## **Other Conversion Mechanisms: Specialized Data Transformations in .NET**

While ToString(), Parse(), and format providers handle a wide range of formatting and parsing needs, .NET provides additional tools for specific conversion scenarios. These mechanisms are found across various types and namespaces, each serving a unique purpose.

### **Convert Class: General-Purpose Base Type Conversions**

The static System.Convert class provides a set of methods for converting between what .NET defines as "base types": bool, char, string, DateTime, DateTimeOffset, and all C# numeric types.

While many of its methods are redundant due to implicit casts or throw exceptions for invalid conversions, some are quite useful:

**Rounding Real to Integral Conversions:**

When converting a floating-point number (double, float) to an integer type (int, long), standard explicit casts ((int)doubleValue) perform truncation (simply cutting off the decimal part). Convert's numeric conversion methods, however, always perform rounding.

| double d1 = 3.9; int i1 = Convert.ToInt32(d1); // i1 == 4 (rounds up)  double d2 = 3.2; int i2 = Convert.ToInt32(d2); // i2 == 3 (rounds down)  double d3 = 3.5; int i3 = Convert.ToInt32(d3); // i3 == 4 (banker's rounding: rounds to nearest even number) double d4 = 4.5; int i4 = Convert.ToInt32(d4); // i4 == 4 (banker's rounding: rounds to nearest even number) |
| --- |

Convert uses **banker's rounding** for midpoint values (e.g., 3.5, 4.5), where it rounds to the nearest even integer. If you need different rounding behavior, use Math.Round() first, which offers more control.

**Parsing Numbers in Different Bases (Binary, Octal, Hexadecimal):**

Convert provides overloads for its ToInt32(), ToUInt32(), etc., methods that allow parsing string representations of numbers in bases other than 10.

| int thirty = Convert.ToInt32("1E", 16); // Parses "1E" as a hexadecimal (base 16) number: thirty == 30 uint five = Convert.ToUInt32("101", 2); // Parses "101" as a binary (base 2) number: five == 5 |
| --- |

The second argument specifies the base (2, 8, 10, or 16).

**Dynamic Conversions (ChangeType):**

When the types involved in a conversion are not known until runtime, Convert.ChangeType() is valuable. It takes an object value and a Type object representing the target type.

| Type targetType = typeof(int); object source = "42"; object result = Convert.ChangeType(source, targetType);  Console.WriteLine(result); // 42 Console.WriteLine(result.GetType()); // System.Int32 |
| --- |

This is useful in scenarios like deserialization or when working with data where types are determined dynamically. It can convert between any of the "base" types and also between an enum and its underlying integral type. A limitation is that it does not allow specifying a format string or parsing flags.

**Base-64 Conversions:**

Base-64 encoding is a standard method for representing binary data (like images, audio) in an ASCII string format, making it safe to embed within text-based documents (e.g., XML, JSON, email).

* Convert.ToBase64String(): Converts a byte[] (binary data) to a Base-64 string.
* Convert.FromBase64String(): Converts a Base-64 string back into a byte[].

### **XmlConvert: XML-Specific Formatting and Parsing**

The System.Xml.XmlConvert class is specifically designed for converting data types to and from string representations that adhere to **XML standards**. This is crucial when your application interacts with XML files or web services that use XML.

**Key features of XmlConvert:**

* **XML Compliance:** Handles XML-specific nuances, such as bool values being "true" or "false" (lowercase), unlike bool.ToString() which produces "True" or "False".
* **Culture Independence:** Its methods are inherently culture-invariant, making them ideal for serialization where consistency is paramount.
* **Methods:** Provided as overloaded ToString() methods for formatting and ToBoolean(), ToDateTime(), etc., for parsing.

| using System.Xml;  string s = XmlConvert.ToString(true); // s = "true" bool isTrue = XmlConvert.ToBoolean(s); // isTrue = true |
| --- |

**DateTime Handling with XmlConvert:**

XmlConvert's ToString() methods for DateTime accept an XmlDateTimeSerializationMode argument, which controls how time zone information is handled during serialization:

* Unspecified: Strips any DateTimeKind info.
* Local: Converts to local time zone and appends offset (e.g., 2010-02-22T14:07:30.9375+09:00).
* Utc: Converts to UTC and appends 'Z' (e.g., 2010-02-22T05:08:30.9375Z).
* RoundtripKind: Preserves the original DateTimeKind so that when reparsed, the DateTime has the same Kind as it did initially. This is often the safest choice for faithful round-tripping of DateTime values.

### **Type Converters: Design-Time and XAML Conversions**

Type converters, found in System.ComponentModel and typically named with a Converter suffix (e.g., ColorConverter), are a specialized mechanism primarily used in:

* **Design-time environments:** (e.g., Visual Studio property windows) to convert string input from developers into complex types (like Color, Image, Point).
* **XAML parsers:** (e.g., in WPF or UWP) to parse attribute values in XAML markup into object instances.

**Characteristics of Type Converters:**

* **Context-Aware Parsing:** They are very flexible, often inferring the correct format without explicit hints. For instance, ColorConverter can parse "Beige", "#800080", or "Window" into a Color object.
* **TypeConverter Base Class:** All type converters inherit from TypeConverter.
* **Obtaining a Converter:** Use TypeDescriptor.GetConverter(typeof(TypeToConvert)).
* **Methods:** ConvertToString() and ConvertFromString() are commonly used.

| using System.ComponentModel; using System.Drawing; // For Color type  TypeConverter cc = TypeDescriptor.GetConverter(typeof(Color)); Color beige = (Color)cc.ConvertFromString("Beige"); Color purple = (Color)cc.ConvertFromString("#800080"); |
| --- |

Type converters are linked to their respective types via a TypeConverterAttribute, allowing development tools to automatically discover and use them. They can also provide additional design-time services like populating dropdown lists for properties.

### **BitConverter: Binary Conversions**

The System.BitConverter class provides methods to convert primitive types to and from an array of bytes (byte[]). This is essential when you need to serialize primitive data into raw binary format for storage or transmission, or deserialize it back.

**Key methods:**

* BitConverter.GetBytes(value): Converts a primitive value (e.g., int, double, float, long, short, bool, char) into a byte[].
* BitConverter.ToDouble(), BitConverter.ToInt32(), etc.: Converts a byte[] (or a portion of it) back into a primitive type.

| foreach (byte b in BitConverter.GetBytes(3.5)) {  Console.Write(b + " "); // Output depends on endianness of your system (e.g., 0 0 0 0 0 0 12 64) }  byte[] bytes = new byte[] { 0, 0, 0, 0, 0, 0, 12, 64 }; // Example bytes for 3.5 (little-endian) double restoredDouble = BitConverter.ToDouble(bytes, 0); // restoredDouble = 3.5 |
| --- |

**Limitations:**

* BitConverter does **not** directly support decimal or DateTime/DateTimeOffset.
  + For decimal, use decimal.GetBits() to get an int[] representation.
  + For DateTime, use DateTime.ToBinary() which returns a long, then you can use BitConverter.GetBytes() on that long. DateTime.FromBinary() performs the reverse.