FANUC PICTURE
Graphic drawing library

Specification

Edition	Date	Person in charge	Changes			
Created		2024.03.08	Person in charge		Approved	

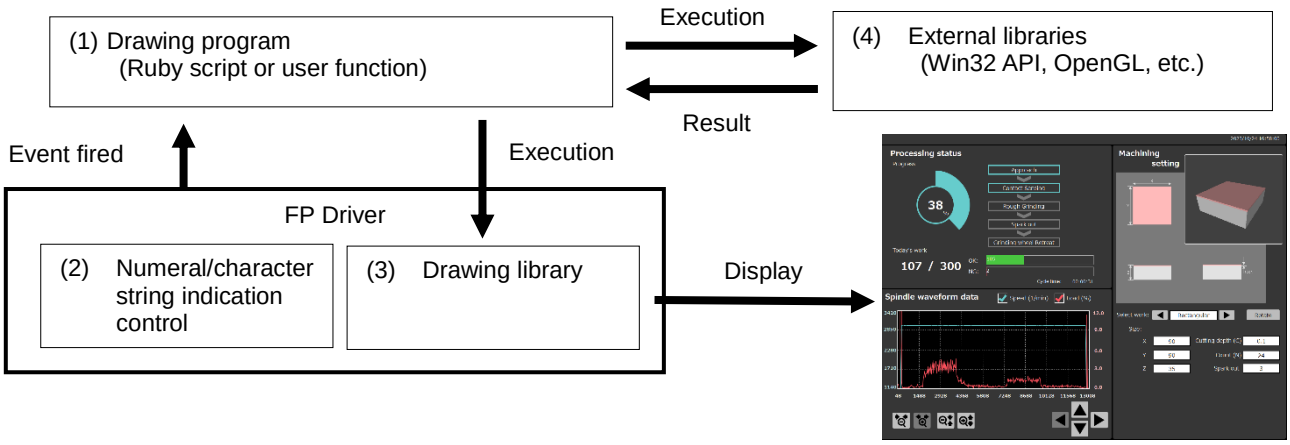
Name	FANUC PICTURE Graphic drawing library Specification
Drawing number	A-42148-00830EN/01
FANUC CORPORATION	Page 1/82

TABLE OF CONTENTS

1 OVERVIEW

This function allows users to display their own graphic drawing on the screen customized with FANUC PICTURE.
By creating a drawing program that executes drawing library, display users own graphic drawings on the numeral/character string indication control.
Drawing programs are executed when screen display or other events (drawing events) are fired.

(1) Create a drawing program with Ruby script or user function.
(2) If a drawing program is specified in the action of the numeral/character string indication control, the drawing program is executed when a drawing event is fired.
(3) Display the user's own graphic drawing when the drawing library in the FP driver is executed by executing a drawing program.
(4) External graphics libraries such as Win32 API or OpenGL can be executed by executing a drawing program.



NOTE

1 This function is included in FANUC PICTURE disk (A08B-9010-J518#ZZ11) 11.1 or later.
2 This function can be used with PANEL *i*H Pro and FANUC *i*PC.
When used with PANEL *i*H, error message "Function name is illegal" is displayed in FP driver.
3 For FANUC PICTURE and FP drivers, refer to FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09).
4 For Ruby script, refer to FANUC PICTURE Specification (Edition 8.0 or later) (A-42146EN-042/31) "4 RUBY SCRIPT".
5 For user function, refer to FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09) "4 EMBEDDING OF C APPLICATIONS" and FANUC PICTURE Specification (Edition 06.2 to less than 08.0) (A-40712EN/55) "3.9 Built in the user making application".
6 For the numeral/character string indication control, refer to "2.3.14 Numeral/Character String Indication Control" in the FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09).

Precautions when using this function

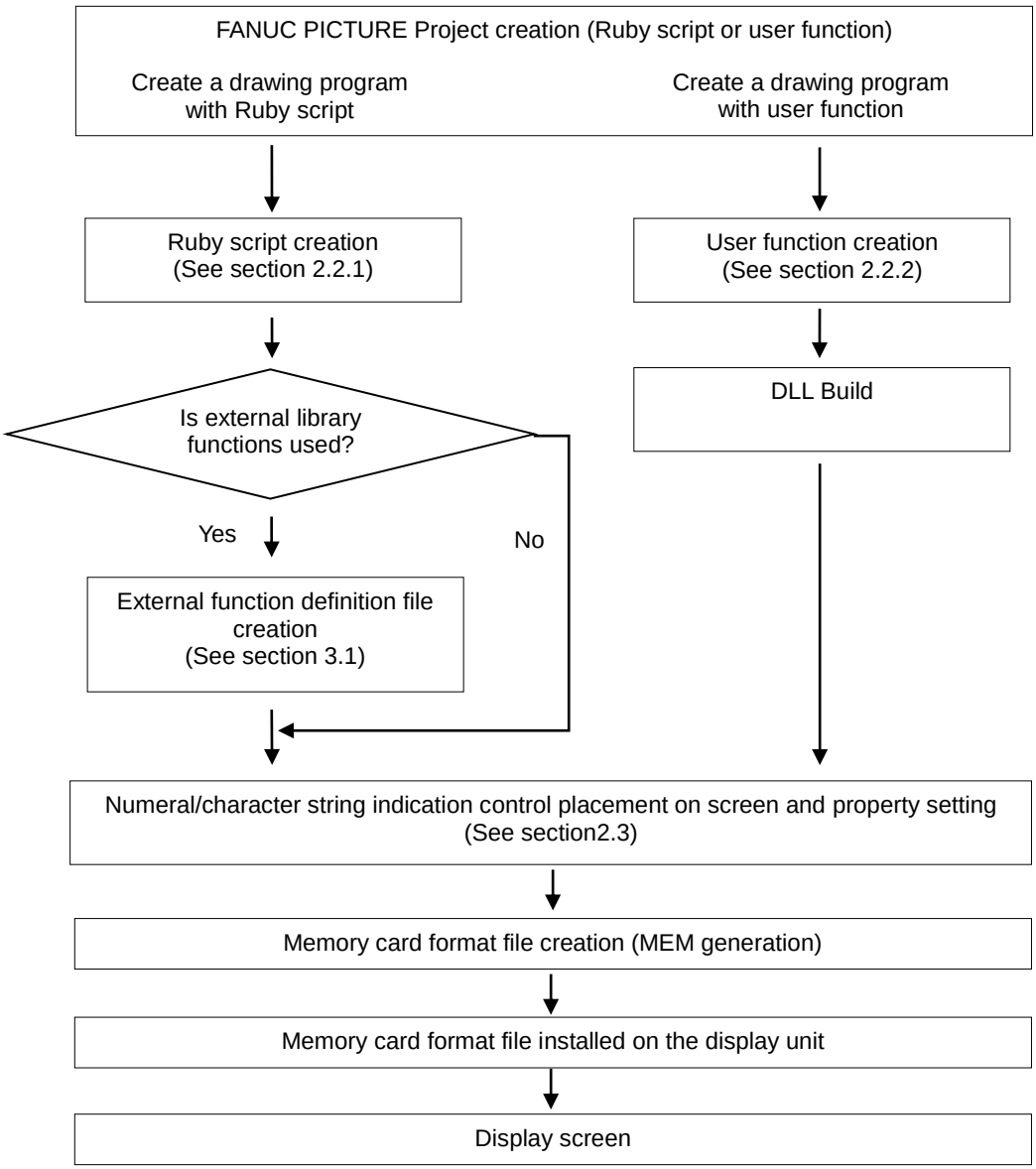
When using this function, note the following points.

1. When using this function, make sure that the user has well checked the operation.
2. When we support this function, we will ask to provide your data of FANUC PICTURE and CNC. Without the data, we cannot support it.

2 SCREEN CREATION

2.1 PROCEDURE

The procedure for creating screens using this function is outlined below.



NOTE

1 In the user function, functions of external libraries can be used as before. For the user function, refer to FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09) "4 EMBEDDING OF C APPLICATIONS" and FANUC PICTURE Specification (Edition 06.2 to less than 08.0) (A-40712EN/55) "3.9 Built in the user making application".
2 For the project creation, refer to FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09) "2.2.7 Creating New Project".
3 For the memory card format file creation, refer to FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09) "2.2.11 Make MEM file".
4 For the way of installing and displaying the screen on the display unit, refer to the FANUC PICTURE Specification (Edition 06.2 to less than 08.0) (A-40712EN/55) included on the CD.

2.2 DRAWING PROGRAM CREATION

Drawing program can be created with Ruby script or user function.
See the table below and select which method you would like to use to create drawing program.

	Ruby script	User function
Programming language	mruby	C++
Precautions	Some restrictions on using external graphics libraries (NOTE 1)	C/C++ compiler is required separately (NOTE 2)

NOTE
1 For the external graphics libraries that cannot be used with Ruby script, see "3.1 EXTERNAL FUNCTION DEFINITION FILE".
2 The C/C++ compiler is compatible with Visual Studio 2008 or later. For details, refer to FANUC PICTURE Specification (Edition 06.2 to less than 08.0) (A-40712EN/55) "3.9 Built in the user making application".

Using the following digital clock display as an example, we will explain how to create drawing program with each method.

- Displays the time in hours, minutes, and seconds.
- The background is transparent and overlays either the screen background or the controls in the background.

2.2.1 Ruby script

The followings explain how to create a drawing program with Ruby script.
Also, you can describe code of defining the drawing program (class creation process) and code of executing the defined drawing program (class execution process) into separated scripts.
In this example, the code of class creation process and class execution process are described in one script.

1. Create a new Ruby script in the screen editor. In this example, create script No.100.

NOTE
For the method of creating Ruby script, refer to "4 Ruby Script" in FANUC PICTURE Specification (Edition 8.0 or later) (A-42146EN-042/31).

Class creation process

2. Create a class that inherits from FpUserControl. In this example, we create a DigitalClock class.

```
class DigitalClock < FpUserControl
end
```

NOTE
For the FpUserControl class, see "5.4 CONTROL".

3. Implement methods of the mruby class that are called when drawing events is fired in the DigitalClock class. In this example, we will implement methods that are called at the start of displaying screen, at screen drawing, and every timer cycle.

```
class DigitalClock < FpUserControl
    def initialize() # Start of displaying screen
        super
        self.timer_interval = 1000 # timer cycles 1 second (=1000ms)
    end
    def paint() # Screen drawing
        s = ctime(time()[1])[1].slice(11, 8) # s = hh:mm:ss
        x, y, w, h = rect
        # Create font
        font = FpFont.new("Arial", 32)
        # Display time
        draw_text(x, y, x + w, y + h, FP_TEXT_CENTER, s, font, fpcolor(255, 255, 255))
        # Release font
        font.dispose()
    end
    def timer() # Every timer cycle
        restore_rect() # Redraw background
        paint()
        update_rect() # Update drawing
    end
end
```

The following methods are called when drawing events are fired.

Method	Description	Precautions (subsection 2.2.4)
Constructor (initialize)	This method is called when creating an instance of the class. Create one as needed. This method is called mainly at the start of screen display.	2,3
dispose	This method is called at the end of a class instance. Create one as needed. This method is called mainly at the end of screen display.	3
paint	This method is called when the screen is displayed. Be sure to create one. Create a drawing process using drawing methods in the FpUserControl class or an external graphics library such as Win32 API.	4,5
timer	After a timer cycles (in ms) is set in the timer_interval instance variable of the FpUserControl class, this method will be called every timer cycles while the control is displayed. Create one as needed. This method can also be used to update the display by calling the paint method and so on. **NOTE** 1 When you update the display in this method, execute update_rect method to update display area. 2 When updating the display in this method, either overwrite all updated area or clear the drawing contents using the restore_rect method before updating the display.	6,7,8,9, 10,11

NOTE

For the precautions on each method, see "2.2.4 Precautions". The number in the "Precautions" column in the table above means the reference number for the method.

Class execution process

4. Make an instance of the DigitalClock class with the new method and execute it with the start method.
   By this, the instance of the class resides in memory and the methods implemented in step 3 are executed when drawing events are fired.
   In this example, the code of this process is described after one of the class creation process.

```
class DigitalClock < FpUserControl
    def initialize() # Start of screen display
    :
    (omitted)
end
DigitalClock.new().start()
```

NOTE

1 The code of the class execution process can be created in a separate script from a script in which the code of the class creation process is described. When creating the codes in separated scripts, the class creation process must be executed before the class execution process.
2 For the code of the class creation process, codes for multiple classes can be described in one script such as the startup script.

2.2.2 User function

The followings explain how to create a drawing program with a user function.

1. Find the template Visual Studio project in the folder "Sample¥iHMI¥UserFunc" in the FANUC PICTURE installation folder. Copy this folder to a disk in your development PC.

2. Open the solution file "UserFunction.sln" in Visual Studio.

NOTE
1 Solution and project files are for Visual Studio 2008. When using a newer version of Visual Studio, upgrade solution and project by the upgrade function that is launched when a former solution file is opened in Visual Studio.
2 The project "UserFunction for ARMv4I" for PANEL *i*H should be deleted during the upgrade.

Class creation process

3. Create a class that inherits from FpUserControl. In this example, we create a DigitalClock class.

```
#include "FPDrawLib.h"
class DigitalClock: public FpUserControl
{
};
```

NOTE
Be sure to include the header file "FPDrawLib.h". For the FpUserControl class, see "5.4 CONTROL".

4. Implement method of the C++ class that are called when drawing events are fired in the DigitalClock class. In this example, we will implement methods that are called at the start of screen display, at screen drawing, and every timer cycle.

```
class DigitalClock : public FpUserControl
{
    void paint() // Screen drawing
    {
        char buf[32];
        _strtime_s(buf, _countof(buf));
        FP_RECT r = getRect();
        // Create font
        FpFont font("Arial", 32);
        // Display time
        drawText(r.x, r.y, r.x + r.width, r.y + r.height, FP_TEXT_CENTER, buf, font, fpColor(255, 255, 255));
    }
    void timer() // Every timer cycle
    {
        restoreRect(); // Redraw background
        paint();
        updateRect(); // Update drawing
    }
public:
    // Constructor: start of screen display
    DigitalClock(void* context, char* param)
```

```
        : FpUserControl(context, param)
    {
        setTimerInterval(1000); // # timer cycles 1 second (=1000ms)
    }
};
```

The following methods are called when drawing events are fired.

Method	Description	Precautions (section 2.2.4)
constructor	This method is called when creating an instance of the class. Be sure to create one. This method is called mainly at the start of screen display. The arguments of this method are as follows void* context Pointer to identify the control char* param User function argument **NOTE** Be sure to pass the arguments of this method to constructor of the base class FpUserControl.	2,3
dispose	This method is called at the end of a class instance. This method is called mainly at the end of screen display. Create one as needed.	3
paint	This method is called when the screen is displayed. Be sure to create one. Create a drawing process using drawing methods in the FpUserControl class or an external graphics library such as Win32 API.	4,5
timer	After the timer cycle (in ms) is set by the setTimerInterval method of the FpUserControl class, this method will be called every timer cycle while the control is displayed. Create one as needed. This method can also be used to update the display by calling the paint method and so on. **NOTE** 1 When you update the display from this method, execute updateRect method to update display area. 2 When updating the display from this method, either overwrite all updated area or clear the drawing contents using the restoreRect method before updating the display.	6,7,8,9, 10,11

NOTE
For the precautions on each method, see "2.2.4 Precautions". The number in the "Precautions" column in the table above means the reference number for the method.

Class execution process

5. Create a user function. In this example, we create the usr_DrawClock function.

```
#include "USERFUNC.h"
USERFUNCTION long usr_DrawClock(void* context, int msg, char* param)
{
```

```
}
```

The arguments of the user function are as follows

void* context	Pointer to identify the control
int msg	Message to the control (reserved, used internally)
char* param	User function argument

NOTE

1 The first letter of the function name must be "usr_". If the first letter of the function name is not "usr_", the FP driver will display the error message "Function name is illegal".
2 Be sure to include the header file "USERFUNC.h".

6. In the user function you create, make an instance of the DigitalClock class created in the class creation process with the new operator and execute the start method. By this, the instance of the class resides in memory and the methods implemented created in step 4 are executed when drawing events are fired.

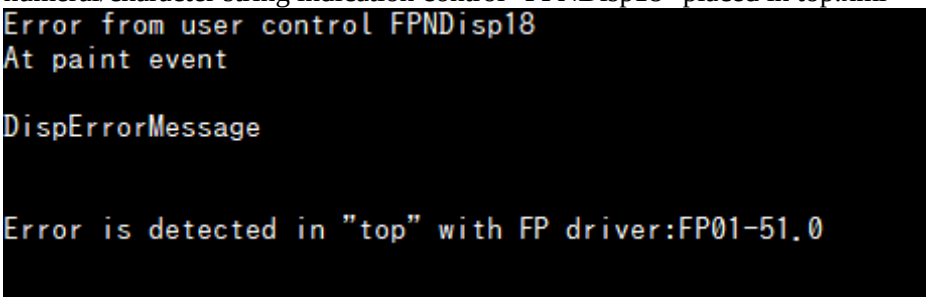
```
USERFUNCTION long usr_DrawClock(void* context, int msg, char* param)
{
    return (new DigitalClock(context, param))->start();
}
```

The user function arguments context and param must be passed to the constructor of the DigitalClock class created during the class creation process.
Also, return the return value of the start method as the return value of the user function.

2.2.3 Anomaly handler creating

An anomaly process can be created in case something goes wrong in the drawing program.

An error message like below is displayed in the drawing program, then the FP driver can be stopped updating display.
Example)
When an error message "DispErrorMessage" is displayed in the paint method of the drawing program of the numeral/character string indication control "FPNDisp18" placed in top.xml



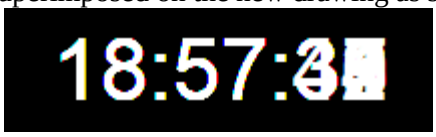
The error message is specified by the member function print_error (Ruby script) or printError (user function) of the FpUserControl class.
For details, see "print_error [mruby], printError [C++]" in "5.4.3 Member function"

NOTE

1 Error messages will be displayed in the FP driver in addition to user-created drawing program. For the error messages displayed in the FP driver, refer to "A.1 Error Messages" in the FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09).
2 If an error message is displayed, the FP driver must be terminated once. For the termination method, refer to the FANUC *i*HMI Home Screen OPERATOR'S MANUAL (B-64644EN).

2.2.4 Precautions

1. In Ruby script, only alphanumeric characters and one-byte symbols can be used in strings. Characters other than those listed above, such as European characters, Japanese, and Chinese, are not supported. This limitation is due to the mruby specific specification, which, unlike Ruby, does not support character encoding.
2. By executing the start method of the FpUserControl class, the instance resides in memory. After this, the Ruby script or user function that called the start method will not be executed until it is determines that it is necessary to create an instance again.
3. When the FP parameter No.6 "D-RAM capacity saving function" is disabled and the property "Free Memory" of the screen structure definition control is disabled, an instance of the FpUserControl class resides permanently in memory, so the constructor after second time and the dispose method are never called.
   For the FP parameters, refer to "2.2.11.6 FP-PARAMETER SETTING screen" in the FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09). Also, for screen structure definition control, refer to "2.2.3 Screen Structure Definition Control" in the FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09).
4. The paint method of the FpUserControl class will be called multiple times when the screen is displayed. Create the process of the paint method assuming that it will be called multiple times.
   In addition to screen switching, this method will be called to update the display when the figure of itself or other controls is changed, such as when the 7-color background is changed.
5. The longer the processing time of the paint method of the FpUserControl class, the longer it takes to display the screen, design the paint method so that the data used in the paint method is collected in a method other than the paint method, and only the results are displayed in the paint method.
6. The timer method is called at the timing of the FP driver's screen updates. Therefore, the timer accuracy depends on the update speed of the screen being displayed, and may be called at intervals greater than the set cycle.
7. The paint and timer methods are called only when the screen on which the control that executes the drawing program is placed is displayed. However, if the control is set NoAction, the above methods will not be called. Also, while the custom screen is displayed on the back of other application, if the FP driver is running, the above methods will be called on the last custom screen displayed.
8. If you update the display from the timer method, execute the update_rect or updateRect method. Otherwise, the display will not be updated. For details on the method, see "update_rect [mruby], updateRect [C++]" in "5.4.3 Member function".
9. When the timer method is called, the previously drawn display contents are not cleared and are maintained. When updating the display, clear the previous display by overwriting all updated areas or by redrawing the background using the restore_rect or restoreRect method. If not cleared, the previous drawing will be superimposed on the new drawing as shown below.

   For details on the method, see "restore_rect [mruby], restoreRect [C++]" in "5.4.3 Member function".
10. Execute update_rect or updateRect method and restore_rect or restoreRect method only when screen update is necessary. Frequently executing methods cause an increase in memory usage.
11. If only restore_rect or restoreRect method is executed, the screen may not be updated. Be sure to execute them together with update_rect or updateRect method.
12. The position and size specified in the drawing program will not be enlarged or shrunk by the "automatic screen enlarged display function". When using the "automatic screen enlarged display function", get the control size before / after scaling from the FpUserControl class, and enlarge or shrink the coordinates and font size in the drawing program itself. For the "automatic screen enlarged display function", refer to "28 Automatic screen enlarged display function for iHMI" in the FANUC PICTURE Specification (Edition 06.2 to less than 8.0) (A-40712EN/55) included on the CD.

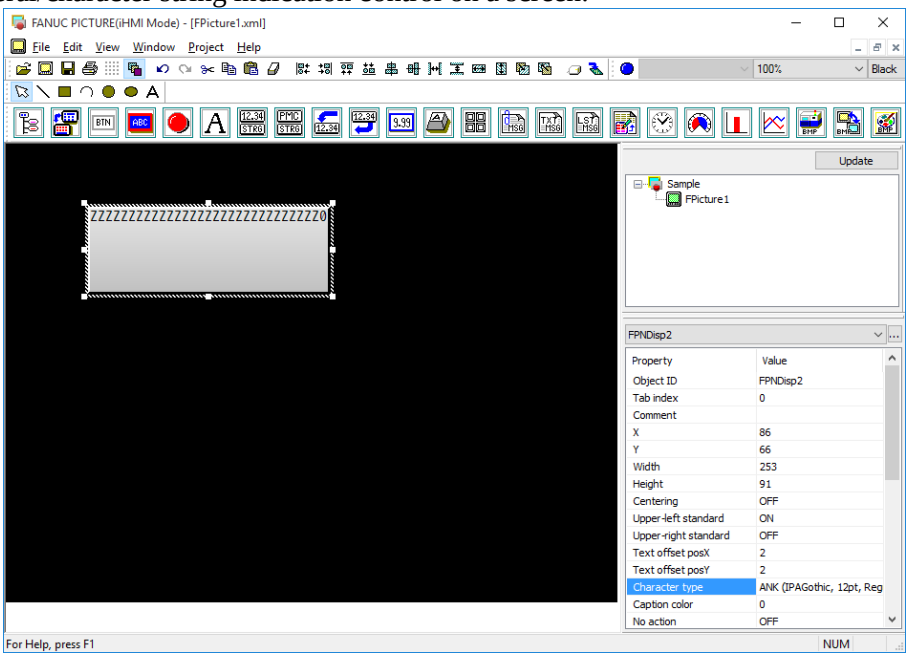
13. Use the source files "FPDrawLib.cpp" and "FPDrawLib.h" included in the template Visual Studio project without changing them with no special reason. These are provided as source file form for debug by user only, reverse analysis of them or inquiries about its internal contents of them are strictly prohibited.

2.3 SCREEN EDITING

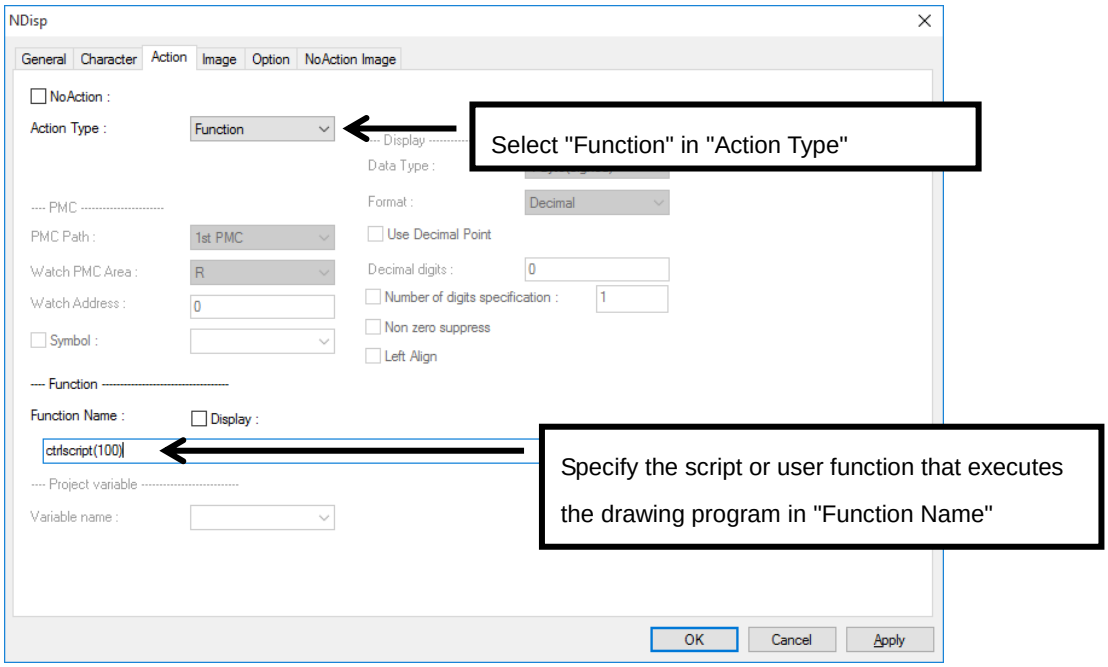
Place a numeral/character string indication control in the screen editor and set a drawing program describing the class execution process to the properties of this control.
The followings introduce how to set up the controls by using drawing program created in subsection 2.2.

2.3.1 Control setting

1. Place a numeral/character string indication control on a screen.



2. Open the Properties dialog box of the placed numeral/character string indication control and set the properties on the "Action" tab as follows.



The function name is set as follows.

Ruby script
ctrlscript(sno(,arg0)(,arg1)...)

Argument
sno
Script number
arg0,arg1...
Arguments to script

Remarks
Up to five arguments to script can be specified and are optional.
The value that can be specified for the argument is an integer, and an integer-type project variable can also be specified. For the project variable, refer to FANUC PICTURE Specification (Edition 8.0 or later) (A-42146EN-042/31) "18 Project variable management function of FANUC PICTURE for *i*HMI".
Arguments can be get in scripts by argument reference variables (ARG variables). For details, refer to FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09) "2.2.16.4 FP script commands (operators, variables, statements)".

Example
```
ctrlscript(100,1,-10)
```

User function
ctrlfunc(name(,arg0)(,arg1)...)

Argument
name
User function name
arg0,arg1...
Arguments to user function

Remarks
Arguments that can be specified for user function are limited to 23 characters, including separators and are optional. For details, refer to "4.2 CREATION OF APPLICATIONS USING TASKS" in the FANUC PICTURE OPERATOR'S MANUAL (B-66284EN/09).
The value can be a string, and the string must be enclosed in double quotation marks (") if it contains white space. Project variable of integer, real, and string type can also be specified, but variables are converted to strings and passed to user function. Only one project variable can be used as an argument. For the project variable, refer to the FANUC PICTURE Specification (Edition 8.0 or later) (A-42146EN-042/31) "18 Project Variable Management Function of FANUC PICTURE for *i*HMI".
The arguments can be referenced by the FpUserControl class member variable "parameter". For details, see "Constructor" in "5.4.3 Member function".

Example
```
ctrlfunc(usr_DrawClock, "text string")
```

NOTE
1 Except for user function arguments, the "Function Name" setting cannot contain spaces.
2 In the "Action" tab of the numeral/character string indication control properties dialog box, if ctrlscript or ctrlfunc is specified for the function name, the following setting values are disabled on the FP driver.
Setting values in the "Character" tab
Setting values other than "NoAction" and "Function Name" in the "Action" tab

2.3.2 Superimposing background and control

The figure of the control, which can be specified in the "Image" tab of the numeral/character string indication control properties, can be used as the background.
When "No Figure" is specified for the figure, the background of the control becomes transparent. In this case, the following controls can be superimposed on the back of the control.

- Label control (display string must be empty)
- Image display control
- Drawing controls (line, rectangle, arc, circle, ellipse)

Example) Superimposing the digital clock created in "2.2 DRAWING PROGRAM CREATION" and a label control.

NOTE
No controls can be superimposed in front of the numeral/character string indication control.

You can also use the 7 background colors as a background by setting the properties "Options" tab.

2.3.3 Precautions

1. A drawing program can be specified in the numeral/character string indication control only. Even if you specify it for other controls, it will not cause an error when generating the MEM, and the error message "Function name is illegal" will be displayed on the FP driver.
2. The "NoAction Image" tab of the numeral/character string indication control properties allows you to set the mask style. However, when in the invalid state, the drawing program is not executed and drawing over the mask style is not possible.

3 USING EXTERNAL LIBRARIES WITH RUBY SCRIPT

This chapter contains procedures for creating drawing programs with Ruby script and using external libraries.

By creating an external function definition file (funcdef.txt) in the FPScript folder of your project, Ruby script can execute Windows API functions or functions in external dynamic link libraries (DLLs) outside of FANUC PICTURE.

NOTE
1 32-bit DLLs can be used. 64-bit DLLs cannot be used.
2 Cannot be applied to functions that use callback functions.

3.1 EXTERNAL FUNCTION DEFINITION FILE

In the external function definition file, define the function names, arguments, structures, and constants of the Windows API or external library that you want to use according to the following rules.

Example of external function definition file.

```
struct LOGFONT {
    LONG lfHeight; LONG lfWidth; LONG lfEscapement; LONG lfOrientation; LONG lfWeight;
    BYTE lfItalic; BYTE lfUnderline; BYTE lfStrikeOut; BYTE lfCharSet; BYTE lfOutPrecision;
    BYTE lfClipPrecision; BYTE lfQuality; BYTE lfPitchAndFamily;
    string lfFaceName[32];
};

const DEFAULT_CHARSET = 1;
const CLEARTYPE_QUALITY = 5;

function HGDIOBJ SelectObject library "gdi32" (HDC, HGDIOBJ);
function BOOL DeleteObject library "gdi32" (HDC, HGDIOBJ);
function HFONT CreateFontIndirect entry CreateFontIndirectA library "gdi32" (LOGFONT*);
```

Structure

Constant

Function

NOTE
1 Only ASCII character encoding can be used in the external function definition file.
2 MEM generation may fail if you use spaces, tabs, or line breaks in the middle of words such as function names or variable names.
3 As the size of the external function definition file increases, the size of the screen data also increases. Do not write definitions that are not used.

3.1.1 Data type used in the definition

The following data type can be used in the external function definition file.
You can also use redefined data type. For information on how to redefine data type, see "3.1.6 Redefinition of data type".

Data type	Description
int8_t	Signed 8-bit integer
uint8_t	Unsigned 8-bit integer
int16_t	Signed 16-bit integer
uint16_t	Unsigned 16-bit integer
unt32_t	Signed 32-bit integer
uint32_t	Unsigned 32-bit integer
bool	Boolean (1 byte)
float	Single precision floating point
double	Double precision floating point
string	ASCII character string (alphanumeric characters and one-byte symbols only)
wstring	UNICODE (UTF-16) character string (alphanumeric characters and one-byte symbols only)
void	No function return value
void*	C language pointer NULL pointers are denoted as NULL or nullptr

NOTE

The data type used in the description of the external function definition file must be the above or redefined above. For the default redefined type, see "3.1.6.1 Built-in definition".

3.1.2 Description rules for type

The type specified for member variables of structures, arguments of functions and so on should be described by the following rules.

Type-specifier:
Sign Type * Identifier [ArrayNumber]

Sign (optional)
Sign
When Type is an integer type, specify the sign by signed or unsigned.

Type
Data type
You can specify the type described in "3.1.6.1 Built-in definition", redefined data type, and structures.

* (optional)
Pointer declarator
Pointer to pointer (**) cannot be specified. If specified, it is treated as a C language pointer (void*) in Ruby script.

Identifier
Identifiers such as variable names (up to 64 characters)
Multiple identifiers cannot be specified at once. If specified, an error will occur during MEM generation.
e.g.) unsigned int uint, *puint // error

ArrayNumber (optional)
Number of arrays

For string, specify the length of the string including the NULL character. For other than string, the number of elements of the array.
Note that it is not possible to specify a pointer declarator at the same time, or to specify a multidimensional array like Identifier[2][3].
If specified, an error will occur during MEM generation.

Example

```
unsigned int value // unsigned integer
double * pvalue // pointer
int array[10] // array
string str[8] // 8-character string
```

3.1.3 Structure definition

Structures can be used by describing according to the following rules.
It can also be used to redefine data type. For the method to redefine data type, see "3.1.6 Redefinition of data type".

```
struct StructName {
   Member;
   Member;
   :
   :
};
```

StructName
Structure name (up to 64 characters)
Member
Member variable
Specify the member variable name in the Identifier of type-specifier according to the type-specifier in "3.1.2 Description rules for type".

Example

```
struct GdiplusStartupInput {
      UINT32 GdiplusVersion;
      void* DebugEventCallback;
      BOOL SuppressBackgroundThread;
      BOOL SuppressExternalCodecs;
};
struct GdiplusStartupOutput {
      void* NotificationHook;
      void* NotificationUnhook;
};
```

NOTE

1 Structures already defined by default do not need to be redefined. For type that is predefined by default, see "3.1.6.1 Built-in definition".

2 The members of the structure cannot contain pointers other than the C language pointer type (void*).

```
struct A {
    void* m1; // OK
    int *m2; // NG
};
```

3 Structures cannot be nested.

```
struct A {
    int m1;
    struct B {
        int m2;
    }; // NG
};
```

The structures should be declared individually as follows.

```
struct B {
    int m2;
};
struct A {
    int m1;
    B b; // OK
};
```

4 Declared strings as string or wstring type, and be sure to specify the string length.

```
struct LOGFONT {
    LONG lfHeight;
    :
    string lfFaceName[32];
};
```

String should be assigned in Ruby script as follows.

```
Logfont.lfFaceName = "Arial"
```

When the function is executed, character code conversion from mruby to C language is automatically performed, and a string to which a terminating NULL character is appended is assigned to the member variable. However, if the length of the string to be assigned exceeds the specified string length, a terminating NULL character is not appended.

5 You cannot specify the alignment of a structure. It follows the default alignment rules of the C language.

3.1.4 Constant definition

Define constants according to the following rules.

const Identifier = Value;

Identifier
Constant identifier (up to 64 characters)

Value
Integer, floating point or string (up to 512 characters)

Integer
Number strings such as "100" and "12", hexadecimal numbers starting with 0x or 0X.

Floating point
Number strings with a decimal point, such as "1.23"
It can also be expressed such as "0.123e1" with an exponential part by E or e.

String
String enclosed in double quotes, such as "FANUC".
Consecutive multiple strings are concatenated and set as constants during MEM generation. (Example: "FANUC PICTURE" = "FANUC " "PICTURE")
The special characters that can be specified in a string are as follows.

Quotation marks	¥'
Double quotes	¥"
Backslash	¥¥
New line	¥n
Return	¥r
Tab	¥t

Example

```
const START = -100;
const END = 0x200;
const STEP = -0.1;
const PI = 314e-2;
const Sample = "Text Sample";
const MultiLine =
        "FANUC"
        "PICTURE";
```

3.1.5 Function definition

Define functions according to the following rules.

function Sign Type CallSeq FuncName entry EntryName library "LibName" (IODir Param, IODir Param, ...);

Sign (optional)
  Sign
  When Type is an integer type, specify the sign by signed or unsigned.
Type
  Return type
  The data type that can be specified are integer, floating point, string (string or wstring type), void, and C language pointer (void*).
  Strings can only use functions whose return value is a pointer to a null terminal string.
  Pointers other than arrays, struct, and void* cannot be specified.
CallSeq (optional)
  Calling convention
  Specify cdecl or stdcall.
  Default when omitted is stdcall.
FuncName
  Function name (up to 64 characters)
EntryName (optional, can be omitted with entry specifier)
  Actual function name in DLL
  If omitted, FuncName is used as the function name in DLL.
LibName
  DLL file name
  The extension ".dll" can be omitted.
  The DLL search rules follow the Windows API LoadLibrary specification.
IODir (optional)
  Argument input/output direction
  Specify "in" for input to a function, "out" for output from a function, or "inout" for both.
  Default when omitted is "in"
Param.
  Argument
  Identifier of type-specifier can be omitted according to the type-specifier of "3.1.2 Description rules for type".
  If the input/output direction of an array, structure, or argument is "out" or "inout", be sure to specify passing-by-reference (passing-by-pointer).

Example

```
typedef int GpStatus;
typedef void* GpGraphicsPtr;
function GpStatus GdipCreateFromHDC library "gdiplus" (HDC hdc, out GpGraphicsPtr *graphics);
function GpStatus GdipDeleteGraphics library "gdiplus" (GpGraphicsPtr graphics);

const MAX_PATH = 260;
function DWORD stdcall GetCurrentDirectory entry GetCurrentDirectoryA library "kernel32" (
    DWORD nBufferLength,
    out string buffer[MAX_PATH]
);
```

NOTE

1 Functions that pass an array or structure by value as an argument cannot be used.
```
struct A { int m; };
function void func library "lib" (A a); // NG
```
2 When an "out" or "inout" specifier is specified for an argument, the argument must be pass-by-reference (pass-by-pointer).

3 To specify a string as an argument, declare the argument as a string or wstring type. This automatically performs a character code conversion from mruby to C language when the function is executed, and sets the argument to a string to which a terminating NULL character is appended. Although it is possible to specify a string length as described below, if the length of the string to be assigned exceeds the specified string length, a terminating NULL character will not be appended.

4 It is possible to specify a fixed-length string or array as an argument, as in the following example.
```
function void func library "lib" (string str[16]);
```
If the specified size differs from the actual size of the argument, the function is called after the argument is truncated or extended to the specified size inside the FP driver.

5 The return value of a string is automatically converted from C language to mruby character code when the function is executed. However, if the return value is NULL, mruby returns nil as the return value.

3.1.6 Redefinition of data type

When redefining a data type, describe according to the following rules.

typedef Sign Type * Identifier [ArrayNumber]

Sign (optional)
Sign
When Type is an integer type, specify the sign by signed or unsigned.

Type
Data type

* (optional)
Pointer declarator
Pointer to pointer (**) cannot be specified. If specified, it is treated as a C language pointer (void*) in Ruby script.

Identifier
New data type name to be defined (up to 64 characters)
Multiple Identifier cannot be specified at once. If you specify more than one Identifier at the same time, an error will occur during MEM generation.

ArrayNumber (optional)
Number of arrays
For string, specify the string length. For other than string, specify the number of array elements.

Note that it is not possible to specify a pointer declarator at the same time, or to specify a multidimensional array such as Identifier[2][3]. If specified, an error will occur during MEM generation.

Example

```
typedef unsigned int32_t UINT32;
typedef UINT32 * PUINT32;
typedef UINT32 UINT32Array[10];
```

NOTE

Type already defined by default do not need to be redefined. For data type that is predefined by default, see "3.1.6.1 Built-in definition".

3.1.6.1 Built-in definition

For the purpose of supporting the use of the Windows API, the following definitions are pre-installed in the FP driver

```
typedef int32_t int;
typedef int8_t char;
typedef int16_t short;
typedef int32_t long;
typedef int16_t wchar_t;
typedef int32_t size_t;
typedef int32_t INT;
typedef uint32_t UINT;
typedef int32_t BOOL;
typedef int8_t CHAR;
typedef uint8_t BYTE;
typedef int16_t SHORT;
typedef uint16_t WORD;
typedef int32_t LONG;
typedef uint32_t DWORD;
typedef int16_t WCHAR;
typedef int32_t SIZE_T;
typedef uint32_t COLORREF;
typedef string LPSTR;
typedef string LPCSTR;
typedef wstring LPWSTR;
typedef wstring LPCWSTR;
typedef int32_t INT32;
typedef uint32_t UINT32;
typedef int32_t INT_PTR;
typedef uint32_t UINT_PTR;
typedef int32_t LONG_PTR;
typedef uint32_t ULONG_PTR;
typedef uint32_t DWORD_PTR;
typedef void VOID;
typedef void* LPVOID;
typedef void* HANDLE;
typedef void* HWND;
```

```
typedef void* HDC;
typedef void* HFONT;
typedef void* HGDIOBJ;
typedef void* HBITMAP;
typedef void* HBRUSH;
typedef void* HPEN;
typedef void* HRGN;

struct POINT {
    LONG x;
    LONG y;
};
struct SIZE {
    LONG cx;
    LONG cy;
};
struct RECT {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
};
```

3.1.7 Roles for inserting comment

A comment can be inserted between /* and */.
It can also be inserted from // to line break.

Example

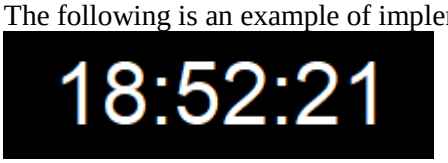
```
/*
  Comment text
*/
const NUM = 10; // Comment text
```

3.2 RUBY SCRIPT

Next, functions defined in external function definition file are used in Ruby script.
The precautions are as follows.

1. Make an instance of the defined structure with "new" as mruby structure.
2. When the argument type is an integer or floating point number, and output to the argument is specified using the "out" specifier, use an array type variable for the argument of the function to be executed.
   Example)
   External function definition file
   ```
   function void func library "lib" (out int *);
   ```
   Ruby script
   ```
   val = 0
   func(val) // no output to val
   val = [0] // create array
   func(val) // output to val[0]
   val = val[0]
   ```

The following is an example of implementing a digital clock display with Windows API.



External function definition file

```
function HDC CreateCompatibleDC library "gdi32" (HDC);
function BOOL DeleteDC library "gdi32" (HDC);
function HGDIOBJ SelectObject library "gdi32" (HDC, HGDIOBJ);
function BOOL DeleteObject library "gdi32" (HDC, HGDIOBJ);
struct LOGFONT {
    LONG lfHeight; LONG lfWidth; LONG lfEscapement; LONG lfOrientation; LONG lfWeight;
    BYTE lfItalic; BYTE lfUnderline; BYTE lfStrikeOut; BYTE lfCharSet; BYTE lfOutPrecision; BYTE lfClipPrecision;
    BYTE lfQuality; BYTE lfPitchAndFamily;
    string lfFaceName[32];
};
const DEFAULT_CHARSET = 1;
const CLEARTYPE_QUALITY = 5;
function HFONT CreateFontIndirect entry CreateFontIndirectA library "gdi32" (LOGFONT*);
function HBRUSH CreateSolidBrush library "gdi32" (COLORREF);
function COLORREF SetTextColor library "gdi32" (HDC, COLORREF);
function int SetBkMode library "gdi32" (HDC, int);
const TRANSPARENT = 1;
const OPAQUE = 2;

function int GetDeviceCaps library "gdi32" (HDC, int);
const LOGPIXELSX = 88; const LOGPIXELSX = 88; const LOGPIXELSX = 88
const LOGPIXELSY = 90; const LOGPIXELSY

function int DrawText entry DrawTextA library "user32" (HDC, LPSTR, int, RECT*, UINT);
const DT_TOP = 1;
const DT_LEFT = 0;
const DT_CENTER = 0x01;
const DT_VCENTER = 0x04;
const DT_SINGLELINE = 0x20;

function int MulDiv library "kernel32" (int, int, int);
```

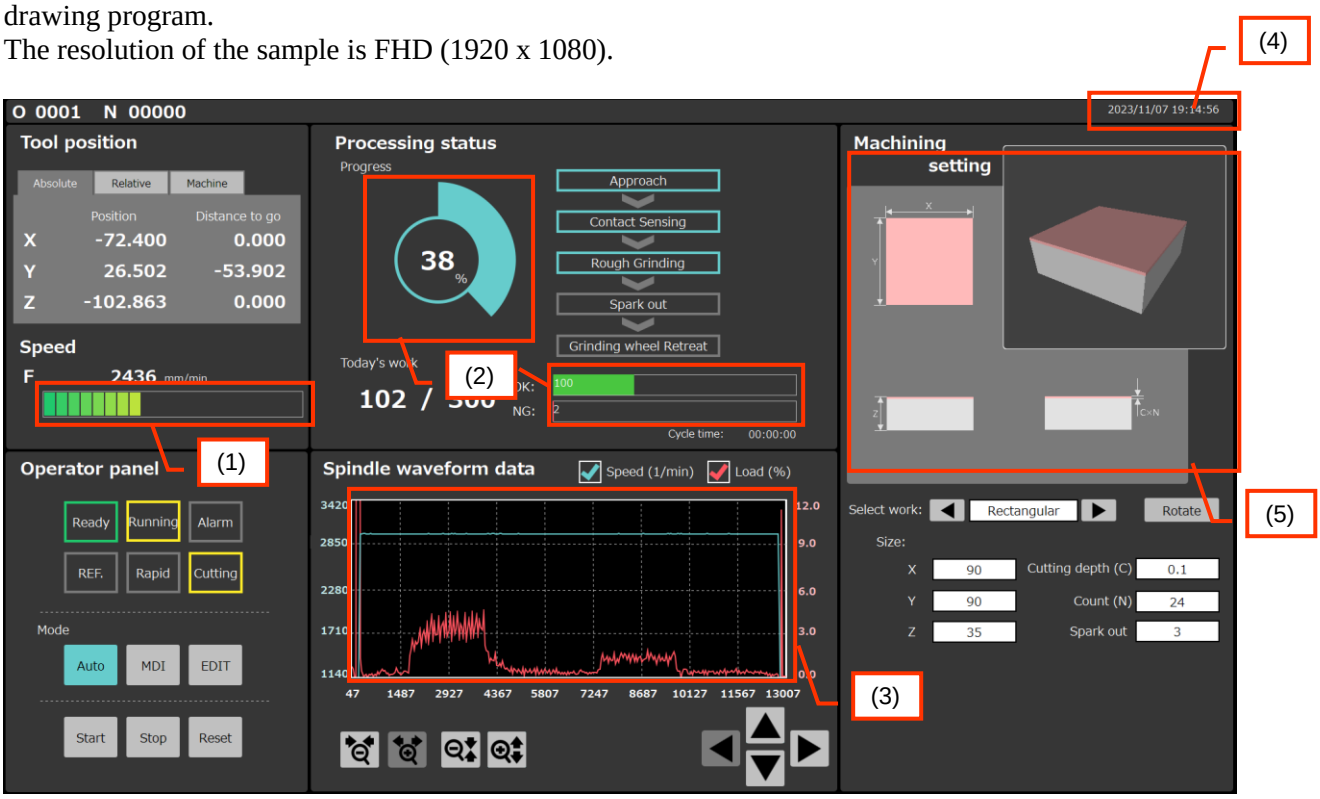
Ruby script

```
class DigitalClock < FpUserControl
    def initialize()
        super
        self.timer_interval = 1000
    end
    def paint()
        s = ctime(time()[1])[1].slice(11, 8) # s = hh:mm:ss
        rc = fprect_to_rect(rect)
        # Create DC
        hdc = CreateCompatibleDC(NULL)
        # Select HBITMAP
        oldbmp = SelectObject(hdc, bitmap)
```

```
            # Create font
            lf = LOGFONT.new
            lf.lfHeight = -MulDiv(32, GetDeviceCaps(hdc, LOGPIXELSY), 72)
            lf.lfCharSet = DEFAULT_CHARSET
            lf.lfQuality = CLEARTYPE_QUALITY
            lf.lfFaceName = "Arial"
            font = CreateFontIndirect(lf)
            # Display time
            oldfont = SelectObject(hdc, font)
            SetTextColor(hdc, RGB(255, 255, 255))
            SetBkMode(hdc, TRANSPARENT)
            DrawText(hdc, s, -1, rc, DT_CENTER | DT_VCENTER | DT_SINGLELINE)
            SelectObject(hdc, oldfont)
            SelectObject(hdc, oldbmp)
            # Release font
            DeleteObject(font)
            # Release DC
            DeleteDC(hdc)
        end
        def timer()
            restore_rect() # Redraw background
            paint()
            update_rect() # Update drawing
        end
    end
    DigitalClock.new().start()
```

4 SAMPLE PROJECT

A sample project of an operation panel screen for a surface grinder is provided as an example of creating a drawing program.
The resolution of the sample is FHD (1920 x 1080).



Screen components with drawing programs:
(1) Feed rate meter
(2) Graph of processing progress rate
(3) Graph of spindle waveform data
(4) Clock
(5) Three-view drawing and 3D view of the work

4.1 FOLDER STRUCTURE

A sample project for the drawing program is installed below.

In the folder where FANUC PICTURE is installed
- Sample
  - iHMI
    - DrawingLibrary ... A set of sample projects for drawing programs
      - DrawingLibrary ... FANNUC PICTURE project folder
      - FPLinkiHMI ... User function project folder
      - data.csv ... Data displayed in graph of spindles waveform data

NOTE

1 When you see or edit the sample, do not open it directly in software such as FANUC PICTURE or Visual Studio, but rather copy it to your local disk and then use the copied project.

2 If sample file is read-only, use it after releasing read-only attribute

4.2 FILES AND CONTROLS

(1) Feed rate meter

This is an example of a drawing program displays feed rate meter with Ruby script.

The object ID of the files and controls in the FANUC PICTURE project relevant to this example are as follows.

	File	Object ID	Description
1	top.xml	FPNDisp1	Meter drawing area
2	FPScript¥dispSpeed.rb	-	Drawing program

(2) Graph of processing progress rate

This is an example of a drawing program displays the number of progress by graph with Ruby script.

Bar graph

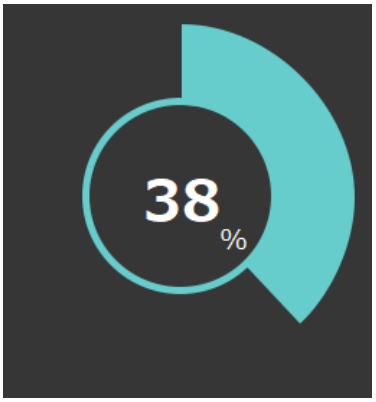
The object ID of the files and controls in the FANUC PICTURE project relevant to this example are as follows.

	File	Object ID	Description.
1	top.xml	FPNDisp2	Bar graph drawing area (for OK)
2		FPNDisp3	Bar graph drawing area (for NG)
3	FPScript¥ dispProgress1.rb	-	Drawing program

The project variables used in this example are as follows

Project variable name	Contents
DATA1	OK value
DATA2	NG value
SCHEDULE	Maximum value of bar graph

Pie chart



The object ID of the files and controls in the FANUC PICTURE project relevant to this example are as follows.

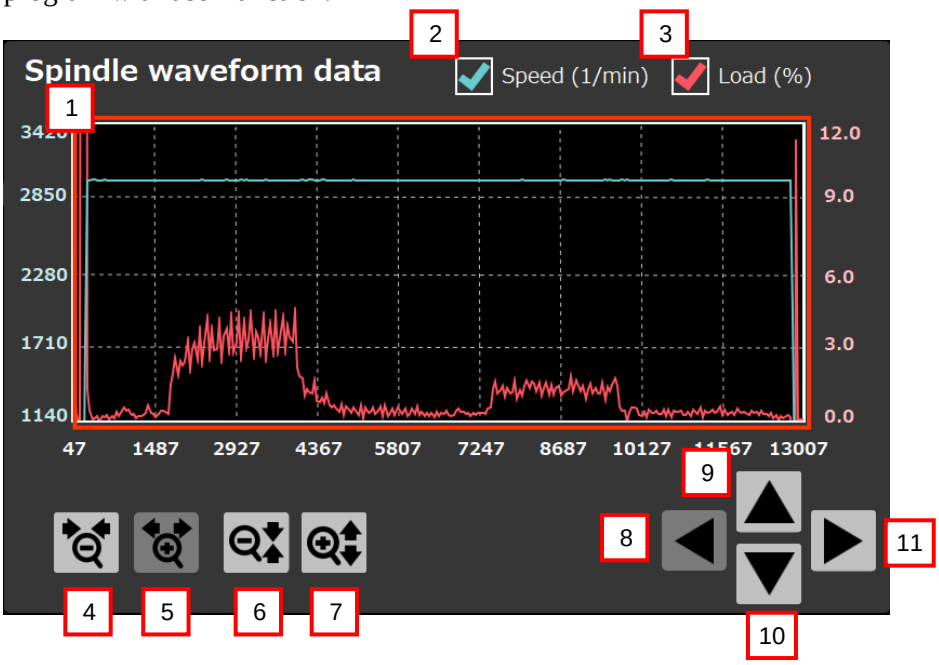
	File	Object ID	Description
1	top.xml	FPNDisp4	Pie chart drawing area
2	FPScript¥ dispProgress2.rb	-	Drawing program

The project variable used in this example is as follows.

	Project variable name	Contents
1	PROGRESS	Rate of progress

(3) Graph of Spindles Waveform Data

This is an example of a trend graph that displays spindle waveform data in CSV format and is a drawing program with user function.



In this sample, the CSV file (data.csv) collected by specifying SPSPD (rotation speed) for channel 1 and SPTCMD (load) for channel 2 in SERVO GUIDE is used. Put the CSV file in the same folder as FP driver (FPDriverApp.exe). For the format of the CSV file, refer to "5.17.1 Automatic Saving of Measurement Data" in "FANUC SERVO GUIDE OPERATOR'S MANUAL" (B-65404EN/03).

The object ID of the file and controls in the FANUC PICTURE project relevant to this example are as follows.

	File	Object ID	Description
1	top.xml	FPNDisp5	Trend graph drawing area
2		FPCheckBox1	Display/Hide "Speed" data
3		FPCheckBox2	Display/Hide "Load" data
4		FPImgButton1	Shrinking graph (horizontal)
5		FPImgButton2	Enlarging graph (horizontal)
6		FPImgButton3	Shrinking graph (vertical)
7		FPImgButton4	Enlarging graph (vertical)
8		FPImgButton5	Scrolling graph (leftward)
9		FPImgButton6	Scrolling graph (upward)
10		FPImgButton7	Scrolling graph (rightward)
11		FPImgButton8	Scrolling graph (downward)

The file and function in the user function project relevant to this example are as follows.

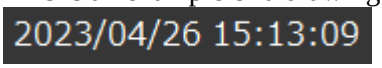
File	Function name	Description
USERFUNC.cpp	usr_DrawGraph	Drawing program

The project variables used in this example are as follows.

Project variable name	Description
SPEED	Display/Hide "Speed" graph
LOAD	Display/Hide "Load" graph
SCROLL_HOR	Scroll request for graph (horizontal)
SCROLL_VER	Scroll request for graph (vertical)
SCALE_HOR	State of graph zooming in/out (horizontal)
SCALE_VER	State of graph zooming in/out (vertical)

(4) Clock

This is an example of a drawing program using external function call (WIN32 API call) with Ruby script.

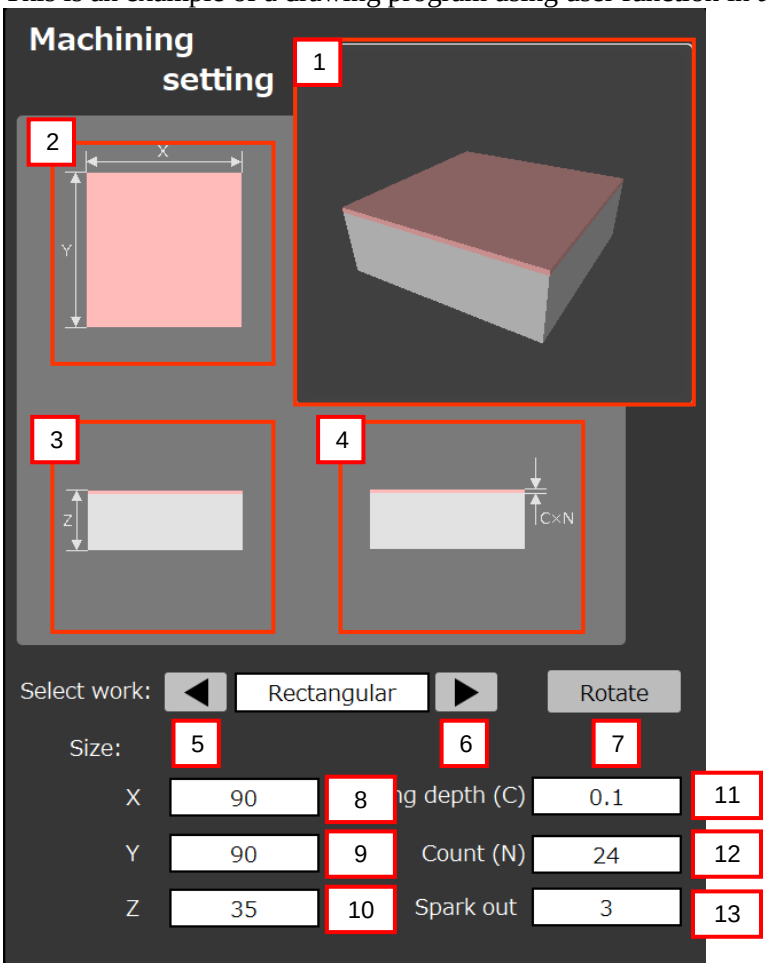


The object ID of the files and controls in the FANUC PICTURE project relevant to this example are as follows.

	File	Object ID	Description
1	top.xml	FPNDisp6	Clock drawing area
2	FPScript¥clock.rb	-	Drawing program
3	FPScript¥funcdef.txt	-	External function definition file

(5) Three-view drawing and 3D view of the work

This is an example of a drawing program using user function in a three-view drawing and 3D view of a work.



The object ID of the file and controls in the FANUC PICTURE project relevant to this example are as follows.

	File	Object ID	Description
1	top.xml	FPNDisp7	3D work display area
2		FPNDisp8	Plan view of three-view drawing
3		FPNDisp9	Front view of three-view drawing
4		FPNDisp10	Side view of three-view drawing
5		FPImgButton9	Switching work shape
6		FPImgButton10	
7		FPButton1	Start/stop work rotation
8		TFPInput1	Work size (X-axis direction)
9		TFPInput2	Work size (Y-axis direction)
10		TFPInput3	Work size (Z-axis direction)
11		TFPInput4	Cutting volume per cut
12		TFPInput5	Number of cuts
13		TFPInput6	Spark out

The file and function in the user function project relevant to this example are as follows

File	Function name	Description
USERFUNC.cpp	usr_DrawWork	Drawing program

The project variables used in this example are as follows

Project variable name	Description
WORK	Work shape
ROTATE	Work rotation start/stop status
POSX	Work size (X-axis direction)
POSY	Work size (Y-axis direction)
POSZ	Work size (Z-axis direction)
CUT_DEPTH	Cutting volume per cut
COUNT	Number of cuts

4.3 PMC SIGNAL

The following PMC signals and PMC symbols are used in the sample.

Address	#7	#6	#5	#4	#3	#2	#1	#0
F0000	OP							RWD
F0001	MA							AL
F0002		CUT					RPDO	
F0003		MEDT	MMEM		MMDI			
G0007						ST		
G0008		RRW	SP					
G0043						MD4	MD2	MD1
R1015	IL_ZMH_M	IL_ZMH_P	IL_ZMH_M	IL_ZMH_P	IL_SCR_R	IL_SCR_L	IL_SCR_D	IL_SCR_U

For details on the F and G signals, refer to the CONNECTION MANUAL (FUNCTION) for each CNC model. For the R signals, see below.

PMC Symbol	Description
IL_ZMH_M	NoAction signal for the shrinking graph button (horizontal) (FPImgButton1)
IL_ZMH_P	NoAction signal for the enlarging graph button (horizontal) (FPImgButton2)
IL_ZM_M	NoAction signal for the shrinking graph button (vertical direction) (FPImgButton3)
IL_ZM_P	NoAction signal for the enlarging graph button (vertical) (FPImgButton4)
IL_SCR_R	NoAction signal for the scrolling graph button (rightward) (FPImgButton7)
IL_SCR_L	NoAction signal for the scrolling graph button (leftward) (FPImgButton5)
IL_SCR_D	NoAction signal for the scrolling graph button (downward) (FPImgButton8)

IL_SCR_U	NoAction signal for the scrolling graph button (upward) (FPImgButton6)

5 DRAWING LIBRARY REFERENCE

This chapter contains the specifications of data and functions in the drawing library.

5.1 COLOR
Color definition and Functions

5.2 COORDINATE
Coordinate definitions and functions

5.3 FONT
Font information class

5.4 CONTROL
Drawing program implementation class

5.5 PROJECT VARIABLE
Functions to access project variables (C++ only)

5.6 BUFFER FOR EXTERNAL FUNCTION CALL
Data array class for external function call (mruby only)

5.7 EXTRA
Extra functions to support drawing program

NOTE

1 The following notation indicates that the description is for Ruby script or user function.
[mruby] : Ruby script
[C++] : User function
[mruby,C++] : Common to Ruby script and user function

2 mruby functions and classes (structures) follows the example below.

```
Integer → arg
func (arg) → String
```

Indicates that the argument "arg" of function "func" is integer type.
Indicates that the return value of function "func" is string type.

```
Integer → a,b
class C
  a
  b
end
```

Indicates that class (structure) "C" has member variables "a" and "b" of integer type.

```
Integer array[n][m] → arg
func (arg) → String
```

Indicates that the argument "arg" of the function "func" is a two-dimensional array of integer type.
"n" and "m" stand for the number of elements. If "n" or "m" is omitted, it indicates an arbitrary number of elements.

3 The argument notation below indicates whether the argument is for "input", "output", or "input and output".
Be sure to set values for "input" and "input and output" arguments before executing the function.
[in] : input
[out] : output
[in / out] : input and output

4 The description of the format of C++ function and classes (structures) follows own language specifications.

5.1 COLOR

5.1.1 Data type

FP_COLOR [C++]

This data type indicates the color value of FANUC PICTURE.

[C++]

```
typedef unsigned long FP_COLOR;
```

NOTE

1 Ruby script dose not have the above data type.

2 Get a color value of FANUC PICTURE with either function fpcolor [mruby] or fpColor [C++] or colorref_to_fpcolor [mruby] in "5.1.2 Function"
In addition, the color value of FANUC PICTURE is a 32-bit integer value whose bit 0 to 7 represents blue intensity, bit 8 to 15 bits represents green intensity, and bit 16 to 23 represents red intensity. Bits 24 to 31 are reserved for alpha color and are currently is not available.

5.1.2 Function

RGB [mruby]

Converts red, green, and blue (RGB) colors to a color value same as the color format WIN32 API COLORREF type.

[mruby]

```
Integer → r, g, b
RBG(r, g, b) → Integer
```

Argument

r [in]
Red intensity (0 - 255)

g [in]
Green intensity (0 - 255)

b [in]
Blue intensity (0 - 255)

Return value

Color value of COLORREF type

fpcolor [mruby], fpColor [C++]

Converts red, green, and blue (RGB) colors to a FANUC PICTURE color value.

[mruby]

```
Integer → r, g, b
fpcolor(r, g, b) → Integer
```

[C++]

```
FP_COLOR fpColor(int r, int g, int b);
```

Argument

r [in]

Red intensity (0 - 255)

g [in]

Green intensity (0 - 255)

b [in]

Blue intensity (0 - 255)

Return value

Color value of FANUC PICTURE

colorref_to_fpcolor [mruby]

Converts a color value of WIN32 API COLORREF type to FANUC PICTURE color type.

[mruby]

```
Integer → color
colorref_to_fpcolor (color) → Integer
```

Argument

color [in]

Value of COLORREF type

Return value

Color value of FANUC PICTURE

fpcolor_to_colorref [mruby]

Converts a color value of FANUC PICTURE to WIN32 API COLORREF type.

[mruby]

```
Integer → color
fpcolor_to_colorref (color) → Integer
```

Argument

color [in]

Color value of FANUC PICTURE

Return value

Value of COLORREF type

5.2 COORDINATE

5.2.1 Structure

FpPoint [mruby], FP_POINT [C++]

Defines coordinates.

[mruby]

```
Integer → x, y
class FpPoint
      x
      y
end
```

[C++]

```
typedef struct _FP_POINT {
      int x;
      int y;
```

```
} FP_ POINT;
```

Member

x
  X coordinate

y
  Y coordinate

FpSize [mruby], FP_SIZE [C++]

Defines size.

[mruby]

```
Integer → width, height
class FpSize
      width
      height
end
```

[C++]

```
typedef struct _FP_SIZE {
      int width;
      int height;
} FP_SIZE;
```

Member

width
  Width

height
  Height

FpRect [mruby], FP_RECT [C++]

Defines a rectangle.

[mruby]

```
Integer→ x, y, width, height
class FpRect
      x
      y
      width
      height
end
```

[C++]

```
typedef struct _FP_RECT {
      int x;
      int y;
      int width;
      int height;
} FP_RECT;
```

Member

x
  X coordinate of the top left corner of the rectangle

y
  Y coordinate of the top left corner of the rectangle

width

Width of the rectangle

height

Height of the rectangle

5.2.2 Function

point_to_fppoint [mruby]

Converts a value of WIN32 API POINT type to FpPoint type.

[mruby]

```
POINT→ point
point_to_fppoint (point) → FpPoint
```

Argument

point [in]

Value of POINT type

Return value

FpPoint type value

fppoint_to_point [mruby]

Converts a value of FpPoint type to one of WIN32 API POINT type.

[mruby]

```
FpPoint→ point
fppoint_to_point (point) → POINT
```

Argument

point [in]

FpPoint type value

Return value

Value of POINT type

size_to_fpsize [mruby]

Converts a value of WIN32 API SIZE type to FpSize type.

[mruby]

```
SIZE→ size
size_to_fpsize (size) → FpSize
```

Argument

size [in]

SIZE type value

Return value

FpSize type value

fpsize_to_size [mruby]

Converts a value of FpSize type to WIN32 API SIZE type.

[mruby]

```
FpSize→ size
fpsize_to_size (size) → SIZE
```

Argument

size [in]

FpSize type value

Return value
SIZE type value

rect_to_fprect [mruby]
Converts a value of WIN32 API RECT type to FpRect type.
[mruby]

```
RECT→ rect
rect_to_fprect (rect) → FpRect
```

Argument
rect [in]
RECT type value
Return value
Value of FpRect type
Remarks
In accordance with the rules of the WIN32 API RECT type, the bottom right coordinate of the rectangle is exclusive (the bottom right coordinate is outside the rectangle) and is calculated as follows.
FpRect.x = RECT.left
FpRect.y = RECT.top
FpRect.width = RECT.right - RECT.left
FpRect.height = RECT.bottom - RECT.top

fprect_to_rect [mruby]
Converts a value of FpRect type to WIN32 API RECT type.
[mruby]

```
FpRect → rect
fprect_to_rect (rect) → RECT
```

Argument
rect [in]
Value of FpRect type
Return value
RECT type value
Remarks
In accordance with the rules of the WIN32 API RECT type, the bottom right coordinate of the rectangle is exclusive (the bottom right coordinate is outside the rectangle) and is calculated as follows.
RECT.left = FpRect.x
RECT.top = FpRect.y
RECT.right = FpRect.x + FpRect.width
RECT.bottom = FpRect.y + FpRect.height

5.3 FONT

FpFont [mruby, C++]
This class contains font information.

5.3.1 Member function

Constructors [mruby, C++]
Creates an instance of the FpFont class.

[mruby]

```
String→ name
Integer→ point, weight
Boolean→ italic
FpFont.new(name, point, weight, italic) → FpFont
```

[C++]

```
FpFont();
FpFont(const char* name, int point, int weight = -1, bool italic = false);
```

Argument

name [in]

Font name (NULL terminated in C++)

point [in]

Specify font size in points (1 point = 1/72 × screen resolution (DPI))

weight [in]

Specify font weight in the range of 0 to 99

To match the FANUC PICTURE character type, specify 50 for standard and 75 for bold.

It becomes standard (=50) when omitted or specifying -1.

italic [in]

Italics if true

It becomes standard when omitted or false.

Return value

Instance of the FpFont class

Example

[mruby]

```
font = FpFont.new("Arial", 32)
draw_text(0, 0, "TEST", font, fpcolor(255, 255, 255))
font.dispose()
```

[C++]

```
FpFont font ("Arial", 32)
draw_text(0, 0, "TEST", font, fpcolor(255, 255, 255))
```

create [C++]

Creates font information.

[C++]

```
bool create(const char* name, int point, int weight = -1, bool italic = false);
```

Argument

Same as constructor arguments

See constructor description (above mentioned).

Return value

True if successful, and false otherwise.

dispose [mruby, C++]

Discards font information.

[mruby]

```
dispose()
```

[C++]

```
void dispose();
```

5.4 CONTROL

FpUserControl [mruby, C++]

This class is to implement control drawing-related events.

5.4.1 Line style

[mruby]

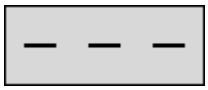
```
FP_LINE_SOLID = 1
FP_LINE_DASH = 2
FP_LINE_DOT = 3
FP_LINE_DASHDOT = 4
FP_LINE_DASHDOTDOT = 5
```

[C++]

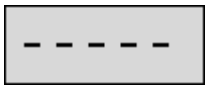
```
enum LineStyle {
    FP_LINE_SOLID = 1, FP_LINE_DASH, FP_LINE_DOT, FP_LINE_DASHDOT,
    FP_LINE_DASHDOTDOT
};
```

FP_LINE_SOLID
Solid line

FP_LINE_DASH
Dashed line



FP_LINE_DOT
Dotted line

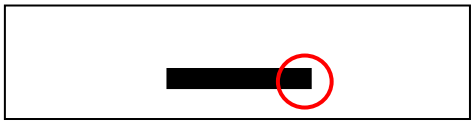


FP_LINE_DASHDOT
Dashed-dotted line

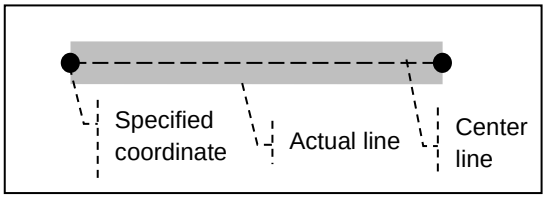
FP_LINE_DASHDOTDOT
Dashed-two dotted line

NOTE

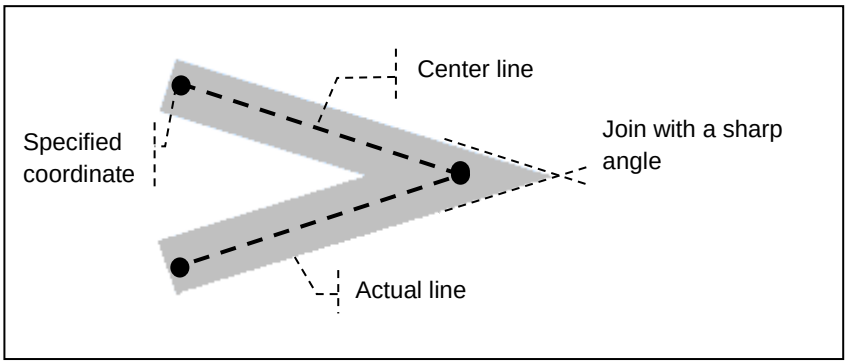
1 The end of the line has a flat shape.



2 The line thickens around the line connecting the specified coordinates.



3 Consecutive lines joins at acute angles.



5.4.2 Text style

[mruby]

```
FP_TEXT_LEFT = 0x00
FP_TEXT_RIGHT = 0x01
FP_TEXT_H_CENTER = 0x02
FP_TEXT_TOP = 0x00
FP_TEXT_BOTTOM = 0x04
FP_TEXT_V_CENTER= 0x20
FP_TEXT_WORDWRAP = 0x100
FP_TEXT_CENTER = FP_TEXT_H_CENTER | FP_TEXT_V_CENTER
```

[C++]

```
enum TextStyle {
    FP_TEXT_LEFT = 0,
    FP_TEXT_RIGHT = 1 << 0,
    FP_TEXT_H_CENTER = 1 << 1,
    FP_TEXT_TOP = 0,
    FP_TEXT_BOTTOM = 1 << 4,
    FP_TEXT_V_CENTER = 1 << 5,
    FP_TEXT_WORDWRAP = 1 << 8,
    FP_TEXT_CENTER = FP_TEXT_H_CENTER | FP_TEXT_V_CENTER
};
```

FP_TEXT_LEFT
  Left-aligned display
FP_TEXT_RIGHT
  Right-aligned display
FP_TEXT_H_CENTER
  Horizontal center display
FP_TEXT_TOP
  Top-aligned display
FP_TEXT_BOTTOM
  Bottom-up display
FP_TEXT_V_CENTER
  Vertical center display
FP_TEXT_WORDWRAP
  Wrap automatically to the next line according to the display width
FP_TEXT_CENTER
  Horizontal + vertical center display

NOTE

The new line break position by the text style of FP_TEXT_WORDWRAP differs between double-byte characters and half-size characters as shown below.

- Double-byte characters: Character that exceed the display width displays on the next line.
- Half-size characters: Word that exceed the display width displays on the next line.

5.4.3 Member function

Constructor

mruby	C++	Description
-	-	Constructor

Events

mruby	C++	Description
paint	paint	Paint event
timer	timer	Timer event
dispose	dispose	Terminate event

Property

mruby	C++	Description
rect	getRect	Gets coordinates of control area (top left corner origin of control)
global_rect	getGlobalRect	Gets coordinates of control area (top left corner origin of screen)
source_rect	getSourceRect	Gets coordinates of control area before scaling (top left corner origin of screen)
name	getName	Gets an object ID of the control
timer_interval	getTimerInterval	Gets an update period of timer
timer_interval=	setTimerInterval	Sets an update period of timer

Drawing

mruby	C++	Description
update_rect	updateRect	Updates a display area
restore_rect	restoreRect	Restores a background of display area
clip_rect	clipRect	Clips a display area
transfer_bits	transferBits	Transfers a bit-image
draw_line	drawLine	Draws a straight line
draw_lines	drawLines	Draws consecutive straight lines
draw_rect	drawRect	Draws a rectangle
fill_rect	fillRect	Fills a rectangle
draw_rects	drawRects	Draws consecutive rectangles
fill_rects	fillRect	Fills consecutive rectangles
draw_ellipse	drawEllipse	Draws an ellipse
fill_ellipse	fillEllipse	Fills an ellipse
draw_arc	drawArc	Draws an arc
draw_pie	drawPie	Draws a pie
fill_pie	fillPie	Fills a pie
draw_polygon	drawPolygon	Draws a polygon
fill_polygon	fillPolygon	Fills a polygon
draw_image	drawImage	Draws an image
draw_text	drawText	Draws a string
text_metrics	textMetrics	Gets a string size

Environment

mruby	C++	Description
screen_size	getScreenSize	Gets a size of the screen
bitmap	getBitmap	Gets WIN32 API HBITMAP of the screen
bits	getBits	Gets a bit image of the screen

Other

mruby	C++	Description
print_error	printError	Displays an error message

Constructors [mruby, C++]

Creates an instance of the FpUserControl class.

[mruby]

```
FpUserControl.new() → FpUserControl
```

[C++]

```
FpUserControl(void* context, char* parameter);
```

Argument

context [in]

Control context

parameter [in]

User function argument

Return value

An instance of the FpUserControl class

Remarks

This function cannot be called directly. It must be called from the constructor of the inherited class.
The arguments to the constructor should be passed as they are specified for the user function.

paint [mruby, C++]

Handler for the paint event.

[mruby]

```
paint()
```

[C++]

```
virtual void paint() = 0;
```

Remarks

This event handler must be implemented in the inherited class. If not implemented, a compile error will occur in the case of C++.

timer [mruby, C++]

Handler for the timer event.

[mruby]

```
timer()
```

[C++]

```
virtual void timer();
```

Remarks

Implement this event handler in the inherited class as necessary. This event handler will be executed at the set update interval of the timer.

dispose [mruby, C++]

Handler for the terminate event.

[mruby]

```
dispose()
```

[C++]

```
virtual void dispose();
```

Remarks

Implement this event handler in the inherited class as necessary. This event handler is executed when the instance is terminated.

rect [mruby], getRect [C++]

Gets coordinates of the control area.

[mruby]

```
rect() → FpRect
```

[C++]

```
FP_RECT getRect();
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".

Return value

Coordinates of the control area

Remarks

The origin of the coordinates is the top left corner of the control. Therefore, the x and y coordinates to be get are always 0.

global_rect [mruby], getGlobalRect [C++]

Gets coordinates of the control area.

[mruby]

```
global_rect() → FpRect
```

[C++]

```
FP_RECT getGlobalRect();
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".

Return value

Coordinates of the control area

Remarks

The origin of the coordinates is the top left corner of the screen.

source_rect [mruby], getSourceRect [C++]

Gets coordinates of the control area before scaling.

[mruby]

```
source_rect() → FpRect
```

[C++]

```
FP_RECT getSourceRect();
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".

Return value

Coordinates of the control area before scaling

Remarks

The origin of the coordinates is the top left corner of the screen.

Example

The screen scaling factor is calculated based on the coordinates of the control area before scaling and the coordinates of the actual control area.

[mruby]

```
rs = source_rect
rg = global_rect
scale_x = rg.x.to_f / rs.x # horizontal scale factor
scale_y = rg.y.to_f / rs.y # vertical scale factor
```

[C++]

```
FP_RECT rs = getSourceRect();
FP_RECT rg = getGlobalRect();
float scale_x = static_cast<float>(rg.x) / rs.x; // horizontal scale factor
float scale_y = static_cast<float>(rg.y) / rs.y; // vertical scale factor
```

name [mruby], getName [C++]

Gets an object ID of the control.

[mruby]

```
name() → String
```

[C++]

```
const char* getName();
```

Return value

Object ID of the control (NULL terminated in C++)

timer_interval [mruby], getTimerInterval [C++]

Gets a timer period to execute the timer event.

[mruby]

```
timer_interval → Integer
```

[C++]

```
unsigned long getTimerInterval();
```

Return value

Timer period

timer_interval= [mruby], setTimerInterval [C++]

Sets a timer period to execute the timer event.

[mruby]

```
Integer → time
timer_interval(time)
```

[C++]

```
void setTimerInterval(unsigned long time);
```

Argument

time [in]

Timer period

update_rect [mruby], updateRect [C++]

Updates a display area.

[mruby]

```
FpRect → rect
update_rect(rect)
```

[C++]

```
void updateRect(const FP_RECT* rect = NULL);
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".

Argument

rect [in]

Display area to be updated

If omitted or NULL (C++) is specified, it becomes the entire control area.

Remarks

The origin of coordinates is the top left corner of the control.

restore_rect [mruby], restoreRect [C++]

Restores a background of display area.

[mruby]

```
FpRect → rect
restore_rect(rect)
```

[C++]

```
void restoreRect(const FP_RECT* rect = NULL);
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".

Argument

rect [in]

Display area to restore background

If omitted or NULL (C++) is specified, it becomes the entire control area.

Remarks

The origin of coordinates is the top left corner of the control.

clip_rect [mruby], clipRect [C++]

Clips a display area.

[mruby]

```
FpRect → rect
clip_rect(rect)
```

[C++]

```
void clipRect(const FP_RECT* rect = NULL);
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".

Argument

rect [in]

Clipping area

If omitted or NULL (C++) is specified, it becomes the entire control area.

Remarks

The origin of coordinates is the top left corner of the control.

transfer_bits [mruby], transferBits [C++]

Transfers a bit image onto the screen.

[mruby]

```
Integer → x, y, w, h
C pointer → bits
Boolean → bottomup
transfer_bits(x, y, bits, w, h, bottomup) → Boolean
```

[C++]

```
bool transferBits(int x, int y, const unsigned char* bits, int w, int h, bool bottomup=false);
```

Argument

x [in]

X coordinate of the top left corner of the destination rectangle

y [in]

Y coordinate of the top left corner of the destination rectangle

bits [in]

Pointer to the source bit image

w [in]

Width of the source bit image

h [in]
Height of the source bit image
bottomup [in]
Specify true if the source bit image is bottom-up (starting from the bottom line), false if it is top-down (starting from the top line)
It becomes false if omitted.

Return value

True if successful, and false if the destination is outside the control area or the source bit image area is less than or equal to 0.

Remarks

The color format of the bit images is 32-bit ARGB. However, A (alpha color) is not used.
Clipping by clip_rect [mruby] or clipRect [C++] is not affected. However, if the bit-image after the transfer exceeds the area of the control, the transfer area will be clipped to fit within the area of the control.
The origin of coordinates is the top left corner of the control.

draw_line [mruby], drawLine [C++]

Draws a straight line.
[mruby]

```
Integer →sx, sy, ex, ey, style, width, color
draw_line(sx, sy, ex, ey, style, width, color)
```

[C++]

```
void drawLine(long sx, long sy, long ex, long ey, LineStyle style, int width, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

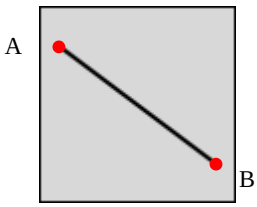
sx [in]
X coordinate of start point
sy [in]
Y coordinate of start point
ex [in]
X coordinate of end point
ey [in]
Y coordinate of end point
style [in]
Line style (see "5.4.1 Line style")
width [in]
Line thickness (in pixels)
color [in]
Line color

Remarks

Draws a line including the endpoint.
The origin of the coordinates is the top left corner of the control.

Example

Draws the following line in a control whose width is 100 and height is 100
- A: Start point (10, 20)
- B: End point (90, 80)
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
draw_line(10, 20, 90, 80, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
drawLine(10, 20, 90, 80, FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

draw_lines [mruby], drawLines [C++]

Draws a consecutive straight line.

[mruby]

```
FpPoint array → points
Integer → style, width, color
draw_lines(points, style, width, color)
```

[C++]

```
bool drawLines(const FP_POINT* points, int num, LineStyle style, int width, FP_COLOR color);
```

For FP_POINT, see "FP_POINT [C++]" in "5.2 COORDINATE".

For FP_COLOR, see "5.1 COLOR".

Argument

points [in]

Array of consecutive straight line vertices

[mruby] The 2nd dimension of the array is stored in the order of x and y coordinates.

num [in]

Number of vertices specified by points

style [in]

Line style (see "5.4.1 Line style")

width [in]

Line thickness (in pixels)

color [in]

Line color

Return value

[C++ only] true if successful, and false if num is negative.

Remarks

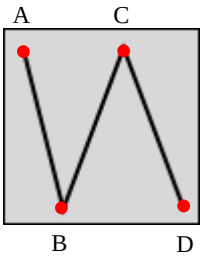
Draws a line including the endpoint.

The origin of the coordinates is the top left corner of the control.

Example

Draws the following consecutive straight lines within a control whose width is 100 and height is 100

- A: 1st point (10, 10)
- B: 2nd point (30, 90)
- C: 3rd point (60, 10)
- D: 4th point (90, 90)
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
points = [ [10,10], [30,90], [60,10], [90,90] ]
draw_lines(points, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
FP_POINT points[] = { {10,10}, {30,90}, {60,10}, {90,90} };
drawLines(points, _countof(points), FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

draw_rect [mruby], drawRect [C++]

Draws a rectangle.

[mruby]

```
Integer →sx, sy, ex, ey, style, width, color
draw_rect(sx, sy, ex, ey, style, width, color)
```

[C++]

```
void drawRect(long sx, long sy, long ex, long ey, LineStyle style, int width, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

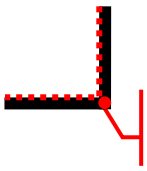
Argument

sx [in]
X coordinate of the top left corner
sy [in]
Y coordinate of the top left corner
ex [in]
X coordinate of the bottom right corner
ey [in]
Y coordinate of the bottom right corner
style [in]
Line style (see "5.4.1 Line style")
width [in]
Line thickness (in pixels)
color [in]
Line color

Remarks

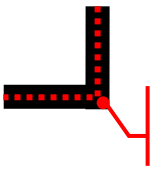
A rectangular line is drawn with its thickness centered on a rectangle whose x and y coordinates of the bottom right corner is 1 pixel inside the specified ones.
If its line thickness is 1, a rectangle containing the x and y coordinates of the bottom right corner is drawn.

Line thickness = 1:



Bottom right x, y coordinates

Line thickness = 2:

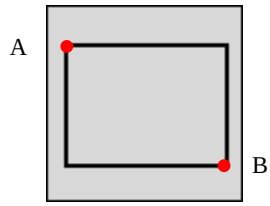


Bottom right x, y coordinates

The origin of the coordinates is the top left corner of the control.

Example

Draws the following rectangle in a control whose width is 100 and height is 100
- A: Top left corner (10, 20)
- B: Bottom right corner (90, 80)
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
draw_rect(10, 20, 90, 80, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
drawRect(10, 20, 90, 80, FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

fill_rect [mruby], fillRect [C++]

Fills a rectangle.

[mruby]

```
Integer → sx, sy, ex, ey, color
fill_rect(sx, sy, ex, ey, color)
```

[C++]

```
void fillRect(long sx, long sy, long ex, long ey, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

sx [in]
X coordinate of the top left corner
sy [in]
Y coordinate of the top left corner
ex [in]
X coordinate of the bottom right corner
ey [in]
Y coordinate of the bottom right corner
color
Fill color

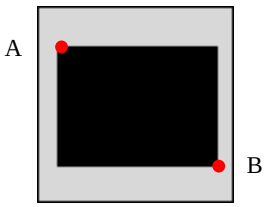
Remarks

A rectangle is filled without including the coordinates of the bottom right corner so that these are 1 pixel outside the drawing area.

The origin of the coordinates is the top left corner of the control.

Example

Fills the following rectangle in a control whose width is 100 and height is 100
- A: Top left corner (10, 20)
- B: Bottom right corner (90, 80)
- Color: Black



[mruby]

```
fill_rect(10, 20, 90, 80, fpcolor(0, 0, 0))
```

[C++]

```
fillRect(10, 20, 90, 80, fpColor(0, 0, 0));
```

draw_rects [mruby], drawRects [C++]

Draws consecutive rectangles.

[mruby]

```
Integer array[][4] →rects
Integer →style, width, color
draw_rects(rects, style, width , color)
```

[C++]

```
bool drawRects(const FP_RECT* rects, int num, LineStyle style, int width, FP_COLOR color);
```

For FP_RECT, see "FP_RECT [C++]" in "5.2 COORDINATE".
For FP_COLOR, see "5.1 COLOR".

Argument

rects [in]
Array of rectangles
[mruby] The 2nd dimension of the array is stored in the order of x and y coordinates of the top left corner, width and height.

num [in]
Number of rectangles specified by rects

style [in]
Line style (see "5.4.1 Line style")

width [in]
Line thickness (in pixels)

color [in]
Line color

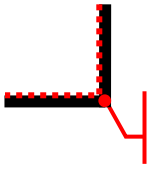
Return value

[C++ only] true if successful, and false if num is negative.

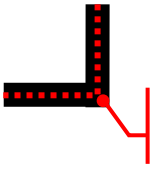
Remarks

A rectangular line is drawn with its thickness centered on a rectangle whose x and y coordinates of the bottom right corner is 1 pixel inside the specified ones.
If its line thickness is 1, a rectangle containing the x and y coordinates of the bottom right corner is drawn.

Line thickness = 1:



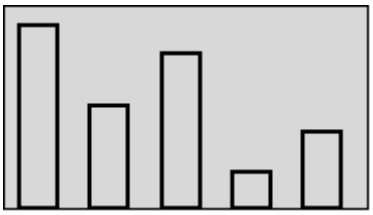
Line thickness = 2:



The origin of the coordinates is the top left corner of the control.

Example

Draws the following consecutive rectangles as bar charts in a control whose width is 200 and height is 120
- Number of data: 5
- Graph width: 20
- Graph interval: 15
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
datas = [90, 50, 80, 10, 30]
rects = []
for i in 0..datas.length-1 do
  data_h = datas[i] / 120 * 100 # Length of graph
  rects.push([i * 35, 120-data_h, 20, data_h])
end
draw_rects(rects, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
int datas[] = {90, 50, 80, 10, 30};
FP_RECT rects[5];
for (int i = 0; i < 5; i++;){
  int data_h = datas[i] / 120 * 100; // Length of graph
  rects[i].x = 120 - data_h;
  rects[i].y = i * 35;
  rects.width = 20;
  rects.height = data_h;
}
drawRects (rects, _countof(rects), FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

fill_rects [mruby], fillRects [C++]

Fills consecutive rectangles.

[mruby]

```
Integer array[][4] →rects
Integer →color
fill_rects(rects, color)
```

[C++]

```
bool fillRects(const FP_RECT* rects, int num, FP_COLOR color);
```

For FP_RECT, see "FP_RECT [C++]" in "5.2 COORDINATE".
For FP_COLOR, see "5.1 COLOR".

Argument

rects [in]
Array of rectangles

[mruby] The 2nd dimension of the array is stored in the order of x and y coordinate of the top left corner, width and height.

num [in]

Number of rectangles specified by rects

color [in]

Fill color

Return value

[C++ only] true if successful, and false if num is negative.

Remarks

A rectangle is filled without including the coordinates of the bottom right corner so that these are 1 pixel outside the drawing area.

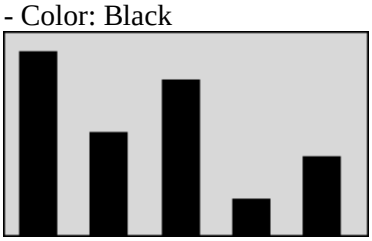
Specify coordinates 1 pixel outside the drawing area

The origin of the coordinates is the top left corner of the control.

Example

Fills the following consecutive rectangles as bar charts in a control whose width is 200 and height is 120

- Number of data: 5
- Graph width: 20
- Graph interval: 15
- Color: Black



[mruby]

```
datas = [90, 50, 80, 10, 30]
for i in 0..datas.length-1 do
  data_h = datas[i] / 120 * 100 # Length of graph
  rects.push([i * 35, 120-data_h, 20, data_h])
end
filll_rects(rects, fpcolor(0, 0, 0))
```

[C++]

```
int datas[] = {90, 50, 80, 10, 30};
FP_RECT rects[5];
for (int i = 0; i < 5; i++;){
  int data_h = datas[i] / 120 * 100; // Length of graph
  rects[i].x = 120 - data_h;
  rects[i].y = i * 35;
  rects.width = 20;
  rects.height = data_h;
}
fillRects (rects, _countof(rects), fpColor(0, 0, 0));
```

draw_ellipse [mruby], drawEllipse [C++]

Draws an ellipse.

[mruby]

```
Integer →cx, cy, rx, ry, style, width, color
draw_ellipse(cx, cy, rx, ry, style, width, color)
```

[C++]

```
void drawEllipse(long cx, long cy, long rx, long ry, LineStyle style, int width, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

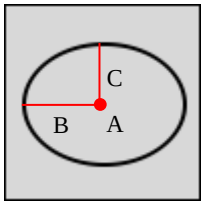
cx [in]
X coordinate of the center point
cy [in]
Y coordinate of the center point
rx [in]
Radius of ellipse on x axis
ry [in]
Radius of ellipse on y axis
style [in]
Line style (see "5.4.1 Line style")
width [in]
Line thickness (in pixels)
color [in]
Line color

Remarks

The origin of the coordinates is the top left corner of the control.

Example

Draws the following ellipses in a control whose width is 100 and height is 100
- A: Center point (50, 50)
- B: Radius of ellipse on x axis 40
- C: Radius of ellipse on y axis 30
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
draw_ellipse(50, 50, 40, 30, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
drawEllipse(50, 50, 40, 30, FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

fill_ellipse [mruby], fillEllipse [C++]

Draws a filled ellipse.

[mruby]

```
Integer →cx, cy, rx, ry, color
fill_ellipse(cx, cy, rx, ry, color)
```

[C++]

```
void fillEllipse(long cx, long cy, long rx, long ry, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

cx [in]
X coordinate of the center point

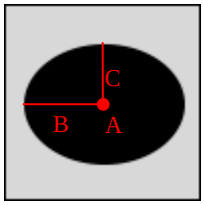
cy [in]
Y coordinate of the center point
rx [in]
Radius of ellipse on x axis
ry [in]
Radius of ellipse on y axis
color [in]
Fill color

Remarks

The origin of the coordinates is the top left corner of the control.

Example

Fills the following ellipses in a control whose width is 100 and height is 100
- A: Center point (50, 50)
- B: Radius of ellipse on x axis 40
- C: Radius of ellipse on y axis 30
- Color: Black



[mruby]

```
fill_ellipse(50, 50, 40, 30, fpcolor(0, 0, 0))
```

[C++]

```
fillEllipse(50, 50, 40, 30, fpColor(0, 0, 0));
```

draw_arc [mruby], drawArc [C++]

Draws an arc.
[mruby]

```
Integer →cx, cy, rx, ry, style, width, color
Float →sa, ea
draw_arc(cx, cy, rx, ry, sa, ea, style, width, color)
```

[C++]

```
void drawArc(long cx, long cy, long rx, long ry, double sa, double ea, LineStyle style, int width, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

cx [in]
X coordinate of the center point
cy [in]
Y coordinate of the center point
rx [in]
Radius of arc on x axis
ry [in]
Radius of arc on y axis
sa [in]
Start angle of arc
ea [in]
End angle of arc

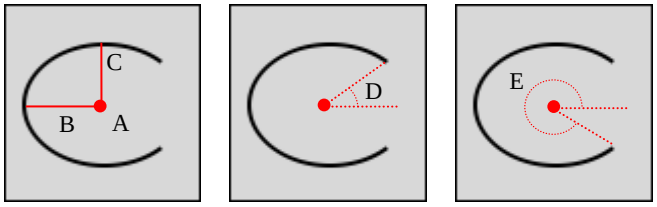
style [in]
Line style (see "5.4.1 Line style")
width [in]
Line thickness (in pixels)
color [in]
Line color

Remarks

Draws an arc with radius rx, ry centered on the center point coordinates cx, cy.
The arc length can be changed by the arc start and end angles sa and ea.
The angle is 0 degrees at 3 o'clock, a positive value means counterclockwise, and a negative value means clockwise direction.
The angle accuracy is 1/16 degree.
The origin of the coordinates is the top left corner of the control.

Example

Draws the following arc in a control whose width is 100 and height is 100
- A: Center point (50, 50)
- B: Radius of arc on X-axis 40
- C: Radius of arc on Y-axis 30
- D: Start angle of arc: 45.0 degrees
- E: End angle of arc: 315.0 degrees
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
draw_arc(50, 50, 40, 30, 45.0, 315.0, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
drawArc(50, 50, 40, 30, 45.0, 315.0, FP_LINE_SOLID, 2, fpcolor(0, 0, 0));
```

draw_pie [mruby], drawPie [C++]

Draws a pie.
[mruby]

```
Integer →cx, cy, rx, ry, style, width, color
Float →sa, ea
draw_pie(cx, cy, rx, ry, sa, ea, style, width, color)
```

[C++]

```
void drawPie(long cx, long cy, long rx, long ry, double sa, double ea, int style, int width, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

cx [in]
X coordinate of the center point
cy [in]
Y coordinate of the center point
rx [in]
Radius of pie on x axis

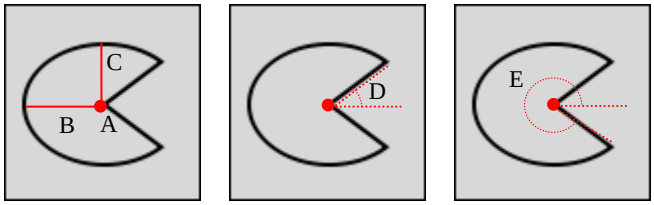
ry [in]
Radius of pie on y axis
sa [in]
Start angle of pie
ea [in]
End angle of pie
style
Line style (see "5.4.1 Line style")
width
Line thickness (in pixels)
color
Line color

Remarks

Draws a pie with radius rx, ry centered on the center point coordinates cx, cy.
The pie length can be changed by the arc start and end angles sa and ea.
The angle is 0 degrees at 3 o'clock, a positive value means counterclockwise, and a negative value means clockwise direction.
The angle accuracy is 1/16 degree.
The origin of the coordinates is the top left corner of the control.

Example

Draws the following pie in a control whose width is 100 and height is 100
- A: Center point (50, 50)
- B: Radius of pie on X-axis 40
- C: Radius of pie on Y-axis 30
- D: Start angle of pie: 45.0 degrees
- E: End angle of pie: 315.0 degrees
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
draw_pie(50, 50, 40, 30, 45.0, 315.0, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
drawPie(50, 50, 40, 30, 45.0, 315.0, FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

fill_pie [mruby], fillPie [C++]

Fills a pie.
[mruby]

```
Integer →cx, cy, rx, ry, color
Float →sa, ea
fill_pie(cx, cy, rx, ry, sa, ea, color)
```

[C++]

```
void fillPie(long cx, long cy, long rx, long ry, double sa, double ea, FP_COLOR color);
```

For FP_COLOR, see "5.1 COLOR".

Argument

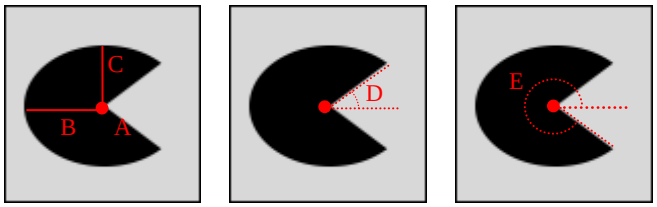
cx [in]
X coordinate of the center point
cy [in]
Y coordinate of the center point
rx [in]
Radius of pie on x axis
ry [in]
Radius of pie on y axis
sa [in]
Start angle of pie
ea [in]
End angle of pie
color [in]
Fill color

Remarks

Fills a pie with radius rx, ry centered on the center point coordinates cx, cy.
The pie length can be changed by the arc start and end angles sa and ea.
The angle is 0 degrees at 3 o'clock, a positive value means counterclockwise, and a negative value means clockwise direction.
The angle accuracy is 1/16 degree.
The origin of the coordinates is the top left corner of the control.

Example

Fills the following pie in a control whose width is 100 and height is 100
- A: Center point (50, 50)
- B: Radius of pie on X-axis 40
- C: Radius of pie on Y-axis 30
- D: Start angle of pie: 45.0 degrees
- E: End angle of pie: 315.0 degrees
- Color: Black



[mruby]

```
fill_pie(50, 50, 40, 30, 45.0, 315.0, fpcolor(0, 0, 0))
```

[C++]

```
fillPie(50, 50, 40, 30, 45.0, 315.0, fpColor(0, 0, 0));
```

draw_polygon [mruby], drawPolygon [C++]

Draws a polygon.
[mruby]

```
Integer array[][2] →points
Integer →style, width, color
draw_polygon(points, style, width, color)
```

[C++]

```
bool drawPolygon(const FP_POINT* points, int num, LineStyle style, int width, FP_COLOR color);
```

For FP_POINT, see "FP_POINT [C++]" in "5.2 COORDINATE".

For FP_COLOR, see "5.1 COLOR".

Argument

points [in]
Array of consecutive straight line vertices
[mruby] The 2nd dimension of the array is stored in the order of x and y coordinates.

num [in]
Number of vertices specified by points

style [in]
Line style (see "5.4.1 Line style")

width [in]
Line thickness (specified in pixels)

color [in]
Line color

Return value

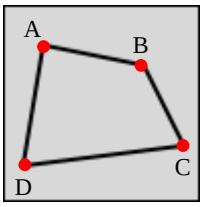
[C++ only] true if successful, and false if num is negative.

Remarks

The origin of the coordinates is the top left corner of the control.

Example

Draws the following polygon in a control whose width is 100 and height is 100
- A: 1st point (20, 20)
- B: 2nd point (70, 30)
- C: 3rd point (90, 70)
- D: 4th point (10, 80)
- Style: Solid line
- Thickness: 2 pixels
- Color: Black



[mruby]

```
points =[ [20,20], [70,30], [90,70], [10,80]]
draw_polyline(points, FP_LINE_SOLID, 2, fpcolor(0, 0, 0))
```

[C++]

```
FP_POINT points[] = { {20,20}, {70,30}, {90,70}, {10,80} };
drawPolyline(points, _countof(points), FP_LINE_SOLID, 2, fpColor(0, 0, 0));
```

fill_polygon [mruby], fillPolygon [C++]

Fills a polygon.

[mruby]

```
Integer array[][2] →points
Integer →color
Boolean →winding
fill_polygon(points, color, winding)
```

[C++]

```
bool fillPolygon(const FP_POINT* points, int num, FP_COLOR color, bool winding = false);
```

For FP_POINT, see "FP_POINT [C++]" in "5.2 COORDINATE".
For FP_COLOR, see "5.1 COLOR".

Argument

points [in]

Array of consecutive straight line vertices

[mruby] The 2nd dimension of the array is stored in the order of x and y coordinates.

num [in]

Number of vertices specified by points

color [in]

Fill color

winding [in]

If true, this function fills by WINDING mode. (It fills any region that has a nonzero winding value. This value is defined as the number of times a pen used to draw the polygon would go around the region.)
In case of the following star shape that has a pentagon in the center, all regions are filled.

If false, this function fills by ALTERNATE mode. (It fills the region between odd-numbered and even-numbered polygon sides on each scan line on x direction for each y coordinate of the polygon.)
In case of the following star shape that has a pentagon in the center, all regions are filled.
In the case of a star with a pentagon in the center, the pentagon is not filled and only the vertices of the star are filled.

The default is false if omitted.

Return value

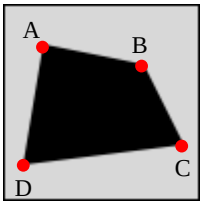
[C++ only] true if successful, and false if num is negative.

Remarks

The origin of the coordinate is the top left corner of the control.

Example

Fills the following polygon in a control whose width is 100 and height is 100
- A: 1st point (20, 20)
- B: 2nd point (70, 30)
- C: 3rd point (90, 70)
- D: 4th point (10, 80)
- Color: Black



[mruby]

```
points = [ [20,20], [70,30], [90,70], [10,80] ]
```

```
fill_polygon(points, fpcolor(0, 0, 0))
```

[C++]

```
FP_POINT points[] = { {20,20}, {70,30}, {90,70}, {10,80} };
fillPolygon(points, _countof(points), fpColor(0, 0, 0));
```

draw_image [mruby], drawImage [C++]

Draws an image of an image file.

[mruby]

```
Integer →sx, sy, ex, ey
String →image
Boolean →fit
draw_image(sx, sy, ex, ey, fname, fit) →Boolean
```

[C++]

```
bool drawImage(long sx, long sy, long ex, long ey, const char* fname, bool fit = false);
```

Argument

sx [in]
X coordinate of the top left corner

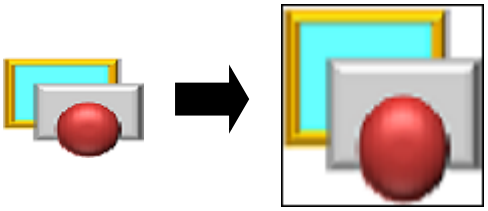
sy [in]
Y coordinate of the top left corner

ex [in]
X coordinate of the bottom right corner

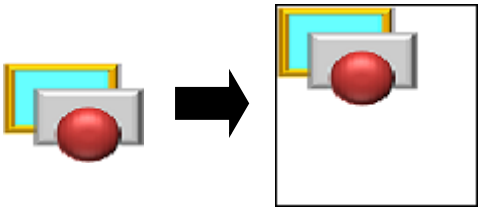
sy [in]
Y coordinate of the bottom right corner

fname [in]
Image file name (NULL terminated in C++)

fit [in]
If true, the image is drawn by being enlarged or shrunk to fit in the drawing rectangle area
For example, when the image is smaller than the drawing area



If false, the image is drawn without scaling
For Example, when the image is smaller than the drawing area



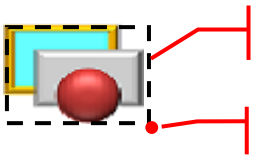
The default if false if omitted.

Return value

True if successful, and false if an image file is not found.

Remarks

An image is drawn without including the coordinates of the bottom right corner so that these are 1 pixel outside the drawing rectangle area.



If the fit argument is false and the image is smaller than the drawing area, the image is drawn based on the top left corner of the drawing area without changing the size of it. Also, since drawing is not performed outside the drawing area, if the image is larger than the drawing area, the image can only be drawn halfway into the image. The origin of the coordinate is the top left corner of the control.

Example

Draws the following image in a control whose width is width of 100 and height is 100
- Top left corner: (10, 20)
- Bottom right corner: (90, 80)
- Image file name: C:¥¥ImageFile.PNG
- Image size: Enlarge or shrink the image to fit in the drawing area

[mruby]

```
image = "C:¥¥ImageFile.PNG"
draw_image(10, 20, 90, 80, image, true)
```

[C++]

```
const char* image = "C:¥¥ImageFile.PNG";
drawImage(10, 20, 90, 80, image, true);
```

draw_text [mruby], drawText [C++]

Draws a string.

[mruby]

```
Integer →x, y, sx, sy, ex, ey, flags, color
String →s
FpFont →font
draw_text(x, y, s, font, color)
draw_text(sx, sy, ex, ey, flags, s, font, color)
```

[C++]

```
bool drawText(long x, long y, const char* s, const FpFont& font, FP_COLOR color);
bool drawText(long sx, long sy, long ex, long ey, int flags, const char* s, const FpFont& font, FP_COLOR color);
```

For FpFont, see "5.3 FONT".
For FP_COLOR, see "5.1 COLOR".

Argument

x [in]
Start x coordinate of baseline
y [in]
Start y coordinate of baseline
sx [in]
X coordinate of the top left corner of the drawing area
sy [in]

Y coordinate of the top left corner of the drawing area

ex [in]

X coordinate of the bottom right corner of the drawing area

ey [in]

Y coordinate of the bottom right corner of the drawing area

s [in]

String (NULL terminated in C++)

flags [in]

Text styles (see "5.4.2 Text style")

font [in]

Font

color [in]

Text color

Return value

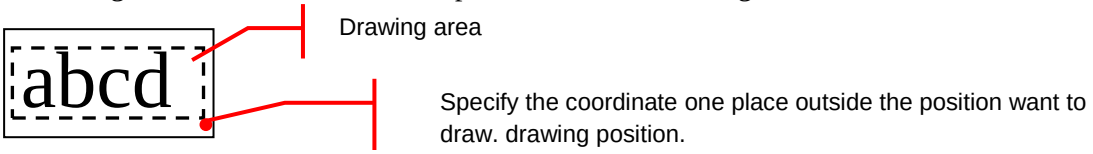
True if successful, and false if an incorrect font is specified.

Remarks

There are two drawing methods, one is to specify a baseline with x and y, and other is to specify a drawing rectangle area with sx, sy, ex and ey.

The baseline is the line (dotted line in the figure below) that is the standard measurement for string.

In the specification by a drawing rectangle area, a string is drawn without including the coordinates of the bottom right corner so that these are 1 pixel outside the drawing area.



When a line break is included in a string, draw by specifying a drawing rectangle area.

When a tab is included in a string, the width of tab is 80 pixels in case of specifying a baseline, and the string is drawn following converting tab codes to half-width spaces in case of specifying a drawing rectangle area.

The origin of the coordinates is the top left corner of the control.

Example

By specifying a baseline

- Start point of baseline: (10, 90)
- String: ABC
- Font: Meiryo UI
- Size: 30
- Color: Black

[mruby]

```
font = FpFont.new("Meiryo UI", 30)
s = "ABC"
draw_text(10, 90, s, font, fpcolor(0, 0, 0))
```

[C++]

```
FpFont* font = new FpFont("Meiryo UI", 30);
char s[] = "ABC";
drawText(10, 90, s, *font, fpColor(0, 0, 0));
```

By specifying a drawing rectangle area
- Top left corner: (10, 20)
- Bottom right corner: (90, 80)
- Text style: Horizontal + vertical center display
- String: ABC
- Font: Meiryo UI
- Size: 30
- Color: Black

[mruby]

```
font = FpFont.new("Meiryo UI", 30)
s = "ABC"
draw_text(10, 20, 90, 80, FP_TEXT_CENTER, s, font, fpcolor(0, 0, 0))
```

[C++]

```
FpFont* font = new FpFont("Meiryo UI", 30);
char s[] = "ABC";
drawText(10, 20, 90, 80, FP_TEXT_CENTER, s, *font, fpColor(0, 0, 0));
```

text_metrics [mruby], textMetrics [C++]

Gets s string size.

[mruby]

```
String →s
FpFont →font
text_metrics(s, font) →FpRect
```

[C++]

```
FP_RECT textMetrics(const char* s, const FpFont& font);
```

For FpRect and FP_RECT, see "FpRect [mruby], FP_RECT [C++]" in "5.2 COORDINATE".
For FpFont, see "5.3 FONT".

Argument

s [in]
String (NULL terminated in C++)
font [in]
Font

Return value

String size

Remarks

Do not include a line break (¥n) in the argument s. If a line break is included, the retrieved string size will not be correct.
To get a size of multiple lines, split the string at line break and get the size for each split string.
Do not include a tab (¥t) in the argument s. If a tab is included, the retrieved string size will not be correct.
The origin of the coordinates is the top left corner of the control.

Example

Gets the size of "ABC" when drawn in 12pt Meiryo UI

[mruby]

```
font = FpFont.new("Meiryo UI", 12)
rect = text_metrics("ABC", font)
font.dispose()
```

[C++]

```
FpFont* font = new FpFont("Meiryo UI", 12);
FP_RECT rect = textMetrics("ABC", *font);
```

screen_size [mruby], getScreenSize [C++]

Gets the size of the screen.

[mruby]

```
screen_size() → FpSize
```

[C++]

```
FP_SIZE getScreenSize();
```

For FpSize and FP_SIZE, see "FpSize [mruby], FP_SIZE [C++]" in "5.2 COORDINATE".

Return value

Size of the entire screen

bitmap [mruby], getBitmap [C++]

Gets WIN32 API HBITMAP of the screen.

[mruby]

```
bitmap() →C pointer
```

[C++]

```
HBITMAP getBitmap();
```

Return value

Handle of the entire screen bitmap (HBITMAP)

bits [mruby], getBits [C++]

Gets a bit image of the screen.

[mruby]

```
bits() →C pointer
```

[C++]

```
unsigned char* getBits();
```

Return value

C language pointer of a bit image of the entire screen.
It is intended to be used in external graphics libraries.
The bit image of the screen is stored in 32-bit RGB value by top-down format.

print_error [mruby], printError [C++]

Displays an error message.

[mruby]

```
String →msg
print_error(msg)
```

[C++]

```
void printError(const char* msg)
```

Argument

msg [in]

Error message (NULL terminated in C++)

Remarks

Error messages are displayed in the following format.

```
Error from user control [control name]
At [event name] event

[message]

Error is detected in "[screen name]" with FP driver:[FP driver version number]
```

Items other than [message] are automatically displayed by the FP driver.
In [event name], the name of the method of the FpUserControl class being executed (initialize/dispose/paint/timer) or load which indicates when screen data is being loading is displayed.

For its operation, see "2.2.3 Anomaly handler creating".

5.5 PROJECT VARIABLE

Functions are for accessing a project variable in user function (C++).

NOTE
For the project variable, refer to FANUC PICTURE Specification (Edition 8.0 or later) (A-42146EN-042/31) "18 Project Variable Management Function of FANUC PICTURE for iHMI".

The only project variables that can be accessed are those referenced in control properties and Ruby script. Project variables that are just defined in the screen editor cannot be accessed.

5.5.1 Data type

[C++]

```
enum FPValType {
      FPVAL_TYPE_INVALID = -1,
      FPVAL_TYPE_BIT = 0,
      FPVAL_TYPE_BYTE,
      FPVAL_TYPE_WORD,
      FPVAL_TYPE_DWORD,
      FPVAL_TYPE_REAL,
      FPVAL_TYPE_LREAL,
      FPVAL_TYPE_STRING
};
```

FPVAL_TYPE_INVALID
Variable undefined
FPVAL_TYPE_BIT
Bit type
FPVAL_TYPE_BYTE
1-byte integer type

FPVAL_TYPE_WORD
2-byte integer type
FPVAL_TYPE_DWORD
4-byte integer type
FPVAL_TYPE_REAL
Single precision floating point type
FPVAL_TYPE_LREAL
Double precision floating point type
FPVAL_TYPE_STRING
String type

5.5.2 Structure

FPVAL [C++]

This structure contains a values of a project variable.
[C++]

```
typedef struct _FPVAL {
    enum FPValType type;
    int reserved;
    union {
        BOOL bit;
        BYTE byte;
        WORD word;
        DWORD dword;
        float real;
        double lreal;
        size_t size;
    };
} FPVAL;
```

Member

type
Data type
reserved
Not used (reserved)
bit
Bit value (when type= FPVAL_TYPE_BIT)
byte
1-byte integer value (when type= FPVAL_TYPE_BYTE)
word
2-byte integer value (when type= FPVAL_TYPE_WORD)
dword
4-byte integer value (when type= FPVAL_TYPE_DWORD)
real
Single precision floating point value (when type= FPVAL_TYPE_REAL)
lreal
Double precision floating point value (when type= FPVAL_TYPE_LREAL)
size
Size of string (when type= FPVAL_TYPE_STRING)

5.5.3 Function

getProjectValue [C++]

Gets a value of an integer or floating point type project variable.

[C++]

```
FPVAL getProjectValue(const char* name);
```

Argument

name [in]

Project variable name (NULL terminated)

Return value

FPVAL type value

Remarks

When the project variable is a string type, the size of the string is retrieved.

When the specified project variable does not exist, FPVAL_TYPE_INVALID is assigned to the FPVAL member "type".

Example

Gets the value of the WORD type project variable "user_data".

[C++]

```
FPVAL val = getProjectValue("user_data");
WORD data = val.word;
```

setProjectValue [C++]

Sets a value of an integer or floating point type project variable.

[C++]

```
BOOL setProjectValue(const char* name, FPVAL val);
```

Argument

name [in]

Project variable name (NULL terminated)

val [in]

Value of FPVAL type to be set

The FPVAL type member "type" (data type) is ignored, so there is no need to set a value in "type".

Return value

TRUE if successful, and FALSE otherwise.

Remarks

The reason why the return value is FALSE is as follows

- Project variable is undefined or unreferenced
- Project variable is string type

Example

Sets 100 to the value of the WORD type project variable "user_data".

[C++]

```
FPVAL val;
val.word = 100;
setProjectValue("user_data", val);
```

getProjectString [C++]

Gets a value of a string-type project variable.

[C++]

```
char* getProjectString(const char* name, char* str, size_t size);
```

Argument

name [in]

Project variable name (NULL terminated)

str [in]

Pointer to buffer to get string

size [in]

Size of buffer to get string

Return value

Pointer to the retrieved string (same as argument str) if success, NULL if failure.

Remarks

The retrieved string is NULL terminated.

The reason why the return value is null is as follows

- Project variable is undefined or unreferenced
- Project variable is not a string type

setProjectString [C++]

Sets a value to a string type project variable.

[C++]

```
BOOL setProjectString(const char* name, const char* str);
```

Argument

name [in]

Project variable name (NULL terminated)

str [in]

Pointer to string

Return value

TRUE if successful, and FALSE otherwise.

Remarks

The string to be set must be NULL terminated.

The reason why the return value is FALSE is as follows.

- Project variable is undefined or unreferenced
- Project variable is not a string type

5.6 BUFFER FOR EXTERNAL FUNCTION CALL

FpFixedBuffer [mruby]

Some C functions, such as OpenGL's vertex specification, require the caller to hold a large array of data as a buffer and specify a pointer to the buffer as a function argument.

The FpFixedBuffer class is designed to support the above C language functions in Ruby script.

First, when creating an instance of the FpFixedBuffer class, specify its buffer data type and array size as shown in the example below.

```
buf = FpFixedBuffer.new("int32_t", 3) // array of 3 int32_t
```

A buffer can be assigned by the set method or by array index,

```
buf.set([1, 2, 3])
buf[0] = 1
```

When passing a pointer to buffer in a function argument, convert it to a C language pointer by the to_p method.

```
cfunc(buf.to_p())
```

NOTE
1 When passing a pointer to buffer in a function argument, the argument type must be defined as a C language pointer (void*) in the external function definition file (funcdef.txt). For details on the external function definition file, see "3.1 EXTERNAL FUNCTION DEFINITION FILE".
2 A buffer is held until the dispose method of the FpFixedBuffer class is executed or the garbage collection is executed.

5.6.1 Member function

Constructor [mruby]

Creates an instance of the FpFixedBuffer class.
[mruby]

```
String→ type
Integer→ num
FpFixedBuffer.new(type, num) → FpFixedBuffer
```

Argument
type [in]
Specify data type as string
num [in]
Number of array
Return value
Instance of the FpFixedBuffer class
Remarks
As shown in the example below, the FpFixedBuffer class can handle fixed-length strings (string type and wstring type) as well as arrays.
buf = FpFixedBuffer.new("string", 260)
However, access to strings by array index is not supported.

set [mruby]

Assigns mruby array to an instance.
[mruby]

```
Array or String→ array
set(array) → C pointer
```

Argument
array [in]
Array or string to be assigned
Return value
C language pointer to instance
Remarks
If the size to be assigned is different from the size of the instance of the FpFixedBuffer class, the size is adjusted internally in the FP driver to match the size of the instance of the FpFixedBuffer class.

get [mruby]

Gets contents of an instance as an mruby array.
[mruby]

```
get → Array or String
```

Return value
Retrieved array or string

[], []= [mruby]
Accesses each element of an instance.
[mruby]

```
Integer→ index
[index]→ mruby variable
mruby value→ value
[index] = value
```

Argument
index [in]
Array index (0 to)
Example

```
buf[2] = 3
val = buf[2] # val = 3
```

Remarks
Access to string by array index is not supported. (An error is displayed during FP driver execution.)
Use get or set to access to string.

to_p [mruby]
Converts an instance to a C language pointer.
[mruby]

```
to_p()→ C pointer
```

Return value
C language pointer to instance

dispose [mruby]
Releases memory allocated inside an instance.
[mruby]

```
dispose()
```

5.7 EXTRA

Functions that indirectly support drawing.

5.7.1 Function

window_handle [mruby], fpWindowHandle [C++]
Gets a WIN32 API window handle of FANUC PICTURE.
[mruby]

```
window_handle() →C pointer
```

[C++]

```
HWND fpWindowHandle();
```

Return value
Window handle

fwlib_handle [mruby], fpFwlibHandle [C++]

Gets a FOCAS2 library handle currently used in FANUC PICTURE.

[mruby]

```
fwlib_handle() →Integer
```

[C++]

```
unsigned short fpFwlibHandle();
```

Return value

FOCAS2 library handle (non-zero value)

Remarks

[C++ only] If this function is executed outside of the FANUC PICTURE managed thread (main, alarm, communication), the return value will be 0.

6 ERROR MESSAGE

6.1 SCREEN EDITOR

The below messages are displayed following "[filename] (number of lines):" during MEM generation.

Funcdef.txt conversion error

Message	Description
[file] cannot be opened	Cannot open ‘file name’ [file].
[file] cannot be written	Cannot write ‘file name’ [file]. Check your disk accident.
error count over 100, conversion is stopped	Number of errors exceeds 100. Conversion will be aborted.
string is not terminated	The string is not properly terminated with double quotes (").
[identifier] is not valid for type name	[identifier] is not valid as a type name.
[identifier] is not a type name	[identifier] is not a type name.
'signed' / 'unsigned' cannot be used for [type]	'signed' / 'unsigned' cannot be used for [type] type.
structure cannot be used for return value	Structure is not allowed for a return value.
[identifier] is not valid for variable / function / structure / library name	[identifier] is not a valid name for a variable/function/structure/library.
pointer declarator(*) and array cannot be used at the same time	Pointer declarators (*) and arrays cannot be used at the same time.
[identifier] is not found	Cannot find [identifier]. Check the definition of [identifier].
[identifier] is not a integer	[identifier] is not an integer. A number of arrays must be an integer.
[identifier] is not valid for array count	[identifier] is not valid as a number of array. Specify an integer greater than or equal to 0.
']' is missing	There is no ']' after a number of array
[identifier] is not valid for identification	[identifier] is not valid as a constant identifier.
[identifier] is already used	[identifier] is already used as an identifier.
'=' is missing	There is no '=' after a constant identifier.
[identifier] is not a value or string	[identifier] is not a value or string.
[identifier] is not a valid number	[identifier] is not a valid number.
';' is missing	There is no ';'.
keyword 'library' is not found	Keyword 'library' is missing.
'(' is missing	Specification of function argument does not begin with '('.
',' or ')' is missing	Specification of function argument does not end with ',' or ')'.
structure [struct] cannot be passed by value	Cannot pass a structure [struct] by value in a function argument.
'out' / 'inout' cannot be used for [type]	Cannot use 'out' / 'inout' for [type] type in specification of function argument. Integer / Floating point / C language pointers cannot be output arguments.
'{' is missing''	Specification of structure member variable does not begin with '{'.
[member] is already used in structure [struct]	The member variable [member] is already used in the structure [struct].
no member in structure [struct]	There are no member variables in the structure [struct]. Check the structure definition.

6.2 FP DRIVER

Screen data loading / Memory release error

Message	Description
There is no name in control	The control does not have a name.
Insufficient memory to create a control	Insufficient memory to create a control.
Memory allocated in a user control is not released	Memory in the control is not released

mruby runtime error

In addition to the standard mruby error messages, one of the following messages is displayed.

Drawing library error

Message	Description
wrong rect in [function] (TypeError)	An invalid rectangle is specified as an argument in the function [function].
wrong number of arguments in [function] (ArgumentError)	The number of arguments of the function [function] is different from specification.
wrong font in [function] (TypeError)	An invalid font is specified as an argument in the function [function].

External function call error

Message	Description
expected [type] at [function] (TypeError)	Arguments of type [type] are required in the function [function].
array is not supported for return value (RuntimeError)	Return value of array is not supported.
structure is not supported for return value (RuntimeError)	Return value of structure is not supported.
[function] does not have an entry point (RuntimeError)	Function [function] is not found.
number of arguments too large (ArgumentError)	There are too many arguments.
invalid type of argument no.[number] (RuntimeError)	The type of the [number]th argument cannot be specified.

Error in buffer class for external function call

Message	Description
memory is disposed in FP fixed buffer (RuntimeError)	Memory has been discarded in the FpFixedBuffer class.
index is not supported for string (IndexError)	The FpFixedBuffer class does not support string index reference.
offset [number] too small for FP fixed buffer(size:[size]) (IndexError)	Index [number] is too small for the FpFixedBuffer class whose size is [size].
offset [number] too large for FP fixed buffer(size:[size]) (IndexError)	Index [number] is too large for the FpFixedBuffer class whose size is [size].

REVISION RECORD

Edition	Date	Contents
01	Mar., 2024	First Registration.