# HansRobot

# Script API Library

Author： Fanghua Huang  Date： 2016 FEB

Shenzhen Han`s Motor S&T Co.,Ltd

# 0 Inrtoduction

## 0.1 Product Information：

| | |
|---|---|
| **Product Name** | HansRobot Script API Library |
| **Product Manager** | Chongchong Liang |
| **Author** | FangHua Huang |
| **First Submitted** | 2015-02-01 |

## 0.2 Edit History：

| Version NO. | Date | Author | Modified Content |
|---|---|---|---|
| V1.0.1 | 2015-02-01 | FangHua Huang | Original Version |
| V1.0.2 | 2016-05-01 | FangHua Huang | Synchronized the modified content of the Script Compiler |
| V1.0.3 | 2016-09-19 | FangHua Huang | Updated the API Interface, examples added |
| V1.0.4 | 2016-09-23 | Lijuan Liang | Bugs fixed |
| V1.0.5 | 2016-11-22 | Lijuan Liang | Added API Descriptions |
| V1.0.6 | 2016-12-26 | Lijuan Liang | Examples updated |
| V1.0.7 | 2017-03-06 | FangHua Huang | APIs Removed: hm_interrupt, hm_continue, hm_get_setCmd_state;<br>APIs Modified:<br>hm_get_multiendat_pos - Changed the parameter type from double to int |
| V1.0.8 | 2017-03-20 | Lijuan Liang | APIs added : hm_set_payloa , hm_set_safeStop, hm_set_zeroforce |

## 0.3 Theme of this Document：

- ✧ Help the developers on the developing and maintenance of the compiler.
- ✧ A Quick Start for users using Han’s Robot Script APIs.

# 1 Instruction

## 1.1 Abstract

Han's Robot Script Compiler is a derivative of Standard C compiler, applying the same grammar as standard C language. However, for the higher efficiency of the compiler, we added some limitations to the grammar, please see the details below.

大族机器人脚本编译器依托于标准的 C 编译器，脚本语法采用 C 语言语法，但为了让机器人编译器更加简练高效，我们对脚本的语法作了一些限制，具体请阅读以下文档。

## 1.2 Variables

- 变量的声明与定义遵循 C 语言中变量的使用语法。
- The definition and declaration of variables stay the same as in standard C Language.

## 1.3 Functions

- 自定义函数的声明与 C 语言函数的声明语法一致。
- The definition and declaration of functions stay the same as Standard C Language.
- 返回类型只支持返回字符串指针(char*)。
- The type of Function Return Value is limited to Char Pointer Only (char*).
- 函数的参数只支持 C 语言基本类型 int 型和 double 型。

- Only the type “int” and “double” can be applied to the function parameters.
- 保留函数（大族机器人函数）以”hm_”开头，自定义函数不能使用与保留函数相同的名称，保留函数见“大族机器人函数”。
- All the reserved functions (Official Han’s Robot APIs) have their names starting with “hm_”. The user defined functions cannot be named the same as any of the reserved functions. For more information about the reserved functions, please see the chapter“Han’s Robot APIs”

## 1.4 Compiling of the Functions

函数以脚本的形式(后缀名为.c 的 C 文件)，通过大族机器人示教器进行编译。

The functions should be stored as .c files and can be compiled using Han’s Robot Demonstrator.

大族机器人示教器会提示编译错误，以帮助用户排除脚本错误。

If there is any grammar Error in the script, error messages will be shown in the demonstrator to help the users debug.

## 1.5 Using the Functions

函数调用格式为：

函数名称,参数 1,参数 2,…参数 n，；

通过 TCP/IP 的方式下发到大族机器人控制系统中。函数名称与各参数之间使用英文逗号分隔，最后以英文逗号和分号结尾。

注意：只有脚本编译成功之后，才能正确调用脚本函数。

Use the format below to call the functions:

Function Name, Parameter1, Parameter2 … Parameter N,;

Please send the package above to the Han’s Robot Control System using TCP/IP protocol. Commas should be inserted between the function name and the parameters, the whole package should end with a comma followed by a semicolon.

# 2 Han’s Robot API

大族机器人内置函数集成了大族机器人控制系统的对外接口，使用者可以在脚本中通过调用大族机器人函数控制机器人，包括运动控制、IO 控制等。

大族机器人内置函数都以字母”hm_”开头，函数名称全部由小写字母组成，自定义函数不能与内置函数重名，详情如下：

Han’s Robot Control System has internally embedded the Control Interface for users. Our official APIs can be called to control the robots, including kinematic control and IO control.

All the reserved functions (Official Han’s Robot APIs) have their names starting with “hm_” and including only lower case characters.

## 2.1 hm_poweron

**Function Description:** Enable the torque of the robot.

**Format:** int hm_poweron(int nRbtID);

**Number of Parameters:** 1

nRbtID：the ID of the robot，starting from 0。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
hm_poweron(0);
```

**Full example:**

```
char* poweron()
{
int nRlt = 0;
int nPowerOnState = 0;
int nErrorState=0;
nRlt =hm_poweron(0);
if(0 != nRlt)
    return "poweron,Fail,;";
while(1)
{
```

```
    nRlt=hm_get_state(0,2,&nPowerOnState);
    if(nPowerOnState==1 && nRlt==0)
      {
          break;
      }
    nRlt=hm_get_state(0,3,&nErrorState);
    if(nErrorState==1 && nRlt==0)
      {
          return "Error,;";
      }
}
return "poweron,OK,;";
}
```

## 2.2 hm_poweroff

**Function Description:** Disable the torque of the robot.

**Format:** int hm_poweroff(int nRbtID);

**Number of Parameters:** 1

nRbtID：the ID of the robot，starting from 0。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
hm_poweroff(0);
```

**Full example:**

```
char* poweroff()
{
int nRlt = 0;
int nPowerOffState = 0;
int nErrorState=0;
nRlt =    hm_poweroff(0);
if(0 != nRlt)
    return "poweroff,Fail,;";
while(1)
{
```

```
    nRlt=hm_get_state(0,2,&nPowerOffState);
    if(nPowerOffState==0 && nRlt==0)
      {
          break;
      }
    nRlt=hm_get_state(0,3,&nErrorState);
    if(nErrorState==1 && nRlt==0)
      {
          return "Error,;";
      }
}
return "poweroff,OK,;";
}
```

## 2.3 hm_stop

**Function Description:** Stop the robot

**Format:** int hm_stop(int nRbtID);

**Number of Parameters:** 1

nRbtID：the ID of the robot，starting from 0。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
hm_stop(0);
```

**Full example:**

```
char* stop()
{
int nRlt = 0;
int nStopState = 0;
int nErrorState=0;
nRlt = hm_stop(0);
if(0 != nRlt)
    return "stop,Fail,;";
while(1)
{
```

```
    nRlt=hm_get_state(0,0,&nStopState);
    if(nStopState==0 && nRlt==0)
      {
          break;
      }
    nRlt=hm_get_state(0,3,&nErrorState);
    if(nErrorState==1 && nRlt==0)
      {
          return "Error,;";
      }
}
return "stop,OK,;";
}
```

## 2.4 hm_reset

**Function Description:** Reset the robot

**Format:** int hm_reset(int nRbtID);

**Number of Parameters:** 1

nRbtID：the ID of the robot，starting from 0。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
hm_reset(0);
```

**Full example:**

```
char* reset()
{
int nRlt = 0;
nRlt = hm_reset(0);
if(0 != nRlt)
    return "reset,Fail,;";
return "reset,OK,;";
}
```

## 2.5 hm_movej

**Function Description:** Set the angle of joints of the robot

**Format:** int hm_movej (

int nRbtID,

const double* pCoord

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pCoord：An Array with a length of 6, storing the angle of each joint.

pCoord[0]~pCoord[5]stand for：J1,J2,J3,J4,J5,J6; Unit: degree。

In case the robot has less than 6 DOFs, please set the redundant members in the array to 0.

**Return Value:** 0 for success, others are error codes for different errors.

**Example 1：**

// define the goal robot position, make the robot move to the first position (dPos1) and then to the second (dPos2).

```
char* movej()
{
double dPos1[6] = {20.0,0.0,0.0,0.0,0.0,0.0};
double dPos2[6]= {20.0,10.0,0.0,0.0,0.0,0.0};
int nMoveState = 0,nRlt = 0;
nRlt = hm_movej(0, dPos1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
   //Moving Done
}
else
{
 return "error";
}
```

```
nRlt = hm_movej(0, dPos2);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
   //Moving Done
}
else
{
    return "error";
 }
return "movej,OK,;";
}
```

**Example 2:**

```
// 使用系统点位，以”_P#”开头，该点位在示教器系统点位中示教完成
// Use the system position, starting with “_P#”, which is already demonstrated in the demonstrator.
// 先移动到_P0 点，再移动到_P1 点
// first move to Position P0, then to P1.
char* movej()
{
int nMoveState = 0,nRlt = 0;
nRlt = hm_movej(0, _P0);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
   //Moving Done Moving Done
}
else
{
 return "error";
```

```
}
nRlt = hm_movej(0, _P1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //Moving Done
}
else
{
    return "error";
 }
return "movej,OK,;";
}
```

## 2.6 hm_movel

**Function Description:** Move the robot End-Effector Linearly

**Format:** int hm_movel(

int nRbtID,

const double* pCoord

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pCoord： An Array with a length of 6, storing the Spatial Position of the End effector.

pCoord[0]~pCoord[5] stand for：X,Y,Z,A,B,C。X、Y、Z(mm)，A、B、C(degree)

**Return Value:** 0 for success, others are error codes for different errors.

**Example 1:**

// Use the user defined position, starting with "_P#", which is already demonstrated in the demonstrator.

```
char* movel()
{
```

```
double dPos1[6] = {420.0,52.0,442.0,157.0,51.0,153.0};
double dPos2[6]= {420.0,52.0,487.0,157.0,51.0,153.0};
int nMoveState = 0,nRlt = 0;
nRlt = hm_movel(0, dPos1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //Moving Done
}
else
{
  return "error";
}
nRlt = hm_movel(0, dPos2);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //Moving Done
}
else
{
    return "error";
}

return "movel,OK,;";
}
```

**Example 2:**

// Use the system position, starting with "_P#", which is already demonstrated in the demonstrator.

// first move to Position P0, then to P1.

```
char* movel()
{
int nMoveState = 0,nRlt = 0;
nRlt = hm_movel(0, _P0);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //Moving Done
}
else
{
  return "error";
}
nRlt = hm_movel(0, _P1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //Moving Done
}
else
{
    return "error";
}
return "movel,OK,;";
}
```

## 2.7 hm_movep

**Function Description:** Move the end effector to a specific point

**Format:** int hm_movep(

```
int nRbtID,
const double* pCoord
);
```

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pCoord： An Array with a length of 6, storing the Spatial Position of the End effector.

pCoord[0]~pCoord[5] stand for：X,Y,Z,A,B,C。X、Y、Z(mm)，A、B、C(degree)

**Return Value:** 0 for success, others are error codes for different errors.

**Example 1:**

```
// Use the user defined position, first move to Position P0, then to P1
char* movep()
{
double dPos1[6] = {420.0,52.0,442.0,157.0,51.0,153.0};
double dPos2[6]= {420.0,52.0,487.0,157.0,51.0,153.0};
int nMoveState = 0,nRlt = 0;
nRlt = hm_movep(0, dPos1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //Moving Done
}
else
{
  return "error";
}
nRlt = hm_movep(0, dPos2);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
```

```
    //Moving Done
}
else
{
    return "error";
}
return "movep,OK,;";
}
```

**Example 2:**

// Use the system position, starting with “_P#”, which is already demonstrated in the demonstrator.

```
char* movep()
{
int nMoveState = 0,nRlt = 0;
nRlt = hm_movep(0, _P0);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
   //Moving Done
}
else
{
   return "error";
}
nRlt = hm_movep(0, _P1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
   //Moving Done
}
```

```
else
{
    return "error";
}
return "movep,OK,;";
}
```

## 2.8 hm_moverell

**Function Description:** Linear movement of a single joint

**Format:** int hm_moverell(

int nRbtID,

int nAxisID,

int nDirection,

double dDistance

);

**Number of Parameters:** 4

nRbtID：ID of the robot, starting from 0;

nAxisID：destination，0~5： X,Y,Z(mm),A,B,C(degree)。

nDirection：direction，0=Negative；1=Positive。

dDistance：Relative moving distance，X、Y、Z(mm)，A、B、C(degree)。

。PS：the type of the parameters is double。

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// move the robot alone the X axis 10mm positively
char* movell()
{
int nMoveState = 0,nRlt = 0;
nRlt = hm_moverell(0,0,1,10.0);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
  {
```

```
    //Moving Done
  }
else
{
   return "error";
}
return "movell,OK,;";
}
```

## 2.9 hm_movewaitdone

**Function Description:** wait for the robot to finish moving.

**Format:** int hm_movewaitdone (int nRbtID,int* pMoveState);

**Number of Parameters:** 2

nRbtID：ID of the robot, starting from 0

pMoveState：0 =Moving Done；others =Error Occured during Moving

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// Use the system position, first move to Position P0, then to P1.
char* move()
{
double dPos1[6] = {30.0,30.0,30.0,30.0,30.0,30.0};
double dPos2[6]= {-30.0,-30.0,-30.0,-30.0,-30.0,-30.0};
int nMoveState = 0,nRlt = 0;
nRlt =hm_movej(0,dPos1);
if(0 != nRlt)
     return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
// dPos1Moving Done
}
else
{
```

```
    return "error";// Error Occured during Moving
}
nRlt =hm_movej(0,dPos2);
if(0 != nRlt)
    return "move,1,0,;";
nRlt =hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
// dPos2Moving Done
}
else
{
   return "error";// Error Occured during Moving
}
return "move,OK,;";
}
```

## 2.10 hm_set_override

**Function Description:** Set the Speed of the Robot

**Format:** int hm_set_override(int nRbtID,double dOverride);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

dOverride：Speed；range 0.01~1。

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
char* SetOverride()
{
int nRlt = 0;
double dOverride = 0.21;
nRlt =hm_set_override(0,dOverride);
if(0 != nRlt)
{
    return " SetOverride ,Fail,;";
```

```
}
return " SetOverride,OK,;";
}
```

## 2.11 hm_set_digital_out

**Function Description:** Set the IO output Status

**Format:** int hm_set_digital_out(int nRbtID,int nIndex,int nState);

**Number of Parameters: 3**

nRbtID：the ID of the robot，starting from 0。

nIndex：which IO you want to set。

nState：IO status。

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// Set the first IO to be high
char* SetDigitalOut()
{
int nRlt = 0;
nRlt =hm_set_digital_out(0,1,1);
if(0 != nRlt)
{
    return " SetDigitalOut,Fail,;";
}
return " SetDigitalOut,OK,;";
}
```

## 2.12 hm_set_serial_do

**Function Description:** Set the Serial Output Status

**Format:** int hm_set_serial_do(

int nBit,

int nState);

**Number of Parameters:** 2

nBit：the IO you want to set

nState：IOstatus。

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// Set the first serial IO to be high
char* SetSerialDo()
{
int nRlt = 0;
nRlt =hm_set_serial_do(1,1);
if(0 != nRlt)
{
    return " SetSerialDo,Fail,;";
}
return " SetSerialDo,OK,;";
}
```

## 2.13 hm_set_kinematic_coordinate

Function Description: Configure the coordinate system of the end-effector.

Format: int hm_set_kinematic_coordinate (

int nRbtID,

const double* pCoord

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pCoord：An Array with a length of 6，storing the 。

pCoord[0]~pCoord[5]stand for：X,Y,Z,A,B,C。

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
char* SetKinematicCoordinate()
{
int nRlt = 0;
double dKineCoord[6] = {38.0,-33.0,75.0,0.0,0.0,0.0};
nRlt =hm_set_kinematic_coordinate(0,&dKineCoord);
if(0 != nRlt)
```

```
{
    return " SetKinematicCoordinate,Fail,;";
}
return " SetKinematicCoordinate,OK,;";
}
```

## 2.14 hm_set_user_coordinate

**Function Description:** Configure the user's coordinate system with a specific coordination.

**Format:** int hm_set_user_coordinate(

int nRbtID,

const double* pCoord

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pCoord：An Array with a length of 6，storing the user defined coordinate。

pCoord[0]~pCoord[5]stand for：X,Y,Z,A,B,C。

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
char* SetUserCoordinate()
{
int nRlt = 0;
double dUserCoord[6] = {300.0,-500.0,200.0,150.0,40.0,10.0};
nRlt=hm_set_user_coordinate(0,dUserCoord);
if(0 != nRlt)
{
    return " SetUserCoordinate,Fail,;";
}
return " SetUserCoordinate,OK,;";
}
```

## 2.15 hm_set_conveyor_scale

**Function Description:** Set the scale ratio of the conveyor

**Format:** hm_set_conveyor_scale(
int nRbtID,
double dScale);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

dScale：the ratio of the conveyor, range from 0 to 1

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// set the conveyor scale ratio to be 0.5
char* SetConveyorScale()
{
int nRlt = 0;
nRlt=hm_set_conveyor_scale(0,0.5);
if(0 != nRlt)
{
    return " SetConveyorScale,Fail,;";
}
return " SetConveyorScale,OK,;";
}
```

## 2.16 hm_set_tracking_switch

**Function Description:** Enable / Disable trackign

**Format:** int hm_set_tracking_switch(int nRbtID,int nFlag);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

nFlag：Switch of the function, set 1 to enable, 0 to disable.

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// enable tracking
char* SetTrackingSwitch()
```

```
{
int nRlt = 0;
nRlt=hm_set_tracking_switch(0,1);
if(0 != nRlt)
{
    return " SetTrackingSwitch,Fail,;";
}
return " SetTrackingSwitch,OK,;";
}
// disable tracking
char* SetTrackingSwitch()
{
int nRlt = 0;
nRlt=hm_set_tracking_switch(0,0);
if(0 != nRlt)
{
    return " SetTrackingSwitch,Fail,;";
}
return " SetTrackingSwitch,OK,;";
}
```

## 2.17 hm_set_payload

**Function Description:** Set Payload Parameters

**Format:** int hm_set_payload(int nRbtID,double mass,double CentersX,double CentersY,double CentersZ);

**Number of Parameters:** 5

nRbtID：the ID of the robot，starting from 0。

mass：mass of the payload, unit: kg

CentersX: X coordinate of the physical center of the payload

CentersY: Y coordinate of the physical center of the payload

CentersZ: Z coordinate of the physical center of the payload

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// set the payload parameters，mss：3KG，coordinate：0,0,0
char* SetPayload()
{
int nRlt = 0;
nRlt=hm_set_payload(0,3.0,0,0,0);
if(0 != nRlt)
{
    return " SetPayload,Fail,;";
}
return " SetPayload,OK,;";
}
```

## 2.18 hm_set_safestop

**Function Description:** Set the threshold torque for safeStop function

**Format:** int hm_set_safeStop(int nRbtID,double J1,double J2,double J3,double J4,double J5,double J6);

**Number of Parameters:** 7

nRbtID：the ID of the robot，starting from 0。

J1：threshold torque of Joint 1, unit: Nm

J2：threshold torque of Joint 2, unit: Nm

J3：threshold torque of Joint 3, unit: Nm

J4：threshold torque of Joint 4, unit: Nm

J5：threshold torque of Joint 5, unit: Nm

J6：threshold torque of Joint 6, unit: Nm

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// Set the threshold torque for safeStop function
char* SetSafestop()
{
int nRlt = 0;
nRlt=hm_set_safeStop(0,50.0,50.0,50.0,50.0,50.0,50.0);
if(0 != nRlt)
```

```
{
    return " SetSafestop,Fail,;";
}
return " SetSafestop,OK,;";
}
```

## 2.19 hm_set_zeroforce

**Function Description:** Set the threshold torque for zeroforce teaching

**Format:** int hm_set_zeroforce(int nRbtID,double J1,double J2,double J3,double J4,double J5,double J6);

**Number of Parameters:** 7

nRbtID：the ID of the robot，starting from 0。

J1：threshold torque of Joint 1, unit: Nm

J2：threshold torque of Joint 2, unit: Nm

J3：threshold torque of Joint 3, unit: Nm

J4：threshold torque of Joint 4, unit: Nm

J5：threshold torque of Joint 5, unit: Nm

J6：threshold torque of Joint 6, unit: Nm

**Return Value:** 0 for success, others are error codes for different errors.

**For Example:**

```
// Set the threshold torque for zeroforce teaching
char* Zeroforce()
{
int nRlt = 0;
nRlt=hm_set_zeroforce(0,100.0,100.0,100.0,100.0,100.0,100.0);
if(0 != nRlt)
{
    return " Zeroforce,Fail,;";
}
return " Zeroforce,OK,;";
}
```

## 2.20 hm_get_override

**Function Description:** Get the speed Ratio of the robot

**Format:** int hm_get_override(

int nRbtID,

double* pOverride);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pOverride：Speed of the robot

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the speed Ratio of the current robot
double dOverride = 0.0;
hm_get_override(0,&dOverride);
```

**Full example:**

```
// Get the speed Ratio of the current robot
char g_chBuff[120] = {0};
char* GetOverrid ()
{
int nRlt = 0;
double dOverride = 0.0;
nRlt=hm_get_override(0,&dOverride);
if(0 != nRlt)
{
    return " GetOverrid,Fail,;";
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"override=%0.3f",dOverride);
return g_chBuff;
}
```

The result shown in log:

override=0.250（PS: the data represents the speed ratio）

## 2.21 hm_get_acs_pos

**Function Description:** Get the ACS position of the robot

**Format:** int hm_get_acs_pos (

int nRbtID,

double *pAcsPos

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pAcsPos：An Array with a length of 6，storing the ACS position of the robot。

pAcsPos[0]~ pAcsPos[5]stand for：J1,J2,J3,J4,J5,J6, unit: degree

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the ACS position of the current robot
double dAcsPos[6] ={0};
hm_get_acs_pos(0,dAcsPos);
```

**Full example:**

```
// Get the ACS position of the current robot
char g_chBuff[120] = {0};
char* GetAcsPos()
{
int nRlt = 0;
double dAcsPos[6] ={0};
nRlt=hm_get_acs_pos(0,dAcsPos);
if(0 != nRlt)
{
    return " GetAcsPos,Fail,;";
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"AcsPos=%0.3f,%0.3f,%0.3f,%0.3f,%0.3f,%0.3f",dAcsPos[0],
dAcsPos[1],dAcsPos[2],dAcsPos[3],dAcsPos[4],dAcsPos[5]);
return g_chBuff;
}
```

The result shown in log:

AcsPos=13.242,-66.944,-33.579,4.547,0.000,0.000（PS：the data represents the ACS position of the current robot, standing for the position of J1 ~ J6）

## 2.22 hm_get_pcs_pos

**Function Description:** Get the PCS Position of the robot

**Format:** int hm_get_pcs_pos (

int nRbtID,

double *pPcsPos

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pPcsPos：An Array with a length of 6，storing the PCS position of the robot。

pPcsPos[0]~ pPcsPos[5]stand for：X,Y,Z (mm) A,B,C (degree)；

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the PCS Position of the current robot
double dPcsPos[6] ={0};
hm_get_pcs_pos(0,dPcsPos);
```

**Full example:**

```
// Get the PCS Position of the current robot
char g_chBuff[120] = {0};
char* GetPcsPos()
{
int nRlt = 0;
double dPcsPos[6] ={0};
nRlt=hm_get_pcs_pos(0,dPcsPos);
if(0 != nRlt)
{
    return " GetPcsPos,Fail,;";
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"PcsPos=%0.3f,%0.3f,%0.3f,%0.3f,%0.3f,%0.3f",dPcsPos[0],dPcsPos[1],dPcsPos[2],dPcsPos[3],dPcsPos[4],dPcsPos[5]);
```

```
return g_chBuff;
}
```

the result shown in the log：

PcsPos=467.866,-172.170,59.674,-49.155,0.000,0.000

（PS：The data represents the ACS position, standing for XYZ,ABC）

## 2.23 hm_get_digital_out

**Function Description:** Get the Digital output status。

**Format:** int hm_get_digital_out (

```
int nRbtID,
int nIOIndex,
int *pnState
);
```

**Number of Parameters:** 3

nRbtID：the ID of the robot，starting from 0。

nIOIndex：the index of IO。

pnState：status of IO。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the status of digital output 1
int nState = 0;
hm_get_digital_out(0,1,&nState);
```

**Full example:**

```
// Get the status of digital output 1
char g_chBuff[120] = {0};
char* GetDigitalOut()
{
int nRlt = 0;
int nState = 0;
nRlt=hm_get_digital_out(0,1,&nState);
if(0 != nRlt)
{
    return " GetDigitalOut,Fail,;";
```

```
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"State=%d",nState);
return g_chBuff;
}
```

the result shown in the log：

State=0 （PS：the data represents the state of Digital Output）

## 2.24 hm_get_digital_in

**Function Description:** Get the status of Digital Input

**Format:** int hm_get_digital_in (

int nRbtID,

int nIOIndex,

int *pnState

);

**Number of Parameters:** 3

nRbtID：the ID of the robot，starting from 0。

nIOIndex：the index of IO。

pnState：status of IO。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the status of Digital Input 1
int nState = 0;
hm_get_digital_in(0,1,&nState);
```

**Full example:**

```
// Get the status of Digital Input 1
char g_chBuff[120] = {0};
char* GetDigitalIn ()
{
int nRlt = 0;
int nState = 0;
nRlt=hm_get_digital_in(0,1,&nState);
if(0 != nRlt)
```

```
{
    return " GetDigitalOut,Fail,;";
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"State=%d",nState);
return g_chBuff;
}
```

the result shown in the log：

State=0 （注：数据为数字输出 1 的状态）

## 2.25 hm_get_serial_di

**Function Description:** Get the Input status of the specific Serial Port

**Format:** int hm_get_serial_di(

int nBit, int* pState);

**Number of Parameters:** 2

nBit：IO bits need to be read。

pState：status of IO。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the Input status of the serial port (IO=0)
int pState = 0;
hm_get_serial_di(0,&pState);
```

**Full example:**

```
// Print the Input status of the serial port (IO=0)
char g_chBuff[120] = {0};
char* GetSerialDi()
{
int nRlt = 0;
int pState = 0;
nRlt=hm_get_serial_di(0,&pState);
if(0 != nRlt)
{
    return " GetSerialDi,Fail,;";
```

```
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"State=%d",pState);
return g_chBuff;
}
```

the result shown in the log：

State=0 （PS：the data represents the Input status of the serial port (IO=0)）

## 2.26 hm_get_serial_do

**Function Description:** Get the Output status of the specific Serial Port

**Format:** int hm_get_serial_do(

int nBit, int* pState);

**Number of Parameters:** 2

nBit：IO bits need to be read.

pState：status of IO.

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the Output status of the serial port (IO=0)
int pState = 0;
hm_get_serial_do(0,&pState);
```

**整体 For Example:**

```
// Print the Output status of the serial port (IO=0)
char g_chBuff[120] = {0};
char* GetSerialDo()
{
int nRlt = 0;
int pState = 0;
nRlt=hm_get_serial_do(0,&pState);
if(0 != nRlt)
{
    return " GetSerialDo,Fail,;";
}
memset(g_chBuff,0,sizeof(g_chBuff));
```

```
sprintf(g_chBuff,"State=%d",pState);
return g_chBuff;
}
```

the result shown in the log：

State=0 （PS：the data represents the Output status of the serial port (IO=0)）

## 2.27 hm_get_analog_input

**Function Description:** Get the Value of Analog Input

**Format:** int hm_get_analog_input(

int nRbtID,

int nAnalogIndex,

int* pnState);

**Number of Parameters:** 3

nRbtID：the ID of the robot，starting from 0

nAnalogIndex：Index of the Analog input

pnState：the value of the analog input.

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the Value of Analog Input 1
int pnState= 0;
hm_get_analog_input（0,1,&pnState）
```

**Full example:**

```
// Get the Value of Analog Input 1
char g_chBuff[120] = {0};
char* GetAnalogInput()
{
int nRlt = 0;
int pnState= 0;
nRlt =hm_get_analog_input(0,1,&pnState);
if(0 != nRlt)
{
    return " GetAnalogInput,Fail,;";
}
```

```
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"State=%d",pnState);
return g_chBuff;
}
```

the result shown in the log：

State=0

（PS：the data represents the value of analog input 1）

## 2.28 hm_get_multiendat_pos

**Function Description:** Get the Absolute Position of All joints (Encoder Angle)

**Format:** int hm_get_multiendat_pos (

int nRbtID,

int *pEndatPos

);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pEndatPos：An Array with a length of 6，storing the encoder value of J1 ~ J6。

pEndatPos[0]~ pEndatPos[5]stand for：encoder value (multi-turn calculated) of J1 ~ J6 (counts)；

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
//Get the encoder output of the robot
int nEndatPos[6] ={0};
hm_get_multiendat_pos(0,nEndatPos);
```

**Full example:**

```
// Get the encoder output of the robot
char g_chBuff[120] = {0};
char* GetMultiendatPos()
{
int nRlt = 0;
int nEndatPos[6] ={0};
nRlt=hm_get_multiendat_pos(0,nEndatPos);
if(0 != nRlt)
```

```
{
    return " GetMultiendatPos,Fail,;";
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"EndatPos=%d,%d,%d,%d,%d,%df",nEndatPos[0],nEndatPos[
1],nEndatPos[2],nEndatPos[3],nEndatPos[4],nEndatPos[5]);
return g_chBuff;
}
```

the result shown in the log：

EndatPos=-8961.000,2391007.000,16206.000,197833.000,0.000,0.000

（PS：the data represents the encoder output of the robot，from J1 to J6）

## 2.29 hm_get_conveyor_value

**Function Description:** Get the encoder value of the conveyor

**Format:** int hm_get_conveyor_value (

int nRbtID,

double* pValue);

**Number of Parameters:** 2

nRbtID：the ID of the robot，starting from 0。

pValue：encoder value of the conveyor。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the encoder value of the conveyor
double pValue ={0};
hm_get_conveyor_value (0,&pValue);
```

**For Example:**

```
// Get the encoder value of the conveyor
char g_chBuff[120] = {0};
char* GetConveyorValue()
{
int nRlt = 0;
double pValue =0;
nRlt=hm_get_conveyor_value (0,&pValue);
```

```
if(0 != nRlt)
{
    return "error";
}
memset(g_chBuff,0,sizeof(g_chBuff));
sprintf(g_chBuff,"Value=%0.3f",pValue);
return g_chBuff;
}
```

the result shown in the log：

Value=0.000

（PS：the data represents the encoder value of the conveyor）

## 2.30 hm_get_state

**Function Description:** Get the status of the robot

**Format:** int hm_get_state (

int nRbtID,

int nStateType,

int *pState

);

**Number of Parameters:** 3

nRbtID：the ID of the robot，starting from 0。

nStateType：index of the status of the robot。

pState：status of the robot。

**Return Value:** 0 for success, others are error codes for different errors.

**Independent Example**

```
// Get the status of Moving
int nMovingState = 0;
hm_get_state(0,0,&nMovingState);
// see if the robot is enabled
int nPowerOnState = 0;
hm_get_state(0,2,&nPowerOnState);
```

**For Example:**

```
// see if the robot is enabled
```

```
char* GetState()
{
int nRlt = 0;
int nPowerOnState = 0;
nRlt=hm_get_state(0,2,&nPowerOnState);
if(nPowerOnState==1 && nRlt==0)
{
  return "GetState,OK,;";//the status of Enable/Disable state
}
else
{
return "GetState,Fail,;";
}
}
```

| Index of the status | explanation |
|---|---|
| 0 | Moving Status，Return: 1: moving；0: not moving |
| 1 | Standby Status（Internal use only） |
| 2 | Enable Switch，Return: 1:Enabled；0:Disabled |
| 3 | Error Status，Return: 1Error Occured；0: No Error Occured |
| 4 | Reserved |
| 5 | Reserved |
| 6 | Reserved |

# 3 System Reserved Task Functions

为了让用户可以在脚本中实现某些监控类工作的功能，目前大族机器人控制器系统提供了三个保留的系统任务函数，大族机器人控制系统会周期性的调用这三个系统任务函数。

In order to make it possible for users to implement functions like monitoring, Han's Robot control system provides 3 system functions to the users, which the control system will call periodically.

用户可以在脚本中重实现此三个系统函数。借此让机器人控制系统不断的进行某些工作，比如监控 IO，初始化系统等。

Users can re-implement the three system function, making the system keep monitoring some specific parts such as IO, booting system .etc.

注意：不要在这三个系统任务函数中出现类似 while(1)这样的死循环代码，大族机器人控制系统会每周期去调用这些系统任务函数。

PS: Do not add endless loops into these functions, the system will call the functions periodically.

## 3.1 _SysTask_1

Format:

char* _SysTask_1();

## 3.2 _SysTask_2

Format:

char* _SysTask_2();

## 3.3 _SysTask_3

Format:

char* _SysTask_3();

**For Example:**

```
char* movej()
{
double dPos1[6] = {20.0,0.0,0.0,0.0,0.0,0.0};
int nMoveState = 0,nRlt = 0;
nRlt = hm_movej(0, dPos1);
if(0 != nRlt)
    return "move,1,0,;";
nRlt = hm_movewaitdone(0,&nMoveState);
if((0 == nRlt) && (0 == nMoveState))
{
  //运动完成
}
else
{
 return "error";
}
return "movej,OK,;";
}
char* _SysTask_1()
{
int nRlt = 0;
int pState = 0;
nRlt=hm_get_serial_di(0,&pState);
if(0 != nRlt)
{
    return " GetSerialDi,Fail,;";
}
if(pState==1)
{
 movej();
}
return "OK,;";
}
```

# 4 Error Codes

| Error Code | Explanation |
|---|---|
| 0 | 命令执行成功 Instruction Executed Successfully |
| 1001 | 机器人未初始化 the robot is not initiated |
| 1002 | 主站未启动 the master is not powered on |
| 1003 | 从站掉线 the Slave is offline |
| 1004 | 安全停车 Safe-stop Triggered |
| 1005 | 物理急停 Emergency Stopped |
| 1006 | 机器人未上电 the robot is not powered on |
| 1007 | 从站报错 Error occurred in the slave |
| 1008 | 机器人超出安全空间 The robot is beyond the safe space |
| 1009 | 机器人运动中 the robot is moving |
| 1010 | 命令无效 Unidentified Instruction |
| 1011 | 参数错误 Parameter Error |
| 1012 | 格式错误 Format Error |
| 1013 | 等待命令执行 Waiting for the execution of the instruction |
| 1014 | IO 不存在 There is no such IO |
| 1015 | 机器人不存在 There is no such Robot |
| 1016 | SocketInValid |
| 1017 | 网络超时 Network Timeout |
| 1018 | 连接失败 Connect Failed |
| 1019 | 串口连接失败 Serial Port Connection Failed |
| 1020 | 未设置零点位置 Zero Position is not set yet |
| 1021 | 上一个同样的命令未完成 The last same instruction is not finished yet |
| 1022 | 串口 DI 为空 Data In buffer of the serial port is empty |
| 1023 | 串口 DO 为空 Data Out buffer of the serial port is empty |
| 1024 | 等待超时 waiting timeout |
| 1025 | 错误状态 Error Status |
| 1026 | 机器人停止中 Robot stopping |

| | |
|---|---|
| 1027 | 机器人去使能中 Robot Disabling |
| 1028 | 机器人使能中 Robot Enabling |
| 1029 | 功能未启用 the function is not available |
| 1030 | 启动主站超时 timeout when initiating the master |
| 1031 | 机器人未上电 the robot is not powered on |
| 2000 | 加载库失败 failed loading the library |
| 2001 | 脚本为空 the script is empty |
| 2002 | 编译错误 Compiling error |
| 2003 | 重新加载脚本错误 Re-load the script error |
| 2004 | 函数不存在 the function does not exist |
| 2005 | 函数返回类型错误 Wrong Return value type of the function |
| 2006 | MissSignal1 |
| 2007 | MissSignal2 |
| 2008 | 参数类型错误 Wrong Parameter Type |
| 2009 | 没有包含头文件 Did not include header files |

# 5 Attentions

控制机器人运动，在机器人成功使能的情况下，可以通过发送 hm_move*指令的方式让机器人运动到目标位置。发送运动指令的过程中有以下几点需要注意：

1、用户调用 hm_move*系列的运动指令时，控制系统会等待 Moving Done 或报错才返回运动指令的调用。

2、连续运动时，用户需要判断上一次运动是否成功完成，如果上一次运动中出错，必须要调用 hm_reset 指令清错

To Control the robot move under the situation that the robot is successfully enabled, users can get the robot move to the target position by sending hm_move* instructions. Here are some facts needed to be noticed:

1、When users are calling hm_move* instructions, the system will not call the function until it gets Moving Done signal or Error codes.

2、When making the robot move continuously, the user must check if the last move is finished. If error occurred during the last move, the function hm_reset must be called to clear the error.