takamoto0206@yahoo.co.jp

NXC API ライブラリ関数一覧

ここでは、NXC の API ライブラリ関数一覧を機能ごとに分けて説明しましょう。

＜本資料は、John Hansen 氏による「LEGOMINDSTORMS NXT POWERPROGRAMMING」の付録 A の NXC API 関数（クイックリファレンス）をまとめたものです。メールにてご指摘・修正を頂ければ、修正してまいります。＞

2012.01.17 版

«作成：高本孝頼»
Takamoto0206@yahoo.co.jp

◆１）タスクおよびタイミング関数

関数名	機能/書式・型・パラメータ
Wait	**ミリ秒単位(1 秒=1000 ミリ秒)での待機時間の設定** void　Wait(ms)：戻り値なし 　unsigned long ms：待機する 1 ミリ秒単位の数値
CurrentTick	**ミリ秒単位で返す現時点のシステム時計（時刻）** unsigned long CurrentTick(): ミリ単位時刻
FirstTick	**プログラム開始からのミリ単位で返すシステム時計（時刻）** unsigned long FirstTick(): ミリ単位時刻
SleepTime	**自動シャットダウン（スリープ）する時間** byte SleepTime()： 設定されているスリープ時間（秒）
SleepTimer	**自動シャットダウンするまでの時間（タイマー）** byte SleepTimer()：スリープするまでの残り時間（秒）
ResetSleepTimer	**タイマーを自動シャットダウンする時間にリセット** byte SleepTimer(): 設定されているスリープ時間（秒）
SetSleepTime	**自動シャットダウンする時間の設定** void SetSleepTime(cons byte n):戻り値なし 　cons byte n: シャットダウンまでの時間を設定
SetSleepTimer	**自動シャットダウンするタイマー設定** void SetSleepTimer(cons byte n):戻り値なし 　cons byte n: シャットダウン時間（タイマー）設定
Stop	**実行中のプログラムの停止** void Stop(bvalue)：戻り値なし 　bool bvalue: ブーリアン値で真のときプログラム停止
StopAllTask	**現在実行中の全てのタスクを停止** void StopAllTasks()：戻り値なし
StartTask	**タスクのスタート** void StartTask(tsk)：戻り値なし 　task tsk: スタートさせるタスク名
StopTask	**タスクのストップ** void stopTask(tsk)：戻り値なし task tsk: ストップさせるタスク名
Acquire	**mutex の取得** void Acquire(m): 戻り値なし 　mutex m：取得する mutex
Release	**mutex の解放** void Realse(m):戻り値なし 　mutex m：　解放する muex
Precedes	**同時に実行するタスク群の設定** void Precedes(task t1, task t2,　･･･, task tn)：戻り値なし 　task tx: 実行するタスク名
Follows	**つぎに実行するタスク群の設定** void Follows(tsk1, tsk2,　･･･, tskn)：戻り値なし 　task tskx: 実行するタスク名
ExitTo	**今のタスクを即終了し、つぎに実行するタスクの設定** void ExitTo(tsk): 戻り値なし 　task tsk：つぎに実行するタスク名

◆２）文字列関数

関数名	機能/書式・型・パラメータ
StrToNum	**文字列を数値に変換** variant StrToNum(str): 数値 string str: 数値を表す文字列（ “125” , “32.34” など）
StrLen	**文字列の長さを返す** int strlen(str) : 文字列の長さ string str : 文字列
StrIndex	**文字列の中の文字コード（Ascii コード）を出力** byte StrIndex(str, idx) : Ascii コードを返す string str: 対象となる文字列 unsigned int idx : 相対文字位置（先頭文字は 0 から）
NumToStr	**数値を文字列に変換** string NumToStr(num): 文字列を返す（ ”72“ など） variant num : 数値
FormatNum	**文字をフォーマットに沿って文字列に変換** string FormatNum(fmt, num) : 変換された文字列 string fmt : 文字列フォーマット “%7.4f RAD”、”%x“、“%04x”、 ”%5d”など variant num: 数字
StrCat	**文字列の結合** string StrCat(str1, str2, ･･･, strn) : 結合された文字列 string strx : 文字列
SubStr	**文字列の切り出し【** SubStr(“abcdefg”,2,3)-> “cde” **】** string SubStr(str, idx, len) : 文字列の位置 idx から長さ len を取り出す string str: 文字列 unsigned int idx: 文字列の位置（先頭文字の位置は 0） unsigned int len : 切り出す文字列の長さ
StrReplace	**文字列の置換【** StrReplace(“abcdefg”,2,”xyz”) -> “abxyzfg” **】** string StrReplace(str,idx,stnew): 置換された文字列 string str: 元の文字列 unsigned int idx: 文字列の位置インデックス（先頭文字の位置は 0 ） string strnew: 置換する文字列
Flatten	**Ascii コードを文字列変換 【** Flatten(49) -> “0” **】** string Flatten(num) : 文字列を返す variant num: 数値
FlattenVar	**ほとんどのデータ群を文字列変換** string FalttenVar(num) :変換された文字列を返す variant num:配列データ（数値）も変換（ただし展開は UnflattenVar 利用）
UnflattenVar	**文字列をデータ群に変換** int UnflatenVar(str, x) :変換成功（ture）/ 失敗（false） string str: 文字列群 variant & x: 変換されデータ群（数値）
ByteArrayToStr	**バイト配列を文字列に変換** （逆の関数は StrToByteArrey） string ByteArrayToStr(data): 変換された文字列を返す byte data[]:バイトの配列値
ByteArrayToStrEx	**バイト配列を文字列に変換** （逆の関数は、StrToByteArray） void ByteArrayToStrEx(data, str):戻り値なし byte data[]:バイト配列値 string & str:変換された文字列

◆３）配列関数

関数名	機能/書式・型・パラメータ
StrToByteArray	**文字列をバイト配列に変換** （逆の関数は、ByteArrayToStr） void StrToByteArray(str,data):戻り値なし string str:　入力の文字列 byte & data[]: 変換されたバイト配列
ArrayLen	**配列の長さを算出** unsigned int ArrayLen(data) :配列の長さを算出して返す variant data[] : 長さを読み取りたい配列
ArrayInit	**配列の初期化** void ArrayInit(aout,value, count): 戻り値なし variant & aout[] : 初期化する対象の配列 variant value: 初期値 unsigned int count: 配列のうちの作成要素数
ArrayMax	**数値配列の最大値検出** variant ArrayMax(src,idx,len): 数値配列の最大値を算出して返す const variant & src[]:最大値を探しだす対象となる配列 unsigned int idx: 探しだす配列の最初のインデックス（省略値は NA） unsigned int len: 探しだす配列の長さ（省略値は NA）
ArrayMean	**数値配列の平均値値算出** variant ArrayMean(src,idx,len): 数値配列の平均値を算出して返す const variant & src[]:対象となる配列 unsigned int idx: 配列の最初のインデックス（省略値は NA） unsigned int len: 配列の長さ（省略値は NA）
ArrayMin	**数値配列の最小値検出** variant ArrayMin(src,idx,len): 数値配列の最大値を算出して返す const variant & src[]:最少値を探しだす対象となる配列 unsigned int idx: 探しだす配列の最初のインデックス（省略値は NA） unsigned int len: 探しだす配列の長さ（省略値は NA）
ArraySubset	**配列の複写** void ArraySubset(aout,asrc,idx,len): 戻り値なし variant & aout[] : 目的となる数値配列 variant asrc[]: ソースとなる数値配列 unsigned int idx: ソート開始のインデックス（省略値は NA） unsigned int len: ソートする配列数（省略値は NA）
ArraySort	**配列の数値的なソート（並び換え）** void ArraySort(dest,src,idx,len): 戻り値なし variant & dest[] : 目的となる数値配列 const variant & str[]: ソースとなる数値配列 unsigned int idx: ソート開始のインデックス（省略値は NA） unsigned int len: ソートする配列数 （省略値は NA）
ArrayBuild	**配列の値設定** void ArrayBuild(aout,src1,src2,･･･,srcN) : 戻り値なし variant & aout[] : 配列 variant srcx : 設定する値

※その他 標準偏差（ArrayStd）,総和（ArraySum）,平方総和（ArraySumSqr）なども装備。

◆４）数式関数

関数名	機能/書式・型・パラメータ
Random	**乱数発生** int Random(n)： 発生した乱数（n が null の場合：整数値範囲で乱数発生） unsigned int n： n-1 から 0 までの乱数発生
Sqrt	**平方根値**（aqrt に同じ） float Sqrt(x)： 平方根値を返す float x： 入力実数
sin	**sine 値** float sin(x)：sin 値を返す float x：入力実数
cos	**cosine 値** float cos(x)：cos 値を返す float x：入力実数
tan	**tangent 値** float tan(x)：tan 値を返す float x：入力実数
asin	**arc sine 値** float asin(x)：arcsin 値を返す float x：入力実数
acos	**arc cosine 値** float acos(x)：arccos 値を返す float x：入力実数
atan	**arc tangent 値** float atan(x)：arc tangent 値を返す float x：入力実数
atan2	**arc tantang 値** float atan2(x)：atan2 値を返す float x：入力実数
sinh	**sinh 値** float sinh(x)：sinh 値を返す float x：入力実数
cosh	**cosh 値** float cosh(x)：cosh 値を返す float x：入力実数
tanh	**tanh 値** float tanh(x)：tanh 値を返す float x：入力実数
ceil	**実数部を切り上げ実数化**【x=3.01 -> 4、 x=3.16 -> 4 など】 float ceil(x)： 切り上げた実数値を返す float x： 入力実数
exp	**e(2.7182818)を底として引数で乗する値** float exp(x)：e を底とし x を乗する値を返す float x： 入力実数
bcd2dec	**16 進数から 10 進数へ変換** byte bcd2dec(bcd) :16 進数を 10 進数に変換した数値を返す byte bcd： 16 進数（例えば 0x3a を 58 に変換）
floor	**引数 を超えない最大の整数を返す関数**【x=12.345 のとき 結果は 12.0】 float floor(x)： 整数化された実数値を返す float x： 入力実数

frac	**小数部の取り出し関数**【x=3.141592 のとき　結果は 0.141592】 float frac(x)：小数部のみの出力 float x：　入力実数
log	**自然対数を求める関数** float log(x)：実数 x の log 値を返す float x：入力実数
log10	**常用対数を求める関数** float log10(x)：実数 x の log 値を返す float x：入力実数
muldiv32	**a*b/c の値を返す関数** long muldiv32(a, b, c)：a*b/c の値を返す long a: 32 ビット整数 long b: 32 ビット整数 long c: 32 ビット整数
pow	**底 の乗数 乗の計算を行って返す関数**　【pow(5,2) -> 25】 float pow(base, exponent)：base の expornet 乗を返す float base：底となる実数 float exponent：乗となる実数
trunc	**実数の整数部を返す関数**【x=12.345 のとき 結果は 12】（floor と類似） long trunc (x) :整数化された結果（整数値）を返す float x：入力実数

◆５）入力関数

関数名	機能/書式・型・パラメータ
SetSensor	**センサーの設定** void SetSensor(port, config)：返り値なし const byte & port：入力ポート（S1,S2,S3,S4 のいずれか） const unsigned int config：センサー設定 SENSOR_TOUCH SENSOR_LIGHT SENSOR_ROTATION SENSOR_CELSIUS SENSOR_FAHRENHEIT SENSOR_PULSE SENSOR_EDGE SENSOR_NXTLIGHT SENSOR_SOUND SENSOR_LOWSPEED_9V SENSOR_LOWSPEED SENSOR_COLORFULL SENSOR_COLORRED SENSOR_COLORGREEN SENSOR_COLORBLUE SENSOR_COLORNONE
SetSensorType	**センサータイプ設定** byte SneosrType(port,type)：　戻り値なし const byte & port：入力ポート（S1,S2,S3,S4 のいずれか） byte type: センサータイプ SENSOR_TYPE_NONE　0 SENSOR_TYPE_TOUCH　1

	SENSOR_TYPE_TEMPERATURE 2 SENSOR_TYPE_LIGHT 3 SENSOR_TYPE_ROTATION 4 SENSOR_TYPE_LIGHT_ACTIVE 5 SENSOR_TYPE_LIGHT_INACTIVE 6 SENSOR_TYPE_SOUND_DB 7 SENSOR_TYPE_SOUND_DBA 8 SENSOR_TYPE_COSTOM 9 SENSOR_TYPE_LOWSPEED 10 SENSOR_TYPE_LOWSPEED_9V 11 SENSOR_TYPE_HIGHSPEED 12 SENSOR_TYPE_COLORFULL 13 SENSOR_TYPE_COLORRED 14 SENSOR_TYPE_COLORGREEN 15 SENSOR_TYPE_COLORBLUE 16 SENSOR_TYPE_COLORNONE 17
SetSensorMode	**センサーモード設定** byte SneosrMode(port,mode)：戻り値なし const byte & port：入力ポート（S1,S2,S3,S4 のいずれか） byte mode: センサーモード SENSOR_MODE_RAW 0x00 Raw 値【0-1023 値】 SENSOR_MODE_BOOL 0x20 Bool 値【0 or 1】 SENSOR_MODE_EDGE 0x40 SENSOR_MODE_PULSE 0x60 SENSOR_MODE_PERCENT 0x80 %値【0-100 値】 SENSOR_MODE_CELSIUS 0xa0 SENSOR_MODE_FAHRENHEIT 0xc0 SENSOR_MODE_ROTATION 0xe0
SetSensorLight	**光センサー設定** void SetSensorLight(port, bActive)： 戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）【標準 S3】 bool bActive: ture（ライト ON）、false（ライト OFF）【省略値 true】
SetSensorSound	**サウンド（音）センサー設定** void SetSensorSoud(port, bActive)：戻り値なし const byt & port: 入力ポート（S1,S2,S3,S4 のいずれか）【標準 S2】 bool bActive：ture(ｄB)、false(ｄBA)【省略値 true】
SetSensorTouch	**タッチセンサー設定** void SetSensorTouch(port) const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）【標準 S1】
SetSensorLowspeed	**I2C デジタルセンサー（超音波センサー）設定** void SetSensorLowSpeed(port, bIsPower)：戻り値なし const byte & port：入力ポート（S1,S2,S3,S4 のいずれか）【標準 S4】 bool bIsPower:【省略値 true】
SetSensorColorBlue	**青色 LED センサー設定** void SetSensorColorBlue(port)：戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SetSensorColorFull	**青色 LED センサー設定** void SetSensorColorBlue(port)：戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SetSensorColorGreen	**緑色 LED センサー設定** void SetSensorColorGreen(port)：戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）

SetSensorColorNon	**LED センサー設定** void SetSensorColorNon(port)：戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SetSensorColorRed	**赤色 LED センサー設定** void SetSensorColorRed(port)：戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SetInput	**入力値の設定** void Setinput(port,field,value)：戻り値なし const byte & port:入力ポート（S1,S2,S3,S4 のいずれか） cons int field:入力種別フィールドの設定 TypeField 0：タイプ（Read/Write） InputModeField 1：モード（Read/write） RawValueField 2：Raw 値（read）(0..1023) NormalizedValueField 3：一般値（read） ScaledValueField 4：スケール値（read/write）(0. .100) InvalidDataField 5：無効値 variant value: 入力値
ClearSensor	**センサー設定のクリア** void ClearSensor(port):戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
ResetSensor	**センサー値をリセット** void ResetSensor(port)：戻り値なし const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SetCustomSensorZeroOffset	**ゼロでオフセット設定** void SetCustomSensorZeroOffset(port, zeroOffset)：戻り値なし byte port: 入力ポート（S1,S2,S3,S4 のいずれか） int zeroOffset：ゼロオフセットする値（整数）
SetCustomSensorPercentFullScale	**パーセント表示設定** void SetCustomSensorPercentFullScale(port, pctFullScale):戻り値なし byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte pctFullScale:新たに 100%にする値
SetCustomSensorActiveStatus	**アクティブな状態に設定** void SetCustomSensorActiveStatus(port, activeStatus):戻り値なし byte prot: 入力ポート（S1,S2,S3,S4 のいずれか） byte activeStatus:新たな設定するアクティブ値
SetSensorDigiPinsDirection	**センサーの直接な値のデジタルピンを設定** void SetSensorDigiPinsDirection(pin,direction): 戻り値なし byte pin: 入力ポート（S1,S2,S3,S4 のいずれか） byte direction:新たなデジタルピンの直接値
SetSensorDigiPinsStatus	**デジタルピンの状態を設定** void SetSensorDigiPinsStatus(port, status): 戻り値なし byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte status:新たなデジタルピンの状態値
SetSensorDigiPinsOutputlevel	**出力レベル値のデジタルピンを設定** void SetSensorDigiPinsOutputlevel(port,outputLevel): 戻り値なし byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte outputLevel:新たなデジタルピンの出力レベル値
SensorValue	**センサースケール値を読込み(Sensor に同じ)** unsigned int SensorValue(port):センサーのスケール値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）

Sensor	**センサースケール値を読込み** unsigned int Sensor(port): センサースケール値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorScaled	**センサースケール値を読込み(Sensor に同じ)** unsigned int SensorScaled(port):センサーのスケール値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorUS	**超音波センサー値の読込み** byte SensorUS(prot):距離（0..255)を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorType	**センサータイプ読込み** byte SensorType(port):センタータイプを返す conts byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorMode	**センサーモード読込み** byte SensorMode(port):モード値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorRaw	**センサーの生データ値読込み** unsigned int SensorRaw(port):Raw 値(0..1023)を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorValueRaw	**センサーの生データ値読込み（SensorRaw に同じ）** unsigned int SensorValueRaw(port):Raw 値(0..1023)を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorNormalized	**センサーの標準データ値読込み** unsigned int SensorNormalized(port):標準値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorInvalid	**センサーの無効なデータ値の判定フラグ** bool SensorInvalid(port): 真（true）無効データ、偽（false）有効データ const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorBoolean	**センサーのブーリアン値読込み** bool SensorBoolean(port):ブーリアン値 const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorValueBool	**センサーのブーリアン値読込み(SensorBoolean に同じ)** bool SensorValueBoolean(port):ブーリアン値 const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
GetInput	**入力値の取得** variant GetInput(port, field):センサーの各種値を取得 const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか） const byte field: 入力種別フィールドの設定
CustomSensorZeroOffset	**カスタムセンサーのゼロ・オフセット取得** unsigned int CustomSensorZeroOffset(port):ゼロ・オフセット値 const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
CustomSensorPercentFullScale	**カスタムセンサーのパーセント表示取得** byte CustomSensorPercentFullScale(port):パーセント値 byte port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorDigiPinsDirection	**直接センサーのデジタルピンの読込み** byte SensorDigiPinsDirection(port):デジタルピンの直接値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
SensorDigiPinsStatus	**センサーのデジタルピンの読込み** byte SensorDigiPinsStatus(port):デジタルピンの状態値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）

SensorDigiPinsOutputLevel	**センサーのデジタルピン出力レベルの読込み** byte SensorDigiPinsOutputLevel(port): const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか）
ColorADRaw	**LEGO カラーセンサーAD 生データ値の読込み** unsigned int ColorADRaw(port,color): byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte color: カラーセンサー値 INPUT_RED 0 INPUT_GREEN 1 INPUT_BLUE 2 INPUT_BLANK 3 INPUT_NO_OF_COLORS 4
ColorBoolean	**LEGO カラーセンサーのブーリアン値の読込み** bool ColorBoolean(port,color): byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte color: カラーセンサー値
ColorCalibration	**LEGO カラーセンサーのキャリブレーション値の読込み** long ColorCalibration(port,point,color): byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte point:キャリブレーション制御値 INPUT_CAL_POINT_0 0 INPUT_CAL_POINT_1 1 INPUT_CAL_POINT_2 2 INPUT_NO_OF_POINTS 3 byte color: カラーセンサー値
ColorCalLimits	**LEGO カラーセンサーのキャリブレーション限界値の読込み** long ColorCalLimits (port,point): byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte point:キャリブレーション制御値
ColorSensorRaw	**LEGO カラーセンサーの生データ値の読込み** unsigned int ColorSensorRaw(port,color):カラーRaw 値を返す byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte color: カラーセンサー値
ColorSensorValue	**LEGO カラーセンサーのスケール値の読込み** unsigned int ColorSensorValue (port,color): byte port: 入力ポート（S1,S2,S3,S4 のいずれか） byte color: カラーセンサー値
ReadSensorColorEx	**LEGO カラーセンサーその他の値の読込み** int ReadSensorColorEx(port,colorval,raw,norm,scaled):データ値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか） int & colorval:カラー値 INPUT_BLACKCOLOR 1 INPUT_BLUECOLOR 2 INPUT_GREENCOLOR 3 INPUT_YELLOWCOLOR 4 INPUT_REDCOLOR 5 INPUT_WHITECOLOR 6 unsigned int & raw[]:Raw 値(0..1024) unsigned int & norm[]:カラーセンサー値 int & scaled[]:カラーセンサー値

ReadSensorColorRaw	**LEGO カラーセンサーの生データ値の読込み** int ReadSensorColorRaw(port,rawVals): Raw データ値を返す const byte & port: 入力ポート（S1,S2,S3,S4 のいずれか） unsigned int & rawVals[]:カラーセンサー値

◆６）出力関数

関数名	機能/書式・型・パラメータ
Off	**モータの即時停止（OFF）** void Off(outputs): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) OUT_A (0x00) OUT_B (0x01) OUT_C (0x02) OUT_AB (0x03) OUT_AC (0x04) OUT_BC (0x05) OUT_ABC (0x06)
OffEx	**モータの OFF とカウンターのリセット** void OffEx(outputs, reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) const byte reset:タコメータ制御リセットフラグ RESET_ALL (0x68) RESET_BLOCK_COUNT (0x20) RESET_BLOCKANDTACHO (0x28) RESET_COUNT (0x08) RESET_NONE (0x00) RESET_ROTATION_COUNT(0x40)
Coast	**モータの減衰停止** void Coast(outputs):戻り値なし byte outputs: 出力ポート(モータ駆動ポート)
Float	**モータの減衰停止(Coast に同じ)** void Float(outputs): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート)
CoastEx	**モータの減衰停止とカウンターのリセット** void CoastEx(outputs,reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) const byte reset: タコメータ制御リセットフラグ
OnFwd	**モータの前進** void OnFwd(outputs,pwr): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr:出力パワー(0..100)【-100..0 も設定可能】
OnFwdEx	**モータの前進とカウンターのリセット** void OnFwdEx(ouputs,pwr,reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 const byte reset: タコメータ制御リセットフラグ
OnRev	**モータの後退** void OnRev(outputs,pwr): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】

OnRevEx	**モータの後退とカウンターのリセット** void OnRevEx(ouputs,pwr,reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 const byte reset:タコメータ制御リセットフラグ
OnFwdReg	**モータの前進制御** void OnFwdReg(outputs,pwr,regmode): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode:出力ポートの調整モード定数 OUT_REGMODE_IDLE (0) OUT_REGMODE_SPEED (1) OUT_REGMODE_SYNC (2) OUT_REGMODE_POS (4)
OnFwdRegPID	**PID 制御によるモータの前進** void OnFwdRegPID(outputs,pwr,regmode,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数 byte p,i,d: PID 制御値(0..256 または以下の制御値を利用) PID_0 0 PID_1 32 PID_2 64 PID_3 96 PID_4 128 PID_5 160 PID_6 192 PID_7 224
OnFwdRegEx	**モータの前進制御とカウンターのリセット** voidOnFwdRegEx(outputs,pwr,regmode,reset): byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数 const byte reset: タコメータ制御リセットフラグ
OnFwdRegExPID	**PID 制御によるモータの前進とカウンターのリセット** void OnFwdRegExPID(outputs,pwr,regmode,reset,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数 const byte reset: タコメータ制御リセットフラグ byte p,i,d: PID 制御値
OnRevReg	**モータの後退制御** void OnRevReg(outputs,pwr,regmode): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数
OnRevRegPID	**PID 制御によるモータの後退** void OnRevRegPID(outputs,pwr,regmode,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数 byte p,i,d: PID 制御値

OnRevRegEx	**モータの後退制御とカウンターのリセット** voidOnRevRegEx(outputs,pwr,regmode,reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数 const byte reset: タコメータ制御リセットフラグ
OnRevRegExPID	**PID 制御によるモータの後退とカウンターのリセット** void OnRevRegExPID(outputs,pwr,regmode,reset,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 byte regmode: 出力ポートの調整モード定数 const byte reset: タコメータ制御リセットフラグ byte p,i,d: PID 制御値
OnFwdSync	**モータの同期前進（両輪で同期を取って前進）** void OnFwdSync(outputs,pwr,turnpct): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct:回転比率（-１００から１００）
OnFwdSyncPID	**PID 制御によるモータの同期前進** void OnFwdSyncPID(outputs,pwr,turnpct,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct:回転比率（-100･･･100） byte p,i,d: PID 制御値
OnFwdSyncEx	**モータの同期前進とカウンターリセット** void OnFwdSyncEx(outputs,pwr,turnpct,reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct: 回転比率（-100･･･100） const byte reset: タコメータ制御リセットフラグ
OnFwdSyncExPID	**PID 制御によるモータの同期前進とカウンターリセット** void OnFwdSyncExPID(outputs,pwr,turnpct,reset,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct: 回転比率（-100･･･100） const byte reset: タコメータ制御リセットフラグ byte p,i,d: PID 制御値
OnRevSync	**モータの同期後退（両輪で同期を取って後退）** void OnRevSync(outputs,pwr,turnpct): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct: 回転比率（-100･･･100）
OnRevSyncPID	**PID 制御によるモータの同期後退** void OnFwdSyncPID(outputs,pwr,turnpct,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct: 回転比率（-100･･･100） byte p,i,d: PID 制御値

OnRevSyncEx	**モータの同期後退とカウンターリセット** void OnRevSyncEx(outputs,pwr,turnpct,reset): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct: 回転比率（-100･･･100） const byte reset: タコメータ制御リセットフラグ
OnRevSyncExPID	**PID 制御によるモータの同期後退とカウンターリセット** void OnRevSyncExPID(outputs,pwr,turnpct,reset,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 char turnpct: 回転比率（-100･･･100） const byte reset: タコメータ制御リセットフラグ byte p,i,d: PID 制御値
RotateMotor	**モータの角度回転** void RotateMotor(outputs,pwr,angle): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 long angle: 回転角度（度）
RotateMotorPID	**PID 制御によるモータの角度回転** void RotateMotorPID(outputs,pwr,angle, p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 long angle:回転角度（度） byte p,i,d: PID 制御値
RotateMotorEx	**モータの角度回転とカウンターリセット** void RotateMotorEx(outputs,pwr,angle,turnpct,sync,stop): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 long angle: 回転角度（度） char turnpct: 回転比率（-100･･･100） bool sync: 両輪の協調動作（真＝協調動作する、偽＝しない） bool stop: 停止状態（真＝即時停止、偽＝減衰停止）
RotateMotorExPID	**PID 制御によるモータの角度回転とカウンターリセット** void RotateMotorEx(outputs,pwr,angle,turnpct,sync,stop,p,i,d): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) char pwr: 出力パワー(0..100)【-100..0 も設定可能】 long angle: 回転角度（度） char turnpct: 回転比率（-100･･･100） bool sync:両輪の協調動作（真＝協調動作する、偽＝しない） bool stop:停止状態（真＝即時停止、偽＝減衰停止） byte p,i,d: PID 制御値
ResetTachoCount	**タコメータカウンターのリセット** void ResetTachoCount(outputs): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート)
ResetBlockTachoCount	**ブロックの相対カウンターのリセット** void ResetBlockTachoCount(outputs): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート)
ResetRotationCount	**プログラム相対カウンターのリセット** void ResetRotationCount(oputputs):戻り値なし byte outputs:出力ポート（モータ駆動ポート）
ResetAllTachoCounts	**全てのタコメータカウンターのリセット**

	void ResetAllTachoCounts(outputs): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート)
SetOutput	**出力データを設定** void SetOutput(outputs,field1,val1,･･･,fieldN,valN): 戻り値なし byte outputs: 出力ポート(モータ駆動ポート) byte fieldx:**出力モジュール定数** UpdateFlagsField 0 OutputModeField 1 PowerField 2 ActualSpeedField 3 TachoCountField 4 TachoLimitField 5 RunStateField 6 TurnRatioField 7 RegModeField 8 OverloadField 9 RegPValueField 10 RegIValueField 11 RegDValueField 12 BlockTachoCountField 13 RotationCountField 14 OutputOptionsField 15 MaxSpeedField 16 MaxAccelerationField 17 variant valx:設定値
GetOutput	**出力データ値の取得** variant GetOutput(output,field):出力データ値を返す byte output: 出力ポート(モータ駆動ポート) const byte field: **出力モジュール定数**
MotorMode	**モータのモード取得** byte MotorMode(output):モータモードを返す byte output: 出力ポート(モータ駆動ポート)
MotorPower	**モータのパワーレベル取得** char MotorPower(output):モータパワーレベルを返す byte output: 出力ポート(モータ駆動ポート)
MotorActualSpeed	**モータの実際のスピード値取得** char MotorActualSpeed(output):モータの実際のスピード値を返す byte output: 出力ポート(モータ駆動ポート)
MotorTachoCount	**タコメータカウンターの取得** long MotorTachoCout(output):タコメータカウンター値を返す byte output: 出力ポート(モータ駆動ポート)
MotorTachoLimit	**タコメータ制限値の取得** long MotorTachoLimit(output):タコメータ制限値を返す byte output: 出力ポート(モータ駆動ポート)
MotorRunState	**モータ実行状態を取得** byte MotorRunState(output):**モータ実行状態**を返す OUT_RUNSTATE_IDLE 0x00 OUT_RUNSTATE_RAMPUP 0x10 OUT_RUNSTATE_RUNNING 0x20 OUT_RUNSTATE_RAMPDOWN 0x40 OUT_RUNSTATE_HOLD 0x60 byte output: 出力ポート(モータ駆動ポート)

MotorTurnRatio	**モータ回転比率の取得** char MotorTurnRatio(output):モータ回転比率を返す byte output: 出力ポート(モータ駆動ポート)
MotorRegulation	**モータ調整モードの取得** byte MotorRegulation(output):モータ調整値を返す OUT_REGOPTION_NO_SATURATION 0x01 byte output: 出力ポート(モータ駆動ポート)
MotorOverload	**モータの過負荷状態の取得** bool MotorOverload(output): モータの過負荷状態を返す byte output: 出力ポート(モータ駆動ポート)
MotorRegPValue	**モータの P 値取得(PID 制御)** byte MotorRegPValue(output): モータの P 値を返す byte output: 出力ポート(モータ駆動ポート)
MotorRegIValue	**モータの I 値取得(PID 制御)** byte MotorRegIValue(output): モータの I 値を返す byte output: 出力ポート(モータ駆動ポート)
MoorRegDvalue	**モータの D 値取得(PID 制御)** byte MotorRegDValue(output): モータの D 値を返す byte output: 出力ポート(モータ駆動ポート)
MotorBlockTachoCount	**モータのブロック相対カウンターの取得** long MotorBlockTachoCount(output): モータのブロック相対カウンターを返す byte output: 出力ポート(モータ駆動ポート)
MotorRotationCount	**モータのプログラム相対カウンターの取得** long MotorRotationCount(output): モータのプログラム相対カウンターを返す byte output: 出力ポート(モータ駆動ポート)
MotorPwnFreq	**モータの周期を取得（ミリ秒）** byte MotorPwnFreq():モータの周期（ミリ秒）を返す
SetMotorPwnFreq	**モータの周期を設定** void SetMotorPwnFreq(n): 戻り値なし byte n: モータの周期（省略値は 100m 秒）

◆7）サウンド関数

関数名	機能/書式・型・パラメータ
PlayTone	**サウンドの発生** char PlayTone(frequency,duration) unsigned int frequency:音の波長周期・高さ（ヘルツ（Hz）単位） TONE_A3 ラ TONE_AS3 ラ# TONE_B3 シ TONE_C4 ド TONE_CS4 ド# TONE_D4 レ TONE_DS4 レ# TONE_E4 ミ TONE_F4 ファ TONE_FS4 ファ# TONE_G4 ソ TONE_GS4 ソ# : unsigned int duration:ミリ秒単位の音発生の長さ

PlayToneEx	**オプション設定によるサウンド発生** char PlayToneEx(frequency,duration,volume,loop): unsigned int frequency:音の高さ（周期） unsigned int duration:発生音の長さ byte volume:ボリューム(0..4) bool loop:繰り返しスイッチ
PlayFile	**メロディ（NXT melody[RMD]）ファイルのプレイ** char PlayFile(filename): string filename: メロディファイル（rso ファイル : RMD）
PlayFileEx	**オプション設定によるメロディ・ファイルのプレイ** char PlayFileEx(filename,volume,loop): string filename: メロディファイル（rso ファイル : RMD） byte volume: ボリューム(0..4) bool loop: 繰り返しスイッチ
SoundFlags	**サウンドモジュール・フラグの取得** byte SoundFlags():現在サウンドモジュール・フラグを返す SOUND_FLAGS_IDLE 0x00 SOUND_FLAGS_UPDATE 0x01 SOUND_FLAGS_RUNNING 0x02
SetSoundFlags	**サウンドモジュール・フラグの設定** void SetSoundFlags(flags):戻り値なし byte flags: サウンドモジュール・フラグ
SoundState	**サウンド状態の取得** byte SoundState():サウンド状態定数を返す SOUND_STATE_IDLE 0x00 SOUND_STATE_FILE 0x02 SOUND_STATE_TONE 0x03 SOUND_STATE_STOP 0x04
SoundMode	**サウンドモードの取得** byte SoundMode():サウンドモード値が返る SOUND_MODE_ONCE 0x00 SOUND_MODE_LOOP 0x01 SOUND_MODE_TONE 0x02
SetSoundMode	**サウンドモードの設定** void SetSoundMode(mode): 戻り値なし byte mode: 新たなサウンドモード
SoundFrequency	**サウンド周波数を取得** unsigned int SoundFrequency() : 現在のサウンドの周波数を返す
SetSoundFrequency	**サウンド周波数を設定** void SetSoundFrequency(frequency): 戻り値なし unsigned int frequency:サウンドの周波数（高さ）
SoundDuration	**サウンドの長さ（ミリ秒）を取得** unsigned int DoundDration():サウンドの長さを返す
SetSoundDuration	**サウンドの長さ（ミリ秒）を設定** void SetSoundDuration(duration):戻り値なし unsigned int duration:サウンドの長さ
SoundSampleRate	**サウンドサンプルの速度比の取得** unsigned int SoundSampleRate() : サンプルの速度比を返す
SetSoundSampleRate	**サウンドサンプルの速度比の設定** void SetSoundSampleRate(sampleRate) : 戻り値なし unsigned int sampleRate:新たなサンプル速度比

SoundVolume	**サウンドボリュームの取得（Volume に同じ）** byte SoundVolume(): ボリューム値（0..4）
SetSoundVolume	**サウンドボリュームの設定** void SetSoundVolume(volume): 戻り値なし byte volume:ボリューム（0..4）
StopSound	**サウンドの中止** byte StopSound():戻り値（特に無関係）

◆８）IO 制御モジュール関数

関数名	機能/書式・型・パラメータ
PowerDown	**NXT のパワーOFF(SleepNow と同じ)** void PowerDown（）: 戻り値なし
SleepNow	**NXT を即スリープ状態に設定** void SleepNow():戻り値なし
RebootInFirmwareMode	**NXT ファームウェアのダウンロードモードをリブート** void RebootInFirmwareMode():戻り値なし

◆９）LCD 画面表示関数

関数名	機能/書式・型・パラメータ
NumOut	**数値の LCD 画面出力** char NumOut(x,y,value,options):出力結果を返す int x: 出力表示開始ｘ座標値（横座標値　0-99) int y: 出力表示開始ｙ座標値（縦座標値　0-63）また行数(LCD_LINEx) LCD_LINE8　0 LCD_LINE7　8 LCD_LINE6　16 LCD_LINE5　24 LCD_LINE4　32 LCD_LINE3　40 LCD_LINE2　48 LCD_LINE1　58 variant value: 出力する数値 unsigned long options : 画面の初期化設定（省略値:false） DRAW_OPT_NORMAL　0x0000 (**false**) DRAW_OPT_CLEAR_WHOLE_SCREEN　0x0001 (**true**) DRAW_OPT_CLEAR_EXCEPT_STATUS_SCREEN　0x0002 DRAW_OPT_CLEAR_PIXELS　0x0004 DRAW_OPT_CLEAR　0x0004 DRAW_OPT_INVERT　0x0004 DRAW_OPT_LOGICAL_COPY　0x0000 DRAW_OPT_LOGICAL_AND　0x0008 DRAW_OPT_LOGICAL_OR　0x0010 DRAW_OPT_LOGICAL_XOR　0x0018 DRAW_OPT_FILL_SHAPE　0x0020 DRAW_OPT_CLEAR_SCREEN_MODES　0x0003 DRAW_OPT_LOGICAL_OPERATIONS　0x0018 DRAW_OPT_POLYGON_POLYLINE　0x0400

TextOut	**文字列の LCD 画面出力** char TextOut(x,y,str,options): 出力結果を返す int x, y: ピクセル座標値(x,y) string str:出力文字列 unsigned long options: 画面の初期化設定（省略値:false）
GraphicOut	**イメージデータの LCD 画面出力** char GraphicOut(x,y,filename,options): 出力結果を返す int x, y: ピクセル座標値(x,y) string filename: 出力するイメージファイルデータ unsigned long options: 画面の初期化設定（省略値:false）
GraphicOutEx	**パラメータ付きでイメージデータの LCD 画面出力** char GraphicOutEx(x,y,filename,vars,options) : 出力結果を返す int x, y: ピクセル座標値(x,y) string filename:出力ファイル名 byte vars[]:データバイト配列 unsigned long options: 画面の初期化設定（省略値:false）
GraphicArrayOut	**バイト配列からグラフィックイメージの描画出力** char GraphicArrayOut(x,y,data,options): 出力結果を返す int x, y: ピクセル座標値(x,y) byte data[]:データバイト配列 unsigned long options: 画面の初期化設定（省略値:false）
GraphicArrayOutEx	**パラメータ付きバイト配列からグラフィックイメージの LCD 画面出力** char GraphicArrayOutEx(x,y,data,vars,options):出力結果を返す int x,y: ピクセル座標値(x,y) byte data[]:データバイト配列（RIC 画像イメージ） byte vars[]:データバイト配列（パラメータ） unsigned long optins : (=DRAW_OPT_NORMAL)
CircleOut	**円の描画** char CircleOut(x,y,radius,options): 出力結果を返す int x,y : 中心座標値（ピクセル） byte radius: 円の半径（ピクセル） unsigned long options: 画面の初期化設定（省略値:false）
LineOut	**線分の描画** char LineOut(x1,y1,x2,y2,options): 出力結果を返す int x1,y1 : 始点座標値（ピクセル） int x2,y2 : 終点座標値（ピクセル） unsigned long options: 画面の初期化設定（省略値:false）
PointOut	**点の描画** char PointOut(x,y,options): 出力結果を返す int x,y:ポイント描画座標値（ピクセル） unsigned long options: 画面の初期化設定（省略値:false）
RectOut	**長方形の描画** char RectOut(x,y,width,height,options): 出力結果を返す int x,y:右上点の座標値（ピクセル） int width:長方形の幅（ピクセル） int height:長方形の高さ（ピクセル） unsigned long options: 画面の初期化設定（省略値:false）
EllipseOut	**楕円の描画** char EllipseOut(x,y,radiusX,radiusY,options): 出力結果を返す int x,y:楕円の中心座標値（ピクセル） byte radiusX:X 軸半径値（ピクセル）

	byte radiusY:Y 軸半径値（ピクセル） unsigned long options: 画面の初期化設定（省略値:false）
PolyOut	**ポリゴンの描画** char PolyOut(point, options) : 出力結果を返す LocationType point[]: 座標値 x ,y の連続配列（LocationType） unsigned long options: 画面の初期化設定（省略値:false）
FontNumOut	**RIC フォントを使った数値の出力** char FontNumOut(x,y,filename,value,options) : 出力結果を返す int x,y: ポイント描画座標値（ピクセル） string filename:RIC フォントファイル(“***.ric”) variang value:出力する数値 unsigned long options: 画面の初期化設定（省略値:false）
FontTextOut	**RIC フォントを使った文字列の出力** char FontTextOut(x,y,filename,str,options) : 出力結果を返す int x,y: ポイント描画座標値（ピクセル） string filename: RIC フォントファイル(“***.ric”) string str: 出力する文字列 unsigned long options: 画面の初期化設定（省略値:false）
ResetScreen	**LCD スクリーンのリセット(消去)** void ResetScreen():戻り値なし
DisplayFlags	**表示フラグの読込み** byte DisplayFlags(): 表示フラグを返す DISPLAY_ON 0x01 DISPLAY_REFRESH 0x02 DISPLAY_POPUP 0x08 DISPLAY_REFRESH_DISPABLED 0x40 DISPLAY_BUSY 0x80
SetDisplayFlags	**表示フラグの設定** void SetDisplayFlags(flags):戻り値なし byte flags: 表示フラグを返す
DisplayContrast	**表示コントラストの読込み** byte DisplayContrast():出力結果を返す
SetDisplayContrast	**表示コントラストの設定** void SetDisplayContrast(contrast):戻り値なし byte contrast: 0 ~ DISPLAY_CONTRAST_MAX DIPLAY_CONTRAST_MAX 0x7F
DisplayEraseMask	**表示削除マスク値の読込み** unsigned long DisplayEraseMask() : 表示削除マスク値を返す
SetDisplayEraseMask	**表示削除マスク値の設定** void SetDisplayEraseMask(eraseMask):戻り値なし unsigned long eraseMask:新たな削除マスク
DisplayUpdateMask	**表示アップデートマスク値の読込み** unsigned long DisplayUpdateMask() : 表示アップデータマスク値を返す
SetDisplayUpdateMask	**表示アップデートマスク値の設定** void SetDisplayUpdateMask(updateMask): 戻り値なし unsigned long updateMask:新しい表示アップデータマスク値
DisplayDisplay	**ディスプレイメモリーアドレスの読込み** unsigned long DisplayDisplay():現在の表示メモリーアドレスを返す
SetDisplayDisplay	**ディスプレイメモリーアドレスの設定** valid SetDisplayDisplay(dispaddr) : 戻り値なし unsigned long dipaddr:新しい表示メモリーアドレス

DisplayTextLinesCenterFlags	**テキスト表示ラインセンターフラグの読込み** byte DisplayTextLinesCenterFlags():現在表示テキストラインセンターフラグ
SetDisplayTextLinesCenterFlags	**テキスト表示ラインセンターフラグの設定** void SetDisplayTextLinesCenterFlags(ctrFlags): 戻り値なし byte ctrFlags:新しい表示テキストラインセンターフラグ
GetDisplayNormal	**標準表示バッファからピクセル値を読込む** void GetDiplayNormal(x,line,cnt,data): 戻り値なし const byte x:読込むピクセルデータからのｘ座標値 const byte line:読込むピクセルデータからの行数 unsigned int:読込むバイト数 byte & data[] : 読込む配列
SetDisplayNormal	**標準表示バッファにピクセル値を書込む** void SetDiplayNormal(x,line,cnt,data): 戻り値なし const byte x: 書込むピクセルデータからのｘ座標値 const byte line: 書込むピクセルデータからの行数 unsigned int cnt:書込むバイト数 byte & data[]:書込む配列
GetDisplayPopup	**ポップアップディスプレィバッファからピクセルデータを読込む** void GetDisplayPopup(x,line,cnt,data):戻り値なし const byte x: 読込むピクセルデータからのｘ座標値 const byte line: 読込むピクセルデータからの行数 unsigned int cnt: 読込むバイト数 byte & data[] : 読込む配列
SetDisplayPopup	**ポップアップディスプレィバッファにピクセルデータを書込む** void SetDisplayPopup(x,line,cnt,data):戻り値なし const byte x: 書込むピクセルデータからのｘ座標値 const byte line: 書込むピクセルデータからの行数 unsigned int cnt: 書込むバイト数 byte & data[] : 書込む配列

◆１０）ファイル I/O 関数

関数名	機能/書式・型・パラメータ
FreeMemory	**空きフラッシュメモリーの取得** unsigned int FreeMemory():未使用フラッシュメモリーのバイト数を返す
CreateFile	**ファイルの作成** unsigned int CreatFile(fname,fsize,handle): ファイル I/O 結果を返す LDR_SUCCESS 0x0000 LDR_INPROGRESS 0x0001 LDR_REQPIN 0x0002 LDR_NOMOREHANDLES 0x8100 LDR_NOSPACE 0x8200 LDR_NOMOREFILES 0x8300 LDR_EOFEXPECTED 0x8400 LDR_ENDOFFILE 0x8500 LDR_NOTLINEARFILE 0x8600 LDR_FILENOTFOUND 0x8700 LDR_HANDLEALREADYCLOSED 0x8800 LDR_NOLINEARSPACE 0x8900 LDR_UNDEFINEDERROR 0x8A00 LDR_FILESBUSY 0x8B00

	LDR_NOWRITEBUFFERS 0x8C00 LDR_APPLENDNOTPOSSIBLE 0x8D00 LDR_FILEISFULL 0x8E00 LDR_FILEXISTR 0x8F00 LDR_MODEULENOTFOUND 0x9000 LDR_OUTOFBOUNDARY 0x9100 LDR_ILLEGALEFILENAME 0x9200 LDR_ILLEGALHANDLE 0x9300 LDR_BTBUSY 0x9400 LDR_BTCONNECTFAIL 0x9500 LDR_BTTIMEOUT 0x9600 LDR_FILETX_TIMEOUT 0x9700 LDR_FILETX_DSTEXISTS 0x9800 LDR_FILETX_SRCMISSING 0x9900 LDR_FILETX_STREAMERROR 0x9A00 LDR_FILETX_CLOSEERROR 0x9B00 LDR_INVALIDSEEK 0x9C00 string fname:ファイル名 unsigned int fsize:ファイルのサイズ（バイト数） byte & handle:ハンドル（ファイル I/O 内部コード）
CreateFileLinear	**リニア―ファイルの新規作成** unsigned int CreatFileLinear(fname,fsize,handle): ファイル I/O 結果を返す string fname:ファイル名 unsigned int fsize: ファイルのサイズ（バイト数） byte & handle: ハンドル（ファイル I/O 内部コード）
CreateFileNonLinear	**非リニア―なファイルの新規作成** unsigned int CreatFileNonLinear(fname,fsize,handle): ファイル I/O 結果 string fname: ファイル名 unsigned int fsize: ファイルのサイズ（バイト数） byte & handle: ハンドル（ファイル I/O 内部コード）
OpenFileAppend	**追記するファイルのオープン** unsigned int OpenFileAppend(fname,fsize,handle): ファイル I/O 結果 string fname: ファイル名 unsigned int & fsize: ファイルのサイズ（バイト数） byte & handle: ハンドル（ファイル I/O 内部コード）
OpenFileRead	**読込みファイルのオープン** unsigned int OpenFileRead(fname,fsize,handle): ファイル I/O 結果 string fname: ファイル名 unsigned int & fsize: ファイルのサイズ（バイト数） byte & handle: ハンドル（ファイル I/O 内部コード）
OpenFileReadLinear	**リニア―ファイルのオープン** unsigned int OpenFileReadLinear(fname,fsize,handle) : ファイル I/O 結果 string fname: ファイル名 unsigned int & fsize: ファイルのサイズ（バイト数） byte & handle: ハンドル（ファイル I/O 内部コード）
CloseFile	**ファイルのクローズ（fclose に同じ）** unsigned int CloseFile(handle): ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード）
fclose	**ファイルのクローズ** int fclose(handle): 読込みエラーコード byte handle: ハンドル（ファイル I/O 内部コード）
ResolveHandle	**ハンドルの取得**

	unsigned int ResolveHandle(filename,handle,writeable): ファイル I/O 結果 string filename: ファイル名 byte & handle: ハンドル（ファイル I/O 内部コード） bool & writeable:書込み（true）か、読込み（false）か
FindFirstFile	**ファイルのサーチ開始** unsigned int FindFirstFile(fname,handle): ファイル I/O 結果 string & fname: ファイル名 byte & handle: ハンドル（ファイル I/O 内部コード）
FindNextFile	**ファイルのサーチ継続** unsigned int FindNextFile(fname,handle): ファイル I/O 結果 string & fname:ファイル名 byte & handle: ハンドル（ファイル I/O 内部コード）
RenameFile	**ファイルのリネーム（名前変更）** unsigned int RenameFile(oldname,newname): ファイル I/O 結果 string & oldname: 古いファイル名 string & newname:新しいファイル名（変更名）
rename	**ファイルのリネーム（RenameFile に同じ）** int rename(old,new): ファイル I/O 結果 string & old: 古いファイル名 string & new: 新しいファイル名
DeleteFile	**ファイルの削除** unsigned int DeleteFile(fname): ファイル I/O 結果 string fname: ファイル名
remove	**ファイルの削除** int remove(filename): ファイル I/O 結果 string filename: ファイル名
ResizeFile	**ファイルのサイズ変更** unsigned int ResizeFile(fname,newsize) : ファイル I/O 結果 string fname: ファイル名 const unsigned int newsize:新しいサイズ
Read	**ファイルからの数値の読込み** unsigned int Read(handle,vale): ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） variant & value:ファイル読込みデータ変数
fputc / putc **fputs**	**ファイルへの文字の書込み（fputs: 文字列の書込み)** #define fputc(ch, handel) char ch: 出力文字 （fputs では、ch の代わりに string str) byte handel: ハンドル（ファイル I/O 内部コード）
fgetc / getc **fgets**	**ファイルからの文字の読込み（fgets: 文字列の読込み)** char fgetc(handle):ファイルから読込み文字を返す string fgets(str, num, handle) : ファイルからの文字列を返す string & str: 文字列を返すアドレス、 int num: 最大文字読み数 byte handle: ハンドル（ファイル I/O 内部コード）
ReadLn	**ファイルからの行単位で数値読込み** unsigned int ReadLn(handle, value): ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） variant & value: ファイル読込みデータ変数
ReadLnString	**ファイルからの行単位で文字列読込み** unsigned int ReadLnString(handle,output): ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） string & output: ファイル読込みデータ変数

ReadBytes	**ファイルからのバイト読込み** unsigned int ReadByte(handel,length,buf)：ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） unsigned int & length:読込みバイト数 byte & buf[]:ファイル読込みデータ変数
Write	**ファイルへの数値書込み** unsigned int Write(handle, value) ：ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） const variang & value: ファイルへ書込む値
WriteLn	**ファイルへの行単位数値の書込み** unsigned int WriteLn(handle, value) ：ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） const variang & value:ファイルへ書込む値
WriteString	**ファイルへの文字列書込み** unsigned int WriteString(handle, str,cnt) ：ファイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） const string & str:ファイルへ書込む文字列 unsigned int & cnt : ファイルへ書込むバイト数
WriteLnString	**ファイルへの行単位文字列書込み** unsigned int WriteLnStringLn(handle, str,cnt)：ァイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） const string & str: ファイルへ書込む文字列 unsigned int & cnt : ファイルへ書込むバイト数
WriteBytes	**ファイルへのバイト書込み** unsigned int WriteBytes(handle, buf, cnt)：ァイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） const byte & buf:ファイルへ書込む変数 unsigned int & cnt : ファイルへ書込むバイト数
WriteBytesEx	**ファイルへの制限付きバイト書込み** unsigned int WriteBytesEx(handle, len,buf)：ァイル I/O 結果 byte handle: ハンドル（ファイル I/O 内部コード） unsigned int & len:ファイルへ書込むバイト数（長さ） const byte & buf[]:データを含むバイト配列または文字列

◆１１）ボタン関数

関数名	機能/書式・型・パラメータ
ButtonCount	**押しボタンの取得** byte ButtonCount(btn,resetCount):ボタンが押されたカウント数を返す const byte btn:ボタン状態チェック BTN1 0 BTN2 1 BTN3 2 BTN4 3 BTNEXIT BTN1 BTNRIGHT BTN2 BTNLEFT BTN3 BTNCENTER BTN4 NO_OF_BTNS 4 bool reset Count:プレスカウンタがリセットされたかどうか？

ButtonPressed	**押しボタンのチェック** bool ButtonPressed (btn, resetCount):ボタンが押されたかを返す const byte btn: ボタン状態チェック bool reset Count: プレスカウンタがリセットされたかどうか？
ReadButtonEx	**ボタン情報の読込み** char ReadButtonEx(btn,reset,pressed,count):関数読込み結果 const byte btn: ボタン状態チェック bool reset:プレスカウンタ状態 bool & pressed:ボタンが押されたかどうか？ unsigned int & count:プレスカウンタ数？
ButtonPressCount	**ボタンの押されたカウント数の取得** byte ButtonPressCount(btn):ボタンが押されたカウンタ数を返す const byte btn: ボタン状態チェック
ButtonLongPressCount	**ボタンの長押しされたカウント数の取得** byte ButtonLongRessCount(btn):ボタンの長押しされたカウンタ数を返す const byte btn: ボタン状態チェック
ButtonShortReleaseCount	**ボタンの短押しされたカウント数の取得** byte ButtonShortRessCount(btn):ボタンの短押しされたカウンタ数を返す const byte btn: ボタン状態チェック
ButtonLongReleaseCount	**ボタンの長リリースされたカウント数の取得** byte ButtonLongReleaseCount(btn):ボタンの長リリースされたカウント数 const byte btn: ボタン状態チェック
ButtonReleaseCount	**ボタンのリリースされたカウント数の取得** byte ButtonReleaseCount(btn): ボタンのリリースされたカウント数 const byte btn: ボタン状態チェック
ButtonState	**ボタンの状態の取得** byte ButtonState(btn):ボタン状態チェックを返す const byte btn:ボタン状態 BTNSTATE_PRESSED_EV 0x01 BTNSTATE_SHORT_RELEASED_EV 0x02 BTNSTATE_LONG_PRESSED_EV 0x04 BTNSTATE_LONG_RELEASED_EV 0x08 BTNSTATE_PRESSED_STATE 0x80 BTNSTATE_NONE 0x10

◆１２）ユーザインタフェース関数

関数名	機能/書式・型・パラメータ
Volume	**ボリューム値の読込み** byte Volume(): ボリューム値(0..4)を返す
SetVolume	**ボリューム値の設定** void SetVolume(volume):戻り値なし byte volume:ボリューム値（0..4）
BatteryLevel	**バッテリーレベルの取得** unsigned int BatteryLevel(): 単位 mV のバッテリーレベルを返す
BluetoothState	**Bluetooth の状態を取得** byte BluetoothState(): Bluetooth の状態を返す UI_BT_STATE_VISIBLE 0x01 UI_BT_STATE_CONNECTED 0x02 UI_BT_STATE_OFF 0x04 UI_BT_ERROR_ATTENTION 0x08

	UI_BT_CONNECT_REQUEST 0x40 UI_BT_PIN_REQUEST 0x80
SetBluetoothState	**Bluetooth の状態設定** void SetBluetoothState(state): 戻り値なし byte state: Bluetooth の状態
CommandFlags	**コマンドフラグの取得** byte CommandFlags():コマンドフラグを返す UI_FLAGS_UPDATE 0x01 UI_FLAGS_DISABLE_LEFT_RIGHT_ENTER 0x02 UI_FLAGS_DISABLE_EXIT 0x04 UI_FLAGS_REDRAW_STATUS 0x08 UI_FLAGS_RESET_SLEEP_TIMER 0x10 UI_FLAGS_EXECUTE_LMS_FILE 0x20 UI_FLAGS_BUSY 0x40 UI_FLAGS_ENABLE_STATUS_UPDATE 0x80
SetCommandFlags	**コマンドフラグの設定** void SetCommandFlages(cmdFlags): 戻り値なし const byte cmdFlags:コマンドフラグ
UIState	**UI モジュール状態の取得** byte UIState(): UI 状態を返す UI_STATE_INIT_DISPLAY 0 UI_STATE_INIT_LOW_BATTERY 1 UI_STATE_INIT_INTRO 2 UI_STATE_INIT_WAIT 3 UI_STATE_INIT_MENU 4 UI_STATE_NEXT_MENU 5 UI_STATE_DRAW_MENU 6 UI_STATE_TEST_BUTTONS 7 UI_STATE_LEFT_PRESSED 8 UI_STATE_RIGHT_PRESSED 9 UI_STATE_ENTER_PRESSED 10 UI_STATE_EXIT_`PRESSED 11 UI_STATE_CONNECT_REQUEST 12 UI_STATE_EXECUTE_FILE 13 UI_STATE_EXECUTING_FILE 14 UI_STATE_LOW_BATTERY 15 UI_STATE_BT_ERROR 16
SetUIState	**UI 状態の設定** void SetUIState(state): 戻り値なし byte state: UI 状態
VMRunState	**VM 実行状態の取得** byte VMRunState():VM 実行状態を返す UI_VM_IDLE 0 UI_VM_RUN_FREE 1 UI_VM_RUN_SINGLE 2 UI_VM_RUN_PAUSE 3 UI_VM_RESET1 4 UI_VM_RESET2 5
SetVMRunState	**VM 実行状態を設定** void SetVMRunState(vmRunState):戻り値なし const byte vmRunState: VM 実行状態
BatteryState	**バッテリー状態を取得**

	byte BatteryState(): バッテリー情報(0..4)を返す
RechargeableBattery	**バッテリータイプを取得** bool RechargeableBattery(): 蓄電用バッテリー（true=yes)、 他(false=no)を返す
ForceOff	**NXT の電源を切る(OFF)** void ForceOff(num): 戻り値なし byte num: ゼロ以外で電源 OFF
UsbState	**UI モジュールの USB 状態を取得** byte UsbState(): USB 状態を返す 0=disconnected, 1=connected, 2=working
OnBrickProgramPointer	**ブロック上のプログラム・ポインター値を取得** byte OnBrickProgramPointer(): 現状 OBP(on-brick program)値を返す
LongAbort	**長い中断設定を取得** byte LongAbort(): ボタン状態を返す BTNSTATE_PRESSED_EV 0x01 BTNSTATE_SHORT_RELEASED_EV 0x02 BTNSTATE_LONG_PRESSED_EV 0x04 BTNSTATE_LONG_RELEASED_EV 0x08 BTNSTATE_PRESSED_STATE 0x80 BTNSTATE_NONE 0x10
SetLongAbort	**長期中断を設定** void SetLongAbort(longAbort): 戻り値なし bool longAbort:エスケープボタンが長押しされた状態 true でプログラム中断
AbortFlag	**中断フラグの取得** byte AbortFlag(): ボタン状態を返す
SetAbortFlag	**中断フラグの設定** void SetAbortFlag(abortFlag): 戻り値なし byte abortFlag:新しい中断フラグの設定

◆１３）Lowspeed　I2C 関数

関数名	機能/書式・型・パラメータ
LowspeedWrite （I2CWrite）	**超音波(I2C)センサーデータの書込み** long LowspeedWrite(port,retlen,buffer):状態コードを返す NO_ERR 0 POOL_MAX_SIZE 32768 STAT_COMM_PENDING 32 STAT_MSG_EMPTY_MAILBOX 64 const byte port:入力ポート番号（S1,S2,S3,S4） byte retlen:バイト数 byte buffer[]：書込むバッファデータ配列
LowSpeedStatus (I2CStatus)	**超音波(I2C)センサー状態を取得** long LowspeedStatus(port,bytesready)：状態コードを返す constbyte port: 入力ポート番号（S1,S2,S3,S4） byte & bytesready: 読込むバイト数
LowspeedCheckStatus (I2CCheckStatus)	**超音波(I2C)センサー状態のチェック** long LowspeedCheckStatus(port): 状態コードを返す const byte port: 入力ポート番号（S1,S2,S3,S4）

LowspeedBytesReady (I2CbytesReady)	**超音波(I2C)センサーの待ちバイト取得（最大 16）** byte LowspeedBytesReady(port):読込みバイト取得を返す const byte port: 入力ポート番号（S1,S2,S3,S4）
LowspeedRead (I2CRead)	**超音波(I2C)センサーのデータ取得** long LowsppeRead(port,buflen,buffer): 状態コードを返す const byte port: 入力ポート番号（S1,S2,S3,S4） byte buflen:出力バッファ数 byte & buffer[]:読込むバッファデータ配列
I2CBytes	**I2C フォーマットの書込み/読込みトランザクション** long I2CBytes(port,inbuf,cons,outbuf) : 状態コードを返す const byte port: 入力ポート番号（S1,S2,S3,S4） byte inbuf[]:書込むバッファデータ配列 byte & count:バイト数 byte & outbuf[]:読込むバッファデータ配列
LSMode	**超音波（I2C）モードを取得** byte LSMode(port):I2C モードを返す LOWSPEED_TRANSMITTING 1 LOWSPEED_RECEIVING 2 LOWSPEED_DATA_RECEIVED 3 const byte port: 入力ポート番号（S1,S2,S3,S4）
LSChannelState	**超音波（I2C）チャネル状態の取得** byte LSChannelState(port):I2C チャンネル状態を返す LOWSPEED_IDLE 0 LOWSPEED_INIT 1 LOWSPEED_LOAD_BUFFER 2 LOWSPEED_COMMUNICATIONG 3 LOWSPEED_ERROR 4 LOWSPEED_DONE 5 const byte port: 入力ポート番号（S1,S2,S3,S4）
LSErrorType	**超音波（I2C）エラータイプの取得** byte LSErrorType(port):I2C エラータイプを返す LOWSPEED_NO_ERROR 0 LOWSPEED_CH_NOT_READY 1 LOWSPEED_TX_ERROR 2 LOWSPEED_RX_ERROR 3 const byte port: 入力ポート番号（S1,S2,S3,S4）
LSState	**超音波（I2C）状態の取得** byte LSState():I2C 状態を返す COM_CHANNEL_NONE_ACTIVE 0x00 COM_CHANNEL_ONE_ACTIVE 0x01 COM_CHANNEL_TWO_ACTIVE 0x02 COM_CHANNEL_THREE_ACTIVE 0x04 COM_CHANNEL_FOR_ACTIVE 0x08
LSSpeed	**超音波（I2C）スピードの取得** byte LSSpeed():I2C スピードを返す

◆１４）Bluetooth 関数

関数名	機能/書式・型・パラメータ
SendRemoteBool	**リモート・メールボックスに bool 値を送信** char SendRemoteBool(conn,queue,bval):関数呼び出しの成功・失敗を返す byte conn: 接続スロット（0..4） CONN_BT0 0x0 CONN_BT1 0x1 CONN_BT2 0x2 CONN_BT3 0x3 CONN_HS4 0x4 CONN_HS_ALL 0x4 CONN_HS_1 0x5 CONN_HS_2 0x6 CONN_HS_3 0x7 CONN_HS_4 0x8 CONN_HS_5 0x9 CONN_HS_6 0xa CONN_HS_7 0xb CONN_HS_8 0xc byte queue: **メールボックス番号** MAILBOX1 0 MAILBOX2 1 MAILBOX3 2 MAILBOX4 3 MAILBOX5 4 MAILBOX6 5 MAILBOX7 6 MAILBOX8 7 MAILBOX9 8 MAILBOX10 9 bool bval:送ったデータの真偽
SendRemoteNumber	**リモート・メールボックスに数値を送信** char SendRemoteNumber(conn,queue,val): 関数呼び出しの成功・失敗を返す byte conn:接続スロット（0..4）4 のみ RS485 通信 byte queue:メールボックス番号 long val: 送信する数値
SendRemoteString	**リモート・メールボックスに文字列を送信** char SendRemoteString(conn,queue,str): 関数呼び出しの成功・失敗を返す byte conn:接続スロット（0..4）4 のみ RS485 通信 byte queue:メールボックス番号 string str: 送信する文字列
SendResponseBool	**ローカル応答メールボックスに bool 値を書込み** char SendResponseBool(queue,bval): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 bool bval:送信するブーリアン値
SendResponseNumber	**ローカル応答メールボックスに数値を書込み** char SendResponseNumber(queue,val): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 long val:送信数値

SendResponseString	**ローカル応答メールボックスに文字列を書込み** char SendResponseString(queue,str): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 string str:送信文字列
ReceiveRemoteBool	**リモート・メールボックスから bool 値を受信** char ReceiveRemoteBool(queue,clear,bval): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 bool clear:読込み後のメールボックスの中のメッセージ削除の状態 bool & bval:メールボックスから読込み成功（true）か失敗（false）か？
ReceiveRemoteNumber	**リモート・メールボックスから数値を受信** char ReceiveRemoteNumber(queue,clear,val): 関数呼び出しの成功・失敗 byte queue: メールボックス番号 bool clear: 読込み後のメールボックスの中のメッセージ削除の状態 long & val: メールボックスから読込み成功（true）か失敗（false）か？
ReceiveRemoteString	**リモート・メールボックスから文字列を受信** char ReceiveRemoteString(queue,clear,val): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 bool clear: 読込み後のメールボックスの中のメッセージ削除の状態 long & val: メールボックスから読込み成功（true）か失敗（false）か？
ReceiveRemoteMessageEx	**待ち行列のメールボックスから値を受信** char ReceiveRemoteMessageEx(queue,clear,str,val,bval): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 bool clear: 読込み後のメールボックスの中のメッセージ削除の状態 string & str:メールボックスから読み込まれる文字列 long & val: メールボックスから読み込まれる数値 long & bval:メールボックスから読み込まれるブーリアン値
SendMessage	**待ち行列のメールボックスへメッセージを送信** char SendMessage(queue,msg): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 string msg:メールボックスへ書込まれるメッセージ
ReceiveMessage	**待ち行列のメールボックスからメッセージを受信** char ReceiveMessage(queue,clear,msg): 関数呼び出しの成功・失敗を返す byte queue: メールボックス番号 bool clear: 読込み後のメールボックスの中のメッセージ削除の状態 string & msg:メールボックスから読込まれるメッセージ
BluetoothStatus	**Bluetooth 状態をチェック** char BluetoothStatus(conn): 関数呼び出しの成功・失敗を返す byte conn: 接続スロット（0..4）4 のみ RS485 通信
BluetoothWrite	**Bluetooth 接続先を書込み** char BluetoothWrite(conn,buffer): 関数呼び出しの成功・失敗を返す byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte buffer[]:書込むバイトデータ（最大 128 バイト）
RemoteMessageRead	**ローカル応答メールボックスからメッセージを受信** char RemoteMessageRead(conn,queue): 関数呼び出しの成功・失敗を返す byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte queue: メールボックス番号
RemoteMessageWrite	**ローカル応答メールボックスへメッセージを送信** char RemoteMessageWrite(conn,queue,msg): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte queue: メールボックス番号

	string msg:メールボックスへ書込むメッセージ
RemoteStartProgram	**リモートでプログラムを起動** char RemoteStartProgram(conn,filename): 関数呼び出しの成功・失敗を返す byte conn: 接続スロット（0..4）4 のみ RS485 通信 string filename:スタートするプログラムファイル名
RemoteStopProgram	**リモートでプログラムを停止** char RemoteStopProgram(conn): 関数呼び出しの成功・失敗を返す byte conn: 接続スロット（0..4）4 のみ RS485 通信
RemotePlaySoundFile	**リモートでサウンドファイルをプレイ** char RemotePlaySoundFile(conn,filename,bloop): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信 string filename:サウンドファイル名 bool bloop:ループのスイッチ
RemotePlayTone	**リモートで音(トーン)をプレイ** char RemotePlayTone(conn,frequency,duration): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信 unsigned int frequency:音の周波数 unsigned int duration:音の長さ
RemoteStopSound	**リモートでサウンドを停止** char RemoteStopSound(conn): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信
RemoteKeepAlive	**KeepAlive メッセージを送信** char RemoteKeepAlive(conn): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信
RemoteResetscaledValue	**ResetScaledValue メッセージを送信** char RemoteResetscaledValue(conn,port): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte port:リセットする入力ポート番号
RemoteResetMotorPosition	**ResetMotorPosition メッセージを送信** char RemoteResetMotorPosition(conn,port,brelative): 関数呼び出しの成功・失敗 byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte port: リセットする入力ポート番号 bool brelative:カウンタリセットのフラグ
RemoteSetInputMode	**SetInputMode メッセージを送信** char RemoteSetInputMode(conn,port,type,mode): byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte port: リセットする入力ポート番号 byte type:センサータイプ番号 byte mode:センサーモード番号
RemoteSetOutputState	**SetOutputMode メッセージを送信** char RemoteSetOutputState (conn,port,speed,mode,regmode,turnpct, runstate, tacholimit) byte conn: 接続スロット（0..4）4 のみ RS485 通信 byte port: リセットする入力ポート番号 char speed:モータパワー（-100...100） byte mode: センサーモード番号 byte regmode: 出力ポートの調整モード定数 char turnpct:モータ両輪の協調動作パーセント（-100...100） byte runstate: モータ実行状態 unsingned long tachoilimit:モータタコメータの制限値

◆１５）USB 関数

関数名	機能/書式・型・パラメータ
GetUSBInputBuffer	**USB 入力バッファデータを取得** void GetUSBInputBuffer(offset,cnt,data):戻り値なし const byte offset:USB 入力バッファのオフセット状態 byte cnt:読込みバイト数 byte & data[]:USB 入力バッファからの読込みデータ配列
SetUSBInputBuffer	**USB 入力バッファデータを設定** void SetUSBInputBuffer(offset,cnt,data): 戻り値なし const byte offset: USB 入力バッファのオフセット状態 byte cnt: 読込みバイト数 byte data[]:USB 入力バッファからの読込みデータ配列
SetUSBInputBufferInPtr	**USB 入力バッファに in-pointer を設定** void SetUSBInputBufferInPtr(n): 戻り値なし byte n: 新たな in-pointer 値(0..63)
USBPollBufferInPtr	**USB ポーリングバッファ in-pointer の取得** byte USBPollBufferInPtr():USB ポートポーリングバッファの in-pointer 値
SetUSBInputBufferOutPtr	**USB 入力バッファ out-pointer の設定** void SetUSBInputBufferOutPtr(n): 戻り値なし byte n:新しい out-pointer 値（0..63）
USBInputBufferOutPtr	**USB ポート入力バッファ out-pointer の取得** byte USBInputBufferOutPtr():USB ポーリングバッファの out-pointer 値
GetUSBOutputBuffer	**USB 出力バッファデータの取得** void GetUSBOutputBuffer(offset,cnt,data):戻り値なし const byte offset:USB 出力バッファのオフセット状態 byte cnt:読込みバイト数 byte & data[]:USB 入力バッファからの読込みデータ配列
SetUSBOutputBuffer	**USB 出力バッファデータの設定** void SetUSBOutputBuffer(offset, cnt,data): 戻り値なし const byte offset: USB 出力バッファのオフセット状態 byte cnt:読込みバイト数 byte data[]:USB 入力バッファへ書込むデータ配列
SetUSBOutputBufferInPtr	**USB 出力バッファ in-pointer の設定** void SetUSBOutputBufferInPtr(n): 戻り値なし byte n: 新しい out-pointer 値（0..63）
USBOutputBufferInPtr	**USB ポート出力バッファ in-pointer の取得** byte USBOutputBufferInPtr():**USB ポート出力バッファ in-pointer 値を返す**
SetUSBOutputBufferOutPtr	**USB 出力バッファ out-pointer の設定** void SetUSBOutputBufferOutPtr(n): 戻り値なし byte n: 新しい out-pointer 値（0..63）
USBOutputBufferOutPtr	**USB ポート出力バッファ out-pointer の取得** byte USBOutputBufferOutPtr():USB ポート出力バッファ out-pointer 値を返す
GetUSBPollBuffer	**USB ポーリングバッファデータの取得** void GetUSBPollBuffer(offset,cnt,data): 戻り値なし const byte offset: USB 出力バッファのオフセット状態 byte cnt:読込みバイト数 byte & data[]:USB 入力バッファからの読込みデータ配列
SetUSBPollBuffer	**USB ポーリングバッファデータの設定** void SetUSBPollBuffer(offset,cnt,data): 戻り値なし

	const byte offset: USB 出力バッファのオフセット状態 byte cnt:読込みバイト数 byte data[]:USB 入力バッファへ書込むデータ配列
SetUSBPollBufferInPtr	**USB ポーリングバッファ in-pointer の設定** void SetUSBPollBufferInPtr(n): 戻り値なし byte n: 新しい out-pointer 値（0..63）
USBPollBufferInPtr	**USB ポートポーリングバッファ in-pointer の取得** byte USBPollBufferInPtr():USB ポートポーリングバッファ in-pointer
SetUSBPollBufferOutPtr	**USB ポーリングバッファ out-pointer の設定** void SetUSBPollBufferOutPtr(n): 戻り値なし byte n: 新しい out-pointer 値（0..63）
USBPollBufferOutPtr	**USB ポートポーリングバッファ out-pointer の取得** byte USBPollBufferOutPtr():USB ポートポーリングバッファ out-pointer
SetUSBState	**USB 状態を設定** void SetUSBState(usbState): 戻り値なし byte usbState:usb 状態（0：未接続、1:接続中、2:通信中）
USBState	**USB 状態** byte USBState():USB 状態を返す（0：未接続、1:接続中、2:通信中）

◆１６）HiTechnic 社センサー関数

関数名	機能/書式・型・パラメータ
	USB 入力バッファデータを取得 void GetUSBInputBuffer(offset,cnt,data):戻り値なし const byte offset:USB 入力バッファのオフセット状態 byte cnt:読込みバイト数 byte & data[]:USB 入力バッファからの読込みデータ配列
SetUSBInputBuffer	**USB 入力バッファデータを設定** void SetUSBInputBuffer(offset,cnt,data): 戻り値なし const byte offset: USB 入力バッファのオフセット状態 byte cnt: 読込みバイト数 byte data[]:USB 入力バッファからの読込みデータ配列
SetSensorHTGyro	**HiTechnic 社ジャイロセンサーの設定** void SetSensorHTGyro (port): 戻り値なし const byte & port: 入力ポート
SensorHTGyro	**HiTechnic 社ジャイロセンサーの読込み** int SensorHTGyro (port , offset) : ジャイロセンサー値を返す Const byte & port : 入力ポート Int offset : ゼロ オフセット値
SetSensorHTEOPD	**HiTechnic 社 EOPD センサーの設定** void SetSensorHTEOPD (port, hStandard): 戻り値なし const byte & port : 入力ポート bool hStandard : 標準または長レンジモードの調整
SensorHTEOPD	**HiTechnic 社 EOPD センサーの読込み** int SensorHTEOPD (port): EOPD センサー読込値を返す const byte & port : 入力ポート bool hStandard : 標準または長レンジモードの調整
SetHTIRSeeker2Mode	**HiTechnic 社 IRSeeker2 のモード設定** char SetHTIRSeeker2Mode (port, mode):状態コードを返す const byte & port : 入力ポート const byte mode :HiTechnic 社 IRSeeker2 のモード値

	HTIR2_MODE_1200 0 HTIR2_MODE_600 1
SensorHTIRSeeker2ACDir	**HiTechnic 社 IRSeeker2 AC 方位センサー値の読込み** int SensorHTIRSeeker2ACDir (port): AC 方位センサー値を返す const byte & port : 入力ポート
SensorHTIRSeeker2DCDir	**HiTechnic 社 IRSeeker2 DC 方位センサー値の読込み** int SensorHTIRSeeker2DCDir (port): DC 方位センサー値を返す const byte & port : 入力ポート
SensorHTIRSeeker2Addr	**HiTechnic 社 IRSeeker2 レジスタ読込み** int SensorHTIRSeeker2Addr (port , reg): IRSeeker2 レジスタ値を返す const byte & port : 入力ポート const byte reg : HiTechnic IRSeeker2 の定数値 HTIR2_REG_MODE 0x41 HTIR2_REG_DCDIR 0x42 HTIR2_REG_DC01 0x43 HTIR2_REG_DC02 0x44 HTIR2_REG_DC03 0x45 HTIR2_REG_DC04 0x46 HTIR2_REG_DC05 0x47 HTIR2_REG_DCAVG 0x48 HTIR2_REG_ACDIR 0x49 HTIR2_REG_AC01 0x4A HTIR2_REG_AC02 0x4B HTIR2_REG_AC03 0x4C HTIR2_REG_AC04 0x4D HTIR2_REG_AC05 0x4E
SensorHTCompass	**HiTechnic 社コンパス・センサーの値の読込み** int SensorHTCompass (port): コンパス方向（センサー値）を返す const byte & port : 入力ポート
ReadSensorHTAccel	**HiTechnic 社 加速度センサーの値の読込み** bool ReadSensorHTAccel (port, x, y, z): 関数の結果を返す const byte & port : 入力ポート int & x, y, z : 各方向（ｘ、 ｙ、 ｚ）の加速度値（リターン値）
ReadSensorHTColor	**HiTechnic 社 カラーセンサーの値の読込み** bool ReadSensorHTColor (port,ColorNum,Red,Green,Blue): 関数結果を返す const byte & port : 入力ポート byte & ColorNum: カラー番号の出力値 byte & Red : 赤色の出力値 byte & Green : 緑色の出力値 byte & Blue : 青色の出力値
ReadSensorHTNormalizedColor	**HiTechnic 社 カラーセンサーの標準値の読込み** bool ReadSensorHTNormalizedColorl (port,ColorIdx,Red,Green,Blue): 関数結果を返す const byte & port : 入力ポート byte & ColorIdx: カラーインデックス値 byte & Red : 赤色の出力標準値 byte & Green : 緑色の出力標準値 byte & Blue : 青色の出力標準値
ReadSensorHTRawColor	**HiTechnic 社 カラーセンサーの Raw 値の読込み** bool ReadSensorHTRawColorl (port, Red,Green,Blue): 関数結果を返す const byte & port : 入力ポート byte & Red : 赤色の出力 Raw 値

	byte & Green： 緑色の出力 Raw 値 byte & Blue： 青色の出力 Raw 値
ReadSensorHTIRSeeker	**HiTechnic 社 IRSeeker の値の読込み** bool ReadSensorHTIRSeeker (port, dir,s1,s3,s5,s7,s9): 関数結果を返す const byte & port： 入力ポート byte & dir： 方向値を返す byte & s1,s3,s5,s7,s9:センサーs1~s9 からのシグナル値を返す
ReadSensorHTIRSeeker2AC	**HiTechnic 社 IRSeeker の AC 値の読込み** bool ReadSensorHTIRSeekerAC (port, dir,s1,s3,s5,s7,s9): 関数結果を返す const byte & port： 入力ポート byte & dir： 方向値を返す byte & s1,s3,s5,s7,s9:センサーs1~s9 からのシグナル値を返す
ReadSensorHTIRSeeker2DC	**HiTechnic 社 IRSeeker の DC 値の読込み** bool ReadSensorHTIRSeekerDC (port, dir,s1,s3,s5,s7,s9): 関数結果を返す const byte & port： 入力ポート byte & dir： 方向値を返す byte & s1,s3,s5,s7,s9:センサーs1~s9 からのシグナル値を返す
ReadSensorHTTouchMultiplexer	**HiTechnic 社 多重タッチセンサー値の読込み** bool ReadSensorHTTouchMultiplexer (port, t1, t2, t3, t4): 関数結果を返す const byte & port： 入力ポート byte & t1,t2,t3,t4： タッチセンサー1~4 の値を返す
HTPowerFunctionCommand	
HTPFTrain	**HiTechnic 社の IRlink デバイスのパワー制御関数** char HTPFTrain(port,channel,func): const byte port: 入力ポート const byte channel: パワー関数チャネル値 PF_CHANNEL_1 0 PF_CHANNEL_2 1 PF_CHANNEL_3 2 PF_CHANNEL_4 3 const byte func: パワー関数 TRAIN_FUNC_STOP 0 TRAIN_FUNC_INCR_SPEED 1 TRAIN_FUNC_DECR_SPEED 2 TRAIN_FUNC_TOGGLE_LIGHT 4
HTIRTrain	**HiTechnic 社の IR Train レシーバのパワー制御関数** char HTIRTrain(port,channel,func): const byte port: 入力ポート const byte channel: IRTrain のチャネル値 TRAIN_CHANNEL_1 0 TRAIN_CHANNEL_2 1 TRAIN_CHANNEL_3 2 TRAIN_CHANNEL_ALL 3 const byte func: IRTrain のパワー関数
HTPFComboDirec	**HiTechnic 社の ComboDirec 関数** char HTPFComboDirec (port, channel,outa, outb): 呼び出し結果を返す const byte port: 入力ポート const byte channel: IRTrain のチャネル値 const byte outa, outb：outa と outb のパワー関数コマンド PF_CMD_STOP 0 PF_CMD_FLOAT 0 PF_CMD_FWD 1

	PFD_CMD_REV 2 PFD_CMD_BRAKE 3
HTPFComboPWM	**HiTechnic 社の ComboPWM 関数** char HTPFComboPWM (port, channel,outa, outb): 呼び出し結果を返す const byte port: 入力ポート const byte channel: IRTrain のチャネル値 const byte outa, outb：outa と outb のパワー関数コマンド PF_PWM_FLOAT 0 PF_PWM_FWD1 1 PF_PWM_FWD 2 2 PF_PWM_FWD 3 3 PF_PWM_FWD4 4 PF_PWM_FWD5 5 PF_PWM_FWD6 6 PF_PWM_FWD7 7 PF_PWM_BREAK 8 PF_PWM_REV7 9 PF_PWM_REV6 10 PF_PWM_REV5 11 PF_PWM_REV4 12 PF_PWM_REV3 13 PF_PWM_REV2 14 PF_PWM_REV1 15
HTPFSingleOutputCST	**HiTechnic 社の PFSingleOutputCST 関数** char HTPFSingleOutputCST (port, channel,out, func): 呼び出し結果を返す const byte port: 入力ポート const byte channel: IRTrain のチャネル値 const byte out:パワー関数の出力定数 PF_OUT_A 0 PF_OUT_B 1 const byte func:パワー関数コマンド PF_CST_CLEAR1_CLEAR2 0 PF_CST_SET1_CLEAR2 1 PF_CST_CLEAR1_SET2 2 PF_CST_SEET1_SET2 3 PF_CST_INCREMENT_PWM 4 PF_CST_DECREMENT_PWM 5 PF_CST_FULL_FwD 6 PF_CST_RULL_REV 7 PF_CST_TOGGLE_DIR 8
HTPFSingleOutputPWM	**HiTechnic 社の PFSingleOutputPWM 関数** char HTPFSingleOutputPWM (port, channel,out, func): 呼び出し結果を返す const byte port: 入力ポート const byte channel: IRTrain のチャネル値 const byte out:パワー関数の出力定数 PF_CHANNEL_1 0 PF_CHANNEL_2 1 PF_CHANNEL_3 2 PF_CHANNEL_4 3 const byte func:パワー関数コマンド

HTPFSinglePin	**HiTechnic 社の** HTPFSinglePin **関数** char HTPFSinglePin (port, channel,out,pin, func,cont): 呼び出し結果を返す const byte port: 入力ポート const byte channel: IRTrain のチャネル値 const byte out:パワー関数の出力定数 const byte pin:ピンの値定数 PF_PIN_C1 0 PF_PIN_C2 1 const byte func : パワー関数定数 TRAIN_FUNC_NOCHANGE 0 TRAIN_FUNC_CLEAR 1 TRAIN_FUNC_SET 2 TRAIN_FUNC_TOGGLE 3 bool cont: 継続か(ture)、中断か(false)のモード
HTPFRawOutput	**HiTechnic 社の HTPFRawOutput 関数** char HTPFRawOutput (port, nibble0,nibble1,nibble2): 呼び出し結果を返す const byte port: 入力ポート const byte nibble0,nibble1,nibble2 :ニブル値
HTPFRepeat	**HiTechnic 社の HTPFRepeat 関数** char HTPFRepeat (port, count,delay): 呼び出し結果を返す const byte port: 入力ポート const byte count: 繰返し数 const unsigned int delay:繰返しの間隔（ミリ秒）