`inline` isim alanları `(inline namespaces)` `C++11` standartları ile dile eklenmiş bir özellik. Bir isim alanı `inline` anahtar sözcüğü ile bildirildiğinde bu isim alanı içindeki isimler onu içine alan isim alanı içinde doğrudan görülür hale geliyor:

```
inline namespace Neco {
 int x = 10;
}
```

gibi bir isim alanı kullanımının

```
namespace Neco {
 int x = 10;
 //
}


using namespace Neco;
```

gibi bir koda karşılık geldiğini düşünebiliriz. Şimdi de aşağıdaki koda bakalım:

```
namespace A {
 inline namespace B {
  int x = 10;
 }
}

int main()
{
 A::B::x = 20;
 A::x = 20;
}
```

`x` isimli değişkenin `A` isim alanı içinde yer alan `B` isim alanı içinde tanımlandığını görüyorsunuz. `B` isim alanı `inline` olarak tanımlandığından `main` işlevi içinde `x` ismini

```
A::B::x
```

biçiminde kullanabileceğimiz gibi doğrudan

```
A::x
```

biçiminde de kullanabiliyoruz. Birden fazla isim alanını da inline olarak tanımlamak mümkün:

```
namespace A {
 inline namespace B {
  inline namespace C {
   int x = 10;
  }
 }

 inline namespace D {
  int y = 20;
 }
}

int main()
{
 A::x = 1;
 A::B::x = 2;
 A::B::C::x = 3;

 A::y = 4;
 A::D::y = 5;

}
```

Yukarıdaki kodda `B`, `C` ve `D` isim alanları inline olarak tanımlandığından `main` işlevi içinde `x` ve `y` isimlerinin doğrudan `A` ismiyle nitelenerek kullanılması mümkün oluyor.

İyi de, ne işe yarıyor inline isim alanları? `inline` isim alanlarının sağladığı en önemli avantaj biri sürüm `(version)` kontrolü. Aşağıdaki koda bakalım:

```
namespace Networking {
 class TCPSocket {
  //
 };
 class UDPSocket {
  //
 };
}
```

`Networking` isim alanı içinde `TcpSocket` ve `UDPSocket` sınıfları tanımlanmış. Bu isim alanı içinde tanımlanan bu sınıflar birçok kaynak dosya tarafından kullanılıyor olsun. Şimdi `TCPSocket` sınıfında yaptığımız bazı geliştirmeler sonucunda yeni bir sınıf oluşturduğumuzu düşünelim. Yani `TCPSocket` sınıfının ikinci bir sürümünü oluşturduk. Müşteri kodların eski `TCPSocket` sınıfı yerine yeni `TCPSocket` sınıfını kullanmalarını istiyoruz. Bunu sağlamaya yönelik birçok seçenek olabilir. Ancak en etkin çözümlerden biri bu sınıfları ayrı birer isim alanı içine koymak:

```
namespace Networking {
 namespace Version1 {
  class TCPSocket {
   //
  };
 }

 namespace Version2 {
  class TCPSocket {
   //
  };
 }

 class UDPSocket {
  //
 };
}
```

Ama müşteri kodlar `TCPSocket` sınıfını

```
Networking::TCPSocket
```

biçiminde niteleyerek kullanıyorlardı değil mi? Müşteri kodları hiç değiştirmeden sürüm geçişini sağlayabilir miyiz? Evet, tahmin edebileceğiniz gibi `inline` isim alanı burada devreye giriyor. Yeni sürümün yer aldığı `Version2` isim alanını inline yapıyoruz:

```
namespace Networking {
 namespace Version1 {
  class TCPSocket {
   //
  };
 }

 inline namespace Version2 {
  class TCPSocket {
   //
  };
 }

 class UDPSocket {
  //
 };
}
```

Artık daha önce eksi sürümü kullanıyor olan müşteri kodlar yeniden derlendiklerinde yeni sürümü kullanıyor olacaklar. `inline` isim alanları kütüphanenin gerçekleştirimini yapan programcıya varsayılan `(default)` bir isim alanı belirleme olanağı sağlıyor. Yukarıdaki örnekte, tüm kullanıcı kodların `TCPSocket` sınıfının son sürümünü kullanmaları için `Version2` isim alanını inline yaptık. Belirli bir nedenden dolayı eski `TCPSocket` sınıfına geri dönmemiz gerekitse bu kez `Version1` isim alanını `inline` yapabiliriz. İstediğimiz sayıda isim alanını inline yapabiliriz. Örneğin kütüphanemizin `3.` sürümünde `UDPSocket` sınıfının da yenilendiğini düşünelim:

```
namespace Networking {
 namespace Verion1 {
  class TCPSocket;
  class UDPSocket;
 }

 inline namespace Version2 {
  class TCPSocket;
 }

 inline namespace Version3 {
  class UDPSocket;
 }
}
```

`Version3` isim alanı inline olarak tanımlandığından kullanıcı kodlar `UDPSocket` sınıfının son sürümünü (bu isim alanı içindeki sürümünü) kullanıyor olacaklar. Dilediğimiz zaman bu içsel isim alanını `inline` olmaktan çıkartıp `Version1` isim alanını inline yaparak o sürümdeki sınıfı kullanmaya geri dönebiliriz.

inline anahtar sözcüğünün kullanımını önişlemci koşullu derleme `(conditional compiling)` komutlarına da bağlayabiliriz:

```
namespace Networking {
 namespace Version1 {
  class TCPSocket;
  class UDPSocket;
 }

 inline namespace Version2 {
  class TCPSocket;
 }

#ifndef USE_RAW_SOCKETS
 inline
#endif
 namespace Version3 {
  class UDPSocket;
 }

#ifdef USE_RAW_SOCKETS
 inline
#endif
 namespace RawUDPSockets {
  class UDPSocket;
}
```

Yukarıdaki kodda `USE_RAW_SOCKETS` makrosu tanımlanmamış ise `Version3` isim alanı `inline` yapılmış olacak. Böylece bu isim alanı içindeki `UDPSocket` sınıfı doğrudan `Networking` isim alanında görülüyor olacak. Eğer `USE_RAW_SOCKETS` makrosu tanımlanmış ise bu kez `RawUDPSockets` isim alanı inline yapılmış olacak. Bu durumda da bu isim alanı içindeki `UDPSocket` sınıfı doğrudan `Networking` isim alanında görülüyor olacak. Bir başka deyişle `UDPSocket` sınıfının hangi sürümünün kullanılacağı `USE_RAW_SOCKETS` makrosunun tanımlanmış olup olmamasına bağlı.