Example 3 (Acrobat 7.0)

Use the XFA-Picture Clause to write the current date to the console.

```
// Execute in console
console.println(
   util.printd("EEE, 'the' D 'of' MMMM, YYYY", new Date(), true));
// The output on this day is
Tue, the 13 of July, 2004
```

Locale-Sensitive Picture Clauses. Normally processing of picture clauses occurs in the ambient locale. It is possible, however, to indicate that picture processing be done in a specific locale. This is of use when formatting or parsing data that is locale-specific and different from the ambient locale. The syntax for this extension to compound picture clauses is:

```
category-name(locale-name){picture-symbols}
```

The code executed in the console,

```
util.printd("date(fr){DD MMMM, YYYY}", new Date(), true)
```

yields the output on this day,

```
13 juillet, 2004
```

The XFA-Picture Clause gives extensive support for Chinese, Chinese (Taiwan), Japanese, and Korean (CCJK) times and dates. The example below, a custom format script of a text field, gives the current date formatted for a Japanese locale.

```
event.value = util.printd("date(ja){ggYY/M/D}", new Date(), true)
```

printf

3.01			

Formats one or more arguments as a string according to a format string. It is similar to the C function of the same name. This method converts and formats incoming arguments into a result string according to a format string (`cFormat`).

The format string consists of two types of objects:

- Ordinary characters, which are copied to the result string.
- Conversion specifications, each of which causes conversion and formatting of the next successive argument to `printf`.

Each conversion specification is constructed as follows:

```
%[,nDecSep][cFlags][nWidth][.nPrecision]cConvChar
```

The following table describes the components of a conversion specification.

`nDecSep`	A comma character (`,`) followed by a digit that indicates the decimal/separator format: **`0`** — Comma separated, period decimal point **`1`** — No separator, period decimal point **`2`** — Period separated, comma decimal point **`3`** — No separator, comma decimal point
`cFlags`	Only valid for numeric conversions and consists of a number of characters (in any order), which will modify the specification: **`+`** — Specifies that the number will always be formatted with a sign. **`space`** — If the first character is not a sign, a space will be prefixed. **`0`** — Specifies padding to the field with leading zeros. **`#`** — Specifies an alternate output form. For `f`, the output will always have a decimal point.
`nWidth`	A number specifying a minimum field width. The converted argument is formatted to be at least this many characters wide, including the sign and decimal point, and may be wider if necessary. If the converted argument has fewer characters than the field width, it is padded on the left to make up the field width. The padding character is normally a space, but is 0 if the zero padding flag is present (`cFlags` contains `0`).
`nPrecision`	A period character (`.`) followed by a number that specifies the number of digits after the decimal point for float conversions.
`cConvChar`	Indicates how the argument should be interpreted: **`d`** — Integer (truncating if necessary) **`f`** — Floating-point number **`s`** — String **`x`** — Integer (truncating if necessary) and formatted in unsigned hexadecimal notation

Parameters

`cFormat`	The format string to use.
`arguments`	The optional arguments(s) that contain the data to be inserted in place of the `%` tags specified in the first parameter, the format string. The number of optional arguments must be the same as the number of `%` tags.

Note: The `util.printf` function does not accept an object literal with properties that contain the arguments. Arguments are entered in the usual comma-delimited list.

Returns

A result string formatted as specified.