Language Specifications for 1-800-Got-Photos

Angela Gui

December 2022

1 Introduction

1-800-Got-Photos is a language that allows users to modify a raster image's file extension and size easily. It relies on the ImageSharp library by SixLabors, and supports image format conversions between Jpeg, Png, Bmp, Gif, Pbm, Tiff, Tga and Webp, as well as basic uniform resizing operations.

The main reason behind the construction of the language is the need for a quick and easy way to modify an image file format from the command line. When users upload an image to a website, they could encounter file format and/or size constraints; instead of dealing with the inconveniences of having to find image file format converters online, they could now depend on 1-800-Got-Photos as a handy tool to convert and resize their image.

2 Design Principles

The main guiding principles in 1-800-Got-Photos's language design are simplicity and usability. As I want my language to be easy to use even for users who are not tech savvy, I modeled my syntax as intuitive as possible for an average user who understands English, such as having function names that are syntactically close to English (eg. parsing in/parsing out images using the Input and Output functions).

3 Examples

Example 1:

```
Input "Cat.jpeg"
```

Run the example using "dotnet run example-1.lang". This example loads in an raster image file with its name and path specified by "Cat.jpeg". If the program execution is successful, the command line output should be:

```
Your image Cat.jpeg is loaded

Program Success!
```

Example 2:

```
Input "Cat.jpeg"
OutPng "Cat.png"
```

```
OutGif "Cat.gif"
```

Run the example using ”dotnet run example-2.lang”. This example loads in an raster image file with its name and path specified by ”Cat.jpeg”, convert the loaded image file format into png and generate a png image file with its name and path specified by ”Cat.png”, convert the loaded image file format into gif and generate a gif image file with its name and path specified by ”Cat.gif”. If the program execution is successful, it will create two new image files Cat.png and Cat.gif in the same directory of the project. The command line output should be:

```
Your image Cat.jpeg is loaded

Your image Cat.png is saved

Your image Cat.gif is saved

Program Success!
```

Example 3:

```
Info "Cat.png"
Input "Cat.png"
MultSize 4
DivSize 2
OutWebp "NCat.webp"
Info "NCat.webp"
```

Run the example using ”dotnet run example-3.lang”. This example prints out the format, height and width about a raster image with its name and path specified by ”Cat.png”, loads in the same image, multiply its width and height by 4, divide its width and height by 2, convert the loaded image file format into webp and generate a webp image file with its name and path specified by ”NCat.webp”, and prints out information about the image ”NCat.webp”. If the program execution is successful, it will create one new image file NCat.webp in the same directory of the project. The command line output should be:

```
Image Format: Png
Image Height: 480px
Image Width: 481px

Your image Cat.png is loaded

Image size has been multiplied by 4

Image size has been divided by 2

Your image NCat.webp is saved

Image Format: WebP
Image Height: 960px
```

```
Image Width: 962px

Program Success!
```

4 Language Concept

In 1-800-Got-Photos, users will be writing mostly functions that take arguments in the form of String type. They need to have basic knowledge on different strings that represent function names, filenames, and/or values that gets passed into a function. Each function usually follows the pattern of FunctionName + white space + Argument. Any function that takes a filename as an argument will be expected to have quotation marks surrounding the filename. The primitives, such as the image, are baked into the program and are not expected to be called directly.

5 Formal Syntax

$\langle expr \rangle := \langle Input \rangle$
$|\langle Output \rangle$
$|\langle OutBmp \rangle$
$|\langle OutGif \rangle$
$|\langle OutJpeg \rangle$
$|\langle OutPbm \rangle$
$|\langle OutPng \rangle$
$|\langle OutTiff \rangle$
$|\langle OutTga \rangle$
$|\langle OutWebp \rangle$
$|\langle Info \rangle$
$|\langle MultSize \rangle$
$|\langle DivSize \rangle$
$|\langle LoadedImage \rangle$
$\langle Sequence \rangle ::= \langle expr \rangle \langle expr \rangle$
$\langle Input \rangle ::= \langle Input \rangle \langle filename \rangle$
$\langle Output \rangle ::= \langle Output \rangle \langle filename \rangle$
$\langle OutBmp \rangle ::= \langle OutBmp \rangle \langle filename \rangle$
$\langle OutGif \rangle ::= \langle OutGif \rangle \langle filename \rangle$
$\langle OutJpeg \rangle ::= \langle OutJpeg \rangle \langle filename \rangle$
$\langle OutPbm \rangle ::= \langle OutPbm \rangle \langle filename \rangle$
$\langle OutPng \rangle ::= \langle OutPng \rangle \langle filename \rangle$
$\langle OutTiff \rangle ::= \langle OutTiff \rangle \langle filename \rangle$
$\langle OutTga \rangle ::= \langle OutTga \rangle \langle filename \rangle$
$\langle OutWebp \rangle ::= \langle OutWebp \rangle \langle filename \rangle$
$\langle Info \rangle ::= \langle Info \rangle \langle filename \rangle$
$\langle MultSize \rangle ::= \langle MultSize \rangle \langle size \rangle$
$\langle DivSize \rangle ::= \langle DivSize \rangle \langle size \rangle$

$\langle filename \rangle ::= ' a' | ' A' | ' b' | ' B' | ... | ' y' | ' Y' | ' z' | ' Z' | ' .'$
$\langle size \rangle ::= ' 0' | ' 1' | ' 2' | ' 3' | ' 4' | ' 5' | ' 6' | ' 7' | ' 8' | ' 9'$
$\langle LoadedImage \rangle ::= Image$

6 Semantics

Input: Input's Syntax is Input "filename". Its Abstract Syntax Type is Input of filename. Its Type is String. It evaluates the filename String, loads in the image whose path, name, and file format is specified by filename and constructs an LoadedImage, which is then stored in a Map environment. There must exist an image whose path, name, and file format is specified by filename in the program directory, otherwise the evaluator fails and alerts users of the error.

Output: Output's Syntax is Output "filename". Its Abstract Syntax Type is Output of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as an image with its name and file format specified by filename. A LoadedImage must have been stored in the Map, otherwise the program fails and alerts users of the error.

OutBmp: OutBmp's Syntax is OutBmp "filename". Its Abstract Syntax Type is OutBmp of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Bmp image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .bmp, otherwise the evaluator fails and alerts users of the error.

OutGif: OutGif's Syntax is OutGif "filename". Its Abstract Syntax Type is OutGif of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Gif image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .gif, otherwise the evaluator fails and alerts users of the error.

OutJpeg : OutJpeg's Syntax is OutJpeg "filename". Its Abstract Syntax Type is OutJpeg of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Jpeg image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .jpeg, otherwise the evaluator fails and alerts users of the error.

OutPbm: OutPbm's Syntax is OutPbm "filename". Its Abstract Syntax Type is OutPbm of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Pbm image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .pbm, otherwise the evaluator fails and alerts users of the error.

OutPng: OutPng's Syntax is OutPng "filename". Its Abstract Syntax Type is OutPng of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Png image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .png, otherwise the evaluator fails and alerts users of the error.

OutTiff: OutTiff's Syntax is OutTiff "filename". Its Abstract Syntax Type is OutTiff of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Tiff image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .tiff, otherwise the evaluator fails and alerts users of the error.

OutTga: OutTga's Syntax is OutTga "filename". Its Abstract Syntax Type is OutTga of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Tga image with its name specified by filename. A LoadedImage must have been stored in the Map, and filename must contain .tga, otherwise the evaluator fails and alerts users of the error.

OutWebp: OutWebp's Syntax is OutWebp "filename". Its Abstract Syntax Type is OutWebp of filename. Its Type is String. It evaluates the filename String, detects whether a LoadedImage has been stored in the Map, then saves the LoadedImage to the program directory as a Webp image with its name specified by filename. A LoadedImage must

have been stored in the Map, and filename must contain .webp, otherwise the evaluator fails and alerts users of the error.

Info: Info's Syntax is Info "filename". Its Abstract Syntax Type is Info of filename. Its Type is String. It evaluates the filename String, and prints out information about the image format, height, width of the image whose path and name is specified by filename.

MultSize: MultSize's Syntax is MultSize "size". Its Abstract Syntax Type is MultSize of size. Its Type is String. It evaluates the size String, converts it to an Int, detects whether a LoadedImage has been stored in the Map, and multiplies the LoadedImage's width and height by the specified Int. A LoadedImage must have been stored in the Map, and the size String must be able to be parsed in as the 32-bit F# Integer type (Int32), otherwise the evaluator fails and alerts users of the error.

DivSize: DivSize's Syntax is DivSize "size". Its Abstract Syntax Type is DivSize of size. Its Type is String. It evaluates the size String, converts it to an Int, detects whether a LoadedImage has been stored in the Map, and divides the LoadedImage's width and height by the specified Int. A LoadedImage must have been stored in the Map, and the size String must be able to be parsed in as the 32-bit F# Integer type (Int32), otherwise the evaluator fails and alerts users of the error.

Sequence: Sequence's Syntax is expr expr. Its Abstract Syntax Type is Sequence of Expr list. Its Type is Expr list. It evaluates a sequence of exprs, returning the last one in the sequence.

LoadedImage: LoadedImage does not have an explicit syntax. Its Abstract Syntax Type is LoadedImage of Image. It is a terminal of Image type, defined by the ImageSharp Library.

7 Remaining Work

The next step in the project would be creating variable and variable assignments so that the language could take in multiple images at the same time. Currently, 1-800-Got-Photos only supports conversion and output of one image at a time; subsequent image load ins will overwrite the previous input, and if Output is called, the program will always generate the new image based on the last image that is loaded in using Input. Implementing variables would allow users to convert, resize, and keep track of multiple images at the same time. In the earlier versions of the language, there were ideas about adding features that allow users to flip and recolor certain pixels of the entire image, however these were discarded due to concerns about usability as well as the complexities of implementations.