1 SEGGER_SYSVIEW_P
1.1 Overview and termir
1.2 Using SEGGER_SY
1.2.1 Supported forma
1.2.2 Omitted error-ch
1.3 Bugs in SEGGER_S
1.3.1 The User-Guide
1.3.2 "%%" causes bu
1.3.3 "%c" with 0x00
1.3.4 "%s" use incorre
2 SystemView Recorder: p
2.1 Timestamps in the th
2.2 The Terminal windo
2.3 Events List window
2.4 Timeline window
3 Upgrading SystemView
3.1 Upgrading the files i
3.2 Upgrading instances
4 SystemView troubleshoo
4.1 SystemView problen
4.2 SystemView problen
4.3 SystemView problen
4.4 SystemView bug: d
4.5 Console output for a

1.2.1 Supported format-placeholder types, e.g., "%d", but not %s

`S...PrintfHost()` only supports some format-placeholder types, e.g., `%d`, but not `%s`. The *User Guide* does not specify which ones are supported.

From testing, the supported types are listed below. There may be others.

- **c** : Character
- **d** : Signed decimal integer
- **p** : Pointer address
- **u** : Unsigned decimal integer
- **x** : Unsigned hexadecimal integer
- **%** : A % followed by another % character will write a single % to the output-string.

From testing, the types that are **not** supported are listed below. There may be others.

- **f** : Decimal floating point, lowercase
- **F**: Decimal floating point, uppercase (from static analysis, not tested)
- **s** : String of characters

1.2.2 Omitted error-checking

`S...PrintfHost()` doesn't provide syntax checking for its arguments. Also, the SystemView app doesn't appear to provide much syntax or error checking for the arguments it receives from `S...PrintfHost()`. A consequence is that when using `S...PrintfHost()` incorrectly, the `S...PrintfHost()` call will typically compile and run, but the SystemView app may display no data or incorrect data, or the app may crash. (In contrast, GCC's `printf()` will detect many errors at compile time.)

For example, "`%f`" and "`%s`" are not supported, but `S...PrintfHost()` does not indicate this when compiled and run. When "`%f`" or "`%s`" are specified, only part of the provided data is sent by `S...PrintfHost()` to the SystemView app. It's not clear what the app does to process "`%f`" and "`%s`". The app may just ignore the format-placeholder, or it may attempt to format the data provided, even though only part of it is provided. For example, with `%s`, the app sometimes displays a few random characters for the string, but it usually displays nothing.

When `S...PrintfHost()` is used incorrectly, the SystemView app may display incorrect data. These are some examples:

- If "`%f %d`" is specified, the "`%d`" will display the data in the floating-point argument's second 4-bytes (for an 8-byte floating-point argument). (No indication is given that "`%f`" is not supported.)
- If "`%d`" is specified, and the argument provided for it is an `unsigned int`, an incorrect value may be displayed. (With GCC's `printf()`, a warning message is issued for such data-type mismatches.)
- If an argument is not provided for a format-placeholder (e.g., "%c"), then bogus argument-data is used, from the call-stack. (With GCC's `printf()`, a warning message is issued for such argument mismatches.)

1.3 Bugs in SEGGER_SYSVIEW_PrintfHost()

This section describes three known bugs in `S...PrintfHost()`. The work-arounds are obvious, so they are not described.

For context, `S...PrintfHost()` is implemented in `SEGGER_SYSVIEW.c`. Most of the processing for `S...PrintfHost()` is done by the function `_VPrintHost()`.

1.3.1 The User-Guide omits needed info

- **Bug:** The *SystemView User Guide* omits info needed to use `SEGGER_SYSVIEW_PrintfHost()`.

The *User Guide* has a very short section on `S...PrintfHost()`. As described earlier, the *User Guide* does not specify what format-placeholders are supported. Also, it doesn't specify how errors in argument-use are detected and processed. These omissions imply that the function works the same as `printf()`, but this is misleading, as there are significant differences with `printf()`.

The *User Guide* describes `S...PrintfHost()`'s arguments in one sentence, but it is unclear. It states, "*All format arguments are treated as 32-bit scalar values.*" ("*Format arguments*" is another term for *format-specifier* arguments.) However, "*treated as*" is vague. It does not adequately describe what argument-types can be used, and how the arguments are processed and used.

For example, `float`-type variables are 4 bytes and scalar, but they are not supported. `float`-type arguments are expanded to 8 bytes when pushed on the call-stack. Internally, `S...PrintfHost()` is only able to correctly-process arguments that are 4-bytes on the call-stack. In contrast, `char`-type variables are 1 byte and scalar, and they are supported. They are expanded to 4 bytes when pushed on the call-stack.

1.3.2 "%%" causes buffer-overflow