Multi-PROG Script Function Manual

1 Brief Introduction

Multi-PROG has a script function, with the<Menu>option in the menu bar of the system's main interface, which includes two sub menus:< Released Features> and <Local Scripts>.

By selecting the <Local Script> submenu, you can bring up the local script interface and perform operations such as creating, opening, modifying, saving, debugging, and publishing scripts.

By selecting the <Released Features> submenu, the Released Features interface can be brought up. The interface will display the attribute information of all released scripts under the system default path in a table format (such as script name, type, manufacturer, function description, remain times, deadline, etc.). Users can import, export, run, delete, and create new script files for released scripts

.

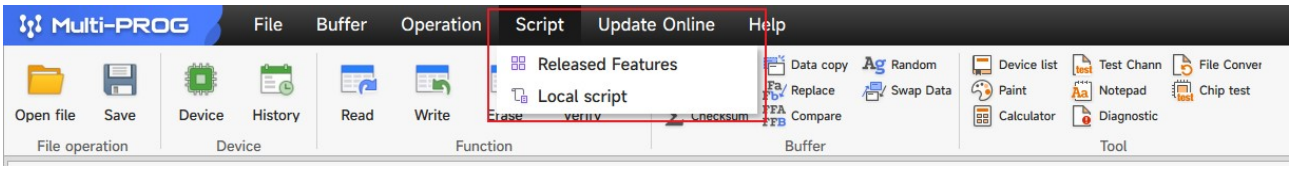


Fig1-1 Script menu

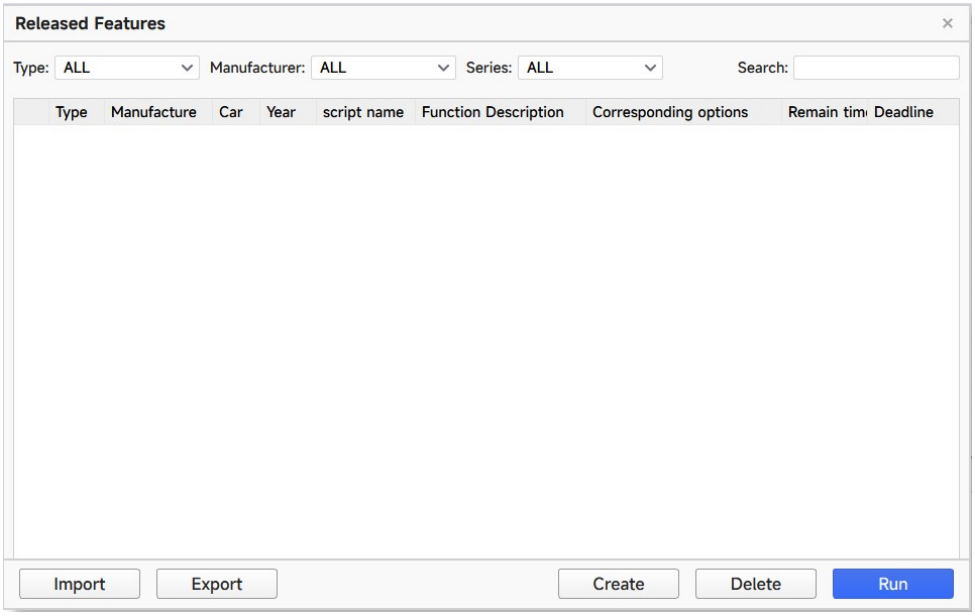


Fig1-2Released Features Interface

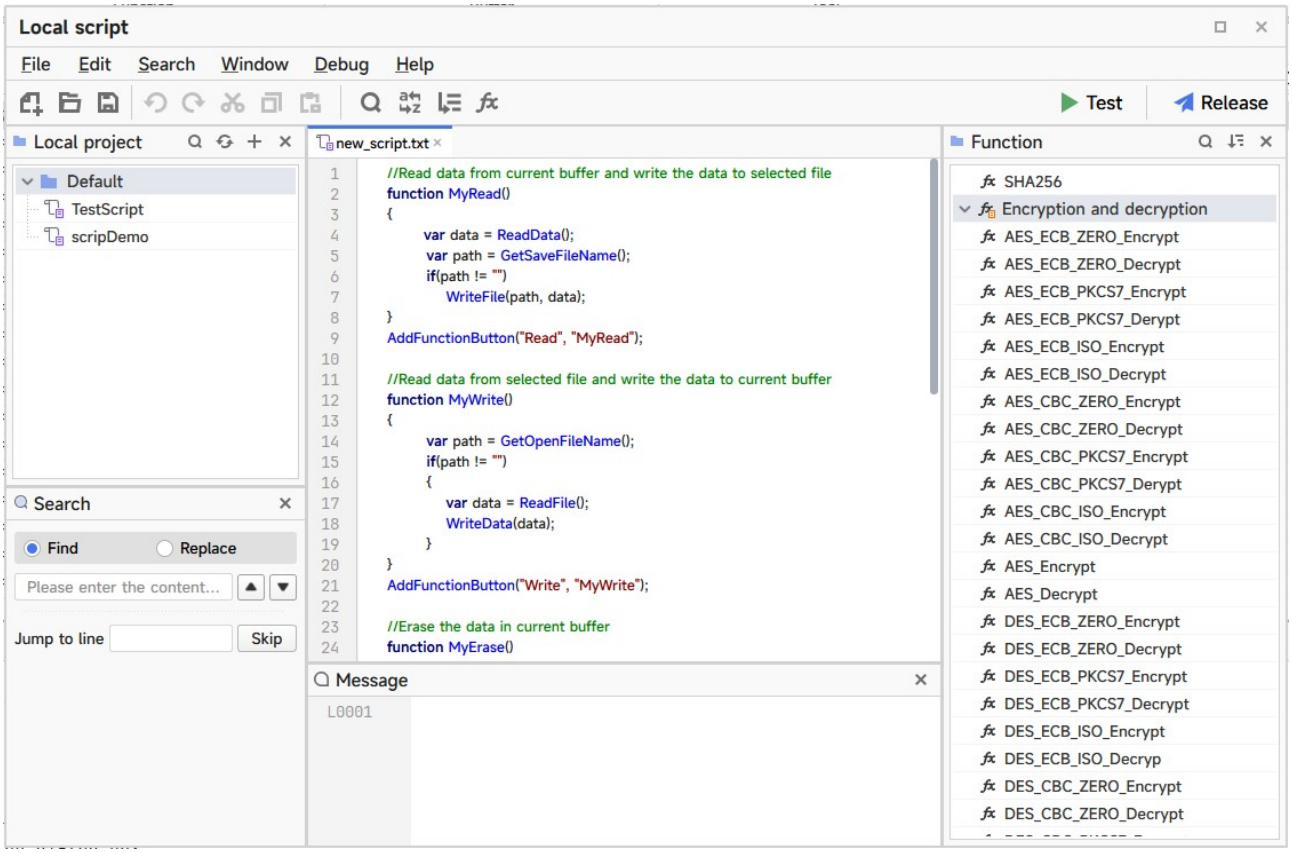


Fig1-3 Local Script Interface

By running the script file, specific function buttons can be added to the script area of the toolbar in the system's main interface. Clicking on a specific function button in the script area will execute the function function corresponding to the button specified in the script. The functions of specific function buttons (actions performed after clicking) are configured by users through simple programming of script files.

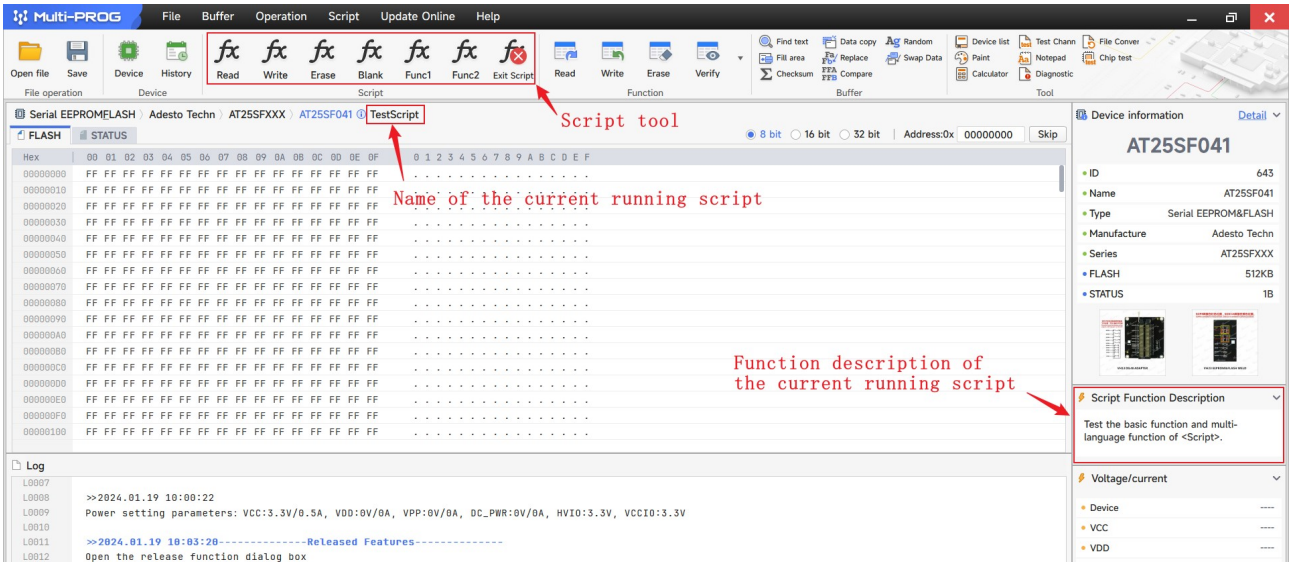


Fig1-4 Script toolbar area

2 <Local Scripts>interface

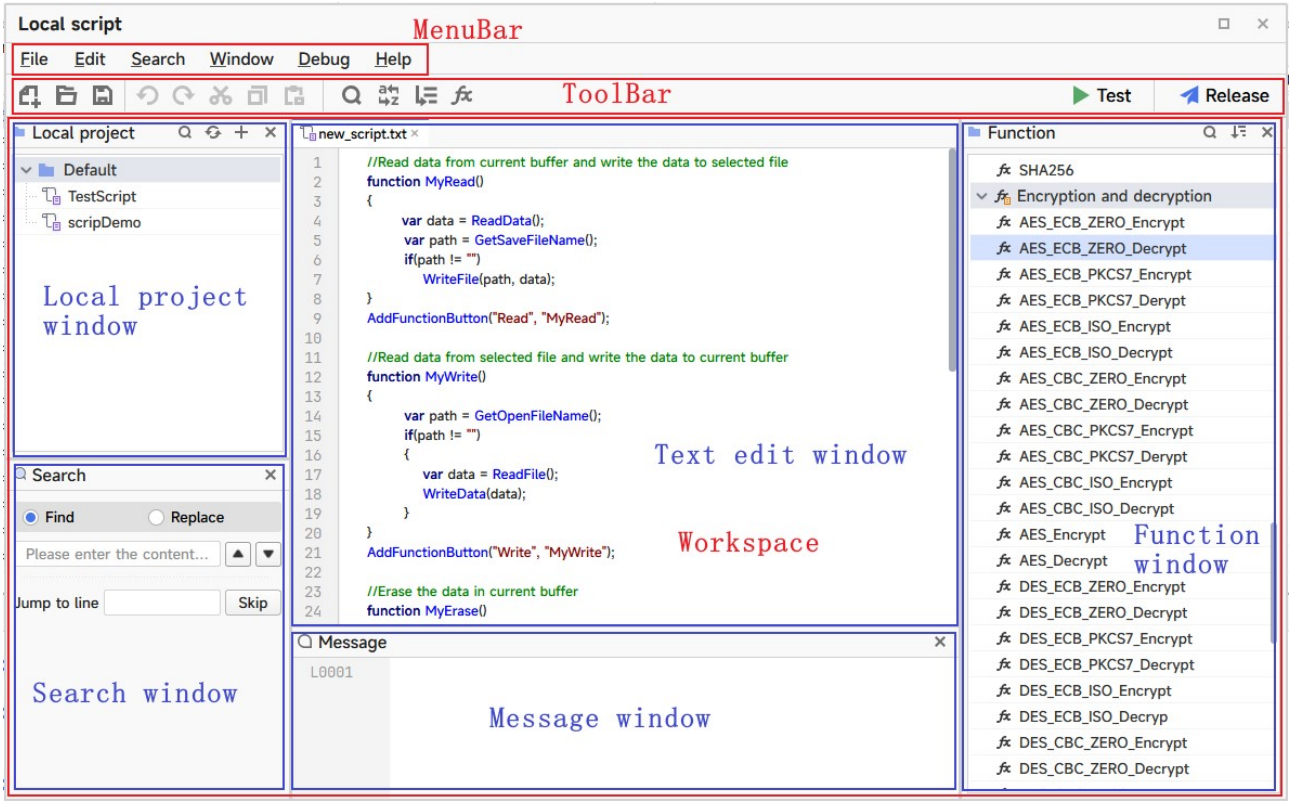


Fig 2-1 Composition of local script interface

The local script interface is shown in the above figure, which mainly consists of a title bar, menu bar, toolbar, and work display area. A brief introduction is as follows:

☑Title bar:

The title bar displays the interface title, with three buttons on the right: minimize, maximize, and close the current interface.

☑Menu bar:

The menu bar contains six commonly used menu function options: file, edit, search, window, debug, and help. The corresponding shortcut keys are displayed on the right side of each submenu.

File menu: includes basic functions such as create, open, save, save as, and exit.

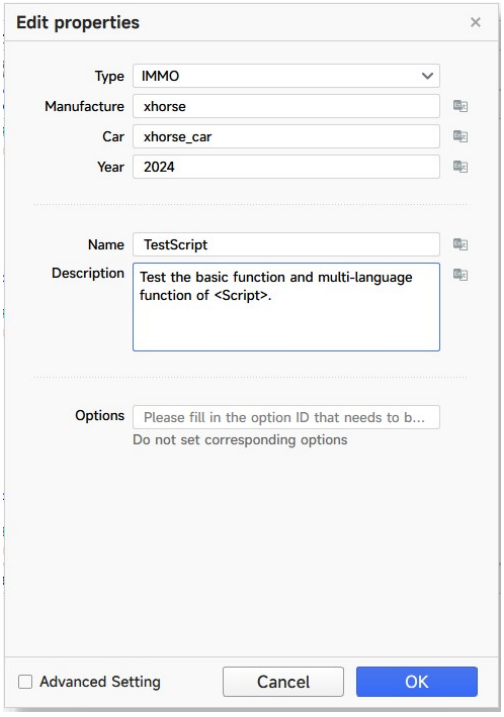


Fig 2-2Create Script Interface

For creating a new script, you need to enter the script information data according to the prompts. If Advanced Settings is checked, you can additionally set the devices supported by the script, the number of available times, as well as whether to enable the script's validity period and the expiration date.

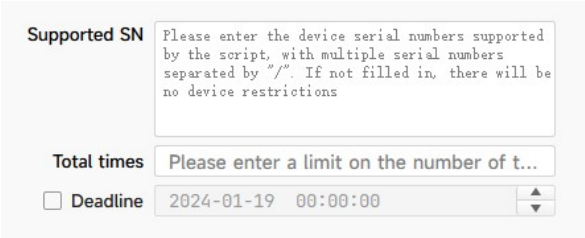


Fig 2-3 Advanced settings for create Script Interface

Edit menu: Contains commonly used editing operations for text files, such as revoke, restore, cut, copy, paste, delete, select all, and clear all.

Search menu: Contains functions for finding, replacing, and redirecting to specified rows.

Window menu: includes local functions, functions, search, and messages. By clicking, you can select whether the corresponding window is displayed in the work display area.

Debug menu: includes testing and publishing functions< Test>Function is to run the currently selected script. If an error occurs during the running process, an error message will be displayed in the message window; If the entire running process is correct, close the local script interface after the script is completed.

Help menu: Contains a help menu that provides help information about the script interface.

☑Toolbar：

The left side contains shortcut operations for creating, opening, saving, revoking, restoring, cutting, copying, pasting, searching, replacing, jumping to a specified line, and customizing functions. The right side contains two function buttons for testing and publishing the currently selected script.

Test：

Check the script syntax and execute it, printing error messages during execution to the message window; If there are no errors during the execution process, close the local script window.

Release：(This feature requires connecting to the network and devices)

Display script option information and the storage path of the generated published scripts, which can be modified before publication as needed. After clicking confirm, execute the syntax check and online verification of the script, and print the error message during the execution process to the message window; If the script has no syntax errors and is validated, generate a published script file (. mjs file) and save it to the specified publishing path (if<Save to Local Default Path>is checked, a copy of the. mjs file will be copied and saved to the default path of the released script in the system)

☑Work Display Area：

The work display area mainly displays and operates on script files. Includes script text editing window, local project window, search window, message window, and function window. You can click on the x symbol in the upper right corner of each window to close it. If you need to reopen it, you need to click on the window menu in the menu bar and select the window you want to display

☑Text Editing Window：

Display the text information of the current script. Allow users to edit and also provide quick editing operations from the right-click menu, making it convenient for users to edit text

☑Local ProjectWindow：

Provide default grouping, allowing for creating new groups and modifying group names. The script files opened through new and open methods will be displayed in the project window for easy management.

Double clicking on a file in the local project window can open the script file and display the script content in the text editing window.

☑Search Window：

Provide text search and replacement for the currently selected script. By specifying the number of lines, the display of the text editing window can be redirected to the specified line, facilitating editing operations on the script.

☑Message Window：

When the script performs testing and publishing operations, error messages generated during the process will be displayed in the message window, making it convenient for users to locate programming errors in the script.

☑Functions Windows：

The function window provides functions such as file, device operation, interface operation, verification, encryption and decryption, and data conversion, making it convenient for users to choose for script programming. Double click a function to insert it at the cursor position in the text editing window. In addition, the function window also provides custom functions. Clicking the “+” button can add user-defined functions, facilitating the reuse of commonly used functions and improving programming efficiency. The “export” and “import” buttons help to share user-defined functions between different computers by exporting them as files or importing them from files. The exported file can be edited externally, but it needs to meet the format requirements.

3 <Released Features>interface

The released function is mainly used to manage and view all released script files (*. mjs files) in the default path of the current system's scripts. This mainly includes a table area for displaying attribute information of all released scripts, a dropdown option box for filtering table entries, function buttons for importing and exporting

released script files, as well as function buttons for creating new scripts, deleting selected released scripts, and running selected released scripts.

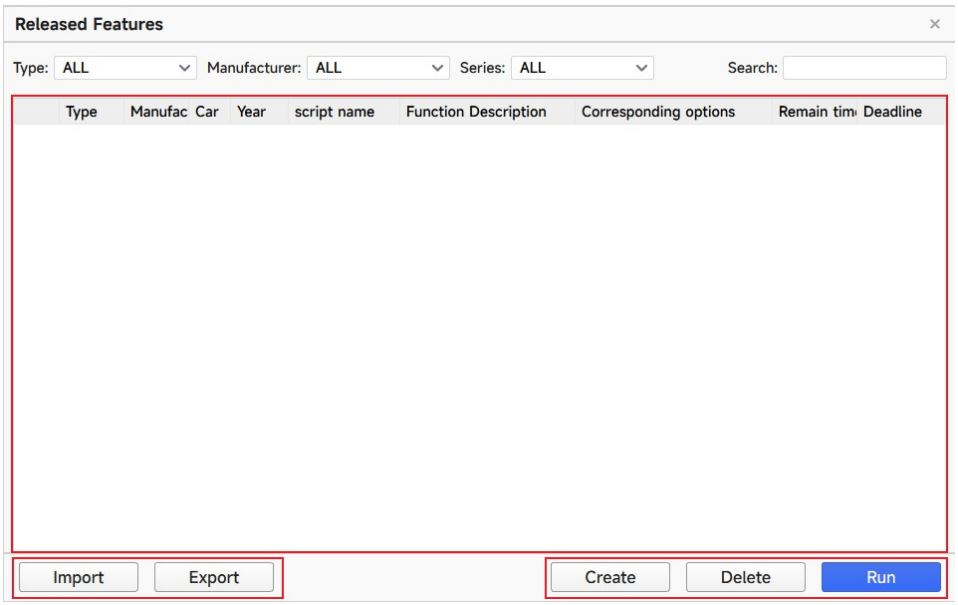


Fig 3-1Main interface composition of released function

The import button can copy the published script under the specified file path to the default script path of the current system, and display the newly added script information in the table.

The export button can copy the selected released script files in the table to the specified file path.

The new button has the same new function as the local script interface.

The delete button is used to delete the selected released script files in the table under the default script path of the current system.

The run button is used to execute the selected released script file in the table.

4 Script programming related: basic syntax

Multi-PROG scripts are written in JavaScript language, which is a lightweight scripting language. The basic syntax of JavaScript is introduced below.

4.1 Data Types

JavaScript data types are mainly divided into two categories: value types (basic data types) and reference data types (object types). Value types mainly include string, number, Boolean, null type, undefined type, and symbol type; Referencing data types mainly includes objects, arrays, and functions. The key explanations for each data type are shown in the table below.

Table 4-1 JavaScript data type

Value Type	String	“Apple”、’watermallon’(Any text enclosed in single or double quotation marks)
	Number	5、3.14、123e5(It can be with or without decimal point, and Scientific notation can be used)
	Boolean	true、false
	Null	null(Clear variables by setting their values to null)
	Undefine	(indicates that the variable does not contain a value)
	Symbol	(represents a unique value)
Referen-cing data types	Object	{name:’John’, age:26}
	Array	[1,2,3,4]
	Function	function funcName(arg) {}

Among them, String are the most commonly used in script programming, and this will be emphasized in the introduction.

String idioms store and process text. Each character in the string can be accessed through the index position (note: the index starts from 0).

If you want to use quotation marks in a string, you can use them by adding escape characters or using quotation marks that are different from the string quotation marks (such as "Call him by 'Herry' not 'Potter').

Table4-2 String common properties and methods

Attribute or Method	Describe	Example

length	Returns the length of a string	var str = “Xhorse”;
		var len = str.length;
		//len is6

charAt()	Returns the character at the specified position	var str = “Xhorse”;
		var ch = str.charAt(3);
		//chisr

concat()	Connect two or more strings and return a new string	var str1 = “Hello ”;
		var str2 = “Xhorse”;
		var str3 = str1.concat(str2);
		//str3is“Hello Xhorse”

startsWith()	Check if the string starts with the specified substring	var str = “Hello world, welcome to Xhorse”; var res = str.startsWith(“Hello”); //res istrue

endsWith()	Determine whether the current string ends with the specified string	var str = “Hello world, welcome to Xhorse”;
		str.endsWith(“Xhorse”); //returntrue
		str.endsWith(“xhorse”); //returnfalse

indexOf()	Returns the first occurrence of a specified string value in a string, if not found, returns -1	var str = “Hello world, welcome to Xhorse”; var res = str.indexOf(“or”); //res is 7
lastIndexOf()	Search the string from back to front and calculate the last occurrence of the returned string starting from the starting position. If not found, return -1	var str = “Hello world, welcome to Xhorse”; var res = str.lastIndexOf(“or”); //res is 26
includes()	Find if the string contains the specified substring	var str = “Hello world, welcome to Xhorse”; var res = str.includes(“world”); //res is true
search()	Find the specified substring in the string, if not found, return -1	var str = “Hello world, welcome to Xhorse”; var res = str.search(“world”); //res is true

match()	Find a match for one or more regular expressions	var str = “Hello world, welcome to Xhorse”;
		var res = str.match(/or/g);
		//res is array[‘or’, ‘or’]

repeat()	Copy strings a specified number of times and concatenate them together to return	var str = “Xhorse”;
		var res = str.repeat(2);
		//res is ”XhorseXhorse”

replace()	Find matching substrings in the string and replace substrings that match the regular expression	var str = “Hello world, welcome to Xhorse”; var res = str.replace(“or”, “xxx”); //res is ”Hello wxxxld, welcome to Xhorse”
replaceAll()	Find a matching string in the string and replace all substrings that match the regular expression	var str = “Hello world, welcome to Xhorse”; var res = str.replaceAll(“or”, “xxx”); //res is ”Hello wxxxld, welcome to Xhxxxse”
split()	Separate strings into string arrays	var str = “Apple,Banana,Cat,Dog”; var res = str.split(“,”); //res is array[‘Apple’, ‘Banana’, ‘Cat’, ‘Dog’]

slice()	Extract a string fragment and return the extracted part in the new string	var str = “Hello world, welcome to Xhorse”;
		var res = str.slice(2, 7);
		//res is ”llo w”

substr()	Extract a specified number of characters from the starting index number of a string	var str = “Hello world, welcome to Xhorse”;
		var res = str.substr(2, 7);
		//res is ”llo wor”

substring()	Extract characters between two specified index numbers in a string	var str = “Hello world, welcome to Xhorse”;
		var res = str.substring(2, 7);
		//res is ”llo w”

toLowerCase()	Convert string to lowercase	var str = “Xhorse”;
		var res = str.toLowerCase();
		//res is ”xhorse”

toUpperCase()	Convert string to uppercase	var str = “Xhorse”;
		var res = str.toUpprtCase();
		//res is ”XHORSE”

trim()	Remove whitespace on both sides of the string	var str = “ Xhorse ”;
		var res = str.trim();
		//res is ” Xhorse”

4.2 Literal and Variables

Literal are fixed values, and 3.14, 'John', [40100,1,5,25] are specific examples of different types of literals.

The opposite of the literal value is the variable. A variable is essentially a memory space used to store different types of data, so its value allows for variation. Define variables using the keyword var and assign values to them using an equal sign. If var length=6;

4.3 Statements and Comments

1. JavaScript, using the Unicode character set, is case sensitive (i.e. myVariable and MyVariable represent two different names).

2. JavaScript uses semicolons to separate statements, and a semicolon needs to be added at the end of each executable statement.

3. JavaScript ignores extra spaces, which can be added to the script to improve code readability.

4. You can use a backslash (\) to wrap the code in a text string. The specific details are as follows:

```
var str = “Hello \
          World”;
```

5.Complex operations can be achieved by using operators in statements.

Assignment, arithmetic, bit operations： = + - * /

Conditions, comparisons, logical operations： == != <>

6.Not all JavaScript statements are commands, and using double slashes // can annotate text to prevent the execution of code lines. For multi line comments, start with/* and end with */.

In addition, JavaScript statements typically start with a statement identifier and execute that statement. The commonly used statement identifiers are as follows:

Table 4-3 JavaScript Useful Sentences

Statement identifier	Describe	Form
var	Declare a **variable**	var num;

function	Define a **function**	function Function Name (FunctionParameters){
		Function body, executed code block;
		}

if…else…	**Branch jump statement**, used to perform different actions based on different conditions
	Note: The else section can be omitted, or else if can be used to continue nesting
		if (condition) {
		If the condition is true, execute the code block;
		} else {
		If the condition is false, execute the code block;
		}

for	**Loop statement,** when the conditional statement is true, can execute the code block a specified number of times	for (var i = 0; i < 5; i++) {
		Code block being executed;
		}

for…in	**Loop statement**, used to traverse the properties of data or objects (loop operations on the properties of arrays or objects)	for (**Variable** in **array**) {
		block;
		}
		Example：
		const numbers = [45, 4, 9, 16,25];
		var txt = “”;
		for (var x in numbers) {
		txt += numbers[x]
		}

while	**Loop** statement, execute statement block when conditional statement is true	while(condition) {
		block;
		}

do…while	**Loop** statement, execute a statement block, and continue executing the statement block when the conditional statement is true (The difference from the while loop is that do... while will first execute the code block once before making conditional judgments)	do { block; } while(condition);

switch	**Branch jump** statement, used to perform different actions based on different conditions
	First, calculate a switch expression, then compare the value of the expression with the label of each case. If there is a match, execute the associated code block
		switch(Expression) {
		case label1:
		block;
		break;
		case label2:
		block;
		break;
		default:
		Default Code Block;
		}

continue	Skip an iteration in the loop	
break	Jumping out of a switch branch or loop	

4.4 Function

Functions can be referenced repeatedly. By writing statements within functions, it is possible to simplify the writing of repetitive statements, achieve reuse, and reduce code volume. Calling a function is equivalent to executing the statement within the function. Function parameters can be empty, and function return values can also be left blank. The function form is as follows:

```
function funcName(arg1, arg2) {
    //do something
    return; //Return value (optional)
}
```

When called, it is called through funcName (function parameter), with the same number and type of parameters as specified when defining the function.

Example：

```
function addFunc(a, b) {
    return a + b;
}
var c = addFunc(2, 3); // The result of c is 2+3=5
```

The above defines a function called addFunc, whose specific function is to add the two parameters passed and return the sum result. The function call passes arguments 2 and 3, and the result of the function call is 5, which is assigned to variable c. The final value of variable c is 5.

5 Script programming related: data types

5.1 Uint8Array

The byte array Uint8Array is widely used in built-in function parameters and function returns. Therefore, this section provides a detailed explanation of the usage of this data type.

The Uint8Array type represents an 8-bit unsigned integer array, whose contents are initialized to 0 when created. After creation, elements in the array can be referenced in the form of objects or using array indices.

5.1.1 Create byte array

The index of a byte array **starts from 0**. There are several methods to create byte arrays:

1、 Create through a variable number of parameters**Uint8Array.of()**

Example：

```
var arr = Uint8Array.of(0x00, 0x01, …);
```

Create from an array or iterable object**Uint8Array.from()**

Example：

```
var s = new Set([1,2,3]);
var arr = Uint8Array.from(s);
```

2、 Create by **specifying the length n**

```
var arr = new Uint8Array(n); //n is the size of the array, with default data of 0
arr[0] = 0x00;
arr[1] = 0x01;
…
arr[n-1] = …;
```

3、 Create through array conversion

Example：

```
var arr = new Uint8Array([0x00, 0x01]);
```

4、 Create by copying

Example：

```
var arr1 = Uint8Array.of(0x00, 0x01);
var arr2 = new Uint8Array(arr1);
```

5.1.2 Common Properties and Methods of Byte Arrays

Several functions have been customized here, and the following table is used to demonstrate the example call usage.

```
function isSatisfied(element, index, array) {//Numbers that meet multiples of 10
    return (element %10 == 0);
}

function isOdd(element, index, array) { //Odd number judgment
    return (elem % 2 == 1);
}

function isEven(element, index, array) { // Even number judgment
    return (elem % 2 == 0);
}
```

Table 5-1 Uint8Array Common attributes and methods

Attributes and methods	**Describe**	**Example**
length	Return the number of elements in Uint8Array	var arr = Uint8Array.of(1,2,3,4); var len = arr.length; //len is4

byteLength	Return Uint8Array length (in bytes)	var arr = Uint8Array.of(1,2,3,4); var len = arr.byteLength; //lenis4

at()	Return the element at the index position (if the index is negative, count backwards from the last item in the array)	var arr = Uint8Array.of(1,2,3,4);
		var item = arr.at(2); //item is 3
		item = arr.at(-1); //item is 4

copyWithin()	Copy the sequence of elements in the array to the starting position from the target (parameter 1). Copy taken from interval [parameter 2, parameter 3)	var arr = Uint8Array.of(1,2,3,4,5,6,7,8); arr.copyWithin(3,1,3); //arr is [1,2,3,2,3,,6,7,8]

every()	Test whether all elements in the array pass the provided function implementation test	var arr = Uint8Array.of(10,50,70);
		arr.every(isSatisfied);
		//All elements are multiples of 10, resulting in true

fill()	Fill an array with static values	var arr = Uint8Array.of(0,0,0,0); arr.fill(4,1,3); //item is [0,4,4,0]
filter()	Create a new array that contains all elements tested using the provided method	var arr = Uint8Array.of(2,23,12,5,7); var res = arr.filter(isEven); //res is [2,12]
find()	Find the first element that satisfies the specified function. If not found, return undefined	var arr = Uint8Array.of(4,5,8,9,12); var res = arr.fine(isOdd); //res is 5
findIndex()	Find elements that meet the specified function, and if found, return their index; If not found, return -1	var arr = Uint8Array.of(4,5,8,9,12); var res = arr.fineIndex(isOdd); //res is 1
findLast()	Similar to find(), the difference is that the search direction is opposite	var arr = Uint8Array.of(4,5,8,9,12); var res = arr.fineLast(isOdd); //res is 9
findLastIndex()	Similar to findIndex(), the difference is that the search direction is opposite	var arr = Uint8Array.of(4,5,8,9,12); var res = arr.fineLastIndex(isOdd); //res is 3

forEach()	Execute the provided function once for each element in the byte array	var arr = Uint8Array.of(10,50,70);
		arr.every((element) => console.log(element));
		//Perform printing on each element

includes()	Judge whether a byte array contains a certain element	var arr = Uint8Array.of(10,20,30,40);
		arr.includes(20); //Result istrue
		arr.includes(50); ///Result isfalse

indexOf()	Returns the index position of the given element, and if it does not exist, returns -1 (the second parameter is optional, indicating the index position to start the search)	var arr = Uint8Array.of(10,50,70); var res = arr.indexOf(50); //res is 1
join()	Concatenate all elements of a byte array into a single string	var arr = Uint8Array.of(10,20,30,40); var res = arr.join('-'); //res is '10-20-30-40'

keys()	Returns a new array iterator that contains the key for each index in the array	var arr = Uint8Array.of(10,20,30,40);
		var keys = arr.keys();
		keys.next();
		var res = keys.next().value; //res = 1

lastIndexOf()	Similar to indexOf(), the difference is that the search direction is opposite	var arr = Uint8Array.of(10,20,50,50,50,60);
		arr.lastIndexOf(50,5); // Result is4
		arr.lastIndexOf(50,3); // Result is3

map()	Create a new byte array, where each element is the result of the original byte array calling the specified function	var arr = Uint8Array.of(25,36,49); var res = arr.map(Math.sqrt); //res is [5,6,7]
reverse()	Invert Byte Array	var arr = Uint8Array.of(10,20,30,40); arr.reverse(); //arr is [40,30,20,10]
set()	Read data from a specified array and store it in a byte array starting from index position	var arr = new Uint8Array(8); arr.set([1,2,3],3); //arrResult is[0,0,0,1,2,3,0,0]
slice()	Slice operation, obtain a portion of the byte array, and specify the range by specifying the interval (left closed and right open)	var arr = Uint8Array.of(10,20,30,40,50); var res = arr.slice(1,3); //arr is [20,30]
some()	Is there any element in the byte array that meets the requirements of the specified function	var arr = Uint8Array.of(1,3,5,7); arr.some(isEven); // Result isflase
sort()	Sort byte arrays	var arr = Uint8Array.of(40,10,50,20,30); arr.sort(); //arr is [10,20,30,40,50]
subarray()	Subarray operation to obtain a portion of a byte array, specifying a range by specifying an interval (left closed and right open)	var arr = Uint8Array.of(10,20,30,40,50); var res = arr.subarray(1,3); //arr is [20,30]

toReversed()	Returns a new byte array, where the elements of the array are the inverted result of the original byte array	var arr = Uint8Array.of(10,20,30,40,50);
		var res = arr.toReversed();
		//arr is [10,20,30,40,50]
		//res is [50,40,30,20,10]

toSorted()	Returns a new byte array, where the elements of the array are the sorted results of the original byte array	var arr = Uint8Array.of(40,10,50,20,30);
		var res = arr.toSorted();
		//arr is [40,10,50,20,30]
		//res is [10,20,30,40,50]

values()	Returns a new array iterator that contains the value of each index in the array	var arr = Uint8Array.of(10,20,30,40);
		var tmp = arr.values();
		tmp.next();
		var res = tmp.next().value; //res = 20

with()	Returns a new byte array, where the element derived from the given index is replaced with the given value	var arr = Uint8Array.of(1,2,3,4,5);
		var res= arr.with(2,6);
		//arr is [1,2,3,4,5]
		//res is [1,2,6,4,5]

5.1.3 Differences between slice Uint8Array.slice and subarray Uint8Array.subarray operations

Both slicing and subarray operations can obtain a portion of a byte array, specifying a range by specifying an interval (left closed and right open). The use of the two is basically the same, with the main difference being the memory space occupation. The detailed usage and differences are as follows:

```
var arr = new Uint8Array([1, 2, 3, 4, 5, 6]);
var subarr = arr.subarray(3,5);    // 4,5
var slicarr = arr.slice(3,5);    //4,5
subarr[0] = 9; // arr: [1,2,3,4,9,6]        subarr: [9,5]        slicearr: [4,5]
```

As can be seen, Uint8Array.subarray shares memory space with the original array; Uint8Array.slice is not shared with the original array memory and is located in a separate space.

5.2 Description of enumeration types

Table 5-2 Script function custom enumeration type

User input type selection **type**	AES Encryption and decryption algorithm **level**	Encryption and decryption algorithm grouping mode **mode**	Filling encryption and decryption algorithm method **padding**
InputString	AES_128	ECB	ZERO
InputInt	AES_192	CBC	PKCS7
InputHex	AES_256	CFB	ISO
InputFloat		OFB	

6 Script programming related: built-in functions

6.1 File

6.1.1 ReadFile

Function: Read data from a specified file
Function prototype: ReadFile(*path*)
Parameter：
path <String> path to read file
Return value:
<Byte array>
Example:

```
var data = ReadFile("C:/temp.bin");
```

6.1.2 WriteFile

Function: Write data to a specified file
Function prototype: WriteFile(*path*, *data*)
Parameter：
path<string>write path to file

data<Byte array>data to be written

Return value:

<Number>write data length

Example:

var length = WriteFile("C:/temp.bin", data);

6.1.3 ReadAndEmbedFile

Function: Read the specified file data and embed the file into the script when publishing (note: only accept the absolute path string literal of the file as the is parameter, and cannot embed the file when passing variables or other types)

Function prototype: ReadAndEmbedFile(*path*)

Parameters:

path<string> path to read file

Return value:

<Byte array>Read data

Example:

var data = ReadAndEmbedFile("C:/temp.bin");

6.1.4 GetOpenFileName

Function: Get the file selected by the user

Function prototype: GetOpenFileName()

Parameters: None

Return value:

<string>Path to the file to be opened

Example:

var path = GetOpenFileName();

6.1.5 GetSaveFileName

Function: Get the file name saved by the user

Function prototype: GetSaveFileName()

Parameters: None

Return value:

<string>Path to the file to be saved

Example:

var path = GetSaveFileName();

6.1.6 ReadFromSelectedFile

Function: Read data from the user's selected file

Function prototype: ReadFromSelectedFile()

Parameters: None

Return value:

<Byte array>Read data

Example:

var data = ReadFromSelectedFile();

6.1.7 WriteToSelectedFile

Function: Read data from the user's selected file
Function prototype: WriteToSelectedFile(*data*)
Parameters:
Data<byte array>data to be written
Return value:
<Number>Write data length
Example:
var length= WriteToSelectedFile(data);

6.2 Device operation

6.2.1 SelectBuffer

Function: Select specified buffer
Function prototype: SelectBuffer(*name*)
Parameters:
name<string>buffer name
Return value:
<Boolean>Whether the switch to the specified buffer was successful
Example:
var success = SelectBuffer("BUFFER NAME");

6.2.2 BlankCheck

Function: Perform blank detection on the current buffer
Function prototype: BlankCheck()
Parameters: None
Return value:None
Example:
BlackCheck();

6.2.3 WriteData

Function: Write data to the current buffer
Function prototype: WriteData(*data*)
Parameter:
data<Byte array>Data to be written
Return value:None
Example:
WriteData(data);

6.2.4 ReadData

Function: Read data from the current buffer
Function prototype: ReadData()
Parameter: None
Return value:

<Byte array>Data read from the current buffer

Example:

var data = ReadData();

6.2.5 EraseData

Function: Erase current buffer data

Function prototype: EraseData()

Parameter: None

Return value:None

Example:

EraseData();

6.2.6 IdCheck

Function: Perform ID detection

Function prototype: IDCheck()

Parameter: None

Return value: None

Example:IDCheck();

6.2.7 PinCheck

Function: Perform pin detection

Function prototype: PinCheck()

Parameter: None

Return value: None

Example:PinCheck();

6.2.8 OpenPowerSetting

Function: Open the power settings window

Function prototype: OpenPowerSetting

Parameter: None

Return value: None

Example:OpenPowerSetting():

6.3 Interface operation

6.3.1 Print

Function: Output text to the message window below in specified color

Function prototype: Print(*text, color*)

Parameters:

text <Number/Boolean/String> the text to be output

color the color of the text (optional, defaults to black), supported types:

1.text type enumeration: MP_Print_Normal/MP_Print_Success/MP_Print_Error

2.color enumeration: MP_Color_Red/MP_Color_Green/MP_Color_Blue

3.custom color: eg #FF0000, rgb(30, 114, 255)

Return value: None
Example:Print(“TEXT”)。

6.3.2 Message

Function: Pop up a message prompt window, which prompts the message specified by the *text*
Function prototype: Message(*text*)
Parameter：
text <Number/Boolean/String>Prompt Text
Return value: None
Example:Message(“MESSAGE”)。

6.3.3 Question

Function: Generate a message pop-up and retrieve the user's selected operation
Function prototype: Question(*type, buttons, title, text*)
Parameter：
type <Type:MsgWarning / MsgSuccess / MsgError>three types of messages to choose from
buttons <Type:BtnCancel / BtnOK / BtnNo / BtnYes>four types of buttons to choose from (note: multiple buttons can be set through the "bitwise OR" operation, such as BtnCancel | BtnOK)
title <String>Title
text <String>prompt text
Return value:
<<BtnCancel / BtnOK / BtnNo / BtnYes>The type of button clicked by the user
Example:var ret = Question(MsgWarning, BtnCancel | BtnOK, “Alert”, “Message”);

6.3.4 GetInput

Function: Obtain user input values (cancellation is not allowed)
Function prototype: GetInput(*type, text*)
Parameter：
type< Type：InputString/InputInt/InputHex/InputFloat>four types of type selection to choose
text<String>User input prompt text
Return value:
<string/integer/HEX/float>user input value
Example:var input = GetInput(InputString, “PLEASE INPUT A STRING”);

6.3.5 RequestInput

Function: Get user input (allow cancellation)
Function prototype: RequestInput(*type, text*)
Parameter：
type <Type:InputString / InputInt / InputHex / InputFloat>four types of type selection to choose
text
<String>User input prompt text
Return value:
<jsonobject,{“button”:BtnOk, “input”: “text”}, button value is BtnOK or BtnCancel，input value is User input>

Example:var jsObj = RequestInput(InputString, “PLEASE INPUT A STRING”);

6.3.6 ShowDataInBufferArea

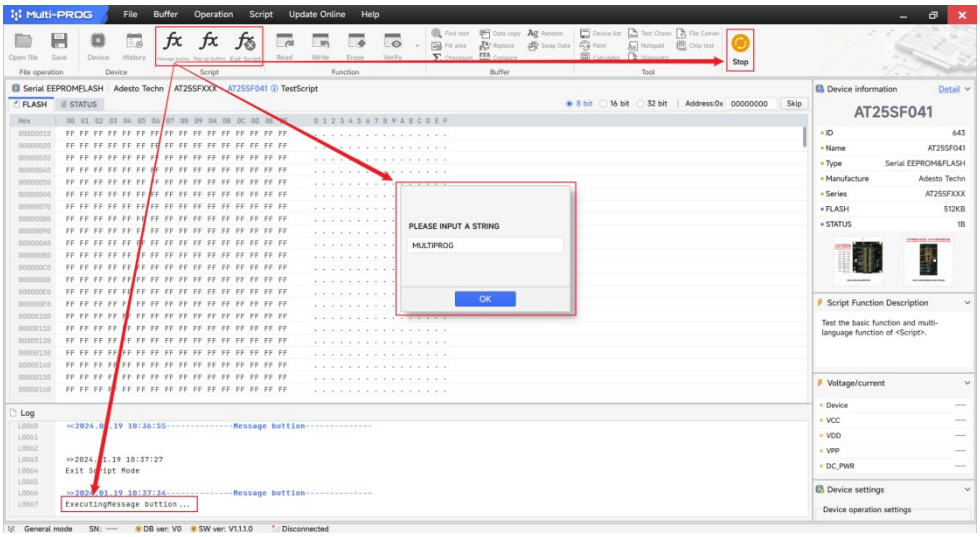
Function: Display specified data in a buffer on the interface
Function prototype: ShowDataInBufferArea(data)
Parameter：
data <Byte array>Data that needs to be displayed in the buffer
Return value:None
Example:ShowDataInBufferArea(data);

6.3.7 AddFunctionButton

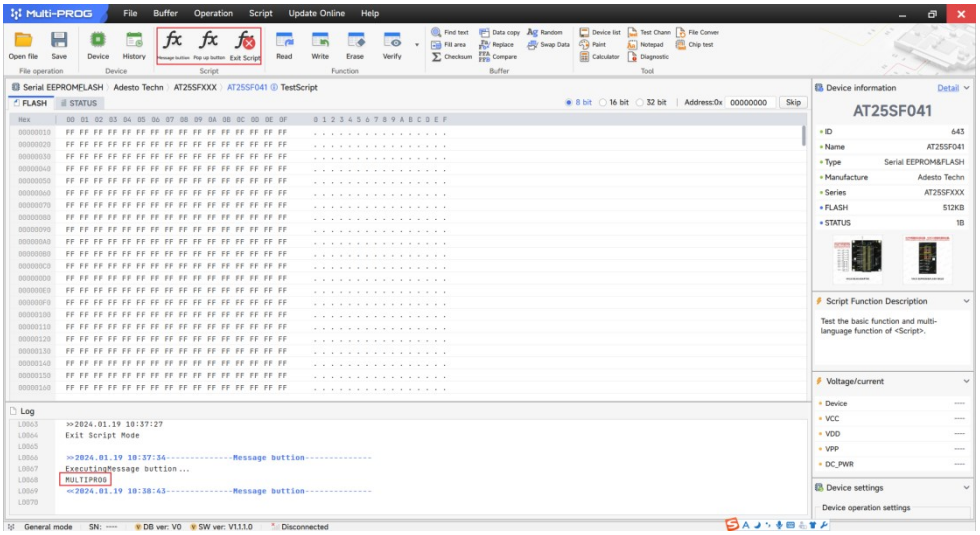
Function: When running in specified language environment, add a button on the interface to execute the specified function
Function prototype: AddFunctionButton(*text*, *functionName, languageId*)
Parameter：
text<String>Text displayed on the button
functionName<String>Function executed after clicking
languageId<Number>Specify the language to use (-1: Do not specify language, 0: Simplified Chinese, 1: English Note: default parameter is -1)
Return value:None
Example:

```
function MyFunc1()
{
    var input = GetInput(InputString, "PLEASE INPUT A STRING:");
    Print(input);
}
AddFunctionButton("Message button", "MyFunc1");

function MyFunc2()
{
    Message("Message pop-up text information");
}
AddFunctionButton("Pop up button", "MyFunc2");
```



(a)Enter a string in the prompt box



(b)Print the result after clicking OK

Fig 6-1Click on custom message button result

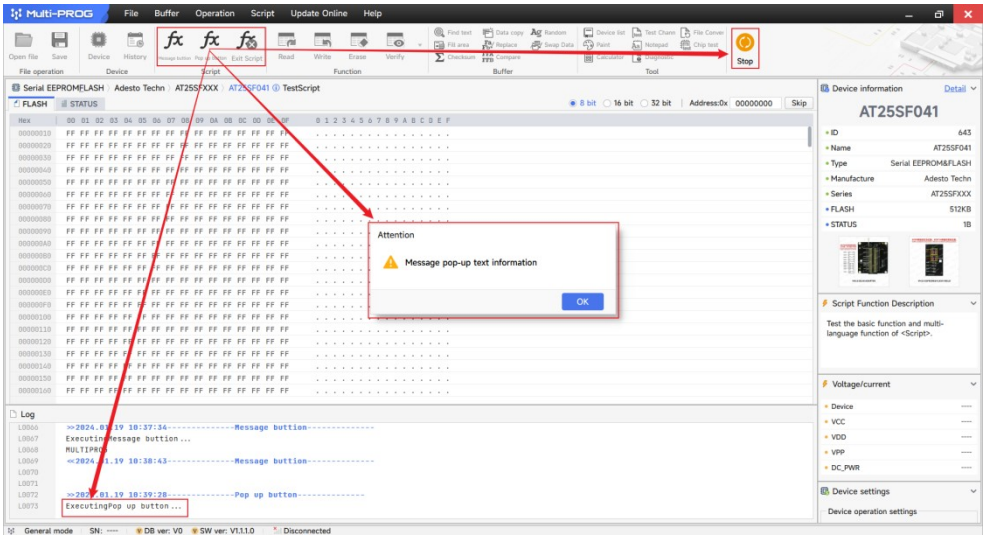


Fig 6-2Click the custom pop-up button result

6.4 Verification

6.4.1 MD5

Function: Calculate the MD5 value of data
Function prototype: MD5(*data*)
Parameter：
data<Byte array>requires calculation of MD5 data
Return value:
<Byte array>Calculation result
Example:
var data = Uint8Array.of(0x12, 0x13, 0x14, 0x15);
var result = MD5(data);
//result is ： 0x5e, 0xe1, 0x64, 0x1d, 0x70, 0xa8, 0x26, 0x54, 0x2e, 0xff, 0xcd, 0xde, 0x13, 0x33, 0xaf, 0x76

6.4.2 CRC16

Function: Calculate the CRC16 value of data
Function prototype: CRC16(*data*, *init*)
Parameter：
data<Byte array>requires computing data for CRC16
init <Byte array>Initial value (can be omitted)
Return value:
<Byte array>Calculation result
Example:
var data = Uint8Array.of(0x12, 0x13, 0x14, 0x15);
var result = CRC16 (data);
//result is ： 0x56, 0x3a

6.4.3 CRC32

Function: Calculate the CRC32 value of data
Function prototype: CRC32(*data*, *init*)
Parameter：
data<Byte array>requires computing data for CRC32
init <Byte array>Initial value (can be omitted)
Return value:
<Byte array>Calculation result
Example:
var data = Uint8Array.of(0x12, 0x13, 0x14, 0x15);
var result = CRC32 (data);
//result is ： 0x86, 0x47, 0x6e, 0x9f

6.4.4 SHA1

Function: Calculate the SHA1 value of data
Function prototype: SHA1 (*data*)
Parameter：

data<Byte array>needs to calculate SHA1 data

Return value:

<Byte array>Calculation result

Example:

var data = Uint8Array.of(0x12, 0x13, 0x14, 0x15);

var result = SHA1 (data);

//result is ：0x80, 0xcc, 0x72, 0x43, 0xf7, 0xd5, 0x21, 0xf7, 0xde, 0xf1, 0x1a, 0xd6, 0xad, 0x75, 0x02, 0x1c, 0x68, 0xe1, 0xa3, 0xea

6.4.5 SHA256

Function: Calculate the SHA256 value of data

Function prototype: SHA256 (*data*)

Parameter：

data<Byte array>requires calculation of SHA256 data

Return value:

<Byte array>Calculation result

Example:

var data = Uint8Array.of(0x12, 0x13, 0x14, 0x15);

var result = SHA256 (data);

//result is ：0x69, 0x69, 0x4c, 0xc0, 0xdd, 0x66, 0x46, 0xb5, 0x4f, 0x69, 0x34, 0xe4, 0xd2, 0x77, 0x8f, 0x79, 0xe9, 0x4e, 0x54, 0xa2, 0xd9, 0xca, 0x44, 0xed, 0x87, 0x79, 0x9c, 0xc5, 0xdb, 0x40, 0x0b, 0x6a

6.5 Encryption

6.5.1 AES_ECB_ZERO_Encrypt / DES_ECB_ZERO_Encrypt

Function: AES128/DES encryption of data, using ECB mode and 0 padding

Function prototype:

AES_ECB_ZERO_Encrypt(*data*, *key*)

DES_ECB_ZERO_Encrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array>Encryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F); //16 bytes

var result = AES_ECB_ZERO_Encrypt(data, key);

6.5.2 AES_ECB_ZERO_Decrypt / DES_ECB_ZERO_Decrypt

Function: AES128/DES decryption of data, using ECB mode and 0 padding

Function prototype:

AES_ECB_ZERO_Decrypt(*data*, *key*)

DES_ECB_ZERO_Decrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array>Decryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F); //16 bytes

var result = AES_ECB_ZERO_Decrypt(data, key);

6.5.3 AES_ECB_PKCS7_Encrypt / DES_ECB_PKCS7_Encrypt

Function: AES128/DES encryption of data, using ECB mode and PKCS7 padding

Function prototype:

AES_ECB_PKCS7_Encrypt(*data*, *key*)

DES_ECB_PKCS7_Encrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array>Encryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F); //16 bytes

var result = AES_ECB_PKCS7_Encrypt(data, key);

6.5.4 AES_ECB_PKCS7_Decrypt / DES_ECB_PKCS7_Decrypt

Function: AES128/DES decryption of data, using ECB mode and PKCS7 padding

Function prototype:

AES_ECB_PKCS7_Decrypt(*data*, *key*)

DES_ECB_PKCS7_Decrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array>Decryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_ECB_PKCS7_Decrypt(data, key);

6.5.5 AES_ECB_ISO_Encrypt / DES_ECB_ISO_Encrypt

Function: AES128/DES encryption of data, using ECB mode and ISO padding

Function prototype:

AES_ECB_ISO_Encrypt(*data*, *key*)

DES_ECB_ISO_Encrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Encryption result

Example:

```
var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes
var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes
var result = AES_ECB_ISO_Encrypt(data, key);
```

6.5.6 AES_ECB_ISO_Decrypt / DES_ECB_ISO_Decrypt

Function: AES128/DES decryption of data, using ECB mode and ISO padding

Function prototype:

AES_ECB_ISO_Decrypt(*data*, *key*)

DES_ECB_ISO_Decrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Decryption result

Example:

```
var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes
var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes
var result = AES_ECB_ISO_Decrypt(data, key);
```

6.5.7 AES_CBC_ZERO_Encrypt / DES_CBC_ZERO_Encrypt

Function: AES128/DES encryption of data, using CBC mode and 0 padding

Function prototype:

AES_CBC_ZERO_Encrypt(*data*, *key*)

DES_CBC_ZERO_Encrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Encryption result

Example:

```
var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes
```

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_CBC_ZERO_Encrypt(data, key);

6.5.8 AES_CBC_ZERO_Decrypt / DES_CBC_ZERO_Decrypt

Function: AES128/DES decryption of data, using CBC mode and 0 padding

Function prototype:

AES_CBC_ZERO_Decrypt(*data*, *key*)

DES_CBC_ZERO_Decrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Decryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_CBC_ZERO_Decrypt(data, key);

6.5.9 AES_CBC_PKCS7_Encrypt / DES_CBC_PKCS7_Encrypt

Function: AES128/DES encryption of data, using CBC mode and PKCS7 padding

Function prototype:

AES_CBC_PKCS7_Encrypt(*data*, *key*)

DES_CBC_PKCS7_Encrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Encryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_CBC_PKCS7_Encrypt(data, key);

6.5.10 AES_CBC_PKCS7_Decrypt / DES_CBC_PKCS7_Decrypt

Function: AES128/DES decryption of data, using CBC mode and PKCS7 padding

Function prototype:

AES_CBC_PKCS7_Decrypt(*data*, *key*)

DES_CBC_PKCS7_Decrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Decryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_CBC_PKCS7_Decrypt(data, key);

6.5.11 AES_CBC_ISO_Encrypt / DES_CBC_ISO_Encrypt

Function: AES128/DES encryption of data, using CBC mode and ISO padding

Function prototype:

AES_CBC_ISO_Encrypt(*data*, *key*)

DES_CBC_ISO_Encrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Encryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_CBC_ISO_Encrypt(data, key);

6.5.12 AES_CBC_ISO_Decrypt / DES_CBC_ISO_Decrypt

Function: AES128/DES decryption of data, using CBC mode and ISO padding

Function prototype:

AES_CBC_ISO_Decrypt(*data*, *key*)

DES_CBC_ISO_Decrypt(*data*, *key*)

Parameter：

data<Byte array>Data to be encrypted

key<Byte array, 16 bytes>16 byte key

Return value:

<Byte array> Decryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_CBC_ISO_Decrypt(data, key);

6.5.13 AES_Encrypt / DES_Encrypt

Function: AES/DES encryption of data

Function prototype:

AES_Enrypt(*level*, *mode*, *data*, *key*, *iv*, *padding*)

DES_Enrypt(*mode*, *data*, *key*, *iv*, *padding*)

Parameter：

level <AES_128 / AES_192 / AES_256> AES digits

mode <ECB / CBC / CFB / OFB>packet mode

data <Byte array>Data to be encrypted

key < Byte array, 16 bytes>16 byte key

iv <Byte array>Initial vector

padding <ZERO / PKCS7 / ISO>fill style

Return value:

<Byte array> Encryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var result = AES_Encrypt(AES_128, ECB, data, key, iv, ZERO);

6.5.14 AES_Decrypt / DES_Decrypt

Function: Perform AES/DES decryption on data

Function prototype:

AES_Derypt(*level*, *mode*, *data*, *key*, *iv*, *padding*)

DES_Derypt(*mode*, *data*, *key*, *iv*, *padding*)

Parameter：

level <AES_128 / AES_192 / AES_256>AES digits

mode <ECB / CBC / CFB / OFB>packet mode

data <Byte array>Data to be encrypted

key < Byte array, 16 bytes>16 byte key

iv <Byte array>Initial vector

padding <ZERO / PKCS7 / ISO>fill style

Return value:

<Byte array> Decryption result

Example:

var data = Uint8Array.of(0x55, 0xC3, 0x82, 0x3A); //Multiple bytes

var key = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0A, 0x0B, 0x0C, 0x0D, 0x0E, 0x0F);//16 bytes

var iv = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07);

var result = AES_Decrypt(AES_128, ECB, data, key, iv, ZERO);

6.6 Others

6.6.1 ByteArrayToHexString

Function: Convert an array to a hex adecimal string

Function prototype: ByteArrayToHexString(*data*)

Parameter：

data<Byte array>Array to be converted

Return value:

<hex string>conversion result

Example:

```
var arr = Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07);
var strHex = ByteArrayToHexString(arr);
Print(strHex);
//strHex result is ： 0001020304050607
```

6.6.2 HexStringToByteArray

Function: Convert hex adecimal strings to arrays

Function prototype: HexStringToByteArray(*str*)

Parameter：

str<hex string>The hex string to be converted

Return value:

<Byte array>conversion result

Example:

```
var str= Uint8Array.of(0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07);
var arr = HexStringToByteArray (str);
Print(arr);
//arr result is ： 0,1,2,3,4,5,6,7
```

6.6.3 GetLanguageId

Function: Obtain the language used by the script during runtime

Function prototype: GetLanguageId()

Parameter: None

Return value:

<Number>Current language ID (0: Simplified Chinese,1： English)

Example:var languageId = GetLanguageId();

6.6.4 CurrentDateTime

Function: Get the current date and time

Function prototype: CurrentDateTime(format)

Parameter：

format <string>The date and time format that needs to be obtained, such as”yyyy-MM-dd hh:mm:ss:zzz”

Return value:

<string>date, time (the return value corresponding to the above example parameters is “2023.12.25 15:01:33:349”)

Example:var date = CurrentDateTime(“yyyy-MM-dd hh:mm:ss:zzz”);

6.7 User Defined Functions

In addition to built-in functions, the scripting feature also allows for custom functions, making it easier to write scripts. Hover the mouse over the custom function item in the function window, and an add button will appear on the right. Click the button and edit the custom function in the pop-up custom function window. After saving, a corresponding function option will be added to the function window.

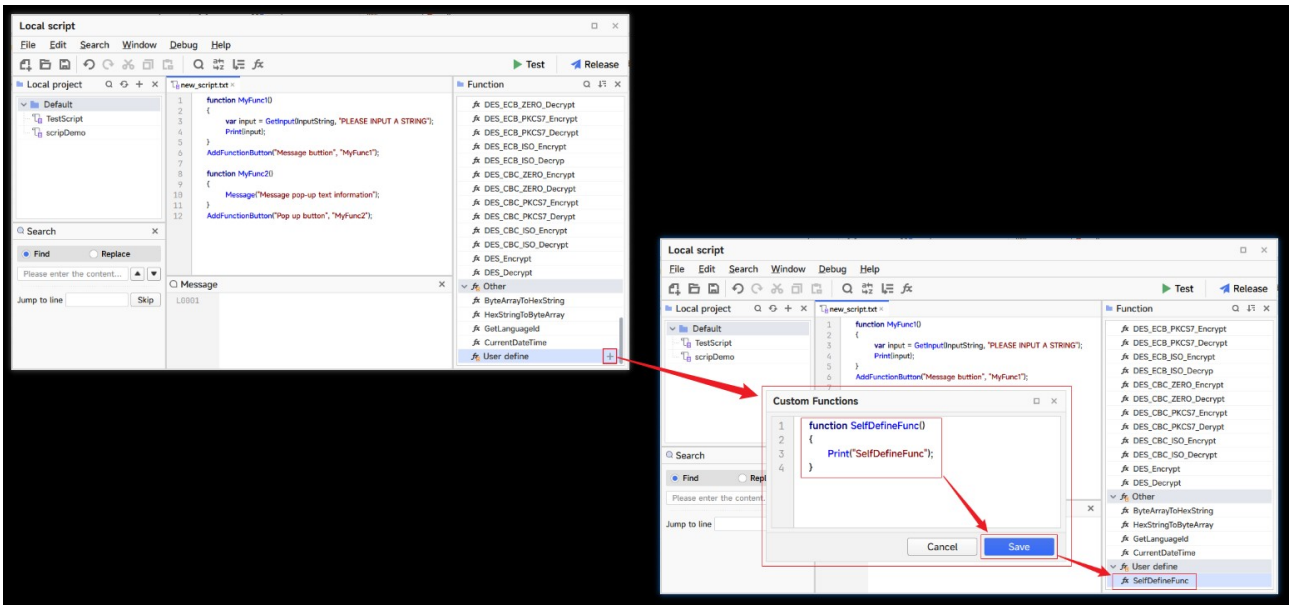


Fig 6-3User Defined Functions

7 Script Casedemo

This section introduces a case study to provide a detailed explanation of the specific process of using the script function, enabling users to have a clearer understanding of the<script>function. Generally, the specific process steps for publishing a script are: creating a new script file (. txt file) → editing script file → testing script file →releasingscript file (→ viewing information on released script files).

7.1 Creating Script Files

In two main ways:

1） Left click on the system's main interface menu bar<Script>→<Local Script>to bring up the local script interface. Click on the menu bar<File>→<Create>to bring up the attribute configuration interface for creating a new script file.

2） Left click on the system's main interface menu bar<Script>→<Released Features>to bring up the Released Features interface. Click the<New>function button to bring up the local script interface and open the attribute configuration interface for creating a new script file.

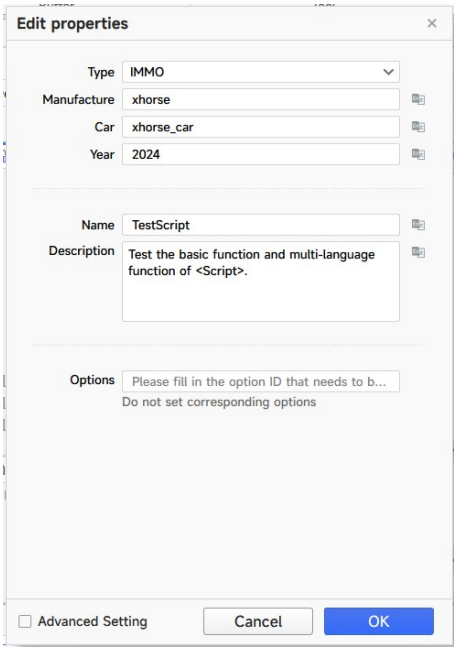


Fig 7-1Process: Create a new script file

Fill in the script attribute information according to the requirements, click confirm, and the specified script file (. txt file) will be generated and displayed in the local project window. The newly created script file will have content for user reference by default.

Note: The name attribute is a property of the script and differs from the file name of the script.

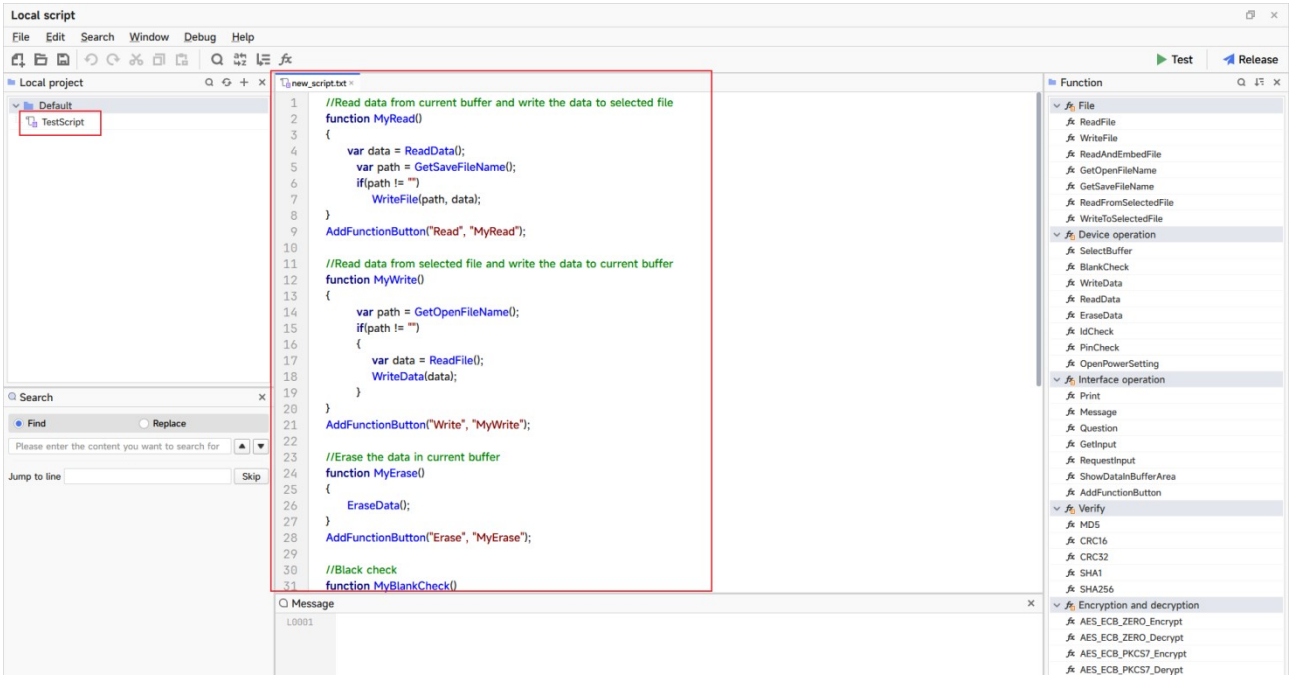


Fig 7-2New Script File Result

7.2 Edit script file

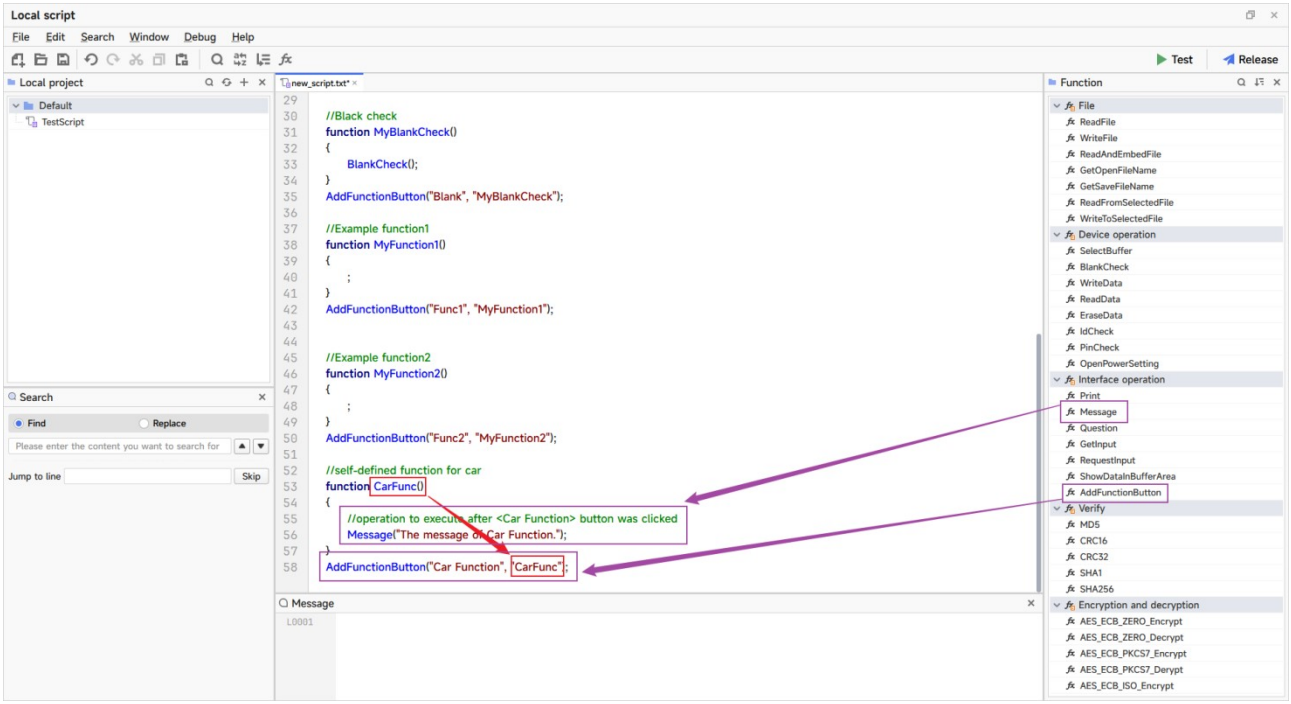


Fig 7-3Process: Editing Script Files

After creating a new script file, you can edit it. The function of the script file can be added by selecting the function window on the right.

This example uses AddFunctionButton () to add a button named "Car Function" in the script area of the toolbar in the main interface window. Clicking this button will execute the custom function CarFunc(). In the CarFunc() function, a Message () function in the function window has been added to open a message pop-up and display the specified prompt message.

7.3 Test Script Files

Click the test button on the right side of the toolbar to test the edited script file.

As shown in the figure below, the "Car Function" function button appears in the system's main interface toolbar. Clicking this button will execute the bound function, resulting in a pop-up message and displaying the executed message pop-up content.

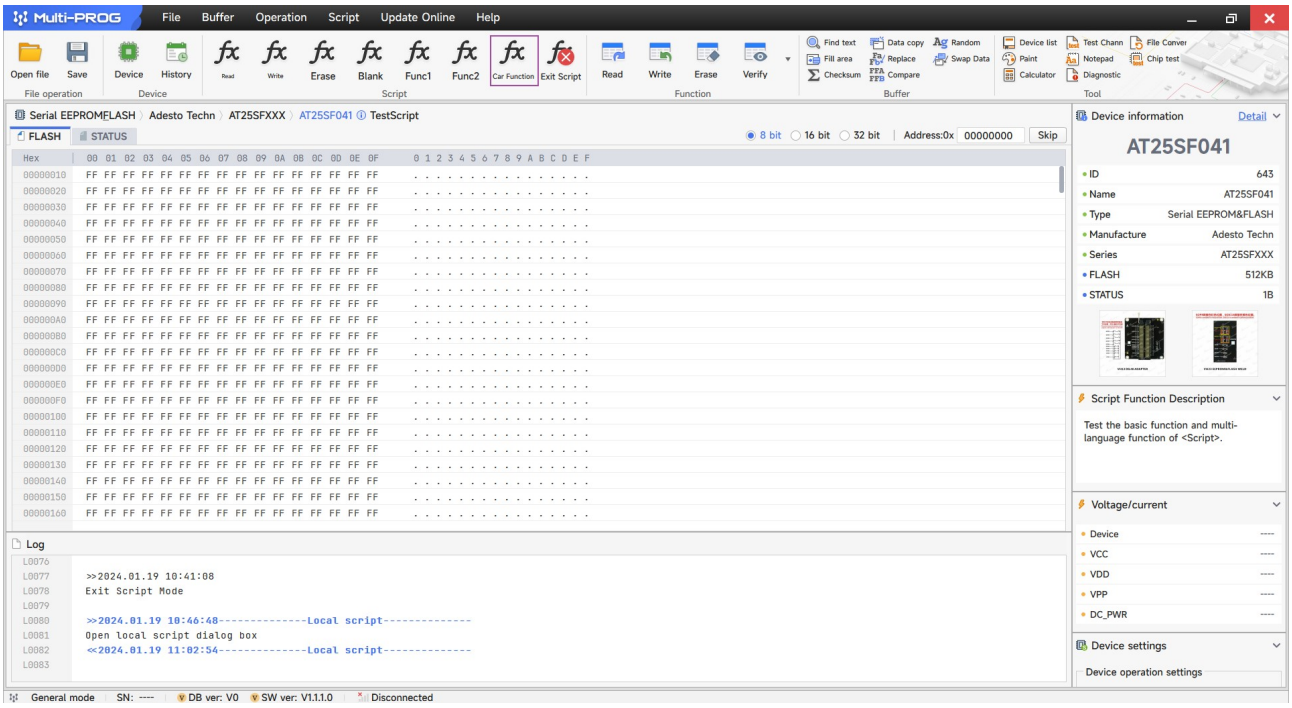


Fig 7-4 Process: Test script file

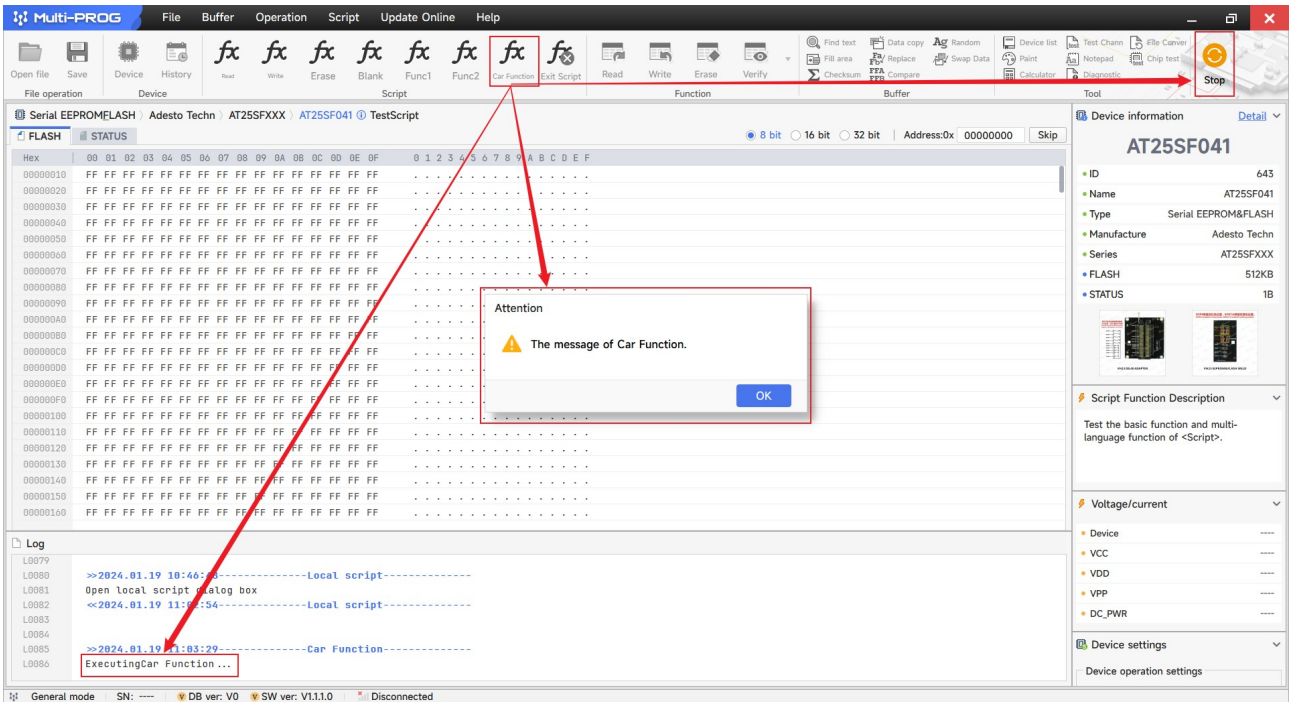


Fig 7-5Process: Test script file (click function button)

7.4 Publish script files

Click the release button on the right side of the toolbar, and a release options interface will pop up, displaying the attribute information of the script file, allowing users to make modifications, and providing the release path and the option to "save to local default path at the same time".

Under the premise of connecting to the network and devices, click confirm to perform syntax check and online verification on the script file (. txt file). After verification, a released script (. mjs file) will be generated and saved to the specified path.

If there is no connection between the device and the network, a message will pop up saying "Device not

connected, please connect the device and try again!".

7.5 Viewing Published Script File Information

Return to the main interface of the system, left click on the menu bar<Script>→<Releaseed Functions>to bring up the released function interface. You can view the released scripts. Select the script in the table to run it and perform other file management operations.

8 Multilingual Function

Using multilingual function, the same script can be"test" or "release" in different language systems, and can display specified language texts on the interface, avoiding the need to write corresponding scripts for multiple language systems.

As shown in Fig 8-1 and Fig 8-2, when creating a new script, configure the corresponding text to be displayed in the English language system at the corresponding location. When the script is executed, the specified text will be displayed at the desired location.

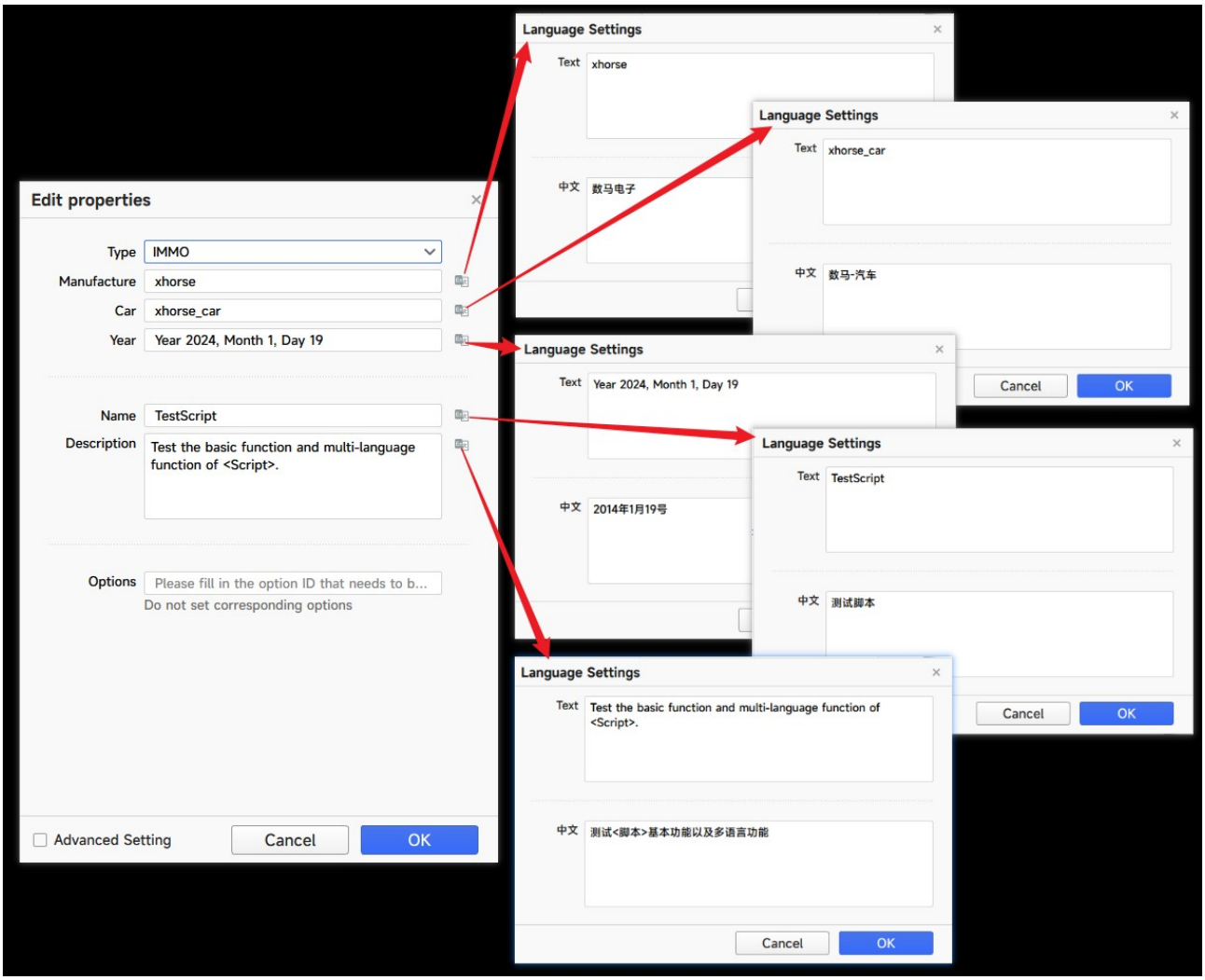


Fig 8-1Configure Script Multilingual Function

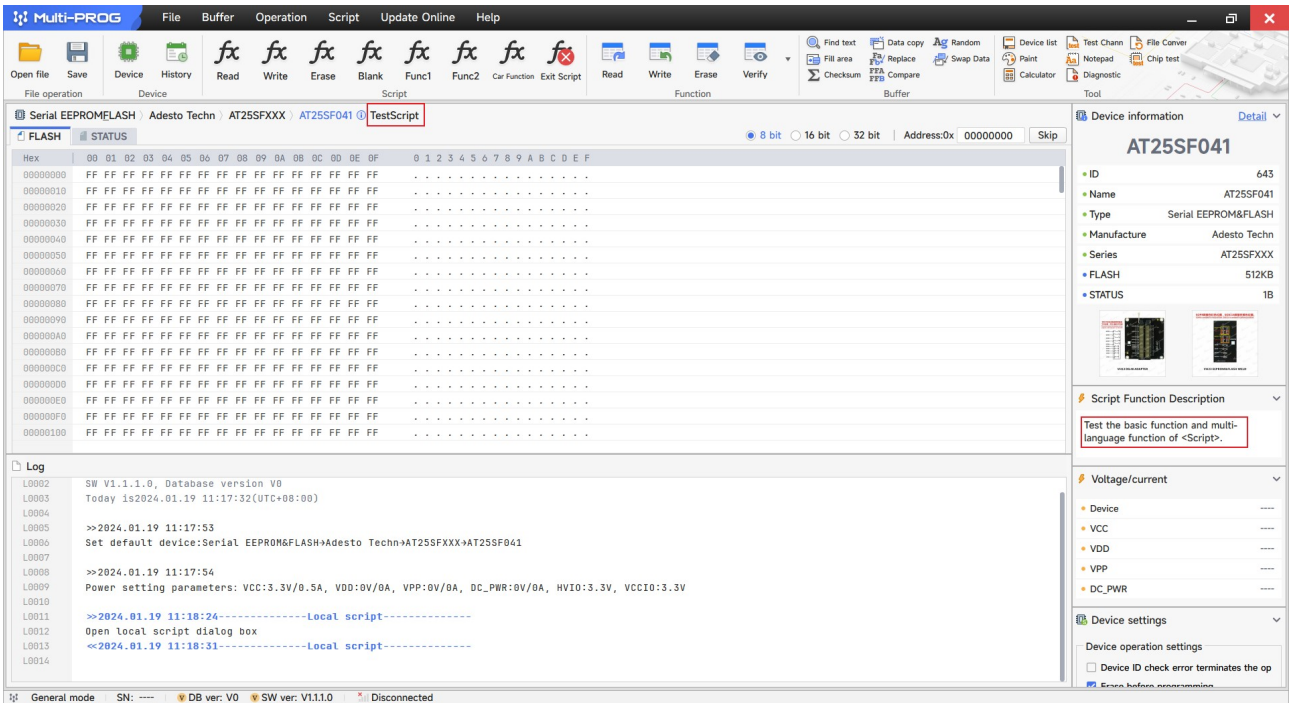


Fig 8-2 System interface during script execution in English mode

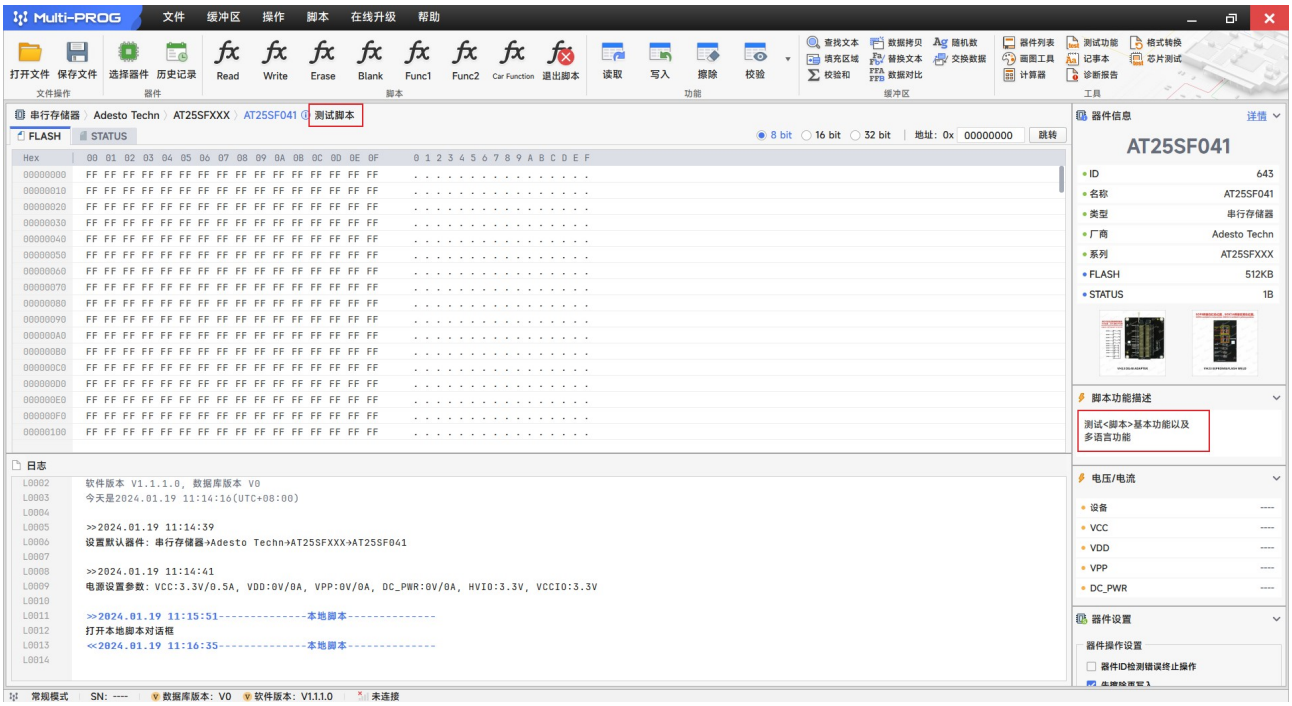


Fig 8-3System interface during script execution in Chinese mode