Часто задают вопрос, имеются ли отдельные реализации или используется условная компиляция. Используются оба подхода, но с явным уклоном в сторону условной компиляции.

На приведенной ниже диаграмме видно, что CoreFX по больше части представляет собой независимый от платформы код, который используется на всех платформах.

.Net standard (it’s specification) – any .net implementations (Core, Framework, Xamarin) must implement that contract. Version .Net standard defines which platforms provides this bunch of API.

- We can’t use for example in lib standart1.6 lib standart2.0

Среда выполнения CoreCLR (Microsoft.CoreCLR) и библиотеки CoreFX распространяются через [NuGet](https://nuget.org/). Поскольку версия .NET Core представляет собой компонентизированный набор библиотек

Среда выполнения и основные библиотеки, связанные с платформой, должны переноситься как единое целое. Не зависящие от платформы библиотеки должны работать "как есть" на любой платформе.

Чтобы улучшить компонентизацию в .NET Core был обновлен факторинг API. Иными словами, существующие библиотеки для .NET Framework придется перекомпилировать для выполнения в .NET Core.

Пакет SDK содержит следующие компоненты:

1. программы командной строки .NET Core, используемые для создания приложений;
2. платформа .NET Core (библиотеки и среда выполнения), обеспечивающая сборку и выполнение приложений;
3. драйвер dotnet для выполнения [команд интерфейса командной строки (CLI)](https://docs.microsoft.com/ru-ru/dotnet/core/tools/index), а также приложений.

Исполняемый файл для .Net Core представляет собой обычную сборку для CLR и файлик, описывающий необходимые зависимости для её запуска. На текущий момент зависимости хранятся в Nuget пакетах. Подобное разделение позволяет обновлять фреймворк "модульно", без разделения на версии... Да, предполагается, что программы .Net Core стартуют так же, как и программы на Java.

Платформа .NET Core разделяется на набор пакетов, которые предоставляют типы-примитивы, типы данных более высокого уровня, составные типы приложений и общие служебные программы. Каждый из этих пакетов представляет отдельную сборку с тем же именем.

Мелкомодульное определение пакетов имеет ряд преимуществ:

* Мелкомодульные пакеты могут выпускаться по собственному графику; необходимость в тестировании других пакетов сравнительно невелика.
* Мелкомодульные пакеты могут обеспечивать различную поддержку ОС и ЦП.
* Мелкомодульные пакеты могут иметь зависимости, относящиеся только к одной библиотеке.
* Приложения имеют меньший размер, так как пакеты, на которые нет ссылок, не включаются в распространяемый пакет приложения.

Метапакеты — это принятое в NuGet соглашение для описания набора пакетов, каким-либо образом связанных друг с другом. Каждый метапакет привязан к целевой платформе. Например, платформа netstandard1.6 ссылается на метапакет NetStandard.Library версии 1.6.0. Аналогичным образом, платформа netcoreapp2.1ссылается на метапакет Microsoft.NETCore.App версии 2.1.0.

Использование метапакетов дает следующие преимущества:

* позволяет пользователям легко ссылаться на большой набор мелкомодульных пакетов;
* позволяет определять наборы пакетов (включая конкретные версии), которые тестируются и применяются как единое целое.