

# Asheninka User Documentation

*H. Andrew Black*

*SIL International*

asheninka\_support@sil.org

14 August 2020

Version: 0.7.0 Alpha

## Contents

1	Introduction . . . . .	2
2	General user interface items . . . . .	3
2.1	Creating a new project . . . . .	3
2.2	Opening an existing project . . . . .	3
2.3	Saving a project . . . . .	3
2.4	Save a project as a new project . . . . .	4
2.5	Setting fonts for vernacular and analysis language display . . . . .	4
2.6	Setting hyphenation parameters . . . . .	4
2.7	User interface language . . . . .	4
2.8	Help menu items . . . . .	5
3	Approaches to syllabification . . . . .	5
4	CV pattern approach . . . . .	5
4.1	CV segment inventory . . . . .	6
4.2	CV natural classes . . . . .	7
4.3	CV syllable patterns . . . . .	8
4.4	Grapheme natural classes . . . . .	8
4.5	Environments . . . . .	9
4.6	CV words . . . . .	11
4.7	CV Try a Word . . . . .	12
4.8	Predicted vs. correct CV words . . . . .	16
5	Sonority hierarchy approach . . . . .	17
5.1	SH segment inventory . . . . .	18
5.2	Sonority hierarchy . . . . .	18
5.3	Grapheme natural classes . . . . .	19
5.4	Environments . . . . .	19
5.5	Sonority hierarchy words . . . . .	19
5.6	Sonority hierarchy Try a Word . . . . .	19
5.7	Predicted vs. correct Son. Hier. words . . . . .	22
6	Onset-nucleus-coda approach . . . . .	22

6.1	ONC segment inventory . . . . .	23
6.2	Sonority hierarchy . . . . .	23
6.3	Syllabification parameters . . . . .	24
6.4	CV natural classes . . . . .	24
6.5	ONC templates . . . . .	24
6.6	ONC filters . . . . .	25
6.7	ONC words . . . . .	26
6.8	Predicted vs. correct ONC words . . . . .	26
6.9	Grapheme natural classes . . . . .	27
6.10	Environments . . . . .	27
6.11	ONC Try a Word . . . . .	27
7	Moraic approach . . . . .	30
8	Nuclear projection approach . . . . .	30
9	Optimality theory approach . . . . .	30
10	Converting predicted syllabification to correct syllabification . . . . .	30
11	Project management . . . . .	31
12	Importing and exporting words . . . . .	32
12.1	Import words . . . . .	32
12.2	Export words . . . . .	32
13	Other tools . . . . .	33
13.1	Find a word . . . . .	33
13.2	Remove all words . . . . .	34
13.3	Remove correct syllable breaks in all words . . . . .	34
13.4	Compare two implementations . . . . .	35
13.4.1	CV pattern approach comparison . . . . .	36
13.4.2	Sonority hierarchy approach comparison . . . . .	37
13.4.3	Onset-nucleus-coda approach comparison . . . . .	39
13.5	Compare syllabification results between approaches . . . . .	41
A.	Why is it called <b>Asheninka</b> ? . . . . .	42
B.	ONC algorithm in detail . . . . .	43
	References . . . . .	45
	Index . . . . .	46

## 1 Introduction



The **Asheninka** program is a linguistic tool with two goals:

1. Explore various syllabification algorithms.
2. Provide a principled way to insert discretionary hyphens in a list of words (which can then be used for typesetting text).

The first is clearly a linguistic research goal. The second is a practical application of the first goal. While one probably will want to use IPA (IPA 2016) for the first, the second should be done using the practical orthography.

## 2 General user interface items

This section notes some common items found in various places throughout the user interface. It also discusses some of the more basic **File**, **Settings**, and **Help** menu items.

While editing data, you add a new item by using the **Edit / New item** menu item or clicking on the  button. To delete an item, use the **Edit / Remove item** menu item or click on the  button.


A number of these have a field called “Active”. If the checkbox is checked, then this item will be used when syllabifying. If it is not checked, then it will not be used. This lets you experiment with various possibilities.

When the view contains a pane with rows and columns, you can click on a column header to sort by that column (it currently only sorts by Unicode code point). You can even hold the Shift key down and click on another column header to get a secondary sort (or on a third column to get a tertiary sort). There are some tables which cannot be sorted because the user controls the order manually. In such cases (such as **CV Syllable Patterns**), the sorting is disabled.

**Asheninka** will attempt to remember the location and size of the main window and most dialog boxes. So if you find that a dialog box is too small or too large, try resizing it (by dragging the edges of the window). Usually the next time you use that dialog box, it will be at the location and size you changed it to.


### 2.1 Creating a new project

**Asheninka** comes with a stock set of segments, etc. That is, when you create a new project, **Asheninka** will fill this new project with this stock set of segments, etc. You can then edit them, delete them, and add any needed items.


When you want to create a new project, use **File / New Project** menu item or click on the  toolbar button. This brings up a standard “Save as...” dialog box so you can save this stock set of data as your new project. The expected file extension to use with all **Asheninka** files is “.ashedata” (for **Asheninka** data). Key the file name you want and press OK.

We suggest you create a directory called “My Asheninka” in your normal documents directory and put all your **Asheninka** projects in it.

### 2.2 Opening an existing project

When you already have an existing **Asheninka** project and you want to open it into **Asheninka**, use the **File / Open Project...** menu item or click on the  toolbar button. This brings up a standard “File Open...” dialog box. Find the file you want and click on OK.

### 2.3 Saving a project

While **Asheninka** will automatically save your work about every 30 seconds, you can also use the **File / Save Project** menu item or click on the  button in the toolbar.

## 2.4 Save a project as a new project

When you decide to save the current project with a different name, use the **File / Save Project As...** menu item. The resulting file will have all the data that the original project did except it will have a new name.

## 2.5 Setting fonts for vernacular and analysis language display

Use the **Settings** menu item to set the font information for displaying the vernacular and/or the analysis languages. Currently, only font family, font size, and font style can be set. We do plan to include color and sorting information in a later version.

## 2.6 Setting hyphenation parameters

When one exports syllabified words in one of three possible formats (as explained in section 12.2), one can also control three variables for each of the three formats:

1. The discretionary hyphen character sequence used.
2. The number of printable characters from the beginning of the word after which hyphenation starts.
3. The number of printable characters from the end of the word where hyphenation stops.

The default values for these can be changed by using the **Settings / Hyphenation Parameters** menu item. This then shows the three export options. Choosing one brings up a dialog box showing the current hyphenation parameter settings for this export method. You can change the values and then click on the **OK** button.

## 2.7 User interface language

You can set the user interface language by using the **Settings / Change the interface language** menu item. This brings up a dialog box showing the current interface language in a drop down chooser. Click on the chooser's drop down button to see other interface language choices. The choices given use the name of the language in the current interface language followed by the name of the language in its own language. So if the current interface language is English, then it will show “Spanish (español)” as an option; if the current interface language is Spanish, it will show English as “inglés (English)”. When you click on the "OK" button, the program will "flash" and show with the new user interface language.

The current version has English, French<sup>1</sup> and a rough, most likely often inaccurate version of Spanish. Any corrections to any of the languages are welcome.

---

<sup>1</sup>Many thanks to the folks at [Mission Assist](#).

## 2.8 Help menu items

Currently, there are four **Help** menu items:

1. **User Documentation** which shows this document.
2. **Suggested Steps** which has some suggestions on how to be effective when using **Asheninka**.
3. **Introduction to syllabification** which has some general discussion of syllabification issues.
4. **About Asheninka** which has some standard information about the current version of **Asheninka**.

## 3 Approaches to syllabification


**Asheninka** will eventually offer six different syllabification algorithms or approaches. These are covered in sections 4–9. Only two are implemented in the current version. See sections 4 and 5.

The various approaches or algorithms are explained and illustrated in the “Overview” document. You can read this document by clicking [here](#) or by using the **Help / Introduction to syllabification** menu item.

## 4 CV pattern approach

In the CV pattern approach, you need to define the following items:

1. segments (section 4.1)
2. natural classes (section 4.2)
3. syllable patterns (section 4.3)
4. grapheme natural classes (section 4.4)
5. environments (section 4.5)

After that, you import a word list (see section 12) or enter a list of words by hand. Then you use the **Parser / Syllabify Words** menu item or click on the  tool bar button. This will apply the algorithm of the CV pattern approach to all the words. You can then see the results in the **CV Words** view. See sections 4.6 and 4.7 for ideas on how to check the results, among other things.

The main “game” to play with the CV pattern approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), natural classes, and syllable patterns so that one gets most, if not all, words to syllabify correctly.

The main algorithm is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of **“Failure: could not parse into segments beginning at 'some characters’”** and quits. The *'some characters'* part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met. You can use the **Try a Word** tool to get more details (see section 4.7).
2. If it succeeded in finding a sequence of segments, it then performs a left-to-right sweep of the segments it posited, trying to find a sequence of natural classes that covers the entire sequence of segments. It uses the set of segments or natural (sub-)classes defined within a natural class to determine a match with a natural class. If two or more natural classes match, it tries each of them in turn. If it cannot find a sequence of natural classes that covers the entire sequence of posited segments, it reports an error of **“Failure: could not parse into natural classes; did find classes 'some classes' covering graphemes 'some graphemes’”** and quits. The *'some classes'* part indicates the sequence of natural classes that were found and the *'some graphemes'* part indicates which graphemes were included in those natural classes. Hopefully this will help you figure out why other natural classes were not found for this word.
3. If it succeeded in finding a sequence of natural classes, it performs a left-to-right sweep of the sequence of the natural classes it posited, trying to find a sequence of syllable patterns that covers the entire sequence. It tries the syllable patterns in the order they are arranged in the **CV Syllable Patterns** view (see section 4.3 below). If it cannot find a sequence of syllable patterns that covers the entire sequence of posited natural classes, it reports an error of **“Failure: could not parse natural classes into syllables”** and quits.
4. If it succeeded in finding a sequence of syllable patterns, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports **“Success”**.


We now give more information on the various views available in the user interface for the CV pattern approach.

#### 4.1 CV segment inventory

Make sure that what is in the **Segment Inventory** view covers all the segments you have in the orthography. **Asheninka** tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme.

In the “Segment” field, key a letter or letters that will help you remember this segment. For example, a low central unrounded vowel might be keyed as **a** while a voiceless alveopalatal affricate might be keyed as **tʃ** or as **ch**.

The “Description” field is for your benefit. Key whatever helps you (and anyone looking at your data) know what the segment is. For example, you could key its phonetic or phonological description.

In the “Graphemes” field, key all the ways the particular segment appears in your orthography, including any upper case forms. Separate each one by a space. When you leave this field (e.g., by clicking in the “Description” field or by pressing the **Tab** key or using **Shift-Tab**), **Asheninka** automatically shows the graphemes in the table at the bottom. You can choose to disable one or more of the graphemes by clicking on the check box in the table at the beginning of the row for the grapheme. You can also select one or more environments where the given grapheme is valid by clicking on the chooser button at the end of the row (see section 4.5 and also section 4.4 for more on environments). The chooser button looks like . Note that whenever a given grapheme can occur as more than one segment, then be sure to include the appropriate environments for the grapheme in every such segment.

To remove a grapheme from the table, merely remove it from the “Graphemes” field. Note that when you uncheck a grapheme check box at the beginning of the row, **Asheninka** will automatically remove that grapheme from the “Graphemes” field. When you check that check box, **Asheninka** will automatically add it back to the “Graphemes” field. Thus, you can uncheck the box to see what might happen if this grapheme is no longer present. This can be useful when testing environments to condition a grapheme.


It is possible to have the same grapheme occur for more than one segment. If you do this, you may well want to add environments for when each grapheme is valid for a given segment.

## 4.2 CV natural classes

Make sure that in the **CV Natural Classes** view, every segment is in a natural class. A given segment may be in more than one class.

The “Name” field is for you to define how this natural class will appear in syllable patterns. We suggest using something short and if there is more than one character in it, make the first one be capitalized and the rest be in lower case.<sup>2</sup> While it is possible to use the same name more than once, it will most likely turn out to be confusing. So we recommend you make the name be unique.

The “Description” field is for your use to document anything special about this natural class.

The “Segment or Natural Class” field has a chooser button on the far right which looks like . Click on it to bring up a dialog box which lets you select segments and/or natural classes which belong to this natural class. Note that it is possible to insert one entire natural class within another natural class. For example, if you have a class of nasal consonants, you can include that class within another class that has all consonants.


<sup>2</sup>This is not required, it is just a suggestion.



### 4.3 CV syllable patterns

Use the **CV Syllable Patterns** view to create CV syllable patterns that cover the kinds of syllables you think the language has. Be sure to allow for vowel-initial syllables if there are vowel-initial syllables in the language.

The “Name” field is for you to define a short name for this syllable pattern. We suggest using something short. The “Description” field is for your use to document anything special about this syllable pattern.

The “Natural Classes” field shows the sequence of any natural classes this pattern consists of. It has a chooser button on the far right which looks like . Click on it to bring up a dialog box which lets you select natural classes which constitute this syllable pattern. This chooser consists of one or more drop-down boxes. Click on the drop-down arrow in the box. It will show you the list of possible natural classes to choose from. In addition, if this is the very first drop-down box or the last one currently being shown, it will also include “Word boundary”. When you choose “Word boundary”, the pattern shown above the drop-down box(es), will show “#” to indicate a word boundary.

Finally, the order in which the syllable patterns occur is important, as mentioned in section 4. Recall that with the CV pattern approach, as **Asheninka** parses a word, it tries to match the patterns in the order in which they appear. You control the order of the syllable patterns by clicking on a pattern in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.


### 4.4 Grapheme natural classes

When your orthography is such that you need to condition some graphemes by one or more environments and you want to capture a generalization about which graphemes can occur around that grapheme via a natural class, then create one or more grapheme natural classes. Note that these are not the same as CV natural classes. The grapheme natural classes are based on the set of graphemes defined in all the segments. The CV natural classes, on the other hand, are defined based on the segments in the segment inventory. The first can be used to determine if a given grapheme is valid for its segment in a given orthographic environment. The second is used to define a class for a segment during the parsing process of a word into syllables.

The “Name” field is for you to define how this grapheme natural class will appear in environments. We suggest using something short and if there is more than one character in it, make the first one be capitalized and the rest be in lower case. The name should be unique among all the grapheme natural classes. If two or more grapheme natural classes share the same name, then any environment which refers to such a grapheme natural class will always use the first one **Asheninka** happens to find.

The “Description” field is for your use to document anything special about this grapheme natural class.



The “Grapheme or natural class” field has a chooser button () at the right edge. Clicking on this brings up a chooser where you can select the graphemes and/or other grapheme natural classes that should belong to the grapheme natural class you are defining. Note that it is possible to insert one entire grapheme natural class within another grapheme natural class. For example, if you have a class of nasal consonants, you can include that class within another class that has all consonants.

## 4.5 Environments

When your orthography is such that you need to condition some graphemes by one or more environments, you add the environments here. Besides the usual Active check box, there are three fields.

The “Name” field is for you to give a short name for this environment.

The “Description” field is for your use to document anything special about this environment, including why it is needed.

The “Environment” field is where you key the environment. Any error message will appear below the Active check box as you type. (When you create a new environment, this field contains “/ \_ ” which causes an error message of **A class or a grapheme is missing; detected at or about position 6**. This is because the environment is not valid until it has at least one grapheme, grapheme natural class, or word boundary symbol.)

The content of the “Environment” field follows a special notation. This notation is one that is reminiscent of what is used in many generative-style rules. The basic rules of thumb are:

- (1)
  1. Begin each environment with the forward slash character /
  2. The location of the grapheme itself is indicated by an underscore character \_
  3. Any letters or grapheme natural classes that must come before the grapheme are typed in before this underscore character. Type them in the order in which they must appear.
  4. Any letters or grapheme natural classes that must come after the grapheme are typed in after this underscore character. Type them in the order in which they must appear.
  5. Any letter is indicated by typing it the way it appears in the practical orthography.
  6. Any grapheme natural classes are indicated by
    - a. typing a left square bracket [,
    - b. typing the name of the grapheme natural class,<sup>3</sup> and
    - c. then typing a right square bracket ]

---

<sup>3</sup>This is another reason why you should use unique names for grapheme natural classes. If you have two or more grapheme natural classes with the same name, it is not clear which one you mean. **Asheninka** will automatically select one, but it may not be the one you intended.

When you type a left square bracket, a drop-down box will appear which has the list of grapheme natural class names. You can click on this drop-down box and then click on the name you want to use.<sup>4</sup>

7. Optional letters or grapheme natural classes are indicated by
  - a. typing an opening parenthesis (,
  - b. typing the letter or grapheme natural class as above, and
  - c. then typing a closing parenthesis )
  - d. Note that one should not nest optional items. The **Asheninka** parser will not handle these properly. You must enter each optional item after the other.

Example (2) gives some sample environments along with what they mean.

(2)	Environment	Meaning
	/ m _	after an m
	/ [V] _	after a vowel (assuming there is a grapheme natural class of vowels called V)
	/ # i _	after a word initial i
	/ # [V] _	after a word initial vowel (assuming there is a grapheme natural class of vowels called V)
	/ [V] y _	after a vowel (assuming there is a grapheme natural class of vowels called V) and a y
	/ _ i	before an i
	/ _ [C]	before a consonant (assuming there is a grapheme natural class of consonants called C)
	/ _ y #	before a y which is word final
	/ _ [C] #	before a word final consonant (assuming there is a grapheme natural class of consonants called C)
	/ m _ w	between an m and a w
	/ [C] _ [C]	between two consonants (assuming there is a grapheme natural class of consonants called C)
	/ ai _	after an a and an i
	/ _ ai	before an a and an i
	/ _ (a) i	before an optional a and an i; that is, either before ai or before i
	/ _ ([C]) #	before an optional word final consonant (assuming there is a grapheme natural class of consonants called C); that is, either before a word final consonant or word finally

A given grapheme may have more than one environment, in which case the various environments are logically ORed with each other. That is, if any one of

<sup>4</sup>You can also press the **Tab** key two times to get to the drop-down box. Then use the **down-arrow** key to move to the one you want and press the **Enter** key. Please note that currently if you click on the drop-down box with the mouse and then use the **down-arrow** key, it will choose the first item in the list. I have yet to figure out how to avoid this poor behavior.

the environments for the grapheme are found, then the grapheme is considered to be valid (as far as its environments are concerned). For example, if a given grapheme can appear either before a consonant or word finally, then you can list both an environment for “before a consonant” and one for “before a word boundary.” Example (3) shows what this might look like, assuming that you have a grapheme natural class of consonants with an abbreviation of C.

$$(3) \quad \frac{\quad}{\quad / \_ [C] \quad / \_ \#}$$

## 4.6 CV words

In the **CV Words** view, the middle pane of the display shows all the words with three other columns:

1. **Predicted Syllable Breaks**
2. **Correct Syllable Breaks**
3. **Parser Result**

When you click on a word in the middle pane, the same fields are shown on the right where you can edit them. If the parse of the given word is successful, then a tree diagram of the parse is also shown as **Tree Diagram**. The  $\mathbb{W}$  constituent is the word and the  $\sigma$  constituent indicates a syllable. The constituents below the syllable are the natural classes used. Below these are the segment and then its grapheme.


Please note that until you either manually add words to the project or until you import a set of words, the list will be empty. See section 12.1 for how to import a set of words.

The **Predicted Syllable Breaks** column/field contains the result of the last time you ran the **Parse all Words** tool. When it is empty, it means either that the parser has never been invoked or that the parser failed to produce a result. If the parser failed to produce a result, the **Parser Result** column/field should have some kind of explanation for where the parser ran into a problem (this will be in red).

When the **Predicted Syllable Breaks** column/field contains a result, it shows how this word was syllabified the last time you ran the **Parse all Words** tool, given the state of the segments, natural classes, and syllable patterns when it was invoked. When there is a result in this column/field, then the **Parser Result** column/field should have “Success” (in green).

Remember that you can click on a column header to sort the contents of the column. If you click on the **Parser Result** header, then any error messages should show at the top. You might find this useful when figuring out how to set the set of segments, natural classes, and syllable patterns to get a desired result.

The **Correct Syllable Breaks** column/field contains what you have indicated to be the correct syllabification for this word. It may be filled in if you imported words from a **ParaText** hyphenatedWords.txt file and that file indicated that this word's hyphenation had been approved. See section 13.3 for how to clear the

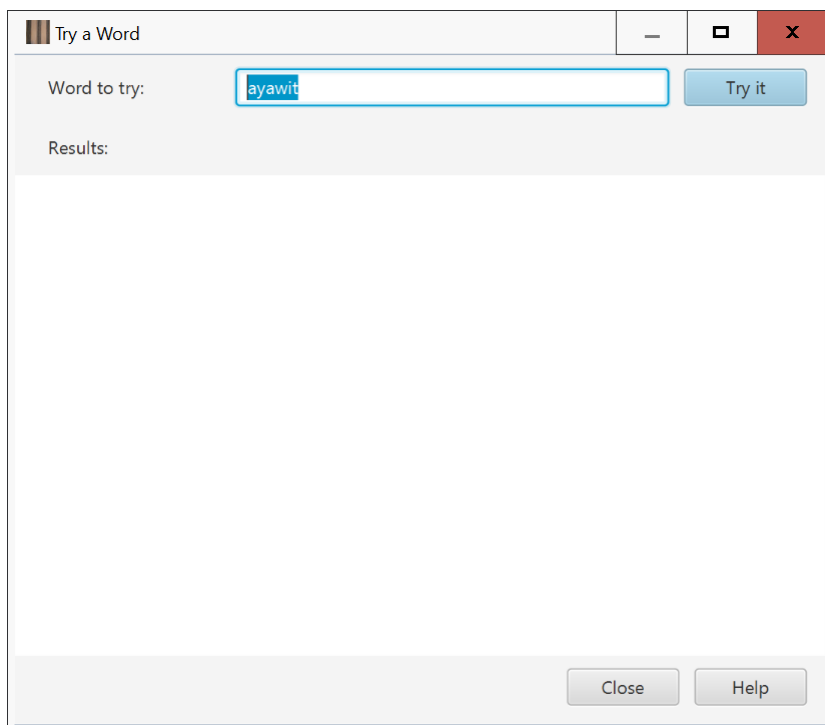
set of **Correct Syllable Breaks**. Otherwise, this column/field is expected to be blank until you either manually enter the correct syllabification or run the **Convert Predicted to Correct Syllabification** tool by using **Tools / Convert predicted to correct syllabification** menu item or clicking on the  tool bar button.

When you run the **Convert Predicted to Correct Syllabification** tool, it brings up a dialog box listing all the words which have a non-empty **Predicted Syllable Breaks** field. See section 10 for more on this.

#### 4.7 CV Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the **Parser / Try a Word** menu item. This brings up a dialog box like the one shown in example (4). (Note: unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it.) Further, the size and location of the **Try a Word** dialog box for the CV pattern approach, Sonority hierarchy approach, and the one for the Onset-nucleus-coda approach can be set independently; you can have several **Try a Word** dialog boxes open at the same time if you wish.)

(4)



You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **CV Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **CV Words** view).

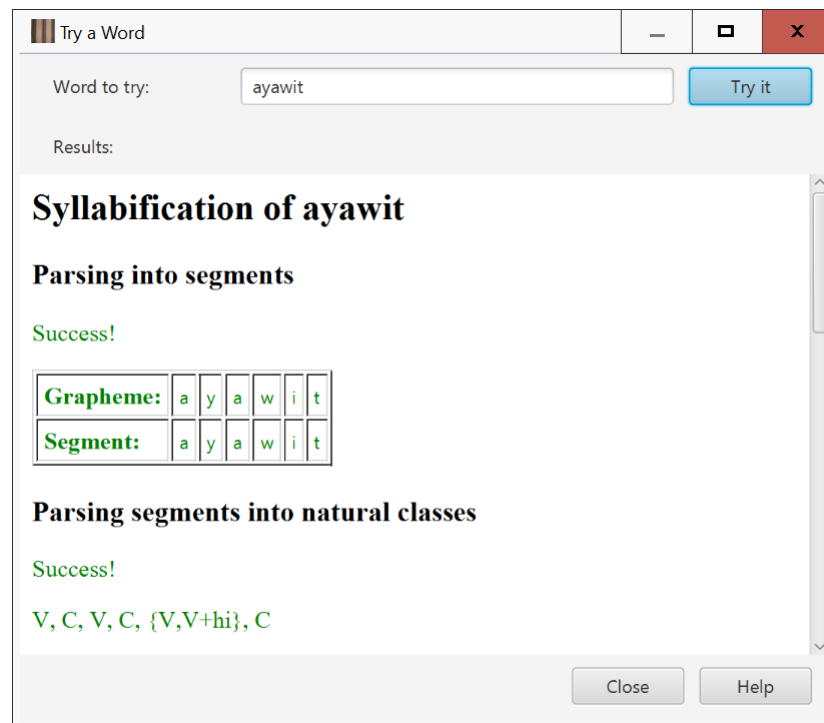
When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the CV pattern approach, there are at most three steps that are tried:

1. Parsing into segments.
2. Parsing segments into natural classes.
3. Parsing natural classes into syllables.

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into natural classes. If this succeeded, it shows the result of parsing natural classes into syllables. Successful steps are shown in **green** while unsuccessful ones are shown in **red**. An example of a successful parse is shown in example (5).

(5)



The parsing into segments part shows which grapheme was matched with what segment, Example (6) shows a case where the grapheme differs from the segment in shape.

(6)

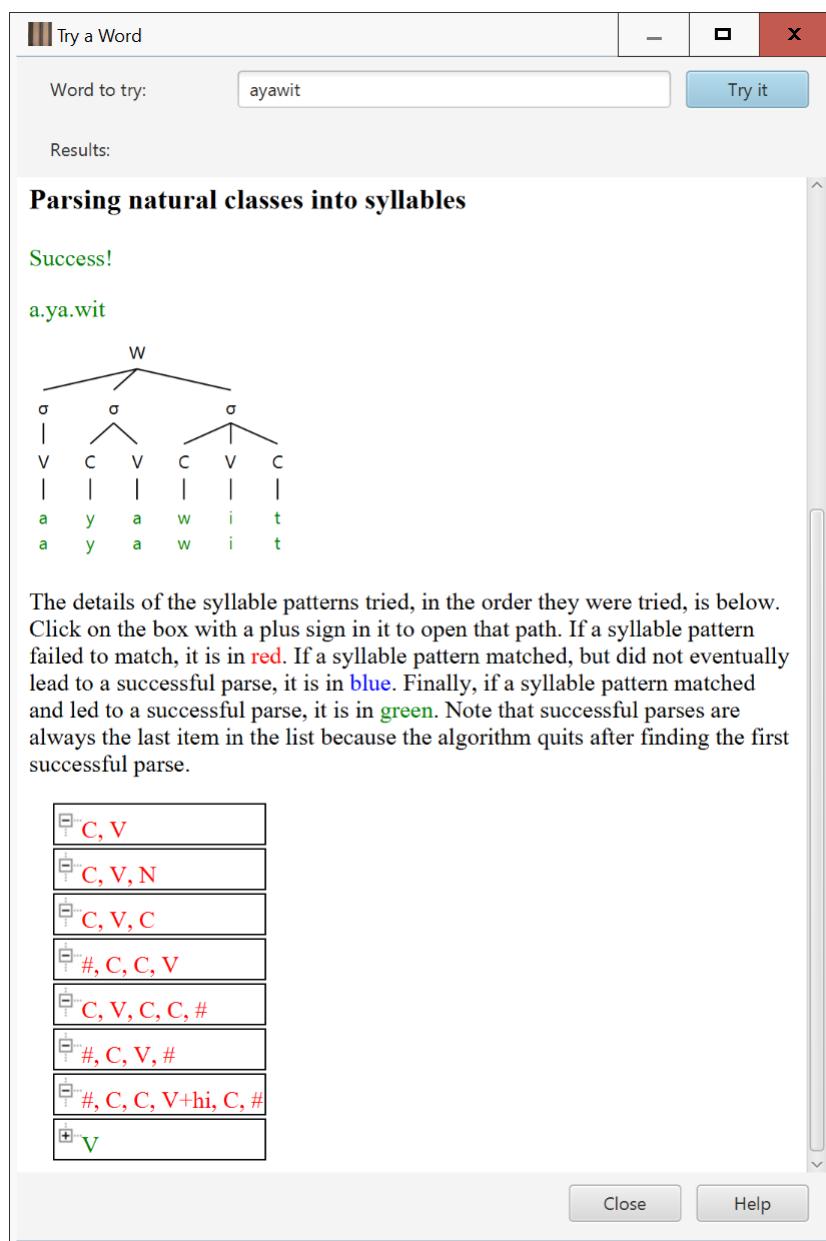
<b>Grapheme:</b>	a	a	ng	u
<b>Segment:</b>	a	a	n	u

The parsing segments into natural classes part shows which natural classes were matched. Sometimes a given segment may be in more than one natural class. In that case, the possible natural classes are shown inside of curly braces as shown in example (7).

(7) V, V, {C,N}, V

Scrolling the results window down shows the last part:

(8)



When the parse succeeds, it shows the syllabification as well as a tree diagram indicating the structure of the word parsed into its syllables. The W constituent is the word and the  $\sigma$  constituent indicates a syllable. The constituents below the syllable are the natural classes used. Below these are the segment and then its grapheme.

In the last part, it shows which syllable patterns were tried in the order they were tried. Successful ones are shown in **green** (and begin with a small box with a plus sign in it). These will always be the last one shown because **Asheninka** stops looking whenever it finds a successful match. Failed ones are shown in **red** (and begin

with a small box with a minus sign in it). Whenever a particular pattern matched the word at a particular point in the parsing process but it did not ultimately lead to a successful parse, it is shown in blue (and begins with a small box with a plus sign in it).

To examine a particular path, click on the small box with a plus sign in it. This will expand that path and show the next set of syllable patterns tried, in the order they were tried.

As mentioned above, whenever one of the three steps fails, there is an error message shown for that step (and no following steps will be tried).

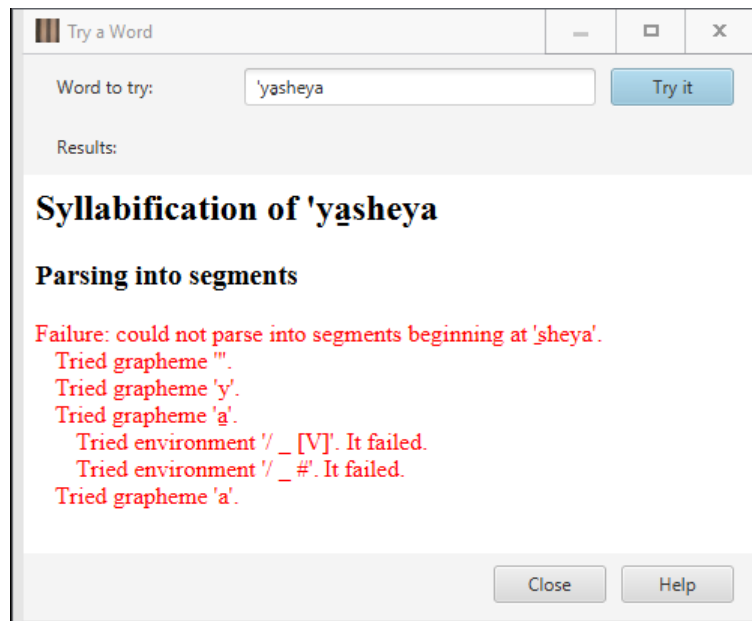
For example, if a word contains a grapheme that is not defined in any segment, when trying to parse the word into segments, it will show a message such as what is in example (9).

- (9) **Failure: could not parse into segments beginning at 'qap'.**

In this case, the word was **aanguqap** but the **q** was not defined in any segment. Note that it shows where the problem was initially found.

It will also show the steps it took in trying to parse the word into segments, but failed as in example (10).

(10)



In this case, the grapheme **a** has two environments: either before a vowel or word finally. Neither succeeded.

As another example, if a word contains a segment that is not in any natural class, it will display a message such as what is shown in example (11).

- (11) **Failure: could not parse into natural classes; did find classes 'V, C' covering graphemes 'am'.**

In this case, the word was **ambu**, but no natural class has the segment **b** in it. So when it tried to find a natural class for **b**, it failed. Notice that it does show what



was found. (If the segment is the first item in the word, the found classes will be " and the covered graphemes will be ".)

For the final step, if the sequence of natural classes could not be parsed into syllables per the ordered syllable patterns, then the following message is reported:

(12) **Failure: could not parse natural classes into syllables**

What follows this is an explanatory paragraph as noted above for example (8) and then the list of syllable patterns tried. In one project, this looked liked what is in example (13) for the word [yanjkuik](#).

(13)

K
C, V
C, V, C
V
V, C
K, C

Notice that none of the patterns are in green. Four are in red and two are in blue. Recall from above that this means that none of the red patterns matched the beginning of the sequence of natural classes (which was **C, V, C, C, K, C**). The two blue ones did match the beginning of the sequence, but did not ultimately lead to a successful parse. Example (14) shows what it looked like when opening the second blue pattern (**C, V, C**).

(14)

C, V, C
K
C, V
C, V, C
V
V, C
K, C

Notice that all the embedded patterns are in red: none of them matched the beginning of the natural class sequence after removing the initial **C, V, C** part that had matched (i.e., with **C, K, C**).

#### 4.8 Predicted vs. correct CV words


The **Predicted vs. Correct CV Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the predictions of the current set of segments, natural classes, and syllable patterns differ from the expected results. In this view, by the way, the predicted and correct words are aligned in pairs with

the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

## 5 Sonority hierarchy approach

In the Sonority hierarchy approach, you need to define the following items:

1. segments (section 5.1)
2. a sonority hierarchy via natural classes (section 5.2)
3. grapheme natural classes (section 5.3)
4. environments (section 5.4)

After that, you either use an existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the **Parser / Syllabify Words** menu item or click on the  tool bar button. This will apply the algorithm of the Sonority hierarchy approach to all the words. You can then see the results in the **Son. Hier. Words** view. See sections 5.5 and 5.6 for ideas on how to check the results, among other things.

The main “game” to play with the Sonority hierarchy approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), and the sonority hierarchy (i.e., its ordered set of natural classes) so that one gets most, if not all, words to syllabify correctly. Depending on the needs of the language you are modeling, you may find that this algorithm is less than you need as noted in the “Overview” document.

The main algorithm is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of “**Failure: could not parse into segments beginning at 'some characters'**” and quits. The *'some characters'* part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.<sup>5</sup> You can use the **Try a Word** tool to get more details (see section 5.6).

---

<sup>5</sup>This is the exact same algorithm used by the CV pattern approach.

2. If it succeeded in finding a sequence of segments, it then performs a left-to-right sweep of the segments it posited. It places the first segment of the word in the first syllable. It then tries to find the natural class of the current segment and the natural class of the following segment (unless the current segment is at the end of the word and also not already in a syllable; in this case, it adds the current segment to the current syllable). It then compares the “level” in the sonority hierarchy of these two natural classes and processes them as follows:
  - a. If they are at the same level, a new syllable is created and the second segment is added to this new syllable.
  - b. If the natural class of the current segment is less sonorous than the natural class of the following segment, then it adds the following segment to the current syllable.
  - c. If the natural class of the current segment is more sonorous than the natural class of the following segment, then it looks at the segment after the following segment. There are two cases considered:
    - i. If the second segment's natural class is the same or more sonorous than the third segment's natural class, then the second segment is added to the current syllable. A new syllable is created and the third segment is added to it.
    - ii. Otherwise, a new syllable is created and the second segment is added to it.
3. Whenever a segment is not found in any natural class, an error is reported.
4. If it succeeded in finding natural classes for all segments, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports “Success”.

We now give more information on the various views available in the user interface for the Sonority hierarchy approach.


## 5.1 SH segment inventory

The **Segment Inventory** view works exactly like the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details.

## 5.2 Sonority hierarchy

Use the **Sonority Hierarchy** view to create an ordered set of natural classes that constitute the sonority hierarchy you wish to use. More sonorous classes are above less sonorous classes.

The “Name” field is for you to define a short name for this natural class. We suggest using something short. The “Description” field is for your use to document anything more specific about this class.

The “Segments” field shows the set of segments this class consists of. It has a chooser button on the far right which looks like . Click on it to bring up a dialog box which lets you select the segments which constitute this class. Note that **Asheninka** currently does not check to see if a given segment has already been used in some other class. It is up to you to avoid using the same segment in

more than one class. If a given segment is in more than one class, the syllabification algorithm will use the topmost class containing that segment.

Finally, the order in which the classes occur is important. The items are in order of sonority: the most sonorous at top and the least sonorous at the bottom. You control the order of the classes by clicking on a class in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

### 5.3 Grapheme natural classes

This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.

### 5.4 Environments

This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

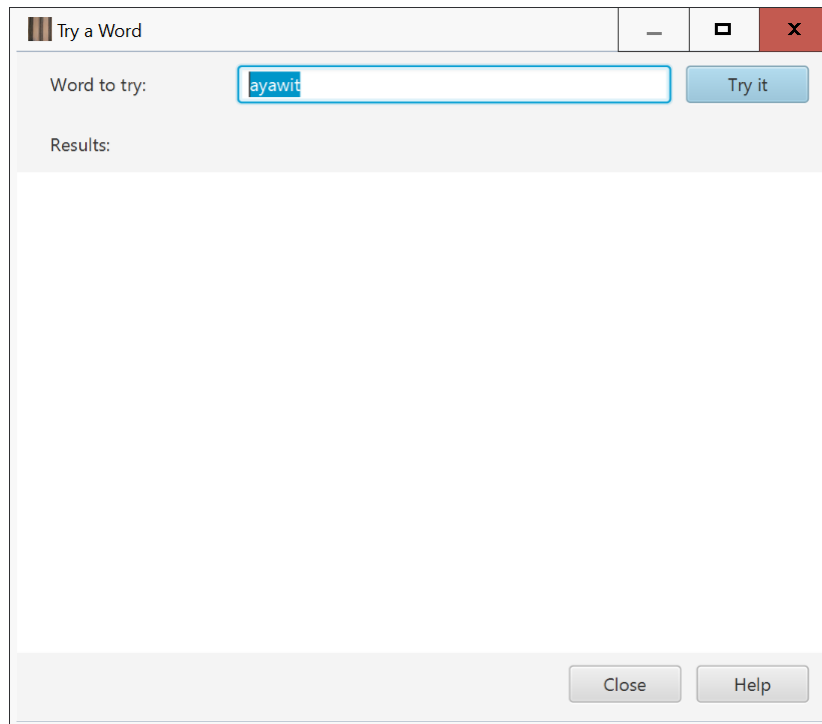
### 5.5 Sonority hierarchy words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Sonority hierarchy approach algorithm. In the tree diagram, there is no intervening structure between the syllable and the segment. See section 4.6 for more on this view.

### 5.6 Sonority hierarchy Try a Word

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the **Parser / Try a Word** menu item. This brings up a dialog box like the one shown in example (15). (Note: unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box for the CV pattern approach, Sonority hierarchy approach, and the one for the Onset-nucleus-coda approach can be set independently; you can have several **Try a Word** dialog boxes open at the same time if you wish.)

(15)



You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **Son. Hier. Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **Son. Hier. Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Sonority hierarchy approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via their natural classes).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in **green** while unsuccessful ones are shown in **red**. An example of a successful parse is shown in example (16).

(16)

**Try a Word**

Word to try:

Results:

### Syllabification of yawit

#### Parsing into segments

Success!

<b>Grapheme:</b>	a	y	a	w	i	t
<b>Segment:</b>	a	y	a	w	i	t

Success!

a.ya.wit

Close Help

The parsing into segments part is exactly the same as it is for the CV pattern approach, although the tree diagram is simpler.

Scrolling the results window down shows the last part:

(17)

**Try a Word**

Word to try:

Results:

The details of the segment to natural class mappings and the sonority comparison between pairs of natural classes is below. If a segment failed to map to a natural class, it is in **red**. Otherwise it is in **green**.

Segment 1	Relation	Segment 2	Starts Syllable
a (Vowels)	>	y (Glides)	σ
y (Glides)	<	a (Vowels)	σ
a (Vowels)	>	w (Glides)	
w (Glides)	<	i (Vowels)	σ
i (Vowels)	>	t (Obstruents)	

Close Help

In the last part, it shows the sequence of segment pairs that were found along with their respective natural class and relationship. The **Relation** column uses mathematical symbols to indicate the relationship as given in example (18).

(18)

Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

The **Starts Syllable** column shows a syllable symbol ( $\sigma$ ) when the algorithm began a new syllable.

As mentioned above, whenever one of the two steps fails, there is an error message shown for that step (and no following steps will be tried). See section 4.7 for examples where the segments could not be parsed.

If a word contains a segment that is not in any natural class, the sonority hierarchy results table will end with a row whose **Relation** column contains “!!!” in red and one of the offending segments will have “(No Natural Class)” after it as shown in example (19).

(19)

Segment 1	Relation	Segment 2	Starts Syllable
a (Vowels)	>	m (Nasals)	$\sigma$
m (Nasals)	!!!	b (No Natural Class)	$\sigma$

In this case, the word was **ambu**, but no natural class has the segment **b** in it. So when it tried to find a natural class for **b**, it failed

## 5.7 Predicted vs. correct Son. Hier. words

The **Predicted vs. Correct Son. Hier. Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the predictions of the current set of segments and sonority hierarchy differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

## 6 Onset-nucleus-coda approach

In the Onset-nucleus-coda approach, you need to define the following items:


1. segments (section 6.1)
2. a sonority hierarchy via natural classes (section 6.2)
3. syllabification parameters (section 6.3)



4. grapheme natural classes (section 6.9)
5. environments (section 6.10)

If needed to successfully parse your language, you may also need to define these items:

1. CV Natural Classes which are used in the next two (section section 6.4)
2. templates (section section 6.5)
3. filters section section 6.6)

After that, you either use a existing word list, import a word list (see section 12) or enter a list of words by hand. Then you use the **Parser / Syllabify Words** menu item or click on the  tool bar button. This will apply the algorithm of the Onset-nucleus-coda approach to all the words. You can then see the results in the **ONC Words** view. See sections 6.7 and 6.11 for ideas on how to check the results, among other things.

The main “game” to play with the Onset-nucleus-coda approach in **Asheninka** is to adjust the segment inventory (including graphemes and their environments), the sonority hierarchy (i.e., its ordered set of natural classes) and the syllabification parameters so that one gets most, if not all, words to syllabify correctly.

In rough terms, it tries to find all possible sequences of segments in the word. If it can do that, it then seeks to find sequences of onset, nucleus, and coda segments that can be placed in a syllable, modulo the parsing parameters. See appendix B for details.

We now give more information on the various views available in the user interface for the Onset-nucleus-coda approach.

### 6.1 ONC segment inventory

The **Segment Inventory** view is similar to the **Segment Inventory** view for the CV pattern approach. See section 4.1 for details. In addition to what the **Segment Inventory** view has, the **Segment Inventory** view also has three check boxes, one each for the following:

1. Can be in onset
2. Can be in nucleus
3. Can be in coda

Check the boxes for what positions the segment can be in.

### 6.2 Sonority hierarchy

The **Sonority Hierarchy** view works exactly like the **Sonority Hierarchy** view does for the Sonority hierarchy approach. See section 5.2 for details.

### 6.3 Syllabification parameters

The **Syllabification Parameters** view is where you set the parameters to control syllabification for your language. To read more about these parameters, you can click [here](#) or use the **Help / Introduction to syllabification** menu item.

The “Codas allowed” field is a check box. If codas are allowed, check the box. Otherwise, leave it unchecked.

The “Onset maximization” field is a check box. If onsets should be maximized, then check the box. Otherwise, leave it unchecked.

The “Onset principle” field has three radio buttons. One of them must be set. The three options are:

1. All but the first syllable must have an onset
2. Every syllable must have an onset
3. Onsets are not required

### 6.4 CV natural classes

The **CV Natural Classes** view is exactly the same as the **CV Natural Classes** view for the CV pattern approach. See section section 4.2 for details.

### 6.5 ONC templates

Templates can be used in two situations currently in **Asheninka**:

1. For a nucleus, it requires the set of slots to be present.
2. For all others, it provides a way to override **Sonority Hierarchy** violations.

When defining a template, you fill in the fields. The “Name” field is for you to remember this template.

Its “Description” field lets you define it more fully, perhaps including some example words as to why you added the template in the first place.

The “Type” field lets you select the constituent structure the template is for.

The “Slots” field is where you define the pattern for when the template will apply. You can use segment names or CV natural class names (see section 6.4) enclosed in square brackets (e.g., “[C]”). If the segment or the natural class can violate the **Sonority Hierarchy**, then key an asterisk (\*) before the segment name or the opening square bracket. If the item is optional, enclose it within parentheses.

Example (20) has some examples from an English project.

(20)	Type	Slots	Description
	Onset	*s [VoicelessNonCont] ([SonorantCV])	Allow /s/ in an onset which violates the SSP
	Word final	*[Coronal] (*[Coronal]) (*[Coronal])	Appendix to allow for <a href="#">siksθs</a> , etc.
	Word final	*[VoicelessNonCont] ([Coronal])	Word final appendix for /sp/ and /sk/

Finally, the order in which the templates occur is important. The items are in order of priority: the first to be tried (of a given type) is higher and the last to be tried (of that type) is lower. You control the order of the templates by clicking on a template in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

## 6.6 ONC filters

Filters can be used in four situations currently in [Asheninka](#):

1. onset,
2. nucleus,
3. coda, and
4. rime.

Filters come in two varieties: fail and repair. When a filter is matched and is marked as fail, then the syllable parsing stops at that point and seeks other possibilities, if any. When the filter matches and is marked as repair, it will seek to fix up the syllable constituents around it to fix the situation. For example, for English, a word such as [Atlantic](#) ([æt.ləntɪk](#)) will normally syllabify as [æt.læn.tɪk](#) but it needs to syllabify as [æt.læn.tɪk](#). By adding an onset repair filter that matches the [tl](#) sequence, one can obtain the correct result.

When defining a filter, you fill in the fields. The “Name” field is for you to remember this filter.

Its “Description” field lets you define it more fully, perhaps including some example words as to why you added the filter in the first place.

The “Scope” field lets you select the constituent structure the filter has scope over.

The “Action” radio buttons let you specify whether the filter will fail when matched or attempt a repair.

The “Slots” field is where you define the pattern for when the filter will apply. You can use segment names or CV natural class names (see section 6.4) enclosed in square brackets (e.g., “[C]”). If the segment or the natural class can violate the [Sonority Hierarchy](#), then key an asterisk (\*) before the segment name

or the opening square bracket. If the item is optional, enclose it within parentheses. If it is a repair filter, then key a vertical line (|) at the point where the repair will occur.

Example (21) has some examples from an English project. They are all repair filters.

(21)	Type	Slots	Description
	Onset	[Non-stridentCoronal]   1	Disallow non-strident coronal before labial in an onset. Example: ætlæntik ( <i>Atlantic</i> ) should be æt.læn.tik, not æ.tlæn.tik.
	Onset	[Stop]   [N]	Disallow stop/nasal in a onset. Example: æpniə ( <i>apnea</i> ) should be æp.ni.ə, not æ.pni.ə.
	Onset	[Labial]   [Labial]	Disallow two labials in an onset. Example: ʃapwoɪn ( <i>shopworn</i> ) should be ʃap.woɪn, not ʃa.pwoɪn.

Finally, the order in which the filters occur is important. The items are in order of priority: the first to be tried (of a given scope) is higher and the last to be tried (of that scope) is lower. You control the order of the filters by clicking on a filter in the middle pane and then using the up and down arrows to change its order. Note that the normal way of sorting by a column via clicking on a column header is disabled for this view.

## 6.7 ONC words

This works like the **CV Words** view for the CV pattern approach, except that the **Predicted Syllable Breaks** values, the **Parser Result** values, and the **Tree Diagram** are the result of the Onset-nucleus-coda approach algorithm. If the parse fails and some word structure was built, then this partial word structure is shown by **Tree Diagram**. In the tree diagram, a syllable may have an O (onset) and a required R (rime) constituent. A rime has an N (nucleus) and an optional C (coda) constituent. See section 4.6 for more on this view.

## 6.8 Predicted vs. correct ONC words

The **Predicted vs. Correct ONC Words** view shows any words which have both a predicted value and a correct value and, in addition, the two values differ. This is intended to give you a way to quickly see how the results of applying the Onset-nucleus-coda approach differ from the expected results. In this view the predicted and correct words are aligned in pairs with the predicted syllabification immediately above the correct syllabification. This is an attempt to make it easier to see the differences between the two.

## 6.9 Grapheme natural classes

This works exactly like the **Grapheme Natural Classes** view for the CV pattern approach. See section 4.4 for details.

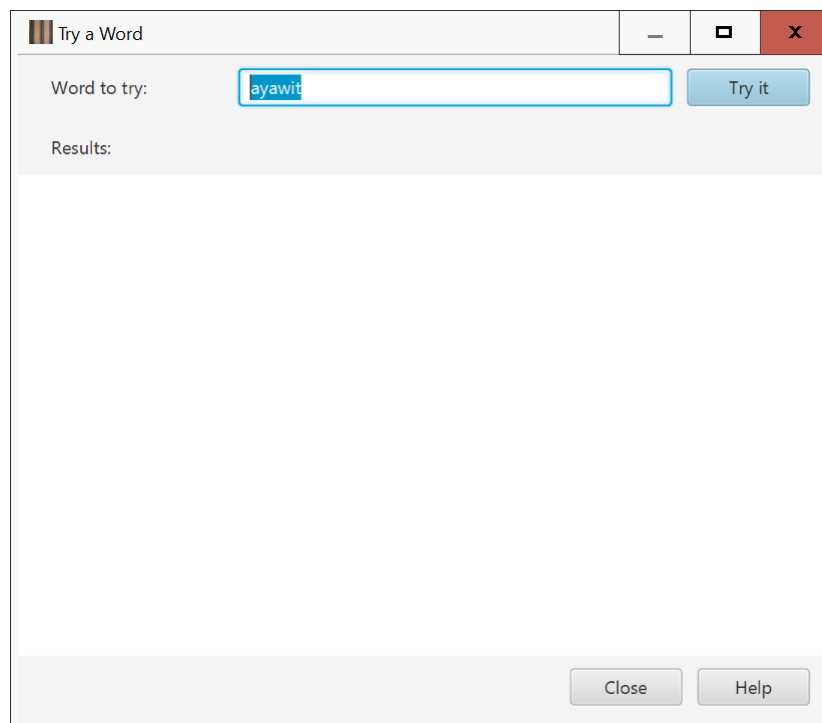
## 6.10 Environments

This works exactly like the **Environments** view for the CV pattern approach. See section 4.5 for details.

## 6.11 *ONC Try a Word*

There are times when it may be difficult to know just why a given word is not syllabifying correctly. That's where the **Try a Word** dialog box can be useful. To see it, use the **Parser / Try a Word** menu item. This brings up a dialog box like the one shown in example (22). (Note: unlike most dialog boxes, you can keep the **Try a Word** dialog box open while still editing other views in the main window. Unfortunately, the minimize button does not currently work so you cannot minimize the dialog, but you can drag it around and resize it. Further, the size and location of the **Try a Word** dialog box for the CV pattern approach, Sonority hierarchy approach, and the one for the Onset-nucleus-coda approach can be set independently; you can have several **Try a Word** dialog boxes open at the same time if you wish.)

(22)



You can key a word to try in the text box. By default, **Asheninka** shows the current word selected in the **ONC Words** view (if you are showing that view) or the last word you used (or nothing if you've never used **Try a Word** before and are not showing the **ONC Words** view).

When you either press the **Enter** key or click on the **Try it** button, **Asheninka** will try to parse the word in the text box and report the result in the Results portion of the dialog box.

For the Onset-nucleus-coda approach, there are at most two steps that are tried:

1. Parsing into segments.
2. Parsing segments into syllables (via the onset-nucleus-coda algorithm; see section 6 above).

The results portion always shows the parsing into segments. If this succeeded, then it also shows the parsing of segments into syllables. Successful steps are shown in **green** while unsuccessful ones are shown in **red**. An example of a successful parse is shown in example (23).

(23)

The screenshot shows a window titled "Try a Word" with a text input field containing "ayawit" and a "Try it" button. Below the input field, the "Results:" section displays the following information:

**Syllabification of yawawit**

**Parsing into segments**

Success!

<b>Grapheme:</b>	a	y	a	w	i	t
<b>Segment:</b>	a	y	a	w	i	t

Success!

a.ya.wit

A constituent structure tree diagram is shown below the text. The root node is **W** (word). It branches into three **σ** (syllable) nodes. The first **σ** branches into **R** (rime), which branches into **N** (nucleus), which branches into the segment **a**. The second **σ** branches into **O** (onset), which branches into the segment **y**, and **R** (rime), which branches into **N** (nucleus), which branches into the segment **a**. The third **σ** branches into **O** (onset), which branches into the segment **w**, and **R** (rime), which branches into **N** (nucleus), which branches into the segment **i**, and **C** (coda), which branches into the segment **t**. The segments **a**, **y**, **a**, **w**, **i**, and **t** are shown in green, indicating a successful parse.

Close Help

The parsing into segments part is exactly the same as it is for the CV pattern approach. If any constituent structure was built during the parse, the tree diagram is also shown. The **W** constituent is the word and the **σ** constituent indicates a syllable. A syllable may have an **O** (onset) and a required **R** (rime) constituent. A rime has an **N** (nucleus) and an optional **C** (coda) constituent. Below these are the segment and then its grapheme. If the parse succeeded, then the segments and

their graphemes are shown in green. If the parse failed but some of the tree was built, then the segments and their graphemes are shown in red.

Scrolling the results window down shows the last part:

(24)

Word to try:

Results:

The details of the segment to natural class mappings, the sonority comparison between pairs of natural classes, and the parsing of onset/nucleus/coda is below. If a step failed, it is in **red**. Otherwise it is in **green**.

Segment 1	Relation	Segment 2	Type	Status
a (Vowels)	>	y (Glides)		Tried segment 1 as an onset, but it was not an onset.
a (Vowels)	>	y (Glides)	nucleus	Added segment 1 as a nucleus.
y (Glides)	<	a (Vowels)	nucleus or coda	Expected segment 1 to be either a nucleus or a coda, but it was not a nucleus and was not a coda; so started a new syllable.
y (Glides)	<	a (Vowels)	onset	Added segment 1 as an onset.
a (Vowels)	>	w (Glides)	nucleus	Added segment 1 as a nucleus.
w (Glides)	<	i (Vowels)	nucleus or coda	Expected segment 1 to be either a nucleus or a coda, but it was not a nucleus and was not a coda; so started a new syllable.
w (Glides)	<	i (Vowels)	onset	Added segment 1 as an onset.
i (Vowels)	>	t (Obstruents)	nucleus	Added segment 1 as a nucleus.
t (Obstruents)	>	— (—)	coda	Added segment 1 as a coda and started a new syllable.

In the last part, it shows the sequence of segment pairs that were found along with their respective natural class and relationship. The **Relation** column uses mathematical symbols to indicate the relationship as given in example (25).

(25)

Relation	Meaning
<	Segment 1 is less sonorous than segment 2.
=	Segment 1 is equal in sonority to segment 2.
>	Segment 1 is more sonorous than segment 2.
!!!	The segment was not in any natural class.

The **Type** column shows the type (or state) the parser is using.

The **Status** column shows what the algorithm did at that part of the processing.

As mentioned above, whenever one of the two steps fails, there is an error message shown for that step (and no following steps will be tried). See section 4.7 for examples where the segments could not be parsed.



If a word contains a segment that is not in any natural class, the onset-nucleus-coda results table will end with a row whose **Relation** column contains “!!!” in red and one of the offending segments will have “(No Natural Class)” after it as shown in example (26).

(26)

Segment 1	Relation	Segment 2	Type	Status
t (Obstruents)	!!!	u (No Natural Class)		Tried segment 1 as an onset, but the sonority hierarchy blocks it as an onset.
t (Obstruents)	!!!	u (No Natural Class)		Expected segment 1 to be a nucleus but it was not.

In this case, the word was **tun**, but no natural class has the segment **u** in it. So when it tried to find a natural class for **u**, it failed

## 7 Moraic approach

This approach has not been implemented yet.


## 8 Nuclear projection approach

This approach has not been implemented yet.

## 9 Optimality theory approach

This approach has not been implemented yet.

## 10 Converting predicted syllabification to correct syllabification

When you run the **Convert Predicted to Correct Syllabification** tool either by using the **Tools / Convert predicted to correct syllabification** menu item or by clicking on the  tool bar button, it brings up a dialog box listing all the words which have a non-empty **Predicted Syllable Breaks** field. You can select which words should be converted by manually checking the box before them<sup>6</sup> or by clicking on the checkbox in the header row and choosing to “Select All”, “Clear All”, or “Toggle”:

1. “Select All” checks every word.
2. “Clear All” unchecks every word.
3. “Toggle” makes every word that is checked be unchecked and every word that is unchecked be checked.

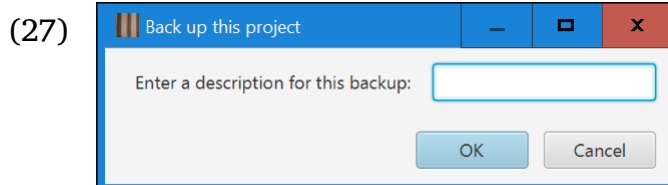
---

<sup>6</sup>To check the box, either click in it or click in the row and press the space bar.

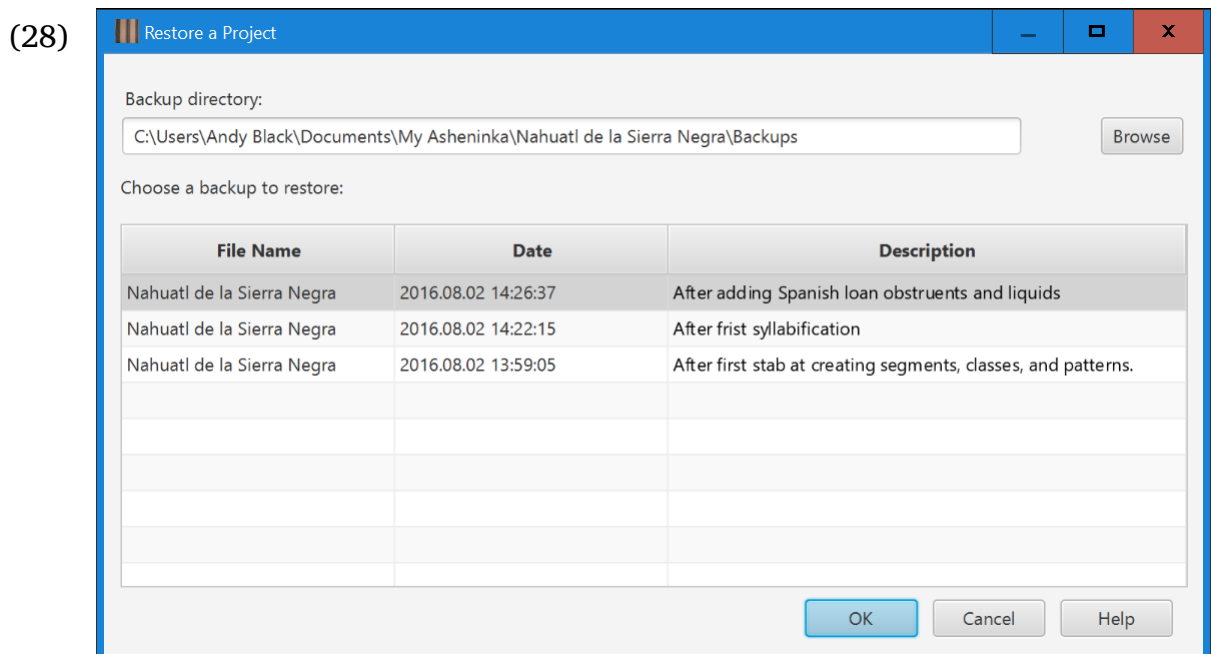
When you click on the OK button, **Asheninka** will copy the value in the **Predicted Syllable Breaks** field of every word that is checked to the corresponding **Correct Syllable Breaks** field.

## 11 Project management

Feel free to make frequent (labeled) backups via the **File / Project Management / Back up this project** menu item. It will bring up a dialog box like the one shown in example (27). Please give it a descriptive label so you can later identify the state of your project when you made the backup. (You can key language data if you wish.) You won't see the label on the file name, though. Rather, you'll see it when you go to the **File / Project Management / Restore a project** menu item.



When you restore a project, you will see a dialog box something like the one shown in example (28).



The backups you have made for this project will be shown in the list. The content in the Description column is the description you keyed while making the backup.

The idea here is that you can make a backup, try something else (e.g., add a segment or class; or re-order the syllable patterns or add a new one), and see how it goes. If it's worse, just restore to the previous state. If it's better, make a backup of that and go on.

## 12 Importing and exporting words

When using **Asheninka**, of course, you need words to parse. Once they are parsed and you have set the correct syllabifications for them (see section 10), you may want to export the set of words to be used by a typesetting tool. Such a tool can use the syllabified words as a list of words with discretionary hyphens so that it can know when to hyphenate long words at ends of lines.

### 12.1 Import words

**Asheninka** allows you to import a list of words in the following forms:

1. A text file with one word per line
2. A word list exported from **ParaText**
3. The hyphenatedWords.txt file from **ParaText**
4. A word list exported from **FieldWorks Language Explorer** (aka FLEEx) (in tab-delimited form)

You can use any or all of these to add words to **Asheninka**'s list of words. A given word form will only be inserted once. By “word form” we mean that word pairs such as **achto** and **Achto** will be considered different since their capitalization is distinct.

To import a word list, use **File / Import Words** menu item and then choose the type of import to use. In each case, a standard file open dialog will appear. Find the file that has the format you need, choose it, and click on “Open”. Depending on the number of words in the imported file, this may take a while to complete.

If you are working on a word list found in **ParaText**, we recommend using the hyphenatedWords.txt file method. This file includes both upper and lower case forms of words and also will include any hyphenation the **ParaText** user has manually approved. At least with some versions of **ParaText**, this file can be found in the “My Paratext Projects” folder and then in the folder of your project name.

Newly added words will appear in the **CV Words** view with a **Parser Result** showing “**Untested**”. This means that the parser has not yet been run on the word.

### 12.2 Export words

When you want to export the word list in **Asheninka** to some kind of typesetting tool, use the **File / Export Hyphenated Words** menu item. The list of hyphenated words can be exported in three formats:

1. Export for **InDesign** (simple list)
2. Export for **ParaText** (hyphenatedWords.txt)
3. Export for **XLingPaper** (hyphenations exception file)

**Asheninka** exports hyphenated words in the following way:

1. When the word has a value in **Correct Syllable Breaks**, that value is used.

2. If **Correct Syllable Breaks** is empty, then if there is a value in **Predicted Syllable Breaks**, it uses that value.
3. Otherwise, it does not export the word.


It also will replace the periods used to demarcate syllables with whatever you have set as the discretionary hyphen character(s) in the hyphenation parameters (see section 2.6). By default, **Asheninka** uses an equals sign for both **ParaText** and **InDesign** and a hyphen for the **XLingPaper** output.

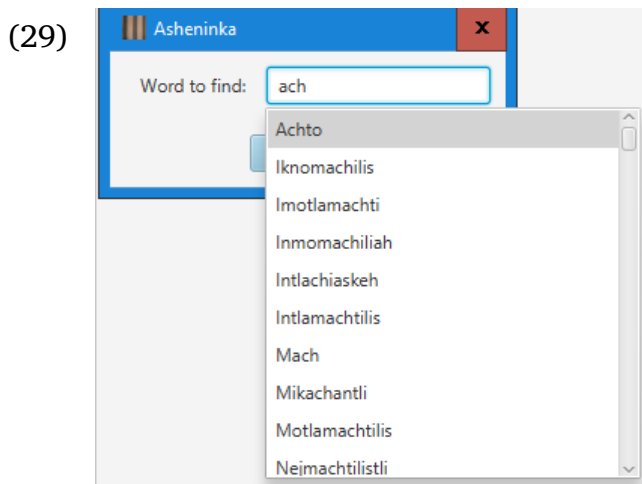
Further, **Asheninka** will not insert a discretionary hyphen if it appears too close to the front or too close to the end of a word, depending on the hyphenation parameters set. By default, the **InDesign** export uses zero characters from the front and zero characters from the end (i.e., every potential discretionary hyphen position is used).<sup>7</sup> By default, both the **ParaText** and **XLingPaper** export methods limit discretionary hyphens from two characters from front and two characters from the back.

## 13 Other tools

**Asheninka** also offers three other tools not previously covered.

### 13.1 Find a word

Use the **Tools / Find Word** menu item or key **Ctrl F** or click on the  button on the toolbar to find a particular word. This brings up a dialog box which allows you to start typing the word you are looking for. As you type, it has a drop down area showing all words which contain the sequence of characters you have typed. Note that it shows all words which have this sequence anywhere in them, not just at the beginning. For example, with one project when I typed “**ach**”, what is in example (29) showed.



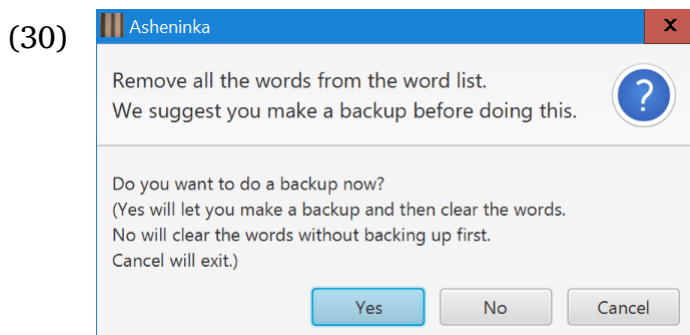
<sup>7</sup>This is because the **InDesign** program has its own way of letting the typesetter control where discretionary hyphens will be used.

This can be very useful not only for finding a particular word, but also for looking for particular sequences of characters.

When you press the **Enter** key or click on the **OK** button, **Asheninka** will show the selected word in the approach you last used: either via the **CV Words** view of the CV pattern approach or the **Son. Hier. Words** view of the Sonority hierarchy approach.

### 13.2 Remove all words

Use the **Tools / Remove all words** menu item to completely clear the list of words in both the **CV Words** view and the **Son. Hier. Words** view. When you invoke this menu item, it will show the dialog box shown in example (30).



When you click on the **Yes** button, **Asheninka** will then show you the backup dialog box (see section 11). When you have done the backup, **Asheninka** will immediately remove all the words from the list of words.

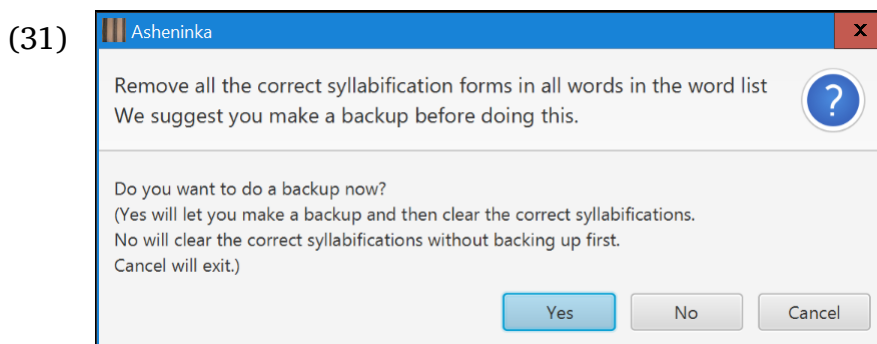
If, instead, you click on the **No** button, **Asheninka** will immediately remove all the words from the list of words.

Note that when the words are removed, it will happen whether or not the **CV Words** view or the **Son. Hier. Words** view is currently being shown. It usually happens very quickly.

If you click on the **Cancel** button (or close the dialog box using the X in the red area), the dialog box will exit and nothing will change.

### 13.3 Remove correct syllable breaks in all words

Use the **Tools / Remove correct syllable breaks in all words** menu item to completely clear the **Correct Syllable Breaks** column in both the **CV Words** view and the **Son. Hier. Words** view. When you invoke this menu item, it will show the dialog box shown in example (31).



Like the dialog box for removing all words in section 13.2, you are asked whether or not to make a backup. When you click on the **Yes** button, **Asheninka** will then show you the backup dialog box (see section 11). When you have done the backup, **Asheninka** will immediately remove all the **Correct Syllable Breaks** forms from the list of words.

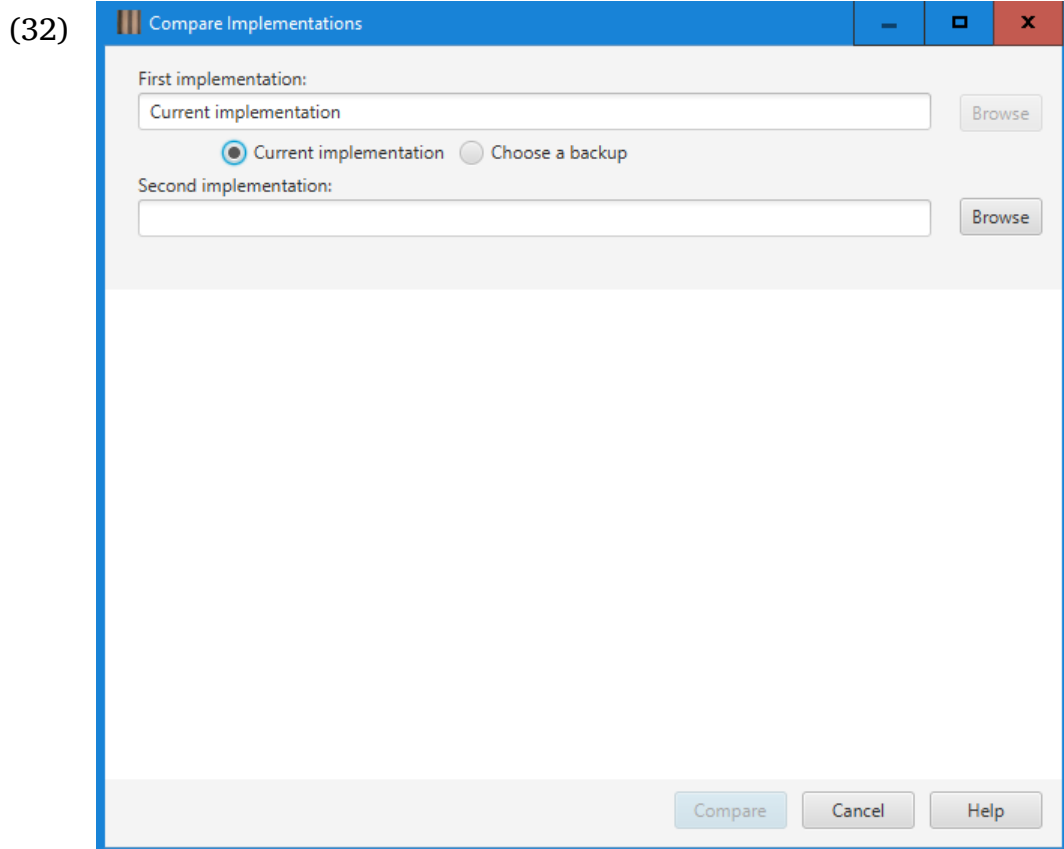
If, instead, you click on the **No** button, **Asheninka** will immediately remove all the **Correct Syllable Breaks** forms from the list of words.

Note that when the **Correct Syllable Breaks** forms are removed, it will happen whether or not the **CV Words** view or the **Son. Hier. Words** view is currently being shown. It usually happens very quickly.

If you click on the **Cancel** button (or close the dialog box using the X in the red area), the dialog box will exit and nothing will change.

### 13.4 Compare two implementations

**Asheninka** offers a way to more easily see what some changes to an approach have on how the words are syllabified. Use the **Tools / Compare Implementations** menu item to do this. It brings up a dialog box which looks like what is in example (32).



You can compare the current implementation (i.e., what is showing in the user interface right now) with what is in a backup file (see section 11) or you can

compare two different backups. As you can see in example (32), the current implementation is set by default. To compare two backups, click on the “Choose a backup” radio button.

Whenever you need to select a backup file, click on the “Browse” button. This will bring up the “Restore a Project” dialog. Choose the backup you want and click OK.

When you have chosen the two implementations to compare, click on the “Compare” button. Depending on how many differences there are, the comparison process can take quite a long time so please be patient. When it is done, a report is shown in the bottom part of the dialog. The dialog box will stay visible until you close it (e.g., by clicking on the “Cancel” button). This lets you examine the report while still being able to make changes to the data in the user interface.

Which approach is used in the comparison is the current one being shown.

#### ***13.4.1 CV pattern approach comparison***

When I did this for one project using the CV pattern approach, I got what is shown in example (33).



(33)

**Compare Implementations**

First implementation:

☒ Current implementation ☐ Choose a backup

Second implementation:

**CV Approach Comparison between:**

1. Current implementation
2. C:\Users\Andy Black\Documents\My Asheninka\Nahuatl de la Sierra Negra\Backups\Nahuatl de la Sierra Negra\220160802-151207.ashebackup (After adding kua, kue, kui as V (work-around for kw))

Performed on November 11, 2016 11:38 AM

**Segment Inventory**

Both have the same segment inventory.

**Natural Classes**

The following natural classes differ between the two.

First	Second
K (kua, kue, kui)	
V (a, aa, e, ee, i, ii, o, oo, u)	V (a, aa, e, ee, i, ii, kua, kue, kui, o, oo, u)

**Syllable Patterns**

The following syllable patterns differ between the two.

First	Second
K (K)	
KC (K, C)	

The order of the syllable patterns is shown below.

First	Second
K (K)	CV (C, V)
CV (C, V)	CVC (C, V, C)

### 13.4.2 Sonority hierarchy approach comparison

When I did this for one project while showing the Sonority hierarchy approach, I got what is shown in example (34).

(34)

Compare Implementations

First implementation:

Current implementation

Browse

☒ Current implementation
 ☐ Choose a backup

Second implementation:

C:\Users\Andy Black\Documents\My Asheninka\SonorityHierarchy\Backups\SHtsting20190129-132625

Browse

## Sonority Hierarchy Approach Comparison between:

- Current implementation
- C:/Users/Andy Black/Documents/My Asheninka/SonorityHierarchy/Backups/SHtsting20190129-132625.ashebackup (Standard)

Performed on January 29, 2019 2:13 PM

### Segment Inventory

Both have the same segment inventory.

### Grapheme Natural Classes

Both have the same set of grapheme natural classes.

### Environments

Both have the same set of environments.

### Sonority Hierarchy

Both have the same set of natural classes in the sonority hierarchy.

The order of the sonority hierarchy is shown below.

First	Second
Vowels (a, e, i, o, u)	Vowels (a, e, i, o, u)
Glides (w, y)	Glides (w, y)
Nasals (m, n, ɲ)	Liquids (l, r)
Liquids (l, r)	Nasals (m, n, ɲ)
Obstruents (ch, d, f, g, h, k, p, s, sh, t, v, x, z)	Obstruents (ch, d, f, g, h, k, p, s, sh, t, v, x, z)
Voiced Obstruents (b, d, g, z)	Voiced Obstruents (b, d, g, z)

### Words

The following words differ between the two.

First	Second	Syllabification
ahwiyalme		a.hwi.ya.lme
	ahwiyalme	a.hwi.yal.me

Compare

Cancel

Help

**13.4.3 Onset-nucleus-coda approach comparison**

When I did this for one project while showing the Onset-nucleus-coda approach, I got what is shown in example (35).

(35)

Compare Implementations

First implementation:

Current implementation

Browse

☒ Current implementation
 ☐ Choose a backup

Second implementation:

C:\Users\Andy Black\Documents\My Asheninka\ONC\Backups\ONCTesting20190823-103204.a

Browse

## ONC Approach Comparison between:

- Current implementation
- C:/Users/Andy Black/Documents/My Asheninka/ONC/Backups/ONCTesting20190823-103204.ashebackup (All but first has onset)

Performed on August 23, 2019 10:32 AM

### Segment Inventory

Both have the same segment inventory.

### Grapheme Natural Classes

Both have the same set of grapheme natural classes.

### Environments

Both have the same set of environments.

### Sonority Hierarchy

Both have the same set of natural classes in the sonority hierarchy.

### Syllabification Parameters

Codas allowed is the same.

Onset maximization is the same.

Onset principle differs.

First	Second
Onsets are not required	All but the first syllable must have an onset

### Words

The following words differ between the two.

First	Second	Syllabification
ababrastro	ababrastro	a.bab.ra.stro
		a.bab.ra.stro
abida	abida	a.bi.da
		a.bi.da

Compare

Cancel

Help

### 13.5 Compare syllabification results between approaches

**Asheninka** also offers a way to more easily see how the words are syllabified between the approaches. Use the **Tools / Compare Syllabification Results between Approaches** menu item to do this. This immediately does a comparison of the predicted syllabification of all words and shows the result in a dialog box. In one implementation I did, it looked like what is in example (36).

(36)

Compare Syllabification Results between Approaches

☒ Use CV approach
 ☐ Use moraic approach

☒ Use SH approach
 ☐ Use nuclear projection approach

☒ Use ONC approach
 ☐ Use OT Approach

### Comparison of Approaches to Syllabification

Performed on August 23, 2019 11:35 AM

#### Syllabifications

The following words have different syllabifications.

CV = CV Pattern  
 SH = Sonority Hierarchy  
 ONC = Onset-Nucleus-Coda

a	CV	failed
	SH	a
	ONC	a
ababrastro	CV	failed
	SH	a.ba.bras.tro
	ONC	a.bab.ra.stro
	CV	failed

Compare Cancel Help

Note that you can choose which approaches to use in the comparison. In the example, all currently implemented approaches were chosen. The result shows only those words which had some difference between the parses. For each such difference, it shows each word with the result of how each chosen approach parsed it. In the case above, no vowel initial words were allowed in the CV pattern approach so that's why there were so many failures. Scrolling down more in the dialog, there was the following portion, which better illustrates the differences:

(37)

mehetabel	CV	me.he.ta.bel
	SH	me.he.ta.bel
	ONC	failed
mehmelaptik	CV	meh.me.lap.tik
	SH	failed
	ONC	failed
mehmelawak	CV	meh.me.la.wak
	SH	me.hme.la.wak
	ONC	me.hme.la.wak
mehwante	CV	meh.wan.te
	SH	me.hwan.te
	ONC	failed
mehwati	CV	meh.wa.ti
	SH	me.hwa.ti
	ONC	me.hwa.ti

Like several other dialog boxes in **Asheninka**, you can keep this dialog box open while doing other things.

### A. Why is it called **Asheninka**?

This syllable parsing program is called **Asheninka** for historic reasons. (The word is pronounced a.<sup>1</sup>ʃɛ.nɪŋ.ka.)

In late 1983 my family and I were living in the jungles of Peru and David Payne came and asked me if I would create a **Consistent Changes** table for him that would insert discretionary hyphens in Asheninka text.<sup>8</sup> The algorithm he suggested was the CV Patterns approach (see section 4). Asheninka has long words and typesetting material in that language would improve readability with such a table. In 1984 I wrote such a table.

While it was functional, it ran slowly. I then wrote the **Hyphen** program (Black et al. 1987) to improve the efficiency. It implemented the same basic approach. Amazingly, the **Hyphen** program is still being used (albeit occasionally) today.

Because of this beginning, I chose to call this tool **Asheninka**. The program icon (shown in example (38) below) is the kind of material used for clothing by Asheninka people.<sup>9</sup>

<sup>8</sup>A more current version of this program is SIL International (2005).

<sup>9</sup>This image was gratefully taken from [here](#) on 12 November, 2015.

(38)



## B. ONC algorithm in detail

The main algorithm for the Onset-nucleus-coda approach is as follows:

1. It does a left-to-right sweep of the word, trying to find the sequence of segments which covers the whole word. It uses the graphemes defined in the segment information to determine a match with a segment. It tries all graphemes which match at the current point it is looking at in the left-to-right sweep. It tries the longest match first, but also applies any environments associated with the grapheme. See section 4.1 for more on this. If it cannot find a sequence that covers the whole word, it reports an error of “**Failure: could not parse into segments beginning at 'some characters'**” and quits. The ‘*some characters*’ part indicates the place in the word where it could not find a character or character sequence that matched any graphemes in any of the segments. Most likely, either there was a typo in the word or a grapheme is missing from some segment or its environment was not met.<sup>10</sup> You can use the **Try a Word** tool to get more details (see section 6.11).
2. If it succeeded in finding a sequence of segments, it then looks at the first segment. If it is not an onset and the Onset Principle is set to *All But First Has Onset*, then it reports an error of “**Onsets are required but segment 1 was not an onset.**” and quits.
3. Otherwise, it performs a left-to-right sweep of the segments. For each segment, it compares the sonority of this onset to the sonority of the following segment. What happens next depends on the onset, nucleus, or coda status of the segment.
4. it looks at the classification of the segment (onset, nucleus, and/or coda).
  - a. If it can be an onset, it compares the sonority of this onset to the sonority of the following segment. If the following segment's sonority is less than or equal to this segment's sonority, this segment is added to the current syllable.
  - b. Otherwise, it compares the sonority of the segment to the sonority of the following segment. It also assigns a type (or state) to the segment's position, depending on what the last segment was. The types are:

<sup>10</sup>This is the exact same algorithm used by the CV pattern approach.

- i. Onset
- ii. Onset or nucleus
- iii. Nucleus
- iv. Nucleus or coda
- v. Coda
- vi. Coda or onset
- vii. Unknown

The initial type is set to unknown.

c. Here is what it does for each type:

- i. Onset:
  1. If the segment can be an onset and its sonority is less than the sonority of the following segment, it adds the segment to the syllable as an onset. The type is set to be an onset or a nucleus.
  2. Otherwise, it sets the type to be a nucleus.
- ii. Onset or nucleus:
  1. If the segment can be an onset and its sonority is less than the sonority of the following segment, it adds the segment to the syllable as an onset. The type is set to be an onset or a nucleus.
  2. Otherwise, if the segment can be a nucleus, it adds it to the syllable as a nucleus. The type is set to be a nucleus or a coda.
  3. Otherwise, the parsing quits as a failure.
- iii. Nucleus:
  1. If the segment can be a nucleus, it is added to the syllable as a nucleus. The type is set to be a nucleus or a coda.
  2. Otherwise, the parsing quits as a failure.
- iv. Nucleus or coda:
  1. If the segment can be a nucleus, it is added to the syllable as a nucleus. The type is set to be a nucleus or a coda.
  2. Otherwise, if the segment can be a coda and the Coda Allowed parameter is set, then
    - a. If the segment can also be an onset and it is less sonorous than the following segment:
      - i. If the Onset Maximization parameter is set, the syllable is added to the word and new a syllable is created. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise the type is set to onset.
      - ii. Otherwise, if the following segment is not an onset and the Onset Principle is not set to *Onsets Not Required*, the syllable is added to the word and new a syllable is created. The type is set to onset or nucleus.
      - iii. Otherwise, the segment is added to the syllable as a coda and a new syllable is started. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise it is set to onset.
    - b. Otherwise:



- i. The segment is added to the syllable as a coda and a new syllable is started. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise the type is set to onset.
- 3. Otherwise, the syllable is added to the word, a new syllable is created and the type is set to be an onset.
- v. Coda or onset:
  - 1. If the segment can be a coda, the Coda Allowed parameter is set, and the segment is more sonorous than the following segment, it adds the segment to the syllable as a coda. The type is set to be a coda or an onset.
  - 2. Otherwise, the segment is added to the syllable as a coda and a new syllable is started. The type is set to onset or nucleus if the Onset Principle is set to *Onsets Not Required*; otherwise the type is set to onset.
- vi. Coda: (This never happens actually.)
- 5. If it succeeded in processing all the segments in the word, it outputs the syllabification into the **Predicted Syllable Breaks** field and reports “Success”.

If you have defined any filters or templates, then these also are applied during this process. In particular, filters are only applied for onset, nucleus, coda, and rime constituents.

Filters come in two varieties: fail and repair. When a filter is matched and is marked as fail, then the syllable parsing stops at that point and seeks other possibilities, if any. When the filter matches and is marked as repair, it will seek to fix up the syllable constituents around it to fix the situation. For example, for English, a word such as *Atlantic* (ætlæntɪk) will normally syllabify as æ.tlæn.tɪk but it needs to syllabify as æt.læn.tɪk. By adding an onset repair filter that matches the *tl* sequence, one can obtain the correct result.

Templates can be applied to onset, nucleus, coda, word initial, and word final constituents. For nucleus, the set of slots must match. For the others, the set of slots can override the **Sonority Hierarchy**.

## References

- Black, H. Andrew, Fred Kuhl, Kathy Kuhl, and David J. Weber. 1987. *Document preparation aids for non-major languages*. Occasional publications in academic computing Issue 7. Dallas, TX: Summer Institute of Linguistics. <http://www.sil.org/resources/publications/entry/29601> (accessed 24 Feb. 2017)
- IPA. 2016. International Phonetic Association. <https://www.internationalphoneticassociation.org/> (accessed 9 November 2016)
- SIL International. 2005. Consistent Changes Program. [http://scripts.sil.org/cms/scripts/page.php?site\\_id=nrsi&id=cc-grkuni](http://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=cc-grkuni) (accessed 24 Feb. 2017)

## Index

- Active field 3
- Add data item 3
- Analysis language, *see* Language, Analysis.
- Compare approaches 41
- Compare two implementations 35
- CV pattern approach 5
  - Algorithm 5
  - Compare two implementations 36
  - Environments 9
  - Grapheme natural classes 8
  - Graphemes 7
  - Natural classes 7
  - Predicted vs. correct syllabification 16
  - Segments 6
  - Syllable Patterns 8
  - Try a Word 12
  - Words 11
- Delete data item 3
- Discretionary hyphen character(s), *see* Hyphenation parameters, Discretionary hyphen character(s).
- English 24, 26, 45. *See also* Language, User interface, English.
- Export words, *see* Words, Export.
- Failure messages
  - CV Pattern approach
    - Parse into natural classes 6, 15
    - Parse into segments 6, 15, 17, 43
    - Parse into syllables 6, 16
  - ONC approach
    - Parse into natural classes 30
    - Parse into segments 28
  - Sonority hierarchy approach
    - Parse into natural classes 22
    - Parse into segments 21
- Filters, *see* Onset-nucleus-coda approach, Filters.
- Fonts, *see* Language.
- French, *see* Language, User interface, French.
- Help 5
  - Introduction to syllabification 5
  - Tutorial 5
  - User documentation 5
- Hyphenation parameters 4
  - Discretionary hyphen character(s) 4, 33
  - Start hyphenating at *i* characters from the front 4, 33

- Stop hyphenating at  $j$  characters from the end 4, 33
- Import words, *see* Words, Import.
- Insert data item, *see* Add data item.
- Language
  - Analysis 4
  - User interface 4
    - English 4
    - French 4
    - Spanish 4
    - Vernacular 4
- Moraic approach 30
- New project, *see* Project, New.
- Nuclear projection approach 30
- Onset-nucleus-coda approach 22
  - Algorithm 43
  - Environments 27
  - Filters 25, 45
  - Graphemes 27
  - Predicted vs. correct syllabification 26
  - Segments 23
  - Sonority hierarchy 23
  - Syllabification parameters 24
  - Templates 24, 45
  - Try a Word 27
  - Words 26
- Open a project, *see* Project, Open.
- Optimality Theory approach 30
- Ordering data in display, *see* Sorting data in display.
- Parsing
  - Characters into segments 6
  - CV algorithm, *see* CV pattern approach, Algorithm.
  - Parsing result failure messages 5, 17, 43
  - Sonority hierarchy algorithm, *see* Sonority hierarchy approach, Algorithm.
- Predicted syllabification
  - Comparing with correct (CV) 16
  - Comparing with correct (ONC) 26
  - Comparing with correct (Son. Hier.) 22
  - Converting to correct 30
- Project
  - Management
    - Backup 31
    - Restore 31
  - New 3
  - Open 3
  - Save 3

- Save project as a new project 4
- Remove data item, *see* Delete data item.
- Save a project, *see* Project, Save.
- Save a project as a new project, *see* Project, Save project as a new project.
- Sonority hierarchy approach 17
  - Algorithm 17
  - Environments 19
  - Graphemes 19
  - Predicted vs. correct syllabification 22
  - Segments 18
  - Sonority hierarchy 18
  - Try a Word 19
- Sorting data in display 3
- Spanish, *see* Language, User interface, Spanish.
- Start hyphenating at *i* characters from the front, *see* Hyphenation parameters, Start hyphenating at *i* characters from the front.
- Stop hyphenating at *j* characters from the end, *see* Hyphenation parameters, Stop hyphenating at *j* characters from the end.
- Syllabification, *see* Help, Introduction to syllabification.
- Syllabification parameters 24
  - Codas allowed 24
  - Onset maximization 24
  - Onset principle 24
    - All but first syllable 24
    - Every syllable 24
    - Onsets not required 24
- Templates, *see* Onset-nucleus-coda approach, Templates.
- Try a Word 12, 19, 27
- Tutorial, *see* Help, Tutorial.
- User documentation, *see* Help, User documentation.
- User interface language, *see* Language, User interface.
- Vernacular language, *see* Language, Vernacular.
- Words
  - Export 4, 32
    - To **InDesign** 32
    - To **ParaText** 32
    - To **XLingPaper** 32
    - To a text file (plain text) 32
  - Find a word 33
  - Import 32
    - From **ParaText** 32
    - From a text file (plain text) 32
    - From **FieldWorks Language Explorer** 32
  - Remove all words 34
  - Remove correct syllable breaks in all words 34