6장 델파이 5 개발자 가이드 소스 코딩 표준 문서

* 일반적인 소스 코드 포매팅 규칙
* 오브젝트 파스칼
* 파일
* 폼과 데이터 모듈
* 패키지
* 컴포넌트

이 문서는 이 책 델파이 5 개발자 가이드에 쓰인 코딩 표준을 설명하는 문서이다. 몇 가지 예외가 있긴 하지만, 이 문서는 볼랜드 인터내셔널(Borland International)이 사용하는 "비공식적" 포매팅 지침을 따른다. 델파이 5 개발자 가이드에 이 문서를 포함시키는 목적은 팀 단위 개발에서 각 팀원들이 일관성 있는 코딩 스타일을 유지하는 데 도움을 주기 위해서이다. 다시 말하면, 한 팀원이 쓴 코드를 다른 팀원이 쉽게 이해할 수 있도록 하는 것이 이 문서의 목적이다. 일정한 규칙에 따라 코드를 작성하면 코드를 이해하기가 매우 쉬워진다.

이 문서는 코딩 표준에 대한 모든 내용들을 담고 있지는 않다. 다만, 코딩 표준을 만들고 적용하고자 하는 독자에게 최소한의 힌트가 될 수는 있을 것이다. 독자의 상황에 따라 이 문서에 나온 표준들을 적절히 수정하기 바란다. 그러나 이 문서에 나온 표준들, 즉 볼랜드의 개발자들이 사용하는 표준과 너무 멀어지는 것은 좋지 않다고 생각한다. 나중에 팀에 새로운 팀원이 들어왔을 때 그 사람이 델파이 경험자라면 이 문서의 표준에 어느 정도 익숙할 가능성이 많기 때문이다. 다른 코딩 표준 문서들과 마찬가지로, 이 문서 역시 계속 개선해 나갈 것이다. 이 문서의 최신 버전은 www.xapware.com/ddg에서 얻을 수 있다(역주 : 번역에 사용한 문서는 98/11/25일 현재 게시되어 있는 문서이고, 이 책은 개정판으로서 홈페이지에는 적용되지 않았음). 그리고 이 문서는 사용자 인터페이스의 표준에 대해서는 다루지 않는다. 물론, 사용자 인터페이스도 매우 중요한 요소이기는 하지만, 코딩 표준과는 별개의 것이기 때문이다. 사용자 인터페이스 설계에 대해서는 다른 책들도 많이 있으므로 이 문서에서는 이야기하지 않겠다. 시중의 책들이나 MSDN(Microsoftware Developers Network)의 사용자 인터페이스 지침을 참고하기 바란다.

일반적인 소스 코드 포매팅 규칙

들여쓰기

들여쓰기는 두 글자 단위로 한다. 그리고 소스 파일에 탭 문자를 저장하는 것은 좋지 않다. 왜냐하면, 텍스트 편집기나 소스 관리 도구(인쇄, 저장, 버전 관리 도구 등)들마다 탭의 크기가 다르기 때문이다.

소스 파일에 탭 문자를 저장하지 않으려면 Editor Options 대화상자(Tools | Editor 메뉴)의 General 페이지에 있는 "Use tab character"와 "Optimal fill" 체크박스를 끄면 된다.

가로 길이

가로 길이는 80자로 한다. 일반적으로, 한 줄이 80자가 넘는 코드를 쓰는 경우는 없지만, 80자라는 한계를 엄격히 지킬 필요는 없다. 어쨌든, 코드가 80자가 넘으면 가능한 한 쉼표나 연산자 등을 이용해서 다음 줄로 내리는 것이 좋다. 한 줄의 코드를 다음 줄로 내리는 경우는 원래 줄보다 두 글자만큼 들여써야 읽기 쉽다.

Begin..End 쌍

begin문은 하나의 줄로 쓴다. 예를 들어서, 다음 코드의 첫 번째 줄은 틀리며, 두 번째 줄이 옳다.

```
for I := 0 to 10 do begin // begin과 for가 같은 줄에 있으므로 틀리다.
for I := 0 to 10 do       // begin이 개별적인 줄에 있으므로 옳다.
begin
```

단, begin이 else절에 포함될 때에는 같은 줄에 쓴다.

```
if some statement = then
begin
  ...
end
else begin
  SomeOtherStatement;
end;
```

end 역시 하나의 줄로 쓴다.

begin문이 else절의 일부가 아닌 이상, 그 begin에 해당하는 else는 항상 begin과 들여쓰기를 맞춘다.

오브젝트 파스칼

괄호

열린 괄호와 다음 문자 사이에 공백이 들어가서는 안된다. 마찬가지로, 닫힌 괄호와 그 앞의 글자 사이에 공백이 들어가서는 안된다. 다음은 괄호 사용의 옳은 예와 틀린 예이다.

```
CallProc( AParameter ); // 틀림.
CallProc(AParameter);  // 옳음.
```

문장에 필요 없는 괄호가 있어서는 안된다. 괄호가 꼭 필요한 곳에만 괄호를 넣어야 한다. 다음은 틀린 예와 옳은 예이다.

```
if (I = 42) then            // 틀림 - 필요 없는 괄호.
if (I = 42) or (J = 42) then// 옳음 - 괄호가 필요함.
```

예약어와 키워드

오브젝트 파스칼 언어의 예약어(reserved word)와 키워드(key word)는 반드시 소문자로만 써야 한다.

프로시저와 함수(루틴)

명명/포매팅 규칙

루틴 이름은 읽기 쉽도록 항상 대문자로 시작하고, 대·소문자를 섞어서 쓴다. 다음은 잘못 붙인 프로시저 이름의 예이다.

```
procedure thisisapoorlyformattedroutinename;
```

다음은 대·소문자를 제대로 섞어서 만든 프로시저 이름이다.

```
procedure ThisIsMuchMoreReadableRoutineName;
```

루틴 이름을 통해 그 루틴의 기능이나 용도를 알 수 있어야 한다. 다음과 같이 기능에 해당하는 동사를 먼저 쓰고, 목적어를 뒤에 쓴다.

```
procedure FormatHardDrive;
```

입력 인자의 값을 설정하는 루틴은 다음과 같이 Set으로 시작해야 한다.

```
procedure SetUserName;
```

어떤 값을 가져오는 루틴은 다음과 같이 Get으로 시작한다.

```
function GetUserName: string;
```

정규 인자

포매팅

같은 데이터형의 정규 인자(formal parameter)들은 가능한 한 하나의 문구로 결합한다.

```
procedure Foo(Param1, Param2, Param3: Integer; Param4: string);
```

명명 규칙

정규 인자들에는 기능과 용도에 맞는 이름을 붙여야 하며, 일반적으로 루틴에 넘겨주는 식별자의 이름에 기초해서 이름을 만든다. 다음과 같이 인자의 이름이 A로 시작하는 경우도 많다.

```
procedure SomeProc(AUserName: string; AUserAge: integer);
```

A를 쓰는 이유는 인자 이름이 클래스의 속성이나 필드 이름과 같아서 생기는 혼동을 피하기 위한 것이다.

인자들의 순서

다음의 정규 인자 순서는 레지스터 호출 규약을 고려해서 프로그램의 수행을 최적화하는 것이 목적이다.

가장 자주 쓰이는(호출하는 쪽에서) 인자들은 첫 번째에 와야 한다. 그리고 가장 자주 쓰이지 않는 인자부터 오른쪽에서 왼쪽으로 배치한다.

입력 인자들은 출력 인자들의 왼쪽에 와야 한다.

가장 일반적인 인자를 먼저 놓고, 가장 구체적인 인자를 뒤에 놓는다(왼쪽에서 오른쪽으로). 예를 들면, 다음과 같다.

```
SomeProc(APlanet, AContinent, ACountry, AState, ACity).
```

TObject형의 Sender라는 인자가 첫 번째 인자인 이벤트 처리 루틴 같은 경우에는 위의 규칙에서 제외된다.

상수 인자

인자가 레코드나 배열, ShortString, interface형이면 루틴 안에서 그 값이 변하지 않는다. 그런 루틴의 인자들은 const 지시자를 사용해서 상수 인자로 지정해야 한다. 그렇게 하면 컴파일러는 인자 전달의 코드를 최적화할 수 있다.

다른 형의 인자들 중에서도 루틴 안에서 값이 변하지 않는 것이라면 const를 써줄 수 있다. 효율적인 면에서는 아무 이득도 없지만, 코드를 읽고 이해하는 데 도움이 된다.

이름 중복

동일한 이름의 루틴을 가진 두 개의 유닛을 사용할 경우, 코드 안에서 그 루틴을 호출하면 uses절의 뒷쪽에 있는 유닛의 루틴이 호출된다. 루틴의 호출을 uses절에 의존하는 것은 좋지 않으므로 다음과 같이 항상 유닛 이름을 앞에 추가해서 어떤 유닛의 루틴인지 명확히 해야 한다.

```
SysUtils.FindClose(SR);
```

또는

```
Windows.FindClose(Handle);
```

변수

변수 명명/포매팅 규칙

변수에는 목적과 기능에 맞는 이름을 붙인다.

루프 제어 변수는 보통 I, J, K 같은 한 글자의 이름을 붙이곤 하지만, UserIndex 같이 좀더 의미 있는 이름을 붙이는 것이 좋다.

Boolean 변수의 경우, True와 False가 의미하는 것이 무엇인지를 한눈에 알 수 있는 이름을 붙여야 한다.

지역 변수

루틴 안에서 사용되는 지역 변수들의 명명/포매팅 규칙도 기본적으로 위에서 말한 변수의 명명/포매팅 규칙에 따른다. 임시 변수들은 적당한 이름을 붙이면 된다.

지역 변수의 초기화가 필요한 경우에는 루틴의 제일 처음 부분에서 초기화한다. AnsiString 지역 변수는 자동으로 빈 문자열로 초기화되며, interface와 dispinterface형 지역 변수는 자동으로 nil로 초기화된다. 또한, Variant와 OleVariant형 지역 변수는 자동으로 Unassigned로 초기화된다.

전역 변수의 사용

전역 변수의 사용은 피하는 것이 좋다. 그러나 전역 변수가 꼭 필요한 경우가 있는데, 그런 경우 전역 변수가 적용되는 범위를 항상 염두에 두고 있어야 한다. 예를 들어서, 한 유닛의 전역 변수는 다른 유닛의 uses절에 그 유닛이 포함되어 있지 않는 한 그 유닛의 implementation부 안에서만 사용할 수 있다.

여러 유닛에서 사용할 전역 데이터(전역 변수, 전역 상수 등등)는 다른 유닛들이 모두 사용하는(uses절을 통해) 공통 유닛에 넣도록 한다.

전역 데이터를 var절 안에서 직접 초기화할 수도 있다. 전역 변수는 항상 자동으로 빈 값(0, il, ‘’, Unassigned 등)으로 초기화되므로 코드에서 직접 빈 값을 초기화하지 않는 것이 좋다. 그 이유 중 하나는, 빈 값으로 초기화되는 전역 데이터들은 실행 파일의 공간을 차지하지 않는다는 것이다. 빈 값으로 초기화되는 전역 데이터는 애플리케이션이 시작할 때 메모리 안에서만 할당되는 가상 데이터 세그먼트에 저장된다. 빈 값으로 자동 초기화되지 않는 전역 데이터들은 하드 디스크의 exe 파일에 저장된다.

형

대·소문자 규칙

형의 이름이 예약어인 경우는 소문자로만 쓴다. Win32 API형들은 모두 대문자로만 쓰는 것이 일반적이며, 그렇지 않은 경우는 Windows.pas나 기타 API 유닛에 나타나는 예를 따른다. 그 외의 형들은 첫 번째 글자를 대문자로 하고, 대·소문자를 적절히 섞는다. 다음을 참고하기 바란다.

```
var
  MyString: string;  // 예약어
  WindowHandle: HWND; // Win32 API형
  I: Integer;        // System 유닛에 정의된 예를 따른다.
```

부동 소수점형들

Real형은 이전 버전의 델파이 코드와 하위 호환성이 필요한 경우에만 사용하는 것이 좋다. 부동 소수점이 필요한 곳에는 Double을 사용한다. Double은 CPU와 버스에 최적화된 형이며, IEEE가 정의한 표준 데이터형이다. Extended는 Double보다 더 큰 범위의 부동 소수점 데이터가 필요한 경우에만 사용한다. Extended형은 인텔 고유의 데이터형이며, 자바가 지원하지 않는다. Single은 부동 소수점 변수의 바이트 크기가 중요한 경우에만(예를 들어, 다른 언어의 DLL을 사용할 때) 사용하도록 한다.

열거형(Enumerated Types)

열거형 이름은 목적에 맞도록 만들어야 한다. 다른 모든 형들과 같이 열거형 이름도 type을 뜻하는 T로 시작해야 한다. 열거형의 식별자 이름들에는 열거형 이름에서 따온 두세 글자를 소문자로 앞에 붙인다. 예를 들면, 다음과 같다.

```
TSongType = (stRock, stClassical, stCountry, stAlternative, stHeavyMetal,
stRB);
```

열거형의 인스턴스 변수는 열거형 이름에서 T를 뺀 이름을 사용하되, 좀더 의미를 정확히 할 필요가 있으면 FavoriteSongType1, FavoriteSongType2와 같이 추가적인 글자를 붙여도 된다.

Variant와 OleVariant형

일반적으로, Variant나 OleVariant형은 사용하지 않는 것이 좋지만, COM이나 데이터베이스 애플리케이션들처럼 실행 시점에서만 데이터형이 결정되는 경우에는 두 형이 필요하다. OLE 자동화나 ActiveX 컨트롤 등 COM 기반 프로그래밍에는 OleVariant를 사용하고, COM 기반이 아닌 경우에는 Variant를 사용한다. Variant는 델파이 고유의 문자열형을 효율적으로 다룰 수 있는 반면(string var와 같이), OleVariant는 모든 문자열들을 Ole 문자열(WideChar 문자열들)들로 변환하며, 참조 횟수를 관리하지 않기 때문에 항상 복사를 수행한다(델파이 문자열들은 copy-on-write 기법을 사용한다). 따라서, COM에 관련된 것이 아니라면 Variant를 사용하는 것이 더 유리하다.

구조체

배열형

배열형 이름은 목적에 맞도록 만들어야 하며, 다른 모든 형들과 같이 배열형의 이름도 type을 뜻하는 T로 시작해야 한다. 배열형을 가리키는 포인터를 선언할 경우, 그 포인터의 이름은 P로 시작해야 하며, 배열형 선언의 바로 위에서 선언해야 한다. 다음은 배열형 포인터 선언의 예이다.

```
type
  PCycleArray = ^TCycleArray;
  TCycleArray = array[1..100] of integer;
```

일반적으로, 배열형의 인스턴스 변수 이름은 배열형 이름에서 T를 뺀 것을 사용한다.

레코드형

레코드형 이름은 목적에 맞도록 만들어야 하며, 다른 모든 형들과 같이 레코드형의 이름도 type을 뜻하는 T로 시작해야 한다. 레코드형을 가리키는 포인터를 선언할 경우에 그 포인터의 이름은 P로 시작해야 하며, 레코드형 선언의 바로 위에서 선언해야 한다. 레코드의 각 필드의 데이터형들은 다음과 같이 열을 맞춰주는 것이 좋다.

```
type
  PEmployee = ^TEmployee;
  TEmployee = record
    EmployeeName: string
    EmployeeRate: Double;
  end;
```

명령문

if문

if/then/else 구조에서 실행 코드는 else보다는 then절에서 실행되도록 조건식을 만들어야 한다.

if문이 계속 연결되는 것은 좋지 않으며, 가능한 한 case문을 사용하는 것이 좋다.

if문을 다섯 단계 이상 중첩시키지 말아야 한다. 코드의 논리 구조 자체를 다시 고려하는 것이 좋다.

if문 안에 필요 없는 괄호를 중첩시키지 말아야 한다.

하나의 if문 안에 하나 이상의 조건식들이 있는 경우, 조건식의 계산이 간단한 것부터 배치한다. 그러면 컴파일러의 조건식 평가 최적화를 최대한 활용할 수 있다(조건식 판단 최적화란, 예를 들어 조건식들에 and가 포함되어 있는 경우에 첫 번째 조건식이 false이면 조건식 전체의 결과는 무조건 false가 되므로 나머지 조건식들은 아예 평가하지 않는 기법을 말한다). 예를 들어서, Condition1이 Condition2보다 빠르고, Condition2가 Condition3보다 빠르다면 if문은 다음과 같이 되어야 한다.

```
if Condition1 and Condition2 and Condition3 then
```

case문

일반적인 규칙

한 case문 안의 개별 경우(case)들은 숫자 또는 알파벳 순서대로 배치해야 한다.

각 경우의 실행 부분은 최대한 간단하게 해야 하며, 일반적으로 4, 5줄을 넘기지 않도록 한다. 만일 그 이상의 작업이 필요하다면 개별적인 프로시저나 함수로 만들어서 호출해야 한다.

case문에서 else절은 적절한 기본 처리나 에러 감지를 위해서만 사용한다.

포매팅

case문의 들여쓰기나 명명 규칙은 다른 문장 구조의 들여쓰기나 명명 규칙에 따른다.

while문

Exit 프로시저로 while 루프를 벗어나는 것은 피해야 한다. 가능한 한 루프 조건을 통해 정식으로 루프를 벗어나야 한다.

while 루프에 필요한 모든 초기화 코드는 while 루프로 진입하기 바로 전에 있어야 하며, 루프 초기화 코드와 while 루프 사이에 루프와 관련이 없는 코드들이 있어서는 안된다.

루프의 마무리 작업은 루프 바로 다음에서 수행해야 한다.

for문

특정한 범위의 숫자가 특정한 수치만큼 증가하는 루프인 경우는 while 대신 for문을 쓴다.

repeat문

repeat문도 while문과 동일한 규칙을 따른다.

with문

일반적인 규칙

with문은 가능한 한 조금만 사용하는 것이 좋다. with문을 남용하거나 with문 안에 여러 개의 객체, 레코드들을 사용하는 것은 피해야 한다. 다음 예를 보자.

```
with Record1, Record2 do
```

이렇게 하면 코드가 혼란스러워지며, 버그를 잡아내기도 어렵다.

포매팅

with문도 이 문서의 명명 규칙이나 들여쓰기 규칙을 따른다.

구조적 예외 처리

일반적인 규칙

에러 교정이나 자원(메모리, 핸들 등등) 보호에 예외 처리를 아끼지 말아야 한다. 자원 할당을 할 때에는 반드시 try..finally를 이용해서 자원의 적절한 해제를 보장하도록 해야 한다. 단, 유닛의 initialization/finalization부나 객체의 생성자/소멸자 메소드에서 자원을 할당/해제하는 경우는 try..finally를 사용하지 않는다.

try..finally의 사용

가능한 한 각각의 할당 작업마다 개별적인 try..finally 블록을 사용한다. 예를 들어서, 다음과 같은 코드는 버그 가능성을 가지고 있다.

```
SomeClass1 := TSomeClass.Create
SomeClass2 := TSomeClass.Create;
try
  { 작업을 수행 }
finally
  SomeClass1.Free;
  SomeClass2.Free;
end;
```

위의 코드를 다음과 같이 좀더 안전한 형태로 만들어야 한다.

```
SomeClass1 := TSomeClass.Create
try
  SomeClass2 := TSomeClass.Create;
  try
    { 작업을 수행 }
  finally
    SomeClass2.Free;
  end;
finally
  SomeClass1.Free;
end;
```

try..except의 사용

try..except는 예외가 발생했을 때 특정 작업을 수행할 필요가 있을 때만 사용한다. 일반적으로, 화면에 에러 메시지를 띄우는 용도로만 try..except를 사용하는 것은 좋지 않다. 에러 메시지 표시는 Application 객체가 자동적으로 수행하기 때문이다. except 절에서 어떤 작업을 수행한 후 기본 예외 처리부로 예외를 넘기려면 raise문을 이용해서 예외를 재발생시켜야 한다.

try..except..else 구조

try..except에서 else절을 사용하는 것은 피해야 한다. except절에서 처리하지 않은 모든 예외들은 else로 넘어가므로 심각한 예외인데도 그냥 무시될 가능성이 있기 때문이다.

클래스

명명/포매팅 규칙

클래스형의 이름은 그 클래스형의 목적에 맞도록 만들어야 하며, 다른 모든 형들과 같이 클래스형의 이름도 type을 뜻하는 T로 시작해야 한다.

```
type
  TCustomer = class(TObject)
```

클래스의 인스턴스 이름은 일반적으로 클래스형 이름에서 T를 뺀 것을 사용한다.

```
 var
  Customer: TCustomer;
```



컴포넌트형의 이름에 대해서는 "컴포넌트형 명명 규칙"을 보기 바란다.

필드

명명/포매팅 규칙

클래스 필드 이름들은 변수 식별자 명명 규칙을 따르되, 필드를 뜻하는 F로 시작하도록 한다.

사용 범위

모든 필드들은 private로 선언한다. 필드를 클래스 범위 외부에서 사용하게 하려면 속성을 통해서만 접근할 수 있도록 한다.

메소드

명명/포매팅 규칙

메소드 이름은 이 문서의 프로시저/함수(루틴) 명명 규칙을 따른다.

정적 메소드의 사용

자손 클래스들에게 상속하지 않을 메소드는 정적 메소드(static method)로 만든다.

가상/동적 메소드의 사용

자손 클래스들이 상속해서 재정의할 메소드는 가상(virtual) 메소드로 선언한다. 동적(dynamic) 메소드는 상속할 자손 클래스들이 많은 경우(직접 또는 간접적으로)에만 사용한다. 예를 들어서, 자주 재정의되지 않는 메소드를 가진 클래스를 100개의 클래스가 상속받는 경우, 그 메소드를 동적 메소드로 선언하면 메모리 사용을 최소화할 수 있다.

추상 메소드의 사용

인스턴스가 생성될 클래스에는 추상(abstract) 메소드를 만들지 않는다. 인스턴스를 생성하지 않을 클래스에만 추상 메소드를 만들어야 한다.

속성 조작 메소드

모든 속성 조작 메소드는 클래스 정의부의 private나 protected절에서 선언해야 한다.

속성 조작 메소드의 이름은 일반적인 프로시저/함수(루틴) 명명 규칙을 따른다. read 메소드 이름은 Get으로 시작한다. write 메소드는 Set으로 시작한다. write 메소드의 인자 이름은 Value로 해야 하며, 그 인자의 형은 속성에 관련된 필드의 형을 따른다.

다음은 그 예이다.

```
TSomeClass = class(TObject)
private
  FSomeField: Integer;
protected
  function GetSomeField: Integer;
  procedure SetSomeField( Value: Integer);
public
  property SomeField: Integer read GetSomeField write SetSomeField;
end;
```

속성

명명/포매팅 규칙

속성은 클래스의 private 필드를 조작하기 위한 것이며, 필드 이름에서 제일 앞의 F를 뺀 것을 이름으로 사용한다.

속성 이름은 동사가 아닌 명사로 해야 한다. 속성은 데이터를 뜻하며, 메소드는 동작을 뜻한다.

배열 속성 이름은 복수로 한다. 일반 속성은 단수로 한다.

조작 메소드의 사용

private 필드를 대표하는 속성의 wrirte 조작 메소드는 최소한 적게 만드는 것이 좋다.

파일

프로젝트 파일

명명 규칙

프로젝트 파일 이름은 이름만으로 어떤 프로젝트인지 알 수 있도록 붙인다. 예를 들어서, 델파이 5 개발자 가이드 버그 관리자(Delphi 5 Developer's Guide Bug Manager) 프로젝트의 이름은 DDGBugs.dpr이다. 또 다른 예로, 시스템 정보(System Information) 프로그램의 프로젝트는 SysInfo.dpr 정도가 적당하다.

폼 파일

명명 규칙

폼 파일은 폼의 용도를 나타내는 단어 뒤에 Frm을 붙인 이름을 사용한다. 예를 들어서, About 폼의 파일 이름은 AboutFrm.dfm으로 한다. 또한, 메인 폼의 파일 이름은 MainFrm.dfm이 된다.

데이터 모듈 파일

명명 규칙

데이터 모듈 파일은 데이터 모듈의 용도를 나타내는 단어 뒤에 DM을 붙인 이름을 사용한다. 예를 들어서, 고객(Customers) 데이터 모듈 파일의 이름은 Custmoers DM.dfm으로 한다.

원격 데이터 모듈 파일

명명 규칙

원격 데이터 모듈 파일은 원격 데이터 모듈의 용도를 나타내는 단어 뒤에 RDM을 붙인 이름을 사용한다. 예를 들어서, 고객(Customers) 원격 데이터 모듈의 이름은 CustmoersRDM.dfm으로 한다.

유닛 파일

유닛의 일반적인 구조

유닛 이름

유닛 파일은 용도를 알 수 있는 이름을 붙여야 한다. 예를 들어서, 애플리케이션의 메인 폼이 담긴 유닛의 이름은 MainFrm.pas로 한다.

uses절

interface부의 uses절에는 interface부에 있는 코드에 필요한 유닛들만 담도록 한다. 델파이가 자동으로 삽입한 유닛들 중 필요 없는 유닛 이름들은 제거한다.

implementation부의 uses절에는 implementation부에 있는 코드에 필요한 유닛들만 담도록 한다. 델파이가 자동으로 삽입한 유닛들 중 필요 없는 유닛 이름들은 제거한다.

interface부

interface부에는 외부 유닛들이 사용할 형, 변수, 프로시저/함수 등의 선행(forward) 선언들만 담도록 한다. 그 외의 것들은 implementation부에서 선언한다.

implementation부

implementation부에는 해당 유닛 안에서만 사용할 형, 변수, 프로시저/함수 등의 선언들을 담는다.

initialization부

유닛의 initialization부에서 시간이 오래 걸리는 코드를 수행하면 안된다. initialization부의 수행 시간이 길면 애플리케이션이 뜨는 시간이 길어진다.

finalization부

finalization부에서는 initialization부에서 할당한 자원들을 확실하게 해제해야 한다.

폼 유닛

명명 규칙

폼 유닛에는 해당 폼 파일의 이름과 같은 이름을 붙인다. 예를 들어서, About 폼의 유닛 이름은 AboutFrm.pas가 된다. 또한, 메인 폼의 유닛 이름은 MainFrm.pas가 된다.

데이터 모듈 유닛

명명 규칙

데이터 모듈 유닛에는 해당 데이터 모듈 파일의 이름과 같은 이름을 붙인다. 예를 들어서, 고객(Customers) 데이터 모듈 유닛의 이름은 CustmoersDM.pas가 된다.

범용 유닛

명명 규칙

범용 유닛에는 그 유닛의 목적에 맞는 이름을 붙여야 한다. 예를 들어서, BugUtilities.pas는 유틸리티 루틴들이 들어 있는 유닛이다. 또, 전역 변수가 들어 있는 유닛에는 CustomerGlobals.pas 같은 이름을 붙이면 된다.

한 프로젝트가 사용하는 모든 패키지들에 포함된 유닛들 중 동일한 이름의 유닛들이 있어서는 안된다. 유닛의 이름을 유닛의 목적을 최대한 구체적으로 표현한 이름으로 하면 유닛 이름이 중복될 가능성이 적어진다.

컴포넌트 유닛

명명 규칙

컴포넌트 유닛들은 그것들이 컴포넌트를 정의하는 유닛임을 쉽게 알 수 있도록 개별적인 디렉토리에 저장해야 한다. 컴포넌트 유닛을 프로젝트 디렉토리에 저장해서는 안된다. 컴포넌트 유닛의 이름은 그 유닛의 목적에 맞도록 만들어야 한다. 컴포넌트 명명 규칙에 대한 좀더 자세한 사항은 “사용자 정의 컴포넌트” 부분을 보기 바란다.

파일 머리말

소스 파일이나 프로젝트 파일, 유닛 등 텍스트 형태의 파일에는 그 파일에 대한 추가적인 정보를 담은 파일 머리말을 넣는 것이 좋다. 파일 머리말에는 다음과 같은 형태의 정보가 들어가는 것이 좋다.

```
{
Copyright © 연도 by 저자 이름
}
```

(역주 : 인터넷이 대중화된 점을 고려한다면 연도와 저자 이름을 영문으로 표기하는 것이 좋다)

폼과 데이터 모듈

폼

폼형 명명 규칙

폼형에는 그 폼의 목적에 맞는 이름을 붙여야 한다. 다른 모든 형들과 같이 폼형의 이름도 type을 뜻하는 T로 시작해야 하며, 폼을 나타내는 Form을 제일 뒤에 붙인다. 예를 들어서, About폼의 형 이름은 다음과 같이 붙이면 된다.

```
TAboutForm = class(TForm)
```

다음은 메인 폼의 예이다.

```
TMainForm = class(TForm)
```

다음은 고객 입력(Customer entry) 폼의 예이다.

```
TCustomerEntryForm = class(TForm)
```

폼 인스턴스 명명 규칙

폼 인스턴스에는 해당 폼형의 이름에서 T를 뺀 이름을 붙인다. 다음은 위에 나온 폼 형들에 해당하는 인스턴스 이름들이다.

형 이름	인스턴스 이름
TAboutForm	AboutForm
TMainForm	MainForm
TCustomerEntryForm	CustomerEntryForm

자동 생성 폼

가능한 한 메인 폼만 자동으로 생성되도록 한다. 다른 폼들은 Project Options 대화상자의 auto-create forms 목록에서 제거하고, 코드 안에서 직접 폼들을 생성한다.

모달 폼 인스턴스 생성 함수들

모든 폼 유닛들에는 폼을 모달로 생성, 설정, 표시, 파괴하는 함수들이 있다. 그런 함수들은 폼의 모달 결과를 돌려준다. 그런 함수들에 넘겨주는 인자들은 이 문서의 "인자 전달" 규칙을 따른다. 인자 전달 규칙의 목적은 코드의 재사용과 관리를 쉽게 하기 위한 것이다.

폼 변수들은 유닛의 전역 변수로 선언하기보다는 폼 조작 함수들 안에서 지역적으로 선언해야 한다.

다음은 UserData라는 폼을 사용하는 예이다.

```
unit UserDataFrm;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
  TUserDataForm = class(TForm)
    edtUserName: TEdit;
    edtUserID: TEdit;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
function GetUserData(var aUserName: String; var aUserID: Integer): Word;
implementation
{$R *.DFM}
function GetUserData(var aUserName: String; var aUserID: Integer): Word;
var
  UserDataForm: TUserDataForm;
begin
  UserDataForm := TUserDataForm.Create(Application);
  try
    UserDataForm.Caption := 'Getting User Data';
    Result := UserDataForm.ShowModal;
    if ( Result = mrOK ) then
```

```
    begin
      aUserName := UserDataForm.edtUserName.Text;
      aUserID := StrToInt(UserDataForm.edtUserID.Text);
    end;
  finally
   UserDataForm.Free;
  end;
end;
end.
```

데이터 모듈

데이터 모듈 명명 규칙

데이터 모듈형에는 그 데이터 모듈의 목적에 맞는 이름을 붙여야 한다. 다른 모든 형들과 같이 데이터 모듈형의 이름도 type을 뜻하는 T로 시작해야 하며, 데이터 모듈임을 나타내는 DataModule을 제일 뒤에 붙인다. 예를 들어서, 고객(Customer) 데이터 모듈의 형 이름은 다음과 같이 붙이면 된다.

```
TCustomerDataModule = class(TDataModule)
```

다음은 주문(Orders) 데이터 모듈의 예이다.

```
TOrdersDataModule = class(TDataModule)
```

데이터 모듈 인스턴스 명명 규칙

데이터 모듈 인스턴스에는 해당 데이터 모듈형의 이름에서 T를 뺀 이름을 붙인다. 다음은 위에 나온 데이터 모듈형들에 해당하는 인스턴스 이름들이다.

형 이름	인스턴스 이름
TCustomerDataModule TOrdersDataModule	CustomerDataModule OrdersDataModule

패키지

런타임 패키지와 설계 패키지의 차이

런타임 패키지에는 그 패키지의 컴포넌트들에 필요한 코드/컴포넌트들만 들어 있다. 속성/컴포넌트 편집기라든지, 설계 전용 코드가 담긴 유닛들은 설계 패키지에 담아야 한다. 등록 유닛도 설계 패키지에 포함된다.

패키지 파일 명명 규칙

패키지 파일의 이름은 다음과 같은 형태를 따른다.

- "iiilibvv.pkg" - 설계 패키지
- "iiistdvv.pkg" - 런타임 패키지

여기서 iii는 패키지를 식별하기 위한 3자짜리 접두어로, 일반적으로 개발사나 개발자 이름의 약자를 사용한다.

vv는 그 패키지가 사용될 델파이 버전을 뜻한다.

iii와 vv 사이에는 그 패키지가 런타임 패키지인지, 설계 패키지인지 구별할 수 있도록 lib나 std 중 하나가 들어간다.

런타임 패키지와 설계 패키지가 모두 존재하는 경우, 두 파일의 이름은 "lib"와 "std"만 빼고는 동일하게 한다. 다음 예는 델파이 5 개발자 가이드의 패키지 파일 이름들이다.

- DdgLib50.pkg - 설계 패키지
- DdgStd50.pkg - 런타임 패키지

컴포넌트

사용자 정의 컴포넌트

컴포넌트형 명명 규칙

컴포넌트의 이름은 "클래스" 부분에 나온 명명 규칙을 따르나, 개발사나 개발자를 식별하기 위해 T 다음에 세 글자의 접두어를 붙인다는 점이 다르다. 예를 들어서, 델파이 5 개발자 가이드가 만든 시계 컴포넌트의 이름은 다음과 같을 것이다.

```
TddgClock = class(TComponent)
```

세 글자의 접두어는 소문자로 쓴다.

컴포넌트 유닛

컴포넌트 유닛들은 하나의 주요(major) 컴포넌트만 담아야 한다. 주요 컴포넌트란, 컴포넌트 팔레트에 나타나는 컴포넌트를 말한다. 그 외의 부차적인 컴포넌트/객체들도 주요 컴포넌트와 같은 유닛에 있을 수 있다.

등록 유닛의 사용

컴포넌트의 등록 프로시저는 그 컴포넌트의 유닛에서 제거한 후 별도의 유닛에 넣어야 한다. 컴포넌트나 속성 편집기, 컴포넌트 편집기, 전문가(expert) 등의 등록은 별도의 유닛에서 해야 한다.

등록 유닛의 이름은 다음과 같은 형태가 좋다.

```
XxxReg.pas
```

"Xxx"는 개발자나 개발사를 뜻하는 3글자의 접두어이다. 예를 들어서, 델파이 5 개발자 가이드의 컴포넌트를 등록하는 등록 유닛의 이름은 DdgReg.pas가 된다.

컴포넌트 인스턴스의 명명 규칙

모든 컴포넌트들은 자신의 목적에 맞는 이름을 가져야 한다. 델파이가 지정한 기본 이름(예를 들면, Edit1)을 그대로 사용해서는 안된다. 컴포넌트 인스턴스의 이름에는 자신의 형을 뜻하는 소문자 접두어를 붙인다. 컴포넌트형을 접미어가 아닌 접두어로 표시하는 것은 객체 조사기(Object Inspector)나 코드 탐색기(Code Explorer)가 항목들을 알파벳 순서로 나열하기 때문이다. 접두어를 사용하면 같은 형의 컴포넌트 인스턴스들이 한 곳에 모이기 때문에 찾기가 쉽다.

컴포넌트 접두어

다음은 델파이 5의 표준 컴포넌트의 인스턴스 이름에 사용하는 접두어들이다. 써드 파티 컴포넌트들에 대해서도 비슷한 방식으로 접두어를 만들 수 있을 것이다.

Standard 탭

접두어	컴포넌트
fms	Frames
mm	TMainMenu
pm	TPopupMenu
mmi	TMainMenuItem
pmi	TPopupMenuItem
lbl	TLabel
edt	TEdit
mem	TMemo
btn	TButton
cb	TCheckBox
rb	TRadioButton
lb	TListBox
cb	TComboBox
scb	TScrollBar
gb	TGroupBox
rg	TRadioGroup
pnl	TPanel
al	TActionList

Additional 탭

접두어	컴포넌트
bbtn	TBitBtn
sb	TSpeedButton
me	TMaskEdit
sg	TStringGrid
dg	TDrawGrid
img	TImage
shp	TShape
bvl	TBevel
sbx	TScrollBox
clb	TCheckListbox
spl	TSplitter
stx	TStaticText
cb	TControlBar
aet	TApplicationEvent
cht	TChart

Win32 탭

접두어	컴포넌트
tbc	TTabControl
pgc	TPageControl
il	TImageList
re	TRichEdit
tbr	TTrackBar
prb	TProgressBar
ud	TUpDown
hk	THotKey
ani	TAnimate
dtp	TDateTimePicker
tv	TTreeView
lv	TListView
hdr	THeaderControl
stb	TStatusBar
tlb	TToolBar
clb	TCoolBar
psr	TPageScroller

System 탭

접두어	컴포넌트
tm	TTimer
pb	TPaintBox
mp	TMediaPlayer
olec	TOleContainer
ddcc	TDDEClientConv
ddci	TDDEClientItem
ddsc	TDDEServerConv
ddsi	TDDEServerItem

Internet Express 탭

접두어	컴포넌트
xlb	TXMLBroker
mppd	TMidasPageProducer

Fastnet 탭

접두어	컴포넌트
nmdt	TNMDayTime
nec	TNMEcho
nf	TNMFinger
nftp	TNMFtp
nhttp	TNMHttp
nMsg	TNMMsg
nmsg	TNMMSGServ
nntp	TNMNNTP
npop	TNMPop3
nuup	TNMUUProcessor
smtp	TNMSMTP
nst	TNMStrm
nsts	TNMStrmServ
ntm	TNMTime
nudp	TNMUdp
psk	TPowerSock
ngs	TNMGeneralServer
html	THtml
url	TNMUrl
sml	TSimpleMail

Internet 탭

접두어	컴포넌트
csk	TClientSocket
ssk	TServerSocket
wbd	TWebDispatcher
pp	TPageProducer
tp	TQueryTableProducer
dstp	TDataSetTableProducer
dspp	TDataSetPageProducer
wbs	TWebBrowser

Data Access 탭

접두어	컴포넌트
ds	TDataSource
tbl	TTable
qry	TQuery
sp	TStoredProc
db	TDataBase
ssn	TSession
bm	TBatchMove
usql	TUpdateSQL
ntdt	TNestedTable

Data Controls 탭

접두어	컴포넌트
dbg	TDBGrid
dbn	TDBNavigator
dbt	TDBText
dbe	TDBEdit
dbm	TDBMemo
dbi	TDBImage
dblb	TDBListBox
dbcb	TDBComboBox
dbch	TDBCheckBox

다음 페이지에 계속

이전 페이지에 이어서

접두어	컴포넌트
dbrg	TDBRadioGroup
dbll	TDBLookupListBox
dblc	TDBLookupComboBox
dbre	TDBRichEdit
dbcg	TDBCtrlGrid
dbch	TDBChart

Decision Cube 탭

접두어	컴포넌트
dcb	TDecisionCube
dcq	TDecisionQuery
dcs	TDecisionSource
dcp	TDecisionPivot
dcg	TDecisionGrid
dcgr	TDecisionGraph

QReport 탭

접두어	컴포넌트
qr	TQuickReport
qrsd	TQRSubDetail
qrb	TQRBand
qrcb	TQRChildBand
qrg	TQRGroup
qrl	TQRLabel
qrt	TQRText
qre	TQRExpr
qrs	TQRSysData
qrm	TQRMemo
qrrt	TQRRichText
qrdr	TQRDBRichText
qrsh	TQRShape
qri	TQRImage
qrdi	TQRDBMImage

다음 페이지에 계속

이전 페이지에 이어서

접두어	컴포넌트
qrcr	TQRCompositeReport
qrp	TQRPreview
qrtf	TQRTextFilter
qrcf	TQRCSVFilter
qrhf	TQRHTMLFilter
qrch	TQRChart

Dialogs 탭

대화상자 컴포넌트들은 실제 폼을 컴포넌트 형태로 만든 것이다. 따라서, 대화상자 컴포넌트들은 폼의 명명 규칙과 비슷한 규칙을 따른다. 컴포넌트형의 이름은 이미 정의되어 있으며, 인스턴스 이름은 그냥 ';'을 뺀 것, 즉 델파이가 붙인 기본 이름 끝의 숫자를 뺀 것을 사용하면 된다. 다음의 예를 참고하기 바란다.

형	인스턴스 이름
TOpenDialog	OpenDialog
TSaveDialog	SaveDialog
TOpenPictureDialog	OpenPictureDialog
TSavePictureDialog	SavePictureDialog
TFontDialog	FontDialog
TColorDialog	ColorDialog
TPrintDialog	PrintDialog
TPrintSetupDialog	PrinterSetupDialog
TFindDialog	FindDialog
TReplaceDialog	ReplaceDialog

Win3.1 탭

접두어	컴포넌트
dbll	TDBLookupList
dblc	TDBLookupCombo
ts	TTabSet
ol	TOutline
tnb	TTabbedNoteBook
nb	TNoteBook
hdr	THeader
flb	TFileListBox
dlb	TDirectoryListBox
dcb	TDriveComboBox
fcb	TFilterComboBox

Samples 탭

접두어	컴포넌트
gg	TGauge
cg	TColorGrid
spb	TSpinButton
spe	TSpinEdit
dol	TDirectoryOutline
cal	TCalendar
ibea	TIBEventAlerter

ActiveX 탭

접두어	컴포넌트
cfx	TChartFX
vsp	TVSSpell
f1b	TF1Book
vtc	TVTChart
grp	TGraph

Midas 탭

접두어	컴포넌트
CDs	TClientDataSet
dcom	TDCOMConnection
sck	TSocketConnection
dspv	TDataSetProvider
sobk	TSimpleObjectBroker
wctn	TWebConnection
cctn	TcorbaConnection

ADO 탭

접두어	컴포넌트
adcn	TADOConnection
adcm	TADOCommand
adds	TADODataSet
adtb	TADOTable
adqr	TADOQuery
adst	TADOStoredproc
rdcn	TRDSConnection

Interbase 탭

접두어	컴포넌트
ibds	TIBDataSource
ibtb	TIBTable
ibqr	TIBQuery
ibsp	TIBStoredProc
ibdb	TIBDatabase
ibts	TIBTransaction
ibus	TIBUpdateSQL
ibdst	TIBDataSet
ibsl	TIBSQL
ibsbi	TIBDatabaseInfo
ibsmt	TIBSqlMonitor
ibets	TIBEvents

쉬어가는 페이지