LoadRunner Script Documentation: Escaping CRLF and Special Characters for JSON Requests

---

Script Overview

This LoadRunner script demonstrates how to safely handle correlation values or dynamic strings that contain special characters when sending them in a JSON POST request. The script ensures that characters like CRLF (`\r\n`), quotes (`"`), and backslashes (`\`) are correctly escaped so the target system (e.g., Appian or any JSON-based API) receives the intended literal values.

Components of the Script

1. globals.h

- Declares global buffers and reusable functions.
- Key elements:

```
char g_escapedBuffer[32768];
```

- A large global buffer to hold escaped strings. - Global scope is used because large local buffers in Action() can cause "too many local variables" compile errors.

```
void lr_escape_for_json(char *input_param, char *output_param)
```

- Utility function to escape special characters for JSON requests. - Input: LR parameter name containing raw data (e.g., `{CorrValue}`) - Output: LR parameter name for escaped value (e.g., `{CorrValue_Escaped}`)

Escapes the following characters:

Character	Escaped Version	Reason
`\`	`\\`	JSON requires backslashes to be escaped
`"`	`\"`	Double quotes must be escaped in JSON strings
`\r`	`\\r`	Preserve carriage return in multi-line strings
`\n`	`\\n`	Preserve line feed in multi-line strings

- Uses `lr_eval_string()` to get the runtime value of an LR parameter.
- Uses `lr_save_string()` to store the escaped result in a new LR parameter.

---

2. Action() function

Local variable declarations

```
char *paramName;
char *escapedParamName;
```

- Only small pointers are declared locally to avoid stack overflow / “too many local variables” errors.

Simulate correlation value

```
lr_save_string("Hello \"World\"\r\nThis is line2\\end", "CorrValue");
```

- Creates a test parameter simulating a correlated value that might contain CRLF, quotes, or backslashes.

Escape special characters

```
lr_escape_for_json("{CorrValue}", "CorrValue_Escaped");
```

- Converts raw parameter into a JSON-safe version. - Ensures the request body contains literal `\r\n`, `\"`, and `\\`.

Output for verification

```
lr_output_message("Original Value: %s", lr_eval_string("{CorrValue}"));
lr_output_message("Escaped Value : %s", lr_eval_string("{CorrValue_Escaped}"));
```

- Prints both original and escaped values in the LR log for debugging. - Confirms that special characters were properly escaped.

Send POST request

```
web_custom_request("MyRequest",
    "URL=https://dummy.example.com/api",
    "Method=POST",
    "Body={\"text\":\"{CorrValue_Escaped}\"}",
    LAST);
```

- Sends a JSON POST request using the escaped parameter. - Guarantees that multi-line strings and special characters are not lost or modified by LoadRunner.

Why We Do It This Way

1. **Preserve CRLF in JSON**: LoadRunner may strip `\r\n` in parameters, breaking multi-line strings or Appian requests.
2. **Escape special JSON characters**: Backslashes and quotes need escaping to produce valid JSON.
3. **Global buffer avoids stack overflow**: Large local arrays cause compile errors, so `g_escapedBuffer` is declared globally.
4. **Reusable function**: `lr_escape_for_json()` can be used for any parameter across your script, making it consistent and maintainable.
5. **Safe and clear logging**: Outputs original and escaped values to verify correctness.

Summary

- Problem: Correlation values contain CRLF, quotes, or backslashes that LoadRunner may strip or mishandle in JSON.
- Solution: Escape all special characters and preserve formatting using a global buffer and helper function.
- Benefit: Ensures JSON POST requests are valid and data is interpreted correctly by the server (e.g., Appian).