNowMillisecondInc(SmartTimer.SETNOWTIMES setField, long dwMaxMillisecond)
bool SetNowMillisecondInc(SmartTimer.SETNOWTIMES setField, long dwMaxMillisecond, int iStep)
```

[인자]

- SmartTimer.SETNOWTIMES setField : 증가, 감소 단위
  - SmartTimer.SETNOWTIMES.HOUR : 시간
  - SmartTimer.SETNOWTIMES.MINUTE : 분
  - SmartTimer.SETNOWTIMES.SECOND : 초
- long dwMaxMillisecond : 증가 가능한 최대값
- long dwMinMillisecond : 감소 가능한 최소값
- int iStep : 증가 및 감소 시간(Step) (기본값 : 1) ※ 단위는 SetField에서 설정됨

[리턴값]

- bool : 설정 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
private void SetValue(bool bIsInc)
{
  // SmartTimer가 중지된 상태일 때만 동작
  if(smartTimer1.IsStart == false)
  {
    // 메소드 호출 시 5초 단위로 속성값을 증가시키며, 최대 600000ms를 넘지 못함
    if(bIsInc == true)
    {
      smartTimer1.SetNowMillisecondInc(SmartTimer.SETNOWTIMES.SECOND, 600000, 5);
    }
    // 메소드 호출 시 5초 단위로 속성값을 감소시키며, 최소 5000ms를 아래로 내려가지 못함
    else { smartTimer1.SetNowMillisecondDec(SmartTimer.SETNOWTIMES.SECOND, 5000, 5); }
    smartLabel1.Text = smartTimer1.StartTime.ToString();
    smartLabel2.Text = smartTimer1.EndTime.ToString();
    smartLabel3.Text = smartTimer1.NowMillisecond.ToString();
  }
}
```

VB 사용법

```
Private Sub SetValue(Dim bIsInc As Boolean)
  If smartTimer1.IsStart = False Then
    If bIsInc = True Then
      smartTimer1.SetNowMillisecondInc(SmartTimer.SETNOWTIMES.SECOND, 600000, 5)
    Else
      smartTimer1.SetNowMillisecondDec(SmartTimer.SETNOWTIMES.SECOND, 5000, 5)
    End If
    smartLabel1.Text = smartTimer1.StartTime.ToString()
    smartLabel2.Text = smartTimer1.EndTime.ToString()
    smartLabel3.Text = smartTimer1.NowMillisecond.ToString()
  End If
End Sub
```

이벤트 OnFinishTick

SmartTimer의 Start 후 시간이 경과하여 EndTime 속성으로 설정된 시간에 도달하는 경우 발생되는 이벤트입니다. 즉, NowMillisecond 값과 EndTime 값이 같아지면 발생합니다.

참고 “EndTime 속성” 내용을 참고하시기 바랍니다.

C# 사용법

```
// Timer의 동작에 따라 NowMillisecond 값과 EndTime 값이 같아지면 발생하는 이벤트
private void smartTimer1_OnFinishTick(object sender, EventArgs e)
{
  smartLabel1.Text = smartTimer1.GetNowTimeFormatString(SmartTimer.FORMATTYPENO.H_M_S_MS);
}
```

VB 사용법

```
Private Sub smartTimer1_OnFinishTick(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles smartTimer1.OnFinishTick
  smartLabel1.Text = SmartTimer1.GetNowTimeFormatString(SmartTimer.FORMATTYPENO.H_M_S_MS)
End Sub
```

이벤트 OnIntervalSeriesTick

IntervalSeries 속성으로 설정된 간격에 따라 순서대로 OnIntervalSeriesTick 이벤트가 발생됩니다.

참고 "IntervalSeries 속성" 내용을 참고하시기 바랍니다.

◉ OnIntervalSeriesTick(int iIntervalIndex)

• int iIntervalIndex : 이벤트를 발생시킨 IntervalSeries 속성의 배열 인덱스

C# 사용법

```
private void smartTimer1_OnIntervalSeriesTick(int iIntervalIndex)
{
  // 현재 Interval 값 표시
  switch (iIntervalIndex)
  {
    case 0:
      smartLabel2.Text = " Interval Time : 3000ms ";
      break;
    case 1:
      smartLabel2.Text = " Interval Time : 1000ms ";
      break;
    case 2:
      smartLabel2.Text = " Interval Time : 5000ms ";
      break;
  }
}
```

VB 사용법

```
Private Sub smartTimer1_OnIntervalSeriesTick(ByVal iIntervalIndex As System.Int32) Handles
smartTimer1.OnIntervalSeriesTick
  Select Case iIntervalIndex
    Case 0 : smartLabel2.Text = " Interval Time : 3000ms "
    Case 1 : smartLabel2.Text = " Interval Time : 1000ms "
    Case 2 : smartLabel2.Text = " Interval Time : 5000ms "
  End Select
End Sub
```

이벤트 Tick

설정된 Interval 속성값에 따라서 Start 시 Interval 주기마다 Tick 이벤트가 발생합니다.

주의 Tick 이벤트 사용 시 주의사항

Timer Tick 이벤트는 아래 두 가지 이유로 실시간으로 처리될 수 없으며 Time Critical한 처리에 적합하지 않습니다.
1. Timer Tick 이벤트의 경우에는 O/S 스케줄링 시 처리 우선 순위가 낮으며, CPU의 성능이 낮은 경우 고성능 CPU에 비해 상대적으로 처리되는 시간이 오래 걸립니다. 또한 Task가 많아 경합이 발생될 경우 O/S 스케줄링이 증가하여 더욱 Timer Tick 이벤트의 발생 시간의 오차가 커지게 됩니다.
2. Tick 이벤트 내부의 처리할 코드가 많은 경우, 이벤트 처리에 걸리는 시간은 느려지게 됩니다.

C# 사용법

```
private void smartTimer1_Tick(object sender, EventArgs e)
{
  smartLabel1.Text = smartTimer1.GetNowTimeFormatString(SmartTimer.FORMATTYPENO.H_M_S_MS);
}
```

VB 사용법

```
Private Sub smartTimer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
smartTimer1.Tick
  smartLabel1.Text = smartTimer1.GetNowTimeFormatString(SmartTimer.FORMATTYPENO.H_M_S_MS)
End Sub
```

3) SmartTimer 예제 사용하기

SmartTimer를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartTimer"

6. SmartFile

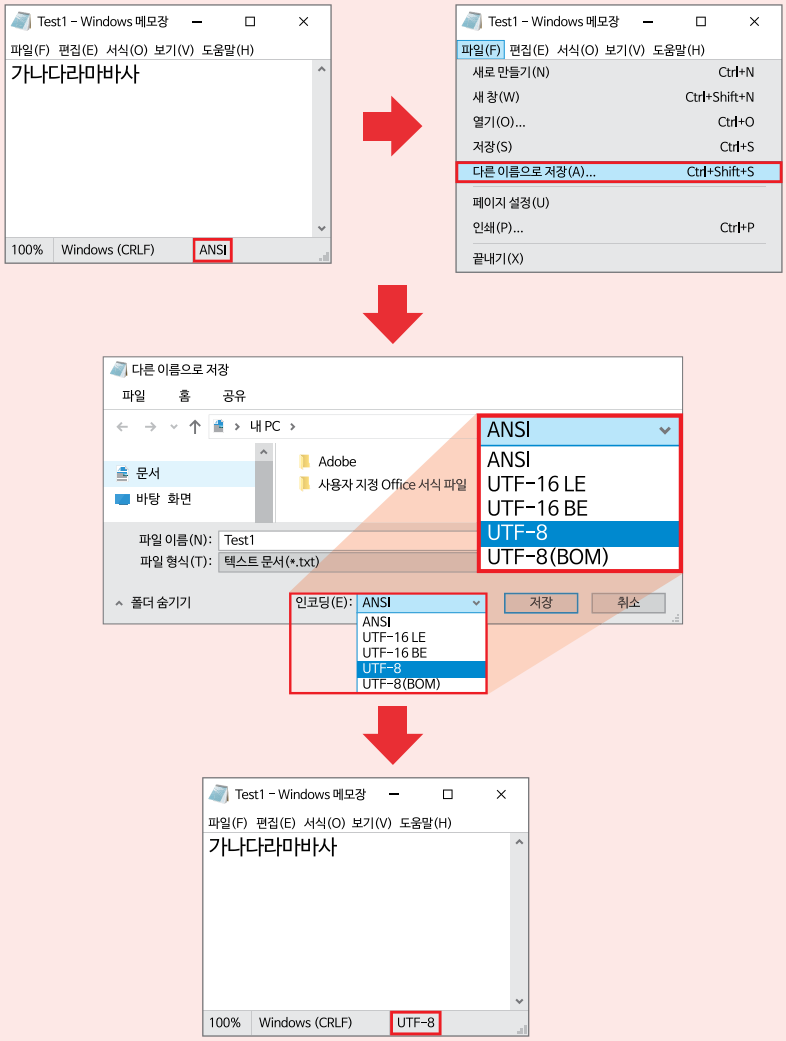
SmartFile은 문자열 및 구조체 데이터의 파일을 읽거나 쓰는 작업을 간소화하여, 파일 처리를 간편하게 할 수 있도록 한 컴포넌트입니다.

- 버퍼를 사용한 데이터의 일괄 처리 기능 지원으로 대량의 데이터를 읽거나 쓸 때 오버헤드 최소화
- 인덱스(Index)를 지원하여 특정 줄(라인)에 데이터를 읽거나 쓸 수 있도록 지원
- 구조체(Structure) 데이터를 처리
- 파일 데이터 처리를 메소드화하여 간편하게 사용하도록 지원

주의 SmartFile에서 한글 사용 시 주의사항

PC나 다른 장치에서 생성된 파일의 문자열 인코딩 방식이 ANSI인 경우, SmartFile에서 해당 파일에 한글을 쓰거나 읽으면 데이터가 깨져서 보이게 됩니다. 따라서 SmartFile에서 한글 문자를 읽거나 쓰는 경우 문자열 인코딩 방식을 반드시 UTF-8로 하여 사용하시기 바랍니다.

[ANSI에서 UTF-8로 인코딩 방식 변경하기]



1) 일괄 처리 방식과 즉시 처리 방식의 비교

SmartFile은 데이터 처리 시 일괄 처리 방식과 즉시 처리 방식을 지원합니다. 즉시 처리 방식은 데이터가 발생하면 즉시 처리하는 방식이며, 일괄 처리 방식은 계속해서 발생되는 데이터를 축적하여 두었다가 일정 시점 또는 일정량이 축적되었을 때 일괄해서 처리하는 자료 처리 방식입니다. 아래 표에서 각 방식의 비교를 확인하시기 바랍니다.

[표] 일괄 처리 방식과 즉시 처리 방식의 비교

<table>
<tr><th>처리 방식</th><th colspan="3">일괄 처리 방식</th><th colspan="3">즉시 처리 방식</th></tr>
<tr><td>설명</td><td colspan="3">데이터를 축적 후 일괄 처리</td><td colspan="3">데이터 발생 시 즉시 처리</td></tr>
<tr><td>사용 시점</td><td colspan="3">데이터를 매우 빈번하게 처리하는 경우</td><td colspan="3">항상 최신의 데이터를 유지해야 하는 경우</td></tr>
<tr><td>장점</td><td colspan="3">데이터를 매우 빈번하게 읽기/쓰기를 할 때, 내부 버퍼를 사용하여 처리하므로 물리적인 저장소의 접근을 최소화하여 성능을 개선할 수 있습니다.</td><td colspan="3">Write중 오류가 발생하는 경우 오류 발생 직전까지의 데이터를 Write합니다.</td></tr>
<tr><td>단점</td><td colspan="3">Write 작업을 올바르게 중지하지 않은 상태에서 프로그램 오류 또는 전원 공급이 중단되는 등의 오류가 발생한다면 데이터의 손실이 발생합니다.</td><td colspan="3">호출할 때마다 물리적 저장소 (SD Card, Flash Disk)에 저장하므로 여러 번 호출할 경우 오버헤드가 많이 발생하여 처리 시간이 늘어나고, 저장소 성능의 저하가 발생합니다.</td></tr>
<tr><td>관련 메소드</td><td colspan="3">ReadStringAllBuffer(), ReadStringBuffer(), WriteString_Begin(), WriteString_End(), ReadStructureAllBuffer(), ReadStructureBuffer(), WriteStructure_Begin(), WriteStructure_End()</td><td colspan="3">ReadString(), WriteString(), ReadStructure(), WriteStructure()</td></tr>
<tr><td>예제 코드</td><td colspan="3"><pre>// 시간 측정 Start
smartTimer1.StartWatch();
// 반복 횟수를 변경하며 테스트
for(int i = 0; i < 10000; i++)
{
  //SmartListBox의 Item을 RAM(Buffer)에 저장
  smartFile1.WriteString_Begin(i.ToString());
}
// RAM(Buffer)에 저장된 데이터를 파일로 Write
smartFile1.WriteString_End();
// 시간 측정 Stop
smartTimer1.StopWatch();
string strWriteTime = smartTimer1.StopWatchElap
sedMicrosecond.ToString(" 000.000 ") + " 초 ";</pre></td><td colspan="3"><pre>// 시간 측정 Start
smartTimer1.StartWatch();
// 반복 횟수를 변경하며 테스트
for(int i = 0; i < 10000; i++)
{
  smartFile1.WriteString(i.ToString());
}
// 시간 측정 Stop
smartTimer1.StopWatch();
string strWriteTime2 = smartTimer1.StopWatchElap
sedMicrosecond.ToString(" 000.000 ") + " 초 ";</pre></td></tr>
<tr><td>반복 횟수</td><td>100</td><td>5000</td><td>10000</td><td>100</td><td>5000</td><td>10000</td></tr>
<tr><td>경과 시간</td><td>0.045 초</td><td>0.422 초</td><td>0.602 초</td><td>0.337 초</td><td>12.115 초</td><td>24.441 초</td></tr>
</table>

참조 자세한 내용은 "홈페이지 → 자료실 → Tech Note → 72. 즉시 처리 방식과 일괄 처리 방식의 비교 및 관련된 SmartX Framework 컴포넌트 안내"를 참조하시기 바랍니다.

주의 SmartFile을 이용하여 File을 Write할 때 주의사항

Writing 중 프로그램 오류가 발생하거나 전원 공급이 불안정해지는 등의 오류가 발생할 경우 파일이 깨지거나 데이터의 손실이 발생할 수 있습니다. 또한, SmartFile을 이용하여 File을 쓰는 경우와 저장소에 물리적으로 Writing하기 때문에 빈번하게 Write하는 경우 저장소 성능의 저하가 발생합니다.

2) 프로그래밍 적용 가이드

SmartFile은 문자열과 구조체를 읽거나 쓸 수 있습니다. 또한, 데이터를 읽거나 쓸 때 일괄 처리 방식과 즉시 처리 방식을 적용할 수 있습니다. 본 가이드는 즉시 처리 방식을 기준으로 설명합니다.

참고 "1) 일괄 처리 방식과 즉시 처리 방식의 비교" 내용을 참고하시기 바랍니다.

CASE-1 문자열 읽기 쓰기

[STEP-1] 파일 경로 설정 및 Open하기

SmartFile을 이용해 파일을 읽거나 쓰기 위해서는 먼저 FilePathName 속성으로 파일의 경로를 설정해야 합니다. 경로가 설정되면 Open() 메소드를 사용해 파일을 Open합니다.

※ 자세한 내용은 "FilePathName 속성", "Open() 메소드"를 참고하시기 바랍니다.

```
// 읽거나 쓸 파일의 경로를 설정합니다.
smartFile1.FilePathName = " Flash Disk₩₩Test1.txt ";
// 파일을 Open 합니다.
if (smartFile1.Open() == true)
{
  // Open에 성공한 경우 다음 Step을 진행합니다.
}
else
{
  // Open에 실패한 경우 파일의 경로가 올바른지 확인합니다.
}
```

[STEP-2] 문자열 쓰기

파일이 정상적으로 Open되면 WriteString() 메소드로 문자열을 쓸 수 있습니다. 또한, 인자값을 설정해 데이터를 쓸 줄(라인)을 선택할 수 있습니다. 만약 데이터를 쓰는 중 오류가 발생하거나, 전원이 차단되면 데이터의 손실이 발생할 수 있습니다.

※ 자세한 내용은 "WriteString() 메소드"를 참고하시기 바랍니다.

```
// 파일의 마지막 줄에 문자열을 쓰고 결과를 확인합니다.
if (smartFile1.WriteString( " 1234567890 " ) == true) {  }
```

[STEP-3] 문자열 읽기

파일이 정상적으로 Open되면 ReadString() 메소드로 문자열을 읽을 수 있습니다. 또한, 인자값을 설정해 데이터를 읽을 줄(라인)을 선택할 수 있습니다.

※ 자세한 내용은 "ReadString() 메소드"를 참고하시기 바랍니다.

```
string strRead1 = smartFile1.ReadString();  // 파일의 마지막 줄의 문자열을 읽습니다.
```

[STEP-4] 파일 Close하기

SmartFile을 이용해 파일을 읽거나 쓴 후, 반드시 Close() 메소드를 호출해 사용한 파일을 닫아야 합니다. 특히, 파일에 데이터를 쓴 경우 Close() 메소드를 호출하지 않으면 데이터가 손상될 수 있습니다.

※ 자세한 내용은 "Close() 메소드"를 참고하시기 바랍니다.

```
smartFile1.Close();  // 파일을 Close합니다.
```

CASE-2 구조체 읽기 쓰기

본 가이드에서 예제로 사용할 구조체는 다음과 같습니다.

```
public struct EXSTRUCT
{
  public int iIntType;
  public double dDblType;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 50)]
```

```
  public string sStringType;  // 문자열인 경우 문자의 최대 길이를 미리 정의합니다.
}
```

[STEP-1] 파일 경로 설정 및 Open하기

SmartFile을 이용해 파일을 읽거나 쓰기 위해서는 먼저 FilePathName 속성으로 파일의 경로를 설정해야 합니다. 경로가 설정되면 Open() 메소드를 사용해 파일을 Open합니다.

※ 자세한 내용은 “FilePathName 속성”, “Open() 메소드”를 참고하시기 바랍니다.

```
// 읽거나 쓸 파일의 경로를 설정합니다.
smartFile1.FilePathName = " Flash Disk₩₩Test2.txt " ;
// 파일을 Open 합니다.
if (smartFile1.Open() == true)
{
  // Open에 성공한 경우 다음 Step을 진행합니다.
}
else
{
  // Open에 실패한 경우 파일의 경로가 올바른지 확인합니다.
}
```

[STEP-2] 구조체 쓰기

파일이 정상적으로 Open되면 WriteStructure() 메소드로 구조체를 쓸 수 있습니다. 또한, 인자값을 설정해 데이터를 쓸 줄(라인)을 선택할 수 있습니다. 만약 데이터를 쓰는 중 오류가 발생하거나, 전원이 차단되면 데이터의 손실이 발생할 수 있습니다.

※ 자세한 내용은 “ WriteStructure() 메소드”를 참고하시기 바랍니다.

```
EXSTRUCT exStruct;  // 파일에 쓰기위한 구조체 선언
// 구조체 각각의 필드마다 데이터를 설정합니다.
exStruct.iIntType = 333777;  exStruct.dDblType = 777733.33;
exStruct.sStringType = " 구조체 저장 테스트 " ;
// 지정된 Index에 설정된 구조체 데이터를 저장합니다.
if (smartFile1.WriteStructure(exStruct, 0) ==  true)
{
  // 구조체 쓰기 성공
}
```

[STEP-3] 구조체 읽기

파일이 정상적으로 Open되면 ReadStructure() 메소드로 구조체를 읽을 수 있습니다. 또한, 인자값을 설정해 데이터를 읽을 줄(라인)을 선택할 수 있습니다.

※ 자세한 내용은 “ReadStructure() 메소드”를 참고하시기 바랍니다.

```
EXSTRUCT exStruct1;  // 데이터를 읽어올 구조체 선언
bool bSuccess;  // 구조체 읽기 결과를 리턴 받을 변수
// 지정된 Index에서 저장된 구조체 데이터를 읽어와 구조체 데이터의 값이 설정됩니다.
exStruct1 = (EXSTRUCT)smartFile1.ReadStructure(0, typeof((EXSTRUCT), ref bSuccess);
if (bSuccess == true)
{
  // 읽기 성공
}
```

[STEP-4] 파일 Close하기

SmartFile을 이용해 파일을 읽거나 쓴 후, 반드시 Close() 메소드를 호출해 사용한 파일을 닫아야 합니다. 특히, 파일에 데이터를 쓴 경우 Close() 메소드를 호출하지 않으면 데이터가 손상될 수 있습니다.

※ 자세한 내용은 "Close() 메소드"를 참고하시기 바랍니다.

```
smartFile1.Close();  // 파일을 Close합니다.
```

3) SmartFile 인터페이스 설명

SmartFile Component Interface		
속성		
FilePathName : string		
메소드		
Close() : void	FileSearch(string strRefPath, string strFilepattern, bool bSubDirSearch) : SEARCHFILEINFOS[]	GetCount() : long (+1개 오버로드)
GetPrivateProfileString(string strApplicationName, string strKeyName, string strDefultValue, out string strReturnValue, string strFileName) : static bool	Open() : bool (+1개 오버로드)	ReadString() : string (+3개 오버로드)
ReadStringAllBuffer() : bool	ReadStringBuffer(int iLineNum) : string	ReadStructure(Type anyType) : object (+3개 오버로드)
ReadStructureAllBuffer() : bool	ReadStructureBuffer(Type anyType, int iIndex) : Object	ReadStructureFirstPosSet() : void
WritePrivateProfileString(string strApplicationName, string strKeyName, string strValue, string strFileName) : static bool	WriteString(string strWrite) : bool (+1개 오버로드)	WriteString_Begin(string strWrite) : bool (+1개 오버로드)
WriteString_End() : void	WriteStructure(object anyType) : bool (+1개 오버로드)	WriteStructure_Begin(object anyType) : bool (+1개 오버로드)
WriteStructure_End() : void		

프로퍼티(속성) FilePathName

읽기 및 쓰기 처리할 파일의 경로와 이름을 설정합니다.

- string : 파일의 경로와 이름

C# 사용법

```
smartFile1.FilePathName = "Flash Disk₩₩Test1.txt";  // 대상 파일을 설정
```

VB 사용법

```
smartFile1.FilePathName = "Flash Disk₩₩Test1.txt"
```

메소드(함수) FileSearch

특정 경로의 파일을 검색합니다. 다중 검색 및 하위 폴더 검색 기능을 지원합니다.

◉ SEARCHFILEINFOS[] FileSearch(string strRefPath, string strFilepattern, bool bSubDirSearch)

[인자]

- string strRefPath : 파일을 검색할 폴더
- string strFilepattern : 검색할 파일의 이름 또는 확장자

참고 다중 검색을 지원하며, 검색어의 구분은 ';' 로 합니다.
Ex) 파일 이름이 test이거나, 확장자가 txt, png인 파일을 모두 검색 = "test.*;*.txt;*.png"

• bool bSubDirSearch : 하위 폴더 검색 활성화 여부
  – true : 하위 폴더 검색
  – false : 하위 폴더 검색 안 함

[리턴값]

• SEARCHFILEINFOS[] : 검색한 파일들의 정보를 저장한 구조체 배열
  – string SEARCHFILEINFOS.strFileName : 파일 이름
  – string SEARCHFILEINFOS.strFileExtension : 파일 확장자
  – string SEARCHFILEINFOS.strPath : 파일 경로

C# 사용법

```
// Flash Disk₩₩New 경로에 파일 이름이 test이거나 파일 확장자가 txt인 파일을 검색. 하위 폴더 검색
SEARCHFILEINFOS[] FileList = smartFile1.FileSearch("Flash Disk₩₩New", "test.*;*.txt", true);
for (int i = 0; i < FileList.Length; i++)
{
  smartListBox1.AddItem("[ " + FileList[i].strPath + " ] " + FileList[i].strFileName + "."
  + FileList[i].strFileExtension);  // SmartListBox에 검색한 파일들의 경로를 저장
}
```

VB 사용법

```
Dim FileList() As SEARCHFILEINFOS = smartFile1.FileSearch("Flash Disk₩₩New", "test.*;*.txt",
True)
For i As Integer = 0 To FileList.Length - 1
  smartListBox1.AddItem("[ " + FileList(i).strPath + " ] " + FileList(i).strFileName + "."
  + FileList(i).strFileExtension)
Next
```

메소드(함수) Open, Close

• Open() : 파일을 읽거나 쓰기 위해 Open합니다.
• Close() : 파일을 읽거나 쓴 후 Close합니다.

주의 파일 Open 및 Close 시 주의사항

1. Open() 메소드 호출 전 반드시 FilePathName 속성을 설정하시기 바랍니다.
2. Open() 메소드는 파일을 핸들링하는 메소드로, 중복 호출 시 에러가 발생하게 됩니다.
따라서 중복 호출에 주의하시기 바랍니다.
3. 파일을 쓴 경우 Close() 메소드를 호출하지 않을 경우 작성한 데이터가 손실될 수 있습니다.

◉ bool Open()
◉ bool Open(int iBufferSize)
◉ bool Close()

[인자]

• int iBufferSize : 읽거나 쓸 파일의 버퍼 용량

참고 SmartFile을 사용하여 파일 Open 시 iBufferSize 인자값을 설정하여 읽기 및 쓰기 가능한 버퍼 용량을 확보할 수 있습니다. 인자값 생략 시 4096Byte로 설정되며, 버퍼 사이즈만큼 Open한 파일을 읽거나 쓸 수 있습니다.

[리턴값]

• bool : 파일 Open 또는 Close 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
smartFile1.FilePathName = "Flash Disk₩₩Test1.txt";  // 대상 파일을 설정
```

```
if (smartFile1.Open() == true)
{
  // 파일 Open 성공. 파일 읽기 및 쓰기 처리 코드 작성
  smartFile1.Close();  // 읽기 및 쓰기 완료 후 파일 Close
}
```

VB 사용법

```
smartFile1.FilePathName = "Flash Disk₩₩Test1.txt"
If smartFile1.Open() = True Then
  '파일 Open 성공
  smartFile1.Close()
End If
```

메소드(함수) GetCount

현재 Open 된 파일의 항목(줄) 수를 리턴합니다.

중요 파일의 데이터 타입에 따른 리턴값

GetCount() 메소드의 경우 데이터 타입에 따라 인자값을 설정해 사용해야 합니다. 아래 표에서 데이터 타입에 따른 사용 메소드와 리턴값을 확인하시기 바랍니다.

데이터 타입	문자열	구조체
사용 메소드	GetCount()	GetCount(Type anyType)
리턴값	파일의 라인(개행된 줄) 개수	구조체 데이터의 개수

◉ void GetCount()
◉ void GetCount(Type anyType)

[인자]

• Type anyType : 구조체 데이터의 타입

[리턴값]

• long : 파일의 항목 수

C# 사용법

```
// 구조체 데이터의 개수를 얻습니다. EXPSTRUCT는 샘플 구조체입니다.
long lstuctCnt = smartFile1.GetCount(typeof(EXPSTRUCT));
long lstrCnt = smartFile1.GetCount();
```

VB 사용법

```
Dim lstuctCnt As long = smartFile1.GetCount(typeof(EXPSTRUCT))
Dim lstrCnt As long = smartFile1.GetCount()
```

메소드(함수) ReadString, WriteString

문자열을 라인 단위로 읽거나 씁니다.

• ReadString() : 파일의 문자열을 읽습니다.
• WriteString() : 파일에 문자열을 씁니다.

참고 "일괄 처리 방식과 즉시 처리 방식의 비교" 내용을 참고하시기 바랍니다.

◉ bool WriteString(string strWrite)
◉ bool WriteString(string strWrite, int iLineNum)
◉ string ReadString()
◉ string ReadString(int iLineNum)
◉ string ReadString(ref bool bSuccess)

◉ string ReadString(int iLineNum, ref bool bSuccess)

[인자]

- string strWrite : 파일에 쓸 문자열
- int iLineNum : 읽거나 쓸 문자열의 줄번호 (시작 번호 : 0)

중요 iLineNum 인자값을 생략할 경우 파일의 마지막 줄에 문자열을 쓰거나 읽습니다.

- ref bool bSuccess : 파일 읽기 성공 여부를 저장받을 변수
  - true : 성공
  - false : 실패

[리턴값]

- string : 읽어온 문자열
- bool : 쓰기 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
if (smartFile1.WriteString( " 1234567890 " ) == true)
{
  // 마지막 줄에 문자열 쓰기 성공
}
if (smartFile1.WriteString( " ABCDEFGHIJK " , 1) == true)
{
  // 지정한 줄에 문자열 쓰기 성공
}
// 파일의 마지막 줄의 문자열을 읽음
string strRead1 = smartFile1.ReadString();
bool bRead1 = false;
// 지정 줄 문자열 읽기 및 성공 여부 확인
string strRead2 = smartFile1.ReadString(1, ref bRead1);
// 읽기 성공 시 문자열을 라벨에 출력
if (bRead1 == true)
{
  lblReadData.Text = strRead2;
}
```

VB 사용법

```
If smartFile1.WriteString( " 1234567890 " ) = True Then
End If
If smartFile1.WriteString( " ABCDEFGHIJK " , 1) = True Then
End If
Dim strRead1 As String = smartFile1.ReadString()
Dim bRead1 As Boolean = False
Dim strRead2 As String = smartFile1.ReadString(1, bRead1)
If bRead1 = True Then
  lblReadData.Text = strRead2
End If
```

메소드(함수) ReadStringAllBuffer, ReadStringBuffer

파일의 문자열 데이터를 읽어와 버퍼에 저장하여 일괄로 처리합니다.

- ReadStringAllBuffer() : 파일의 모든 문자열을 버퍼에 일괄 저장합니다.
- ReadStringBuffer() : 버퍼에서 지정한 줄의 문자열을 읽습니다.

참고 "일괄 처리 방식과 즉시 처리 방식의 비교" 내용을 참고하시기 바랍니다.

◉ `bool ReadStringAllBuffer()`
◉ `string ReadStringBuffer(int iLineNum)`

[인자]
• `int iLineNum` : 읽을 문자열의 줄 번호 (시작 번호 : 0, 생략 시 마지막 줄의 문자열을 읽음)
[리턴값]
• `string` : 읽어온 문자열
• `bool` : 읽기 성공 여부
  – `true` : 성공
  – `false` : 실패

C# 사용법

```
int iDataCnt = smartFile1.GetCount();  // 현재 파일의 줄 수를 읽어옴
string[] strData = new string[iDataCnt];  // 파일의 데이터를 저장할 배열의 크기를 줄 수로 생성
if (smartFile1.ReadStringAllBuffer() == true)
{
  // 파일의 데이터를 일괄로 읽어오고, 성공할 경우 데이터를 처리
  for (int i = 0; i < iDataCnt; i++)
  {
    strData[i] = smartFile1.ReadStringBuffer(i);
  }
}
```

VB 사용법

```
Dim iDataCnt As Integer = smartFile1.GetCount()
Dim strData() As String = New String(iDataCnt) {}
If smartFile1.ReadStringAllBuffer() = True Then
  For I As Integer = 0 To iDataCnt
    strData(i) = smartFile1.ReadStringBuffer(i)
  Next
End If
```

메소드(함수) WriteString_Begin, WriteString_End

파일에 쓸 데이터를 버퍼에 저장 후 일괄로 파일에 데이터를 씁니다.
• `WriteString_Begin()` : 파일에 쓸 문자열을 버퍼에 저장합니다.
• `WriteString_End()` : 버퍼에 저장한 문자열들을 일괄로 파일에 씁니다.

중요
1. WriteString_Begin() 메소드는 호출 시 한 번에 한 줄의 문자열을 저장합니다.
2. WriteString_Begin() 메소드를 사용해 버퍼에 모든 문자열을 저장 후 반드시 WriteString_End() 메소드를 호출하시기 바랍니다. 메소드를 호출하지 않으면 버퍼에 저장된 데이터가 실제 파일에 써지지 않습니다.

참고 "일괄 처리 방식과 즉시 처리 방식의 비교" 내용을 참고하시기 바랍니다.

◉ `bool WriteString_Begin(string strWrite)`
◉ `bool WriteString_Begin(string strWrite, int iLineNum)`
◉ `void WriteString_End()`

[인자]
• `string strWrite` : 파일에 쓸 문자열
• `int iLineNum` : 파일에 쓸 문자열의 줄 번호 (시작 번호 : 0, 생략 시 마지막 줄에 추가됨.)

[리턴값]
• bool : 버퍼에 문자열 저장 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// 버퍼에 100줄의 문자열을 저장
for (int i = 0; i < 100; i++)
{
  smartFile1.WriteString_Begin(i.ToString());
}
smartFile1.WriteString_End();  // 버퍼에 저장된 문자열들을 실제 파일로 저장
```

VB 사용법

```
For I As Integer = 0 To iDataCnt
  smartFile1.WriteString_Begin(i.ToString())
Next
smartFile1.WriteString_End()
```

메소드(함수) ReadStructureFirstPosSet

ReadStructure() 메소드를 사용해 저장된 구조체를 읽는 경우, 순차적으로 구조체 데이터를 읽기 위해 현재 파일 읽기 위치를 강제로 처음(Index = 0)으로 설정합니다.

◉ void ReadStructureFirstPosSet()

C# 사용법

```
// 읽기 위치를 처음으로 설정. 데이터를 순차적으로 읽게함
smartFile1.ReadStructureFirstPosSet();
```

VB 사용법

```
smartFile1.ReadStructureFirstPosSet()
```

메소드(함수) ReadStructure, WriteStructure

구조체 데이터를 읽거나 씁니다.
• ReadStructure() : 파일의 구조체를 읽습니다.
• WriteStructure() : 파일에 구조체를 씁니다.

중요 구조체에 문자열이 포함된 경우, 문자열의 최대 길이를 미리 정의하시기 바랍니다.

[C#] 구조체 정의 예시

```
public struct EXPSTRUCT
{
  public int iIntType;
  public double dDblType;
  public long lLongType;
  [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 50)]
  public string sStringType;  // 문자열인 경우 문자의 최대 길이를 미리 정의합니다.
}
```

참고 “일괄 처리 방식과 즉시 처리 방식의 비교” 내용을 참고하시기 바랍니다.

◉ object ReadStructure(Type anyType)
◉ object ReadStructure(int iIndex, Type anyType)

◉ object ReadStructure(Type anyType, ref bool bSuccess)
◉ object ReadStructure(int iIndex,Type anyType, ref bool bSuccess)
◉ bool WriteStructure(object anyType)
◉ bool WriteStructure(object anyType, int iIndex)

[인자]

- Type anyType : 파일에서 지정된 구조체 데이터를 읽기 위한 구조체 타입 정보
- object anyType : 파일에 저장될 구조체 데이터 변수
- int iIndex : 파일에 저장될 위치 순서 (시작 번호 : 0, 생략 시 마지막 데이터)

중요 iLineNum 인자값을 생략할 경우 파일의 마지막 줄에 문자열을 쓰거나 읽습니다.

- ref bool bSuccess : 파일 읽기 성공 여부를 저장받을 변수
  - true : 성공
  - false : 실패

[리턴값]

- object : 읽어온 구조체. 사용하는 구조체 형식으로 타입 캐스팅하여 사용
- bool : 쓰기 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
EXPSTRUCT exStruct;  // 파일에 쓰기 위한 구조체 선언 및 구조체 각각의 필드마다 데이터를 설정
exStruct.iIntType = 333777; exStruct.lLongType = 77773333; exStruct.sStringType = "구조체 저장";
smartFile1.WriteStructure(exStruct, 0);  // 지정된 Index에 설정된 구조체 데이터를 저장
bool bSuccess;
// 지정된 Index에서 저장된 구조체 데이터를 읽어와 구조체 데이터에 값이 설정됨
EXPSTRUCT exStruct1 = (EXPSTRUCT)smartFile1.ReadStructure(0, typeof(EXPSTRUCT), ref bSuccess);
if (bSuccess == true)
{
  // 읽기 성공 시 구조체 처리 코드를 작성
}
```

VB 사용법

```
Dim exStruct As EXPSTRUCT
exStruct.iIntType = 333777 : exStruct.lLongType = 77773333 : exStruct.sStringType = "구조체 저장"
smartFile1.WriteStructure(exStruct, 0)
Dim bSuccess As Boolean
Dim exStruct1 As EXPSTRUCT = DirectCast(smartFile1.ReadStructure(0, GetType(EXPSTRUCT), bSuccess),
EXPSTRUCT)
If bSuccess = True Then
End If
```

메소드(함수) ReadStructureAllBuffer, ReadStructureBuffer

파일의 구조체 데이터를 모두 버퍼에 저장하여 일괄로 처리합니다.

- ReadStructureAllBuffer() : 파일의 모든 구조체를 버퍼로 일괄 읽어옵니다.
- ReadStructureBuffer() : 버퍼에서 지정한 줄의 구조체를 읽습니다.

참고 "일괄 처리 방식과 즉시 처리 방식의 비교" 내용을 참고하시기 바랍니다.

◉ bool ReadStructureAllBuffer()
◉ string ReadStructureBuffer(int iLineNum)

[인자]

- int iLineNum : 읽을 구조체의 줄 번호 (시작 번호 : 0, 생략 시 마지막 데이터)

[리턴값]
• object : 읽어온 구조체
• bool : 읽기 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// 구조체 생성자
EXPSTRUCT[] m_EXPSTRUCT = new EXPSTRUCT[10];
If(smartFile1.ReadStructureAllBuffer() == true)
{
  for (int i = 0; i < 10; i++)
  {
    // ReadStructureAllBuffer와 ReadStructureBuffer는 쌍으로 동작
    m_EXPSTRUCT[i] = (EXPSTRUCT)smartFile1.ReadStructureBuffer(typeof(EXPSTRUCT), i);
  }
}
```

VB 사용법

```
Dim expstruct(10) As EXPSTRUCT
If smartFile1.ReadStructureAllBuffer() = True Then
  For i As Integer = 0 To 10
    expstruct(i) = DirectCast(smartFile1.ReadStructureBuffer(GetType(EXPSTRUCT), i)
  Next
End If
```

메소드(함수) WriteStructure_Begin, WriteStructure_End

파일에 쓸 구조체를 버퍼에 저장 후 일괄로 파일에 데이터를 씁니다.
• WriteStructure_Begin() : 파일에 쓸 구조체를 버퍼에 저장합니다.
• WriteStructure_End() : 버퍼에 저장한 구조체들을 일괄로 파일에 씁니다.

중요
1. WriteStructure_Begin() 메소드는 호출 시 한번에 한 줄의 문자열을 저장합니다.
2. WriteStructure_Begin() 메소드를 사용해 버퍼에 모든 문자열을 저장 후 반드시 WriteStructure_End() 메소드를 호출하시기 바랍니다. 메소드를 호출하지 않으면 버퍼에 저장된 데이터가 실제 파일에 써지지 않습니다.

참고 "일괄 처리 방식과 즉시 처리 방식의 비교" 내용을 참고하시기 바랍니다.

◉ bool WriteStructure_Begin(object anyType)
◉ bool WriteStructure_Begin(object anyType, int iIndex)
◉ void WriteStructure_End()

[인자]
• object anyType : 파일에 쓸 구조체
• int iIndex : 파일에 쓸 구조체의 줄 번호 (시작 번호 : 0, 생략 시 마지막에 추가됨)

[리턴값]
• bool : 버퍼에 구조체 저장 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// 버퍼에 100줄의 문자열을 저장
for (int i = 0; i < 100; i++)
{
```

```
  smartFile1.WriteStructure_Begin(exStruct);
}
smartFile1.WriteStructure_End();  // 버퍼에 저장된 문자열들을 실제 파일로 저장
```

VB 사용법

```
For I As Integer = 0 To iDataCnt
  smartFile1.WriteString_Begin(i.ToString())
Next
smartFile1.WriteString_End()
```

정적 메소드(함수) GetPrivateProfileString

ini 형식의 파일을 읽습니다.

참고 아래의 "ini 파일 관련 기본 설명" 내용을 참고하시기 바랍니다.

◉ `static bool GetPrivateProfileString(string strApplicationName, string strKeyName, string strDefulValue, out string strReturnValue, string strFileName)`

[인자]

- `string strApplicationName` : 섹션명
- `string strKeyName` : 매개 변수 이름
- `string strDefultValue` : 읽기 실패 시 Default로 읽을 값
- `out string strReturnValue` : 매개 변수의 값을 읽어 저장할 문자열 변수
- `string strFileName` : ini 파일의 경로

[리턴값]

- `bool` : 읽기 성공 여부
  - `true` : 성공
  - `false` : 실패

C# 사용법

```
string strReturnValue;  // 값을 저장할 변수
SmartFile.GetPrivateProfileString("TITLE2", "KEYNAME2", "200", out strReturnValue,
                                  "Flash Disk₩₩Test₩₩Test1.ini");
```

VB 사용법

```
Dim strReturnValue As String
SmartFile.GetPrivateProfileString("TITLE2", "KEYNAME2", "200", strReturnValue,
                                  "Flash Disk₩₩Test₩₩Test1.ini")
```

정적 메소드(함수) WritePrivateProfileString

ini 형식의 파일을 씁니다.

◉ `static bool WritePrivateProfileString(string strApplicationName, string strKeyName, string strValue, string strFileName)`

[인자]

- `string strApplicationName` : 섹션명
- `string strKeyName` : 매개 변수 이름
- `string strValue` : 매개 변수값
- `string strFileName` : ini 파일의 경로

[리턴값]

- `bool` : 쓰기 성공 여부
  - `true` : 성공
  - `false` : 실패

C# 사용법

```
SmartFile.WritePrivateProfileString( " TITLE1 " , " KEYNAME1 " , " 100 " , " Flash Disk₩₩Test₩₩
                                     Test1.ini " );
```

VB 사용법

```
SmartFile.WritePrivateProfileString( " TITLE1 " , " KEYNAME1 " , " 100 " , " Flash Disk₩₩Test₩₩
                                     Test1.ini " )
```

참고 ini 파일 관련 기본 설명

ini(Initialization) 파일 포맷은 일부 플랫폼이나 소프트웨어의 구성 파일에 대한 비공식 표준입니다. ini 파일은 섹션, 속성 및 값으로 구성된 기본 구조의 간단한 텍스트 파일입니다. 보통 마이크로소프트 윈도우 계열에서 지원되고 있지만, 다른 운영체제에서도 사용할 수 있습니다. 사람이 읽을 수 있고 파싱하기 쉽기 때문에 복잡하지 않은 구성 파일에 사용할 수 있는 형식입니다.

[표] ini 파일 포맷 설명 및 예시

포맷	설명
섹션	매개 변수는 임의의 이름으로 지정된 여러 개의 섹션으로 구분할 수 있다. 이 섹션 이름은 대괄호"[ ]" 로 구분합니다
매개 변수 이름	매개 변수의 이름을 지정합니다. GetPrivateProfileString(), WritePrivateProfileString() 메소드의 strKeyName 인자값에서 매개 변수 이름을 지정할 수 있습니다.
매개 변수값	매개 변수의 값을 지정합니다. 매개 변수의 값은 매개 변수 이름 뒤에 등호와 함께 작성됩니다. WritePrivateProfileString() 메소드의 strValue 인자값에서 매개 변수값을 지정할 수 있으며, 등호는 자동으로 추가됩니다.
주석	문자열 앞에 세미콜론(;)을 추가하여 주석으로 사용할 수 있습니다. 주석 내용은 SmartFile에서 읽거나 쓸 수 없습니다.

소스 코드 예시	생성된 ini 파일
SmartX.SmartFile.WritePrivateProfileString(" First " , " Name " , " Mike " , " Flash Disk₩₩Test1.ini "); SmartX.SmartFile.WritePrivateProfileString(" First " , " Status " , " Single " , " Flash Disk₩₩Test1.ini "); SmartX.SmartFile.WritePrivateProfileString(" First " , " Age " , " 20 " , " Flash Disk₩₩Test1.ini ");	[First] Name=Mike Status=Single Age=20

4) SmartFile 예제 사용하기

SmartFile을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartFile1, 2"

7. SmartUpdate

IEC-Series의 RunTime 모드에서 실행 중인 장치 응용 프로그램을 업데이트하기 위해서는 제품을 개발 모드로 접근하여 수작업으로 파일들을 복사해야 하는 불편함이 발생합니다. SmartUpdate는 장치 응용 프로그램에서 편리하게 프로그램을 업데이트 처리할 수 있도록 만들어진 컴포넌트입니다.

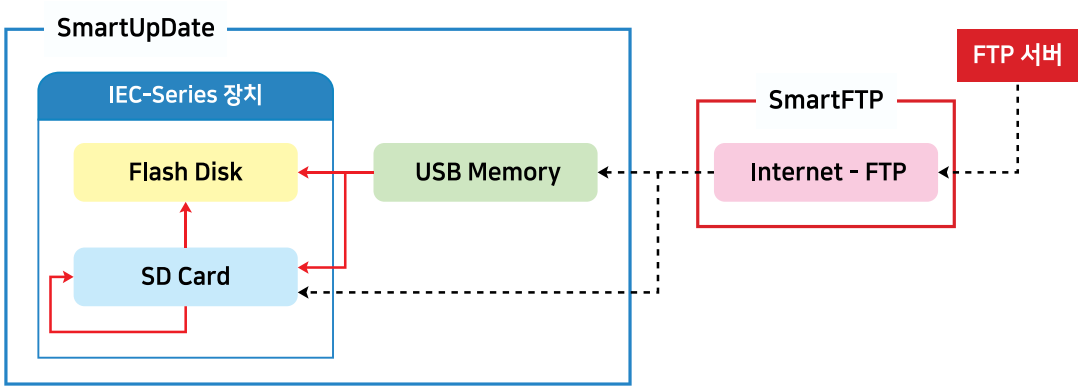
- 실행 중인 프로그램을 업데이트할 수 있도록 처리 지원
- Run 폴더의 모든 파일 업데이트 처리 대상
- USB Memory를 통한 간편한 업데이트 지원
- 업데이트 진행 과정을 다이얼로그 창으로 출력 및 설정 지원
- 다양한 업데이트 유형 지원

주의 SmartUpdate 사용 시 주의사항

1. SmartUpdate는 파일 단위(여러 개의 파일 가능)의 업데이트만 가능합니다. 즉 업데이트 대상 폴더 내부에 폴더와 파일이 있는 경우 파일만 업데이트됩니다.
2. SmartUpdate는 파일 업데이트 시 Checksum으로 비교합니다. 동일한 파일인 경우 업데이트하지 않습니다.

1) 업데이트 지원 경로와 유형

SmartUpdate는 ToFilePath, FromFilePath 속성으로 업데이트 대상 파일이 있는 경로와 업데이트 파일이 있는 경로를 설정해야 합니다. SmartUpdate 사용 시 업데이트 지원 경로와 유형은 아래와 같습니다.



[그림] 업데이트 지원 경로

[표] 업데이트 지원 유형

CASE	ToFilePath(업데이트 대상 파일이 있는 경로)	FromFilePath(업데이트 파일이 있는 경로)
1	Flash Disk	SD Card
2		USB Memory
3		Internet – FTP
4	SD Card	SD Card
5		USB Memory
6		Internet – FTP

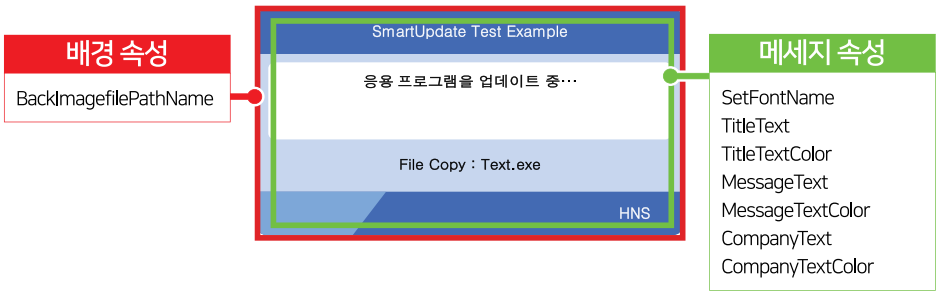
※ "ToFilePath, FromFilePath 속성" 내용을 참고하시기 바랍니다.

참고 SmartFTP를 사용하여 업데이트할 경우 (CASE-3, CASE-6인 경우)

SmartFTP를 이용한 업데이트인 경우 우선 임시 저장소(SD Card, USB Memory)에 업데이트할 대상 파일을 다운로드 합니다. 이후 SmartUpdate에서 임시로 다운로드 받은 업데이트 파일로 장치 응용 프로그램을 업데이트합니다. 업데이트 진행 시 대상 파일과 업데이트 파일을 서로 비교 후 업데이트가 이루어집니다.

※ 자세한 내용은 "SmartFTP" 내용을 참고하시기 바랍니다.

2) 디자인 요소별 속성 명칭



3) 프로그래밍 적용 가이드

STEP-1 업데이트 진행 시 다이얼로그 창의 디자인 속성 설정하기

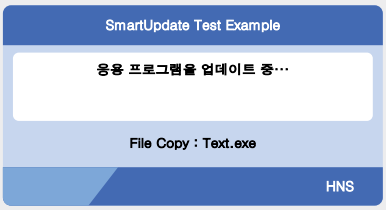
업데이트 진행 시 다이얼로그 창의 배경 이미지는 기본 이미지를 사용하는 방법과 사용자 이미지를 사용하는 방법이 있습니다.

[CASE-1] 기본 이미지를 사용할 경우

기본 이미지를 사용할 경우 BackImagefilePathName 속성을 사용하지 않습니다. 업데이트 진행 시 다이얼로그 창에 표시되는 메시지는 Title, Message, Company로 나누어지며, 순서대로 각각 상단 가운데, 중앙, 하단 우측의 메시지 텍스트 및 색상을 설정할 수 있습니다.

※ 자세한 내용은 "SetFontName, TitleText, TitleTextColor, MessageText, MessageTextColor, CompanyText, CompanyTextColor 속성"을 참고하시기 바랍니다.

```
smartUpdate1.SetFontName( " gulim " );  // 메시지 폰트 설정
// 상단 가운데 메시지 속성 설정
smartUpdate1.TitleText = " SmartUpdate Test Example " ; smartUpdate1.TitleTextColor = Color.White;
// 중앙 메시지 속성 설정
smartUpdate1.MessageText = " 응용 프로그램을 업데이트 중… " ;
smartUpdate1.MessageTextColor = Color.Black;
// 하단 우측 메시지 속성 설정
smartUpdate1.CompanyText = " HNS " ;  smartUpdate1.CompanyTextColor = Color.White;
```



[그림] 기본 이미지 사용할 경우 다이얼로그 창

[CASE-2] 사용자 이미지를 사용할 경우

사용자 이미지를 사용할 경우 BackImagefilePathName 속성을 사용합니다. 업데이트 진행 시 다이얼로그 창에 표시되는 메시지는 Title, Message, Company로 나누어지며, 순서대로 각각 상단 가운데, 중앙, 하단 우측의 메시지 텍스트 및 색상을 설정할 수 있습니다.

※ 자세한 내용은 "BackImagefilePathName, SetFontName, TitleText, TitleTextColor, MessageText, MessageTextColor, CompanyText, CompanyTextColor 속성"을 참고하시기 바랍니다.

```
smartUpdate1.BackImagefilePathName = " Flash₩₩UpdateImg.bmp " ;  // 이미지 설정
// 이외의 설정은 CASE-1과 동일합니다.
```

[그림] 사용자 이미지 사용할 경우 다이얼로그 창

STEP-2 업데이트 경로 설정하기

업데이트 진행 시 업데이트 파일과 업데이트할 대상 파일이 필요합니다. FromFilePath 속성으로 업데이트 파일이 있는 경로를 설정할 수 있으며, ToFilePath 속성으로 업데이트할 대상 파일의 경로를 설정합니다. "업데이트 지원 유형" 표를 참고하여 경로를 설정하시기 바랍니다.

※ 자세한 내용은 "FromFilePath, ToFilePath 속성"을 참고하시기 바랍니다.

```
// CASE-2 : Flash Disk ← USB Memory
smartUpdate1.FromFilePath = " 하드 디스크₩₩UpdateFiles " ;
smartUpdate1.ToFilePath = " Flash Disk₩₩Run " ;
```

STEP-3 업데이트 시작하기

Start() 메소드 조건문에서 True인 경우 Application.Exit()를 호출하여 현재 응용 프로그램을 종료합니다. 만약 응용 프로그램의 복잡도에 따라 종료가 지연된다면 업데이트가 제대로 시작되지 못하므로 응용 프로그램이 종료된 후 업데이트 시작까지의 지연 시간을 SetStartInterval 속성으로 설정합니다.

※ 자세한 내용은 "SetStartInterval 속성"과 "Start() 메소드"를 참고하시기 바랍니다.

```
// 업데이트 시작 지연 시간 설정
smartUpdate1.SetStartInterval = 3000;
// Update File을 확인하여 업데이트할 파일이 있을 경우 업데이트 시작
if (smartUpdate1.Start() == true)
{
  Thread.Sleep(200);  // 200ms 동안 쉼
  Application.Exit();  // 현재 응용 프로그램을 종료
}
```

STEP-4 업데이트 완료 후 동작 설정하기

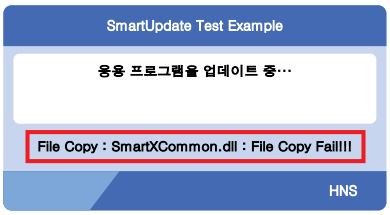
업데이트 완료 후 Restart 속성을 사용하여 IEC-Series를 재부팅 또는 업데이트된 새로운 장치 응용 프로그램을 실행할지를 설정합니다.

※ 자세한 내용은 "Restart 속성"을 참고하시기 바랍니다.

```
// 업데이트 완료 후 업데이트된 새로운 장치 응용 프로그램 실행
smartUpdate1.Restart = SmartX.SmartUpdate.RESTARTFLAG.AUTORUNEXECUTE;
```

4) 업데이트 오류가 발생할 경우

업데이트가 안 되는 대부분의 원인은 프로그램 내에서 만들어진(생성된) 스레드가 종료되지 않아 프로그램(관련 DLL 파일 포함)의 삭제가 안 되기 때문에 발생합니다. 즉, SmartUpdate는 기존 파일을 삭제하고 새 파일을 복사하는 과정에서 정상 복사될 수 없으며, 아래와 같은 "File Copy Fail!!!" 메시지가 표시되면서 복사가 되지 않습니다.



4-1) 응용 프로그램이 정상 종료됐는지 확인하는 방법

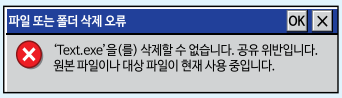
STEP-1 IEC-Series 제품에서 직접 확인하기

Flash Disk₩Run 또는 SD Card₩Run 폴더에서 EXE 파일을 삭제해봅니다. 파일이 삭제된다면 프로그램 내부의 전체 프로세스가 정상 종료됨을 의미합니다.

주의 EXE 파일이 삭제돼도 스레드에 남아있는 경우가 있습니다. 반드시 STEP-2까지 확인하시기 바랍니다.

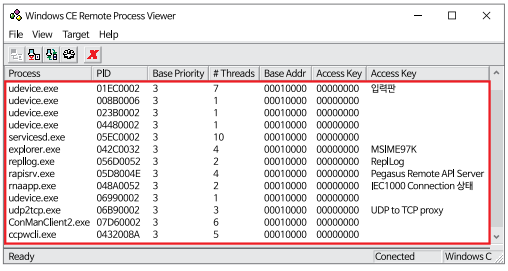
참고 프로그램이 정상적으로 종료되지 않았을 경우

프로그램이 정상적으로 종료되지 않았다면 DLL 또는 EXE 파일 삭제 시 아래와 같이 에러 메시지가 나타납니다. 이런 경우 업데이트 시 실패하게 됩니다.



STEP-2 Remote Process Viewer(원격 프로세스 뷰어)에서 확인하기

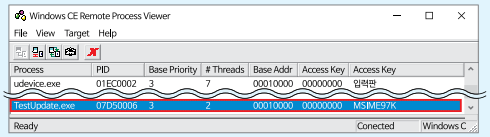
VisualStudio Remote Tools의 Remote Process Viewer(원격 프로세스 뷰어)를 실행하여 현재 리스트에 해당 프로그램이 실행 중이지 않다면 프로그램 내부의 전체 프로세스가 종료됨을 의미합니다.



참고 프로그램이 정상적으로 종료되지 않았을 경우

프로그램이 정상적으로 종료되지 않았다면 아래와 같이 응용 프로그램이 리스트에 보입니다.

※ 아래 화면은 응용 프로그램 "TestUpdate"가 업데이트 대상일 때, 종료가 안 된 화면입니다.



참조 Visual Studio Remote Tools의 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 61. Visual Studio 2005 및 2008 Remote Tools 사용 안내"를 참조하시기 바랍니다.

4-2) 해결 방법

업데이트를 시작하기 위해 스레드를 종료하는 방법은 프로그램 종료 지연으로 인한 업데이트 오류인 경우와 스레드를 호출하는 컴포넌트를 사용하는 경우가 있습니다. 아래 CASE를 확인하여 적용하시기 바랍니다.

CASE-1 프로그램 종료 지연으로 인한 업데이트 오류인 경우

프로그램의 용량이 크거나 종료 처리가 길어져서 바로 종료하지 않고 약간의 시간이 지난 후 프로그램이 종료되는 현상이 발생하여 업데이트를 정상 시작할 수 없다면 SetStartInterval 값(지연 시간)을 설정하여 해결합니다.

※ 자세한 내용은 "SetStartInterval 속성"과 "프로그래밍 적용 가이드"를 참고하시기 바랍니다.

```
// 업데이트 시작 지연 시간 설정. 테스트 시 값을 조절하여 Test하시기 바랍니다.
smartUpdate1.SetStartInterval = 3000;
// Update File을 확인하여 업데이트 할 파일이 있을 경우 업데이트 시작
if (smartUpdate1.Start() == true)
{
  Thread.Sleep(200);   // 200ms 동안 쉼
  Application.Exit();  // 현재 응용 프로그램을 종료
}
```

CASE-2 스레드를 호출하는 컴포넌트를 사용하는 경우

SmartX Framework 컴포넌트 중 스레드를 호출하는 컴포넌트가 있으며, 이 컴포넌트를 사용하여 시작하는 메소드를 호출했다면 반드시 폼 클로징에서 종료하는 메소드를 호출해야 합니다. 컴포넌트의 종류 및 호출해야 하는 시작/종료 메소드는 아래 표를 참고하시기 바랍니다.

[표] 스레드를 호출하는 컴포넌트별 시작/종료 메소드

컴포넌트	시작 메소드	종료 메소드
SmartSplash	Start()	Finish()
SmartGPIO	PortXWatchStart()	PortXWatchStop()
SmartSerialPort	Open()	Close()
SmartMemory	Start()	Close()
SmartModbusSlave	Start()	End()
SmartSound	Play()	Stop()
SmartPWM	StartPWMX()	StopPWMX()
SmartWatchDog	Start()	Stop()
SmartVideo	Play()	Stop()
SmartDraw	AnimationGIFPlay()	AnimationGIFStop()
SmartMultiServer	Start()	Stop()
SmartThread	Start()	End()
SmartScreenSaver	Start()	Stop()
SmartPlayer	Play()	Stop()

※ 각 컴포넌트에 해당하는 메소드 내용을 참고하시기 바랍니다.

※ 아래 예시는 SmartGPIO를 사용할 경우 처리해야하는 코드입니다.

```
private void Start_Click(object sender, EventArgs e)
{
  smartGPIO1.PortAWatchStart();  // PORT-A의 입력 상태가 변경되는지 감시를 시작함
}
```

```
// 프로그램 종료 시 반드시 처리해야함
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartGPIO1.PortAWatchStop();  // PORT-A의 입력 상태가 변경되는지 감시를 중단함
}
```

참고 스레드를 직접 생성했을 경우 스레드를 종료하시기 바라며, 그 밖에 시스템 리소스의 사용에 따라서 문제가 발생할 수도 있습니다.

5) SmartUpdate 인터페이스 설명

SmartUpdate Component Interface		
속성		
BackImagefilePathName : string	CompanyText : string	CompanyTextColor : Color
FromFilePath : string	MessageText : string	MessageTextColor : Color
Restart : SmartUpdate.RESTARTFLAG	SetStartInterval : uint	TitleText : string
TitleTextColor : Color	ToFilePath : string	
메소드		
SetFontName(string strFontName) : void	Start() : bool	

프로퍼티(속성) BackImageFilePathName Image

업데이트 진행 시 다이얼로그 창 배경 이미지 파일의 경로 및 이름을 문자열로 설정합니다. (이미지 확장자는 BMP만 가능)

- `string` : 배경 이미지 파일의 경로 및 이름의 문자열

참고 BackImagefilePathName 속성을 사용하지 않을 경우 기본 이미지로 출력됩니다.

배경 이미지 설정(기본 이미지일 경우)

Image 이미지 사용 시 주의사항

사용되는 이미지가 “이미지 제작 가이드”에 맞게 제작되지 않은 경우, 다양한 오류가 발생할 수 있습니다.
※ 반드시 “이미지 제작 가이드”를 참고하여 이미지를 제작하시기 바랍니다.

C# 사용법

```
smartUpdate1.BackImageFilePathName = “Flash Disk₩₩UpBKImg.bmp”;
```

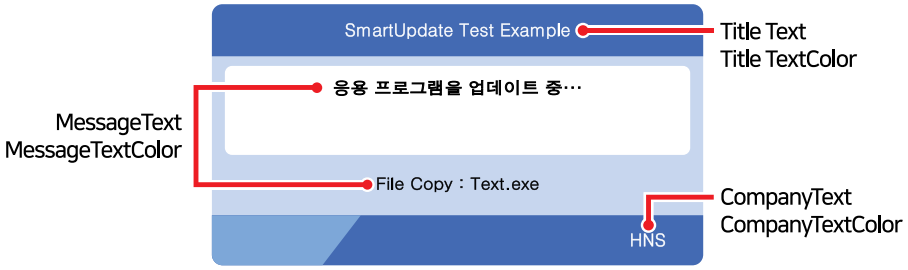
VB 사용법

```
smartUpdate1.BackImageFilePathName = “Flash Disk₩₩UpBKImg.bmp”
```

프로퍼티(속성) CompanyText, CompanyTextColor, MessageText, MessageTextColor, TitleText, TitleTextColor

업데이트 진행 시 화면에 표시될 메시지 텍스트 및 색상을 설정합니다.

- string CompanyText : 하단 우측에 표시할 메시지 텍스트를 설정합니다.
- Color CompanyTextColor : 하단 우측에 표시할 메시지 텍스트의 색상을 설정합니다.
- string MessageText : 중앙에 표시할 설명 메시지 텍스트를 설정합니다.
- Color MessageTextColor : 중앙에 표시할 설명 메시지 텍스트의 색상을 설정합니다.
- string TitleText : 상단 가운데 표시할 메시지 텍스트를 설정합니다.
- Color TitleTextColor : 상단 가운데 표시할 메시지 텍스트의 색상을 설정합니다.



C# 사용법

```
smartUpdate1.TitleText = " SmartUpdate Test Example " ;
smartUpdate1.TitleTextColor = Color.White;
smartUpdate1.CompanyText = " HNS " ;
smartUpdate1.CompanyTextColor = Color.White;
smartUpdate1.MessageText = " 응용 프로그램을 업데이트 중… " ;
smartUpdate1.MessageTextColor = Color.Black;
```

VB 사용법

```
smartUpdate1.TitleText = " SmartUpdate Test Example "
smartUpdate1.TitleTextColor = Color.White
smartUpdate1.CompanyText = " HNS "
smartUpdate1.CompanyTextColor = Color.White
smartUpdate1.MessageText = " 응용 프로그램을 업데이트 중… "
smartUpdate1.MessageTextColor = Color.Black
```

프로퍼티(속성) FromFilePath, ToFilePath

업데이트 경로를 설정합니다.

- string FromFilePath : 업데이트 파일이 있는 경로를 설정합니다.
- string ToFilePath : 업데이트될 대상 파일이 있는 경로를 설정합니다.

참고 FromFilePath 경로의 파일이 기준이며, ToFilePath에 있는 모든 파일을 Checksum으로 비교하여 다른 경우 업데이트합니다. 만약 FromFilePath의 파일이 ToFilePath에 없는 경우 그대로 복사합니다.

C# 사용법

```
// USB Memory → Flash Disk인 경우
smartUpdate1.FromFilePath = " 하드 디스크₩₩UpdateFiles " ;
smartUpdate1.ToFilePath = " Flash Disk₩₩Run " ;
```

VB 사용법

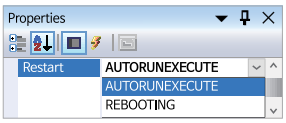
```
smartUpdate1.FromFilePath = " 하드 디스크₩₩UpdateFiles "
smartUpdate1.ToFilePath = " Flash Disk₩₩Run "
```

프로퍼티(속성) Restart

업데이트 완료 후 IEC-Series를 재부팅 또는 업데이트된 새로운 장치 응용 프로그램을 실행할지를 설정합니다.
(단, 업데이트 내용이 없을 경우 해당 사항 없습니다.)

- SmartUpdate.RESTARTFLAG.AUTORUNEXECUTE : 업데이트 완료 후 Run 폴더의 파일(EXE 파일)을 실행합니다.
- SmartUpdate.SmartUpdate.REBOOTING : 업데이트 완료 후 IEC-Series를 재부팅합니다.

사용법



프로퍼티(속성) SetStartInterval

업데이트 시작의 지연 시간을 설정합니다. 업데이트하기 위해서 프로그램이 종료될 경우 프로그램의 용량 및 종료 관련 처리 코드로 인하여 바로 종료되지 않고 약간의 시간이 지난 후 프로그램이 종료됩니다. 업데이트를 정상적으로 시작할 수 없을 경우 설정합니다.

- uint : 업데이트 시작 지연 시간 (단위 : ms)

C# 사용법

```
// 업데이트 시작의 지연 시간을 설정합니다.
smartUpdate1.SetStartInterval = 3000;
```

VB 사용법

```
smartUpdate1.SetStartInterval = 3000
```

메소드(함수) SetFontName

업데이트 시 다이얼로그 창에 표시되는 모든 텍스트의 폰트를 설정합니다.

중요 Start() 메소드 호출 전에 호출해야 합니다.

◉ void SetFontName(string strFontName)

[인자]

- string strFontName : 폰트 이름

주의 "Flash Disk₩₩Fonts" 폴더 또는 "SD Card₩₩Fonts" 폴더에 해당 사용자 폰트가 존재해야 하며, True Type의 ttc, ttf만 가능합니다.

참조 IEC-Series에 탑재된 Font가 아닌 다른 Font를 사용하고 싶은 경우에는 "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 20. IEC-Series에서 다양한 폰트(Fonts 폴더, 트루타입)사용 방법 안내"를 참조하시기 바랍니다.

C# 사용법

```
// 업데이트 시 다이얼로그 창에 표시되는 모든 텍스트의 폰트 설정
smartUpdate1.SetFontName( " 나눔고딕 " );  // Start() 메소드 호출 전에 호출해야 함
smartUpdate1.SetStartInterval = 3000;     // 3초
```

VB 사용법

```
smartUpdate1.SetFontName( " 나눔고딕 " )
smartUpdate1.SetStartInterval = 3000
```

메소드(함수) **Start**

설정된 값에 따라 업데이트를 시작합니다. 업데이트 할 파일이 있는 경우에 Update 모듈이 실행되어 본격적인 업데이트 작업이 시작됩니다.

◉ bool Start()

주의 업데이트 할 파일이 있는 경우 반드시 현재 응용 프로그램을 종료해야 합니다.

[리턴값]

- bool : 업데이트 할 파일 유무
  - true : 있음
  - false : 없음

C# 사용법

```
// Update File을 확인하여 업데이트 할 파일이 있을 경우 업데이트 시작
if (smartUpdate1.Start() == true)
{
  Thread.Sleep(200);   // 200ms 동안 쉼
  Application.Exit();  // 현재 응용 프로그램을 종료
}
```

VB 사용법

```
If smartUpdate1.Start() = True Then
  Thread.Sleep(200)
  Application.Exit()
End If
```

6) SmartUpdate 예제 사용하기

SmartUpdate를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

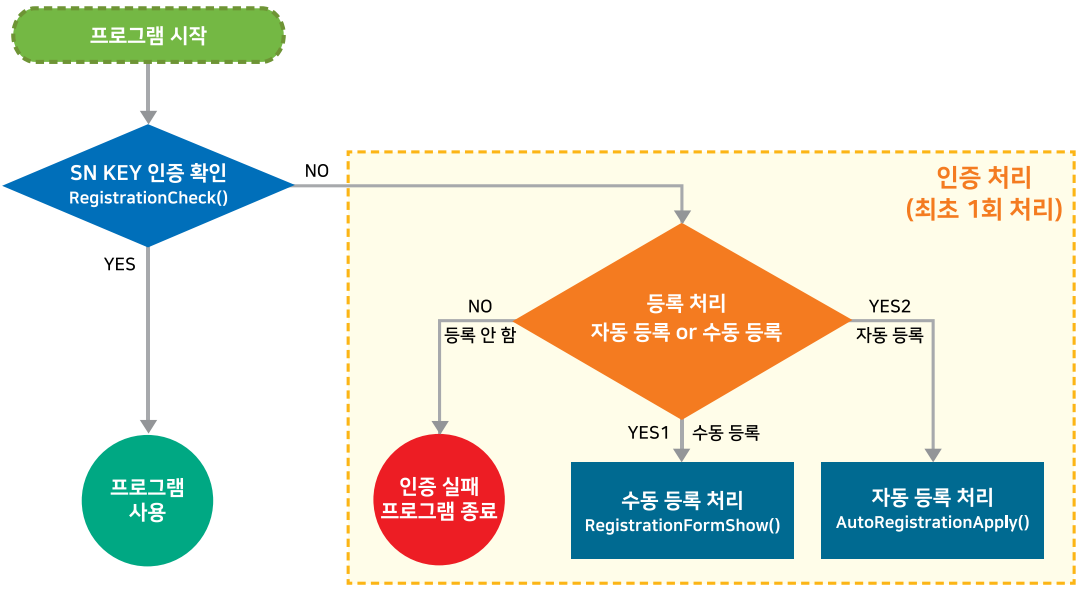
[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → “SmartUpdate”

8. SmartLock

SmartLock은 소프트웨어 방식의 락(Software Lock) 기능을 지원하는 컴포넌트입니다. 개발된 장치 응용 프로그램의 불법 복제 방지 기능을 쉽고 간편하게 적용할 수 있도록 설계되어 있으며, 다음과 같은 특징을 가지고 있습니다.

- 양산 제품에 적용 시 프로그램 등록을 자동으로 처리할 수 있는 기능 제공
  - ↳ 제품 키 입력을 외부 파일로 설정 가능
- 긴급 실행 모드 등의 기능을 구현할 수 있도록 핵심 기능들만 지원
- 인증과 관련한 정책을 쉽게 변경
- 수동 제품 키 입력 기능



1) 제품 키 설정 및 형식 정의 시 주의사항

인증 키(제품 키, Serial Number로도 표기함)는 프로그램의 최초 실행 시 프로그램을 등록하기 위해서 사용되는 키로, 일정 형식을 갖는 형태로 정의하여 사용할 수 있도록 관리하시기 바랍니다.
실행 프로그램 내부에 정의된 제품 키들과 등록 시 입력된 제품 키의 동일 여부에 따라서 인증을 수행하며, 인증은 최초 실행 시 1회만 처리됩니다. 등록이 완료되면 다음 실행부터는 정상 실행되도록 처리합니다. 아래의 제품 키의 형식 및 설정 시 주의사항을 참고하여 제품 키의 형식 정의 및 설정을 하시기 바랍니다.

주의 제품 키의 형식 및 설정 시 주의사항

1. 제품 키는 고정 길이로만 설정
제품 키를 여러 개 입력 시, 가변 길이가 아닌 고정 길이로만 설정해야 합니다. 즉, ReferenceSNKeys 속성에 제품 키를 여러 개 설정하는 경우 각 제품 키의 크기를 모두 같게 설정하시기 바랍니다.

옳은 예	`smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002;";`
잘못된 예	`// 첫번째 제품 키와 두번째 제품 키의 길이가 다름` `smartLock1.ReferenceSNKeys = "000-000-000-001;0000-0000-0000-0002;";`

2. 제품 키 마지막에 세미콜론 추가
제품 키 입력 시 세미콜론에 주의하셔야 합니다. 즉, ReferenceSNKeys 속성에 제품 키 설정 시, 제품 키의 개수와 상관없이 각 제품 키의 마지막에 ';'(세미콜론)을 추가하셔야 합니다.

옳은 예	smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;"; smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002;";

잘못된 예	// 세미콜론이 없음
	smartLock1.ReferenceSNKeys = "0000-0000-0000-0001";
	smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002";
	// 세미콜론이 앞과 뒤에 사용됨
	smartLock1.ReferenceSNKeys = ";0000-0000-0000-0001";
	smartLock1.ReferenceSNKeys = ";0000-0000-0000-0001;0000-0000-0000-0002";

3. 제품 키 구성 시 주의사항
제품 키 입력 시 문자 입력에 주의해야 합니다.
3-1. 사용 가능한 문자는 0~9(숫자), A~Z(대문자), -(기호)이며, 각 제품 키의 구분 문자는 ';'(세미콜론)입니다. 또한, 공백 문자를 주의하시기 바랍니다.
3-2. 하나의 키가 20자를 넘을 수 없습니다. "-" 포함 5~19자리로 설정해야 하며, 200개 미만의 데이터를 사용하시기 바랍니다. 또한 각 제품 키의 마지막에는 반드시 ';'(세미콜론)을 추가해야 합니다.

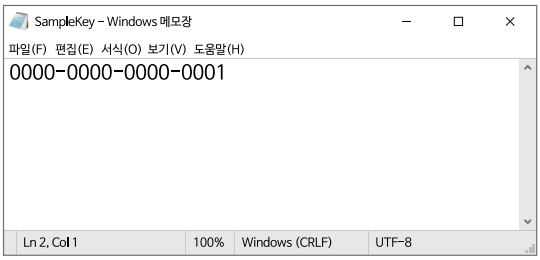
옳은 예	smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;"; smartLock1.ReferenceSNKeys = "A000-B000-C000-D001;0000-0000-0000-0002;";

잘못된 예	// 제품 키가 20자를 넘음
	smartLock1.ReferenceSNKeys = "0000-0000-0000-0001233";
	// 제품 키에 한글이 포함됨
	smartLock1.ReferenceSNKeys = "가나다라-0000-0000-0001";
	// 제품 키에 특수문자가 포함됨
	smartLock1.ReferenceSNKeys = "!@#$-0000-0000-0001;*&A4-0000-0000-0002";

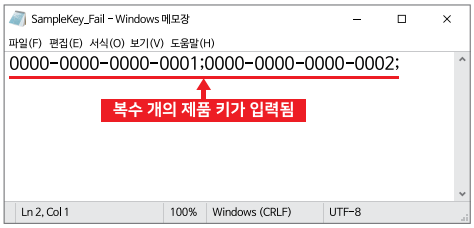
4. 자동 등록 파일 사용 시 주의사항
AutoRegistrationApply() 메소드를 사용해 제품 키 자동 등록 기능을 사용할 때 자동 등록 파일 안에는 반드시 올바른 형식의 제품 키 하나만 존재해야 합니다. 그렇지 않은 경우 제품 키 등록이 되지 않습니다. 또한, ';'(세미콜론)을 제외한 제품 키만이 저장되어 있어야 하며, 엔터 키, 띄어쓰기, 잘못된 키 등이 입력되면 등록이 되지 않습니다.

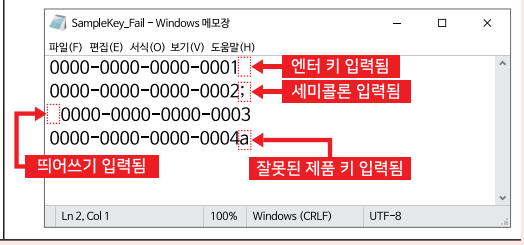
제품 키	smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;";

옳은 예



잘못된 예





2) 프로그래밍 적용 가이드

STEP-1 제품 키 형식 및 운영 방식 정의하기

제품 키의 형식을 정의하며 그에 따른 운영 방식을 정의합니다. 제품 키의 형식은 “제품 키 설정 및 형식 정의 시 주의사항” 내용을 참고하여 정의하시기 바랍니다.
운영 방식은 실행 프로그램에 따른 입력 가능한 제품 키의 정의를 어떻게 할 것인지에 대한 방식이며, 아래와 같이 사전에 정의하시기 바랍니다.

[운영 방식]
1. 실행 파일(프로그램)별 여러 개의 제품 키를 포함하는 경우
2. 실행 파일(프로그램)별 하나의 제품 키를 포함하는 경우
3. 제품 키 생성 로직에 의하여 랜덤하게 처리되는 경우

STEP-2 인증 확인하기

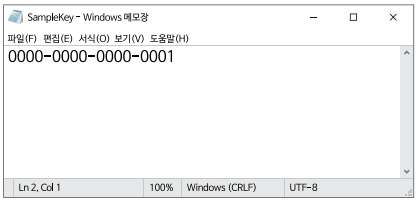
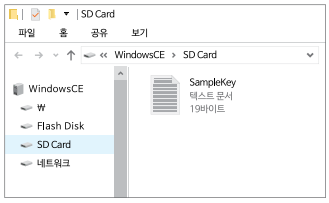
프로그램 실행 후 RegistrationCheck() 메소드를 호출해 프로그램이 인증이 되었는지 확인합니다. 인증이 되었다면 프로그램을 실행하도록 하며, 그렇지 않은 경우 다음 STEP으로 넘어가도록 코드를 작성합니다.

※ 자세한 내용은 “RegistrationCheck() 메소드”를 참고하시기 바랍니다.

```
private void Form1_Load(object sender, EventArgs e)
{
  // 프로그램 실행 시 해당 프로그램이 인증되었는지 확인
  if (smartLock1.RegistrationCheck() == false)
  {
    // 인증되지 않은 경우, 자동 또는 수동 등록을 시도합니다.
    // STEP-3 예제 코드
  }
  // 인증이 완료되어 프로그램을 실행합니다.
}
```

STEP-3 등록 처리하기

STEP-2에서 인증되지 않은 프로그램으로 확인된 경우, 자동 또는 수동으로 제품 키를 입력받아야 합니다.
이 경우 STEP-1에서 정의한 운영 방식에 따라 프로그램상에서 등록 가능한 제품 키를 ReferenceSNKeys 속성으로 설정해야 합니다. 만약, 자동으로 등록을 처리한다면, 제품 키가 저장된 파일이 필요하며 해당 파일을 메모리에 복사하여 IEC-Series에서 접근이 가능하도록 합니다.

제품 키가 저장된 파일 예시	메모리에 복사
	

※ 자세한 내용은 “ReferenceSNKeys 속성”을 참고하시기 바랍니다.

[CASE-1] 자동 등록 처리하기

자동 등록 처리를 위해서는 먼저 제품 키가 저장된 파일 경로를 AutoRegistrationFilePath 속성에서 설정합니다. 그 다음 AutoRegistrartionApply() 메소드를 호출해 자동 등록을 시도합니다. 자동 등록에 성공했다면 프로그램을 실행하도록 하며, 그렇지 않은 경우 다음 STEP으로 넘어가도록 코드를 작성합니다.

※ 자세한 내용은 “AutoRegistrationFilePath 속성”, “AutoRegistrartionApply() 메소드”를 참고하시기 바랍니다.

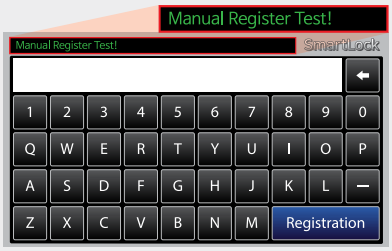
```
// 인증 가능한 제품 키들을 설정합니다. 마지막에 구분자 ';'(세미콜론)을 꼭 넣어주시기 바랍니다.
smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002;0000-0000-0000-0003;";
// 자동 등록을 위한 제품 키가 저장된 경로를 설정합니다.
smartLock1.AutoRegistrationFilePath = "SD Card₩₩SampleKey.txt";

// 자동 등록을 시도하며 결과를 등록 결과를 확인합니다.
if (smartLock1.AutoRegistrationApply() != SmartX.SmartLock.REGISTRATIONSTATUS.SUCESS)
{
  // 정상 등록에 실패한 경우, 프로그램을 종료하거나, 특정 동작만을 수행하도록 합니다.
  // STEP-4 예제 코드
}
```

[CASE-2] 수동 등록 처리하기

수동 등록 처리를 위해서는 먼저 TitleMsg 속성으로 수동 입력창에 출력되는 메시지를 변경합니다. 메시지 변경이 완료되면 RegistrationFormShow() 메소드를 호출하여 수동 입력창을 출력합니다. 키 입력 후 Registration 버튼을 클릭하여 수동으로 제품 키를 등록할 수 있습니다. 수동 등록에 성공했다면 프로그램을 실행하도록 하며, 그렇지 않은 경우 다음 STEP으로 넘어가도록 코드를 작성합니다.

※ 자세한 내용은 "TitleMsg 속성", "RegistrationFormShow() 메소드"를 참고하시기 바랍니다.

```
// 인증 가능한 제품 키들을 설정합니다. 마지막에 구분자 ';'(세미콜론)을 꼭 넣어주시기 바랍니다.
smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002;0000-0000-0000-0003;";
// 수동 등록창에 보여질 메시지를 설정합니다.
smartLock1.TitleMsg = "Manual Register Test!";

// 수동 등록을 시도합니다. 메소드 호출 시 수동 등록창이 출력되어 제품 키를 입력할 수 있습니다.
if (smartLock1.RegistrationFormShow() != SmartX.SmartLock.REGISTRATIONSTATUS.SUCESS)
{
  // 수동 등록에 실패하여 프로그램 종료 처리 또는 등록 실패에 따른 처리 코드를 작성합니다.
  // STEP-4 예제 코드
}
```



[수동 입력창 출력 예시]

STEP-4 등록 실패 처리하기

위 STEP을 진행하여 자동 및 수동 등록에 실패한 경우, 인증되지 않은 프로그램이라고 판단할 수 있습니다. 이럴 경우 프로그램을 사용할 수 없도록 프로그램을 종료하거나, 특정 동작만을 수행하도록 코드를 작성하셔야 합니다.

```
Application.Exit();  // 자동 및 수동 등록에 실패한 경우 프로그램을 종료합니다.
// 또는 특정 동작만을 수행하도록 코드를 작성합니다.
while (true)
{
  SmartX.SmartMessageBox.Show("인증되지 않은 프로그램입니다!!");
}
```

3) SmartLock 인터페이스 설명

SmartLock Component Interface		
속성		
AutoRegistrationFilePath : string	ReferenceSNKeys : string	TitleMsg : string
메소드		
AutoRegistrationApply() : SmartLock.RegistrationStatus	RegistrationCheck() : bool	RegistrationFormShow() : SmartLock.RegistrationStatus

프로퍼티(속성) AutoRegistrationFilePath

제품 키의 자동 등록 처리를 위해 제품 키가 입력된 파일의 경로와 파일명을 설정합니다.

참고 "1) 제품 키 설정 및 형식 정의 시 주의사항", "AutoRegistrationApply() 메소드" 내용을 참고하시기 바랍니다.

• string : 제품 키가 입력되어 있는 파일 경로

C# 사용법

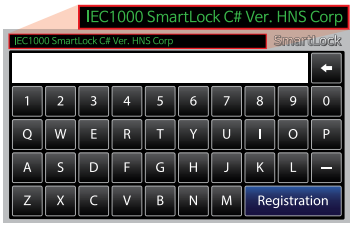
```
smartLock1.AutoRegistrationFilePath = " SD Card₩₩Run₩₩lock.lic " ;  // 제품 키가 입력된 파일 경로
```

VB 사용법

```
smartLock1.AutoRegistrationFilePath = " SD Card₩₩Run₩₩lock.lic "
```

프로퍼티(속성) TitleMsg

제품 키를 수동으로 입력하기 위한 입력창 상단의 메시지 내용을 설정합니다.



참고 "RegistrationFormShow() 메소드" 내용을 참고하시기 바랍니다.

• string : 입력창 상단 메시지

C# 사용법

```
smartLock1. TitleMsg = " IEC1000 SmartLock C# Ver. HNS Corp " ;  // 입력창 상단 메시지 설정
```

VB 사용법

```
smartLock1. TitleMsg = " IEC1000 SmartLock C# Ver. HNS Corp "
```

프로퍼티(속성) ReferenceSNKeys

프로그램의 등록을 위해 사용할 제품 키들을 설정합니다.

중요 ReferenceSNKeys 속성값 설정 시 중요사항

1. 사용 가능한 문자는 0~9(숫자), A~Z(대문자), -(기호)이며, 각 제품 키의 구분 문자는 ';'(세미콜론)입니다. 또한, 공백 문자를 주의하시기 바랍니다.
2. 하나의 키가 20자를 넘을 수 없습니다. "-" 포함 5~19자리로 설정해야 하며, 200개 미만의 데이터를 사용하시기 바랍니다. 또한 각 제품 키의 마지막에는 반드시 ';'(세미콜론)을 추가해야 합니다.

참고 "제품 키 설정 및 형식 정의 시 주의사항" 내용을 참고하시기 바랍니다.

• string : 제품 키

C# 사용법

```
// 제품 키 한 개 등록 (마지막에 구분자 ';'(세미콜론)을 꼭 넣어주시기 바랍니다.)
smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;"
// 제품 키 한 개 이상 등록 (마지막에 구분자 ';'(세미콜론)을 꼭 넣어주시기 바랍니다.)
smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002;"
```

VB 사용법

```
smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;"
smartLock1.ReferenceSNKeys = "0000-0000-0000-0001;0000-0000-0000-0002;"
```

메소드(함수) AutoRegistrationApply

AutoRegistrationFilePath 속성에서 설정된 경로의 파일을 읽어 제품 키를 등록합니다

참고 "AutoRegistrationFilePath 속성" 내용을 참고하시기 바랍니다.

◉ SmartLock.REGISTRATIONSTATUS AutoRegistrationApply()

[리턴값]

• SmartLock.REGISTRATIONSTATUS : 제품 키 등록 결과
  – SmartLock.REGISTRATIONSTATUS.SUCESS : 등록 완료 (성공)
  – SmartLock.REGISTRATIONSTATUS.INVALID_PATHNAME : 제품 키 파일이 지정된 경로에 없는 경우 (실패)
  – SmartLock.REGISTRATIONSTATUS.INVALID_SNKEY : 잘못된 제품 키 입력 (실패)
  – SmartLock.REGISTRATIONSTATUS.REGISTRATION_FAIL : 제품 키 등록 오류 (실패)

C# 사용법

```
if (smartLock1.AutoRegistrationApply() != SmartLock.REGISTRATIONSTATUS.SUCESS)
{
  // 자동 등록 실패에 따른 처리 코드를 작성
}
```

VB 사용법

```
If smartLock1.AutoRegistrationApply() <> SmartLock.REGISTRATIONSTATUS.SUCESS Then
End If
```

메소드(함수) RegistrationCheck

프로그램의 정상 등록 여부를 확인합니다.

◉ bool RegistrationCheck()

[리턴값]

• bool : 등록 여부
  – true : 등록됨
  – false : 등록되지 않음

C# 사용법

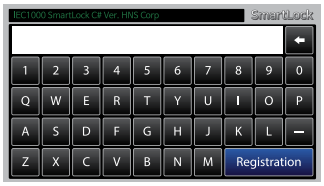
```
if (smartLock1.RegistrationCheck() == false)
{
  // 정상 등록되지 않은 경우 처리 코드 작성
}
```

VB 사용법

```
If smartLock1.RegistrationCheck() = False Then
End If
```

메소드(함수) RegistrationFormShow

수동으로 프로그램을 등록하기 위해 제품 키 입력창을 출력합니다.



◉ SmartLock.REGISTRATIONSTATUS RegistrationFormShow()

[리턴값]

- SmartLock.REGISTRATIONSTATUS : 제품 키 등록 결과
  - SmartLock.REGISTRATIONSTATUS.SUCESS : 등록 완료 (성공)
  - SmartLock.REGISTRATIONSTATUS.INVALID_PATHNAME : 제품 키 파일이 지정된 경로에 없는 경우 (실패)
  - SmartLock.REGISTRATIONSTATUS.INVALID_SNKEY : 잘못된 제품 키 입력 (실패)
  - SmartLock.REGISTRATIONSTATUS.REGISTRATION_FAIL : 제품 키 등록 오류 (실패)

C# 사용법

```
// 수동 등록 성공
if(smartLock1.RegistrationFormShow() == SmartLock.REGISTRATIONSTATUS.SUCESS) {  }
```

VB 사용법

```
If smartLock1.RegistrationFormShow() = SmartLock.REGISTRATIONSTATUS.SUCESS Then
End If
```

4) SmartLock 예제 사용하기

SmartLock을 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartLock"

9. SmartFileSetting

SmartFileSetting은 IEC-Series를 사용하여 제품 개발이 완료되어 생산부서로 이관될 때 유용하게 사용되는 컴포넌트가 아닌 프로그램입니다. 장치 관련 실행 환경의 자동 일괄 배포 기능을 구현할 수 있어 배포 시스템을 체계화할 수 있습니다. 제품 생산에 따른 IEC-Series에서 여러 가지 설정해야 하는(아래 특징 참고) 사항들을 일괄적으로 자동 처리되어 생산 시 편리함과 작업자의 실수를 방지할 수 있습니다. 본 기능을 A/S 처리 부서에서도 사용할 수 있도록 하시면 실행 환경 배포를 체계화할 수 있습니다.

- SD Card에서 Flash Disk₩Run 폴더로 실행 파일 복사(하위 폴더 및 모든 파일)
- Font 폴더의 Font 구성 (Font는 TureType의 ttc, ttf만 가능)
- 바탕화면 이미지 적용
- BootLogo 이미지 적용
- 기타 사용자가 원하는 폴더 복사

참조 SmartFileSetting의 동영상 자료는 "홈페이지 → 공지사항 → 92. SmartFileSetting 프로그램 지원 안내"를 참조하시기 바랍니다.

1) SmartFileSetting 사용하기

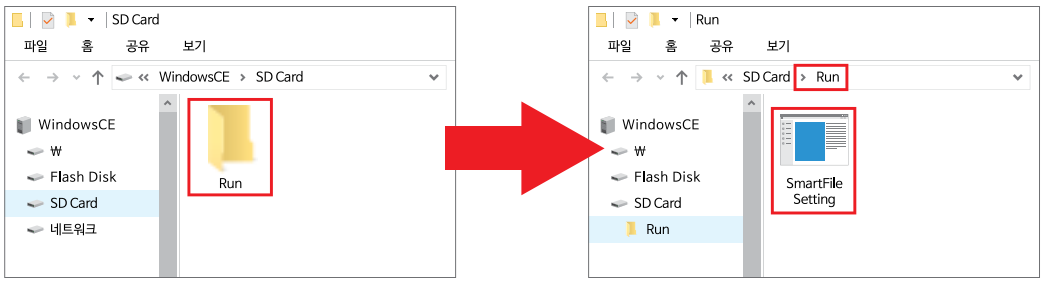
SmartFileSetting을 사용하기 앞서 "SmartFileSetting 프로그램"을 다운로드합니다.

다운로드 경로 ▶ 자사 홈페이지 → 자료실 → SmartX Framework 관련 → "SmartFileSetting 프로그램"

중요 SmartFileSetting을 사용하기 위해서는 SD Card 메모리를 사용해야 합니다.

STEP-1 SmartFileSetting.exe 파일 복사하기

SD Card에 Run 폴더를 생성한 뒤 Run 폴더 안에 다운로드한 SmartFileSetting.exe 파일을 복사해 넣어줍니다.



주의 Run 폴더 내부에 SmartFileSetting.exe 파일 이외의 응용 프로그램 파일이 있으면 RunTime 모드로 부팅 시 SmartFileSetting 프로그램이 아닌 다른 프로그램이 실행될 수 있습니다.

STEP-2 Run 폴더에 하위 폴더 생성하기

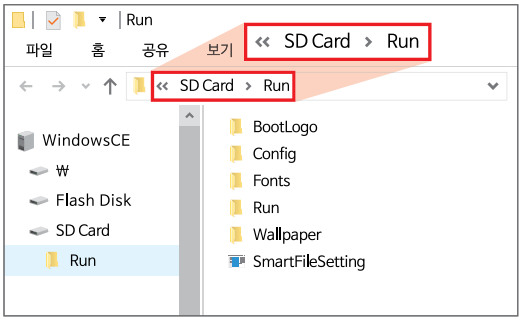
Flash Disk 폴더에 복사될 폴더들을 생성합니다. 아래 표의 폴더명을 참고하시어 동일한 이름의 폴더를 생성하시기 바랍니다. 만약 특정 기능을 사용하고 싶지 않은 경우 해당 기능의 폴더를 생성하지 않습니다.

[표] 하위 폴더명에 따른 기능

폴더명	해당 폴더의 기능(역할)
Run	Flash Disk의 Run 폴더에 복사될 폴더 및 응용 프로그램 파일 원본 위치
Fonts	Flash Disk의 Fonts 폴더에 복사될 폰트 파일 원본 위치

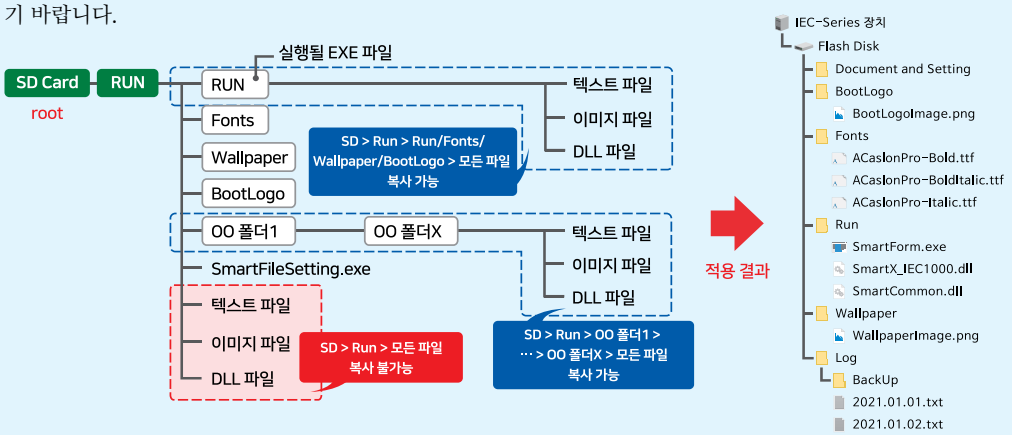
Wallpaper	IEC-Series의 윈도우 배경화면에 적용될 원본 이미지
BootLogo	IEC-Series 부팅 시 BootLogo 화면을 변경할 원본 이미지
XXX (기타 폴더명)	기타 Flash Disk에 복사를 원하는 폴더들

주의 폴더명은 지정된 이름으로 철자가 틀리지 않도록 동일하게 작성하시기 바랍니다. (폴더명은 대소문자 구분 안 함)



참고 SD Card에서 Flash Disk로 파일 복사가 가능한 경로와 불가능한 경로

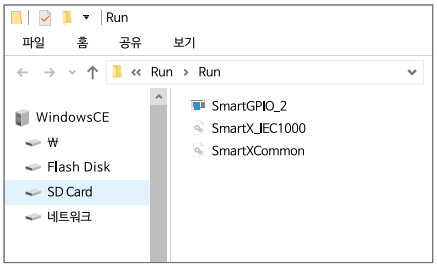
SD Card에서 Flash Disk로 파일 복사가 가능한 경로도 있지만 불가능한 경로도 있습니다. 복사할 파일이 각 기능에 해당하는 SD Card₩Run 하위 폴더 안에 없고 나와 있다면 복사가 불가능합니다. 아래 그림을 참고하시기 바랍니다.



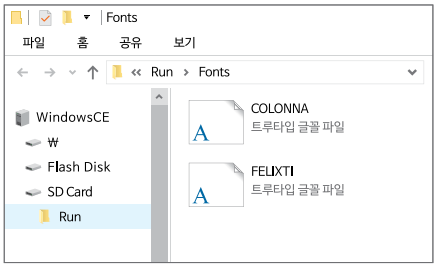
STEP-3 생성된 각각의 폴더 안에 해당하는 파일 복사하기

[STEP-2]에서 생성된 폴더 안에 각 기능에 해당하는 파일들을 복사합니다.

Run 폴더 (SD Card₩Run₩Run)	Fonts 폴더 (SD Card₩Run₩Fonts)

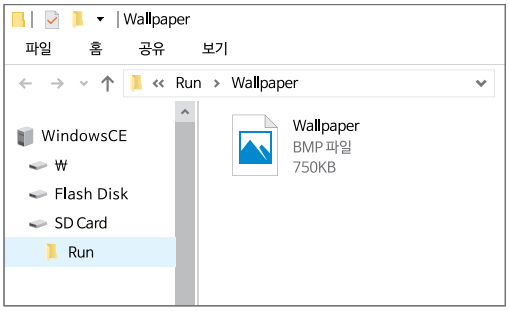


IEC-Series에서 실행될 Run 폴더에 위치한 EXE/DLL 파일과 폴더를 복사 (실행 파일(exe), DLL 파일, 기타 파일)



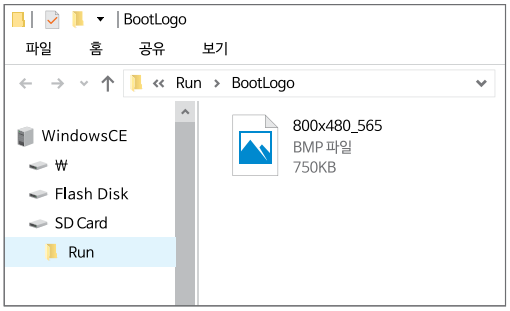
시스템에서 사용할 Font 파일 복사 (TrueType의 ttc,ttf만 가능)

Wallpaper 폴더 (SD Card₩Run₩Wallpaper)



윈도우 바탕화면 배경 이미지 복사

BootLogo 폴더 (SD Card₩Run₩BootLogo)

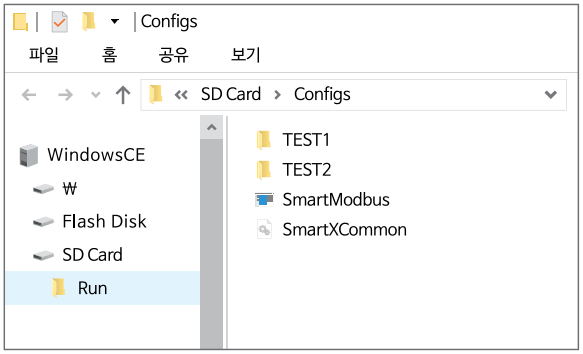


부팅 시 BootLogo에 사용할 이미지 복사

주의 Run 폴더 내부에 SmartFileSetting.exe 파일 이외의 응용 프로그램 파일이 있으면 RunTime 모드로 부팅 시 SmartFileSetting 프로그램이 아닌 다른 프로그램이 실행될 수 있습니다.

참고 BootLogo 이미지에 대한 자세한 설명은 SmartBootLogo를 참고하시기 바랍니다.

기타 폴더 (SD Card₩Run₩기타 폴더명)



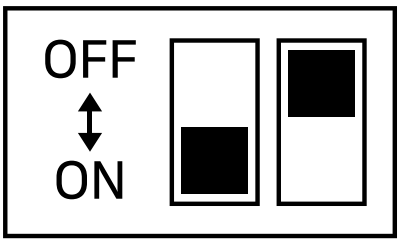
기타 제품 구동에 필요한 기타 폴더 및 파일들 복사

STEP-4 제품에 적용하기

SmartFileSetting은 경우에 따라 제품의 딥(Dip) 스위치 1번 설정과 SmartFileSetting 프로그램 실행 방식이 다르므로 아래 CASE를 참고하시기 바랍니다.

[CASE-1] 제품을 생산하는 경우 (처음 설치 시)

SD Card를 제품에 삽입한 뒤 제품의 딥(Dip) 스위치 1번을 ON으로 설정하여 RunTime 모드로 부팅되도록 합니다. 제품 부팅 후 자동으로 SD Card₩Run 폴더에 있는 SmartFileSetting 프로그램이 실행됩니다. SmartFile Setting 프로그램이 실행되면 SmartFileSetting에 의해서 기능들의 파일과 폴더가 복사되고 IEC-Series 제품에 해당 기능이 적용됩니다.

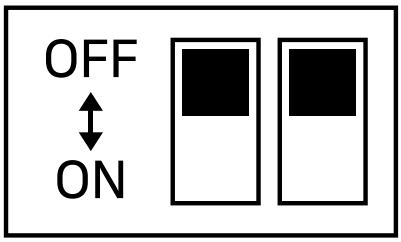


[CASE-2] A/S가 발생하거나 업데이트하는 경우 (Flash Disk/Run 폴더에 이미 실행 파일이 복사된 경우)

SD Card를 제품에 삽입한 뒤 제품의 딥(Dip) 스위치 1번을 OFF로 설정하여 Development 모드로 부팅되도록 합니다. 부팅 후 윈도우 CE 바탕화면이 나오면 "내 장치" – "SD Card" – "Run" 폴더 안에 있는 SmartFileSetting.exe 파일을 클릭하여 직접 실행합니다. 이 후 SmartFileSetting에 의해서 기능들의 파일과 폴더가 복사되고 IEC-Series 제품에 해당 기능이 적용됩니다.

주의 RunTime 모드로 부팅 시 주의사항

딥(Dip) 스위치 1번을 ON으로 설정하여 RunTime 모드로 부팅 시 제품의 Flash Disk₩Run 폴더에 실행 파일이 있으므로 SD Card의 SmartFileSetting 프로그램보다 먼저 실행될 수 있습니다.



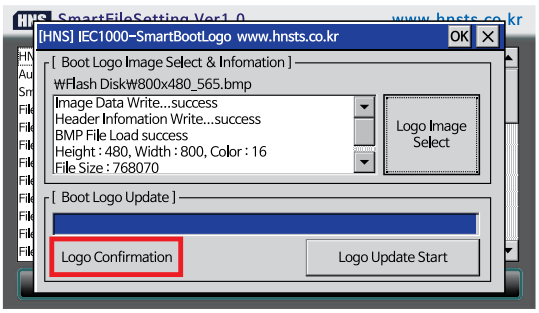
주의 SD Card로부터 동일한 Font가 Flash Disk의 Fonts 폴더에 존재하고 사용 중이라면 해당 Font는 복사가 되지 않습니다.

[그림]SmartFileSetting 실행되는 모습

참조 딥(Dip) 스위치 관련하여 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 동작 모드 관련"을 참조하시기 바랍니다.

STEP-5 모든 설정이 적용됐는지 확인하기

모든 폴더가 복사되면 SmartBootLogo가 자동으로 실행되며, BootLogo가 적용됩니다. "Logo Confirmation" 버튼을 클릭하여 제품을 재부팅합니다. 재부팅 후 IEC-Series의 기능 적용과 파일 복사가 정상적으로 되었는지 확인합니다.



10. SmartThread

SmartThread는 기존의 Thread 프로그램 시 코딩을 최소화하여 Thread의 구현을 직관적으로 처리할 수 있도록한 컴포넌트입니다.

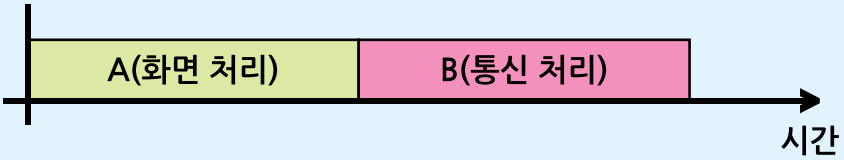
- 이벤트 형식으로 Thread 루틴을 등록하여 구현의 편리함
- Thread의 다양한 종료 기능 제공 및 명확한 처리
- Work Thread와 UI Thread 기능 제공
- Work Thread와 UI Thread의 장점을 취합한 기능 제공

참고 Thread란?

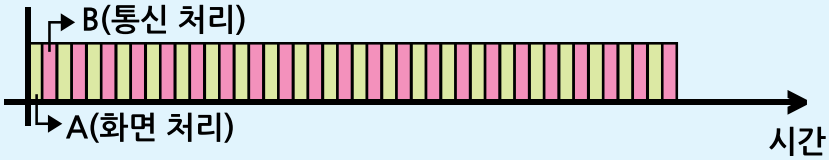
Thread는 특정 프로세스를 아주 작게 쪼갠 단위로, 우선 순위에 따라 CPU에 스케줄링하여 여러 개의 작게 쪼갠 프로세스를 순차적으로 처리하여 동시 처리되는 것처럼 효과를 줄 수 있습니다. Thread는 다음과 같은 경우에 주로 사용됩니다.

1. 화면 처리와 I/O의 입력 처리를 분리하여 처리하고자 할 경우
2. 여러 개의 처리를 동시에 처리하고자 할 경우

특히, IEC-Series는 단일 코어 CPU로, 여러 작업을 동시에 처리하는 경우 특정 작업이 끝날 때까지 다른 작업은 대기하게 되어 CPU의 효율적인 처리 및 시스템의 응답성을 저해하게 될 수 있습니다. 이때 Thread를 사용하게 되면 아래 그림처럼 작업을 분할 처리하게 되어 이러한 현상을 개선할 수 있습니다.



※ 단일 코어 CPU에서 Thread를 사용하지 않는 경우



위 그림처럼 A처리와 B처리가 교차되어 실행하여 마치 A와 B가 동시에 실행되는 효과를 얻을 수 있습니다.

1) Work Thread와 UI Thread의 차이

SmartThread는 사용처에 따라 Work Thread와 UI Thread 두 가지로 나눌 수 있습니다. Work Thread는 빠르고 반복적인 연산 처리가 요구될 경우 사용하지만 사용자 인터페이스(UI)에 직접 접근이 불가능합니다. UI Thread는 연산 횟수가 적으며 UI 갱신이 요구될 경우 사용하지만 비교적 속도가 느립니다. Work Thread와 UI Thread의 차이점은 아래의 표를 참고하시기 바랍니다.

[표] Work Thread와 UI Thread 비교

Thread	Work Thread	UI Thread
장점	UI Thread에 비해 고속	Work Thread에 비해 저속
단점	화면 UI 컨트롤에 직접 접근 불가능	화면 UI 컨트롤에 직접 접근 가능
관련 이벤트	ThreadFunction	UIThreadFunction

※ “ThreadFunction, UIThreadFunction 이벤트” 내용을 참고하시기 바랍니다.

2) SmartThread가 사용되는 형태

SmartThread는 원칙적으로 한 가지의 Thread만 사용할 수 있습니다. Work Thread는 사용자 인터페이스(UI)를 갱신하지 않고 많은 데이터를 빠르고 반복적인 연산하는 경우 사용되며, UI Thread는 적은 데이터를 느리게 연산하여 조건에 따라 사용자 인터페이스를 갱신할 경우 사용됩니다. 또한 Work Thread와 UI Thread의 장점만을 조합하여 많은 데이터를 빠르고 반복적이게 연산하고 사용자 인터페이스를 갱신해야할 경우에도 간단하게 사용할 수 있습니다.

CASE-1 Work Thread

Work Thread는 많은 데이터를 빠르고 반복적인 연산을 할 수 있지만 사용자 인터페이스(UI)에 접근할 수 없습니다. 이러한 특성상 Work Thread는 주로 데이터 처리를 백그라운드에서 처리할 경우 사용됩니다.

CASE-2 UI Thread

UI Thread는 사용자 인터페이스(UI)에 접근할 수 있지만 연산 처리 속도는 Work Thread에 비교적 느립니다. 이러한 특성상 UI Thread는 적은 데이터 또는 속도가 중요하지 않을 경우 취급하며, 데이터 처리에 따른 사용자 인터페이스의 접근이 필요할 경우 사용됩니다.

CASE-3 Work Thread + UI Thread

Work Thread의 빠른 연산 처리 속도와 UI Thread의 사용자 인터페이스(UI)에 직접적으로 접근할 수 있다는 각각의 장점을 조합하여 사용할 수 있습니다. ReportProgress 함수를 호출하여 UI Thread를 간접적으로 호출합니다.

주의 SmartThread 사용 시 주의사항

SmartThread는 원칙적으로 한 가지의 Thread를 사용할 수 있습니다. 연산 처리를 빠르게 함과 동시에 데이터 처리에 따른 사용자 사용자 인터페이스의 접근이 필요한 경우 반드시 [CASE-3]의 방법으로 구현하시기 바랍니다.

3) 프로그래밍 적용 가이드

CASE-1 Work Thread 사용하기

ThreadFunction 이벤트에 코드 작성 후, Start() 메소드를 호출하면 됩니다. Thread를 종료하고 싶은 경우, 이벤트의 인자 args의 bEnd 인자값을 True로 설정하거나, End() 메소드를 호출하여 종료할 수 있습니다. 이벤트가 종료되면 OnEnding 이벤트가 발생합니다.

※ 자세한 내용은 “1) Work Thread와 UI Thread의 차이”, “2) SmartThread가 사용되는 형태”, “ThreadFunction, OnEnding 이벤트”, “Start(), End() 메소드”를 참고하시기 바랍니다.

```
private int[] m_iWorkDatas;  // Work Thread에서 작업할 배열 1
private string[] m_strRecvDatas;  // Work Thread에서 작업할 배열 2
private void WorkThreadStart()
{
  // Work Thread 시작
  smartThread1.Start();
}
private void smartThread1_ThreadFunction(SmartX.ThreadArgs args)
{
  // Work Thread는 주로 데이터의 처리를 백그라운드에서 처리할 경우 사용
  // 수신 받은 데이터 배열의 크기만큼 배열 생성
  m_iWorkDatas = new int[m_strRecvDatas.Length];
  // 문자열 배열을 정수형 배열로 변환하여 저장
```

```
  for(int i = 0; i < m_iWorkDatas.Length; i++)
  {
    m_iWorkDatas[i] = Convert.ToInt32(m_strRecvDatas[i]);
  }
  // Work Thread를 다시 처리할 경우 bEnd 인자를 False로 설정
  // bEnd 인자의 default 값은 false이므로 생략 가능
  args.bEnd = false;
}
```

CASE-2 UI Thread 사용하기

UIThreadFunction 이벤트에 코드 작성 후, Start() 메소드를 호출하면 됩니다. Thread를 종료하고 싶은 경우, 이벤트의 인자 args의 bEnd 인자값을 True로 설정하거나, End() 메소드를 호출하여 종료할 수 있습니다. 이벤트가 종료되면 OnEnding 이벤트가 발생합니다.

※ 자세한 내용은 "1) Work Thread와 UI Thread의 차이", "2) SmartThread가 사용되는 형태", "UIThreadFunction, OnEnding 이벤트", "Start(), End() 메소드"를 참고하시기 바랍니다.

```
private void UIThreadStart()
{
  // UI Thread 시작
  smartThread1.Start();
}
private void smartThread1_UIThreadFunction(SmartX.ThreadArgs args)
{
  // UI Thread는 데이터 처리에 따른 사용자 인터페이스의 접근이 발생할 경우 주로 사용
  if (smartFTP1.Percentage != -1)
  {
    // SmartFTP 업/다운로드 진행 상태를 SmartProgressBar에 표시
    smartProgressBar1.Value = smartFTP1.Percentage;
  }
  else if (smartFTP1.Percentage == 100)
  {
    // 업/다운로드 완료 시 라벨에 표시
    lblStatus.Text = " Complete! ";
    // UI Thread를 종료할 경우 bEnd 인자를 ture로 설정
    args.bEnd = true;
  }
}
```

CASE-3 Work Thread + UI Thread 사용하기

UI Thread를 간접적으로 호출하기 위해 WorkerReportsProgress 속성을 true로 설정합니다. ThreadFunction 이벤트에 연산 처리하는 코드 작성합니다. 처리한 데이터에 따른 사용자 인터페이스(UI) 접근이 필요하므로 ReportProgress 메소드를 ThreadFunction 이벤트 코드에서 호출하여 사용자 인터페이스(UI)의 갱신을 하기 위한 UIThreadFunction 이벤트를 간접 호출합니다. Thread를 종료하고 싶은 경우, 이벤트의 인자 args의 bEnd 인자값을True로 설정하거나, End() 메소드를 호출하여 종료할 수 있습니다. 이벤트가 종료되면 OnEnding 이벤트가 발생합니다.

※ 자세한 내용은 "1) Work Thread와 UI Thread의 차이", "2) SmartThread가 사용되는 형태", "ThreadFunction, UIThreadFunction, OnEnding 이벤트", "WorkerReportsProgress 속성", "ReportProgress(), Start(), End() 메소드"를 참고하시기 바랍니다.

```
private int[] m_iWorkDatas;  // Work Thread에서 작업할 배열 1
private string[] m_strRecvDatas;  // Work Thread에서 작업할 배열 2
```

```
private void WorkThreadStart()
{
  // UI Thread 간접 호출 활성화
  smartThread1.WorkerReportsProgress = true;
  // Work Thread 시작
  smartThread1.Start();
}

private void smartThread1_ThreadFunction(SmartX.ThreadArgs args)
{
  // 수신 받은 데이터 배열의 크기만큼 배열 생성
  m_iWorkDatas = new int[m_strRecvDatas.Length];

  // 문자열 배열을 정수형 배열로 변환하여 저장
  for(int i = 0; i < m_iWorkDatas.Length; i++)
  {
    m_iWorkDatas[i] = Convert.ToInt32(m_strRecvDatas[i]);
    // UIThreadFunction 이벤트 간접적으로 호출
    smartThread1.ReportProgress(m_iWorkDatas.Length);
  }
}

private void smartThread1_UIThreadFunction(SmartX.ThreadArgs args)
{
  // 화면 UI 갱신을 하기 위한 코드
  smartLabel1.Text = args.iProgressPercentage.ToString();
}
```

간접 호출

4) SmartThread 인터페이스 설명

SmartThread Component Interface		
속성		
Priority : System.Threading.ThreadPriority	State : SmartThread.ThreadState	WorkerReportsProgress : bool
메소드		
End() : void	Resume() : void	Start() : void
Suspend() : void	ReportProgress(int iProgress) : void	
이벤트		
OnEnding : SmartThread.EndingHandler	ThreadFunction : SmartThread.ThreadHandler	UIThreadFunction : SmartThread.UIThreadHandler

프로퍼티(속성) State

현재 SmartThread의 동작 상태를 얻습니다.

참고 State 속성은 읽기 전용입니다.

- SmartThread.ThreadState : SmartThread 동작 상태
  - SmartThread.ThreadState.Running : 동작 중

- SmartThread.ThreadState.Stopped : 정지됨
- SmartThread.ThreadState.Suspended : 일시 정지됨
- SmartThread.ThreadState.SuspendRequested : 일시 정지 요청됨

C# 사용법

```
switch (smartThread1.State)
{
  case SmartThread.ThreadState.Running:
    // SmartThread가 동작 중
    break;
  case SmartThread.ThreadState.Stopped:
    // SmartThread가 정지됨
    break;
  case SmartThread.ThreadState.Suspended:
    // SmartThread가 일시 정지됨
    break;
  case SmartThread.ThreadState.SuspendRequested:
    // SmartThread에 일시 정지 요청됨
    break;
}
```

VB 사용법

```
Select Case smartThread1.State
  Case SmartThread.ThreadState.Running
    ' SmartThread가 동작 중
  Case SmartThread.ThreadState.Stopped
    ' SmartThread가 정지됨
  Case SmartThread.ThreadState.Suspended
    ' SmartThread가 일시 정지됨
  Case SmartThread.ThreadState.SuspendRequested
    ' SmartThread에 일시 정지 요청됨
End Select
```

프로퍼티(속성) Priority

SmartThread의 우선 순위를 설정합니다.

주의 Priority 속성 변경 시 주의사항

각각의 스레드는 우선 순위를 가지며, 우선 순위가 낮은 스레드는 우선 순위가 상대적으로 높은 스레드가 멈춤 또는 정지되기 전까지 실행되지 않습니다. 예를 들어 특정 스레드의 우선 순위가 터치 입력 스레드보다 우선 순위가 높다면, 터치 입력이 잘 되지 않는 증상이 발생할 수 있습니다. 이러한 이유로 스레드 우선 순위 변경 시 다양한 테스트가 필요합니다.

- System.Threading.ThreadPriority : SmartThread의 우선 순위
  - System.Threading.ThreadPriority.AboveNormal : Highest와 Normal 사이
  - System.Threading.ThreadPriority.BelowNormal : Normal과 Lowest 사이
  - System.Threading.ThreadPriority.Highest : 최상위
  - System.Threading.ThreadPriority.Lowest : 최하위
  - System.Threading.ThreadPriority.Normal : 중간

C# 사용법

```
// SmartThread의 우선 순위를 Normal로 설정
smartThread1.Priority = System.Threading.ThreadPriority.Normal;
```

VB 사용법

```
smartThread1.Priority = System.Threading.ThreadPriority.Normal
```

프로퍼티(속성) WorkerReportsProgress

Work Thread에서 사용자 인터페이스의 갱신이 필요할 경우 UI Thread를 간접 호출(ReportProgress)을 허용할 것인지 설정합니다.

- bool : UI Thread 간접 호출 활성화 여부
  - true : ReportProgress 메소드 호출 사용함
  - false : ReportProgress 메소드 호출 사용 안 함

VB 사용법

```
//Background Worker 기능 활성화
smartThread1.WorkerReportsProgress = true;
```

메소드(함수) End, Resume, Start, Suspend

SmartThread의 동작 상태를 컨트롤합니다.

- Start() : SmartThread를 시작합니다.
- End() : SmartThread를 종료합니다.
- Suspend() : SmartThread를 일시 정지합니다.
- Resume() : 일시 정지된 SmartThread를 재개합니다.

주의 Visual Studio에서 디버깅(Debugging) 중 SmartThread 사용 시 주의사항

SmartThread의 Suspend() 메소드는 Visual Studio에서 디버깅을 지원하지 않습니다. 따라서 프로그램 개발 중 Visual Studio의 디버깅 모드에서 중단점(Break Point)을 설정하고 추적(Tracing)할 때 Suspend() 메소드를 호출하게 되면 호출 부분에서 프로그램이 멈추게 됩니다.
※ "디버깅 하지 않고 시작" 모드에서는 정상적으로 동작합니다.

◉ void Start()
◉ void End()
◉ void Suspend()
◉ void Resume()

C# 사용법

```
// Work 또는 UI Thread를 시작
smartThread1.Start();
// Work 또는 UI Thread를 일시 정지
smartThread1.Suspend();
// 일시 정지된 Work 또는 UI Thread를 재개
smartThread1.Resume();
// Work 또는 UI Thread를 종료
smartThread1.End();
```

VB 사용법

```
smartThread1.Start()
smartThread1.Suspend()
smartThread1.Resume()
smartThread1.End()
```

메소드(함수) ReportProgress

Work Thread에서 사용자 인터페이스 갱신을 하기 위해 UI Thread를 간접적으로 호출하도록 처리합니다. ReportProgress를 통해 UI Thread를 간접 호출 시 Work Thread와 UI Thread 각각의 장점을 조합하여 사용할 수 있습니다.

주의 ReportPorgress 함수를 사용하기 전 반드시 WorkerReportProgress 속성을 true로 설정하여 Background Worker 기능을 활성화하시기 바랍니다.

◉ `void SmartThread.ReportProgress(int iProgress)`

[인자]

- `iProgress` : UIThreadFunction이벤트 코드의 인자로 전달될 값

C# 사용법

```
//Background Worker 기능 활성화
smartThread1.WorkerReportsProgress = true;

private void smartThread1_ThreadFunction(SmartX.ThreadArgs args)
{
  // 빠른 속도의 연산 처리
  // 중략

  // UIThreadFunction 이벤트 코드를 우회하여 호출
  smartThread1.ReportProgress(30);
}

private void smartThread1_UIThreadFunction(SmartX.ThreadArgs args)
{
    // UI 갱신
    SmartLabel.Text = args.iProgressPercentage.ToString();
}
```

이벤트 OnEnding

SmartThread가 종료될 때 발생하는 이벤트입니다. 즉, End() 메소드가 호출되거나, ThreadFunction 또는 UIThread Function에서 bEnd 인자를 True로 설정할 경우 발생됩니다.

◉ `OnEnding(ThreadArgs args)`

[인자]

- `bool bEnd` : SmartThread의 종료 여부 설정 (쓰기 전용)
  - `true`: 종료함
  - `false`: 종료하지 않음
- `ulong dwCallCount` : 현재까지 수행된 스레드 횟수 (읽기 전용)

C# 사용법

```
private void smartThread1_OnEnding(ThreadArgs args)
{
  // SmartThread가 종료됨
}
```

VB 사용법

```
Private Sub smartThread1_OnEnding(ByVal args As ThreadArgs) Handles smartThread1.OnEnding
  ' SmartThread가 종료됨
End Sub
```

이벤트 ThreadFunction

이벤트 코드를 Work Thread로 동작시키는 이벤트입니다.

주의 ThreadFunction는 화면 UI 컴포넌트에 접근할 수 없습니다.

참고 "1) Work Thread와 UI Thread의 차이" 내용을 참고하시기 바랍니다.

◉ ThreadFunction(ThreadArgs args)

[인자]

- bool bEnd : SmartThread의 종료 여부 설정 (쓰기 전용)
  - true: 종료함
  - false: 종료하지 않음 (기본값)

참고 bEnd 인자를 True로 설정하면 OnEnding 이벤트가 발생합니다. Thread를 계속해서 동작시키는 경우 이벤트 코드 마지막에서 bEnd 인자를 False로 설정하시기 바랍니다.
"OnEnding 이벤트" 내용을 참고하시기 바랍니다.

- ulong dwCallCount : 스레드가 시작되어 ThreadFunction 이벤트(스레드 코드)가 호출된 횟수 (읽기 전용)

C# 사용법

```
private void smartThread1_ThreadFunction(ThreadArgs args)
{
  // Work Thread로 동작시킬 코드를 작성
  // … 중략
  // Work Thread를 다시 처리할 경우 bEnd 인자를 False로 설정
  args.bEnd = false;
}
```

VB 사용법

```
Private Sub smartThread1_ThreadFunction(ByVal args As ThreadArgs) Handles smartThread1.
ThreadFunction
  ' … 중략
  args.bEnd = False
End Sub
```

이벤트 UIThreadFunction

이벤트 코드를 UI Thread로 동작시키는 이벤트입니다.

참고 "1) Work Thread와 UI Thread의 차이" 내용을 참고하시기 바랍니다.

◉ UIThreadFunction(ThreadArgs args)

[인자]

- bool bEnd : SmartThread의 종료 여부 설정 (쓰기 전용)
  - true: 종료함
  - false: 종료하지 않음 (기본값)

참고 bEnd 인자를 True로 설정하면 OnEnding 이벤트가 발생합니다. Thread를 계속해서 동작시키는 경우 이벤트 코드 마지막에서 bEnd 인자를 False로 설정하시기 바랍니다.
"OnEnding 이벤트" 내용을 참고하시기 바랍니다.

- ulong dwCallCount : 스레드가 시작되어 UIThreadFunction 이벤트(스레드 코드)가 호출된 횟수 (읽기 전용)
- int iProgressPercentage : WorkThread(Thread Function)에서 ReportProgress 메소드 호출 시 전달된 인자값 (읽기 전용)

C# 사용법

```
private void smartThread1_UIThreadFunction(ThreadArgs args)
{
  // UI Thread로 동작시킬 코드를 작성하며, UI Thread를 다시 처리할 경우 bEnd 인자를 False로 설정
  args.bEnd = false;
}
```

VB 사용법

```
Private Sub smartThread1_UIThreadFunction(ByVal args As ThreadArgs) Handles smartThread1.
UIThreadFunction
  args.bEnd = False
End Sub
```

5) SmartThread 예제 사용하기

SmartThread를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartThread"

11. SmartFTP

SmartFTP는 IEC-Series에서 FTP Client의 기능들을 쉽고 편리하게 사용할 수 있도록 구현된 컴포넌트입니다.

- 메소드를 사용해 FTP Client 기능을 간편하게 구현
- FTP Client PassiveMode 지원
- 업/다운로드 진행 상황 확인 기능

참고 SmartFTP 사용 전 참고사항

SmartFTP는 개발 및 테스트 시 ALFTP와 FileZiller를 사용했으며, SmartFTP 적용 시 FTP 서버와 연결에 어려움을 겪는 경우 HNS로 연락주시면 기술 지원해 드리도록 하겠습니다.

※ 자세한 FTP 서버 구축 방법은 사용하는 FTP 서버 프로그램의 제작사에 문의하시기 바랍니다.

참고 원격지의 FTP 서버로부터 업데이트 파일을 다운받아 SmartUpdate와 연동하여 프로그램의 업데이트를 처리할 수 있습니다. 자세한 내용은 "SmartUpdate 컴포넌트"를 참고하시기 바랍니다.

1) 프로그래밍 적용 가이드

STEP-1 서버 정보 설정 및 FTP 서버와 연결하기

FTP 서버에 연결하기 위해 각종 필요한 정보를 설정합니다. ServerFTPAddress 속성으로 FTP 서버 주소를, PortNo 속성으로 통신 포트 번호를, UserID와 Password 속성으로 사용자 계정과 비밀번호를 설정합니다.
통신 정보 설정이 완료되면 Connect() 메소드를 호출하여 FTP 서버와 연결합니다. 또한, Disconnect() 메소드를 호출해 FTP 서버와 연결을 해제할 수 있습니다.

※ 자세한 내용은 "ServerFTPAddress, PortNo, UserID, Password 속성", "Connect(), DisConnect() 메소드"를 참고하시기 바랍니다.

```
smartFTP1.ServerFTPAddress = " 192.168.123.100 ";
smartFTP1.PortNo = 21;
smartFTP1.UserID = " hns ";
smartFTP1.Password = " hns ";
if (smartFTP1.Connect() == true)
{
  // FTP 서버와 연결 성공 시 처리 코드 작성
}
// smartFTP1.DisConnect();  // FTP 서버와 연결 해제
```

STEP-2 경로 설정 및 파일 리스트 가져오기

FTP 서버에 파일을 업/다운로드하기 위한 기본 경로를 설정합니다. SetCurrentDirectory() 메소드를 호출하여 기본 경로를 설정합니다. 경로 설정이 완료되면 GetFileList() 메소드를 호출하여 기본 경로에 위치한 모든 파일과 폴더의 경로를 가져올 수 있습니다.

※ 자세한 내용은 "SetCurrentDirectory(), GetFileList() 메소드"를 참고하시기 바랍니다.

```
// 기본 경로를 설정합니다.
if (smartFTP1.SetCurrentDirectory( " FTP₩₩FTPTEST " ) == true)
{
  // GetFileList 메소드에서 인자로 사용될 변수로 최대 3000개의 문자값을 할당
  StringBuilder strFileList = new StringBuilder(3000);
```

```
  // 현재 경로의 파일 리스트를 얻음
  if (smartFTP1.GetFileList(strFileList) == true)
  {
    string strFileList1 = strFileList.ToString();  // StringBuilder를 문자열로 처리하기 위해 변환
    string[] strItem = strFileList1.Split( ' : ' );  // ' : ' 기준으로 항목을 나누어 문자열 배열로 저장
    // 각각의 파일 이름과 폴더명을 리스트에 출력
    for (int i = 0; i < strItem.Length; i++)
    {
      smartListBox1.AddItem(strItem[i]);
    }
  }
}
```

STEP-3 설정된 경로의 파일과 폴더를 생성 및 삭제하기

STEP-2에서 설정한 경로에 CreateFolder() 메소드를 호출해 폴더를 생성할 수 있으며, DeleteFolder() 메소드를 호출해 특정 폴더를 삭제할 수 있습니다. 또한, DeleteFiles() 메소드를 호출해 특정 파일을 삭제할 수 있습니다.

※ 자세한 내용은 "CreateFolder(), DeleteFolder(), DeleteFiles() 메소드"를 참고하시기 바랍니다.

```
// 설정된 경로에 하위 폴더를 만듦
if (smartFTP1.CreateFolder( " TestFoder " ) == true)
{
  smartFTP1.DeleteFolder( " TestFoder " );  // 설정된 경로의 특정 하위 폴더를 지움
}
// 설정된 경로의 특정 파일을 지움
smartFTP1.DeleteFiles( " Test.dat " );
```

STEP-4 파일 업/다운로드 및 진행 상황 확인하기

FileDownload() 메소드를 호출해 FTP 서버의 파일을 IEC-Series로 다운로드 할 수 있으며, FileUpload() 메소드를 호출해 IEC-Series에 저장된 파일을 FTP 서버로 업로드 할 수 있습니다. 또한, 타이머를 사용하여 주기적으로 Percentage 속성을 확인해 업/다운로드 진행 상황을 확인할 수 있습니다. 업/다운로드를 취소하려면 FileUpNDownloadCancel() 메소드를 호출하면 됩니다.

※ 자세한 내용은 "Percentage 속성", "FileDownload(), FileUpload(), FileUpNDownloadCancel() 메소드"를 참고하시기 바랍니다.

```
private void FileDown()
{
  // 입력된 경로의 파일을 다운로드 시작 후 진행 상태 표시를 위해 타이머를 사용
  if (smartFTP1.FileDownload( " //123.DAT " , " Flash Disk₩₩123.DAT " ) == true)
  {
    smartTimer1.Start();
  }
}
private void FileUp()
{
  // 입력된 경로의 파일의 업로드를 시작 후 진행 상태 표시를 위해 타이머를 사용
  if (smartFTP1.FileUpload( " SD Card₩₩123₩₩ABC.DAT " , " FTP₩₩SMARTFTP₩₩ABC.DAT " ) == true)
  {
    smartTimer1.Start();
  }
}
// 파일의 다운로드 및 업로드 시 진행 상태를 표시하기 위한 타이머 이벤트
```

```
private void smartTimer1_Tick(object sender, EventArgs e)
{
  // 현재 100이면 업/다운로드가 완료된 상태
  if (smartFTP1.Percentage == 100) {  }
  // -1이면 업/다운로드 실패
  else if (smartFTP1.Percentage == -1) {  }
  // 그 외의 값은 업/다운로드 진행 중
  else {  }
}
```

2) SmartFTP 인터페이스 설명

SmartFTP Component Interface		
속성		
PassiveMode : bool	Password : string	Percentage : int
PortNo : int	ServerFTPAddress : string	UserID : string
메소드		
Connect() : bool	CreateFolder(string strFolderName) : bool	DeleteFiles(string strFileName) : bool
DeleteFolder(string strFolderName) : bool	DisConnect() : bool	FileDownload(string strFTPFilePathName, string strLocalFilePathName) : bool
FileUpload(string strLocalFilePathName, string strFTPFilePathName) : bool	FileUpNDownloadCancel() : void	GetCurrentDirectory(ref string strPath) : bool
GetFileList(StringBuilder strRetList) : bool	ReName(string strOldName, string strNewName) : bool	SetCurrentDirectory(string strPath) : bool
SetRootDirectory() : bool		

프로퍼티(속성) ServerFTPAddress, PortNo, UserID, Password

FTP 서버에 연결하기 위하여 필요한 정보를 설정합니다.

- ServerFTPAddress : FTP 서버 주소를 설정합니다.(URL or IP)
- PortNo : 포트 번호를 설정합니다.
- UserID : 사용자 접속 계정을 설정합니다.
- Password : 사용자 접속 비밀번호를 설정합니다.

C# 사용법

```
// FTP 서버의 연결 설정 정보를 설정
smartFTP1.ServerFTPAddress = " 192.168.123.100 ";
smartFTP1.PortNo = 21;      smartFTP1.PassiveMode = true;
smartFTP1.UserID = " hns ";  smartFTP1.Password = " hns ";
```

VB 사용법

```
smartFTP1.ServerFTPAddress = " 192.168.123.100 "
smartFTP1.PortNo = 21   :   smartFTP1.PassiveMode = true
smartFTP1.UserID = " hns "  :   smartFTP1.Password = " hns "
```

프로퍼티(속성) PassiveMode

FTP 서버 접속 시 PassiveMode로 접속할지를 설정합니다.

- bool : PassiveMode 사용 여부

- true : 사용
- false : 미사용

C# 사용법

```
// PassiveMode 사용
smartFTP1.PassiveMode = true;
```

VB 사용법

```
smartFTP1.PassiveMode = true
```

프로퍼티(속성) Percentage

파일의 업/다운로드 시 진행 상태를 백분율 값으로 얻습니다. 또는 실패 유무를 확인합니다.

참고 FileDownload() 또는 FileUpload() 메소드 호출 후 타이머를 사용해 진행 상황을 확인할 수 있습니다. 자세한 내용은 "FileDownload(), FileUpload() 메소드"를 참고하시기 바랍니다.

• int : 업/다운로드 진행 상태
※ 0~100 : 진행 중, -1 : 업/다운로드 실패

C# 사용법

```
// 파일의 다운로드 및 업로드 시 진행 상태를 표시하기 위한 타이머 이벤트
private void smartTimer1_Tick(object sender, EventArgs e)
{
  if (smartFTP1.Percentage == 100)
  {
    // 업/다운로드가 완료된 상태일 때 처리 코드를 작성
  }
  else if (smartFTP1.Percentage == -1)
  {
    // 업/다운로드 실패 시 처리 코드를 작성
  }
  else
  {
    // 업/다운로드가 진행 중일 때 처리 코드를 작성
  }
}
```

VB 사용법

```
Private Sub smartTimer1_Tick(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
smartTimer1.Tick
  If smartFTP1.Percentage = 100 Then
  Else If smartFTP1.Percentage = -1 Then
  Else
  End If
End Sub
```

메소드(함수) Connect, DisConnect

설정된 FTP 서버로 연결합니다.
• Connect() : FTP 서버에 연결합니다.
• DisConnect() : FTP 서버와 연결을 해제합니다.

◉ bool Connect()
◉ bool DisConnect()

[리턴값]
• bool : 연결 및 연결 해제 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// FTP 서버로 연결 요청 성공
if (smartFTP1.Connect() == true) {  }
// FTP 서버와 연결 해제
smartFTP1.DisConnect();
```

VB 사용법

```
If smartFTP1.Connect() = True Then
End If
smartFTP1.DisConnect()
```

메소드(함수) SetCurrentDirectory, GetCurrentDirectory

업/다운로드 경로를 설정하거나, 설정된 경로를 얻습니다.
◉ bool SetCurrentDirectory(string strPath)
◉ bool GetCurrentDirectory(ref string strPath)

[인자]
• string strPath : 설정할 경로
• ref string strPath : 설정된 경로

[리턴값]
• bool : 경로 설정 및 얻기 성공 여부
  – true : 성공
  – false : 실패

C# 사용법

```
// 경로 설정 성공
if (smartFTP1.SetCurrentDirectory( " FTP₩₩FTPTEST " ) == true) {  }
string strCurrentPath; // 설정된 경로를 얻기 위한 변수
// 현재 설정된 경로를 변수에 저장 및 성공 여부 확인
if (smartFTP1.GetCurrentDirectory(ref strCurrentPath) == true) {  }
```

VB 사용법

```
If smartFTP1.SetCurrentDirectory( " FTP₩₩FTPTEST " ) = True Then
End If
Dim strCurrentPath As String
If smartFTP1.GetCurrentDirectory(strCurrentPath) = True Then
End If
```

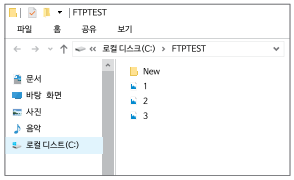
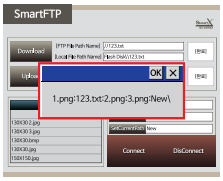
메소드(함수) GetFileList

SetCurrentDirectory 속성에서 설정된 경로의 모든 파일과 폴더 리스트를 얻습니다.
◉ bool GetFileList(StringBuilder strRetList)

중요 StringBuilder 리턴 형식

GetFileList() 메소드는 인자값으로 설정한 StringBuilder 변수에 다음과 같은 형식의 문자열을 리턴합니다.

[저장 형식]
파일이름.확장자: … 중략 … 폴더 이름₩:파일 이름.확장자: …

FTP 서버의 경로	GetFileList() 리턴값
	

[인자]

• StringBuilder strRetList : 현재 설정된 경로의 모든 파일과 폴더명

[리턴값]

• bool : 경로 설정 및 얻기 성공 여부
  – true : 성공 / – false : 실패

C# 사용법

```
// GetFileList 메소드에서 인자로 사용될 변수로 최대 3000개의 문자값을 할당
StringBuilder strFileList = new StringBuilder(3000);
// 현재 경로의 파일 리스트를 얻음
if (smartFTP1.GetFileList(strFileList) == true)
{
  string strFileList1 = strFileList.ToString();  // StringBuilder를 문자열로 처리하기 위해 변환함
  string[] strItem = strFileList1.Split(':');  // ':' 기준으로 항목을 나누어 문자열 배열로 저장
  // 각각의 파일 이름과 폴더명을 리스트에 출력
  for (int i = 0; i < strItem.Length; i++)
  {
    smartListBox1.AddItem(strItem[i]);
  }
}
```

VB 사용법

```
Dim strFileList As New StringBuilder(3000)
If smartFTP1.GetFileList(strFileList) = True Then
  Dim strFileList1 As String = strFileList.ToString()
  Dim strItem() As String = strFileList1.Split(':')
  For I As Integer = 0 To strItem.Length -1
    smartListBox1.AddItem(strItem(i))
  Next
End If
```

메소드(함수) SetRootDirectory

Root Directory로 이동합니다.

◉ bool SetRootDirectory()

[리턴값]

• bool : 설정 성공 여부
  – true : 성공 / – false : 실패

C# 사용법

```
smartFTP1.SetRootDirectory();
```

VB 사용법

```
smartFTP1.SetRootDirectory()
```

메소드(함수) FileDownload, FileUpload

- FileDownload() : FTP 서버로부터 지정된 경로의 파일을 다운로드합니다.
- FileUpload() : IEC-Series의 파일을 FTP 서버의 지정된 경로에 업로드합니다.

주의 FileDownload(), FileUpload() 메소드 사용 시 주의사항

1. strFTPFilePathName 인자는 "//"로 경로를 구분합니다.
2. strLocalFilePathName 인자는 "₩₩"로 경로를 구분합니다.
3. strFTPFilePathName 인자의 경로명 지정 시 처음부터 파일 명으로 시작되는 경우 가장 앞에 "//"를 추가하고, 폴더 명으로 시작되는 경우 가장 앞에 "//"를 추가하지 않습니다.

```
iFTPFilePathName = " //TestText.txt " ;  // 파일명으로 시작되는 경우
iFTPFilePathName = " SubFolder1//SubFolder2//TestText.txt " ;  // 폴더명으로 시작되는 경우
```

4. 대소문자 및 서버 측의 방화벽이 체크되었는지 확인합니다.
5. 업로드 진행 중 다운로드 실행 시 진행 중이던 업로드 작업이 취소됩니다. (반대의 경우도 동일)

중요 FileDownload와 FileUpload는 NoneBlocking 메소드입니다. 함수가 호출 즉시 리턴하는 구조로 되어 있습니다. 따라서 업/다운로드 진행 상태를 확인하는 방법이 필요하게 됩니다.
타이머에서 주기적으로 Percentage 속성값을 확인하면 현재 업/다운로드 상태를 확인할 수 있습니다.

참고 "Percentage 속성" 내용을 참고하시기 바랍니다.

◉ bool FileDownload(string strFTPFilePathName, string strLocalFilePathName)
◉ bool FileUpload(string strLocalFilePathName, string strFTPFilePathName)

[인자]

- string strFTPFilePathName : FTP 서버의 경로 및 파일 이름 (경로 구분자 : //)
- string strLocalFilePathName : IEC-Series의 Local 경로 및 파일 이름 (경로 구분자 : ₩₩)

[리턴값]

- bool : 업/다운로드 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// 입력된 경로의 파일을 다운로드 시작 후 다운로드 진행 상태를 타이머로 표시
if (smartFTP1.FileDownload( " //123.DAT " , " Flash Disk₩₩123.DAT " ) == true)
{
  smartTimer1.Start();
}
else
{
  // 다운로드 실패
}
// 입력된 경로의 파일을 업로드 시작 후 다운로드 진행 상태를 타이머로 표시
if (smartFTP1.FileUpload( " SD Card₩₩123₩₩ABC.DAT " , " FTP//SMARTFTP//ABC.DAT " ) == true)
{
  smartTimer1.Start();
}
else
{
  // 업로드 실패
}
```

VB 사용법

```
If smartFTP1.FileDownload( " //123.DAT " , " Flash Disk₩₩123.DAT " ) = True Then
```

```
  smartTimer1.Start()
Else
  ' 다운로드 실패
End If
If smartFTP1.FileUpload( " SD Card₩₩123₩₩ABC.DAT " , " FTP//SMARTFTP//ABC.DAT " ) = True Then
  smartTimer1.Start()
Else
  ' 업로드 실패
End If
```

메소드(함수) FileUpNDownloadCancel

파일의 업/다운로드 작업을 취소합니다.

◉ void FileUpNDownloadCancel()

C# 사용법

```
// 업/다운로드 작업을 취소합니다.
smartFTP1.FileUpNDownloadCancel();
```

VB 사용법

```
smartFTP1.FileUpNDownloadCancel()
```

메소드(함수) CreateFolder, DeleteFolder, DeleteFiles

- CreateFolder() : SetCurrentDirectory 속성에서 설정된 경로에 폴더를 생성합니다.
- DeleteFolder() : SetCurrentDirectory 속성에서 설정된 경로의 폴더를 삭제합니다.
- DeleteFiles() : SetCurrentDirectory 속성에서 설정된 경로의 파일을 삭제합니다.

참고 "SetCurrentDirectory 속성" 내용을 참고하시기 바랍니다.

중요 DeleteFolder() 메소드 호출 시 해당 폴더 안에 폴더나 파일이 존재하면 폴더가 삭제되지 않습니다.

◉ bool CreateFolder(string strFolderName)
◉ bool AddImageList(string strFolderName)
◉ bool AddImageList(string strFileName)

[인자]

- string strFolderName : 생성 및 삭제할 폴더 이름
- string strFileName : 삭제할 파일 이름

[리턴값]

- bool : 생성 및 삭제 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// 설정된 경로에 하위 폴더를 만듬
if (smartFTP1.CreateFolder( " TestFolder " ) == true)
{
  // 설정된 경로의 특정 하위 폴더를 지움
  smartFTP1.DeleteFolder( " TestFolder " );
}

// 설정된 경로의 특정 파일을 지움
smartFTP1.DeleteFiles( " Test.dat " );
```

VB 사용법

```
If smartFTP1.CreateFolder( " TestFolder " ) = True Then
  smartFTP1.DeleteFolder( " TestFolder " )
End If
smartFTP1.DeleteFiles( " Test.dat " )
```

메소드(함수) **ReName**

특정 경로의 파일 및 폴더의 이름을 변경합니다.

◉ bool ReName(string strOldName, string strNewName)

[인자]

- string strOldName : 현재 파일/폴더 이름 변경하는 대상 이름
- string strNewName : 새로 변경 될 파일/폴더 이름

[리턴값]

- bool : 이름 변경 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
// 폴더 이름 변경
smartFTP1.ReName( " FTP//FTPTEST " , " FTP//SMARTFTP " );
// 파일 이름 변경
smartFTP1.ReName( " FTP//SMARTFTP//123.DAT " , " FTP//SMARTFTP//ABC.DAT " );
```

VB 사용법

```
smartFTP1.ReName( " FTP//FTPTEST " , " FTP//SMARTFTP " )
smartFTP1.ReName( " FTP//SMARTFTP//123.DAT " , " FTP//SMARTFTP//ABC.DAT " )
```

3) SmartFTP 예제 사용하기

SmartFTP를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartFTP"

12. SmartScreenSaver

IEC-Series를 고정된 화면으로 장시간 켜 놓는 경우 LCD 패널의 개별 화소의 수명에 따라 핫 픽셀, 데드 픽셀, 스턱 픽셀이 발생할 수 있으며, Backlight-LED 수명이 줄어들어 화면이 어두워지거나 켜지지 않을 수 있습니다. HNS에서 사용되는 LCD 패널의 수명은 개별 화소 수명보다 Backlight-LED의 수명이 가장 큰 영향을 줍니다. 따라서 LCD의 수명을 높이는 가장 좋은 방법은 Backlight-LED를 오랜 시간 사용하지 않는 경우 OFF 하는 것이 가장 좋습니다.

SmartScreenSaver는 이런 증상을 방지하기 위해 지정된 시간 동안 화면 터치 입력이 없는 경우 화면 보호 기능을 최소의 코딩으로 편리하게 구현할 수 있도록 만들어진 컴포넌트입니다. SmartScreenSaver를 적용하시어 개별 화소 수명과 Backlight-LED 수명의 각 요소를 사용 환경에 맞게 적용하시기 바랍니다.

■ 다양한 Screen Saver 모드 지원
  └ 이벤트 컨트롤, Backlight 컨트롤, Text 출력 모드, 외부 프로그램 실행
■ 텍스트 출력 모드인 경우 다양한 속성 지원
  └ 출력 방식 선택 가능(대각선 이동, 랜덤 출력)
  └ 출력 빈도 및 속도 설정 기능
  └ Font, 크기, 색상 설정 기능
■ Screen Saver 동작 시간 설정 기능
■ 터치 입력 기준으로 Screen Saver 기능 동작
  └ Screen Saver 모드에서 터치 입력 없이 Saver 모드 강제 중지 기능 지원

주의 SmartScreenSaver 사용 시 주의사항

1. 본 기능은 IEC667/1000-Series에서만 지원됩니다.
2. 프로그램 종료 시 반드시 Stop() 메소드를 호출하시기 바랍니다.
3. MoveSpeedMin 값은 반드시 MoveSpeedMax 값보다 작아야 하며, 두 값은 모두 0보다 커야 합니다.

[표] LCD 패널의 수명 요소에 따른 저감 방법

수명 요소	개별 화소	Backlight-LED
불량 원인	장시간 동일 화면이 출력될 경우	제품을 장시간 켜 놓을 경우
저감 방법 (장시간 터치 입력이 없는 경우)	검정화면으로 출력된 상태에서 동적인 화면 출력	Backlight-LED OFF

참고 Screen Saver 동작에 대한 참고사항

1. Start() 메소드를 호출 후 ScreenSaverTimeOutSec 속성에서 설정된 시간동안 터치 입력이 없을 경우 Screen Saver가 동작합니다.
2. ScreenSaver는 일반적으로 사용자가 직접 화면을 터치하여 중지할 수 있으며, 또한 특정 상황(GPIO, ADC, 통신 관련 입력 발생)에서 IsScreensaverState 속성값을 확인하여 프로그램상에서 임의로 중지할 수 있습니다.
3. Stop() 메소드를 호출하여 Screen Saver를 완전히 종료합니다.

1) 프로그래밍 적용 가이드

Screen Saver의 동작 모드는 크게 외부 프로그램 사용과 프로그램 내 Screen Saver 사용으로 분류할 수 있습니다.
외부 프로그램 사용 모드는 RUN_PROGRAM_BACKGROUND, RUN_PROGRAM_WAITFOREXIT가 있으며, 사용 시 반드시 외부 프로그램이 필요합니다. 프로그램 내 Screen Saver 사용 모드는 BACKLIGHT_CONTROL, Draw_TEXT, EVENT_ONLY가 있습니다. 아래 STEP은 Draw_TEXT 모드로 동작하는 경우를 설명합니다.

STEP-1 Screen Saver 속성 설정하기

ScreensaverActionMode와 ScreenSaverTimeoutSec 속성으로 Screen Saver의 동작 모드와 마지막 터치 입력 후 Screen Saver가 동작할 시간을 설정합니다. Draw_TEXT 모드로 할 경우 Screen Saver에 보여질 메시지 설정과 폰트의 디자인 요소를 설정 후 메시지 표현 방식을 선택합니다. 2개의 표현 방식이 있으며, TextDraw_RefreshInterval 속성으로 메시지 갱신 간격을 설정합니다. 메시지 표현 방식은 DIAGONAL_MOVE(대각선 이동), FLICKER_RANDOM(랜덤 이동)이 있으며, DIAGONAL_MOVE인 경우 Text Draw_MoveSpeedMax와 TextDraw_MoveSpeedMin 속성으로 메시지가 움직일 간격을 설정해야 합니다.

※ 자세한 내용은 "ScreensaverActionMode, ScreenSaverTimeoutSec, Draw_TEXT관련 속성" 내용을 참고하시기 바랍니다.

```
// Screen Saver 모드를 DRAW_TEXT로 설정
smartScreenSaver1.ScreensaverActionMode = SmartX.SmartScreenSaver.SCREENSAVERACTIONCODE.DRAW_TEXT;
// 마지막 터치 입력 후 Screen Saver 동작 시간 설정
smartScreenSaver1.ScreenSaverTimeoutSec = 60;
// Screen Saver에 보여질 메세지 설정
smartScreenSaver1.TextDraw_Message = " SmartScreenSaver Test ";
// 폰트 설정
smartScreenSaver1.TextDraw_FontName = " Gulim ";
// 폰트 사이즈 설정
smartScreenSaver1.TextDraw_FontSize = 20;
// 폰트 굵기 설정 (false 시 기본)
smartScreenSaver1.TextDraw_FontBold = true;
// 폰트 색상을 무지개색으로 설정
List<Color> listColor = new List<Color>();
listColor.Add(Color.Red);
listColor.Add(Color.Orange);
listColor.Add(Color.Yellow);
listColor.Add(Color.Green);
listColor.Add(Color.Blue);
listColor.Add(Color.DarkBlue);
listColor.Add(Color.Purple);
smartScreenSaver1.TextDraw_FontColors = listColor;
// 메세지 갱신 간격 설정
smartScreenSaver1.TextDraw_RefreshInterval = 10;
// 메세지 표현 방식을 DIAGONAL_MOVE로 설정
smartScreenSaver1.TextDraw_MoveType = SmartX.SmartScreenSaver.TEXTDRAWMOVETYPECODE.DIAGONAL_MOVE;
// 메시지가 움직일 간격을 설정(Min~Max 값 사이에서 랜덤으로 변경됨)
smartScreenSaver1.TextDraw_MoveSpeedMax = 50;  smartScreenSaver1.TextDraw_MoveSpeedMin = 10;
```

STEP-2 Screen Saver 시작/종료하기

Screen Saver는 Start() 메소드를 호출하여 화면 터치 후 ScreenSaverTimeoutSec 속성에서 설정한 시간이 지나면 Screen Saver를 시작할 수 있으며, Stop() 메소드를 호출하여 Screen Saver를 완전히 종료할 수 있습니다.

※ 자세한 내용은 " Start(), Stop() 메소드" 내용을 참고하시기 바랍니다.

```
smartScreenSaver1.Start();  // 스크린 세이버 시작
smartScreenSaver1.Stop();   // 스크린 세이버 종료
```

참고 프로그램 상에서 임의로 Screen Saver를 강제로 종료하는 방법

Screen Saver 모드로 진입하여 Screen Saver가 동작 중인 상태에서 특정 상황(외부 입력 상태의 변화 – 통신, GPIO, ADC 등)에 따라 Screen Saver 모드를 강제 종료할 수 있습니다.

아래 예제 코드는 SmartGPIO에서 Port-A의 Pin0에 입력되는 신호의 변화(LevelChange : LOW → HIGH 또는 HIGH → LOW)가 감지된 경우 IsScreensaverState 속성으로 동작 여부를 확인 후 Smart Configs의 MouseClick() 메소드를 사용하여 Screen Saver를 강제로 중지하는 예시입니다.

※ 자세한 내용은 "IsScreensaverState 속성", "SmartGPIO, SmartConfigs"를 참고하시기 바랍니다.

[C#] 예시 코드

```
private void smartGPIO1_EvtPortADatasChange(object sender, EventArgs e)
{
  int iPortDatas  // 현재 Port 상태를 저장할 변수
  SmartX.SmartGPIO.PORTPIN iBit;  // 신호 변화가 감지된 Pin 정보를 저장할 변수
  // Event 방식일 경우 인자값을 이용하여 Port-A의 상태를 변수에 저장
  iPortDatas = ((SmartX.PORTDataEvtArgs)e).iPortDatas;
  // Port-A의 Pin1에 입력되는 신호 변화(Low → High, High → Low)를 감지하는 경우
  iBit = smartGPIO1.PortDetection(SmartX.SmartGPIO.PORTID.PORTA,
  SmartX.SmartGPIO.TRIGGERMODE.LevelChange, iPortDatas, SmartX.SmartGPIO.PORTPIN.PIN1);
  // 신호 변화가 감지된 Pin의 정보(iBit)가 Pin1일 경우
  if (iBit == SmartX.SmartGPIO.PORTPIN.PIN1)
  {
    // Screen Saver가 동작 중인 경우 true
    if (smartScreenSaver1.IsScreensaverState == true)
    {
      // smartConfigs를 사용하여 현재 Screen Saver가 동작 중이라면
      // 임의로 특정 위치를 클릭하여 Screen Saver 중지
      smartConfigs1.Display.MouseClick(0, 0);
    }
  }
}
```

2) SmartScreenSaver 인터페이스 설명

SmartScreenSaver Component Interface		
속성		
IsScreensaverState : bool	RunProgramPathName : string	ScreensaverActionMode : SmartScreen Saver.SCREENSAVERACTIONCODE
ScreenSaverTimeoutSec : int	TextDraw_FontBold : bool	TextDraw_FontColors : List<Color>
TextDraw_FontName : string	TextDraw_FontSize : int	TextDraw_Message : string
TextDraw_MoveSpeedMax : int	TextDraw_MoveSpeedMin : int	TextDraw_MoveType : SmartScreen Saver.TEXTDRAWMOVETYPECODE
TextDraw_RefreshInterval : int		
메소드		
Start() : void	Stop() : void	
이벤트		
ONScreenSaverTimeOut : void		

프로퍼티(속성) IsScreensaverState

Screen Saver 동작 여부를 확인합니다. 사용자가 직접 화면을 터치하여 Screen Saver를 중지하는 것이 아닌 프로그램 상에서 임의로 Screen Saver를 중지하는데 유용한 속성입니다.

[ScreensaverActionMode에 따른 리턴값]

- EVENT_ONLY : Screen Saver 동작 시 true, 동작 안하는 경우 false 리턴
- BACKLIGHT_CONTROL : Screen Saver 동작 여부와 관계없이 false를 리턴
- DRAW_TEXT : Screen Saver 동작 시 true, 동작 안하는 경우 false 리턴
- RUN_PROGRAM_BACKGROUND : Screen Saver 동작 시 true, 동작 안하는 경우 false 리턴
- RUN_PROGRAM_WAITFOREXIT : Screen Saver 동작 시 true, 동작 안하는 경우 false 리턴

C# 사용법

```
// ScreenSaver가 동작 중인 경우 특정 좌표를 임의로 클릭하게 하여 ScreenSaver를 중지
if (smartScreenSaver1.IsScreensaverState == true)
{
  smartConfigs1.Display.MouseClick(0, 0);
}
// ScreensaverActionMode가 BACKLIGHT_CONTROL인 경우 IsScreensaverState는 항상 false를 리턴하므로
// BackLight 상태를 확인하여 현재 Off인 경우 BackLight를 On하도록 함
else if (smartConfigs1.Display.BacklightOnOffStatus == CDisplay.OnOff.OFF)
{
  smartConfigs1.Display.BacklightControl(CDisplay.OnOff.ON);
}
```

VB 사용법

```
If smartScreenSaver1.IsScreensaverState = True Then
  smartConfigs1.Display.MouseClick(0, 0)
Else If smartConfigs1.Display.BacklightOnOffStatus = CDisplay.OnOff.OFF Then
  smartConfigs1.Display.BacklightControl(CDisplay.OnOff.ON)
End If
```

프로퍼티(속성) RunProgramPathName

ScreensaverActionMode 속성이 RUN_PROGRAM_BACKGROUND 또는 RUN_PROGRAM_WAITFOREXIT일 경우에 실행할 외부 프로그램 경로를 설정합니다.

- string : ScreenSaver 모드에서 실행될 외부 프로그램 경로

C# 사용법

```
// 외부 프로그램 경로 설정
smartScreenSaver1.RunProgramPathName = " SD Card₩₩extProgram₩₩DrawTextMoving.exe ";
```

VB 사용법

```
smartScreenSaver1.RunProgramPathName = " SD Card₩₩extProgram₩₩DrawTextMoving.exe "
```

프로퍼티(속성) ScreensaverActionMode

Screen Saver 동작 모드를 설정합니다.

- SmartScreenSaver.SCREENSAVERACTIONCODE.BACKLIGHT_CONTROL : Backlight-LED를 OFF
- SmartScreenSaver.SCREENSAVERACTIONCODE.DRAW_TEXT : TextDraw_MoveType 속성에서 설정한 방식으로 Text를 출력 (대각선 이동, 랜덤 출력)
- SmartScreenSaver.SCREENSAVERACTIONCODE.EVENT_ONLY : OnScreenSaverTimeOut 이벤트가 발생하여 사용자가 원하는 동작을 실행

• SmartScreenSaver.SCREENSAVERACTIONCODE.RUN_PROGRAM_BACKGROUND : 지정한 외부 프로그램을 호출, Blocking 방식으로 원 프로그램은 백그라운드에서 동작

• SmartScreenSaver.SCREENSAVERACTIONCODE.RUN_PROGRAM_WAITFOREXIT : 지정한 외부 프로그램을 호출, None-Blocking 방식으로 호출한 프로그램이 종료될 때까지 원 프로그램의 UI 갱신 대기

주의 RUN_PROGRAM_BACKGROUND, RUN_PROGRAM_WAITFOREXIT 동작 모드 사용 시 반드시 외부 프로그램이 필요합니다.

C# 사용법

```
// Screen Saver 동작 모드를 텍스트 출력 방식으로 설정
smartScreenSaver1.ScreensaverActionMode = SmartScreenSaver.SCREENSAVERACTIONCODE.DRAW_TEXT;
```

VB 사용법

```
smartScreenSaver1.ScreensaverActionMode = SmartScreenSaver.SCREENSAVERACTIONCODE.DRAW_TEXT
```

프로퍼티(속성) ScreenSaverTimeoutSec

화면 터치 후 Screen Saver가 동작할 때까지의 시간을 설정합니다.

• int : 화면 터치 후 Screen Saver 동작 시간 (최소값 : 60, 단위 : 초)

C# 사용법

```
// Screen Saver 동작 시간 설정
smartScreenSaver1.ScreenSaverTimeoutSec = 60;
```

VB 사용법

```
smartScreenSaver1.ScreenSaverTimeoutSec = 60
```

프로퍼티(속성) TextDraw_FontBold

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Font를 굵게 설정합니다.

• bool : Font 굵기 상태
  - true : 굵게
  - false : 보통

C# 사용법

```
// 보여지는 Font를 굵게 설정
smartScreenSaver1.TextDraw_FontBold = true;
```

VB 사용법

```
smartScreenSaver1.TextDraw_FontBold = True
```

프로퍼티(속성) TextDraw_FontColors

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Text 색상을 리스트로 설정합니다. 색상 설정을 하지 않는 경우 기본으로 지정된 색상으로 변경되어 보여집니다.

• List<Color> : Text 색상 리스트

C# 사용법

```
// ScreenSaver의 FontColor를 무지개색으로 설정
List<Color> listColor = new List<Color>();
listColor.Add(Color.Red);
// …중략…
listColor.Add(Color.Purple);
```

```
smartScreenSaver1.TextDraw_FontColors = listColor;
```

VB 사용법

```
Dim listColor As List(Of Color) = New List(Of Color)
listColor.Add(Color.Red)
' …중략…
listColor.Add(Color.Purple)
smartScreenSaver1.TextDraw_FontColors = listColor
```

프로퍼티(속성) TextDraw_FontName

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Font를 설정합니다.

- `string` : Font 이름

참조 IEC-Series에 탑재된 Font가 아닌 다른 Font를 사용하고 싶은 경우에는 "홈페이지(www.hnsts.co.kr) → 자료실 → Tech Note → 20. IEC-Series에서 다양한 폰트(Fonts 폴더, 트루타입)사용 방법 안내"를 참조하시기 바랍니다.

C# 사용법

```
// 보여지는 Font를 굴림으로 설정
smartScreenSaver1.TextDraw_FontName = "Gulim";
```

VB 사용법

```
smartScreenSaver1.TextDraw_FontName = "Gulim"
```

프로퍼티(속성) TextDraw_FontSize

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Font 크기를 설정합니다.

- `int` : Font 크기

C# 사용법

```
// 보여지는 Font 크기를 20으로 설정
smartScreenSaver1.TextDraw_FontSize = 20;
```

VB 사용법

```
smartScreenSaver1.TextDraw_FontSize = 20
```

프로퍼티(속성) TextDraw_Message

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Text를 설정합니다.

- `string` : Text 문자열

C# 사용법

```
// 보여지는 Text를 설정
smartScreenSaver1.TextDraw_Message = "SmartScreenSaver 테스트";
```

VB 사용법

```
smartScreenSaver1.TextDraw_Message = "SmartScreenSaver 테스트"
```

프로퍼티(속성) TextDraw_MoveSpeedMax

ScreensaverActionMode 속성이 DRAW_TEXT 이고 TextDraw_MoveType 속성이 DIAGONAL_MOVE일 때,

Screen Saver에 보여지는 Text가 움직이는 최대 속도를 지정합니다.

• int : Text가 움직이는 최대 속도

주의 반드시 0 이상의 값이여야 하며, TextDraw_MoveSpeedMin 속성값보다 크게 설정하시기 바랍니다.

C# 사용법

```
smartScreenSaver1.TextDraw_MoveSpeedMax = 50;
```

VB 사용법

```
smartScreenSaver1.TextDraw_MoveSpeedMax = 50
```

프로퍼티(속성) TextDraw_MoveSpeedMin

ScreensaverActionMode 속성이 DRAW_TEXT 이고 TextDraw_MoveType 속성이 DIAGONAL_MOVE일 때, Screen Saver에 보여지는 Text가 움직이는 최소 속도를 지정합니다.

• int : Text가 움직이는 최소 속도

주의 반드시 0 이상의 값이여야 하며, TextDraw_MoveSpeedMax 속성값보다 작게 설정하시기 바랍니다.

C# 사용법

```
smartScreenSaver1.TextDraw_MoveSpeedMin = 10;
```

VB 사용법

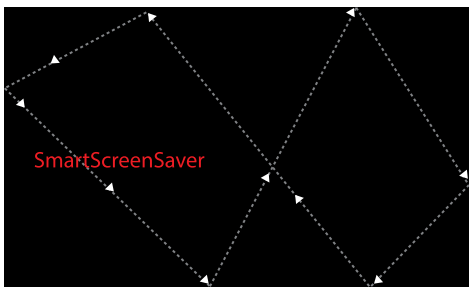
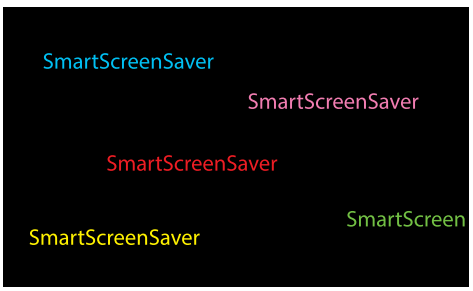
```
smartScreenSaver1.TextDraw_MoveSpeedMin = 10
```

프로퍼티(속성) TextDraw_MoveType

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Text의 표현 방식을 설정합니다.

• SmartScreenSaver.TEXTDRAWMOVETYPECODE.DIAGONAL_MOVE : Text가 대각선으로 이동 (Text가 테두리에 부딪히면 튕겨져서 다시 대각선으로 이동 반복)

• SmartScreenSaver.TEXTDRAWMOVETYPECODE.FLICKER_RANDOM : Text가 랜덤으로 나타남

DIAGONAL_MOVE	FLICKER_RANDOM
	

C# 사용법

```
smartScreenSaver1.TextDraw_MoveType = SmartScreenSaver.TEXTDRAWMOVETYPECODE.DIAGONAL_MOVE;
```

VB 사용법

```
smartScreenSaver1.TextDraw_MoveType = SmartScreenSaver.TEXTDRAWMOVETYPECODE.DIAGONAL_MOVE
```

프로퍼티(속성) TextDraw_RefreshInterval

ScreensaverActionMode 속성이 DRAW_TEXT일 때, Screen Saver에 보여지는 Text가 갱신되는 속도를 설정합니다.

• int : Text가 갱신되는 속도 (최소값 : 1, 단위 : ms)

[표] TextDraw_RefreshInterval값에 따른 TextDraw_MoveType 속성별 출력 결과

TextDraw_RefreshInterval	TextDraw_MoveType	
	DIAGONAL_MOVE	FLICKER_RANDOM
낮음	빠르게 움직임	화면 출력 빈도가 높아짐
높음	부드럽게 움직임	화면 출력 빈도가 낮아짐

C# 사용법

```
// Text 갱신 속도를 설정
smartScreenSaver1.TextDraw_RefreshInterval = 10;
```

VB 사용법

```
smartScreenSaver1.TextDraw_RefreshInterval = 10
```

메소드(함수) Start

Screen Saver를 시작합니다.

◉ void Start()

C# 사용법

```
// Screen Saver 시작
smartScreenSaver1.Start();
```

VB 사용법

```
smartScreenSaver1.Start()
```

메소드(함수) Stop

Screen Saver를 종료합니다.

◉ void Stop()

주의 프로그램 종료 시 반드시 Stop() 메소드를 호출하시기 바랍니다.

C# 사용법

```
// Screen Saver 종료
smartScreenSaver1.Stop();
```

VB 사용법

```
smartScreenSaver1.Stop()
```

이벤트 ONScreenSaverTimeOut

ScreenSaverTimeoutSec 속성에서 지정한 시간이 될 때까지 터치 입력이 없으면 발생하는 이벤트입니다.

C# 사용법

```
private void smartScreenSaver1_ONScreenSaverTimeOut(object sender, EventArgs e)
{
  // ScreensaverActionMode 속성에 따른 처리
  if (smartScreenSaver1.ScreensaverActionMode == SmartScreenSaver.SCREENSAVERACTIONCODE.EVENT_
                                                 ONLY)
  {
    SmartMessageBox.Show(" 오랫동안 터치 입력이 되지 않았습니다!! " + "  " + smartScreenSaver1.
                         IsScreensaverState.ToString());
  }
}
```

VB 사용법

```
Private Sub smartScreenSaver1_ONScreenSaverTimeOut(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles smartScreenSaver1.ONScreenSaverTimeOut
  If smartScreenSaver1.ScreensaverActionMode =
                    SmartScreenSaver.SCREENSAVERACTIONCODE.EVENT_ONLY Then
    SmartMessageBox.Show( " 오랫동안 터치 입력이 되지 않았습니다!! " )
  End If
End Sub
```

3) SmartScreenSaver 예제 사용하기

SmartScreenSaver를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartScreenSaver"

13. SmartPlayer

SmartPlayer는 IEC-Series에서 동영상 재생을 위하여 지원되는 컴포넌트로 동영상 관련 프로그램 개발을 쉽게 처리할 수 있는 기능들을 제공합니다.

- IEC667-Series 이상의 제품군 사용 권장
- O/S Pro 버전에서만 지원
- 동영상 코덱 AVI(MPEG-4 V2, MPEG-4 V3, wmv1, wmv2), WMV(wmv1, wmv2) 지원

주의 IEC266-Series에서 SmartPlayer 사용 시 주의사항

IEC266-Series인 경우 CPU 성능상의 원인으로 SmartPlayer 사용을 권장하지 않으며, C++(MFC)에서만 지원됩니다. (C#, VB.NET 지원 안 함)

중요 IEC-Series에서 지원되는 컨테이너 포맷 형식 및 코덱

IEC-Series에서 동영상을 재생하기 위해서는 아래 표에서 지원하는 컨테이너 포맷 형식 및 코덱을 확인하여 인코딩하시기 바랍니다.

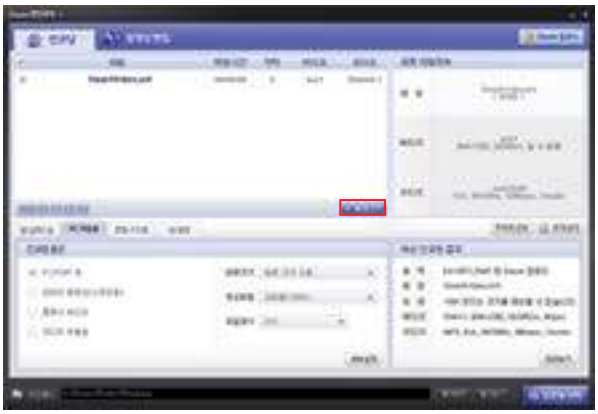
[표] IEC-Series에서 지원하는 컨테이너 포맷 형식 및 코덱 지원표

컨테이너 포맷 형식	비디오 코덱	오디오코덱	IEC667-Series	IEC1000-Series	
AVI	MPEG-1	MP3	오디오만 나옴	오디오만 나옴	
	MPEG-2 / MPEG-3				
	MPEG-4 V2	MP3	지원	지원	권장
	MPEG-4 V3				
	wmv1				
	wmv2				
WMV	wmv1	wma v1	지원 (음질/영상 조금 저하)	지원	
	wmv2				

1) 동영상 파일 포맷 형식 및 코덱 변경 방법

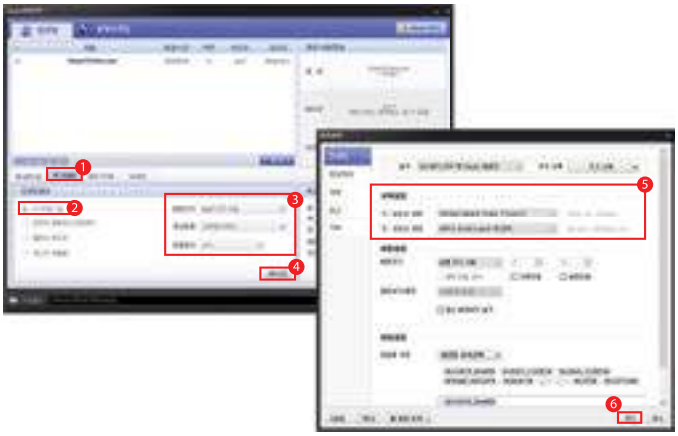
STEP-1 Daum 팟인코더 실행하기

Daum 팟인코더 실행 후 "불러오기" 버튼을 클릭하여 변경할 동영상 파일을 불러옵니다.



STEP-2 코덱 설정하기

1. “PC 저장용” 탭을 선택해 출력 형식을 설정합니다.
2. PC/PMP용 라디오 버튼을 선택합니다.
3. 화면 크기, 영상 화질, 파일 형식을 아래의 권장 내용을 참고하여 선택합니다.
4. 비디오 코덱과 오디오 코덱을 설정하기 위해 세부 설정 버튼을 클릭합니다.
5. “인코딩” – “코덱 설정”의 비디오 코덱과 오디오 코덱을 아래의 권장 내용을 참고하여 선택합니다.
6. 인코딩 설정이 완료되면 “확인” 버튼을 클릭하여 세부 설정창을 닫습니다.



권장 인코딩 옵션 설정 시 권장사항

1. 화면 크기는 영상의 크기가 클수록 영상 실행 속도가 저하될 수 있으므로 가급적 1024*768 이내의 화면 크기를 권장합니다.
2. 세부 설정의 “코덱 설정”에서 설정 시 IEC-Series 제품별 권장하는 속성값이 있습니다. 아래 표를 확인하여 설정하시기 바랍니다.

[표] 인코딩 코덱 설정 권장 속성표

제품	비디오 코덱	압축 방식	비트 레이트	키프레임 간격	프레임 (fps)	리사이즈 필터	오디오 코덱
IEC266 - Series	IEC266-Series는 CPU 성능상 영상 및 음성화질이 저하되어 사용을 권장하지 않습니다.						
IEC667 - Series	WMV7	1PASS-CBR	1000	3	10	BICUBIC	MPEG Audio Layer 3(CBR)
IEC1000 - Series	WMV7	1PASS-CBR	3000	3	30	BICUBIC	MPEG Audio Layer 3(CBR)

참고 비트 레이트, 프레임에 따른 CPU 및 파일 크기 비교

원본 영상이 800X480 크기에 비트 레이트 2600, 프레임(fps) 24, 용량 7.3MB라면 비트 레이트, 프레임 설정에 따라 파일 용량과 영상 재생 시 CPU 사용률 및 선명 정도, 끊김이 달라집니다. 비트 레이트가 낮은 경우 파일 용량이 작고 영상은 비선명하며 높은 경우에는 파일 용량이 커지고 영상은 선명합니다. 프레임이 낮은 경우 CPU 사용률이 적고 재생 시 조금씩 끊기며 높은 경우에는 CPU 사용률이 커지고 안 끊기며 정상적으로 재생됩니다.

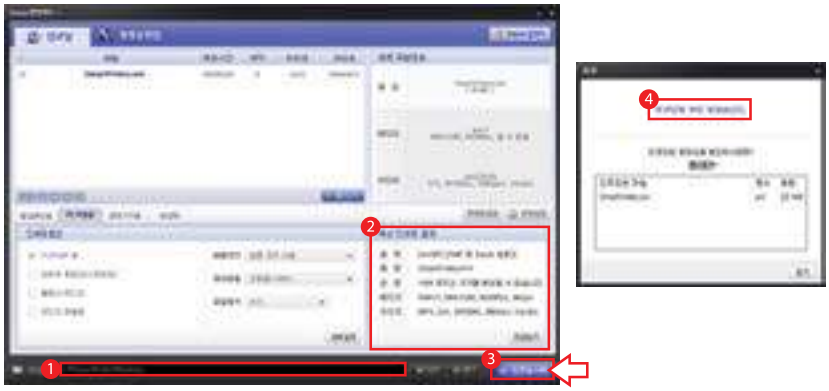
[표] 비트 레이트, 프레임에 따른 CPU 및 파일 크기 비교표

비트 레이트	프레임 (fps)	파일 용량 (MB)	영상 재생 시		
			CPU 사용률	선명한 정도	끊김 현상
낮음 (Ex : 100)	낮음(Ex : 5)	0.6	10~15%	비선명	조금씩 끊김
	높음(Ex : 60)	1	50~60%	비선명	안 끊김(정상)
높음 (Ex : 1000)	낮음(Ex : 5)	3	15~20%	선명	조금씩 끊김
	높음(Ex : 60)	8	50~60%	선명	많이 끊김

※ IEC1000-Series 기준입니다.

STEP-3	인코딩 시작하기

1. 하단의 “저장 폴더”를 확인하여 인코딩이 완료된 영상이 저장될 경로를 확인하거나 변경합니다.
2. “예상 인코딩 결과”를 통해 현재 설정한 변경될 코덱의 환경을 알맞게 설정했는지 확인합니다.
3. “인코딩 시작” 버튼을 클릭해 인코딩을 시작합니다.
4. 인코딩이 완료되면 알림창에 “인코딩이 완료되었습니다.”가 출력됩니다.



2) 프로그래밍 적용 가이드

STEP-1	동영상 파일 선택 및 Open하기

SmartPlayer가 사용되기 전 Initialization() 메소드를 호출하여 초기화를 해준 후 Open() 메소드를 호출해 재생할 동영상 파일의 경로와 동영상 화면크기를 설정하여 동영상 파일을 Open합니다. (단, iWidth, iHeight 값을 0으로 설정할 경우 화면의 크기는 SmartPlayer 컨트롤의 크기로 재생됩니다.)

※ 자세한 내용은 “Initialization(), Open() 메소드”를 참고하시기 바랍니다.

```
smartPlayer1.Initialization();  // SmartPlayer가 사용되기 전 초기화
smartPlayer1.Open( " SD Card₩₩Test1.exe " , 0, 0);  // 동영상 파일 Open 처리
```

STEP-2	동영상 파일 Play, Pause, Stop하기

Play(), Pause(), Stop() 메소드를 호출하여 Open된 동영상 파일을 시작, 일시 정지, 종료를 합니다.

※ 자세한 내용은 “Play(), Pause(), Stop() 메소드”를 참고하시기 바랍니다.

```
// 동영상 파일 재생, 일시 정지, 종료
smartPlayer1.Play();   smartPlayer1.Pause();   smartPlayer1.Stop();
```

STEP-3	동영상 파일의 소리 설정하기

Volume() 메소드를 호출하여 동영상 파일의 소리를 설정합니다.

※ 자세한 내용은 “Volume() 메소드”를 참고하시기 바랍니다.

```
smartPlayer1.Volume += 1000;  // 소리 크기 증가
smartPlayer1.Volume -= 1000;  // 소리 크기 감소
```

STEP-4	SmartPlayer 종료하기

SmartPlayer 프로그램을 종료할 경우 폼 클로징에서 Stop() 메소드 호출 후 Release() 메소드를 호출하여 프로그램 종료 시 메모리에 SmartPlayer 프로세스가 남아있는 것을 방지합니다.

※ 자세한 내용은 “Stop(), Release() 메소드”를 참고하시기 바랍니다.

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartPlayer1.Stop();     // Open된 동영상 파일을 Stop
  smartPlayer1.Release();  // SmartPlayer 프로세스를 완전히 제거
}
```

3) SmartPlayer 인터페이스 설명

SmartPlayer Component Interface		
속성		
CurrentPosition : uint	IsComplete : bool	MaxPosition : uint
Volume : int		
메소드		
Initialization() : void	Open() : bool (+1개 오버로드)	Pause() : bool
Play() : bool	Release() : void	Stop() : bool
ViewFullSize(bool bMode) : void		

프로퍼티(속성) CurrentPosition

현재 재생 중인 동영상의 재생 시점을 설정하거나 읽습니다.

- uint : 동영상 재생 시점 (단위 : 초)

C# 사용법

```
smartPlayer1.CurrentPosition = 30;  // 현재 Play 재생 시점을 30초로 설정
```

VB 사용법

```
smartPlayer1.CurrentPosition = 30
```

프로퍼티(속성) IsComplete

현재 재생 중인 동영상의 재생 완료 여부를 설정합니다.

- bool : 동영상의 재생 완료 여부
  - true : 재생 완료
  - false : 재생 중

C# 사용법

```
if (smartPlayer1.IsComplete == true)
{
  labMsg.Text = "완료!!!";
}
else
{
  labMsg.Text = "재생 중...";
}
```

VB 사용법

```
If smartPlayer1.IsComplete = True Then
  labMsg.Text = "완료!!!"
Else
  labMsg.Text = "재생 중..."
End If
```

프로퍼티(속성) MaxPosition

현재 Open되어 있는 동영상 파일의 총 재생 시간을 가져옵니다.

• uint : 동영상의 총 재생 시간 (단위 : 초)

C# 사용법

```
// 현재 Open되어 있는 동영상 파일의 재생 시간 표시
labMsg.Text = ((int)(smartPlayer1.MaxPosition / 60)).ToString( " 00 " ) + " : " +
((int)(smartPlayer1.MaxPosition % 60)).ToString( " 00 " );
```

VB 사용법

```
labMsg.Text = (CInt(smartPlayer1.MaxPosition / 60)).ToString( " 00 " ) + " : " +
(CInt(smartPlayer1.MaxPosition Mod 60)).ToString( " 00 " )
```

프로퍼티(속성) Volume

현재 재생 중인 동영상의 소리 크기를 설정하거나 가져옵니다.

• int : 동영상의 소리 크기 (최대값 : 10000)

C# 사용법

```
smartPlayer1.Volume = 1000;
```

VB 사용법

```
smartPlayer1.Volume = 1000
```

메소드(함수) Initialization

SmartPlayer가 사용되기 전에 폼 로드에서 호출하여 초기화합니다.

중요 최초 1회 실행되어야 합니다.

◉ void Initialization()

C# 사용법

```
private void Form1_Load(object sender, EventArgs e)
{
  smartPlayer1.Initialization();  // SmartPlayer가 사용되기 전 초기화
}
```

VB 사용법

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
  smartPlayer1.Initialization()
End Sub
```

메소드(함수) Open

재생할 동영상 파일의 경로와 동영상의 크기를 설정합니다.

◉ bool Open(string strFilePathName)

◉ bool Open(string strFilePathName, int iWidth, int iHeight)

[인자]

• string strFilePathName : 재생할 동영상 파일의 경로와 파일명을 설정

• int iWidth : 화면에 보여질 동영상의 폭 설정

• int iHeight : 화면에 보여질 동영상의 높이 설정

주의 iWidth, iHeight 설정 시 주의사항

동영상 파일을 Open하면서 재생할 화면의 크기를 설정하며, 재생 중에는 화면의 크기를 변경할 수 없습니다. (단, 전체 화면 기능은 사용 가능합니다.) 만약 iWidth, iHeight 값을 0으로 설정할 경우 화면의 크기는 SmartPlayer 컨트롤의 크기로 재생됩니다.

[리턴값]

- bool : 동영상 파일의 Open 여부
  - true : Open 성공
  - false : Open 실패

C# 사용법

```
OpenFileDialog openDlg = new OpenFileDialog();
if (openDlg.ShowDialog() == DialogResult.OK)
{
  // 선택된 동영상 파일을 Open 처리, SmartPlayer 컨트롤과 같은 크기로 재생됨
  smartPlayer1.Open(openDlg.FileName, 0, 0);
}
```

VB 사용법

```
Dim openDlg As OpenFileDialog = New OpenFileDialog()
If (openDlg.ShowDialog() = DialogResult.OK) Then
    smartPlayer1.Open(openDlg.FileName, 0, 0)
End If
```

메소드(함수) Pause, Play, Stop

- Pause() : 현재 재생 중인 동영상을 일시 정지합니다. 다시 재생하려면 Play() 메소드를 호출합니다.
- Play() : 현재 Open된 동영상 파일을 재생합니다.
- Stop() : 현재 재생 중인 동영상을 종료합니다.

◉ bool Pause()
◉ bool Play()
◉ void Stop()

[리턴값]

- bool : 동영상 파일의 Pause, Play, Stop 성공 여부
  - true : 성공
  - false : 실패

C# 사용법

```
smartPlayer1.Pause();
smartPlayer1.Play();
smartPlayer1.Stop();
```

VB 사용법

```
smartPlayer1.Pause()
smartPlayer1.Play()
smartPlayer1.Stop()
```

메소드(함수) Release

동영상 처리 관련 리소스를 모두 해제합니다. Stop() 메소드 호출 이후 프로그램 종료 시 메모리에 SmartPlayer 프로세스가 남아있는 것을 방지하는 기능입니다.

◉ void Release()

C# 사용법

```
private void Form1_Closing(object sender, CancelEventArgs e)
{
  smartPlayer1.Stop();
  smartPlayer1.Release();
}
```

VB 사용법

```
Private Sub Form1_Closing(ByVal sender As System.Object, ByVal e As System.Component
Model.CancelEventArgs) Handles MyBase.Closing
  smartPlayer1.Stop()
  smartPlayer1.Release()
End Sub
```

메소드(함수) **ViewFullSize**

재생 중 화면의 크기를 전체 화면으로 변경할 수 있는 기능으로 영상 비율에 따라서 상단과 하단이 검정색으로 표시될 수 있습니다.

◉ `void ViewFullSize(bool bMode)`

[인자]

- `bool bMode` : 전체 화면 표시 여부
  - `true` : 전체 화면으로 표시 / `false` : 이전 화면 크기로 표시

C# 사용법

```
smartPlayer1.ViewFullSize(true);  // 전체 화면 모드
```

VB 사용법

```
smartPlayer1.ViewFullSize(True)
```

4) SmartPlayer 예제 사용하기

SmartPlayer를 활용한 예제를 제공하고 있으며, 예제 파일은 다음 경로에서 다운로드하여 참고하시기 바랍니다.

[예제 파일 다운로드 위치]

홈페이지(www.hnsts.co.kr) → 자료실 → SmartX Framework 예제 파일 다운로드 → "SmartPlayer"

14. SmartLaunch

SmartLaunch는 바탕화면에 사용할 프로그램의 바로가기를 만들고 재부팅해도 바로가기가 제거되지 않도록 해주는 컴포넌트가 아닌 프로그램입니다. 이 프로그램은 IEC667/1000-Series에서 사용 가능합니다.

■ 바탕화면에 사용할 프로그램 바로가기 생성 지원
■ 제품 재부팅 후에도 바로가기 유지
■ Flash Disk₩Run₩Run1 폴더를 이용한 RunTime 모드 지원

중요 SmartLaunch는 RunTime 모드에서만 지원되는 기능입니다.

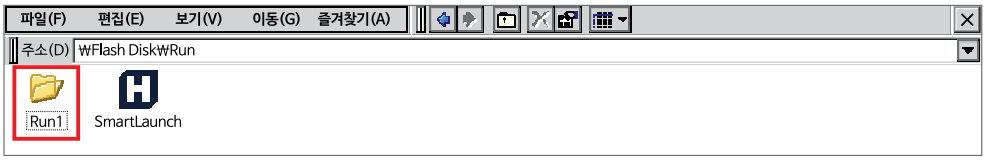
주의 RunTime 모드에서 SmartLaunch와 SmartUpdate를 같이 사용할 경우

RunTime 모드에서 SmartLaunch를 사용하면 Run 폴더에 실행 프로그램이 있어도 기존의 RunTime 모드 동작과 다르게 제품 부팅 후 바로 실행되지 않습니다. SmartUpdate를 사용하여 프로그램을 업데이트한다면 아래 STEP을 참고하시기 바랍니다.

[STEP-1] SmartUpdate의 ToFilePath 속성값 "Flash Disk₩₩Run₩₩Run1"으로 설정

[STEP-2] Flash Disk₩Run 폴더 안에 Run1 폴더 생성 – 실행할 프로그램 파일 저장

Flash Disk₩Run 폴더에 Run1 폴더를 생성합니다. Run1 폴더 안에 부팅 시 실행할 exe 파일과 dll 파일을 저장한 후 딥(Dip) 스위치 1번을 ON으로 하여 RunTime 모드로 재부팅 하면 Run1에 저장된 프로그램이 실행됩니다. (기존의 RunTime 모드와 동일하게 동작)



※ SmartLaunch를 사용할 경우 Run 폴더는 Flash Disk₩Run 폴더에서 Flash Disk₩Run₩Run1으로 변경됩니다.

1) 바로가기 생성 및 등록하기

SmartLaunch를 사용하기 앞서 "SmartLaunch 프로그램"을 다운로드합니다.

다운로드 경로 ▶ 자사홈페이지 → 자료실 → SmartX Framework 관련 → "SmartLaunch 프로그램"

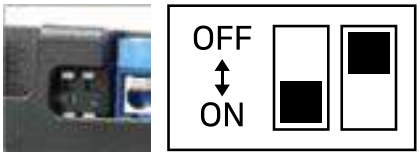
[STEP-1] Flash Disk에 Run 폴더 생성 – SmartLaunch.exe 파일 저장

제품의 Flash Disk에 Run 폴더를 생성한 뒤 Run 폴더 안에 다운로드한 SmartLaunch를 넣어줍니다.

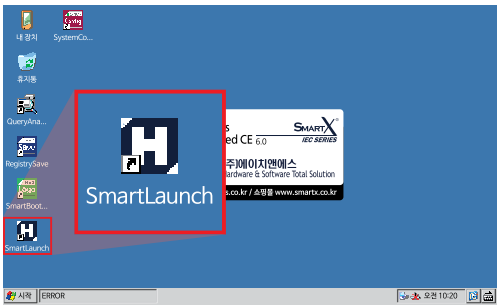
중요 반드시 SmartLaunch 프로그램은 FlashDisk₩Run 폴더에 저장해야 합니다.

[STEP-2] 제품의 딥(Dip) 스위치 1번 ON 설정 – 제품 재부팅

제품의 딥(Dip) 스위치 1번을 ON으로 설정 후 제품을 재부팅하면 바탕화면에 SmartLaunch 바로가기가 생성됩니다.



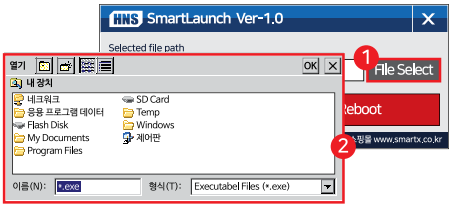
참조 딥(Dip) 스위치 관련하여 자세한 내용은 "홈페이지(www.hnsts.co.kr) → 자료실 → 제품관련 → IEC-Series 제품 매뉴얼 → 동작 모드 관련"을 참조하시기 바랍니다.



[STEP-3] SmartLauch 실행 → [File Select] 클릭 → 바로가기 생성할 프로그램 선택 → [OK] 클릭

SmartLaunch를 실행 후 "File Select" 버튼을 클릭합니다. 바로가기 생성할 프로그램 파일을 선택 후 "OK" 버튼을 클릭하여 경로를 설정합니다.

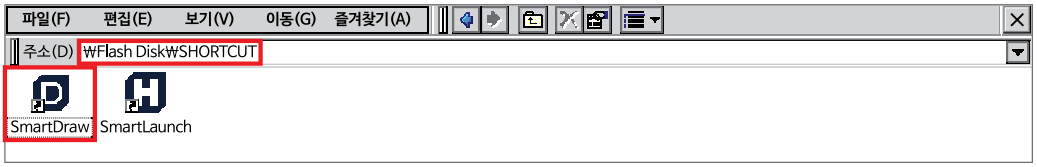
참고 외부저장장치(SD Card, USB)에 저장된 프로그램도 바로가기 생성이 가능합니다.



주의 바로가기 대상 프로그램 파일은 미리 준비되어 있어야 합니다.

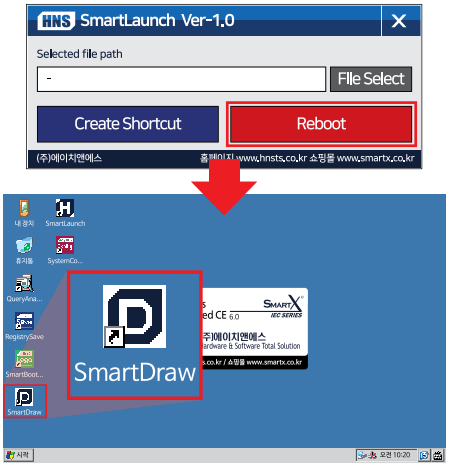
[STEP-4] [Create Shortcut] 클릭 → [OK] 클릭

"Create Shortcut" 버튼을 클릭 후 "OK" 버튼을 클릭하면 Flash Disk에 SHORTCUT 폴더가 생성되어 선택한 프로그램 파일과 SmartLaunch의 바로가기가 SHORTCUT 폴더 안에 생성됩니다.



[STEP-5] [Reboot] 클릭 → 바탕화면 확인

"Reboot" 버튼을 클릭하여 재부팅을 진행합니다. 재부팅 후 바탕화면에 바로가기가 생성됩니다.

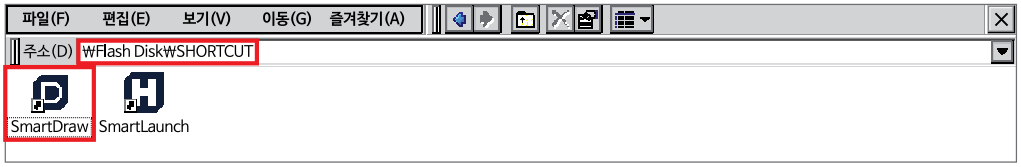


주의 딥(Dip) 스위치 1번을 ON으로 하여 사용하는 경우에만 바탕화면에 바로가기가 생성됩니다.

2) 바로가기 삭제하기

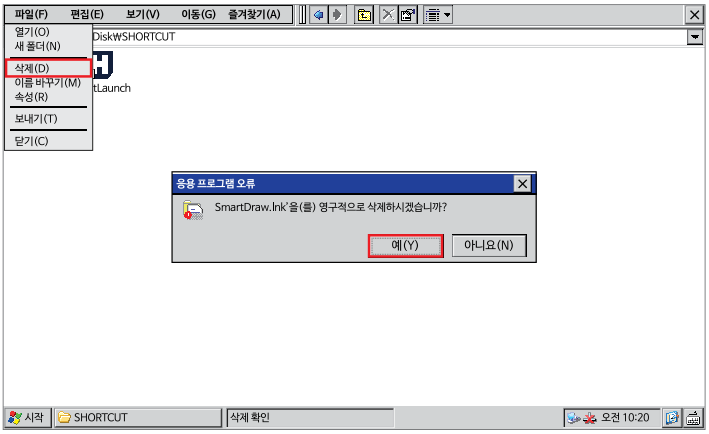
[STEP-1] Flash Disk의 SHORTCUT 폴더에서 삭제할 프로그램(바로가기) 선택

제품의 Flash Disk에 SHORTCUT 폴더에서 삭제할 프로그램(바로가기)을 선택합니다.



[STEP-2] [파일(F)] 탭 클릭 → [삭제(D)] → [예(Y)] 버튼 클릭

삭제할 프로그램(바로가기)를 선택한 상태로 "파일(F)" 탭 클릭 후 "삭제(D)"를 클릭합니다. 삭제 확인창이 뜨면 "예(Y)" 버튼을 클릭하여 바로가기를 삭제합니다.



[Memo]

SMARTX® Framework

프로그래밍 가이드

1쇄 : 2020년 6월 30일
2쇄 : 2021년 1월 15일

제품 및 자세한 문의 사항은 아래의 연락처로 연락 및 메일 문의주시기 바랍니다. 감사합니다.

㈜에이치앤에스
서울특별시 금천구 가산디지털1로 181, 1505호(가산 W센터)
대표전화 : 02-6402-8001 / 팩스 : 02-6442-9775

부서안내	연락처	직통 전화	이메일
제품 구매 및 견적문의	02-6402-8001(내선 1번)	070-7094-5770	sales@hnsts.co.kr
하드웨어 기술문의	02-6402-8001(내선 2번)	070-7094-5001	hns@hnsts.co.kr
소프트웨어 기술문의	02-6402-8001(내선 3번)	070-7094-5002	app@smartx.co.kr
제품서비스 기술문의	02-6402-8001(내선 4번)	070-7094-5003	tech@smartx.co.kr

홈페이지 : www.hnsts.co.kr / 쇼핑몰(제품 구매) : www.hnsstore.co.kr