MATLAB code for base-to-base conversion

Authors
Deeksha Hareesha Kulal (PUID - 0036389500)
Archana Kumar (PUID - 0036389786)

Introduction

Historical Background

The concept of base-to-base conversion, also known as numeral system conversion, has deep historical roots that trace back to the earliest human civilizations. The need to represent quantities and values in a systematic way led to the development of various numeral systems, each with its unique symbols and rules. From the Babylonian sexagesimal system to the Roman numerals, diverse cultures contributed to the evolution of numerical representation.

In the European Renaissance, the Hindu-Arabic numeral system gained prominence. Fibonacci, an Italian mathematician, played a key role in popularizing this system in Europe during the 13th century. The decimal system, with its base-10 structure and the concept of zero, became the standard for arithmetic operations and laid the groundwork for the scientific revolution.

In the modern era, the advent of digital computers and programming languages brought a renewed emphasis on base-to-base conversion. This process involves converting numbers from one numeral system (or base) to another, facilitating communication and computation across different mathematical frameworks. The foundational principles of these conversions are rooted in the work of mathematicians and logicians who laid the groundwork for our understanding of number systems.

One of the notable contributors to the understanding of numeral systems is the Indian mathematician Brahmagupta, who introduced the concept of zero and negative numbers in the seventh century. His seminal work "Brahmasphutasiddhanta" played a crucial role in shaping the development of arithmetic and algebra.

Moreover, the influential work of George Boole in the 19th century, particularly his development of Boolean algebra, laid the foundation for digital logic and binary representation in computing. This work, coupled with the later innovations of Claude Shannon, established the binary numeral system as fundamental to digital computing.

The binary system's simplicity and compatibility with electronic circuits became the basis for modern computers.

Importance of Base-to-Base Conversion

In the context of modern programming languages like MATLAB, base-to-base conversion plays a pivotal role in data manipulation, communication between different systems, and numerical analysis. MATLAB, a high-level language widely used in engineering, mathematics, and scientific computing, provides powerful tools for handling numerical data.

Understanding and implementing base-to-base conversion in MATLAB is essential for several reasons:

1. Data Representation:
MATLAB often deals with diverse data types and requires the ability to seamlessly convert between different representations. Base-to-base conversion facilitates the manipulation and analysis of data in various numeral systems.

2. Algorithm Development:
Many algorithms and mathematical models are based on specific numeral systems. Adapting these algorithms to different bases is crucial for solving a wide range of problems efficiently.

3. Communication with External Systems:
Interfacing with external systems, such as databases or communication protocols, may require conversion between different numerical bases. MATLAB's flexibility in base conversion allows for seamless integration with external data sources.

4. Numerical Analysis:
MATLAB is extensively used for numerical analysis and simulations. Base-to-base conversion is fundamental for handling different numerical bases during these analyses.

In this report, we will delve into the MATLAB code for base-to-base conversion, exploring its practical implementation and the versatility it offers in numerical computing.

Methods

1) Basic Base to Base conversion
2) Application 1 -Encoding and Decoding Text
3) Application 2 -Network Communication: IP Address Conversion
4) Application 3 - Geocoding Coordinate Conversion

Method of Base to Base Conversion

Key Components

baseConversion Function:

This function serves as the core of the code and is responsible for converting a number from one base to another. It handles both integer and fractional parts of the number.

Input Parameters:

- number: The input number as a string.
- fromBase: The base of the input number.
- toBase: The desired base for the output.

Output:

- The function returns the converted number in the specified base as a string.

Function Workflow:

- The input number is split into integer and fractional parts.
- The integer part is converted to decimal using MATLAB's base2dec function.
- If a fractional part exists, it is converted to decimal manually.
- The integer and fractional parts are combined to obtain the decimal number.
- The decimal number is then converted to the desired base using MATLAB's dec2base function.
- The resulting integer and fractional parts in the new base are combined to form the final converted number.

Method of Encoding and Decoding Text

Introduction:

The method of encoding and decoding text is a fundamental process in computer science, where textual information is transformed into a different representation for various purposes, such as efficient storage, secure transmission, and data integrity. In this context, we'll delve into a self-contained analysis of a specific method for encoding and decoding text implemented in MATLAB. The implemented method involves two MATLAB functions: ‘EncodeString’ and ‘DecodeString’. These functions facilitate the conversion of text between its original form and a specified numerical base. The numerical base acts as a fundamental parameter, allowing flexibility in representing text in different formats.

Encoding Process (Function: 'EncodeString'):

The `EncodeString` function is designed to encode a given text into a specified numerical base.

Input Parameters:

- `textToEncode`: This is the input text that you want to encode.
- `baseNo`: This parameter represents the numerical base into which the text will be encoded.

Output :

- The function returns the encoded string, denoted as `encodedStr`.

Function: 'EncodeString' - Workflow

ASCII Conversion:

- The input text, `textToEncode`, is converted into an array of ASCII values using the `double` function in MATLAB. The resulting array is stored in the variable `asciiValues`.

Initialization:

- An empty string, `encodedStr`, is initialized to hold the final encoded result.

Iteration through ASCII Values:

- The function iterates through each ASCII value in the `asciiValues` array using a `for` loop.

Base-to-Base Conversion:

- For each ASCII value, it is converted to the specified numerical base (`baseNo`) using a custom base-to-base conversion function called `baseToBase_custom`. The result of this conversion is stored in the variable `encode`.

String Construction:

- The encoded value (`encode`) is then appended to the `encodedStr` string. If it's not the first iteration (i.e., `i` is not equal to 1), a delimiter ("/") is added before appending. This helps separate individual encoded values in the final string.

The final encoded string (`encodedStr`) is constructed and represents the entire input text encoded into the specified numerical base.

```
function encodedStr = EncodeString(textToEncode,baseNo)
asciiValues = double(textToEncode);
encodedStr = '';

for i=1:length(asciiValues)
     encode = baseConversion(num2str(asciiValues(i)),10,baseNo);
     if i~=1
        encodedStr = [encodedStr,'/',encode];
     else
        encodedStr = [encodedStr,encode];
     end

end
end
```

Function: 'DecodeString' - Workflow

The ``'DecodeString' function is designed to decode a specified numerical base into text.

Input Parameters:

- `textToDecode`: This is the input text that you want to encode.
- `baseNo`: This parameter represents the numerical base into which the text will be encoded.

Output :

- The function returns the encoded string, denoted as `decodedStr`.

WorkFlow:

Splitting the Encoded String:

- The encoded string is split into substrings using a delimiter, typically '/'.

Base-to-Base Conversion:

- Each substring is then converted back to decimal from the specified numerical base using the `baseToBase_custom` function.

Decimal to Character Conversion:

- Decimal values are converted to characters using MATLAB functions.

String Construction:

- The characters obtained from each substring are concatenated into the final decoded string.

```
function decodedStr = DecodeString(encodedStr,baseNo)
decodedStr = '';
splitStrings = strsplit(encodedStr, '/');
for i = 1:numel(splitStrings)
    currentCell = splitStrings{i};
    decode = baseConversion(num2str(currentCell),baseNo,10);
    decodedStr = [decodedStr,char(str2double(decode))];
end
end
```

The resulting string represents the original information, providing a reverse transformation to the encoding process. The code assumes integration with MATLAB App Designer, as evident from the function names like '**Encode/Decode'**. It suggests the code is part of an interactive application where users can trigger IP address conversions by interacting with buttons or other UI elements.

Method of Network Communication: IP Address Conversion

The MATLAB code is designed to convert IP addresses between different bases, specifically between binary (base 2), decimal (base 10), and hexadecimal (base 16). The code provides a flexible and reusable function for converting IP addresses, allowing users to specify the input base, target base, and the format of the output.

Key Components

'baseConversion' Function:

This function serves as the core of the code and is responsible for converting a number from one base to another. It handles both integer and fractional parts of the number.

convertIP Function:

This function uses the baseConversion function to convert entire IP addresses between different bases. It is designed to be used within the MATLAB App Designer.

Input Parameters:

- ipAddress: The IP address as a string.
- toBase: The desired base for the output IP address.

Output:

- The function returns the converted IP address in the specified base as a string.

Function Workflow:

Validation:

- The function starts by validating the input bases (fromBase and toBase). It checks whether they are among the supported bases (2 for binary, 10 for decimal, and 16 for hexadecimal).

Octet Splitting:

- The input IP address is split into octets using the dot ('.') as a delimiter. Each octet represents a segment of the IP address.

Conversion Loop:

- The function iterates through each octet, converting it to the desired base using the baseConversion function.
- The converted octets are stored in a cell array.

Formatting for Base 2 and Base 16:

- For base 2 (binary) conversion, the function ensures that each binary octet has a fixed length of 8 bits by using the padZeros function.
- For base 16 (hexadecimal) conversion, the function removes spaces between octets.

Final Conversion:

- The function joins the converted octets to form the final converted IP address.
- For base 10 (decimal) conversion, the octets are separated by dots ('.').

Output:

- The function returns the final converted IP address.

```
function convertedIP = convertIP(ipAddress, toBase)

    % Validate input bases
    validBases = [2, 10, 16];
    if ~ismember(10, validBases) || ~ismember(toBase, validBases)
        error('Invalid base. Use 2 for binary or 16 for hexadecimal.');
    end

    % Split the IP address into octets
    octets = strsplit(ipAddress, '.');

    % Convert each octet to the desired base using baseConversion
    convertedOctets = cellfun(@(octet) baseConversion(octet, 10, toBase), octets, 'UniformOutput', false);

    % Combine the octets without spaces for base 16 or with leading zeros for base 2
    if toBase == 16
        convertedIP = strjoin(convertedOctets, '');
    elseif toBase == 2
        convertedIP = strjoin(cellfun(@(octet) padZeros(octet, 8), convertedOctets, 'UniformOutput', false), ' ');
    elseif toBase == 10
        convertedIP = strjoin(convertedOctets, '.');
    else
        error('Unsupported "toBase" value. Use 2 for binary, 10 for decimal, or 16 for hexadecimal.');
    end
end
```

```
function paddedNumber = padZeros(number, targetLength)
    % Pad the binary or hexadecimal representation with leading zeros

    currentLength = length(number);
    if currentLength < targetLength
        numZeros = targetLength - currentLength;
        paddedNumber = [repmat('0', 1, numZeros), number];
    else
        paddedNumber = number;
    end
end
```

The code assumes integration with MATLAB App Designer, as evident from the function names like IPAddressConversionButtonPushed. It suggests the code is part of an interactive application where users can trigger IP address conversions by interacting with buttons or other UI elements.

Geocoding Coordinate Conversion

The MATLAB code aims to convert latitude or longitude coordinates represented in decimal degrees to the degrees, minutes, and seconds (DMS) format. This conversion involves internal usage of the baseConversion function for flexible and customizable base conversions.

Input Parameters:

- The user provides a decimal degree value (decimalDegrees) representing a latitude or longitude coordinate.

Output:

- The final DMS representation is displayed with degrees, minutes, and seconds in the specified base.
- The example uses Base 16 for DMS representation, providing a customizable approach for different bases.

Function Workflow:

Extraction of Integer and Fractional Parts

- The function convertDecimalToDMS function extracts the integer part (degrees) and the fractional part from the provided decimal degree value.

Conversion to Minutes and Seconds:

- The fractional part is then converted to minutes and seconds using base conversion techniques:
- The fractional part is multiplied by 60 to get minutes.
- The remaining fractional part is multiplied by 60 to get seconds.

Base Conversion using baseConversion:

- The baseConversion function is applied to convert degrees, minutes, and seconds to the desired base (Base 16 in this example).
- The baseConversion function internally handles the conversion of each part.
- The baseConversion function is internally used to convert fractional parts (minutes and seconds) to the desired base (Base 16 in this example).
- This internal base conversion allows flexibility in choosing the base representation.

The MATLAB code demonstrates the application of base-to-base conversion in geocoding, allowing users to represent coordinates in different bases. The use of the baseConversion function adds flexibility to the conversion process, making it customizable for various scenarios. The example provides insight into the conversion of decimal degrees to DMS format, showcasing the versatility of base conversion in geospatial applications.

```
% Function to convert decimal degrees to degrees, minutes, and
% seconds using baseConversion
function dms = convertDecimalToDMS(decimalDegrees)
    % Extract the integer part (degrees)
    degrees = floor(decimalDegrees);

    % Extract the fractional part
    fractionalPart = decimalDegrees - degrees;

    % Convert the fractional part to minutes and seconds using
    % baseConversion
    minutesDecimal = fractionalPart * 60;
    minutes = floor(minutesDecimal);

    secondsDecimal = (minutesDecimal - minutes) * 60;
    seconds = floor(secondsDecimal);

    % Convert the degrees, minutes, and seconds to the desired base
    degreesBase = baseConversion(num2str(degrees), 10, 16);
    minutesBase = baseConversion(num2str(minutes), 10, 16);
    secondsBase = baseConversion(num2str(seconds), 10, 16);

    % Format the result
    dms = [degreesBase '° ' minutesBase ''' ' secondsBase '"'];
end
```

Applications

Base-to-Base Convertor

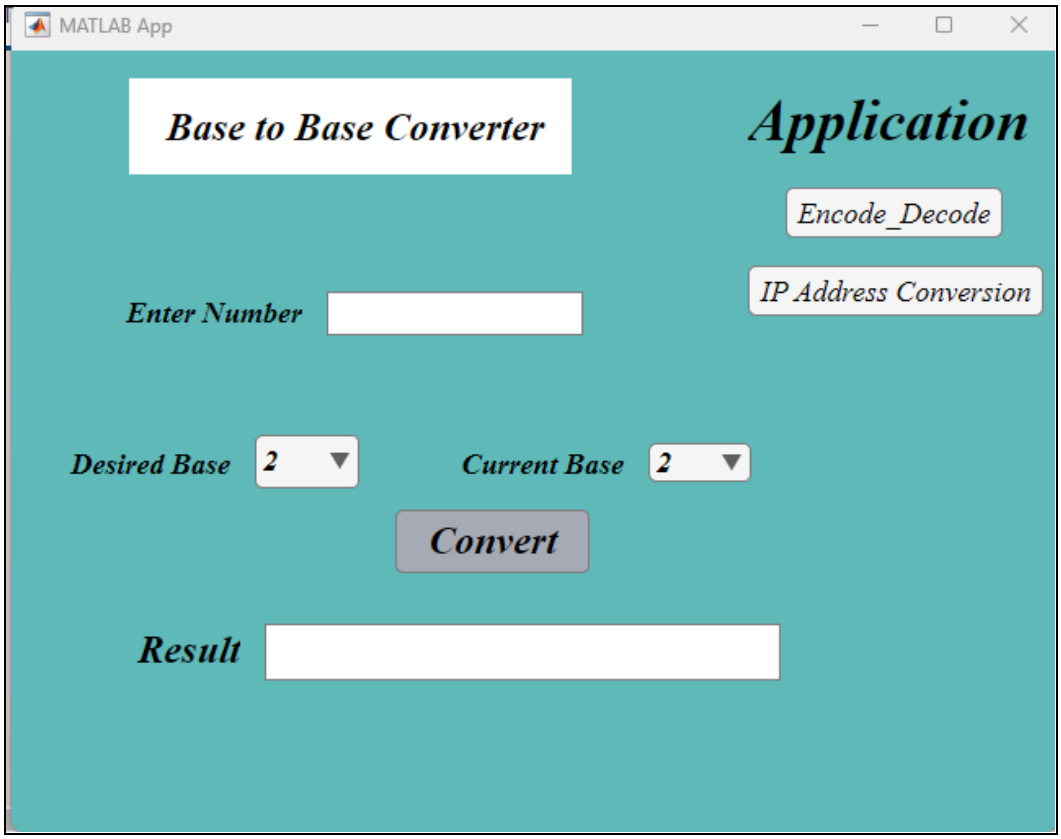
This function is an event handler that is executed when a button, named "Convert," is pressed in a MATLAB App Designer application. It retrieves values from the UI components of the app, performs a base conversion, and updates the result in another UI component.

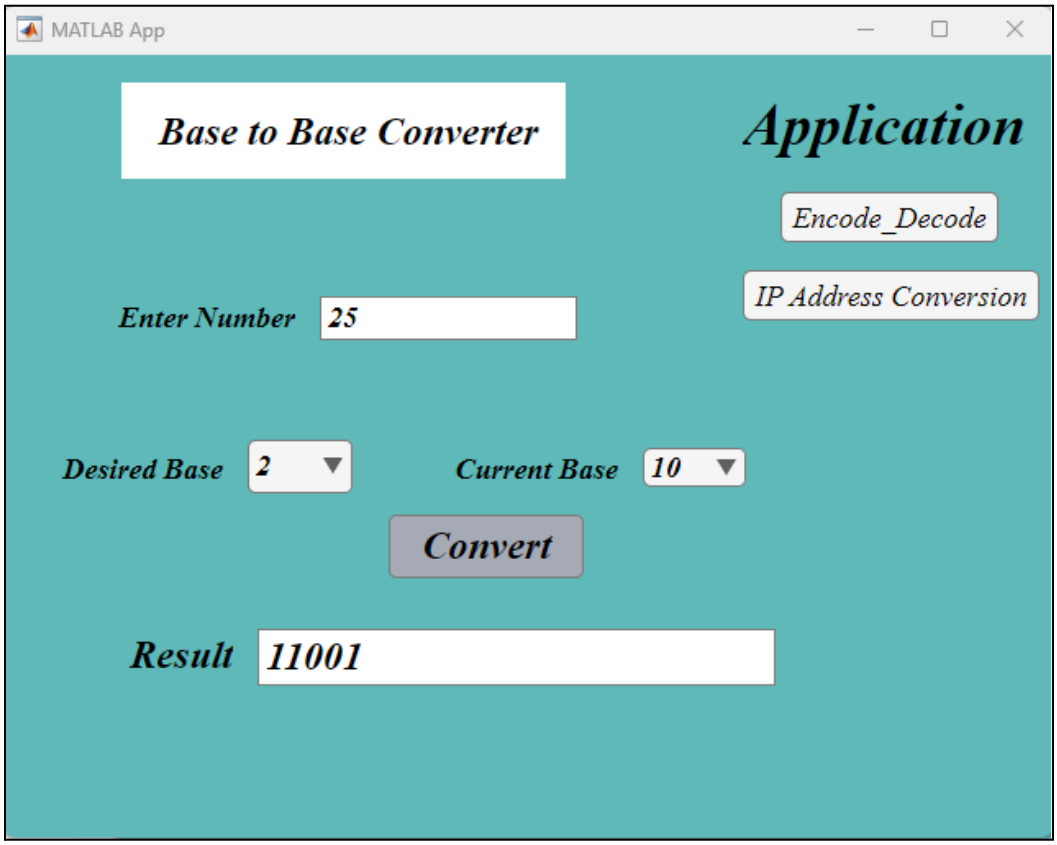
Steps

1. **Get Input Values:**
    - num: Retrieves the value entered in the "EnterNumberEditField" component. This is the number that will be converted.
    - currentBase: The current base is obtained from the "CurrentBaseDropDown" component. It likely represents the base of the entered number.
    - desiredBase: The desired base is obtained from the "DesiredBaseDropDown" component. It represents the base to which the number will be converted.
2. **Call the baseConversion function with the extracted values:**
    - num is the number to be converted.
    - str2num(currentBase): Converts the current base from a string to a numeric value.
    - str2num(desiredBase): Converts the desired base from a string to a numeric value.
    - result: Stores the result of the base conversion.
3. **Update Result UI Component:**
    - Update the "ResultEditField" UI component in the app with the result of the conversion.

```
classdef app1 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: ConvertButton
        function ConvertButtonPushed(app, event)
            num=app.EnterNumberEditField.Value;
            currentBase=app.CurrentBaseDropDown.Value;
            desiredBase=app.DesiredBaseDropDown.Value;
            type baseConversion.m
            result = baseConversion(num, str2num(currentBase), str2num(desiredBase));
            app.ResultEditField.Value = result;
        end

        % Button pushed function: IPAddressConversionButton
        function IPAddressConversionButtonPushed(app, event)
            createComponents(app3);
        end

        % Button pushed function: Encode_DecodeButton
        function Encode_DecodeButtonPushed(app, event)
            createComponents(app4);
        end

        % Button pushed function: GeocodingButton
        function GeocodingButtonPushed(app, event)
            createComponents(app5);
        end
    end
```

```
    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app) ...
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app1 ...

        % Code that executes before app deletion
        function delete(app) ...
    end
end
```


MATLAB App
Base to Base Converter
Application
Encode_Decode
IP Address Conversion
Enter Number
Desired Base
2
Current Base
2
Convert
Result


MATLAB App
Base to Base Converter
Application
Encode_Decode
IP Address Conversion
Enter Number
25
Desired Base
2
Current Base
10
Convert
Result
11001

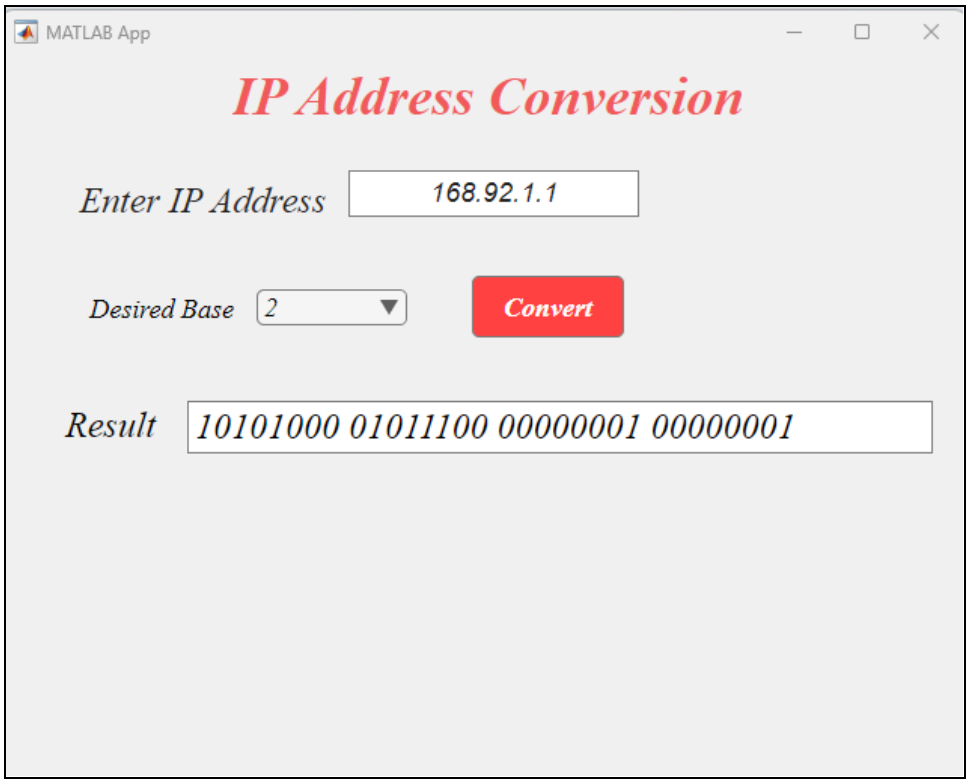
Network Communication: IP Address Conversion

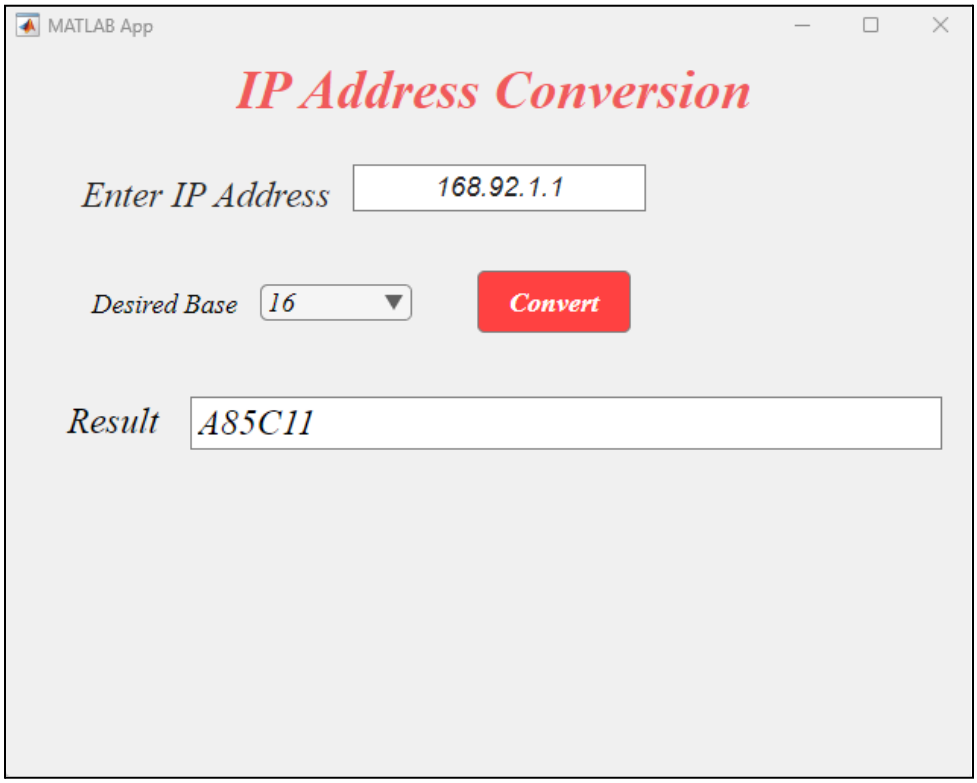
This function is an event handler that is executed when a button, likely named "Convert" , is pressed in a MATLAB App Designer application. It retrieves an IP address from the UI component, performs a base conversion using an external function, and updates the result in another UI component.

Steps:

1. **Get Input Values:**
- ipAddress: Retrieve the value entered in the "EnterIPAddressEditField" component. This is the IP address to be converted.
- toBase: Get the desired base from the "DesiredBaseDropDown" component. It represents the base to which the IP address will be converted.
2. **Call convertIP Function:**
- Call the convertIP function with the extracted values.
- ipAddress: The IP address to be converted.
- str2double(toBase): Convert the desired base from a string to a numeric value.
- result: Store the result of the IP address conversion.
3. **Update Result UI Component:**
- Update the "ResultEditField" UI component in the app with the result of the IP address conversion.

```
classdef app3 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: ConvertButton
        function ConvertButtonPushed(app, event)
            ipAddress=app.EnterIPAddressEditField.Value;
            toBase=app.DesiredBaseDropDown.Value;
            type convertIP.m
            result = convertIP(ipAddress, str2double(toBase));
            app.ResultEditField.Value=result;

        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app) ...
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app3 ...

        % Code that executes before app deletion
        function delete(app) ...
    end
end
```


MATLAB App
IP Address Conversion
Enter IP Address
168.92.1.1
Desired Base
2
Convert
Result
10101000 01011100 00000001 00000001


MATLAB App
IP Address Conversion
Enter IP Address
168.92.1.1
Desired Base
16
Convert
Result
A85C11

Encoding and Decoding Text

This function is an event handler that is executed when a button, likely named "EncodeDecodeButton," is pressed in a MATLAB App Designer application. It retrieves a string from the UI component, performs encoding and decoding operations using an external function, and updates the results in other UI components.

Steps:

Input Values:

- textToEncode: Retrieves the value entered in the "EntertheStringEditField" component. This is the string to be encoded and decoded.
- baseNo: Gets the base value from the "BaseDropDown" component. It likely represents the base used for encoding and decoding.

EncodeString Function:

- The EncodeString function is called with the extracted values.
- textToEncode is the string to be encoded.
- str2double(baseNo): Converts the base from a string to a numeric value.
- encodedStr: Stores the result of the encoding operation.

Updation of Result UI Component (Encoding):

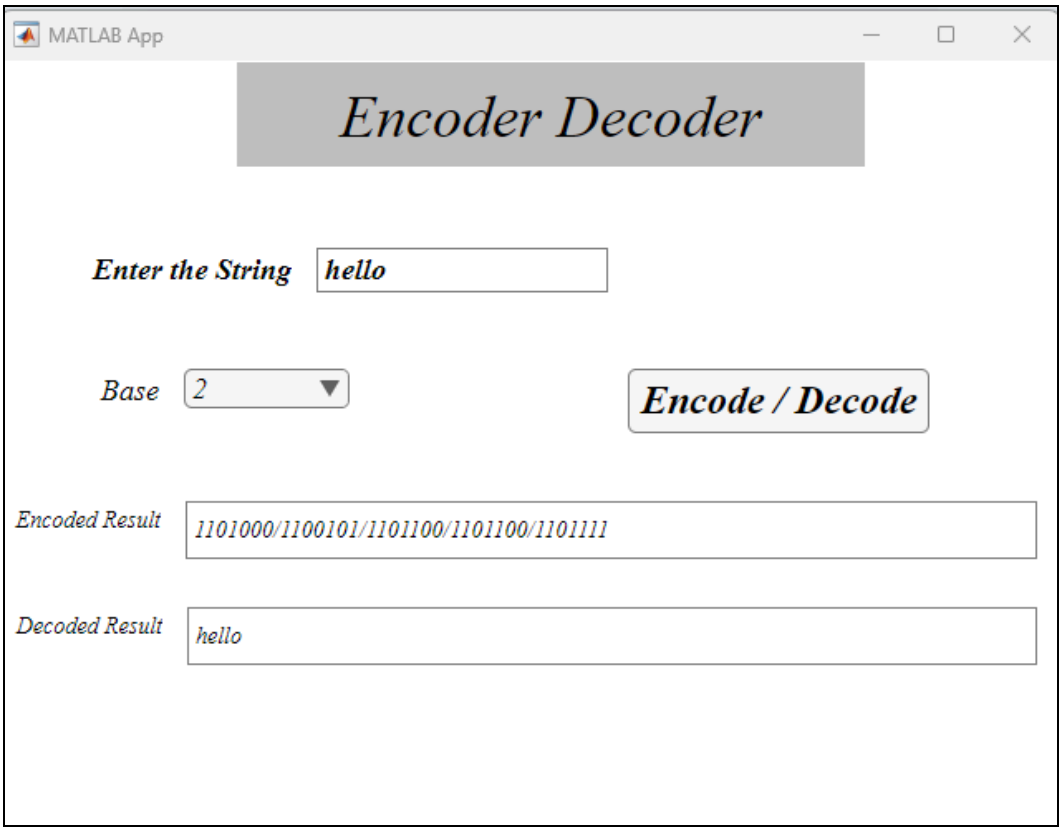
- The "EncodedResultEditField" UI component is updated in the app with the result of the encoding operation.

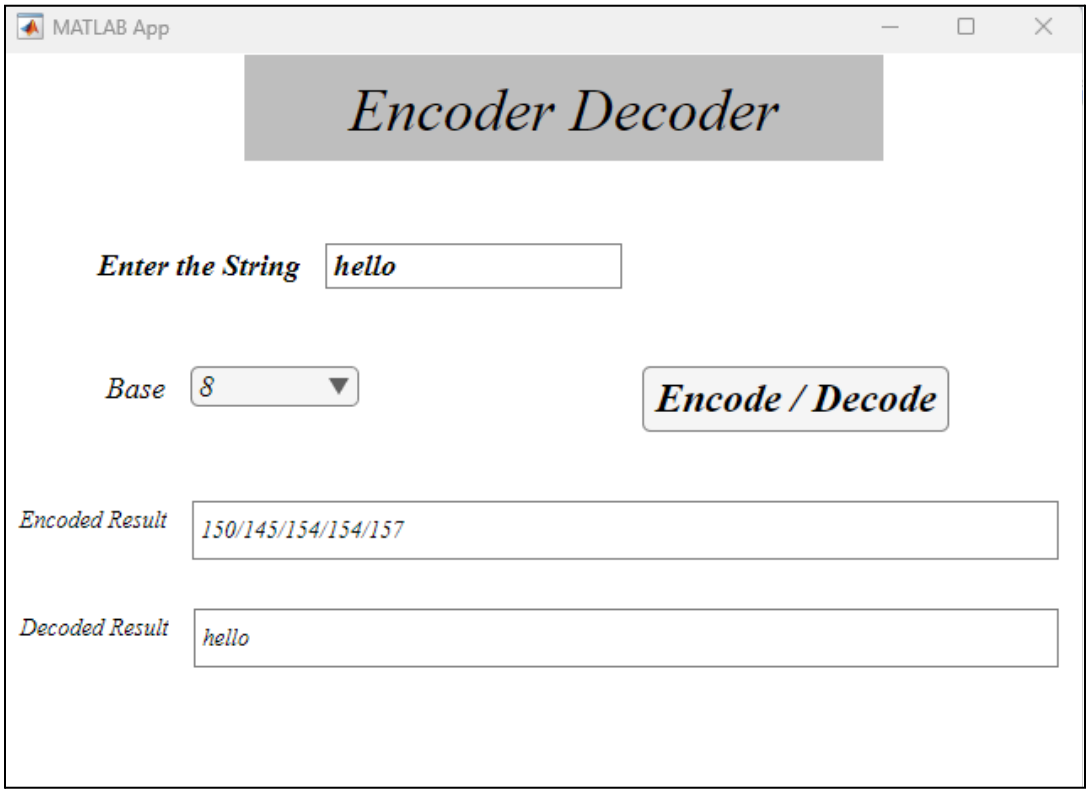
DecodeString Function:

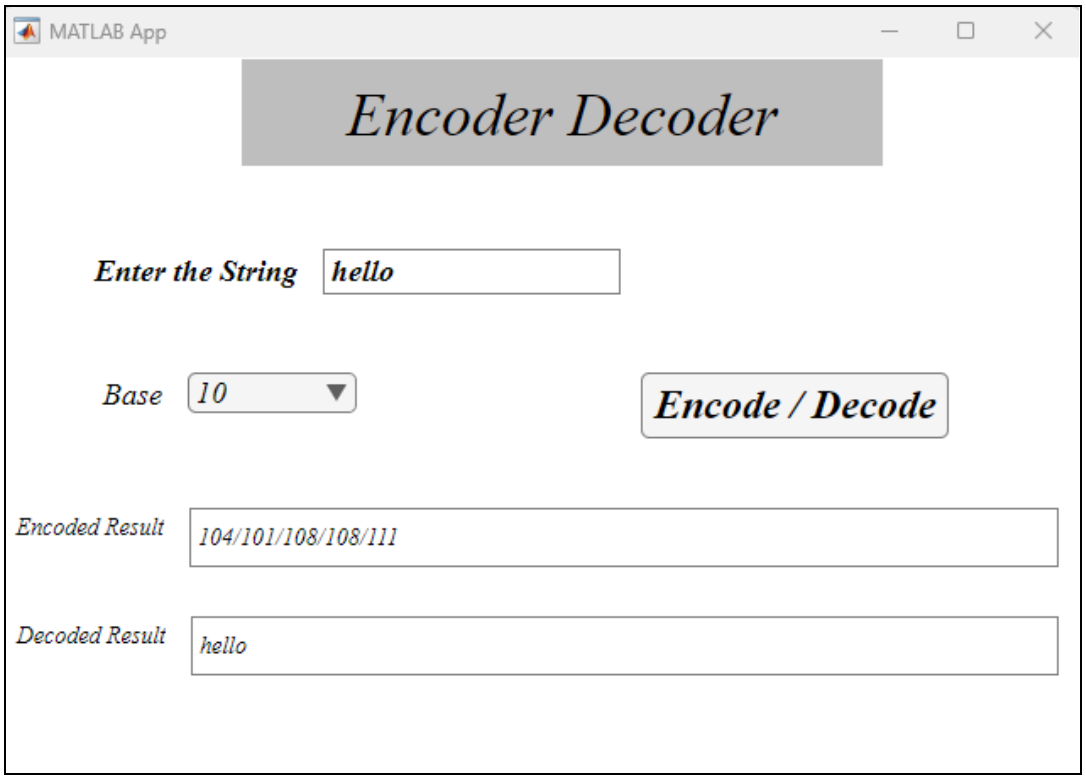
- The DecodeString function is called with the encoded string and base values.
- encodedStr is the string to be decoded.
- str2double(baseNo): Converts the base from a string to a numeric value.
- decodedStr: Stores the result of the decoding operation.
- Result UI Component (Decoding) is updated.

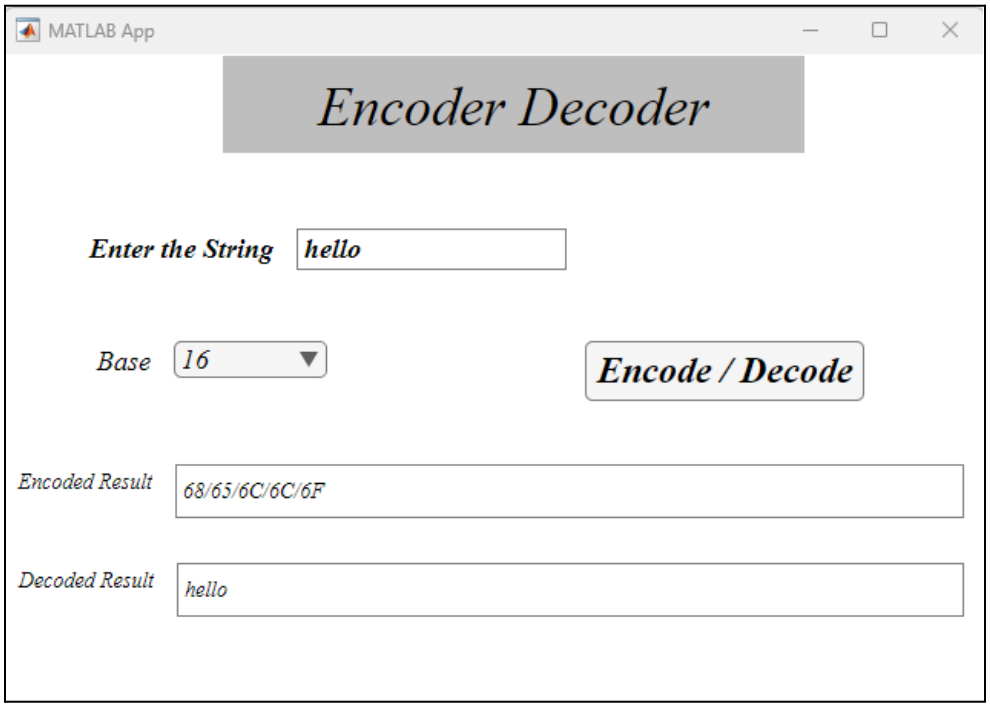
We update the "DecodedResultEditField" UI component in the app with the result of the decoding operation.

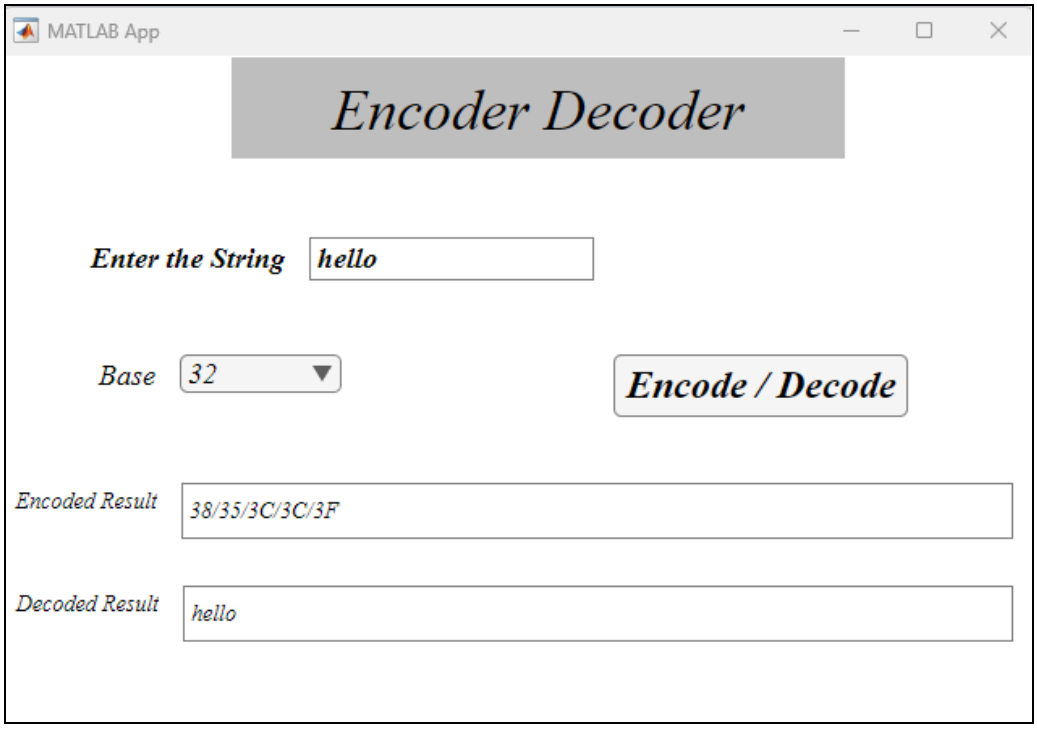
```
classdef app4 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: EncodeDecodeButton
        function EncodeDecodeButtonPushed(app, event)
            textToEncode=app.EntertheStringEditField.Value;
            baseNo=app.BaseDropDown.Value;
            type EncodeString.m
            encodedStr = EncodeString(textToEncode,str2double(baseNo));
            app.EncodedResultEditField.Value=encodedStr;

            type DecodeString.m
            decodedStr = DecodeString(encodedStr,str2double(baseNo));
            app.DecodedResultEditField.Value=decodedStr;
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app) ...
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app4 ...

        % Code that executes before app deletion
        function delete(app) ...
    end
end
```


MATLAB App
Encoder Decoder
Enter the String
hello
Base
2
Encode / Decode
Encoded Result
1101000/1100101/1101100/1101100/1101111
Decoded Result
hello


MATLAB App
Encoder Decoder
Enter the String
hello
Base
8
Encode / Decode
Encoded Result
150/145/154/154/157
Decoded Result
hello


MATLAB App
Encoder Decoder
Enter the String
hello
Base
10
Encode / Decode
Encoded Result
104/101/108/108/111
Decoded Result
hello


MATLAB App
Encoder Decoder
Enter the String
hello
Base
16
Encode / Decode
Encoded Result
68/65/6C/6C/6F
Decoded Result
hello



GeoCoding Coordinate Conversion

ConvertButtonPushed Function:

- This function is triggered when a button is pushed in the MATLAB App. Converts the provided decimal degree value to DMS format using the convertDecimalToDMS function.
- It retrieves the decimal degree value from the app's UI element. It calls the convertDecimalToDMS function and displays the resulting DMS value in the app's UI element.

convertDecimalToDMS Function:

- This function extracts the integer and fractional parts and converts the fractional part to minutes and seconds using base conversion.
- It applies base conversion to degrees, minutes, and seconds separately and formats and returns the DMS representation.

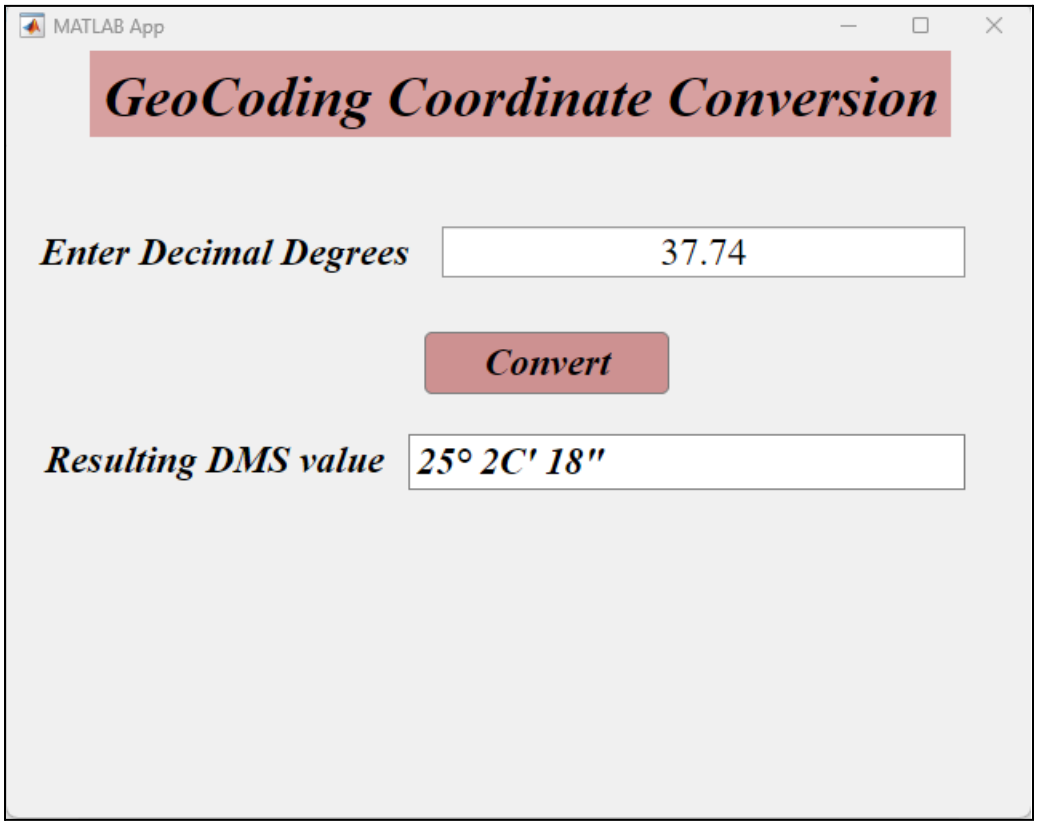
Usage:

- The ConvertButtonPushed function is an event handler triggered by a button in a MATLAB App.
- The decimal degree value is retrieved from a UI element (app.EnterDecimalDegreesEditField).

- The convertDecimalToDMS function is called with the decimal degree value.
- The resulting DMS value is displayed in a UI element (app.ResultingDMSvalueEditField).

Interaction with convertDecimalToDMS:

- The convertDecimalToDMS function is responsible for the actual conversion.
- It utilizes base conversion internally to achieve the conversion of degrees, minutes, and seconds to the desired format.



```
classdef app5 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public) ...

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: ConvertButton
        function ConvertButtonPushed(app, event)
            decimalDegrees=app.EnterDecimalDegreesEditField.Value;
            type convertDecimalToDMS.m
            dms = convertDecimalToDMS(decimalDegrees);
            app.ResultingDMSvalueEditField.Value=dms;
        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app) ...
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = app5 ...

        % Code that executes before app deletion
        function delete(app) ...
    end
end
```

The resulting DMS representation is displayed in a designated UI element. The ConvertButtonPushed function serves as a user interface event handler for triggering the conversion of decimal degrees to DMS format. It interacts with the convertDecimalToDMS function, which is responsible for the core conversion logic, utilizing base conversion for flexibility and customization. The resulting DMS representation is displayed within the MATLAB App, enhancing user experience in handling geographic coordinates.

REFERENCES

Rawat, Saurabh & Nautiyal, Bhaskar & Sah, Anushree. (2012). A Different and Realistic Approach to Inter Base Conversion for Number System. International Journal of Computer Applications. 52. 37-44. 10.5120/8306-1941.

https://www.geeksforgeeks.org/convert-a-number-from-base-a-to-base-b/

https://www.researchgate.net/post/Why-do-we-need-various-number-base-conversions-number-system-conversions-eg-octal-to-hexadecimal

https://www.academia.edu/73383752/Number_Systems_Base_Conversions_and_Computer_Data_Representation

MATLAB for Graphical User Interface (GUI) Design | by Iiesbangalorebl | Medium