
Posh Predictive Text

Release 0.1.1

David Plummer

Dec 11, 2023

CONTENTS:

1	Supported Command Line tools	3
2	Sponsorship	5
3	Screenshots	7
4	Quick Start	9
4.1	Prerequisites	9
4.2	Download and install PowerShell module	9
4.3	Update PowerShell profiles	9
4.3.1	Installation And Setup	9
4.3.2	Features	10
4.3.3	Cmdlet Help Files	12
4.3.4	Development	20
4.3.5	Upgrading And Change Log	30



PowerShell module providing predictive text completions for common CLI tools.

Most people will be familiar with predictive text on mobile phones. Posh-predictive-text brings the same capability to the PowerShell command line interface for common CLI tools used within the software development and data science community.

Modern command line tools are highly configurable and have many parameters. It is difficult to remember them all. Posh Predictive Text improves productivity, not only by providing suggested completions, but also providing a pop-up list of suggestions with tooltips. These tooltips remove the need to break off from enter the command and search documentation for the correct parameter name.

In addition, Posh Predictive Text is also able to suggest a limited number of parameter arguments. For instance when using `conda` to activate an environment a list of environments is shown.

PoshPredictiveText provides suggested completions in the following situations:

- When a partial argument is entered and the `tab` key is pressed. Successive suggestions appear each time the `tab` key is pressed.
- Pressing `ctrl + space` displays a popup list of options with tooltips.
- If `PSReadLine` options `-PredictionSource` is set to use the plugin and `-PredictionViewStyle` is set to `ListView` then suggestions will appear below the command line.

SUPPORTED COMMAND LINE TOOLS

Predictive text is available for the following commands line tools.

- conda

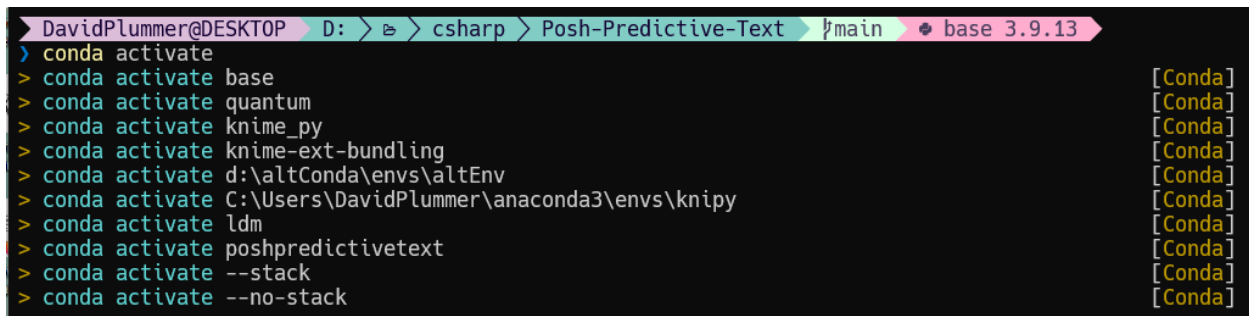
If a tools is not supported then please consider helping by developing the syntax tree file needed to support it. Further information is available in the developer documentation.

SPONSORSHIP

We all love open-source software. It's the freedom to use it and adapt it as you like. Its the no cost option. However, without funding open-source projects die. If you find this software useful then consider sponsoring the developers. It doesn't have to be a lot. The cost of a cup of coffee will do. Little contributions help and it is all very much appreciated.

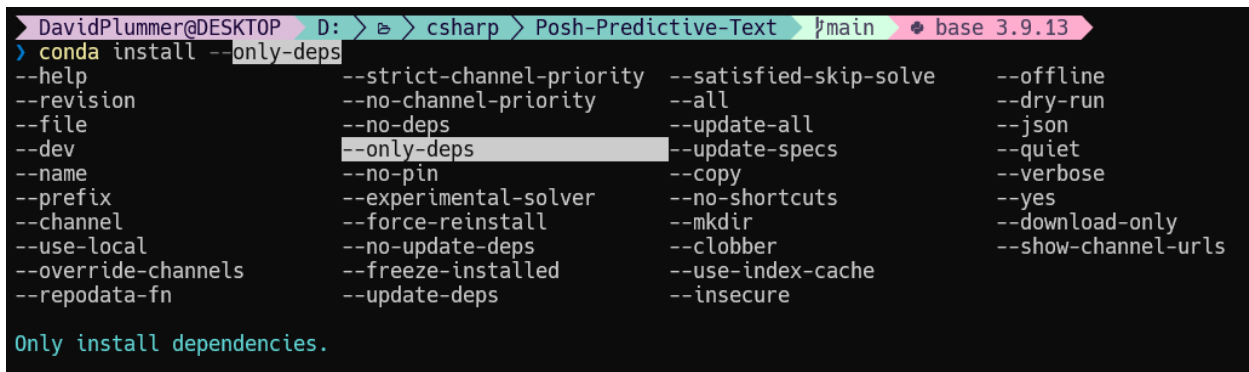
GitHub Sponsors	Ko-fi
---------------------------------	-----------------------

SCREENSHOTS



```
DavidPlummer@DESKTOP D: > csharp > Posh-Predictive-Text |>main base 3.9.13
> conda activate
> conda activate base [Conda]
> conda activate quantum [Conda]
> conda activate knime_py [Conda]
> conda activate knime-ext-bundling [Conda]
> conda activate d:\altConda\envs\altEnv [Conda]
> conda activate C:\Users\DavidPlummer\anaconda3\envs\knipy [Conda]
> conda activate ldm [Conda]
> conda activate poshpredictivetext [Conda]
> conda activate --stack [Conda]
> conda activate --no-stack [Conda]
```

Fig. 1: Suggestions appear in the drop-down list when Posh Predictive Text is configured as a PSReadLine plugin.



```
DavidPlummer@DESKTOP D: > csharp > Posh-Predictive-Text |>main base 3.9.13
> conda install --only-deps
--help --strict-channel-priority --satisfied-skip-solve --offline
--revision --no-channel-priority --all --dry-run
--file --no-deps --update-all --json
--dev --only-deps --update-specs --quiet
--name --no-pin --copy --verbose
--prefix --experimental-solver --no-shortcuts --yes
--channel --force-reinstall --mkdir --download-only
--use-local --no-update-deps --clobber --show-channel-urls
--override-channels --freeze-installed --use-index-cache
--repodata-fn --update-deps --insecure

Only install dependencies.
```

Fig. 2: Pressing ``ctrl+space`` shows a pop-up list of suggested commands with tooltips.

The following short video shows Posh Predictive Text in action.

QUICK START

Posh Predictive Text is available from PowerShell Gallery and is installed and activated using the following instructions. A more detailed set of instructions are available on the installation page.

4.1 Prerequisites

Posh predictive text requires PowerShell version 7.2 or greater and PSReadLine version 2.2.6 or greater.

4.2 Download and install PowerShell module

Install PoshPredictive text from the PowerShell Gallery. This will download the module to the local user account. You may be asked for permission if you have not already set PowerShell gallery as trusted source.

```
Install-Module -name PoshPredictiveText
```

4.3 Update PowerShell profiles

Add the following commands to the PowerShell profile. To locate the PowerShell profile open a command prompt and type *\$PROFILE*.

```
Set-PredictiveTextOption -RemoveCondaTabExpansion  
Install-PredictiveText
```

4.3.1 Installation And Setup

Pre-requisites

PoshPredictiveText requires PowerShell version 7.2 or greater, and PSReadLine version 2.6 or greater.

You can check the version of PowerShell using the following command.

```
$PSVersionTable.PSVersion
```

The latest version of PowerShell is available on [Github](#).

You can check the version of PSReadLine using the following command.

```
Get-Module PSReadLine | Format-List
```

Version 2.2.6 is pre-release and to install it the *AllowPreRelease* and *-force* options are required.

```
Install-Module PSReadLine -AllowPrerelease -Force
```

Installing Posh Predictive Text

Install PoshPredictiveText from the [PowerShell Gallery](#). This will download the module to the local user account. You may be asked for permission if you have not already set PowerShell gallery as trusted source.

```
Install-Module -name PoshPredictiveText
```

Add the following commands to the PowerShell profile. To locate the PowerShell profile open a command prompt and type *\$PROFILE*.

```
Set-PredictiveTextOption -RemoveCondaTabExpansion  
Install-PredictiveText
```

The first command removes tab-expansion that is already installed by conda, and which prevents Posh Predictive Text from providing completions. The second command installs Posh Predictive Text.

Completions will appear when a partial argument is entered and the tab key is pressed. A longer list of options with tooltips is available by pressing ctrl-space.

Configuring PSReadLine

It is recommended to add the following PSReadLine options in the PowerShell profile so that suggestions appear below the command line as command arguments are entered.

```
Set-PSReadLineOption -PredictionSource HistoryAndPlugin -PredictionViewStyle ListView
```

4.3.2 Features

Posh Predictive Text provides the features to improve productivity on the command line. These are listed for both the module and individual CLI commands.

Module Features

The following options are applicable to the Posh Predictive Text module. Options specific to each command line tool are listed on the page for each command.

List of Support CLI commands

To get a list of CLI commands supported by Posh Predictive Text.

```
Get-PredictiveTextOption -ListCommands
```

Version

To get the version of Posh Predictive Text

```
Get-PredictiveTextOption -Version
```

Initialisation Script

When posh predictive text is installed it runs a PowerShell script to register an argument completer. This script is executed by `Install-PredictiveText` and can be printed using the following command so that the code can be reviewed.

```
Get-PredictiveTextOption -PrintScript
```

Logging

Logging is not recommended.

The release version of Posh Predictive Text supports limited logging to a file to facilitate problem resolution. To set up logging the following command identifies the log file to which records are written and the level of logging required (INFO, WARN, ERROR).

```
Set-PredictiveTextOption -LogFile C:\temp\logfile.txt -LogLevel INFO
```

The path to the log file must exist - folders will not be created automatically and logging will not start if the log file cannot be created.

Conda

Conda is an open source package management system and environment management system. Posh Predictive Text provides suggestions for all commands and parameters as well as arguments listed below.

Suppress Built-in Conda Tab-Expansion

The script that initialises Conda in PowerShell installs a function providing limited tab-expansion capabilities. Unfortunately, this is not an optional install and it also prevents other tab-expansion providers working. To remove the Conda function from PowerShell the following option must be set prior to installing Posh Predictive Text.

```
Set-PredictiveTextOption -RemoveCondaTabExpansion
```

If Conda removes the tab-expansion functionality from their code in future releases then this option will not need setting.

Environments

Posh Predictive Text suggests completions for available environments. For example:

```
conda activate b
```

Pressing `tab` will cycle through environments starting with the letter `b`. Pressing `ctrl-tab` will provide a pop-up list of environments starting with the letter `b`, the toolTips will show the path to the environment. If there is a space after `activate` and `ctrl-tab` is pressed, then all environments will be shown.

The environments include both those defined using the `--Name` and the `--Path` parameters.

4.3.3 Cmdlet Help Files

The Posh Predictive Text module includes several PowerShell cmdlets. Each cmdlet has its own help file accessed using `Get-Help`. For example:

```
Get-Help Get-PredictiveTextOption
```

Online Help

The help files are also available online.

Install-PredictiveText

SYNOPSIS

Installs Posh Predictive Text and activates the plugin and tab-expansion of command arguments.

SYNTAX

```
Install-PredictiveText [<CommonParameters>]
```

DESCRIPTION

Install Posh Predictive Text and activates both the PSReadLine plugin and enables tab-expansion of command line arguments.

A list of supported commands is provided by `Get-PredictiveTextOption -ListCommands`.

PARAMETERS

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

INPUTS

None

OUTPUTS

System.String

System.Management.Automation.CompletionResult

NOTES

RELATED LINKS

[GitHub PoshPredictiveText Documentation](#)

Get-PredictiveTextOption

SYNOPSIS

Gets options for Posh Predictive Text.

SYNTAX

Version

```
Get-PredictiveTextOption [-Version] [<CommonParameters>]
```

ListCommands

```
Get-PredictiveTextOption [-ListCommands] [<CommonParameters>]
```

PrintScript

```
Get-PredictiveTextOption [-PrintScript] [<CommonParameters>]
```

DESCRIPTION

Gets information about and options set for Posh Predictive Text argument completer.

PARAMETERS

-ListCommands

List commands supported with predictive text.

```
Type: SwitchParameter  
Parameter Sets: ListCommands  
Aliases: List, l  
Required: False  
Position: Named  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-PrintScript

Print PowerShell script used to initialise Powershell Predictive Text.

```
Type: SwitchParameter  
Parameter Sets: PrintScript  
Aliases: Print, p  
Required: False  
Position: Named  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

-Version

Returns the version.

```
Type: SwitchParameter  
Parameter Sets: Version  
Aliases: Ver, v  
Required: False  
Position: Named  
Default value: None  
Accept pipeline input: False  
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

INPUTS

None

OUTPUTS

System.String

System.Management.Automation.CompletionResult

NOTES

RELATED LINKS

[GitHub PoshPredictiveText Documentation](#)

Set-PredictiveTextOption

SYNOPSIS

Sets options for Posh Predictive Text.

SYNTAX

Logging

```
Set-PredictiveTextOption -LogFile <String> [-LogLevel <String>] [<CommonParameters>]
```

RemoveCondaTabExpansion

```
Set-PredictiveTextOption [-RemoveCondaTabExpansion] [<CommonParameters>]
```

DESCRIPTION

Sets options for Posh Predictive Text.

EXAMPLES

Example 1

```
Set-PredictiveTextOption -RemoveCondaTabExpansion
```

Conda installs its own tab-expansion within PowerShell. The conda code has higher precedence and prevents Posh Predictive Text providing completions for conda commands. This command removes the conda code and allows Posh Predictive Text to provide completions for conda.

Example 2

```
Set-PredictiveTextOption -LogFile 'C:\logfiles\logfile.txt' -LogLevel INFO
```

Enables logging of diagnostic messages to a file. Messages are recorded against three levels: ERROR, WARN, INFO.

PARAMETERS

-LogFile

Enable logging to log file.

Type: String
Parameter Sets: Logging
Aliases:

Required: True
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

-LogLevel

Level of information to log (INFO, WARN, ERROR). Default: ERROR.

Type: String
Parameter Sets: Logging
Aliases:
Accepted values: INFO, WARN, ERROR

Required: False
Position: Named

(continues on next page)

(continued from previous page)

```

Default value: None
Accept pipeline input: False
Accept wildcard characters: False

```

-RemoveCondaTabExpansion

Remove conda installed tab expansion.

```

Type: SwitchParameter
Parameter Sets: RemoveCondaTabExpansion
Aliases:

Required: False
Position: Named
Default value: None
Accept pipeline input: False
Accept wildcard characters: False

```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

INPUTS

None

OUTPUTS

System.String

System.Management.Automation.CompletionResult

NOTES

RELATED LINKS

[GitHub PoshPredictiveText Documentation](#)

Get-PredictiveText

SYNOPSIS

Gets a list of suggested completions for command line arguments.

SYNTAX

```
Get-PredictiveText [[-WordToComplete] <String>] [[-CommandAst] <CommandAst>] [[-  
↪CursorPosition] <Int32>]  
[<CommonParameters>]
```

DESCRIPTION

Native argument completer used to provide tab-expansion suggestions on the PowerShell command line.

A list of supported commands is provided by `Get-PredictiveTextOption -ListCommands`.

EXAMPLES

Example 1

```
Register-ArgumentCompleter -CommandName $cmdNames -Native -ScriptBlock {  
    param(  
        [string]$wordToComplete,  
        [System.Management.Automation.Language.CommandAst]$commandAst,  
        [int]$cursorPosition)  
  
    try {  
        $suggestions = Install-PredictiveText -WordToComplete $wordToComplete -  
↪CommandAst $commandAst -CursorPosition $cursorPosition  
    }  
    catch {  
        Write-Host "Error."  
    }  
  
    $suggestions  
}
```

Registers `Install-PredictiveText` as a native argument completer for the commands listed in `$cmdNames`.

PARAMETERS

-CommandAst

Abstract Syntax Tree for current input line.

```
Type: CommandAst
Parameter Sets: (All)
Aliases:

Required: False
Position: 1
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-CursorPosition

Command entered by user at the prompt.

```
Type: Int32
Parameter Sets: (All)
Aliases:

Required: False
Position: 2
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

-WordToComplete

Value provided by the user before they pressed tab.

```
Type: String
Parameter Sets: (All)
Aliases:

Required: False
Position: 0
Default value: None
Accept pipeline input: False
Accept wildcard characters: False
```

CommonParameters

This cmdlet supports the common parameters: -Debug, -ErrorAction, -ErrorVariable, -InformationAction, -InformationVariable, -OutVariable, -OutBuffer, -PipelineVariable, -Verbose, -WarningAction, and -WarningVariable. For more information, see [about_CommonParameters](#).

INPUTS

None

OUTPUTS

System.String

System.Management.Automation.CompletionResult

NOTES

This cmdlet is intended for use as a registered argument completer and is not intended for use as a stand-alone cmdlet.

RELATED LINKS

[GitHub PoshPredictiveText Documentation](#)

4.3.4 Development

Application Development

PoshPredictiveText is developed using Visual Studio. The majority of the code is C# with a small amount of PowerShell scripting to register the native argument completer and remove conda tab-expansion code, if it is installed.

Coding Standards

It is expected that coding will follow the coding standards, however, it may take some time for the conventions used in this application to settle and code to comply with those conventions.

Coding should follow [Microsoft Coding Conventions](#).

Additional guidance provided by [Google Csharp Style Guide](#) is used where appropriate.

The following standards are used in the code which may conflict with the above:

- *var* should be used sparingly and only when the type is **very** obvious. Use of *var* can make code difficult to review if the type is not clear.
- Indentation is four spaces (no tabs).
- Don't use underscore to indicate private, protected, internal and protected internal fields. It adds noise.

- *const* should be in lower case, except where the value of the constant is determined by a third party application outside of the code. For instance, `CONDA_BASE` is capitalised to indicate that the value it contains must match that provided by a third party, and may be subject to change. The use of upper case is to draw attention to constant values which may generate errors if they do not match the third party application.
- Do not use braces for single statements where it improves readability, e.g. if statements followed by return to exit a method early `if (x is null) return null;`.

Testing

Testing uses Xunit, which should install from NuGet. Test coverage uses Fine Code Coverage, which is installed as an extension in Visual Studio.

Code should be covered with reasonable unit tests to validate that it works on multiple platforms.

All tests must pass before code is accepted.

PowerShell Help Documents

PowerShell has traditionally used MAML XML files for console help. These are difficult to prepare by hand. As an alternative, help files can be written in Markdown and then exported in required formats using the PowerShell PlatyPS module.

The PlatyPS module is described on [GitHub](#)

The PlatyPS module must be downloaded and installed on the local development machine.

```
Install-Module -Name platyPS -Scope CurrentUser
Import-Module platyPS
```

The base documentation is already created within the folder `PowerShellHelpDocs`.

To create the external helpfiles run the following command whilst in the `PowerShellHelpDocs` folder.

```
New-ExternalHelp .\ -OutputPath en-US\
```

Source Code Documentation

Documentation is compiled using Sphinx to create a static website that is shared using Read The Docs. Source documents are generated using several tools and then collated within the sphinx documentation.

1. Source code documentation is generated by Doxygen and compiled into an XML file. This file is then parsed by Breathe/Sphinx-csharp and added to sphinx documents. All source code should be fully documented.
2. The PowerShell Help files are copied into the sphinx source directory and compiled into the sphinx documents.
3. The CHANGELOG.md is copied into the sphinx source directory and added to the sphinx documents.
 - Doxygen: <https://doxygen.nl/index.html>
 - Sphinx: <https://www.sphinx-doc.org/en/master/>
 - Breathe: <https://breathe.readthedocs.io/en/latest/index.html>
 - Sphinx-csharp: <https://github.com/rogerbarton/sphinx-csharp>
 - Myst-parser: <https://myst-parser.readthedocs.io/en/latest/>

- Sphinx RTD Theme: <https://sphinx-rtd-theme.readthedocs.io/en/stable/>
1. Download and install Doxygen <https://doxygen.nl/download.html>. If installing on Windows, add the doxygen bin folder to the path C:\Program Files\doxygen\bin.
 2. Create a Python environment for the Sphinx tools and install packages.

```
conda create -n poshpredictivetext python=3.10 sphinx sphinx_rtd_theme breathe myst-  
↪parser
```

3. Install Sphinx-csharp to add c# support within Sphinx/Breathe.

```
pip install git+https://github.com/rogerbarton/sphinx-csharp.git
```

All documentation is generated during packaging and assumes that a ‘conda’ environment named poshpredictivetext is configured with the required dependencies.

API Documentation

Class: CommandAstVisitor

Note: Neither Doxygen nor sphinx-csharp support Python record types. Introduced in C# 9.

Once record types are recognised then details for Token will appear below.

The token is a record with the following values:

- readonly string Value - Text representing a token on the command line.
- readonly Type Type - The type of the token.

And the following read only properties which return true if the record is the indicated type:

- bool IsCommand
- bool IsCommandExpression
- bool IsCommandParameter
- bool IsStringConstantExpression

Warning: doxygenclass: Cannot find class “PoshPredictiveText::Token” in doxygen xml output for project “PoshPredictiveText” from directory: /home/docs/checkouts/readthedocs.org/user_builds/posh-predictive-text/checkouts/latest/docs/source/xml

class PoshPredictiveText.CommandAstVisitor : AstVisitor

Implements a visitor for the PowerShell abstract syntax tree. Reads the input no the command line and creates an ordered list of tokens representing user input.

Public Functions

override AstVisitAction **DefaultVisit** (Ast *ast*)

Default action if the token has not been processed by a more specific token handler. Adds the token text with a generic text type.

Param ast

Node in the abstract syntax tree.

Return

Flag to continue processing the abstract syntax tree.

override AstVisitAction **VisitCommand** (CommandAst *commandAst*)

Process commandAst node in the abstract syntax tree. Does not process this node and skips to next node in the tree.

Param commandAst

Node in the abstract syntax tree.

Return

Continue to next node.

override AstVisitAction **VisitCommandExpression** (CommandExpressionAst *commandExpressionAst*)

Process CommandExpressionAst node in the abstract syntax tree. Add a token to the list of tokens on the command line.

Param commandExpressionAst

Return

Continue to next node

override AstVisitAction **VisitCommandParameter** (CommandParameterAst *commandParameterAst*)

Process CommandParameterAst node in the abstract syntax tree. Add token to the list of tokens on the command line.

Param commandParameterAst

Node in the abstract syntax tree.

Return

Continue to next node.

override AstVisitAction **VisitConstantExpression** (ConstantExpressionAst *constantExpressionAst*)

Process ConstantExpressionAst node in the abstract syntax tree. Add token to the list of tokens on the command line.

Param constantExpressionAst

Node in the abstract syntax tree.

Return

Continue to next node.

override AstVisitAction **VisitStringConstantExpression** (StringConstantExpressionAst *stringConstantExpressionAst*)

Process stringConstantExpressionAst node in the abstract syntax tree.

Identifies Gnu/Posix formatted parameters which start with a double dash and reclassifies them as CommandParameterAst type.

Adds token to the list of tokens on the command line.

Param stringConstantExpressionAst

Node in the abstract syntax tree.

Return

Continue to next node.

Predictor: Predictor

class PoshPredictiveText.Init : IModuleAssemblyInitializer, IModuleAssemblyCleanup

Register the predictor on module loading and unregister it on module un-loading.

Public Functions

void **OnImport** ()

Gets called when assembly is loaded.

void **OnRemove** (PSModuleInfo *psModuleInfo*)

Gets called when the binary module is unloaded.

class PoshPredictiveText.Predictor : ICommandPredictor

PSReadLine plugin providing predictive text capabilities.

Public Functions

SuggestionPackage **GetSuggestion** (PredictionClient *client*, PredictionContext *context*,
CancellationToken *cancellationToken*)

Get the predictive suggestions. It indicates the start of a suggestion rendering session.

Param client

Represents the client that initiates the call.

Param context

The PredictionContext object to be used for prediction.

Param cancellationToken

The cancellation token to cancel the prediction.

Return

An instance of SuggestionPackage.

bool **CanAcceptFeedback** (PredictionClient *client*, PredictorFeedbackKind *feedback*)

Gets a value indicating whether the predictor accepts a specific kind of feedback.

Param client

Represents the client that initiates the call.

Param feedback

A specific type of feedback.

Return

True or false, to indicate whether the specific feedback is accepted.

void **OnSuggestionDisplayed** (PredictionClient *client*, uint *session*, int *countOrIndex*)

One or more suggestions provided by the predictor were displayed to the user.

Param client

Represents the client that initiates the call.

Param session

The mini-session where the displayed suggestions came from.

Param countOrIndex

When the value is greater than 0, it's the number of displayed suggestions from the list returned in *session*, starting from the index 0. When the value is less than or equal to 0, it means a single suggestion from the list got displayed, and the index is the absolute value.

void **OnSuggestionAccepted** (PredictionClient *client*, uint *session*, string *acceptedSuggestion*)

The suggestion provided by the predictor was accepted.

Param client

Represents the client that initiates the call.

Param session

Represents the mini-session where the accepted suggestion came from.

Param acceptedSuggestion

The accepted suggestion text.

void **OnCommandLineAccepted** (PredictionClient *client*, IReadOnlyList<string> *history*)

A command line was accepted to execute. The predictor can start processing early as needed with the latest history.

Param client

Represents the client that initiates the call.

Param history

History command lines provided as references for prediction.

void **OnCommandLineExecuted** (PredictionClient *client*, string *commandLine*, bool *success*)

A command line was done execution.

Param client

Represents the client that initiates the call.

Param commandLine

The last accepted command line.

Param success

Shows whether the execution was successful.

Properties

Guid **Id** { **get**; **set**; }

Gets the unique identifier for a subsystem implementation.

string? **Name** { **get**; **set**; }

Gets the name of a command for which suggestion are being made.

string **Description** { **get**; **set**; }

Gets the description of a subsystem implementation.

Cmdlet: GetPredictiveText

class PoshPredictiveText.GetPredictiveText : PSCommandlet

The Posh Predictive Text cmdlet provides suggested completions for command arguments in PowerShell.

Properties

string? **WordToComplete** { **get**; **set**; }

Gets or sets the partial word provided before the user pressed Tab.

CommandAst? **CommandAst** { **get**; **set**; }

Gets or sets the Abstract Syntax Tree (AST) for the current input line.

int? **CursorPosition** { **get**; **set**; }

Gets or sets the position of the cursor when tab was pressed.

Cmdlet: GetPredictiveTextOption

class PoshPredictiveText.GetPredictiveTextOption : PSCommandlet

The Posh Predictive Text cmdlet provides suggested completions for command arguments in PowerShell.

Properties

SwitchParameter **Version** { **get**; **set**; }

Gets or sets the parameter requesting the version to be display.

SwitchParameter **ListCommands** { **get**; **set**; }

Gets or sets the switch indicating List, ListCommand or L flag supported commands.

SwitchParameter **PrintScript** { **get**; **set**; }

Gets or sets a flag to print the PowerShell script used to initialise tab-expansion of arguments.

Cmdlet: InstallPredictiveText

class PoshPredictiveText.InstallPredictiveText : PSCommandlet

The Posh Predictive Text cmdlet provides suggested completions for command arguments in PowerShell.

Cmdlet: SetPredictiveTextOption

class `PoshPredictiveText.SetPredictiveTextOption` : `PSCmdlet`

The Posh Predictive Text cmdlet provides suggested completions for command arguments in PowerShell.

Properties

`string? LogFile { get; set; }`

Gets or sets the file to which messages will be written.

`string? LogLevel { get; set; }`

Gets or sets the flag indicating *Init* command generates and invokes a PowerShell script to add the cmdlet as an Argument Resolver in PowerShell.

`SwitchParameter RemoveCondaTabExpansion { get; set; }`

Gets or sets the flag that conda tabexpansion should be removed.

Syntax Tree Development

The tab-completion algorithm uses a syntax tree to determine valid completion options. Each command has its own syntax tree which is loaded at first use of that command.

Command syntaxes typically follow similar patterns. They start with one or several command words to identify which programme or module within the system is going to process the command. For instance, Conda commands start with the command `conda`, followed by one or more sub-commands such as `list`, `install`, `config`. The syntax tree identifies all permutations of commands and sub-commands, the leaves of the tree identifying callable commands and their arguments.

The syntax tree starts with the base command (e.g. `conda`) and then identifies each of the sub-commands that may be added (e.g. `install`). The leaves are identified by concatenating the commands with a period (.) to create a path (e.g. `conda.install`).

Each leaf can be followed by either another sub-COMMAND (e.g. `install`), an OPTIONAL parameter (e.g. `--help`), a PARAMETER with values (e.g. `--name environment_name`), or POSITIONAL parameters (e.g. `conda install Package_1 Package_2 Package_3`). The syntax tree has one row for each argument that may be added to that command. Therefore, the same path `conda.install` may appear in the syntax tree with several rows identifying unique arguments. The arguments for each leaf node must be unique and must identify their type.

As the user enters text at the prompt, the algorithm scans the syntax tree to identify suitable suggestions for next words. It uses the command path (`conda.install`) to filter results as the command is entered; and then suggest further commands and their arguments as more text is entered. Note that certain optional arguments may appear more than once, whilst others are unique; and some parameter arguments may also have more than one value, whilst others are unique. The algorithm also needs to identify when the command is expecting a value to be entered and not to suggest further arguments at that point; ideally suggesting possible values where the choice is limited.

PowerShell parses input at the command line to create a PowerShell Command Abstract Syntax Tree (CommandAST); note this is different from this *language* syntax tree used to describe permissible commands and arguments. This contains a parsed list of words entered by the user when the `tab` key is pressed. The final leaf in the command is the partial word to be completed, and for which suggestions are required. The algorithm visits the CommandAST and constructs a list of tokens, each token representing a word entered by the user. In this description, where the word token is used, it means an individual word entered by the user at an ordered position on the command line. The algorithm

uses knowledge in the language syntax tree to enhance information about tokens in the CommandAST to suggest likely user input.

The language syntax tree needs the following information to determine for any command path (leaf) possible future tokens and whether additional information (parameter values) is also required:

- **COMMAND** type identifies additional sub-commands that may be added to the command path to increase the commands specificity. For example, if the command path is `conda` then there are **COMMAND** entries for `activate`, `clear`, `install`, ... which then lead to the next level in the tree `conda.activate`, `conda.clear`, `conda.install`. The sub-command text is recorded within the **ARGUMENT** column of the syntax tree and **MULTIPLE_USE** is `false`. An alias for the command can also be recorded in the **ALIAS** column.
- **OPTIONAL** type are parameter flags (e.g. `--help`) that do not require parameter values. The text for the parameter is recorded in **ARGUMENT**, and an **ALIAS** may be entered. By convention, the **ARGUMENT** column should hold the most descriptive text, and the **ALIAS** the shortened form. For example, **ARGUMENT** : `--help`, **ALIAS**: `-h`. Some parameter flags may be used multiple times; if this is the case the **MULTIPLE_USE** must be `true` otherwise it must be `false`.
- **PARAMETER** type are parameters that are followed by values. As with **OPTIONAL**, they may have both an **ARGUMENT** and an **ALIAS**. They must also indicate whether they are **MULTIPLE_USE**. In addition, **PARAMETER** type must identify what value information is required. This is recorded in the **PARAMETER** column, which is a unique value, in capital letters, that the algorithm uses to determine what values to suggest. The **MULTIPLE_PARAMETER** flag must be set to indicate whether multiple values can be entered (`true`) or not (`false`).
- **POSITIONAL** type are parameter values that are entered after the command, but not preceded by a **PARAMETER** identifier. For instance, `conda.activate` is followed by a positional parameter to indicate the environment to be activated. The **POSITIONAL** type has no **ARGUMENT** or **ALIAS**; but does require the **PARAMETER**, **MULTIPLE_USE** (always `false`) and **MULTIPLE_PARAMETER** columns (can be `true` or `false`) with the same rules as a **PARAMETER** type.

Each command path will have one or more entries in the syntax tree, one for each **COMMAND**, **OPTIONAL**, **PARAMETER** or **POSITIONAL** suggested option. Each can also have a descriptive tooltip to provide the user with additional information in situations where tooltips can be displayed. It is recommended to keep tooltips as short as possible - they are helpful prompts and not intended as a substitute for help text or a language specification.

The syntax tree is stored in a table with the following columns.

Table 1: Syntax Tree Table Structure

Column Name	Type	Mand -atory	Description
COMMAND	string	yes	The command or sub-command token entered at the prompt.
COMMAND_PATH	string	yes	The full command (command and sub-command tokens). Each argument separated by a period.
TYPE	string	yes	The type of argument (COMMAND, OPTIONAL,PARAMETER, POSITIONAL).
ARGUMENT	string	yes ¹	The text representation of argument.
ALIAS	string	no	An alternate text representation of the argument.
MULTIPLE_USE	bool	yes	True if the argument can appear multiple times on The command line.
PARAMETER	string	no ²	The name of the parameter value. The name is used to call specific parameter value handlers.

4.3. Update PowerShell profiles

MULTIPLE_PARAMETER	bool	no ^{Page 30, 2}	True if the parameter has multiple values.
--------------------	------	--------------------------	--

4.3.5 Upgrading And Change Log

Recent versions of Posh Predictive Text have a corresponding git tag for each version released to [PowerShell Gallery](#).

Changelog

All notable changes to this project will be documented in this file.

The format is based on [Keep a Changelog](#) and this project adheres to [Semantic Versioning](#).

[0.1.1] - 2022-09-16

Changed

- Improved documentation for PowerShell Gallery.

[0.1.0] - 2022-09-16

Added

- Initial release.
- New text completions: `conda`.

Changed

- Nothing

¹ The argument is mandatory for COMMAND, OPTIONAL and PARAMETER types; but does not apply to the POSITIONAL type.

² The parameter and multiple parameter columns are mandatory for PARAMETER and POSITIONAL types; but does not apply to COMMAND and OPTIONAL types.