

# Module Interface Specification for OCRacle

Phillip Tran

March 14, 2025

# 1 Revision History

Date	Version	Notes
March 13, 2025	1.0	Initial Document Creation

## 2 Symbols, Abbreviations and Acronyms

See SRS Documentation at <https://github.com/ptrandev/OCRacle/blob/main/docs/SRS/SRS.pdf> for symbols, abbreviations and acronyms.

# Contents

<b>1</b>	<b>Revision History</b>	<b>i</b>
<b>2</b>	<b>Symbols, Abbreviations and Acronyms</b>	<b>ii</b>
<b>3</b>	<b>Introduction</b>	<b>1</b>
<b>4</b>	<b>Notation</b>	<b>1</b>
<b>5</b>	<b>Module Decomposition</b>	<b>1</b>
<b>6</b>	<b>MIS of Application Module</b>	<b>3</b>
6.1	Module . . . . .	3
6.2	Uses . . . . .	3
6.3	Syntax . . . . .	3
6.3.1	Exported Constants . . . . .	3
6.3.2	Exported Access Programs . . . . .	3
6.4	Semantics . . . . .	3
6.4.1	State Variables . . . . .	3
6.4.2	Environment Variables . . . . .	3
6.4.3	Assumptions . . . . .	3
6.4.4	Access Routine Semantics . . . . .	3
6.4.5	Local Functions . . . . .	4
<b>7</b>	<b>MIS of Input Format Module</b>	<b>4</b>
7.1	Module . . . . .	4
7.2	Uses . . . . .	4
7.3	Syntax . . . . .	4
7.3.1	Exported Constants . . . . .	4
7.3.2	Exported Access Programs . . . . .	4
7.4	Semantics . . . . .	4
7.4.1	State Variables . . . . .	4
7.4.2	Environment Variables . . . . .	5
7.4.3	Assumptions . . . . .	5
7.4.4	Access Routine Semantics . . . . .	5
7.4.5	Local Functions . . . . .	5
<b>8</b>	<b>MIS of Model Output Module</b>	<b>5</b>
8.1	Module . . . . .	5
8.2	Uses . . . . .	5
8.3	Syntax . . . . .	5
8.3.1	Exported Constants . . . . .	5
8.3.2	Exported Access Programs . . . . .	6

8.4	Semantics . . . . .	6
8.4.1	State Variables . . . . .	6
8.4.2	Environment Variables . . . . .	6
8.4.3	Assumptions . . . . .	6
8.4.4	Access Routine Semantics . . . . .	6
8.4.5	Local Functions . . . . .	6
<b>9</b>	<b>MIS of Model Training Module</b>	<b>7</b>
9.1	Module . . . . .	7
9.2	Uses . . . . .	7
9.3	Syntax . . . . .	7
9.3.1	Exported Constants . . . . .	7
9.3.2	Exported Access Programs . . . . .	7
9.4	Semantics . . . . .	7
9.4.1	State Variables . . . . .	7
9.4.2	Environment Variables . . . . .	7
9.4.3	Assumptions . . . . .	7
9.4.4	Access Routine Semantics . . . . .	7
9.4.5	Local Functions . . . . .	8
<b>10</b>	<b>MIS of Model Testing Module</b>	<b>8</b>
10.1	Module . . . . .	8
10.2	Uses . . . . .	8
10.3	Syntax . . . . .	8
10.3.1	Exported Constants . . . . .	8
10.3.2	Exported Access Programs . . . . .	8
10.4	Semantics . . . . .	8
10.4.1	State Variables . . . . .	8
10.4.2	Environment Variables . . . . .	9
10.4.3	Assumptions . . . . .	9
10.4.4	Access Routine Semantics . . . . .	9
10.4.5	Local Functions . . . . .	9
<b>11</b>	<b>MIS of Prediction Model Module</b>	<b>9</b>
11.1	Module . . . . .	9
11.2	Uses . . . . .	10
11.3	Syntax . . . . .	10
11.3.1	Exported Constants . . . . .	10
11.3.2	Exported Access Programs . . . . .	10
11.4	Semantics . . . . .	10
11.4.1	State Variables . . . . .	10
11.4.2	Environment Variables . . . . .	10
11.4.3	Assumptions . . . . .	10

11.4.4	Access Routine Semantics . . . . .	10
11.4.5	Local Functions . . . . .	11
<b>12</b>	<b>MIS of Image Preprocessing Module</b>	<b>11</b>
12.1	Module . . . . .	11
12.2	Uses . . . . .	11
12.3	Syntax . . . . .	11
12.3.1	Exported Constants . . . . .	11
12.3.2	Exported Access Programs . . . . .	11
12.4	Semantics . . . . .	11
12.4.1	State Variables . . . . .	11
12.4.2	Environment Variables . . . . .	11
12.4.3	Assumptions . . . . .	11
12.4.4	Access Routine Semantics . . . . .	11
12.4.5	Local Functions . . . . .	12
<b>13</b>	<b>MIS of Performance Metrics Module</b>	<b>12</b>
13.1	Module . . . . .	12
13.2	Uses . . . . .	12
13.3	Syntax . . . . .	12
13.3.1	Exported Constants . . . . .	12
13.3.2	Exported Access Programs . . . . .	12
13.4	Semantics . . . . .	12
13.4.1	State Variables . . . . .	12
13.4.2	Environment Variables . . . . .	12
13.4.3	Assumptions . . . . .	13
13.4.4	Access Routine Semantics . . . . .	13
13.4.5	Local Functions . . . . .	13
<b>14</b>	<b>MIS of Graphical User Interface Module</b>	<b>13</b>
14.1	Module . . . . .	13
14.2	Uses . . . . .	13
14.3	Syntax . . . . .	13
14.3.1	Exported Constants . . . . .	13
14.3.2	Exported Access Programs . . . . .	13
14.4	Semantics . . . . .	14
14.4.1	State Variables . . . . .	14
14.4.2	Environment Variables . . . . .	14
14.4.3	Assumptions . . . . .	14
14.4.4	Access Routine Semantics . . . . .	14
14.4.5	Local Functions . . . . .	14
<b>15</b>	<b>Appendix</b>	<b>16</b>

## 3 Introduction

The following document details the Module Interface Specifications for OCRacle, an optical character recognition (OCR) program for identifying Latin alphabet characters.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/ptrandev/OCRacle/>.

## 4 Notation

[You should describe your notation. You can use what is below as a starting point. —SS]

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol  $:=$  is used for a multiple assignment statement and conditional rules follow the form  $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$ .

The following table summarizes the primitive data types used by OCRacle.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in $(-\infty, \infty)$
natural number	$\mathbb{N}$	a number without a fractional component in $[1, \infty)$
real	$\mathbb{R}$	any number in $(-\infty, \infty)$

The specification of OCRacle uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, OCRacle uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

## 5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding	
Behaviour-Hiding	Input Format Module Model Output Module Model Training Module Model Testing Module Prediction Model Module Application Module OAR Performance Metrics Module
Software Decision	Image Preprocessing Module Performance Metrics Module Graphical User Interface Module

Table 1: Module Hierarchy



## 6 MIS of Application Module

### 6.1 Module

[Short name for the module —SS]

### 6.2 Uses

- Graphical User Interface Module [14](#)
- Input Format Module [7](#)
- Model Output Module [8](#)

### 6.3 Syntax

#### 6.3.1 Exported Constants

#### 6.3.2 Exported Access Programs

Name	In	Out	Exceptions
<a href="#">[accessProg —SS]</a>	-	-	-

### 6.4 Semantics

#### 6.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

#### 6.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

#### 6.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

#### 6.4.4 Access Routine Semantics

[\[accessProg —SS\]](#)():

- transition: [\[if appropriate —SS\]](#)

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

### 6.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 7 MIS of Input Format Module

### 7.1 Module

input

### 7.2 Uses

- Image Preprocessing Module [12](#)

### 7.3 Syntax

#### 7.3.1 Exported Constants

N/A

#### 7.3.2 Exported Access Programs

Name	In	Out	Exceptions
input	string	$T_{28 \times 28}$ where $T_{i,j} \in \{0, \dots, 255\}$	DimensionsError, FormatError

### 7.4 Semantics

#### 7.4.1 State Variables

N/A

### 7.4.2 Environment Variables

- $n_{min}$ : minimum number of pixels in the vertical direction
- $n_{max}$ : maximum number of pixels in the vertical direction
- $m_{min}$ : minimum number of pixels in the horizontal direction
- $m_{max}$ : maximum number of pixels in the horizontal direction
- supportedFormats: list of supported image formats (e.g. PNG, JPEG)

### 7.4.3 Assumptions

N/A

### 7.4.4 Access Routine Semantics

input(path):

- output: image := preprocessing(inputImage)
- exception: A DimensionsError exception is thrown if the input image exceeds the bounds of the  $n_{min}$ ,  $n_{max}$ ,  $m_{min}$ , and  $m_{max}$  parameters. A FormatError exception is thrown if the input image is not one of the supported formats.

### 7.4.5 Local Functions

## 8 MIS of Model Output Module

### 8.1 Module

output

### 8.2 Uses

- Prediction Model Module [11](#)

### 8.3 Syntax

#### 8.3.1 Exported Constants

N/A

### 8.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

## 8.4 Semantics

### 8.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

### 8.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

### 8.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

### 8.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

### 8.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 9 MIS of Model Training Module

### 9.1 Module

[Short name for the module —SS]

### 9.2 Uses

- Performance Metrics Module 13
- Model Output Module 8
- Input Format Module 7

### 9.3 Syntax

#### 9.3.1 Exported Constants

#### 9.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

### 9.4 Semantics

#### 9.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

#### 9.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

#### 9.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

#### 9.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

#### 9.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 10 MIS of Model Testing Module

### 10.1 Module

[Short name for the module —SS]

### 10.2 Uses

- Performance Metrics Module 13
- Model Output Module 8
- Input Format Module 7

### 10.3 Syntax

#### 10.3.1 Exported Constants

#### 10.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

### 10.4 Semantics

#### 10.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

### 10.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

### 10.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

### 10.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

### 10.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 11 MIS of Prediction Model Module

### 11.1 Module

[Short name for the module —SS]

## 11.2 Uses

## 11.3 Syntax

### 11.3.1 Exported Constants

### 11.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

## 11.4 Semantics

### 11.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

### 11.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

### 11.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

### 11.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]



### 11.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 12 MIS of Image Preprocessing Module

### 12.1 Module

[Short name for the module —SS]

### 12.2 Uses

### 12.3 Syntax

#### 12.3.1 Exported Constants

#### 12.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

### 12.4 Semantics

#### 12.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

#### 12.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

#### 12.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

#### 12.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]

- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

#### 12.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 13 MIS of Performance Metrics Module

### 13.1 Module

[Short name for the module —SS]

### 13.2 Uses

### 13.3 Syntax

#### 13.3.1 Exported Constants

#### 13.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

### 13.4 Semantics

#### 13.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

#### 13.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

### 13.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

### 13.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

### 13.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## 14 MIS of Graphical User Interface Module

### 14.1 Module

[Short name for the module —SS]

### 14.2 Uses

- Hardware-Hiding Module

### 14.3 Syntax

#### 14.3.1 Exported Constants

#### 14.3.2 Exported Access Programs

Name	In	Out	Exceptions
[accessProg —SS]	-	-	-

## 14.4 Semantics

### 14.4.1 State Variables

[Not all modules will have state variables. State variables give the module a memory. —SS]

### 14.4.2 Environment Variables

[This section is not necessary for all modules. Its purpose is to capture when the module has external interaction with the environment, such as for a device driver, screen interface, keyboard, file, etc. —SS]

### 14.4.3 Assumptions

[Try to minimize assumptions and anticipate programmer errors via exceptions, but for practical purposes assumptions are sometimes appropriate. —SS]

### 14.4.4 Access Routine Semantics

[accessProg —SS]():

- transition: [if appropriate —SS]
- output: [if appropriate —SS]
- exception: [if appropriate —SS]

[A module without environment variables or state variables is unlikely to have a state transition. In this case a state transition can only occur if the module is changing the state of another module. —SS]

[Modules rarely have both a transition and an output. In most cases you will have one or the other. —SS]

### 14.4.5 Local Functions

[As appropriate —SS] [These functions are for the purpose of specification. They are not necessarily something that is going to be implemented explicitly. Even if they are implemented, they are not exported; they only have local scope. —SS]

## References

- Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.

## 15 Appendix

[Extra information if required —SS]