CSTA Connector for BroadSoft BroadWorks Deployment Guide

Hiding Data in Logs

12/17/2025

Hiding Data in Logs

The following topics describe the data hiding functionality for the CSTA Connector:

Hiding Data in Logs Overview

CSTA Connector provides data filtering capabilities based on the Perl-Compatible Regular Expressions (PCRE) library. Sensitive data is defined by a set of regular expressions provisioned in the application configuration though Genesys Management Framework. All data to be hidden is overwritten with an asterisk (*) symbol.

CSTA Connector supports two major modes of hiding sensitive data:

1. **Simple:** Data identified by a matching prefix is hidden up to the end of line.
2. **Complex:** If the regular expression contains one or more *capture* subexpressions, the data matching the subexpressions is hidden while the rest of the text is not. This allows partial hiding of data, such as displaying the last four digits of a credit card number, and preservation of syntactic elements such as parentheses and quotes.

The technical definition of the behaviour of complex patterns is as follows:

- If the regular expression positively matches, and there are no captured sub-strings, the remainder of the text line after the match is hidden.
- If the regular expression matches and there are one or more captured sub-strings, all captured sub-strings are hidden.

Multiple Regular Expressions

Multiple regular expressions can be provided that are arranged in ASCII order by their corresponding option name and applied sequentially. Each subsequent expression is applied to a string that could have been already modified by the preceding expression. It is thus essential that the preceding regular expressions do not hide the keyword part of the succeeding expressions. It is recommended that the expressions do not overlap in their matches and that multiple sensitive data chunks are handled in a single expression.

Examples:

- The `non-capture` sub-expressions denoted by `(?:)` are not hidden and can be used for grouping the keyword expressions.
  For example, the expression: `User (?:PIN|account)` has no `capture` strings and thus the remainder of the string after the match is hidden.
  If the expression `User (PIN|account)` is used, the words `PIN` or `account` are hidden, while the following data is not, because `(PIN|account)` forms a `capture` sub-expression.

- The `non-capture` and `capture` sub-expressions can be combined in one expression with the expected results. For example, the expression:

`[Aa](?:uthentication|ccess)(?: code)?: *"([^"]).{3}"` has two non-capture and one capture strings; the latter, which is the leading characters of the code enclosed between double quotes, is hidden; the last three characters of the code and the closing double quotes are displayed. (The `non-capture` part matches the words `authentication` or `authorization` with an optionally capitalized initial `A` and optionally followed by the word `code`).

Note: The use of optional capture strings can lead to obscure results:

- For example, the expression: `User( name)?`, which matches `User` or `User name`, results in hiding the trailing string in the former case, but results in hiding the word `name` in the latter case, because `( name)` is a capture sub-expression. In this instance, a non-capture string should be used, such as `User(?: name)?`.

A valid example of an optional capture string is a code that can come in one or two parts:

- For example, the expression: `Code ([A-Z]{3-5} )? ([0-9]{10})`, which matches the word `Code` followed by a code that is composed of an optional alphanumeric prefix of three to five symbols, and a whitespace, followed by a mandatory 10-digit code. Both the prefix and the digital code are masked.