# Virtuoso NLP Syntax Documentation - AI Optimized (Validated)

## Table of Contents



## Introduction

Virtuoso NLP (Natural Language Processing) syntax enables users to automate tests using natural language commands. It supports a wide range of actions, from navigation and clicking to assertions and API calls, making it a versatile tool for quality assurance (QA) teams. The syntax is designed to be intuitive, with features like variable support and special key handling, and is backed by AI and machine learning for enhanced functionality.

## System Parameters and Limitations

| Parameter | Value |
|---|---|
| Default Wait Time | 20 seconds |
| Maximum Wait Time | 10 minutes |
| File Download Limit | 20 MB |
| Comment Character Limit | 500 characters |
| Special Comment Keywords | `TODO`, `FIXME` (highlighted for better project management) |

## Syntax Categories

### 1. Navigation

**Commands:** `browse`, `go`, `navigate`, `open`

**Examples:**

```
Navigate to "https://google.com"
Navigate to "https://amazon.com" in new tab
```

**Notes:** Navigation commands direct the test to a specific URL or open a new tab.

### 2. Click

**Commands:** `click`

**Examples:**

```
Click on bottom "Submit"
Click on $variableTarget
Click on $name
```

**Notes:** Clicking can be forced using JavaScript `<element>.click()` if necessary.

**Force-clicking:** After executing a click step, you will sometimes see a step side effect of "Failed to directly click on element; the element was force-clicked." This means that Virtuoso was unable to use a regular browser interaction to click the element (using the WebDriver), and instead resorted to executing the JavaScript command `<element>.click()` in order to perform the interaction.

**Dynamic clicking:** You can use variables (e.g., `$name`) to target elements dynamically. This works for every command that has elements. For example: `Click on $name`

### 3. Write

**Commands:** `type`, `enter`, `write`

**Examples:**

```
Write "Joe" in field "First name"
Write "24" in field "Age"
```

**Notes:** Used to input text into form fields or other input elements.

### 4. Select

**Commands:** `select`, `pick`

**Examples:**

```
Pick "March" from "Month"
Pick option 7 from "Addresses"
```

**Notes:** Option indexes start at 1; negative indexes can be used to select from the end (e.g., -1 for the last option).

**Additional selection methods:**
You can also select options by their index either through ordinals up to ten (e.g., pick third option), last item (e.g., pick last option), or to specify it (e.g., pick option five, pick option 42).

### 5. Wait

**Commands:** `wait`, `pause`

**Examples:**

```
Wait 1 second
Wait 3 seconds for "Logged in!"
```

**Notes:** Default wait time is 20 seconds; maximum wait time is 10 minutes. Waiting for an element without specifying the time causes Virtuoso to wait for up to 20 seconds or until the element appears.

### 6. Mouse Actions

**Commands:** `mouse` with actions like `up`, `down`, `enter`, `over`, `hover`, `click`, `right click`, `double click`, `move`, `drag`

**Examples:**

```
Mouse double click "Element"
Mouse move to 100, 400
```

**Notes:** Drag and drop actions are not supported on iOS devices.

**Mouse coordinates:**

- `move to` accepts absolute x and y integer values to move the mouse to, while `move by` accepts relative x, y values.
- x refers to horizontal movement, while y refers to vertical movement.
- Negative coordinates are accepted for relative movement. For example `move by -10, 40`, moves the mouse backwards on the horizontal axis and forward on the vertical axis.
- The target coordinate can be provided as a variable using the `coordinate` keyword with formats:
  - `"10, -10"`
  - `[10, -10]`
  - `{ "x": 10, "y": -10 }`

**Click vs Mouse click:**

- `Click` and `Mouse click` are separate commands
- `Click` performs a natural user click with intelligent mechanisms (popup auto-dismiss, fallback to alternatives, etc.)
- `Mouse click` performs a simple mouse click on the position
- Prefer `Click` for general cases and only use `Mouse click` for special interactions like clicking on specific coordinates

## 7. Store

**Commands:** `store`

**Examples:**

```
Store element text of "password" in $user_pass
Store value "Hello World" in $myVariable
```

**Notes:** Used to capture and store values or element properties for later use.

**Store element details:**

This allows you to make sure you use the element on further test steps. Available element details can be seen by executing the journey with the store test step, and looking at variable contents in the side effect panel associated to the test step.

## 8. Frames and Tabs

**Commands:** `switch`

**Examples:**

```
Switch iframe to "search"
Switch to next tab
Switch to parent iframe
Switch to previous tab
```

**Notes:** Switching tabs resets the context to the main frame.

**Additional information:**

- **Parent frame:** Use `Switch to parent iframe` to return to the immediate parent frame of the current frame (not the top-level frame)
- **Traversing tabs:** The meaning of `previous` and `next` relates to the order of tabs displayed and not to when they were accessed. For example, when on 3rd tab and switching to 2nd, then calling `Switch to previous tab` will end up on 1st tab instead of the 3rd that was previously accessed.
- **Frame context reset:** Switching a tab always resets the context to the main frame, so if you switch back to a tab that had its last interactions within an iframe and want to continue interacting with elements inside it, you will need to switch to that iframe again.

## 9. Scroll

**Commands:** `scroll`

**Examples:**

```
Scroll to page top
Scroll to 20, 40
```

**Notes:** Allows scrolling to specific positions or elements on the page.

**Scroll input:** `scroll to` accepts absolute x and y integer values to scroll the browser window to, while `scroll by` accepts relative values.

### 10. Upload

**Commands:** `upload`

**Examples:**

```
Upload "https://example.com/file.pdf" to "Résumé:"
```

**Notes:** File uploads are not supported on Edge 15, all Safari browsers, or mobile devices.

### 11. Cookie

**Commands:** `cookie`

**Examples:**

```
Cookie create "login" as "username"
Cookie wipe all
```

**Notes:** Manages cookies, including setting, getting, and deleting them.

### 12. Window

**Commands:** `window`

**Examples:**

```
Window resize to 640, 480
```

**Notes:** Controls the browser window, such as resizing or maximizing.

### 13. Execute (Extension)

**Commands:** `execute`

**Examples:**

```
Execute "my script name"
```

**Notes:** Executes custom scripts or extensions.

**Data format:**

- Script input will always be passed as text, so you may need to deserialize it in your extension.
- Script output should be either text, primitive object/value, or a DOM element. Otherwise, the serialization result can be unexpected.
- The `toString()` and `JSON.stringify()` methods can come in handy when returning the output. For example: Use `new Date().toString()` instead of `new Date()`.

### 14. Dismiss

**Commands:** `dismiss`

**Examples:**

```
Dismiss alert
Dismiss prompt respond "text to write"
```

**Notes:** Handles alerts, confirms, and prompts by dismissing them or providing responses.

### 15. Press Keyboard Keys

**Commands:** `press`

**Examples:**

```
Press RETURN in "Search"
Press "CTRL_SHIFT_X"
Press "CTRL_a"
Press "CTRL_c"
Press "CTRL_x"
Press "CTRL_v"
Press "COMMAND_c"
Press F1 in "body"
```

**Notes:**

- Special keys should be in ALL CAPS and in quotes
- Does not support expressions
- Keys separated by underscore `_` indicate that they'll be held down together (only possible with modifier keys such as CTRL, COMMAND, SHIFT, ALT)
- When not specifying a target, commands target the active element
- To specifically target the body, use: `Press F1 in "body"`

**Common key combinations:**

- Select all: `Press "CTRL_a"` represents CTRL + A
- Copy: `Press "CTRL_c"` represents CTRL + C
- Cut: `Press "CTRL_x"` represents CTRL + X
- Paste: `Press "CTRL_v"` represents CTRL + V

**For macOS:**
Use COMMAND instead of CTRL: `Press "COMMAND_c"`

## 16. Assertions

**Commands:** Various assertion commands

**Assert Exists**

**Examples:**

```
Look for element "First name" on page
Look for element "Submit button" on page
Look for element $foo on page
```

**Notes:** Verifies if an element exists on a page. You can assert elements dynamically by using variables.

**Assert Not Exists**

**Examples:**

```
Assert that element "Error message" does not exist on page
Assert that element "//*[text()='Dashboard']" does not exist on page
```

**Notes:** Verifies that an element does not exist on a page. By default, information provided will match any property of elements in the page (including non-visible information). To match only visible text, use an xpath expression: `Assert that element "//*[text()='Dashboard']" does not exist on page`

**Assert Equals**

**Examples:**

```
Assert that element "First name" equals "John"
Assert $result equals "expected value"
```

**Assert Not Equals**

**Examples:**

```
Assert that element "Status" is not equal to "Failed"
```

**Assert Less Than**

**Examples:**

```
Assert that element "Price" is less than "100"
```

**Assert Less Than Or Equal**

**Examples:**

```
Assert that element "Quantity" is less than or equal to "50"
```

**Assert Greater Than**

**Examples:**

```
Assert that element "Total" is greater than "0"
```

**Assert Greater Than Or Equal**

**Examples:**

```
Assert that element "Score" is greater than or equal to "75"
```

**Assert Matches**

**Examples:**

```
Assert that element "Email" matches "/^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]
{2,}$/"
```

**Notes:** Regular expressions are indicated by placing them between `/` characters, with proper escaping of special characters like `\`.

**Assert Selected**

**Examples:**

```
Assert that "January" is selected in "Month"
```

**Notes:** Verifies if a given drop down list has the specified value selected.

**Assert Checked**

**Examples:**

```
Assert that element "Accept terms" is checked
```

**Notes:** Verifies if a checkbox or radio button is checked.

## 17. Comments

**Commands:** Comments start with `//`

**Examples:**

```
// This is a comment explaining the next step
// **TODO:** Implement this part when the feature is ready
// **FIXME:** This is currently failing
```

**Notes:** Comments are limited to 500 characters; `TODO` and `FIXME` are highlighted for project management.

## 18. API Call

**Commands:** `API call` or `api`

**Examples:**

```
API call "Salesforce.createaccount"($accountapi, $_endpoint, $version,
$NumberOfEmployees, $Name, $AccountNumber, $BillingStreet, $BillingCity,
$BillingPostalCode, "UK", $Rating, $token, $newfield) returning $response
API call "Salesforce.createopportunity"("opportunity", $_endpoint, $version, $token,
$recordid, $opportunityType, $leadSource, $stageName, $oppName, $CloseDate)
returning $response
API call "Salesforce.deleteaccount"($accountapi, $_endpoint, $recordid, $version,
$token) returning $response
API call "Salesforce.gettoken"($_endpoint, $clientid, $clientsecret, $code,
$redirecturi) returning $token
```

**Notes:** Used for making API requests. Parameters can be passed directly, and the response can be stored in a variable.

### 19. Variables

**Syntax:** Variables denoted with `$` (e.g., `$variableName`)

**Variable source representation (color-coding):**

- Yellow: Variables declared within the journey (e.g., output of `store` command)
- Blue: Variables declared in a test data table
- Green: Variables coming from a project environment
- Orange: Variables coming from data provided while triggering the execution
- Red: Unknown variables (indicating either a typo, a journey under maintenance, or a journey that can only execute successfully when triggered with initial data)

**Variable with default value:**

```
Write $message with default "hello world" in field "input"
```

**Variable precedence (in case of naming conflicts):**

1. Journey variables (highest priority)
2. Data table variables
3. Environment variables (lowest priority)

**Variable containing downloaded files:**

- `LAST_DOWNLOADED_FILE`: allows access to the file in any test step where a URL can be used
- `LAST_DOWNLOADED_FILE_DETAILS`: contains additional details about the downloaded file

**Execution context:**

- `VIRTUOSO_CONTEXT`: special variable available throughout execution, providing access to goal name and journey name

**Using expressions:**

- Format: `${expression}` (e.g., `${$variable + $anotherVar.toLowerCase()}`)
- Expressions are evaluated in the current page's context, allowing access to DOM and properties

## Target Element Definition

### Guess (hint) selectors

Given a case-sensitive guess selector, Virtuoso attempts to intelligently find an element to interact with. For example: `Click on "Login"`

### Relative position

You can provide `top`, `right`, `left`, `bottom` hints (or a mix such as `top right`) to distinguish between multiple elements. For example: `Click on top "Login"`

### Element type

When multiple elements of different types can match your query, you can provide more context to Virtuoso with element types:

- `link`: Links such as `<a>` elements
- `button`: Buttons such as `<button>` or `<input type="submit">` elements
- `input`: Input elements such as `<input>` or `<textarea>` elements
- `image`: Image elements such as `<img>` elements
- `dropdown`: Dropdowns such as `<select>` elements
- `file`: File input elements such as `<input type="file">` elements or file drop zones

Position ordering: In natural language, the type always comes after the position. For example: `Click on top right button "Login"`

### Standard selectors: CSS and Xpath

When writing a natural language test step, you can use standard selectors. For example:

```
Click on "#submit-button"
Click on "//button[@id='submit']"
```

### Advanced selectors: JS path

For targeting elements inside a shadow DOM or other complex scenarios, Virtuoso supports JS path selectors that can be added through the test step editor.

## Data-driven Testing

You can identify and locate elements in a page using a test data table:

1. Using a variable to store the element hint: `Click on $productTitle`
2. Creating data tables with the necessary values
3. Associating the test data with the journey

## Best Practices

1. Use `Click` for general cases of clicking elements, and only use `Mouse click` for special interactions
2. For expressions in press commands, first store the value and then use the variable
3. Consider wiping out large variables that are no longer needed by overwriting them
4. When using variables, be aware of their precedence order
5. Use the browser clipboard with standard key combinations ("CTRL_c", "CTRL_v", etc.)
6. Lock explicit selectors when dealing with dynamic elements or when you need to match elements containing specific text

## Compatibility Notes

- File upload is not supported in Edge 15, all Safari browsers, or mobile devices
- Drag and drop is not currently supported on iOS