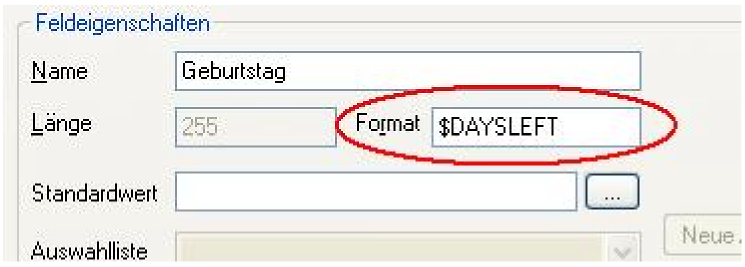
Seit AdressPLUS 11 Release 3 darf die Feld-Eigenschaft OutputFormat auch den Eintrag für *Format* aus den Feldeigenschaften im Dialog Dateieinstellungen (AdressPLUS -> Datei -> Einstellungen -> Adressfelder oder Zusatzdaten oder Kontakte) enthalten.



Name: Geburtstag
Länge: 255
Format: $DAYSLEFT
Standardwert:
Auswahlliste:

Es können die AdressPLUS eigenen Datums-Formate verwendet werden

- $DAYSLEFT
  wie viel Tage bis zum nächsten Auftreten des jährlichen Ereignisses, also wie viel Tage bis zum nächsten Weihnachten oder zum nächsten Geburtstag
- $AGE
  wie alt in ganzen Jahren
- $DAYOFWEEK
  Wochentag als Zahl von 1 bis 7: Sonntag 1, Montag 2, Dienstag 3, Mittwoch 4, Donnerstag 5, Freitag 6, Samstag 7

Neben den AdressPLUS eigenen Formatierungen können .Net Formatierungen verwendet werden. Siehe dazu die folgenden Auszüge aus der Dokumentation zu .Net.

Auszug aus der MSDN zu Format Strings

Standard Numeric Format Strings

Standard numeric format strings are used to format common numeric types. A standard format string takes the form *Axx* where *A* is a single alphabetic character called the format specifier, and *xx* is an optional integer called the precision specifier. The format specifier must be one of the built-in format characters. The precision specifier ranges from 0 to 99 and controls the number of significant digits or zeros to the right of a decimal. The format string cannot contain white spaces.

If the format string does not contain one of the standard format specifiers, then a FormatException is thrown. For example, the format string "z" is interpreted as a standard numeric format string because it contains one alphabetic character, but the alphabetic character is not one of the standard numeric format specifiers so a **FormatException** is thrown. Any numeric format string that does not fit the definition of a standard numeric format string is interpreted as a custom numeric format string. The format string "c!" is interpreted as a custom format string because it contains two alphabetic characters, even though the character "c" is a standard numeric format specifier.

The following table describes the standard numeric format strings. Note that the result string produced by these format specifiers is influenced by the settings in the Regional Options control panel. Computers using different settings will generate different result strings.

Format specifier	Name	Description
C or c	Currency	The number is converted to a string that represents a currency amount. The conversion is controlled by the currency format information of the NumberFormatInfo object used to format the number. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default currency precision given by the **NumberFormatInfo** is used.
D or d	Decimal	This format is supported for integral types only. The number is converted to a string of decimal digits (0-9), prefixed by a minus sign if the number is negative. The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier.
E or e	Scientific (exponential)	The number is converted to a string of the form "-d.ddd...E+ddd" or "-d.ddd...e+ddd", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. One digit always precedes the decimal point. The precision specifier indicates the desired number of digits after the decimal point. If the precision specifier is omitted, a default of six digits after the decimal point is used. The case of the format specifier indicates whether to prefix the exponent with an 'E' or an 'e'. The exponent always consists of a plus or minus sign and a minimum of three digits. The exponent is padded with zeros to meet this minimum, if required.
F or f	Fixed-point	The number is converted to a string of the form "-ddd.ddd..." where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the **NumberFormatInfo** is used.
G or g	General	The number is converted to the most compact of either fixed-point or scientific notation, depending on the type of the number and whether a precision specifier is present. If the precision specifier is omitted or zero, the type of the number determines the default precision, as indicated by the following list. • **Byte** or **SByte**: 3 • **Int16** or **UInt16**: 5 • **Int32** or **UInt32**: 10 • **Int64** or **UInt64**: 19

		• **Single**: 7 • **Double**: 15 • **Decimal**: 29 Fixed-point notation is used if the exponent that would result from expressing the number in scientific notation is greater than -5 and less than the precision specifier; otherwise, scientific notation is used. The result contains a decimal point if required and trailing zeroes are omitted. If the precision specifier is present and the number of significant digits in the result exceeds the specified precision, then the excess trailing digits are removed by rounding. If scientific notation is used, the exponent in the result is prefixed with 'E' if the format specifier is 'G', or 'e' if the format specifier is 'g'. The exception to the preceding rule is if the number is a **Decimal** and the precision specifier is omitted. In that case, fixed-point notation is always used and trailing zeroes are preserved.
N or n	Number	The number is converted to a string of the form "-d,ddd,ddd.ddd...", where each 'd' indicates a digit (0-9). The string starts with a minus sign if the number is negative. Thousand separators are inserted between each group of three digits to the left of the decimal point. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by the **NumberFormatInfo** is used.
P or p	Percent	The number is converted to a string that represents a percent as defined by the NumberFormatInfo.PercentNegativePattern property or the NumberFormatInfo.PercentPositivePattern property. If the number is negative, the string produced is defined by the **PercentNegativePattern** and starts with a minus sign. The converted number is multiplied by 100 in order to be presented as a percentage. The precision specifier indicates the desired number of decimal places. If the precision specifier is omitted, the default numeric precision given by **NumberFormatInfo** is used.
R or r	Round-trip	The round-trip specifier guarantees that a numeric value converted to a string will be parsed back into the same numeric value. When a numeric value is formatted using this specifier, it is first tested using the general format, with 15 spaces of precision for a **Double** and 7 spaces of precision for a **Single**. If the value is successfully parsed back to the same numeric value, it is formatted using the general format specifier. However, if the value is not successfully parsed back to the same numeric value, then the value is formatted using 17 digits of precision for a **Double** and 9 digits of precision for a **Single**. Although a precision specifier can be appended to the round-trip format specifier, it is ignored. Round trips are given precedence over precision when using this specifier. This format is supported by floating-point types only.
X or x	Hexadecimal	The number is converted to a string of hexadecimal digits. The case of the format specifier indicates whether to use uppercase or lowercase characters for the hexadecimal digits greater than 9. For example, use 'X' to produce "ABCDEF", and 'x' to produce "abcdef". The precision specifier indicates the minimum number of digits desired in the resulting string. If required, the number is padded with zeros to its left to produce the number of digits given by the precision specifier. This format is supported for integral types only.

The following example illustrates how to use the standard numeric format specifiers to format numeric base types.

```
[C#]
using System;
using System.Threading;
using System.Globalization;

class Class1
{
    static void Main()
    {
        Thread.CurrentThread.CurrentCulture = new CultureInfo("en-us");
            double MyDouble = 123456789;

        Console.WriteLine("The examples in en-US culture.\n");
        Console.WriteLine(MyDouble.ToString("C"));
        Console.WriteLine(MyDouble.ToString("E"));
        Console.WriteLine(MyDouble.ToString("P"));
        Console.WriteLine(MyDouble.ToString("N"));
        Console.WriteLine(MyDouble.ToString("F"));

        Thread.CurrentThread.CurrentCulture = new CultureInfo("de-DE");
        Console.WriteLine("The examples in de-DE culture.\n");
        Console.WriteLine(MyDouble.ToString("C"));
        Console.WriteLine(MyDouble.ToString("E"));
        Console.WriteLine(MyDouble.ToString("P"));
        Console.WriteLine(MyDouble.ToString("N"));
        Console.WriteLine(MyDouble.ToString("F"));
    }
}
```

The preceding code example displays the following to the console.

```
The examples in en-US culture:
$123,456,789.00
1.234568E+008
12,345,678,900.00%
123,456,789.00
123456789.00
The examples in de-DE culture:
123.456.789,00 DM
1,234568E+008
12,345,678,900.00%
123.456.789,00
123456789,00
```

Custom Numeric Format Strings

If the standard numeric format specifiers do not provide the type of formatting you require, you can use custom format strings to further enhance string output. A standard format string consists of a single alphabetic character optionally followed by a sequence of digits that form a value between 0 and 99; all other format strings are custom format strings.

The following table shows the characters you can use to create custom numeric format strings and their definitions. Note that the result strings produced by some of these characters are influenced by the settings in the Regional Options control panel of the NumberFormatInfo object associated with the current thread. Computers using different cultures will generate different result strings.

Format character	**Name**	**Description**
0	Zero placeholder	If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the result string. The position of the leftmost '0' before the decimal point and the rightmost '0' after the decimal point determines the range of digits that are always present in the result string. The "00" specifier causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "00" would result in the value 35.
#	Digit placeholder	If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the result string. Otherwise, nothing is stored in that position in the result string. Note that this specifier never displays the '0' character if it is not a significant digit, even if '0' is the only digit in the string. It will display the '0' character if it is a significant digit in the number being displayed. The "##" format string causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "##" would result in the value 35.
.	Decimal point	The first '.' character in the format string determines the location of the decimal separator in the formatted value; any additional '.' characters are ignored. The actual character used as the decimal separator is determined by the NumberDecimalSeparator property of the **NumberFormatInfo** that controls formatting.
,	Thousand separator and number scaling	The ',' character serves two purposes. First, if the format string contains a ',' character between two digit placeholders (0 or #) and to the left of the decimal point if one is present, then the output will have thousand separators inserted between each group of three digits to the left of the decimal separator. The actual character used as the decimal separator in the result string is determined by the NumberGroupSeparator property of the current **NumberFormatInfo** that controls formatting. Second, if the format string contains one or more ',' characters immediately to the left of the decimal point, then the number will be divided by the number of ',' characters multiplied by 1000 before it is formatted. For example, the format string "0,," will represent 100 million as simply 100. Use of the ',' character to indicate scaling does not include thousand separators in the formatted number. Thus, to scale a number by 1 million and insert thousand separators you would use the format string "#,##0,,".
%	Percentage placeholder	The presence of a '%' character in a format string causes a number to be multiplied by 100 before it is formatted. The appropriate symbol is inserted in the number itself at the location where the '%' appears in the format string. The percent character used is dependent on the current **NumberFormatInfo** class.
E0	Scientific notation	If any of the strings "E", "E+", "E-", "e", "e+", or "e-" are present in the format string and are followed immediately by at least one '0'

E+0 E-0 e0 e+0 e-0		character, then the number is formatted using scientific notation with an 'E' or 'e' inserted between the number and the exponent. The number of '0' characters following the scientific notation indicator determines the minimum number of digits to output for the exponent. The "E+" and "e+" formats indicate that a sign character (plus or minus) should always precede the exponent. The "E", "E-", "e", or "e-" formats indicate that a sign character should only precede negative exponents.
\	Escape character	In C# and the Managed Extensions for C++, the backslash character causes the next character in the format string to be interpreted as an escape sequence. It is used with traditional formatting sequences like '\n' (new line). In some languages, the escape character itself must be preceded by an escape character when used as a literal. Otherwise, the compiler interprets the character as an escape sequence. Use the string "\\" to display '\'. Note that this escape character is not supported in Visual Basic; however, **ControlChars** provides the same functionality.
'ABC' "ABC"	Literal string	Characters enclosed in single or double quotes are copied to the result string literally, and do not affect formatting.
;	Section separator	The ';' character is used to separate sections for positive, negative, and zero numbers in the format string.
Other	All other characters	All other characters are copied to the result string as literals in the position they appear.

Note that for fixed-point format strings (strings not containing an "E0", "E+0", "E-0", "e0", "e+0", or "e-0"), numbers are rounded to as many decimal places as there are digit placeholders to the right of the decimal point. If the format string does not contain a decimal point, the number is rounded to the nearest integer. If the number has more digits than there are digit placeholders to the left of the decimal point, the extra digits are copied to the result string immediately before the first digit placeholder.

Different formatting can be applied to a string based on whether the value is positive, negative, or zero. To produce this behavior, a custom format string can contain up to three sections separated by semicolons:

- **One section**: The format string applies to all values.
- **Two sections**: The first section applies to positive values and zeros, and the second section applies to negative values. If the number to be formatted is negative, but becomes zero after rounding according to the format in the second section, then the resulting zero is formatted according to the first section.
- **Three sections**: The first section applies to positive values, the second section applies to negative values, and the third section applies to zeros. The second section might be left empty (by having nothing between the semicolons), in which case the first section applies to all nonzero values. If the number to be formatted is nonzero, but becomes zero after rounding according to the format in the first or second section, then the resulting zero is formatted according to the third section.

This type of formatting ignores any preexisting formatting associated with a number when the final value is formatted. For example, negative values are always displayed without a minus sign when section separators are used. If you want the final formatted value to have a minus sign, you should explicitly include the minus sign as part of the custom format specifier. The following example illustrates how section separators can be used to produce formatted strings.

```
[C#]
double MyPos = 19.95, MyNeg = -19.95, MyZero = 0.0;
```

```
string MyString = MyPos.ToString("$#,##0.00;($#,##0.00);Zero");

// In the U.S. English culture, MyString has the value: $19.95.

MyString = MyNeg.ToString("$#,##0.00;($#,##0.00);Zero");

// In the U.S. English culture, MyString has the value: ($19.95).
// The minus sign is omitted by default.

MyString = MyZero.ToString("$#,##0.00;($#,##0.00);Zero");

// In the U.S. English culture, MyString has the value: Zero.
```

The following example demonstrates custom number formatting.

```
[C#]
Double myDouble = 1234567890;
String myString = myDouble.ToString( "(###) ### - ####" );
// The value of myString is "(123) 456 – 7890".

int  MyInt = 42;
MyString = MyInt.ToString( "My Number \n= #" );
// In the U.S. English culture, MyString has the value:
// "My Number
// = 42".
```

Standard DateTime Format Strings

A standard **DateTime** format string consists of a single format specifier character from the following table. If the format specifier is not found in the table below, a runtime exception is thrown. If the format string is longer than a single character (even if the extra characters are white spaces), the format string is interpreted as a custom format string.

Note that the result string produced by these format specifiers are influenced by the settings in the Regional Options control panel. Computers with different cultures or different date and time settings will generate different result strings.

The date and time separators displayed by format strings are defined by the DateSeparator and TimeSeparator characters associated with the DateTimeFormat property of the current culture. However, in cases where the InvariantCulture is referenced by the 'r', 's', and 'u' specifiers, the characters associated with the **DateSeparator** and **TimeSeparator** characters do not change based on the current culture.

The following table describes the standard format specifiers for formatting the **DateTime** object.

Format specifier	Name	Description
d	Short date pattern	Displays a pattern defined by the DateTimeFormatInfo.ShortDatePattern property associated with the current thread or by a specified format provider.
D	Long date pattern	Displays a pattern defined by the DateTimeFormatInfo.LongDatePattern property associated with the current thread or by a specified format provider.
t	Short time pattern	Displays a pattern defined by the DateTimeFormatInfo.ShortTimePattern property associated with the current thread or by a specified format provider.
T	Long time pattern	Displays a pattern defined by the DateTimeFormatInfo.LongTimePattern property associated with the current thread or by a specified format provider.
f	Full date/time pattern (short time)	Displays a combination of the long date and short time patterns, separated by a space.
F	Full date/time pattern (long time)	Displays a pattern defined by the DateTimeFormatInfo.FullDateTimePattern property associated with the current thread or by a specified format provider.
g	General date/time pattern (short time)	Displays a combination of the short date and short time patterns, separated by a space.
G	General date/time pattern (long time)	Displays a combination of the short date and long time patterns, separated by a space.
M or m	Month day pattern	Displays a pattern defined by the DateTimeFormatInfo.MonthDayPattern property associated with the current thread or by a specified format provider.
R or r	RFC1123 pattern	Displays a pattern defined by the DateTimeFormatInfo.RFC1123Pattern property associated with the current thread or by a specified format provider. This is a defined standard and the property is read-only; therefore, it is always the same regardless of the culture used, or the format provider supplied. The property references the **CultureInfo.InvariantCulture** property and follows the custom pattern "ddd, dd MMM yyyy HH:mm:ss G\MT". Note that the 'M' in "GMT" needs an escape character so it is not interpreted. Formatting does not modify the value of the **DateTime**; therefore, you must adjust the value to GMT before formatting.
s	Sortable date/time pattern; conforms to ISO 8601	Displays a pattern defined by the DateTimeFormatInfo.SortableDateTimePattern property associated with the current thread or by a specified format provider. The property

		references the **CultureInfo.InvariantCulture** property, and the format follows the custom pattern "yyyy-MM-ddTHH:mm:ss".
u	Universal sortable date/time pattern	Displays a pattern defined by the DateTimeFormatInfo.UniversalSortableDateTimePattern property associated with the current thread or by a specified format provider. Because it is a defined standard and the property is read-only, the pattern is always the same regardless of culture or format provider. Formatting follows the custom pattern "yyyy-MM-dd HH:mm:ssZ". No time zone conversion is done when the date and time is formatted; therefore, convert a local date and time to universal time before using this format specifier.
U	Universal sortable date/time pattern	Displays a pattern defined by the DateTimeFormatInfo.FullDateTimePattern property associated with the current thread or by a specified format provider. Note that the time displayed is for the universal, rather than local time.
Y or y	Year month pattern	Displays a pattern defined by the DateTimeFormatInfo.YearMonthPattern property associated with the current thread or by a specified format provider.
Any other single character	Unknown specifier	

The following example illustrates how to use the standard format strings with **DateTime** objects.

```
[C#]
DateTime dt = DateTime.Now;
DateTimeFormatInfo dfi = new DateTimeFormatInfo();
CultureInfo ci = new CultureInfo("de-DE");

// Make up a new custom DateTime pattern, for demonstration.
dfi.MonthDayPattern = "MM-MMMM, ddd-dddd";

// Use the DateTimeFormat from the culture associated
// with the current thread.
Console.WriteLine( dt.ToString("d") );
Console.WriteLine( dt.ToString("m") );

// Use the DateTimeFormat from the specific culture passed.
Console.WriteLine( dt.ToString("d", ci ) );

// Use the settings from the DateTimeFormatInfo object passed.
Console.WriteLine( dt.ToString("m", dfi ) );

// Reset the current thread to a different culture.
Thread.CurrentThread.CurrentCulture = new CultureInfo("fr-BE");
Console.WriteLine( dt.ToString("d") );
```

Custom DateTime Format Strings

You can exercise greater control over how a DateTime object is formatted by using custom **DateTime** format specifiers to create your own custom **DateTime** format string. Combine one or more custom format specifiers to construct a **DateTime** formatting pattern that yields the output you prefer. In fact, most of the standard **DateTime** format specifiers are aliases for formatting patterns specified in the currently applicable DateTimeFormatInfo Class.

The following table describes the custom format specifiers and the results they produce. The output of these format specifiers is influenced by the current culture and the settings in the Regional Options control panel.

Format specifier	**Description**
d	Displays the current day of the month, measured as a number between 1 and 31, inclusive. If the day is a single digit only (1-9), then it is displayed as a single digit. Note that if the 'd' format specifier is used alone, without other custom format strings, it is interpreted as the standard short date pattern format specifier. If the 'd' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
dd	Displays the current day of the month, measured as a number between 1 and 31, inclusive. If the day is a single digit only (1-9), it is formatted with a preceding 0 (01-09).
ddd	Displays the abbreviated name of the day for the specified **DateTime**. If a specific valid format provider (a non-null object that implements IFormatProvider with the expected property) is not supplied, then the AbbreviatedDayNames property of the DateTimeFormat and its current culture associated with the current thread is used. Otherwise, the **AbbreviatedDayNames** property from the specified format provider is used.
dddd (plus any number of additional "d" characters)	Displays the full name of the day for the specified **DateTime**. If a specific valid format provider (a non-null object that implements **IFormatProvider** with the expected property) is not supplied, then the DayNames property of the **DateTimeFormat** and its current culture associated with the current thread is used. Otherwise, the **DayNames** property from the specified format provider is used.
f	Displays seconds fractions represented in one digit. Note that if the 'f' format specifier is used alone, without other custom format strings, it is interpreted as the full (long date + short time) format specifier. If the 'f' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
ff	Displays seconds fractions represented in two digits.
fff	Displays seconds fractions represented in three digits.
ffff	Displays seconds fractions represented in four digits.
fffff	Displays seconds fractions represented in five digits.
ffffff	Displays seconds fractions represented in six digits.
fffffff	Displays seconds fractions represented in seven digits.
g or gg (plus any number of additional "g" characters)	Displays the era (A.D. for example) for the specified **DateTime**. If a specific valid format provider (a non-null object that implements **IFormatProvider** with the expected property) is not supplied, then the era is determined from the calendar associated with the **DateTimeFormat** and its current culture associated with the current thread. Note that if the 'g' format specifier is used alone, without other custom format strings, it is interpreted as the standard general format specifier. If the 'g' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
h	Displays the hour for the specified **DateTime** in the range 1-12. The hour

	represents whole hours passed since either midnight (displayed as 12) or noon (also displayed as 12). If this format is used alone, then the same hour before or after noon is indistinguishable. If the hour is a single digit (1-9), it is displayed as a single digit. No rounding occurs when displaying the hour. For example, a **DateTime** of 5:43 returns 5.
hh, hh (plus any number of additional "h" characters)	Displays the hour for the specified **DateTime** in the range 1-12. The hour represents whole hours passed since either midnight (displayed as 12) or noon (also displayed as 12). If this format is used alone, then the same hour before or after noon is indistinguishable. If the hour is a single digit (1-9), it is formatted with a preceding 0 (01-09).
H	Displays the hour for the specified **DateTime** in the range 0-23. The hour represents whole hours passed since midnight (displayed as 0). If the hour is a single digit (0-9), it is displayed as a single digit.
HH, HH (plus any number of additional "H" characters)	Displays the hour for the specified **DateTime** in the range 0-23. The hour represents whole hours passed since midnight (displayed as 0). If the hour is a single digit (0-9), it is formatted with a preceding 0 (01-09).
m	Displays the minute for the specified **DateTime** in the range 0-59. The minute represents whole minutes passed since the last hour. If the minute is a single digit (0-9), it is displayed as a single digit. Note that if the 'm' format specifier is used alone, without other custom format strings, it is interpreted as the standard month day pattern format specifier. If the 'm' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
mm, mm (plus any number of additional "m" characters)	Displays the minute for the specified **DateTime** in the range 0-59. The minute represents whole minutes passed since the last hour. If the minute is a single digit (0-9), it is formatted with a preceding 0 (01-09).
M	Displays the month, measured as a number between 1 and 12, inclusive. If the month is a single digit (1-9), it is displayed as a single digit. Note that if the 'M' format specifier is used alone, without other custom format strings, it is interpreted as the standard month day pattern format specifier. If the 'M' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
MM	Displays the month, measured as a number between 1 and 12, inclusive. If the month is a single digit (1-9), it is formatted with a preceding 0 (01-09).
MMM	Displays the abbreviated name of the month for the specified **DateTime**. If a specific valid format provider (a non-null object that implements **IFormatProvider** with the expected property) is not supplied, the AbbreviatedMonthNames property of the **DateTimeFormat** and its current culture associated with the current thread is used. Otherwise, the **AbbreviatedMonthNames** property from the specified format provider is used.
MMMM	Displays the full name of the month for the specified **DateTime**. If a specific valid format provider (a non-null object that implements **IFormatProvider** with the expected property) is not supplied, then the MonthNames property of the **DateTimeFormat** and its current culture associated with the current thread is used. Otherwise, the **MonthNames** property from the specified format provider is used.
s	Displays the seconds for the specified **DateTime** in the range 0-59. The second represents whole seconds passed since the last minute. If the second is a single digit (0-9), it is displayed as a single digit only. Note that if the 's' format specifier is used alone, without other custom format strings, it is interpreted as the standard sortable date/time pattern format specifier. If the 's' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
ss, ss (plus any number of additional "s" characters)	Displays the seconds for the specified **DateTime** in the range 0-59. The second represents whole seconds passed since the last minute. If the second is a single digit (0-9), it is formatted with a preceding 0 (01-09).

t	Displays the first character of the A.M./P.M. designator for the specified **DateTime**. If a specific valid format provider (a non-null object that implements **IFormatProvider** with the expected property) is not supplied, then the AMDesignator (or PMDesignator) property of the **DateTimeFormat** and its current culture associated with the current thread is used. Otherwise, the **AMDesignator** (or **PMDesignator**) property from the specified **IFormatProvider** is used. If the total number of whole hours passed for the specified **DateTime** is less than 12, then the **AMDesignator** is used. Otherwise, the **PMDesignator** is used. Note that if the 't' format specifier is used alone, without other custom format strings, it is interpreted as the standard long time pattern format specifier. If the 't' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
tt, tt (plus any number of additional "t" characters)	Displays the A.M./P.M. designator for the specified **DateTime**. If a specific valid format provider (a non-null object that implements **IFormatProvider** with the expected property) is not supplied, then the **AMDesignator** (or **PMDesignator**) property of the **DateTimeFormat** and its current culture associated with the current thread is used. Otherwise, the **AMDesignator** (or **PMDesignator**) property from the specified **IFormatProvider** is used. If the total number of whole hours passed for the specified **DateTime** is less than 12, then the **AMDesignator** is used. Otherwise, the **PMDesignator** is used.
y	Displays the year for the specified **DateTime** as a maximum two-digit number. The first two digits of the year are omitted. If the year is a single digit (1-9), it is displayed as a single digit. Note that if the 'y' format specifier is used alone, without other custom format strings, it is interpreted as the standard short date pattern format specifier. If the 'y' format specifier is passed with other custom format specifiers or the '%' character, it is interpreted as a custom format specifier.
yy	Displays the year for the specified **DateTime** as a maximum two-digit number. The first two digits of the year are omitted. If the year is a single digit (1-9), it is formatted with a preceding 0 (01-09).
yyyy	Displays the year for the specified **DateTime**, including the century. If the year is less than four digits in length, then preceding zeros are appended as necessary to make the displayed year four digits long.
z	Displays the time zone offset for the system's current time zone in whole hours only. The offset is always displayed with a leading sign (zero is displayed as "+0"), indicating hours ahead of Greenwich mean time (+) or hours behind Greenwich mean time (-). The range of values is –12 to +13. If the offset is a single digit (0-9), it is displayed as a single digit with the appropriate leading sign. The setting for the time zone is specified as +X or –X where X is the offset in hours from GMT. The displayed offset is affected by daylight savings time.
zz	Displays the time zone offset for the system's current time zone in whole hours only. The offset is always displayed with a leading or trailing sign (zero is displayed as "+00"), indicating hours ahead of Greenwich mean time (+) or hours behind Greenwich mean time (-). The range of values is –12 to +13. If the offset is a single digit (0-9), it is formatted with a preceding 0 (01-09) with the appropriate leading sign. The setting for the time zone is specified as +X or –X where X is the offset in hours from GMT. The displayed offset is affected by daylight savings time.
zzz, zzz (plus any number of additional "z" characters)	Displays the time zone offset for the system's current time zone in hours and minutes. The offset is always displayed with a leading or trailing sign (zero is displayed as "+00:00"), indicating hours ahead of Greenwich mean time (+) or hours behind Greenwich mean time (-). The range of values is –12:00 to +13:00. If the offset is a single digit (0-9), it is formatted with a preceding 0 (01-09) with the appropriate leading sign. The setting for the time zone is specified as +X or –X where X is the offset in hours from GMT. The displayed offset is affected by daylight savings time.

:	Time separator.
/	Date separator.
"	Quoted string. Displays the literal value of any string between two quotation marks preceded by the escape character (/).
'	Quoted string. Displays the literal value of any string between two " ' " characters.
%*c*	Where *c* is both a standard format specifier and a custom format specifier, displays the custom format pattern associated with the format specifier. Note that if a format specifier is used alone as a single character, it is interpreted as a standard format specifier. Only format specifiers consisting of two or more characters are interpreted as custom format specifiers. In order to display the custom format for a specifier defined as both a standard and a custom format specifier, precede the specifier with a % symbol.
\c	Where *c* is any character, the escape character displays the next character as a literal. The escape character cannot be used to create an escape sequence (like "\n" for new line) in this context.
Any other character	Other characters are written directly to the result string as literals.

When you pass a custom pattern to **DateTime.ToString**, the pattern must be at least two characters long. If you pass only "d", it is interpreted by the common language runtime as a standard format specifier because all single format specifiers are interpreted as standard. If you pass a single "h", an exception is thrown because there is not a standard "h" format specifier. To format using a single custom-defined format only, include a space before or after the specifier. For example, the format string "h " is interpreted as a custom-defined format string.

The following example illustrates how to create custom formatted strings from a **DateTime**. This example assumes that the current culture is U.S. English (en-US).

```
[C#]
DateTime MyDate = new DateTime(2000, 1, 1, 0, 0, 0);
String MyString = MyDate.ToString("dddd - d - MMMM");
// In the U.S. English culture, MyString has the value:
// "Saturday - 1 - January".
MyString = MyDate.ToString("yyyy gg");
// In the U.S. English culture, MyString has the value: "2000 A.D.".
```

Der Vollständigkeit halber sei noch erwähnt, dass es auch Formatierungen gibt, die von dem verwendeten Werkzeug **ActiveReports** selber umgesetzt werden. Diese werden auch im Experten in der Dialog-Box, die sich öffent, wenn man in der Eigenschaft *OutputFormat* auf die "..."-Schaltfläche klickt, angeboten. **Achtung**: Diese Formatierungen ergeben immer ein **amerikanisches** Format (bei Zahlen z.B. 1,234,567.00 statt 1.234.567,00). Am besten werden die .Net-Formatierungen verwendt.

Auszug aus der Online-Dokumentation von ActiveReports zu OutputFormat

OutputFormat Strings

ActiveReports allows you to set formatting strings for date, time, currency, and other numeric values using the OutputFormat property on the textbox control. The OutputFormat dialog also allows you to select international currency values and select from various built-in string expressions. In addition to the built-in string expressions, you may use any .NET standard formatting strings. Information about these strings (Numerics and Date/Time formats) may be found on MSDN.

Times:

- hh:mm tt = 09:00 AM
- HH:mm = 21:00 (twenty-four hour clock)
- HH = hours in 24 hour clock
- hh = hours in 12 hour clock
- mm = minutes
- ss = seconds
- tt = AM or PM

Dates:

- dddd, MMMM d, yyyy = Saturday, December 25, 1999
- dd/MM/yyyy = 25/12/1999
- d or dd = day in number format
- ddd = day in short string format (Ex. Sat for Saturday)
- dddd = day in string format (Ex. Saturday)
- MM = month in number format
- MMM = month in short string format (Ex. Dec for December)
- MMMM = month in string format (Ex. December)
- y or yy = year in two digit format (Ex. 99 for 1999)
- yyyy or yyyy = year in four digit format (Ex. 1999)

Currency and numbers:

- $00.00 = $25.50
- $#,##0.00 = $06.25
- 0 = digit or zero
- # = digit or nothing
- % = percent-multiplies the string expression by 100