* **Framework Design Guideline.**

#Section 1 Introduction

Well Designed Framework is:

* Simple
* Expensive to Design
* Full of Trade-Offs
* Borrow from the Past
* Designed to Involve
* Integrated
* Consistent

#End of Section 1

#Section 2 Framework Design Fundamentals

To be a framework: **Simple things should be simple and complex thing should be possible**.

What the .NET fwk does is to unify existing programming models. It gives us consistent API that is available everywhere regardless of what language or programming model you are targeting.

    Do **explicitly** design for a broad range of developers with different programming styles, requirements, skill levels, and using different programming languages.

    Do understand the broad range of developers using multi-language frameworks.（Understand the users' needs)

Frame design should be focused around a set of common scenarios to the point where the whole design process is scenario driven.

Frame Design Principles

* Frameworks must be designed starting from a set of usage scenarios and code samples implementing these scenarios.

             Do make sure that the API design specification is the central part of the design of any feature that includes a publicly accessible APIs.

             Do define top usage scenarios for each major feature area

             Do ensure that the scenarios correspond to an appropriate abstraction level. Not too small.

             Do design APIs by first writing code samples for the main scenarios and the defining the object model to support the samples.

             Do write main scenario code samples in at least two different language families

             Do not rely solely on standard design methodologies when designing the public layer of a framework. prototyping, usability studies and iterations is a much better approach.

                Do organize usability studies to test APIs in main scenarios.

* Framework must offer a low barrier to entry for non-expert users through ease of experimentation.

             Do ensure that each main feature area namespace contains only types that are used in the most common scenarios. Types used in advanced scenarios should be placed in subnamespaces.

             Do provide simple overloads of constructors and methods. A simple overload has a very small number of parameters and all parameters are primitives.

             Do not have members intended for advanced scenarios on types intended for mainline scenarios.

             Do not require users to explicitly instantiate more than one type in the most basic scenarios

             Do not require that users perform any extensive initialization before they can start programming basic scenarios.

             Do provide good defaults for all properties and parameters if possible

             Do communicate incorrect usage of APIs using exceptions.

* In simple scenarios, frameworks must be usable without the need for documentation.

             Do ensure that APIs are intuitive and can be successfully used in basic scenarios without referring to the reference documentation.

             Do provide great documentation with all APIs.

             Naming

                 Do make the discussion about the identifier naming choices a significant part of the specification reviews.

                 Do not be afraid to use verbose identifier names.

                 Do involve user education experts early in the design process

                 Consider reserving the best type names for the most commonly used types.

             Exceptions

                 Do use exception messages to communicate framework usage mistakes.

             Strong Typing

                 Do provide strongly typed APIs if at all possible

             Consistency

                 Do ensure consistency with the .NET framework and other frameworks your customers are likely to use.

             Limiting Abstractions

                 Avoid many abstractions in mainline scenarios APIs

* Layered design makes it possible to provide both the power and the ease of use in a single framework.

             Consider a layered framework with high level APIs optimized for productivity and low  level APIs opotimized for power and expressiveness

             Avoid mixing low-level and hig-level APIs in a single namespace if the low-level APIs are very complex(i.e., they contain many types).

             Do ensure that layers of a single feature area are well integrated.

#End of Section 2