

21

プログラム命令、 記述式一覧

この章では、GP-Pro EX の「スクリプトで使用する各命令・記述式」について説明します。
なお、スクリプトのプログラミング方法については「20章 機能をプログラミングしたい（部品を使わないプログラミング）」(20-1 ページ)をお読みください。

21.1	ビット操作	21-2
21.2	描画	21-3
21.3	メモリ操作	21-7
21.4	SIO ポート操作	21-24
21.5	CF ファイル操作	21-36
21.6	プリンタ操作	21-56
21.7	その他	21-61
21.8	記述式	21-66
21.9	比較	21-69
21.10	演算子	21-71
21.11	文字列操作	21-74
21.12	演算例	21-90
21.13	命令一覧	21-94

21.1 ビット操作

ビット操作	動作概要
	ビット設定 「21.1.1 ビット設定」(21-2 ページ) 指定したビットアドレスを 0 → 1 にします。
	ビットクリア 「21.1.2 ビットクリア」(21-2 ページ) 指定したビットアドレスを 1 → 0 にします。
	ビットトグル 「21.1.3 ビットトグル」(21-2 ページ) 指定したビットアドレスを 1 → 0 もしくは 0 → 1 にします。

21.1.1 ビット設定

項目	内容
概要	指定したビットアドレスを 0 → 1 にします。
書式	set ()

記述例

```
set ([b:[#INTERNAL]LS010000])
```

上記の例では、LS0100 の 00 ビット目を 0 → 1 にします。

21.1.2 ビットクリア

項目	内容
概要	指定したビットアドレスを 1 → 0 にします。
書式	clear ()

記述例

```
clear ([b:[#INTERNAL]LS010000])
```

上記の例では、LS0100 の 00 ビット目を 1 → 0 にします。

21.1.3 ビットトグル

項目	内容
概要	指定したビットアドレスを 1 → 0 もしくは 0 → 1 にします。
書式	toggle ()

記述例

```
toggle ([b:[#INTERNAL]LS010000])
```

上記の例では、LS0100 の 00 ビット目を 1 → 0 もしくは 0 → 1 にします。

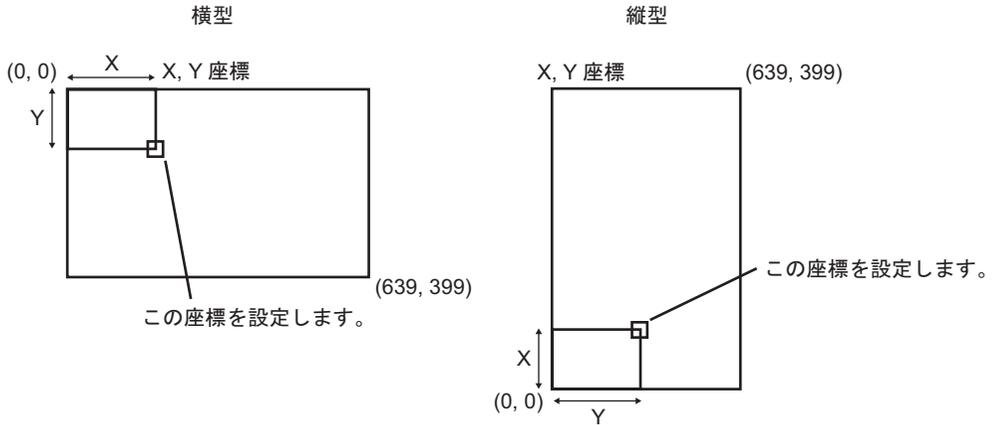
21.2 描画

描画	動作概要
	画面呼出  「21.2.1 画面呼出」(21-3 ページ) 指定した画面番号の画面 (ベース画面) を呼び出します。 拡張スクリプトでは使用できません。
	円  「21.2.2 円」(21-4 ページ) 指定した円を描画します。
	ドット  「21.2.3 ドット」(21-5 ページ) 指定した点を描画します。
	直線  「21.2.4 直線」(21-5 ページ) 指定した直線を描画します。
	四角  「21.2.5 四角」(21-6 ページ) 指定した四角を描画します。

21.2.1 画面呼出

項目	内容
概要	ライブラリ呼び出しを行う関数です。指定した X,Y 座標に指定した画面番号の画面 (ベース画面) を呼び出します。 拡張スクリプトでは使用できません。
書式	b_call (画面番号, X 座標, Y 座標) <div style="text-align: center;">  </div> <div style="border: 1px solid black; padding: 2px; width: fit-content; margin-top: 10px;">MEMO</div> <ul style="list-style-type: none"> 呼び出す画面の中央座標を X 座標、Y 座標で指定します。

座標位置



21.2.2 円

項目	内容
概要	<p>円の描画を指定アドレスに行います。「パターン」をチェックすると塗り込み円を描画します。線種（パターン選択時は塗り込みパターン）、色属性、中心座標、半径を設定します。また、中心座標、半径は、間接指定することができます。</p>
書式	<p>dsp_circle (X 座標, Y 座標, 半径, 表示カラープリnk + 表示カラー, 背景カラープリnk + 背景カラー, 線種)</p> <div data-bbox="578 981 985 1271" style="border: 1px solid gray; padding: 5px; margin: 10px 0;"> </div> <p>MEMO</p> <ul style="list-style-type: none"> 黒 + プリnkの設定をした場合、背景色は透明色になります。

21.2.3 ドット

項目	内容
概要	ドットの描画を指定アドレスに行います。X 座標、Y 座標、表示色を設定します。
書式	<p>dsp_dot (X 座標、Y 座標、ブリンク + 表示カラー)</p>  <p>MEMO</p> <ul style="list-style-type: none"> 黒 + ブリンクの設定をした場合、背景色は透明色になります。

21.2.4 直線

項目	内容
概要	直線の描画を指定アドレスに行います。線種、色属性、始点、終点座標を設定します。
書式	<p>dsp_line (始点 X 座標、始点 Y 座標、終点 X 座標、終点 Y 座標、表示カラーブリンク + 表示カラー、背景カラーブリンク + 背景カラー、線種および矢印)</p>  <p>MEMO</p> <ul style="list-style-type: none"> 黒 + ブリンクの設定をした場合、背景色は透明色になります。

21.2.5 四角

項目	内容
概要	<p>四角形の描画を指定アドレスに行います。「パターン」をチェックすると塗り込み四角形を描画します。 線種（パターン選択時は塗り込みパターン）、色属性、始点、終点座標を設定します。</p>
書式	<p>dsp_rectangle (始点 X 座標, 始点 Y 座標, 終点 X 座標, 終点 Y 座標, 表示カラープリンク + 表示カラー, 背景カラープリンク + 背景カラー, パターンおよび線種)</p> <div data-bbox="583 401 980 695" style="text-align: center;"> </div> <p>MEMO</p> <ul style="list-style-type: none"> 黒 + プリンクの設定をした場合、背景色は透明色になります。

重要

- 描画関数で色指定する場合、0 ~ 255 までのカラーコードで設定してください。E1 ~ E12 を設定するとスクリプト保存の際に、エラーとして出力されます。

21.3 メモリ操作

メモリ操作	動作概要
	オフセットアドレス ☞「21.3.1 オフセットアドレス」(21-8 ページ) アドレスのオフセットを指定します。
	メモリ比較 ☞「21.3.2 メモリ比較」(21-9 ページ) 2つのデバイスのメモリを比較し、結果を指定アドレスに格納します。
	メモリコピー ☞「21.3.3 メモリコピー」(21-11 ページ) デバイスのメモリを一括コピーします。
	メモリコピー (可変指定) ☞「21.3.4 メモリコピー (可変指定)」(21-14 ページ) デバイスのメモリを一括コピーします。コピー先アドレス、コピー元アドレス、アドレス数を任意に変更できます。
	メモリリング ☞「21.3.5 メモリリング」(21-15 ページ) 指定ワード数単位でリングシフトします。
	メモリ検索 ☞「21.3.6 メモリ検索」(21-17 ページ) ブロック単位で比較し、検索結果を指定アドレスに格納します。
	メモリ初期化 ☞「21.3.7 メモリ初期化」(21-20 ページ) デバイスを一括初期化します。
	メモリ初期化 (可変指定) ☞「21.3.8 メモリ初期化 (可変指定)」(21-21 ページ) デバイスを一括初期化します。先頭アドレス、セットデータ、アドレス数を任意に変更できます。
	メモリシフト ☞「21.3.9 メモリシフト」(21-22 ページ) ブロック単位で上位に移動します。

21.3.1 オフセットアドレス

項目	内容																							
概要	アドレスのオフセット指定が可能です。オフセットアドレスには、テンポラリアドレスのみ指定可能です。																							
書式	<p>[ワードアドレス]#[オフセットアドレス]</p>  <p>定数入力範囲</p> <table border="1"> <thead> <tr> <th rowspan="2">データ形式</th> <th colspan="2">定数入力</th> </tr> <tr> <th>最小値</th> <th>最大値</th> </tr> </thead> <tbody> <tr> <td>Bin16</td> <td>0</td> <td>65535</td> </tr> <tr> <td>Bin32</td> <td>0</td> <td>4294967295</td> </tr> <tr> <td>Bin16+/-</td> <td>-32768</td> <td>32767</td> </tr> <tr> <td>Bin32+/-</td> <td>-2147483648</td> <td>2147483647</td> </tr> <tr> <td>BCD16</td> <td>0</td> <td>9999</td> </tr> <tr> <td>BCD32</td> <td>0</td> <td>99999999</td> </tr> </tbody> </table>	データ形式	定数入力		最小値	最大値	Bin16	0	65535	Bin32	0	4294967295	Bin16+/-	-32768	32767	Bin32+/-	-2147483648	2147483647	BCD16	0	9999	BCD32	0	99999999
データ形式	定数入力																							
	最小値	最大値																						
Bin16	0	65535																						
Bin32	0	4294967295																						
Bin16+/-	-32768	32767																						
Bin32+/-	-2147483648	2147483647																						
BCD16	0	9999																						
BCD32	0	99999999																						

記述例 1

```
[w:[PLC1]D0200]=[w:[PLC1]D0100]#[t:0000]
```

上記の例は、[t:0000]の値が2とすると、D0102に格納されている値をD0200へ代入します。

記述例 2

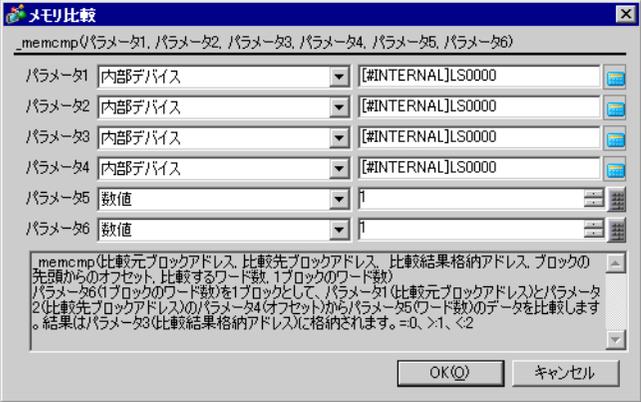
```
[w:[PLC1]D0100]#[t:0000]=30
```

上記の例は、[t:0000]の値が8とすると、30をD0108へ代入します。

重要

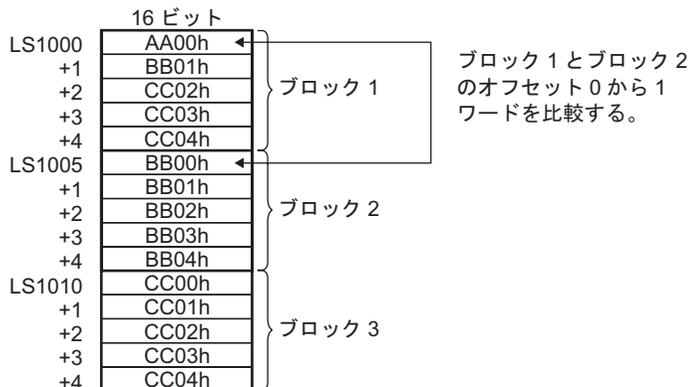
- オフセットアドレスの書式で使用するワードアドレスは、Dスクリプトのアドレス数のカウントには加算されません。
- オフセット指定されたデバイスの読み出しは、常時接続機器から読み出しは行わず、Dスクリプトの処理が実行されるたびに、その都度接続機器から読み出しが行われます。読み出しで通信エラーとなった場合には、値は0として処理されます。また、GP内部の特殊リレーLS2032のビット12がONします。正常にデータ読み出しが終了した場合には、ビット12はOFFします。
- 演算結果が16ビット（最大値：65535）を超えるような場合、15ビット目までを有効なビットとして扱い、16ビット目以上は切り捨てられます。

21.3.2 メモリ比較

項目	内容
概要	2つのブロックの任意の位置（オフセット）にあるデータを比較し、比較結果を格納アドレスに返します。 比較結果は、値が一致する場合は「0」、比較元より比較先の値が大きい場合は「1」、比較元より比較先の値が小さい場合は「2」が格納されます。エラーが発生した場合、LS9152 にエラーステータスを書き出します。
書式	<p>_memcmp (比較元ブロックアドレス, 比較先ブロックアドレス, 比較結果格納アドレス, ブロックの先頭からのオフセット, 比較するワード数, 1ブロックのワード数)</p>  <p>パラメータ 1： 内部デバイス パラメータ 2： 内部デバイス パラメータ 3： 内部デバイス パラメータ 4： 数値 (0 ~ 639)、内部デバイス、テンポラリ変数 パラメータ 5： 数値 (1 ~ 640) パラメータ 6： 数値 (1 ~ 640)</p> <p>格納されるデータ 0： 一致 1： 比較元 < 比較先 2： 比較元 > 比較先</p>

記述例 1

_memcmp ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1005], [w:[#INTERNAL]LS0100], 0, 1, 5)
 (ブロック 1 とブロック 2 のオフセット 0 から 1 ワードを比較し、比較結果を LS0100 に格納する場合)

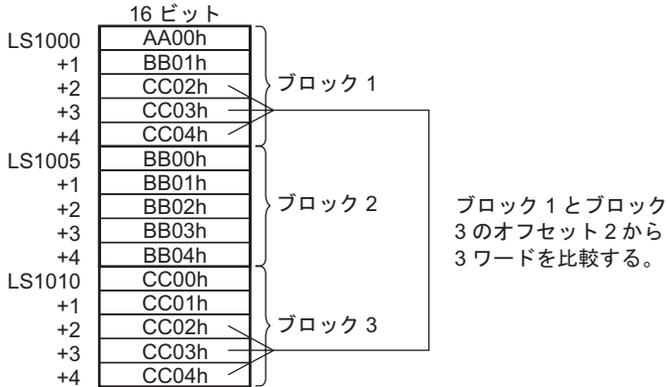


比較元の値が、比較先の値よりも小さいため、LS0100 に格納される比較結果は「2」となります。



記述例 2

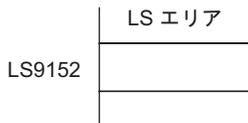
`_memcmp ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1010], [w:[#INTERNAL]LS0100], 2, 3, 5)`
 (ブロック 1 とブロック 3 のオフセット 2 から 3 ワードを比較し、比較結果を LS0100 に格納する場合)



比較元の値と、比較先の値が一致するため、LS0100 に格納される比較結果は「0」となります。



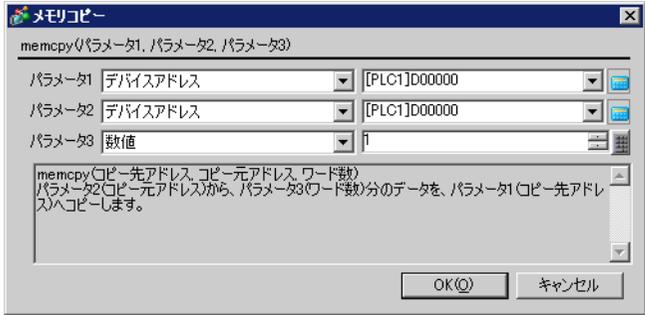
エラーステータス



エディタ関数名	LS エリア	エラーステータス	要因
<code>_memcmp ()</code>	LS9152	0000h	正常終了
		0001h	パラメータエラー
		0003h	書き込み、読み込みエラー

- 重要**
- 比較結果の格納アドレスを指定できる内部デバイスの有効範囲はユーザエリア (LS20 ~ LS2031、LS2096 ~ LS8191) のみです。
 - ブロック先頭からのオフセットに、1 ブロックのワード数を超える値が指定されると動作しません。
 - 比較するワード数、1 ブロックのワード数を超える値が指定されると動作しません。

21.3.3 メモリコピー

項目	内容
概要	デバイスのメモリを一括コピーします。コピー元アドレスからワード数分のデータをコピー先アドレスにコピーします。アドレス数は1～640までです。
書式	<p>memcpy (コピー先アドレス, コピー元アドレス, ワード数)</p> 

記述例

```
memcpy ([w:[PLC1]D0200], [w:[PLC1]D0100], 10)
```

上記の例は、D0100～D0109のデータがD0200～D0209にコピーされます。

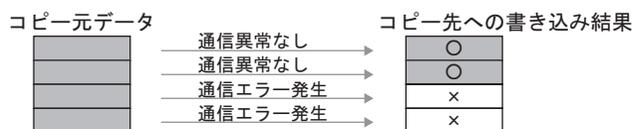
重要

- コピー元データの読み出しは、必要時に一度だけ接続機器からデータ読み出しを行います。データ読み出し時に通信エラーとなった場合には、GP内部の特殊リレーLS2032のビット12がONします。正常にデータ読み出しが終了した場合には、ビット12はOFFします。
- コピー元データの読み出し、コピー先へのデータ書き込みは、コピー元データのアドレス数により一括または分割で行われます。コピー元データの読み出し中に通信エラーが発生した場合、コピー先へのデータ書き込み結果は一括/分割で以下のとおり異なります。(コピー先の書き込み結果：書き込み完了、×:いっさい書き込みされません)

<一括メモリコピー>



<分割メモリコピー>



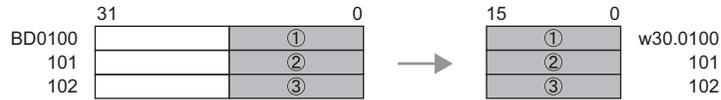
- アドレス数が多くなるに従って、それだけ PLC への書き込み時間が長くなり、かかる場合があります。アドレス数によっては、数十秒、数分以上かかる場合があります。
- 書き込みにおいて、デバイスの範囲外になった場合は通信エラーとなり、電源のON/OFFをしないと復旧することはできませんのでご注意ください。
- メモリコピー (memcpy) 関数で内部デバイスに書き込むときは、ユーザーエリアのみ書き込みます。システムエリア (LS0000～LS0019)、特殊エリア (LS2032～LS2047)、予約エリア (LS2048～LS2095) は書き込むことができません。ただし、読み出すことは可能です。

次のページに続きます。

重要

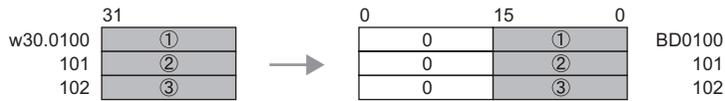
- D スクリプトのビット長の設定が 16 ビットの場合、32 ビットデバイス → 16 ビットデバイスにコピーしたときは、下位の 16 ビット分のデータのみがコピーされません。

例 : memcopy ([w:[PLC1]w30.0100], [w:[PLC1]BD0100], 3)



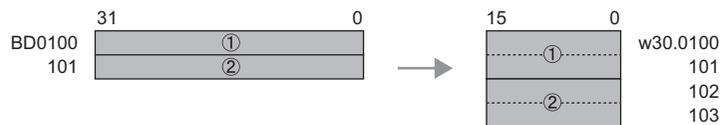
また、16 ビットデバイス → 32 ビットデバイスにコピーしたときは下位の 16 ビットにデータをコピーし、上位 16 ビットは 0 がセットされます。

例 : memcopy ([w:[PLC1]BD0100], [w:[PLC1]w30.0100], 3)

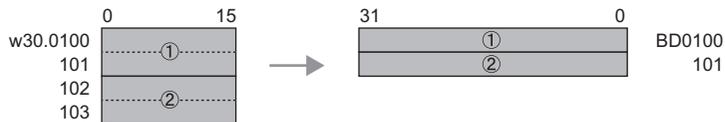


- D スクリプトのビット長の設定が 32 ビットの場合、32 ビットデバイス → 16 ビットデバイスにコピーしたとき、16 ビットデバイス → 32 ビットデバイスにコピーしたときは以下ようになります。また、片方が 32 ビットデバイスで片方が 16 ビットデバイスの場合、memcopy () のアドレス数の指定は 16 ビットデバイス側のアドレス数で指定してください。

例 : memcopy ([w:[PLC1]w30.0100], [w:[PLC1]BD0100], 4)



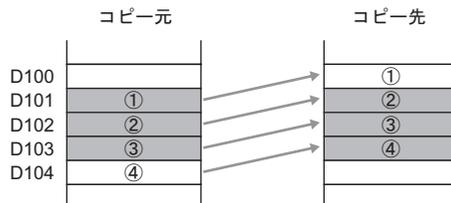
例 : memcopy ([w:[PLC1]BD0100], [w:[PLC1]w30.0100], 4)



- コピー元の範囲とコピー先の範囲が重なった場合、重なった部分のデータは以下のように書き替わります。

例 : D101 ~ D104 の 4 ワードを D100 ~ D103 にコピーする場合

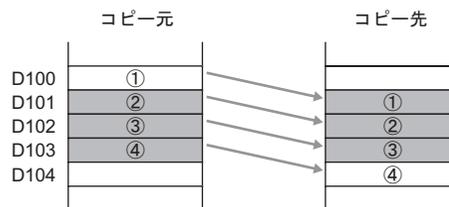
コピー先への書き込みは、前のアドレス (小さいアドレス) の方から行われます。



次のページに続きます。

重要

例：D100 ~ D103 の4ワードを D101 ~ D104 にコピーする場合
コピー先への書き込みは、後アドレス（大きいアドレス）の方から行われます。



- この関数ではアドレスを2つ指定していますが、Dスクリプトのアドレス数のカウントには加算されません。
- 代入にデバイスアドレスを使用する場合、接続機器との通信がありますので、すぐには書き込んだ値が代入されません。

21.3.4 メモリコピー（可変指定）

項目	内容
概要	デバイスのメモリを一括コピーします。パラメータ 2 で指定したコピー元アドレスから、パラメータ 3 で指定したワード数分のデータをパラメータ 1 で指定したコピー先アドレスにコピーします。 ワード数は 1 ~ 640 までです。この <code>_memcpy_EX</code> では、コピー元アドレス、コピー先アドレス、ワード数をそれぞれ間接的に指定することができます。
書式	<p><code>_memcpy_EX</code> (コピー先アドレス, コピー元アドレス, ワード数) パラメータ 1: デバイスアドレス + テンポラリアドレス パラメータ 2: デバイスアドレス + テンポラリアドレス パラメータ 3: 数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 1 ~ 640 です。</p> 

記述例

```
[t:0000]=10, [t:0001]=20
```

```
_memcpy_EX ([w:[#INTERNAL]LS0100]#[t:0000], [w:[PLC1]D0100]#[t:0001], 5)
```

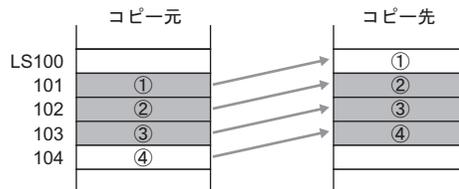
上記の例は、D0120 から 5 ワード分読み出して、LS0110 ~ LS0114 に書き込まれます。

重要

- コピー元の範囲とコピー先の範囲が重なった場合、重なった部分のデータは以下のように書き替わります。

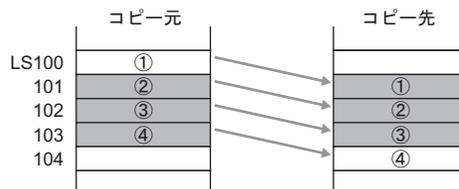
例：LS101 ~ LS104 の 4 ワードを LS100 ~ LS103 にコピーする場合

コピー先への書き込みは、前のアドレス（小さいアドレス）の方から行われます。



例：LS100 ~ LS103 の 4 ワードを LS101 ~ LS104 にコピーする場合

コピー先への書き込みは、後アドレス（大きいアドレス）の方から行われます。



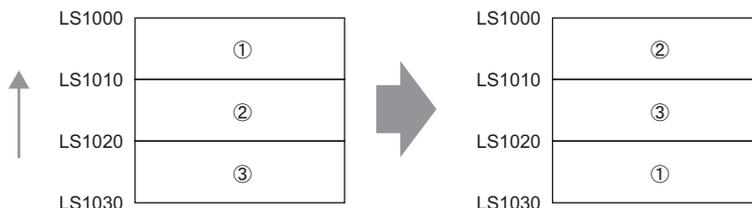
21.3.5 メモリリング

項目	内容
概要	メモリ上のデータをブロック単位でリングシフトします。 開始アドレス - 終了アドレス間で、ブロック単位（指定ワード数単位）のリングシフトを行います。エラーが発生した場合、LS9150 にエラーステータスを書き出します。
書式	<p>memring (開始アドレス, 終了アドレス, 1 ブロックのワード数)</p>  <p>パラメータ 1: 内部デバイス パラメータ 2: 内部デバイス パラメータ 3: 数値 (1 ~ 640)</p> <ul style="list-style-type: none"> ・パラメータ 1 < パラメータ 2 の場合、ブロックデータは上に移動する。 ・パラメータ 1 > パラメータ 2 の場合、ブロックデータは下に移動する。 <p>重要</p> <ul style="list-style-type: none"> ・開始ワードアドレスと終了ワードアドレスに指定するデバイスは、必ず種類 (LS もしくはUSR) を統一させてください。

記述例 1

```
memring ([w:#INTERNAL]LS1000], [w:#INTERNAL]LS1030], 10)
```

(パラメータ 1 < パラメータ 2 の場合)

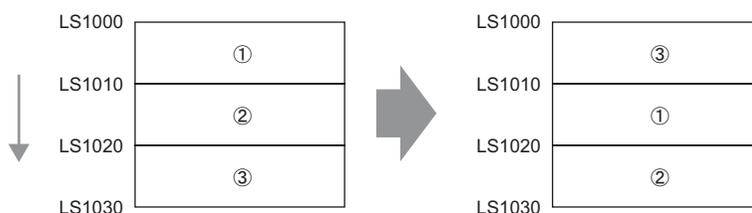


10 ワード単位で、データが上に移動します。

記述例 2

```
memring ([w:#INTERNAL]LS1030], [w:#INTERNAL]LS1000], 10)
```

(パラメータ 1 > パラメータ 2 の場合)

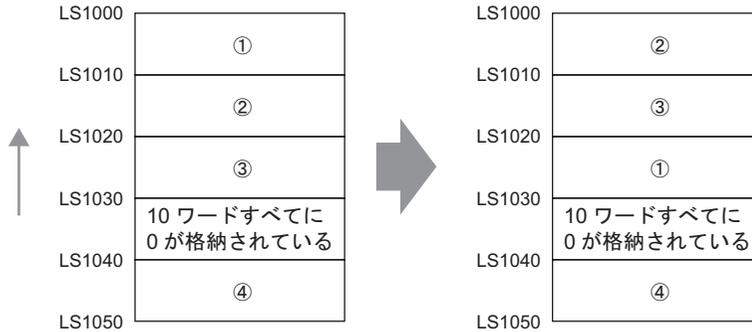


10 ワード単位で、データが下に移動します。

記述例 3

memring ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1050], 10)

(データがすべて0のブロックが範囲内に存在する場合)

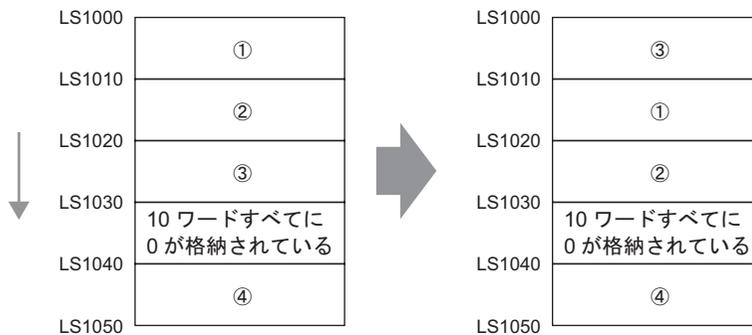


先頭ブロックからデータ0のブロックまでの範囲内だけで、ブロック（10ワード）単位でデータが上に移動します。データ0のブロック以降にデータが存在しても無視されます。

記述例 4

memring ([w:[#INTERNAL]LS1050], [w:[#INTERNAL]LS1000], 10)

(データ0のブロックが範囲内に存在する場合)



先頭ブロックからデータ0のブロックまでの範囲内だけで、ブロック（10ワード）単位でデータが下に移動します。データ0のブロック以降にデータが存在しても無視されます。

エラーステータス

	LS エリア
LS9150	

エディタ関数名	LS エリア	エラーステータス	要因
memring ()	LS9150	0000h	正常終了
		0001h	パラメータエラー
		0003h	書き込み、読み込みエラー

- 重要**
- 処理時間は、開始アドレスと終了アドレスで指定される範囲に比例します、広範囲を指定するほど処理時間がかかります。処理が完了するまで部品処理は更新されません。
 - 開始アドレス、終了アドレスで指定できる内部デバイスの有効範囲はユーザエリア (LS20 ~ LS2031、LS2096 ~ LS8191) のみです。

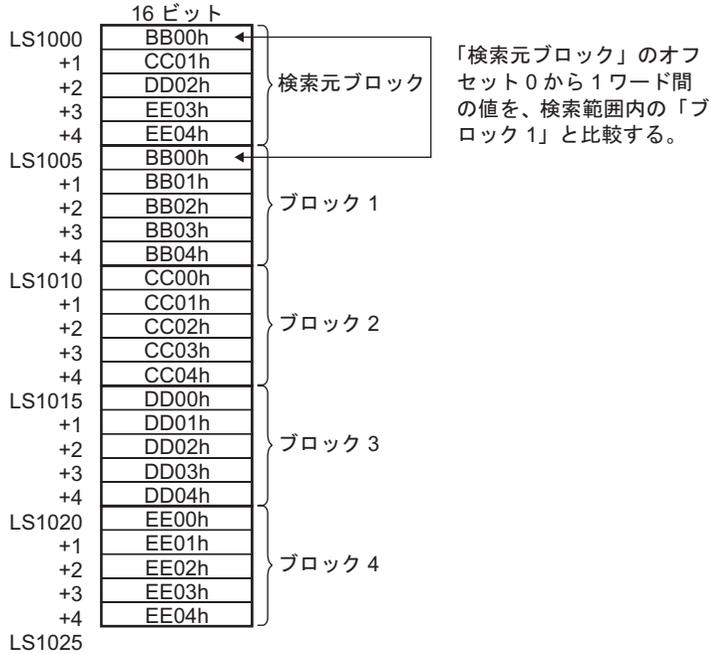
21.3.6 メモリ検索

項目	内容
概要	<p>指定された範囲からブロック単位でデータの検索を行います。ブロックの先頭から任意の位置（オフセット）にあるデータをブロック単位で比較し、検索結果を格納アドレスに返します。一致するブロックがある場合、ブロックのオフセット値（1～）が入り、一致するブロックがない場合、FFFFh が格納されます。エラーが発生した場合、LS9153 にエラーステータスを書き出します。</p>
書式	<p>_memsearch (検索元ブロックアドレス, 検索開始アドレス, 検索終了アドレス, 検索結果格納アドレス, 先頭ブロックからのオフセット, 比較するワード数, 1 ブロックのワード数)</p> <div data-bbox="463 484 1103 919" style="border: 1px solid gray; padding: 5px; margin: 10px 0;"> </div> <p>パラメータ 1: 内部デバイス パラメータ 2: 内部デバイス パラメータ 3: 内部デバイス パラメータ 4: 内部デバイス パラメータ 5: 数値 (0 ~ 639)、内部デバイス、テンポラリ変数 パラメータ 6: 数値 (1 ~ 640) パラメータ 7: 数値 (1 ~ 640)</p> <p>書込まれるデータ 一致するブロック有り: ブロックのオフセット値 (1 ~) 一致するブロックなし: FFFFh</p> <div style="border: 1px solid black; padding: 2px; margin: 10px 0;"> 重要 </div> <ul style="list-style-type: none"> 検索開始アドレスと検索終了アドレスに指定するデバイスは、必ず種類（LS もしくは USB）を統一させてください。ただし、「検索元ブロックアドレス」と「検索結果格納アドレス」は、内部デバイスを任意に設定できます。 必ず「パラメータ 2」 < 「パラメータ 3」で設定してください。エラーの原因になります。

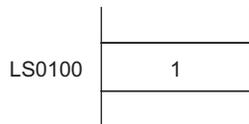
記述例 1

```
_memsearch ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1005], [w:[#INTERNAL]LS1025],
[w:[#INTERNAL]LS0100], 0, 1, 5)
```

(検索元ブロックのオフセット 0 から 1 ワード間と同じ値のブロックがないか、LS1005 から LS1025 間で検索し、結果を LS0100 に格納する場合)



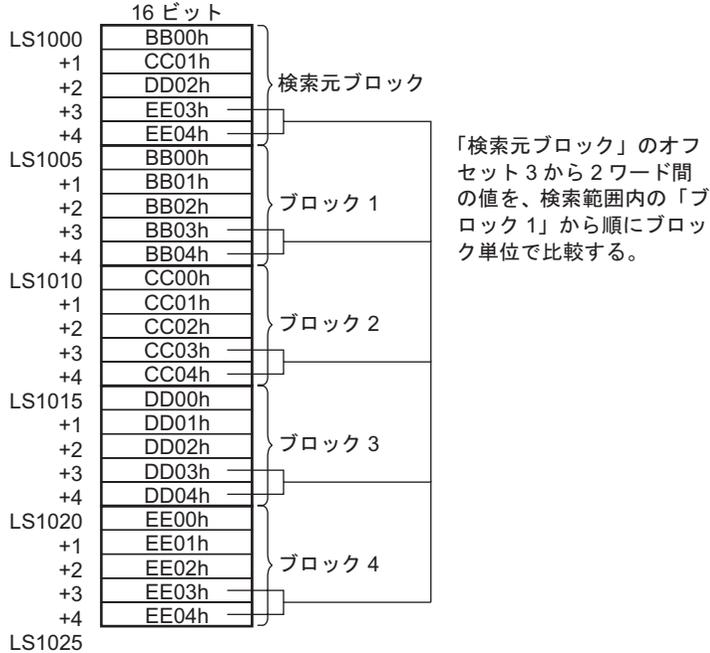
検索範囲先頭から検索した結果、「ブロック 1」の値が「検索元ブロック」の値と一致するため、LS0100 に格納される検索結果は「1」となります。



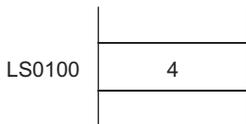
記述例 2

```
_memsearch ([w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS1005], [w:[#INTERNAL]LS1025],
[w:[#INTERNAL]LS0100], 3, 2, 5)
```

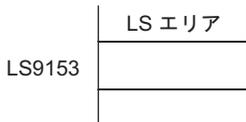
(検索元ブロックのオフセット 3 から 2 ワード間と同じ値のブロックがないか、LS1005 ~ LS1025 間で検索し、結果を LS0100 に格納する場合)



検索範囲先頭から検索した結果、「ブロック 4」の値が「検索元ブロック」の値と一致するため、LS0100 に格納される検索結果は「4」となります。



エラーステータス



エディタ関数名	LS エリア	エラーステータス	要因
_memserch ()	LS9153	0000h	正常終了
		0001h	パラメータエラー
		0003h	書き込み、読み込みエラー

- 重要**
- 処理時間は、開始アドレスと終了アドレスで指定される範囲に比例します、広範囲を指定するほど処理時間がかかります。処理が完了するまで部品処理は更新されません。
 - 開始アドレス、終了アドレスで指定できる内部デバイスの有効範囲はユーザエリア (LS20 ~ LS2031、LS2096 ~ LS8191) のみです。

21.3.7 メモリ初期化

項目	内容
概要	デバイスを一括初期化します。書き込み先アドレスからワード数分に書き込みデータをセットします。ワード数の範囲は、1 ~ 640 までです。
書式	<p>memset (書き込み先アドレス, 書き込みデータ, ワード数)</p> 

記述例

```
memset ([w:[PLC1]D0100], 0, 10)
```

上記の例は、D0100 ~ D0109 の全てのアドレスに 0 がセットされます。

重要

- アドレス数が多くなるに従って、それだけ PLC への書き込み時間が長くなりかかります。アドレス数によっては、数十秒、数分以上かかる場合があります。
- 書き込みにおいて、デバイスの範囲外になった場合は通信エラーとなり、電源の ON/OFF をしないと復旧することはできませんのでご注意ください。
- この関数ではアドレスを指定しますが、D スクリプトのアドレス数のカウントには加算されません。
- メモリ初期化 (memset) 関数で内部デバイスに書き込むときは、ユーザーエリアのみ書き込めます。システムエリア (LS0000 ~ LS0019) 特殊エリア (LS2032 ~ LS2047) 予約エリア (LS2048 ~ LS2095) は書き込むことができません。
- 代入にデバイスアドレスを使用する場合、PLC との通信がありますので、すぐには書き込まれた値が代入されません。

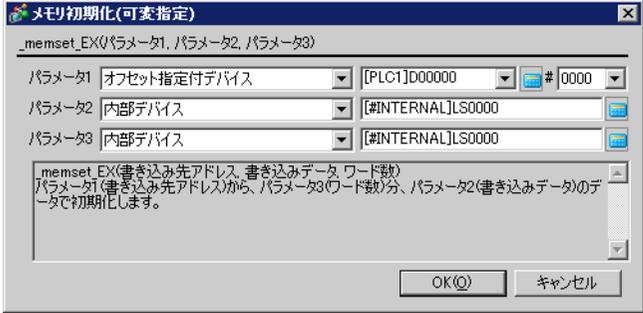
(例)

```
memset ([w:D0100], 0, 10) //D100 ~ D109 を 0 に初期化
```

```
[w:D200]=[w:D100] //D100 の内容を D200 に代入
```

この場合は、演算結果として D100 に書き込んだ 0 の値が、D200 にはまだ代入されていません。

21.3.8 メモリ初期化（可変指定）

項目	内容
概要	デバイスを一括初期化します。パラメータ 1 で指定した書き込み先アドレスからパラメータ 3 で指定したワード数分にパラメータ 2 で書き込みデータをセットします。ワード数の範囲は、1 ~ 640 までです。書き込み先アドレス、書き込みデータ、ワード数は、個々に間接的に指定することができます。
書式	<p>_memset_EX (書き込み先アドレス, 書き込みデータ, ワード数)</p>  <p>パラメータ 1 : デバイスアドレス + テンポラリアドレス パラメータ 2 : 数値、内部デバイス、テンポラリアドレス（パラメータ 2 に設定できる範囲は Dec : 0 ~ 65535、Hex : 0 ~ FFFF） パラメータ 3 : 数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 1 ~ 640 です。</p>

記述例

```
[t:0000]=10
```

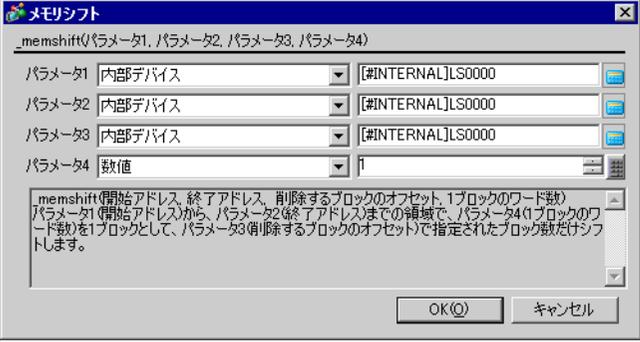
```
[w:LS0050]=0
```

```
[w:LS0051]=5
```

```
_memset_EX ([w:[#INTERNAL]LS0100]#[t:0000], [w:[#INTERNAL]LS0050], [w:[#INTERNAL]LS0051])
```

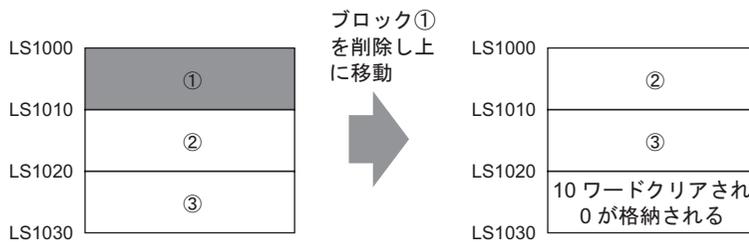
上記の例は、LS0110 から LS0114 の 5 ワード分に 0 を書き込まれます。

21.3.9 メモリシフト

項目	内容
概要	指定された 1 ブロックを削除し、以降のデータをブロック単位で上に移動します。削除するブロックの指定はオフセットで指定します。エラーが発生した場合、LS9151 にエラーステータスを書き出します。
書式	<p>_memshift (開始アドレス, 終了アドレス, 削除するブロックのオフセット, 1 ブロックのワード数)</p>  <p>パラメータ 1: 内部デバイス パラメータ 2: 内部デバイス パラメータ 3: 数値 (1 ~ 65535) 内部 デバイス、テンポラリ変数 パラメータ 4: 数値 (1 ~ 640)</p> <p>重要</p> <ul style="list-style-type: none"> 開始アドレスと終了アドレスに指定するデバイスは、必ず種類 (LS もしくは USR) を統一させてください。 必ず「パラメータ 1」 < 「パラメータ 2」で設定してください。エラーの原因になります。

記述例 1

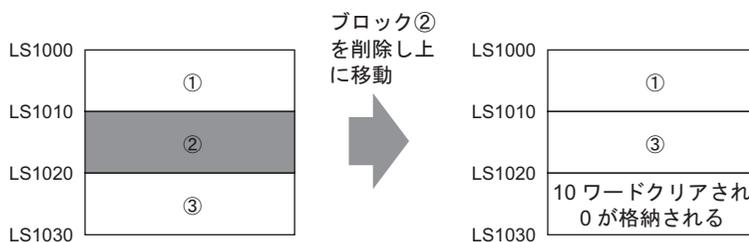
`_memshift ([w:#INTERNAL]LS1000), [w:#INTERNAL]LS1030], 1, 10)`



ブロック (10ワード) 単位でデータが上に移動し、最終ブロック (10ワード) が0クリアされます。

記述例 2

`_memshift ([w:#INTERNAL]LS1000), [w:#INTERNAL]LS1030], 2, 10)`



ブロックのオフセット 2 の位置から、ブロック (10ワード) 単位でデータが上に移動し、最終ブロック (10ワード) が0クリアされます。

エラーステータス

	LS エリア
LS9151	

エディタ関数名	LS エリア	エラーステータス	要因
<code>_memshift ()</code>	LS9151	0000h	正常終了
		0001h	パラメータエラー
		0003h	書き込み、読み込みエラー

重要

- 処理時間は、開始アドレスと終了アドレスで指定される範囲に比例します、広範囲を指定するほど処理時間がかかります。処理が完了するまで部品処理は更新されません。
- 削除するブロックのオフセットに、開始アドレスと終了アドレスに指定された範囲を超える値が指定されると、動作しません。
- 開始アドレス、終了アドレスで指定できる内部デバイスの有効範囲はユーザエリア (LS20 ~ LS2031、LS2096 ~ LS8191) のみです。

21.4 SIO ポート操作

SIO ポート操作	動作概要
	<p>ラベル設定</p> <p>☞「21.4.1 ラベル設定」(21-26 ページ)</p> <p>コントロール、ステータス、受信データ数、受信関数、送信関数から指定します。</p>
	<p>受信</p> <p>☞「21.4.2 受信」(21-30 ページ)</p> <p>指定のシリアルポート (COM1 または COM2) から受信データを読み込みます。</p>
	<p>送信</p> <p>☞「21.4.3 送信」(21-31 ページ)</p> <p>指定のシリアルポート (COM1 または COM2) へ書き込みを行います。</p>
	<p>拡張受信</p> <p>☞「21.4.4 拡張受信」(21-32 ページ)</p> <p>指定のシリアルポート (COM1 または COM2) から受信データを読み込みます。</p> <p>拡張スクリプトのみ使用できます。</p>
	<p>拡張送信</p> <p>☞「21.4.5 拡張送信」(21-33 ページ)</p> <p>指定のシリアルポート (COM1 または COM2) へ書き込みを行います。</p> <p>拡張スクリプトのみ使用できます。</p>
	<p>待ち受け受信関数</p> <p>☞「21.4.6 待ち受け受信関数」(21-34 ページ)</p> <p>指定文字列を受信するまで受信待ちになります。</p> <p>拡張スクリプトのみ使用できます。</p>
	<p>待機関数</p> <p>☞「21.4.7 待機関数」(21-35 ページ)</p> <p>指定した時間分、処理を待機 (ウェイト) します。</p> <p>拡張スクリプトのみ使用できます。</p>

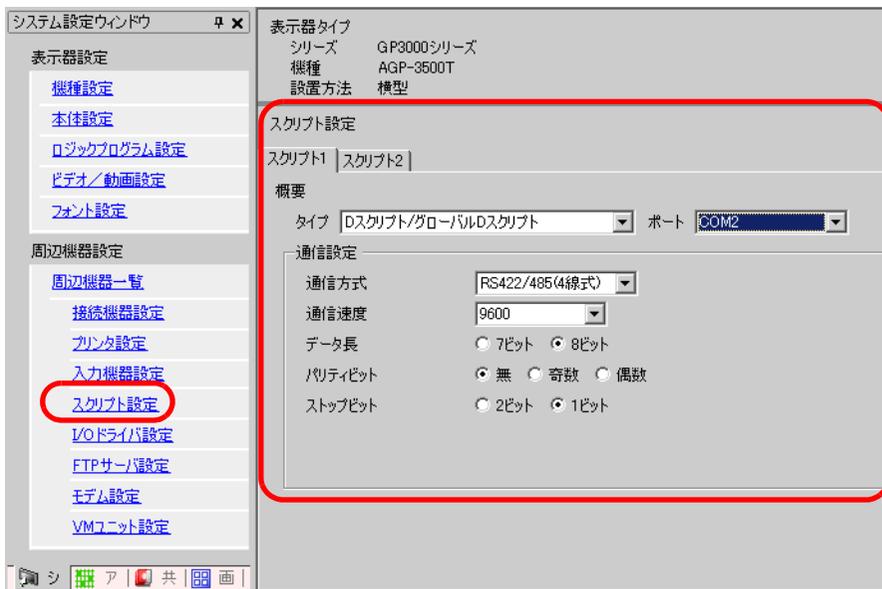
重要

- ラベル設定、送信、受信はD スクリプト / グローバルD スクリプトでも簡易的にできます。
- D スクリプト / グローバルD スクリプトで通信するために、各関数の設定と合わせて、以下のとおりスクリプト設定も必ず行ってください。スクリプト設定されない場合、実行できません。

【D スクリプト / グローバルD スクリプトのスクリプト設定】

システム設定ウィンドウの[スクリプト設定]をクリックします。

「タイプ」は「D スクリプト / グローバルD スクリプト」を必ず指定してください。



スクリプト設定では2 つタブがあります。上記では [スクリプト 1] を使用していません。

[ポート] は COM1 または COM2、[通信設定] の詳細は通信相手の外部機器に合わせて指定してください。

- SIO ポート操作を使用してより高度な通信プログラムを作成する場合、「拡張スクリプト」のご使用をお勧めします。拡張スクリプトを使用した通信例について、 「20.5 対応していない周辺機器と通信させたい」(20-21 ページ)

21.4.1 ラベル設定

コントロール

ビット指定の場合 [c:EXT_SIO_CTRL**] (書き込みのみ有効)

ワード指定の場合 [c:EXT_SIO_CTRL] (書き込みのみ有効)

ステータス

ビット指定の場合 [s:EXT_SIO_STAT**] (読み込みのみ有効)

ワード指定の場合 [s:EXT_SIO_STAT] (読み込みのみ有効)

受信データ数

[r:EXT_SIO_RCV] (読み込みのみ有効)

受信関数

IO_READ ([p:EXT_SIO], 内部格納アドレス, 転送バイト数)

送信関数

IO_WRITE ([p:EXT_SIO], 内部格納アドレス, 転送バイト数)

コントロール

項目	内容
概要	送信バッファ、受信バッファ、エラーステータスのクリアを行うためのコントロール変数です。このコントロール変数は、書き込みのみ有効です。
書式	ビット指定の場合 [c:EXT_SIO_CTRL**] (**:00 ~ 15) ワード指定の場合 [c:EXT_SIO_CTRL]

記述例

ビット指定の場合 [c:EXT_SIO_CTRL00] = 1

ワード指定の場合 [c:EXT_SIO_CTRL] = 0x0007

EXT_SIO_CTRL の内容

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ビット	内容
15	
14	
13	
12	
11	
10	
9	予約
8	
7	
6	
5	
4	
3	1: 受信タイムアウトクリア
2	1: エラークリア
1	1: 受信バッファクリア
0	1: 送信バッファクリア

MEMO

- ・ワード指定の場合（複数ビットを同時にセットした場合）処理する順は以下の通りです。
エラークリア → 受信バッファクリア → 送信バッファクリア

ステータス

項目	内容
概要	ステータスの情報としては、以下のものがあります。 このステータス変数は、読み込みのみ有効です。
書式	ビット指定の場合 [s:EXT_SIO_STAT**] (**:00 ~ 15) ワード指定の場合 [s:EXT_SIO_STAT]

記述例

ビット指定の場合 `if ([s:EXT_SIO_STAT00] == 1)`

ワード指定の場合 `if (([s:EXT_SIO_STAT] & 0x0001) <> 0)`

EXT_SIO_STAT の内容

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ビット	内容	
15	0: D スクリプト / グローバル D スクリプト無し 1: D スクリプト / グローバル D スクリプト有り	
14	0: 拡張スクリプト無し 1: 拡張スクリプト有り	
13	予約	
12		
11		
10		
9		
8		
7		
6	予約	
5		
4		0: 正常 1: 受信タイムアウト
3		0: 正常 1: 受信エラー
2		0: 受信データ無し 1: 受信データ有り
1		0: 正常 1: 送信エラー
0		0: 送信バッファにデータ有り 1: 送信バッファエンpty

MEMO

- 予約ビットは将来使用する可能性がありますので、必要なビットのみをチェックするようにして下さい。
- 送信エラーには送信タイムアウトエラーと送信バッファフルエラーがあり、どちらかのエラーが発生すれば、送信エラーのビットが ON します。送信タイムアウト時間は 5 秒です。
- 受信エラーにはパリティエラー、オーバーランエラー、フレンジエラー、オーバーフローがあります。このうちいずれかのエラーが発生すれば、受信エラーのビットが ON します。
- 送信エラーを検出した場合、送信データは送信バッファに溜まったままになります。また、送信エラーを検出できない場合、送信データは送信バッファに溜まったままにならず、送信されます。
- シリアルインタフェース COM2 使用時は、COM2 が RS-422 であるため、CS(CTS) 信号を検出できません。このため、シリアルケーブル抜け等が検出できません。

受信データ数

項目	内容
概要	その時点の受信しているデータ数 (バイト数) がわかります。また、受信データ数は、読み込みのみ有効です。
書式	[r:EXT_SIO_RECV]

重要

- 受信データ数（バイト数）のラベル名について
GP-PRO/PB V6.0 以前で設定されたラベル名は [r:EXT_SIO_RCV] でしたが、
[r:EXT_SIO_RCV]、[r:EXT_SIO_RECV] のどちらの記述でも同様の動作になります
ので、修正する必要はありません。
-

21.4.2 受信

項目	内容
概要	外部機器から受信データを読み込む場合に以下のように記述します。
書式	<p>IO_READ (SIO ポート名, データ格納アドレス, 受信バイト数)</p>  <p>パラメータ 1 : EXT_SIO パラメータ 2 : 内部デバイス パラメータ 3 : 数値</p>

記述例

IO_READ ([p:EXT_SIO], [w:[#INTERNAL]LS0100], 10)

上記の例は、LS0100 に受信データ数が格納され、LS0101 から 10 バイト分の受信データが格納されます。下記に受信データ格納イメージ図を示します。

MEMO

- 受信時の最大転送バイト数は 2011 バイトです。各ワードアドレスに 1 バイト単位でデータが書き込まれます。

LS0100	受信データ数	… 10 バイト
LS0101	00	バイト 1
LS0102	00	バイト 2
LS0103	00	バイト 3
LS0104	00	バイト 4
LS0105	00	バイト 5
LS0106	00	バイト 6
LS0107	00	バイト 7
LS0108	00	バイト 8
LS0109	00	バイト 9
LS0110	00	バイト 10

受信データ格納イメージ図

21.4.3 送信

項目	内容
概要	外部機器に対して、書き込みを行う場合に以下のように記述します。
書式	<p>IO_WRITE (p:SIO ポート名, データ格納アドレス, 送信バイト数)</p>  <p>パラメータ 1 : EXT_SIO パラメータ 2 : 内部デバイス パラメータ 3 : 数値</p>

記述例

IO_WRITE ([p:EXT_SIO], [w:[#INTERNAL]LS0100], 10)

上記の例は、LS0100 から 10 バイト分のデータを送信します。下記に送信データ格納イメージ図を示します。

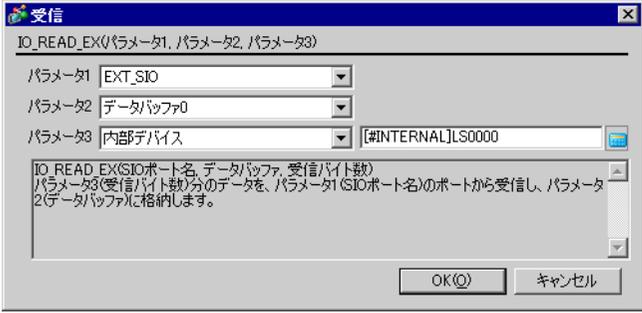
MEMO

- 送信時の最大転送バイト数は 2012 バイトです。
- 送信バッファ用の内部デバイスには、各ワードアドレスに 1 バイト単位のデータを書き込んで下さい。

LS0100	00	バイト 1
LS0101	00	バイト 2
LS0102	00	バイト 3
LS0103	00	バイト 4
LS0104	00	バイト 5
LS0105	00	バイト 6
LS0106	00	バイト 7
LS0107	00	バイト 8
LS0108	00	バイト 9
LS0109	00	バイト 10

送信データ格納イメージ図

21.4.4 拡張受信

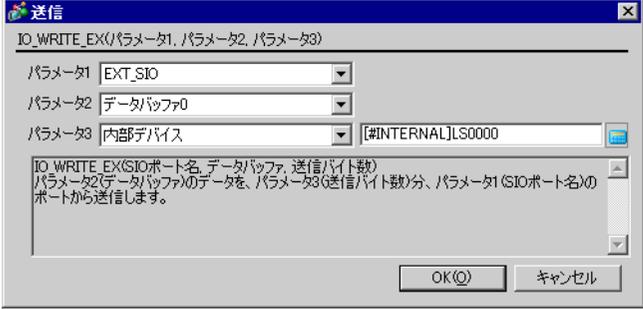
項目	内容
概要	受信バイト数分のデータを外部機器から受信して、データバッファに格納します。パラメータ 3 で指定したバイト数分、外部機器から受信して、パラメータ 2 で指定したデータバッファに格納します。 拡張スクリプトのみ使用できます。
書式	<p>IO_READ_EX (SIO ポート名, データバッファ, 受信バイト数)</p>  <p>パラメータ 1 : [p:EXT_SIO] パラメータ 2 : データバッファ パラメータ 3 : 数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 1 ~ 1024 です。</p>

記述例

```
IO_READ_EX ([p:EXT_SIO], databuf1, 10)
```

上記の例では、外部機器で受信したデータから 10 バイト分のデータを受信して、databuf1 に格納します。

21.4.5 拡張送信

項目	内容
概要	データバッファのデータを送信バイト数分、外部機器から送信します。パラメータ 2 で指定したデータバッファの内容をパラメータ 3 で指定した長さ分、外部機器から送信します。 拡張スクリプトのみ使用できます。
書式	<p>IO_WRITE_EX (SIO ポート名, データバッファ, 送信バイト数)</p>  <p>パラメータ 1 : [p:EXT_SIO] パラメータ 2 : データバッファ パラメータ 3 : 数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 1 ~ 1024 です。</p>

記述例

```
IO_WRITE_EX ([p:EXT_SIO], databuf0, 10)
```

上記の例では、databuf0 のデータを 10 バイト分、外部機器から送信します。

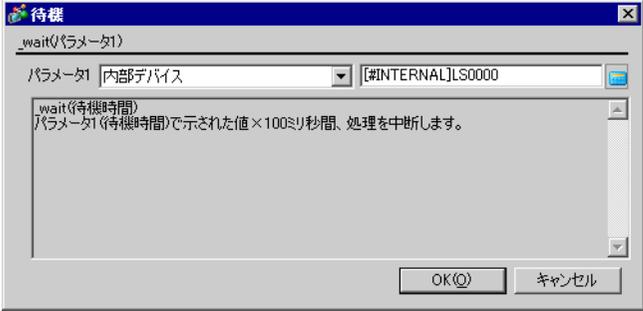
21.4.6 待ち受け受信関数

項目	内容
概要	<p>指定文字列を受信するまで受信待ちになります。タイムアウト時間が経過した場合、ステータス [s :EXT_SIO_STAT] のビット 4 (受信タイムアウトエラー) がセットされます。タイムアウト時間単位は 100msec です。</p> <p>パラメータ 2 で指定した文字列または文字コードを受信するまで受信待ちになります。パラメータ 3 には、タイムアウト時間を設定します。</p> <p>拡張スクリプトのみ使用できます。</p>
書式	<p>IO_READ_WAIT (SIO ポート名, 文字列, タイムアウト時間)</p> <div data-bbox="459 473 1104 782" style="border: 1px solid gray; padding: 5px; margin: 10px 0;"> </div> <p>パラメータ 1 : [p:EXT_IO] パラメータ 2 : 数値、文字列、データバッファ パラメータ 3 : 数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 1 ~ 600 です。</p>

重要

- 指定した文字列を受信するまでに、受信したデータは使用することができません。(破棄されます。)
- 指定する文字列は最大 128 文字 (バイト) です。これ以上の文字列を指定した場合は、正しく受信待ちが行えませんのでご注意ください。

21.4.7 待機関数

項目	内容
概要	指定した時間分、処理を待機（ウェイト）します。単位は 100ms です。拡張スクリプトのみ使用できます。
書式	<p data-bbox="310 305 495 334">_wait(待機時間)</p> <div data-bbox="460 355 1103 666" style="border: 1px solid gray; padding: 5px; margin: 10px auto; width: fit-content;">  </div> <p data-bbox="310 693 1243 753">パラメータ 1： 内部デバイス、テンポラリアドレス、数値 パラメータ 1 に設定できる範囲は 1 ~ 600 です。</p>

記述例

```
_wait(10)
```

上記の例では、1 秒間待機（ウェイト）します。

21.5 CF ファイル操作

CF ファイル操作	動作概要
	ラベル設定 ☞「21.5.1 ラベル設定」(21-37 ページ) ファイルリスト数、読み出しバイト数、CF カードエラーステータスから指定します。
	ファイルライト ☞「21.5.2 ファイルライト」(21-44 ページ) 読み出し先アドレスから指定バイト数分の内容を指定ファイルに書き込みます。
	ファイル名変更 ☞「21.5.3 ファイル名変更」(21-47 ページ) ファイル名を変更します。
	CSV ファイルリード ☞「21.5.4 CSV ファイルリード」(21-49 ページ) CSV ファイルからセル単位で読み込み、ワードアドレスに書き込みます。
	ファイルリード ☞「21.5.5 ファイルリード」(21-51 ページ) ファイルの内容をオフセットから指定バイト数分、書き込み先アドレスに書き込みます。
	ファイルリスト出力 ☞「21.5.6 ファイルリスト出力」(21-53 ページ) 指定したフォルダに存在するファイルのリストを内部デバイスに書き込みます。
	ファイル削除 ☞「21.5.7 ファイル削除」(21-54 ページ) ファイルを削除します。

21.5.1 ラベル設定

CF カードステータスには、以下のステータスがあります。

ステータス名	ラベル名	内容
ファイルリスト数	[s:CF_FILELIST_NUM]	ファイルリスト出力関数 <code>_CF_dir()</code> を実行した時に実際に存在したファイルリストの数を格納します。
読み出しバイト数	[s:CF_READ_NUM]	ファイルリード関数 <code>_CF_read()</code> を実行した時に実際に読み出せたバイト数を格納します。
CF カードエラーステータス	[s:CF_ERR_STAT]	CF カードアクセス時に発生するエラーのステータスを格納します。

ファイルリスト数

ファイルリスト出力関数 `_CF_dir()` を実行した時に実際に内部デバイスに書き込んだファイルリストの数を「ファイルリスト数 [s:CF_FILELIST_NUM]」に格納します。

使用例

```
_CF_dir ("%DATA¥*.*", [w:[#INTERNAL]LS0100], 10, 0)
[w:LS0200] = [s:CF_FILELIST_NUM]
```

```
¥DATA ─┬─ DATA0000.BIN
        ├── DATA0001.BIN
        ├── DATA02.BIN
        ├── DATA003.BIN
        └─ DATA0004.BIN
```

10 ファイル分のファイルリストを得ようとしたが、フォルダにファイルが 5 つしかなかった場合には、[s:CF_FILELIST_NUM] に 5 が格納されます。

重要

- ファイルリスト出力関数のパラメータ 3 (ファイル名数) が 0 の時はフォルダ内のファイル総数を [s:CF_FILELIST_NUM] に書き込みます。

読み出しバイト数

ファイルリード関数 `_CF_read()` を実行した時に実際に読み出せたバイト数を「読み出しバイト数 [s:CF_READ_NUM]」に格納します。

使用例

```
_CF_read ("%DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 16, 16)
[w:[#INTERNAL]LS0200] = [s:CF_READ_NUM]
```

16 バイト読み出そうとしたが 12 バイトしかデータを読み出せなかった場合には、[s:CF_READ_NUM] に 12 が格納されます。

CF カードエラーステータス

CF カードアクセス時に発生するエラーのステータスを格納するステータスです。

ビット位置	エラー名	内容
15	予約	予約
14		
13		
12		
11		
10		
9		
8		
7		
6	ファイルリネームエラー	<ul style="list-style-type: none"> • 実行中に CF カードを抜き取った • 指定したファイルが存在しなかった • リードオンリー属性のファイルをリネームしようとした
5	ファイル削除エラー	<ul style="list-style-type: none"> • 実行中に CF カードを抜き取った • 指定したファイルが存在しなかった • リードオンリー属性のファイルを削除しようとした
4	ファイルライトエラー	<ul style="list-style-type: none"> • 実行中に CF カードを抜き取った • CF カードの空き容量がなかった • リードオンリー属性のファイルに書き込もうとした • 格納方法が「上書き」の場合に指定ファイルが存在していない
3	ファイルリードエラー	<ul style="list-style-type: none"> • 実行中に CF カードを抜き取った • 指定したファイルが存在しなかった
2	ファイルリストエラー	<ul style="list-style-type: none"> • 実行中に CF カードを抜き取った • 指定したフォルダが存在しなかった
1	CF カードエラー	<ul style="list-style-type: none"> • CF カードが異常 • CF カードでないカードが挿入されている
0	CF カード無し	<ul style="list-style-type: none"> • CF カードが挿入されていないか • ハッチがオープンしている

- CF カードのエラーが発生した場合でも処理はそのまま続行されますので、必ず CF カードのファイル操作関数を使用したときには、エラーを確認するスクリプトを記述してください。

例

```

_CF_dir ("%DATA%*.*", [w:[#INTERNAL]LS0100], 2, 1) // ファイルリスト出力
if ([s:CF_ERR_STAT02] <> 0) // エラーステータスの確認
{
    set ([b:[#INTERNAL]LS005000]) // エラー表示用のビットアドレスをセット
}
endif

```

CF カードエラー詳細ステータス 格納エリア

エラーが発生した際に各ビットがセットされますが、どのような要因でエラーが発生したか、詳細ステータスをするにより確認できます。それぞれの関数において、拡張システムエリアのLS9132 ~ LS9137 に詳細ステータスが格納されます。これらのエリアは、読み込み専用です。

LS エリア	
LS0000	
:	
LS9132	CF リストステータス
LS9133	CF リードステータス
LS9134	CF ライトステータス
LS9135	CF デリートステータス
LS9136	CF リネームステータス
LS9137	CSV リードステータス
:	
LS9999	

各関数エラー詳細リスト

エディタ関数名		エラーステータス	要因
_CF_dir ()	LS9132	0010h	D スクリプトデータ異常 (固定文字列のフォルダ名取り出しエラー)
		0012h	ファイル名 (パス名) エラー
		0018h	LS エリア書き込み範囲エラー
		0020h	CF カード無し
		0021h	CF カード異常
		0100h	ディレクトリオープンエラー
_CF_read ()	LS9133	0010h	D スクリプトデータ異常 (固定文字列のフォルダ名、ファイル名取り出しエラー)
		0011h	LS エリア読み込み範囲エラー
		0012h	ファイル名 (パス名) エラー
		0018h	LS エリア書き込み範囲エラー
		0020h	CF カード無し
		0021h	CF カード異常
		0101h	ファイルシークエラー (オフセットエラー)
		0102h	読み出しバイト数エラー
		0110h	ファイル作成 (オープン) エラー

次のページに続きます。

エディタ関数名		エラーステータス	要因
_CF_write ()	LS9134	0010h	D スクリプトデータ異常 (固定文字列のフォルダ名、ファイル名取り出しエラー)
		0011h	LS エリア読み込み範囲エラー
		0012h	ファイル名 (パス名) エラー
		0020h	CF カード無し
		0021h	CF カード異常
		0101h	ファイルシークエラー (オフセットエラー)
		0104h	フォルダ作成エラー
		0108h	書き込みモードエラー
		0110h	ファイル作成 (オープン) エラー
		0111h	ファイルライトエラー (CF カードの容量が足りないなど)
_CF_delete ()	LS9135	0010h	D スクリプトデータ異常 (固定文字列のフォルダ名、ファイル名取り出しエラー)
		0011h	LS エリア読み込み範囲エラー
		0012h	ファイル名 (パス名) エラー
		0020h	CF カード無し
		0021h	CF カード異常
		0112h	ファイル削除エラー (ファイルが存在しない、ReadOnly ファイルであるなど)
_CF_rename ()	LS9136	0010h	D スクリプトデータ異常 (固定文字列のフォルダ名、ファイル名取り出しエラー)
		0011h	LS エリア読み込み範囲エラー
		0012h	ファイル名 (パス名) エラー
		0020h	CF カード無し
		0021h	CF カード異常
		0114h	ファイルリネームエラー (ファイルが存在しない、ReadOnly ファイルである、同一ファイル名が存在しているなど)
_CF_read_csv ()	LS9137	0001h	パラメータエラー
		0002h	CF カードエラー (CF カード無し、ファイルオープンエラー、ファイルリードエラー)
		0003h	書き込みエラー

データ格納モード

ファイルリード、ファイルライト関数実行時にデバイスアドレスに書き込む場合や、読み出す場合に、書き込む（読み出す）格納順序を設定します。

LS9130 にデータ格納モードを設定することで格納順序を変更することが可能です。モードは 0, 1, 2, 3 の 4 通りがあります。

モード 0

例：ファイルリード関数を使用してデバイスアドレスに文字列 ABCDEFG が書き込まれた場合

```
[w:[#INTERNAL]LS9130] = 0
```

```
_CF_read ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)
```

- デバイスアドレスが 16 ビット長の場合

LS0100	'A'	'B'
LS0101	'C'	'D'
LS0102	'E'	'F'
LS0103	'G'	0

格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'A'	'B'	'C'	'D'
LS0101	'E'	'F'	'G'	0
LS0102

格納するデータ数のあまりバイトに0が書き込まれます。

モード 1

例：ファイルリード関数を使用してデバイスアドレスに文字列 ABCDEFG を書き込む場合

```
[w:[#INTERNAL]LS9130] = 1
```

```
_CF_read ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)
```

- デバイスアドレスが 16 ビット長の場合

LS0100	'B'	'A'
LS0101	'D'	'C'
LS0102	'F'	'E'
LS0103	0	'G'

格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'B'	'A'	'D'	'C'
LS0101	'F'	'E'	0	'G'
LS0102

格納するデータ数のあまりバイトに0が書き込まれます。

モード 2

例：ファイルリード関数を使用してデバイスアドレスに文字列 ABCDEFG を書き込む場合

[w:[#INTERNAL]LS9130] = 2

_CF_read ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)

- デバイスアドレスが 16 ビット長の場合

LS0100	'C'	'D'
LS0101	'A'	'B'
LS0102	'G'	0
LS0103	'E'	'F'

格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'C'	'D'	'A'	'B'
LS0101	0	'G'	'E'	'F'
LS0102				

格納するデータ数のあまりバイトに0が書き込まれます。

モード 3

例：ファイルリード関数を使用してデバイスアドレスに文字列 ABCDEFG を書き込む場合

[w:[#INTERNAL]LS9130] = 3

_CF_read ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 7)

- デバイスアドレスが 16 ビット長の場合

LS0100	'D'	'C'
LS0101	'B'	'A'
LS0102	0	'G'
LS0103	'F'	'E'

格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'D'	'C'	'B'	'A'
LS0101	0	'G'	'F'	'E'
LS0102				

格納するデータ数のあまりバイトに0が書き込まれます。

重要

- データ格納モードとシステムの設定にある文字列データモードとは一致していません。文字列格納モードとの対比は以下のようになります。

データのデバイス格納順序	ワード内のバイト LH/HL 格納順序	ダブルワード内のバイト LH/HL 格納順序	D スクリプトデータ格納モード	文字列データモード
先頭データから格納	HL 順	HL 順	0	1
	LH 順		1	2
	HL 順	LH 順	2	5
	LH 順		3	4
最終データから格納	HL 順	HL 順	-	3
	LH 順		-	7
	HL 順	LH 順	-	8
	LH 順		-	6

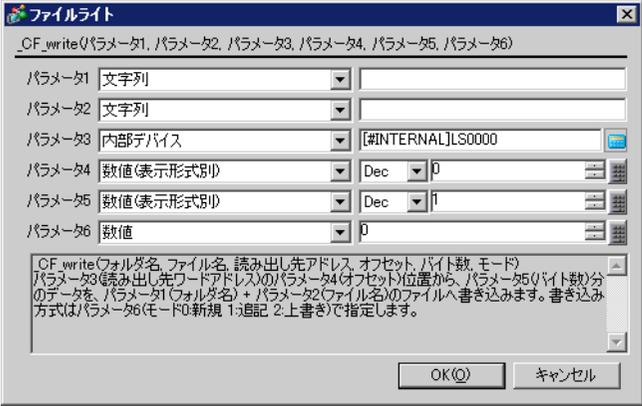
重要

- CF カードにはデータの書き換え回数に制限があります。必ず他の記録媒体にバックアップをとってください。(500K バイトの DOS 形式のデータの書き換えで、約 10 万回)
- CF カード処理中にエラーが発生した場合は、CF カードエラーステータス [s:CF_ERR_STAT] が書きこまれます。詳細については、
☞ 「 CF カードエラーステータス」(21-38 ページ)
- 以下の記号文字はフォルダ名、ファイル名として指定することはできません。使用した場合エラーとなります。

:	,	=	+	/	"	[
]		<	>	(スペース)	?	

- ルートフォルダ (ディレクトリ) を指定する場合には、フォルダ名に "" (空文字列) を指定してください。

21.5.2 ファイルライト

項目	内容
概要	読み出し先アドレスから指定バイト数分の内容を指定ファイルに書き込みます。データの格納方法（モード）は、下表のように「新規」、「追記」、「上書き」があり、格納順序は後述する「データ格納モード」を参照してください。
書式	<p>_CF_write (フォルダ名, ファイル名, 読み出し先アドレス, オフセット, バイト数, モード)</p>  <p>パラメータ 1 フォルダ名：固定文字列（最大文字数は、半角 32 文字）</p> <p>パラメータ 2 ファイル名：固定文字列、内部デバイス（最大文字数は半角 32 文字）、内部デバイス + テンポラリアドレス</p> <p>パラメータ 3 読み出し先アドレス：デバイスアドレス、デバイスアドレス + テンポラリアドレス</p> <p>パラメータ 4 オフセット：数値、デバイスアドレス、テンポラリアドレス（指定最大数は 16 ビット長の時 65535、32 ビット長の時 4294967295）</p> <p>パラメータ 5 バイト数：数値、デバイスアドレス、テンポラリアドレス（指定最大数は 1280）</p> <p>パラメータ 6 モード：数値、デバイスアドレス、テンポラリアドレス（値は 0,1,2）</p>

格納方法（モード）の概要

モード	名称	内容
0	新規	ファイルを新規に作成します。ファイルが存在していれば、古いファイルを削除します。
1	追記	追記書き込みを行います。ファイルが存在していなければ新たに作成します。
2	上書き	ファイルの一部を上書きで書き換えます。オフセットをファイルサイズより大きくした場合は、超えた分を 0 で埋めて、その後にデータを書き込みます。オフセットをファイルの最後に指定すると追記書き込みとなります。ファイルが存在していなければエラーとなります。エラーの詳細については、「CF カードエラーステータス」(21-38 ページ)を参照してください。

記述例

```
[w:[#INTERNAL]LS0200] = 0 // オフセット (モードが新規の場合は、“0” 固定です。)
[w:[#INTERNAL]LS0202] = 100 // バイト数 (100 バイト)
[w:[#INTERNAL]LS0204] = 0 // モード (新規)
_CF_write ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], [w:[#INTERNAL]LS0200],
[w:[#INTERNAL]LS0202], [w:[#INTERNAL]:LS0204])
```

上記の例は、LS0100 から 100 バイト読み出したデータを ¥DATA フォルダに DATA0001.BIN ファイルを新規に作成します。オフセット、バイト数、モードに内部デバイスを指定することにより間接的にバイト数、モードを指定できます。

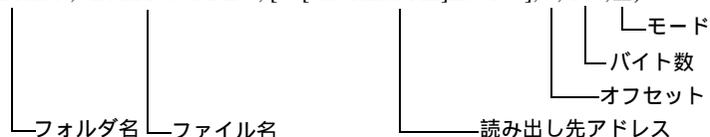
重要

- モードが「上書き」の時のみ、オフセットが有効です。「新規」、「追記」ではオフセットは無効です。「上書き」以外の場合は、オフセットの値を 0 にしてください。
- モードを新規に指定したときに、すでにファイルが存在しているときはそのファイルを上書きします。
- 「ファイル名」に内部デバイスを指定した場合、また「読み出し先アドレス」は、D スクリプトのアドレス数には加算されません。
- 読み出し先アドレスに PLC デバイスを指定した場合、関数を実行したときに一度だけ PLC からデータを読み出します。データ読み出し時にエラーとなった場合には、CF カードエラーステータス [s:CF_ERR_STAT] にエラーがセットされます。正常に読み出しが終了した場合には、エラーはクリアされます。
- 読み出すバイト数にもよりますが、分割しながらデータを読み出しますので、データの読み出し途中で通信エラーが発生した場合には、途中までのデータがファイルに書き込まれます。
- ファイル名にフルパスを指定する場合は、フォルダ名に "*" (アスタリスク) を指定してください。
例：_CF_read ("*", "¥DATA¥DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 10)

格納方法 (モード) の記述例

モードを「新規」にしたとき

```
_CF_write ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 100, 0)
```



上記式を実行すると、LS0100 から 100 バイト読み出したデータを ¥DATA フォルダに DATA0001.BIN ファイルを新規に作成します。

重要

- ファイル名は、8.3 フォーマット (ファイル名 8 文字、拡張子 3 文字の最大 12 文字) のみ使用できます。これ以上長いファイル名は使用できません。

モードを「追記」にしたとき

```
_CF_write ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 0, 100, 1)
```

フォルダ名 | ファイル名 | 読み出し先アドレス | オフセット | バイト数 | モード

すでに指定ファイル（例では DATA0001.BIN）が存在している場合に上記式を実行すると、LS0100 から 100 バイト読み出したデータを ¥DATA フォルダの DATA0001.BIN ファイルに追記します。

モードを「上書き」にしたとき (1)

```
_CF_write ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 16, 10, 2)
```

フォルダ名 | ファイル名 | 読み出し先アドレス | オフセット | バイト数 | モード

すでに指定ファイル（例では DATA0001.BIN）が存在している場合に上記式を実行すると、LS0100 から 10 バイト読み出したデータを ¥DATA フォルダの DATA0001.BIN ファイルのオフセット 17 バイト目から 10 バイト分上書きします。

モードを「上書き」にしたとき (2)

（上書きするファイルがオフセット + 追加バイト数より小さい場合）

```
_CF_write ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 96, 10, 2)
```

フォルダ名 | ファイル名 | 読み出し先アドレス | オフセット | バイト数 | モード

すでに指定ファイル（例では DATA0001.BIN）が存在していて、ファイルサイズが 100 バイトある場合に、オフセットを 96 バイト、バイト数を 10 バイトにして書き込んだ場合は、LS0100 から 10 バイト読み出したデータを 97 バイト目から 4 バイト分上書きし、残りの 6 バイトを追記書き込みします。従って、最終的には 106 バイトのファイルが作成されます。

モードを「上書き」にしたとき (3)

（上書きするファイルがオフセットより小さい場合）

```
_CF_write ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 110, 10, 2)
```

フォルダ名 | ファイル名 | 読み出し先アドレス | オフセット | バイト数 | モード

すでに指定ファイル（例では DATA0001.BIN）が存在していて、ファイルサイズが 100 バイトある場合に、オフセットを 110 バイト、バイト数を 10 バイトにして書き込んだ場合は、101 バイト目から 110 バイト分 0 で埋められて、LS0100 から 10 バイト読み出したデータを 111 バイト目から追記書き込みします。従って、最終的には 120 バイトのファイルが作成されます。

重要

- 第 1 パラメータのフォルダ名と第 2 パラメータのファイル名の最大文字数は、半角 32 文字までです。
- 第 2 パラメータのファイル名には、内部デバイスが指定可能です。内部デバイスを指定することにより間接的にファイル名を指定することが出来ます。また、ファイル名の文字数は最大 32 文字まで可能です。

例 :_CF_write ("¥DATA", [w:LS0100], [w:[#INTERNAL]LS0200], 0, 100, 0)

LS0100 にファイル名を格納することで、ファイル名を間接的に指定可能になります。ここでは、LS0100 から LS0106 に以下のようにファイル名を格納します。

16bit	
LS0100	'D' 'A'
LS0101	'T' 'A'
LS0102	'O' 'O'
LS0103	'O' '1'
LS0104	'.' 'B'
LS0105	'I' 'N'
LS0106	'¥0' '¥0'
	⋮

ファイル名の最後にはNULL文字を格納してください。表示器はNULL文字までをファイル名として扱います。

上記式を実行することで、LS0200 から 100 バイト読み出して "¥DATA¥DATA0001.BIN" のファイルを新規に作成します。

- ファイル名は、8.3 フォーマット（ファイル名 8 文字、拡張子 3 文字の最大 12 文字）のみ使用可能です。ロングファイル名は使用できません。

21.5.3 ファイル名変更

項目	内容
概要	ファイル名を変更します。パラメータ 1 で指定したフォルダのパラメータ 2 で指定したファイル名をパラメータ 3 で指定したファイル名に変更します。
書式	<p>_CF_rename (フォルダ名, ファイル名, 変更ファイル名) ファイル名は内部アドレスで間接指定することも可能です。</p>  <p>パラメータ 1 フォルダ名：固定文字列</p> <p>パラメータ 2 ファイル名：固定文字列、内部デバイス、内部デバイス + テンポラリアドレス</p> <p>パラメータ 3 ファイル名：固定文字列、内部デバイス、内部デバイス + テンポラリアドレス</p>

記述例

```
_CF_rename ("¥DATA", "DATA0001.BIN", "DATA1234.BIN")
```

上記例は、"¥DATA¥DATA0001.BIN" ファイル名 "¥DATA¥DATA1234.BIN" に変更します。

重要

- ファイル名は、8.3 フォーマット（ファイル名 8 文字、拡張子 3 文字の最大 12 文字）のみ使用可能です。ロングファイル名は使用できません。
- 第 1 パラメータのフォルダ名と第 2 パラメータのファイル名の最大文字数は、半角 32 文字までです。
- 第 2、3 パラメータのファイル名には、内部デバイスが指定可能です。内部デバイスを指定することにより間接的にファイル名を指定することが出来ます。また、ファイル名の文字数は最大 32 文字まで可能です。

例

```
_CF_rename ("¥DATA", [w:[#INTERNAL]LS0100], [w:[#INTERNAL]LS0200])
```

LS0100、LS0200 にファイル名を格納することで、ファイル名を間接的に指定可能になります。

- LS0100 から LS0106 に以下のようにファイル名を格納して下さい。

16bit	
LS0100	'D' 'A'
LS0101	'T' 'A'
LS0102	'0' '0'
LS0103	'0' '1'
LS0104	'.' 'B'
LS0105	'I' 'N'
LS0106	'¥0' '¥0'
	:

← ファイル名の最後にはNULL文字を格納してください。GPはNULL文字までをファイル名として扱います。

16bit	
LS0200	'D' 'A'
LS0201	'T' 'A'
LS0202	'1' '2'
LS0203	'3' '4'
LS0204	'.' 'B'
LS0205	'I' 'N'
LS0206	'¥0' '¥0'
	:

上記式を実行することで、“¥DATA¥DATA0001.BIN” ファイルを“¥DATA¥DATA1234.BIN” ファイルにリネームします。

- 「ファイル名」に内部デバイスを指定した場合は、D スクリプトのアドレス数には加算されません。
- ルートフォルダ（ディレクトリ）を指定する場合には、フォルダ名に ""（空文字列）を指定してください。
- ファイル名にフルパスを指定する場合は、フォルダ名に "*"（アスタリスク）を指定してください。

21.5.4 CSV ファイルリード

項目	内容
概要	セルイメージ (「,」区切り) で構成された CSV ファイルから、セル単位で読み込み、ワードアドレスに書き込みます。
書式	<p><code>_CF_read_csv (フォルダ名, ファイル名, 書き込み先アドレス, 開始行, 読み込み行数)</code></p> <p>パラメータ 1: 文字列 (最大文字数は半角 32 文字) パラメータ 2: 文字列 (最大文字数は半角 32 文字) 内部デバイス、内部デバイス + テンポラリアドレス パラメータ 3: 内部デバイス、オフセット指定された内部デバイス パラメータ 4: 数値 (1 ~ 65535) 内部デバイス、テンポラリ変数 パラメータ 5: 数値 (1 ~ 65535) 内部デバイス、テンポラリ変数</p>

記述例

```
_CF_read_csv ("¥CSV", "SAMPLE.CSV", [w:[#INTERNAL]LS1000], 1, 2)
```

(CF カード内の「¥CSV¥SAMPLE.CSV」ファイルの 1 行目から 2 行分を `_CF_read_csv ()` 関数で読み込む場合)

SAMPLE.CSV

001, " DAT01-01 ", " DAT01-2 "
002, " DAT02-01 ", " DAT02-2 "

CSV ファイルの 1 行目から 2 行分を読み込みます。1 文字目が数値 ('0' ~ '9', '-') の場合、数値として格納されます。1 文字目が「,」の場合、文字として扱われ、文字列の末尾には「00h」が格納されます。例えば、「DAT01-01」を格納する場合、8 文字の偶数であるため文字列の格納に 4 ワードと末尾 00h を格納する 1 ワードで 5 ワード使用されます。「DAT01-2」を格納する場合、7 文字の奇数であるため文字列の格納に 4 ワード使用し、最後に 00h が格納されます。

16 ビット	
LS1000	1
+1	'D' 'A'
+2	'T' '0'
+3	'1' '-'
+4	'0' '1'
+5	00h 00h
+6	'D' 'A'
+7	'T' '0'
+8	'1' '-'
+9	'2' 00h
LS1010	2
+1	'D' 'A'
+2	'T' '0'
+3	'2' '-'
+4	'0' '1'
+5	00h 00h
+6	'D' 'A'
+7	'T' '0'
+8	'2' '-'
+9	'2' 00h

データ格納モードが 0 の場合

MEMO

- セルの 1 文字目が数値（「0」～「9」、「-」）の場合、数値データに変換し内部デバイスに書き込みを行います。数値データの有効範囲は -32768 ~ 32767 です。
- セルの 1 文字目が「"」の場合、「"」で囲まれた範囲を文字列データとして内部デバイスに書き込みを行います。文字列データが奇数バイトの場合、最後に 0x00 が書き込まれ、文字列データが偶数バイトの場合、最終アドレスの次のアドレスに 0x0000 が書き込まれます。1 セルの最大文字数は半角 32 文字までです。
- CSV ファイル内に複数行がある場合、特定の行から任意の行数を読み出すことができます。CSV ファイルの 1 行の最大文字数は半角 200 文字、最大行数は 65535 行です。
- エラーが発生した場合、LS9137 にエラーステータスを書き出します。
- CSV ファイルの文字列データを内部デバイスに書き込む場合、格納順序はデータ格納モードに依存します。

エラーステータス

	LS エリア
LS9137	

エディタ関数名	LS エリア	エラーステータス	要因
_CF_read_csv ()	LS9137	0000h	正常終了
		0001h	パラメータエラー
		0002h	CF カードエラー (CF カードなし / ファイルオープンエラー / ファイルリードエラー)
		0003h	書き込み、読み込みエラー

重要

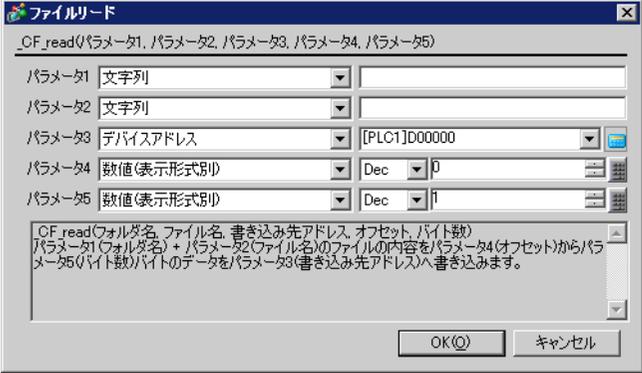
- フォルダ名に「"*"」を指定すると、ファイル名にフルパスを指定できます。
- ファイル名は、8.3 フォーマット（ファイル名 8 文字、拡張子 3 文字の最大 12 文字）のみ使用できます。これ以上長いファイル名は使用できません。
- CSV ファイルから読み込んだデータを格納できる内部デバイスの有効範囲はユーザーエリア（LS20 ~ LS2031、LS2096 ~ LS8191）のみです。
- 読み込みに必要な処理時間は、読み込まれる CSV ファイルのデータ量に比例します。また、処理が完了するまで、部品処理は更新されません。（1 行あたり 40 文字、100 行を使用した CSV ファイルの先頭から 100 行目まで読み込む場合、約 10 秒かかります）
- 「_CF_read()」関数と異なり、関数実行後に [s:CF_ERR_STAT] にステータスは格納されません。（不定値が格納される場合があります）
- 数字で始まる文字列には、必ず「[」を文字列の最初と最後に入れてください。

(例)

[123, 2-D4EA] [123, "2-D4EA"]

x

21.5.5 ファイルリード

項目	内容
概要	ファイルの内容をオフセットから指定バイト数分、書き込み先アドレスに書き込みます。データの格納順序については後述する「データ格納モード」を参照してください。
書式	<p><code>_CF_read (フォルダ名, ファイル名, 書き込み先アドレス, オフセット, バイト数)</code></p>  <p>パラメータ 1 フォルダ名：固定文字列（最大文字数は、半角 32 文字）</p> <p>パラメータ 2 ファイル名：固定文字列、内部デバイス、内部デバイス + テンポラリアドレス（最大文字数は、半角 32 文字）</p> <p>パラメータ 3 書き込み先アドレス：デバイスアドレス、デバイスアドレス + テンポラリアドレス</p> <p>パラメータ 4 オフセット：数値、デバイスアドレス、テンポラリアドレス（指定最大数は 16 ビット長の時 65535、32 ビット長の時 4294967295）</p> <p>パラメータ 5 バイト数：数値、デバイスアドレス、テンポラリアドレス（指定最大数は 1280）</p>

記述例

指定ファイルのオフセット 16 から 16 バイト分読み出す場合

```
_CF_read ("¥DATA", "DATA0001.BIN", [w:[#INTERNAL]LS0100], 16, 16)
```

上記記述を実行すると、「¥DATA¥DATA0001.BIN」ファイルの 17 バイト目から 16 バイト分のデータを LS0100 以降に書き込みます。

重要

- ファイル名は、8.3 フォーマット（ファイル名 8 文字、拡張子 3 文字の最大 12 文字）のみ使用可能です。ロングファイル名は使用できません。
- 第 1 パラメータのフォルダ名と第 2 パラメータのファイル名の最大文字数は、半角 32 文字までです。
- 第 2 パラメータのファイル名には、内部デバイスが指定可能です。内部デバイスを指定することにより間接的にファイル名を指定することが出来ます。また、ファイル名の文字数は最大半角 32 文字まで可能です。

例

LS0100 ~ で指定されたファイルのオフセット 0 から 10 バイト分読み出す場合
`_CF_read ("¥DATA", [w:LS0100], [w:LS0200], 0, 10)`

LS0100 にファイル名を格納することで、ファイル名を間接的に指定可能になります。ここでは、LS0100 から LS0106 に次項のようにファイル名を格納しています。

	16bit	
LS0100	'D'	'A'
LS0101	'T'	'A'
LS0102	'0'	'0'
LS0103	'0'	'1'
LS0104	'.'	'B'
LS0105	'I'	'N'
LS0106	'¥0'	'¥0'
	⋮	

← ファイル名の最後には NULL 文字を格納してください。表示器は NULL 文字までをファイル名として扱います。

上記式を実行することで、“¥DATA¥DATA0001.BIN” のファイルの先頭から 10 バイト読み出して LS0200 ~ に書き込みます。

- 実際に読み込めたバイト数は、CF カード読み出しバイト数 [s:CF_READ_NUM] に書き込まれます。詳細については、「21.5.1 ラベル設定 CF カードエラーステータス」(21-38 ページ)
- 「ファイル名」に内部デバイスを指定した場合と「書き込み先アドレス」は、D スクリプトのアドレス数には加算されません。
- 書き込み先アドレスに PLC デバイスを指定した場合、書き込むワード数（バイト数）が多くなるに従って、PLC への書き込み時間が長くなります。ワード数によっては数秒かかる場合があります。
- ファイルから読み出したデータを書き込む場合に、PLC のデバイスの範囲外になった場合は通信エラーとなり、電源の ON/OFF をしないと復旧することは出来ませんのでご注意ください。
- 書き込み先に PLC デバイスを指定した場合、PLC との通信がありますので、すぐに書き込んだ値が反映されません。

例

次のスクリプトで、 の命令文ではファイルから 10 バイト読み出したデータを [w:D0100] から書き込みますが、通信を行っているため時間がかかり、 の命令文ではファイルから読み出したデータが [w:[PLC1]D0100] にはまだ書き込まれていません。

```
_CF_read ("¥DATA", "DATA0001.BIN", [w:[PLC1]D0100], 0, 10) .....
[w:[PLC1]D0200] = [w:[PLC1]D0100] + 1 .....
```

このような場合は次のように一度内部デバイスに格納して実行するようにして下さい。

```
_CF_read ("¥DATA", "DATA0001.BIN", [w:[PLC1]D0100], 0, 10)
memcpy ([w:[#INTERNAL]LS0100], [w:[PLC1]D0100], 10)
[w:[PLC1]D0200] = [w:[#INTERNAL]LS0100] + 1
```

21.5.6 ファイルリスト出力

項目	内容
概要	指定したフォルダに存在するファイルのリストを内部デバイスに書き込みます。パラメータ 1 で指定したフォルダのパラメータ 4 で指定したオフセット値から、パラメータ 3 で指定したファイル数分、パラメータ 2 で指定した内部デバイスへ書き込みます。オフセット「0」で 1 ファイル目（先頭ファイル）からとなります。
書式	<p>_CF_dir (フォルダ名, 書き込み先アドレス, ファイル数, オフセット)</p>  <p>パラメータ 1 フォルダ名：固定文字列（指定最大数は、半角 32 文字）</p> <p>パラメータ 2 書き込み先アドレス：内部デバイス、オフセット指定された内部デバイス</p> <p>パラメータ 3 ファイル名数：数値、デバイスアドレス、テンポラリアドレス（指定最大数 32）</p> <p>パラメータ 4 オフセット：数値、デバイスアドレス、テンポラリアドレス</p>

記述例

オフセット 1（2 ファイル目）から 2 ファイル分のファイルリストを出力する場合

_CF_dir ("¥DATA¥*.¥*", [w:[#INTERNAL]LS0100], 2, 1)

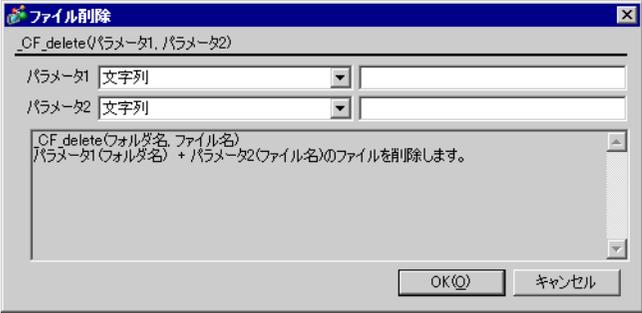
DATA フォルダ内に次のファイルが存在しているときに上記式を実行するとファイル名「DATA0001.BIN」、 「DATA02.BIN」を LS0100 以降に書き込みます。

(フォルダ内容)	(LS エリア内容)																														
<pre>¥DATA ├── DATA0000.BIN ├── DATA0001.BIN ├── DATA02.BIN ├── DATA0003.BIN └── DATA0004.BIN</pre>	<table border="1" style="border-collapse: collapse; text-align: center;"> <thead> <tr> <th colspan="2">16bit</th> </tr> </thead> <tbody> <tr><td>LS0100</td><td>'D' 'A'</td></tr> <tr><td>LS0101</td><td>'T' 'A'</td></tr> <tr><td>LS0102</td><td>'0' '0'</td></tr> <tr><td>LS0103</td><td>'0' '1'</td></tr> <tr><td>LS0104</td><td>'.'</td></tr> <tr><td>LS0105</td><td>'I' 'N'</td></tr> <tr><td>LS0106</td><td>'¥0' '¥0'</td></tr> <tr><td>LS0107</td><td>'D' 'A'</td></tr> <tr><td>LS0108</td><td>'T' 'A'</td></tr> <tr><td>LS0109</td><td>'0' '2'</td></tr> <tr><td>LS0110</td><td>'.'</td></tr> <tr><td>LS0111</td><td>'I' 'N'</td></tr> <tr><td>LS0112</td><td>'¥0' '¥0'</td></tr> <tr><td>LS0113</td><td>'¥0' '¥0'</td></tr> </tbody> </table>	16bit		LS0100	'D' 'A'	LS0101	'T' 'A'	LS0102	'0' '0'	LS0103	'0' '1'	LS0104	'.'	LS0105	'I' 'N'	LS0106	'¥0' '¥0'	LS0107	'D' 'A'	LS0108	'T' 'A'	LS0109	'0' '2'	LS0110	'.'	LS0111	'I' 'N'	LS0112	'¥0' '¥0'	LS0113	'¥0' '¥0'
16bit																															
LS0100	'D' 'A'																														
LS0101	'T' 'A'																														
LS0102	'0' '0'																														
LS0103	'0' '1'																														
LS0104	'.'																														
LS0105	'I' 'N'																														
LS0106	'¥0' '¥0'																														
LS0107	'D' 'A'																														
LS0108	'T' 'A'																														
LS0109	'0' '2'																														
LS0110	'.'																														
LS0111	'I' 'N'																														
LS0112	'¥0' '¥0'																														
LS0113	'¥0' '¥0'																														
	} 7 ワード使用します。 } 7 ワード使用します。																														

重要

- オフセット「0」で1ファイル目（先頭ファイル）からとなります。
- ファイル名は8.3フォーマット（ファイル名8文字、拡張子3文字の最大12文字）のみ使用可能です。ロングファイル名は使用できません。
- フォルダ内に指定したファイル数分のファイルが存在していないときは、内部デバイスをNULL文字（'¥0'）で埋められます。
- ファイル名が12文字に満たない場合は、残りはNULL文字（'¥0'）で埋められます。
- フォルダ名を指定するときには、“¥DATA¥*.*”のように必ず“*.*”も記述して下さい。“*.*”は、全てのファイルをリスト出力することを意味します。
- 実際にリスト出力したファイル数は、CFカードリストファイル数[s:CF_FILELIST_NUM]に書き込まれます。
詳細については、「CFカードエラーステータス」(21-38ページ)
- 書き込み先内部デバイスはDスクリプトのアドレス数には加算されません。
- ファイル名を内部デバイスに書き込むときにファイル名のソート処理は行われません。ファイル作成順（FATのエントリ順）になります。
- ファイルの拡張子を指定してリスト出力を行うことが可能です。特定の拡張子だけをリスト出力するときは、“¥DATA¥*.BIN”などで指定することが出来ます。ただし、ファイル名の途中に「*」を付けることは出来ません。

21.5.7 ファイル削除

項目	内容
概要	指定したファイルを削除します。パラメータ1で指定したフォルダのパラメータ2で指定したファイルを削除します。
書式	<p>_CF_delete (フォルダ名, ファイル名) ファイル名は内部アドレスで間接指定することも可能です。</p>  <p>パラメータ1 フォルダ名：固定文字列</p> <p>パラメータ2 ファイル名：固定文字列、内部デバイス、内部デバイス + テンポラリアドレス</p>

記述例

```
_CF_delete ("¥DATA", "DATA0001.BIN")
```

上記例は、“¥DATA¥DATA0001.BIN” ファイルを削除します。

重要

- ファイル名は、8.3 フォーマット（ファイル名 8 文字、拡張子 3 文字の最大 12 文字）のみ使用可能です。ロングファイル名は使用できません。
- 第 1 パラメータのフォルダ名と第 2 パラメータのファイル名の最大文字数は、半角 32 文字までです。
- 第 2 パラメータのファイル名には、内部デバイスが指定可能です。内部デバイスを指定することにより間接的にファイル名を指定することが出来ます。また、ファイル名の文字数は最大半角 32 文字まで可能です。

ここでは、LS0100 から LS0106 に以下のようにファイル名を格納しています。

16bit	
LS0100	'D' 'A'
LS0101	'T' 'A'
LS0102	'0' '0'
LS0103	'0' '1'
LS0104	'.' 'B'
LS0105	'I' 'N'
LS0106	'¥0' '¥0'
	⋮

ファイル名の最後にはNULL文字を格納してください。表示器はNULL文字までをファイル名として扱います。

上記式を実行することで、“¥DATA¥DATA0001.BIN”のファイルを削除します。

- ルートフォルダ（ディレクトリ）を指定する場合には、フォルダ名に ""（空文字列）を指定してください。
- 「ファイル名」に内部デバイスを指定した場合、また「書き込み先アドレス」は、D スクリプトのアドレス数には加算されません。
- ファイル名にフルバスを指定する場合は、フォルダ名に "*"（アスタリスク）を指定してください。

21.6 プリンタ操作

プリンタ操作	動作概要
	<p>ラベル設定 ☞「21.6.1 ラベル設定」(21-56 ページ) コントロール、ステータスから指定します。</p> <p>送信 ☞「21.6.2 送信」(21-58 ページ) COM ポートに指定したバイト数分だけデータを出力します。</p>

重要 • プリンタ操作関数として使用できるポートは、COM1 または USB/PIO(USB-PIO 変換) となります。

21.6.1 ラベル設定

コントロール

コントロール (PRN_CTRL) は、送信バッファ、エラーステータスのクリアを行うためのコントロール変数です。このコントロール変数は、書き込み専用です。

- コントロール (PRN_CTRL) の内容

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
<input type="checkbox"/>															

ビット	内容
15	
14	
13	
12	
11	
10	
9	予約
8	
7	
6	
5	
4	
3	
2	1: エラークリア
1	予約
0	1: 送信バッファクリア

重要 • ワード指定の場合 (複数ビットを同時にセットした場合) 処理する順は以下の通りです。

エラークリア

送信バッファクリア

- 予約ビットは将来使用する可能性がありますので、必要なビットのみをセットするようにして下さい。

ステータス

ステータス (PRN_STAT) は、送信バッファ内のデータの有無、エラーの状態を得るためのステータス変数です。このステータス変数は、読み込み専用です。

- ステータス (PRN_STAT) の内容

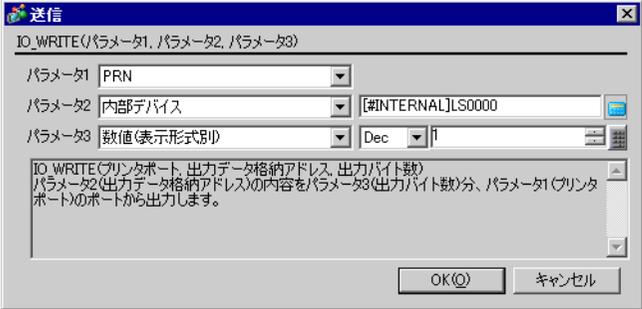
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

ビット	内容
15	予約
14	プリンタ I/F ERROR 信号 <プリンタエラー (入力)>の状態 0:Error 1:Normal
13	プリンタ I/F SLCT 信号 <セレクト (入力)>の状態 0:Off line 1:On line
12	プリンタ I/F PE 信号 <紙切れ (入力)>の状態 0:Normal 1:Paper Empty
11	予約
10	
9	
8	
7	
6	
5	
4	
3	
2	
1	0: 正常 1: 送信エラー
0	0: 送信バッファにデータ有り 1: 送信バッファエンプティ

重要

- 送信エラーには送信バッファオーバーフローエラーがあります。このエラーが発生すると、送信エラーのビットが ON します。
- 送信バッファは 8192 バイトです。
- 予約ビットは将来使用する可能性がありますので、必要なビットのみをチェックするようにして下さい。

21.6.2 送信

項目	内容
概要	COM ポートに指定したバイト数だけデータを出力します。システム設定のプリンタタイプの設定には依存せず、指定されたデータをそのまま出力します。
書式	<p>IO_WRITE (プリンタポート、出力データ格納アドレス、出力バイト数)</p>  <p>パラメータ 1 : [p:PRN] パラメータ 2 : 内部デバイス パラメータ 3 : 整数値、デバイスアドレス、テンポラリアドレス</p>

重要

- パラメータ 3 に設定できる数値は最大 1024 までです。これ以上の数値を設定した場合は、送信データ数を 1024 にして COM ポートから出力します。

記述例 1

```
IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 10)
```

上記式を実行した場合には、LS1000 から 10 バイト分のデータを COM ポートから出力します。

記述例 2

```
IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], [w:[#INTERNAL]LS0800])
```

上記式を実行した場合には、LS1000 から LS0800 に書き込まれたバイト数分のデータを COM ポートから出力します。

記述例 3

```
IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], [t:0010])
```

上記式を実行した場合には、LS1000 からテンポラリ [t:0010] に書き込まれたバイト数分のデータを COM ポートから出力します。

データ格納モード

COMポート操作関数実行時に、デバイスアドレスから読み出す場合に、読み出す格納順序を設定します。

LS9130 にデータ格納モードを設定することで格納順序を変更することが可能です。

モードは0, 1, 2, 3 の4通りがあります。

モード0

例：COMポート操作関数を使用してデバイスアドレスから文字列 ABCDEFG を読み込む場合

[w:[#INTERNAL]LS9130] = 0

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- デバイスアドレスが 16 ビット長の場合

LS0100	'A'	'B'
LS0101	'C'	'D'
LS0102	'E'	'F'
LS0103	'G'	0

← 格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'A'	'B'	'C'	'D'
LS0101	'E'	'F'	'G'	0
LS0102				

← 格納するデータ数のあまりバイトに0が書き込まれます。

モード1

例：COMポート操作関数を使用してデバイスアドレスから文字列 ABCDEFG を読み込む場合

[w:[#INTERNAL]LS9130] = 1

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- デバイスアドレスが 16 ビット長の場合

LS0100	'B'	'A'
LS0101	'D'	'C'
LS0102	'F'	'E'
LS0103	0	'G'

← 格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'B'	'A'	'D'	'C'
LS0101	'F'	'E'	0	'G'
LS0102				

← 格納するデータ数のあまりバイトに0が書き込まれます。

モード 2

例：COM ポート操作関数を使用してデバイスアドレスから文字列 ABCDEFG を読み込む場合

[w:[#INTERNAL]LS9130] = 2

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- デバイスアドレスが 16 ビット長の場合

LS0100	'C'	'D'
LS0101	'A'	'B'
LS0102	'G'	0
LS0103	'E'	'F'

格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'C'	'D'	'A'	'B'
LS0101	0	'G'	'E'	'F'
LS0102				

格納するデータ数のあまりバイトに0が書き込まれます。

モード 3

例：COM ポート操作関数を使用してデバイスアドレスから文字列 ABCDEFG を読み込む場合

[w:[#INTERNAL]LS9130] = 3

IO_WRITE ([p:PRN], [w:[#INTERNAL]LS1000], 7)

- デバイスアドレスが 16 ビット長の場合

LS0100	'D'	'C'
LS0101	'B'	'A'
LS0102	0	'G'
LS0103	'F'	'E'

格納するデータ数のあまりバイトに0が書き込まれます。

- デバイスアドレスが 32 ビット長の場合

LS0100	'D'	'C'	'B'	'A'
LS0101	0	'G'	'F'	'E'
LS0102				

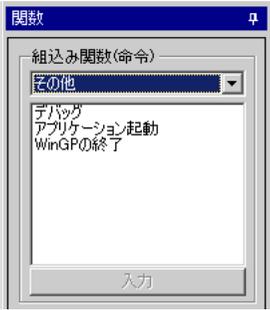
格納するデータ数のあまりバイトに0が書き込まれます。

重要

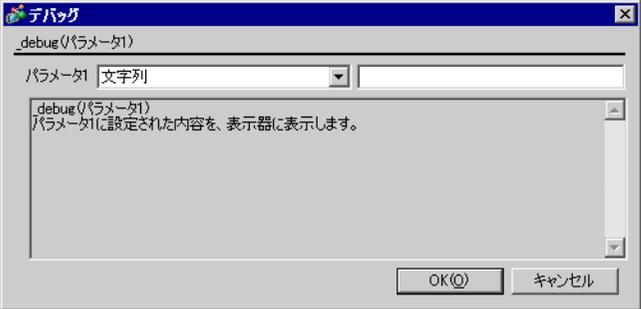
- データ格納モードとシステムの設定にある文字列データモードとは一致していません。文字列格納モードとの対比は以下のようになります。

データのデバイス格納順序	ワード内のバイト LH/HL 格納順序	ダブルワード内のバイト LH/HL 格納順序	D スクリプト データ格納モード	文字列データモード
先頭データから格納	HL 順	HL 順	0	1
	LH 順		1	2
	HL 順	LH 順	2	5
	LH 順		3	4
最終データから格納	HL 順	HL 順	-	3
	LH 順		-	7
	HL 順	LH 順	-	8
	LH 順		-	6

21.7 その他

その他	動作概要
	<p>デバッグ関数</p> <p>☞「21.7.1 デバッグ関数」(21-61 ページ)</p> <p>指定したアドレスの値や文字列をデバッグ用として画面に表示させます。</p>
	<p>アプリケーション起動</p> <p>☞「21.7.2 アプリケーション起動」(21-63 ページ)</p> <p>指定した EXE を実行して、アプリケーションを起動します。</p>
	<p>WinGP の終了</p> <p>☞「21.7.3 WinGP の終了」(21-65 ページ)</p> <p>WinGP を終了します。</p>

21.7.1 デバッグ関数

項目	内容
<p>概要</p>	<p>指定したアドレスの値や文字列をデバッグ用として画面に表示させます。デバッグ完了後は、スクリプトエディタの「デバッグ関数を有効にする」のチェックを外すと、実行文中から記述を削除することなく、画面上に表示されなくなります。</p>
<p>書式</p>	<p>_debug (パラメータ 1)</p>  <p>パラメータ 1 : 文字列 (指定最大数は、半角 32 文字、全角 16 文字)</p>

パラメータ 1 の内容について

パラメータ 1	書式	内容
文字列	<code>_debug ("ABC")</code>	" " 内の文字列を表示します。文字列は最大半角 32 文字です。
ワードアドレスまたはテンポラリアドレス	<code>_debug (w:D1000)</code>	設定したワードアドレスまたはテンポラリアドレスの値を表示します。
改行	<code>_debug (_CRLF)</code>	次の行の先頭にカーソルを移動させます。
復帰	<code>_debug (_CR)</code>	同じ行の先頭にカーソルを移動させます。

記述例 1

以下のスクリプトでワードアドレスの値を表示させます。

```
[w:[#INTERNAL]LS0100]=100
_debug([w:[#INTERNAL]LS0100))
_debug(_CRLF)
[w:[#INTERNAL]LS0100]=50
_debug([w:[#INTERNAL]LS0100))
```

```
100(64h)
50(32h)
```

表示は以下のフォーマットで表示されます。
`***** (***h)`
 Dec表示 Hex表示

記述例 2

以下のスクリプトで改行して文字列を表示させます。

```
_debug("Test1")
_debug(_CRLF)
_debug("Test2")
```

```
Test1
Test2
```

1 行段落を下げて "Test2" を表示します。

21.7.2 アプリケーション起動

機種が IPC シリーズの場合のみ動作する関数です。

項目	内容
概要	<p>指定した EXE を実行して、アプリケーションを起動します。 起動時のパラメータや多重起動の監視などが設定できます。</p>
書式	<p>Exec Process (パラメータ 1, パラメータ 2, パラメータ 3, パラメータ 4)</p>  <p>パラメータ 1 EXE パス: 起動したいアプリケーションの実行ファイル (.exe) の絶対パスを入力します。最大 255 文字まで入力できます。</p> <p>パラメータ 2 パラメータ: 実行ファイル起動時の引数を入力します。最大 255 文字まで入力できます。</p> <p>パラメータ 3 ウィンドウタイトル: 多重起動を禁止するときは、「多重起動を禁止」を選択して [ウィンドウタイトル] を入力します。最大 63 文字まで入力できます。 [ウィンドウタイトル] と完全に一致するウィンドウが見つかれば、アプリケーションを起動しません。 「多重起動を許可」を選択した場合や [ウィンドウタイトル] に何も設定されていない場合は、多重起動が許可されます。</p> <p>パラメータ 4 完全にタイトルが一致するウィンドウを検索: パラメータ 3 で「多重起動を禁止」を選択した場合にのみ有効です。 「0: 部分一致」を選択した場合は [ウィンドウタイトル] に入力したタイトルと部分的に一致するウィンドウが、「1: 完全一致」を選択した場合は [ウィンドウタイトル] に入力したタイトルと完全に一致するウィンドウが見つかると、指定したアプリケーションが実行されません。</p>

MEMO

- パラメータ 1 には必ず文字列 (EXE パス) が必要です。文字列が入力されていない場合はエラーになります。
- IPC シリーズ以外の機種に「アプリケーション起動」のスク립トが転送された場合、この機能は動作しません。

パラメータ 1 (EXE パス) の入力方法

EXE パスの入力方法は、次の 3 種類があります。

以下の説明では、C:¥Documents and Settings¥user¥Local Settings¥Temp の sample.exe を実行する場合を例としています。

1. フルパス指定

例) C:¥Documents and Settings¥user¥Local Settings¥Temp¥sample.exe

2. EXE 名のみ

IPC シリーズの環境設定で Path に設定されているフォルダのいずれかに実行ファイルがある場合

例) sample.exe

(Path=C:¥Documents and Settings¥user¥Local Settings¥Temp と設定されている場合は起動)

3. 環境変数でパス指定

IPC シリーズの環境設定で環境変数に設定されているフォルダに実行ファイルがある場合

例) %TEMP%¥sample.exe

(環境変数 TEMP=C:¥Documents and Settings¥user¥Local Settings¥Temp と設定されている場合は起動)

記述例 1

多重起動を許可 (ノートパッドを起動し、Readme.txt を表示)

```
Exec_Process ( "C:¥WINDOWS¥SYSTEM32¥notepad.exe","D:¥TEMP¥Readme.txt","",0)
```

```
Exec_Process ( "%SystemFolder%¥notepad.exe","D:¥TEMP¥Readme.txt","",1)
```

記述例 2

多重起動を禁止 : 部分一致 (ノートパッドを起動し、Readme.txt を表示)

```
Exec_Process ( "C:¥WINDOWS¥SYSTEM32¥notepad.exe","D:¥TEMP¥Readme.txt","Readme",0)
```

記述例 3

多重起動を禁止 : 完全一致 (ノートパッドを起動し、Readme.txt を表示)

```
Exec_Process ( "C:¥WINDOWS¥SYSTEM32¥notepad.exe","D:¥TEMP¥Readme.txt"," Readme.txt",1)
```

記述例 4

多重起動を禁止 : 部分一致 (ノートパッドを起動)

```
Exec_Process ( "C:¥WINDOWS¥SYSTEM32¥notepad.exe","", "メモ帳",0)
```

記述例 5

パラメータなし (ノートパッドを起動)

```
Exec_Process ( "C:¥WINDOWS¥SYSTEM32¥notepad.exe","", "",0)
```

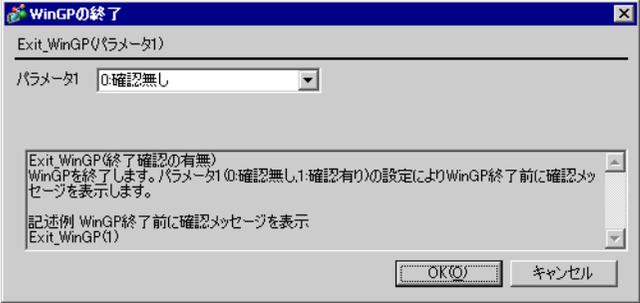
記述例 6

複数パラメータ (sample.exe を起動)

```
Exec_Process ( "C:¥WINDOWS¥SYSTEM32¥sample.exe","/v /a/s","",1)
```

21.7.3 WinGP の終了

機種が IPC シリーズの場合のみ動作する関数です。

項目	内容
概要	WinGP を終了します。終了時に確認メッセージを表示することもできます。
書式	<p>Exit_WinGP(パラメータ 1)</p>  <p>パラメータ 1 フォルダ名:「0: 確認無し」または「1: 確認有り」を選択します。</p>

MEMO

- パラメータ 1 には必ず文字列 (EXE パス) が必要です。文字列が入力されていない場合はエラーになります。
- IPC シリーズ以外の機種に「WinGP の終了」のスク립トが転送された場合、この機能は動作しません。

記述例

WinGP 終了時に確認メッセージを表示する場合

Exit_WinGP(1)

21.8 記述式

記述式	動作概要
記述式 if - endif if - else - endif loop - endloop break return	if - endif ☞「21.8.1 if - endif」(21-66 ページ) if に続く () 内の条件式が成立時、if () より後の処理を実行します。
	if - else - endif ☞「21.8.2 if - else - endif」(21-66 ページ) if に続く () 内の条件式が成立時に if () より後の処理を実行します。 不成立時には else 後の処理を実行します。
	loop - endloop ☞「21.8.3 loop - endloop」(21-67 ページ) loop に続く () 内のテンポリアドレスの値の回数だけループ処理 (繰り返し処理) を行います。
	break ☞「21.8.4 break」(21-68 ページ) loop () 式の実行途中で、その loop () 式から抜ける処理を行います。
	return ☞「21.8.5 return」(21-68 ページ) 再度、先頭から処理を行います。 拡張スクリプトのみ使用できます。

21.8.1 if - endif

if に続く () 内の条件式が成立時、if () より後の処理を実行します。

MEMO

- 条件式には代入「=」は使用できません。

21.8.2 if - else - endif

if に続く () 内の条件式が成立時に if () より後の処理を実行します。不成立時には else 後の処理を実行します。

MEMO

- 条件式には代入「=」は使用できません。

21.8.3 loop - endloop

loop に続く () 内のテンポラリアドレスの値の回数だけループ処理（繰り返し処理）を行います。

無限ループ

loop () の () 内に何も記述しないときは無限ループとなります。

無限ループは、拡張スクリプトのみ使用できます。

記述例

```
loop ( )
{
  [w:[#INTERNAL]LS0100]=[w:[#INTERNAL]LS0100]+1
  if ( [w:[#INTERNAL]LS0100] >10)
  {
    break
  }
  endif
}
endloop
```

MEMO

- loop () の書式の例は以下の通りです。

例)

```
loop ( ループ数 ) <= ループする回数がセットされたテンポラリアドレスを指
定
{
  動作式
  break <= 途中でループを抜ける場合に記述 (省略可)
} endloop <= ループの最後に記述
```

- (ループ数) にはテンポラリアドレスのみ指定可能 (例 :loop ([t:000]))
- loop () はトリガ式内では使用できません。
- ループ数に指定したテンポラリアドレス内の値はループをするたびに 1 ずつ減少していき、0 になった時点でループから抜けます。ループ内の動作式でループ数に指定したテンポラリアドレスを加工したりすると永久ループになりますのでご注意ください。また、テンポラリアドレスは、グローバルなワードになっていますので、別の箇所と同じテンポラリアドレスを使用する場合は、プログラムによっては、永久ループになりますのでご注意ください。
- ループ処理が終わるまで、部品などの表示は更新されません。
- loop () のネストは可能です。ネストしている場合、break は一番内側の loop () だけ抜けます。

```
loop ([t:0000]) // ループ 1
{
  loop ([t:0001]) // ループ 2
  {
    break // ループ 2 を抜ける
  } endloop
} endloop // ループ 1 を抜ける
```

- 途中でループを抜けずにループを終了した場合は、テンポラリアドレスの値は 0 になっています。

MEMO

- テンポラリワークアドレスの値の範囲は、データ形式 (Bin、BCD) ビット長、符号 +/- により異なります。もしも、設定が符号ありでテンポラリワークアドレスの値がマイナス値になった場合には、ループの先頭で条件判定されてループを抜けます。
- ループ内では PLC デバイスを使用せず、GP 内部デバイスのユーザーエリアのデバイス、またはテンポラリワークアドレスを使用するようにしてください。

例えば、以下のような記述の場合には、短時間の間に多数 (以下の例では 100 個) の PLC への書き込みが発生することになり、通信の処理 (PLC への書き込み) が間に合わずシステムエラーが発生する場合があります。

例)

```
[t:0000] = 100 //100 回ループ
loop ([t:0000])
{
  [w:[PLC1]D0200] = [w:[#INTERNAL]LS0100] //D0200 に書き込む
  [w:[#INTERNAL]LS0100] = [w:[#INTERNAL]LS0100] + 1 //LS0100 をインクリメント
}endloop
```

次のように変更してください。

```
[t:0000] = 100 //100 回ループ
loop ([t:0000])
{
  [w:[#INTERNAL]LS0200] = [w:[#INTERNAL]LS0100] //D0200 に書き込む
  [w:[#INTERNAL]LS0100] = [w:[#INTERNAL]LS0100] + 1 //LS0100 をインクリメント
}endloop
[w:[PLC1]D0200]=[w:[#INTERNAL]LS0200] //LS0200 の内容を D0200
//に書き込む
```

- D スクリプト関数の関数名に “loop”、“break” を使用するとエラーになります。

21.8.4 break

loop () 式の実行途中で、その loop () 式から抜ける処理を行います。

MEMO

- break は loop () の { } 内でのみ使用可能です。

21.8.5 return

「ユーザー定義関数」に return が存在するとき
関数の処理を終了し、関数の呼び出し元に戻ります。

「実行 (メイン関数)」に return が存在するとき
メイン関数の処理をその時点で終了し、再度メイン関数の先頭から処理を行います。

MEMO

- 条件式には代入「=」は使用できません。

記述例

```
[w:[#INTERNAL]LS0100]=[w:[#INTERNAL]LS0200]>> 8) & 0xFF
if ([w:[#INTERNAL]LS0100]==0) //LS0100 が 0 のときは以降の処理は実行しない。
{
  set([b:[#INTERNAL]LS005000]) // エラー表示用の起動ビットを ON
  return // 終了
}
endif
```

21.9 比較

比較	動作概要
<div data-bbox="234 629 436 915" style="border: 1px solid gray; padding: 5px;"> 比較 論理積(and) 論理和(or) 否定(not) 未満(<) 以下(<=) 等しくない(<>) 超える(>) 以上(>=) 等しい(==) </div>	論理積 (and) ☞「21.9.1 論理積 (and)」(21-69 ページ) N1 and N2 : N1 と N2 がともに ON の時に真となります。
	論理和 (or) ☞「21.9.2 論理和 (or)」(21-69 ページ) N1 or N2 : N1 もしくは N2 のどちらかが ON の時に真となります。
	否定 (not) ☞「21.9.3 否定 (not)」(21-70 ページ) not N1 : N1 が 1 ならば 0、N1 が 0 ならば 1 となります。
	未満 (<) ☞「21.9.4 未満 (<)」(21-70 ページ) N1 < N2 ならば真となります。
	以下 (< =) ☞「21.9.5 以下 (< =)」(21-70 ページ) N1 < = N2 (N1 N2)ならば真となります。
	等しくない (< >) ☞「21.9.6 等しくない (< >)」(21-70 ページ) N1 < > N2 (N1 N2)ならば真となります。
	超える (>) ☞「21.9.7 超える (>)」(21-70 ページ) N1 > N2 ならば真となります。
	以上 (> =) ☞「21.9.8 以上 (> =)」(21-70 ページ) N1 > = N2 (N1 N2)ならば真となります。
	等しい (==) ☞「21.9.9 等しい (==)」(21-70 ページ) N1 == N2 (N1=N2)ならば真となります。

21.9.1 論理積 (and)

左辺と右辺の論理積を実行します。0 を OFF、0 以外を ON とします。

N1 and N2 : N1 と N2 がともに ON の時に真となります。その他は偽となります。

21.9.2 論理和 (or)

左辺と右辺の論理和を実行します。0 を OFF、0 以外を ON とします。

N1 or N2 : N1 もしくは N2 のどちらかが ON の時に真となります。両方とも偽の時は OFF となります。

21.9.3 否定 (not)

右辺の否定を実行します。0 を 1、0 以外を 0 とします。
notN1 : N1 が 1 ならば 0、N1 が 0 ならば 1 となります。

21.9.4 未満 (<)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の比較を実行します。
N1 < N2 ならば真となります。

21.9.5 以下 (<=)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の比較を実行します。
N1 <= N2 (N1 N2) ならば真となります。

21.9.6 等しくない (<>)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の比較を実行します。N1 <> N2
(N1 N2) ならば真となります。

21.9.7 超える (>)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の比較を実行します。
N1 > N2 ならば真となります。

21.9.8 以上 (>=)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の比較を実行します。
N1 >= N2 (N1 N2) ならば真となります。

21.9.9 等しい (==)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の比較を実行します。
N1 == N2 (N1=N2) ならば真となります。

コマンド		文例
論理積	and	if ((演算式) and (演算式))
論理和	or	if ((演算式) or (演算式))
否定	not	if (not (演算式))
未満	<	< 項 1 > < 項 2 >
以下	<=	< 項 1 > <= < 項 2 >
等しくない	<>	< 項 1 > <> < 項 2 >
超える	>	< 項 1 > > < 項 2 >
以上	>=	< 項 1 > >= < 項 2 >
等しい	==	< 項 1 > == < 項 2 >

21.10 演算子

演算子	動作概要
	加算 (+) ☞「21.10.1 加算 (+)」(21-72 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の加算を実行します。
	減算 (-) ☞「21.10.2 減算 (-)」(21-72 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の減算を実行します。
	余り (%) ☞「21.10.3 余り (%)」(21-72 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の剰余算を実行します。
	掛け算 (*) ☞「21.10.4 掛け算 (*)」(21-72 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の掛算を実行します。
	割り算 (/) ☞「21.10.5 割り算 (/)」(21-72 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の割算を実行します。
演算子 加算(+) 減算(-) 余り(%) 掛け算(*) 割り算(/) 代入(=)	代入 (=) ☞「21.10.6 代入 (=)」(21-72 ページ) 左辺に右辺の値を代入します。
左シフト(<<) 右シフト(>>) ビット演算子 論理積(&) ビット演算子 論理和() ビット演算子 排他的論理和(^) ビット演算子 1の補数(~)	左シフト (<<) ☞「21.10.7 左シフト (<<)」(21-72 ページ) 左辺のデータを右辺の数分、左にシフトします。
	右シフト (>>) ☞「21.10.8 右シフト (>>)」(21-72 ページ) 左辺のデータを右辺の数分、右にシフトします。
	ビット演算子 論理積 (&) ☞「21.10.9 ビット演算子 論理積 (&)」(21-73 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の論理積を実行します。
	ビット演算子 論理和 () ☞「21.10.10 ビット演算子 論理和 ()」(21-73 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の論理和を実行します。
	排他的論理和 (^) ☞「21.10.11 排他的論理和 (^)」(21-73 ページ) ワードデバイス間のデータもしくはワードデバイスのデータと定数の排他的論理和を実行します。
	ビット演算子 1の補数 (~) ☞「21.10.12 ビット演算子 1の補数 (~)」(21-73 ページ) ビットを反転します。

21.10.1 加算 (+)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の加算を実行します。演算結果が桁あふれをした場合は切り捨てられます。

21.10.2 減算 (-)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の減算を実行します。演算結果が桁あふれをした場合は切り捨てられます。

21.10.3 余り (%)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の剰余算を実行します。(割算を行い余りを検出) 剰余算の場合は右辺と左辺の符号により演算結果が異なります。

21.10.4 掛け算 (*)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の掛算を実行します。演算結果が桁あふれをした場合は切り捨てられます。

21.10.5 割り算 (/)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の割算を実行します。割算結果の小数点以下は切り捨てられます。演算結果が桁あふれをした場合は切り捨てられます。

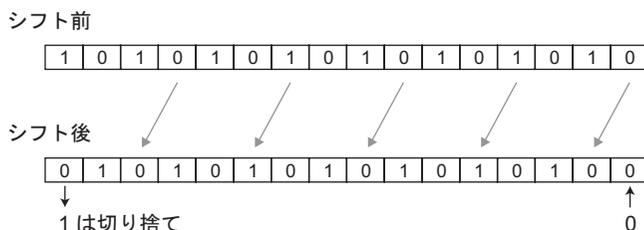
21.10.6 代入 (=)

左辺に右辺の値を代入します。左辺にはデバイスのみ記述することができます。右辺にはデバイス、定数を記述することができます。演算結果が桁あふれをした場合は切り捨てられます。

21.10.7 左シフト (<<)

左辺のデータを右辺の数分、左にシフトします。論理シフトのみサポートします。

(例) 左シフトの場合 (左に1ビットシフト)



21.10.8 右シフト (>>)

左辺のデータを右辺の数分、右にシフトします。論理シフトのみサポートします。

21.10.9 ビット演算子 論理積 (&)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の論理積を実行します。ある特定のビットを抜き出したり、あるビット列をマスクする場合に使用します。

21.10.10 ビット演算子 論理和 (|)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の論理和を実行します。ある特定のビットを ON する場合に使用します。

21.10.11 排他的論理和 (^)

ワードデバイス間のデータもしくはワードデバイスのデータと定数の排他的論理和を実行します。

21.10.12 ビット演算子 1 の補数 (~)

ビットを反転します。

MEMO

- 演算結果の桁あふれ、剰余算の演算結果の違いおよび小数点の切り捨てについて、
☞ 「20.9.4 演算結果の注意事項」(20-58 ページ)

優先順位・結合規則

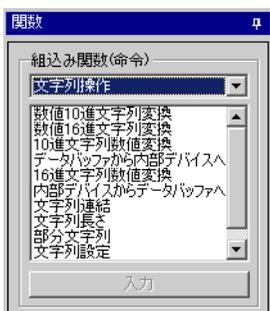
起動条件には優先順位があります。なお、同順位ならば結合規則の示す方向に従います。

優先順位	演算子	結合規則	
高	()	→	
	not ~	←	
	* / %	→	
	+ -	→	
	<< >>	→	
	< <= > >=	→	
	== <>	→	
	& ^	→	
	and or	→	
	低	=	←

21.11 文字列操作

文字列操作の各関数は、拡張スクリプトのみ使用できます。

文字列操作	動作概要
	<p>10 進文字列数値変換関数</p> <p>☞「21.11.1 10 進文字列数値変換」(21-75 ページ)</p> <p>10 進文字列を整数値に変換する関数です。</p>
	<p>16 進文字列数値変換関数</p> <p>☞「21.11.2 16 進文字列数値変換」(21-77 ページ)</p> <p>16 進文字列をバイナリデータに変換する関数です。</p>
	<p>内部デバイスからデータバッファにコピー</p> <p>☞「21.11.3 内部デバイスからデータバッファへ」(21-79 ページ)</p> <p>内部デバイスからデータバッファに文字列データをコピーします。</p>
	<p>データバッファから内部デバイスにコピー</p> <p>☞「21.11.4 データバッファから内部デバイスへ」(21-81 ページ)</p> <p>データバッファから内部デバイスに文字列データをコピーします。</p>
	<p>ステータス</p> <p>☞「21.11.5 文字列操作エラーステータス」(21-83 ページ)</p> <p>発生したエラーを格納します。</p>
	<p>数値 10 進文字列変換関数</p> <p>☞「21.11.6 数値 10 進文字列変換」(21-84 ページ)</p> <p>整数値を 10 進文字列に変換する関数です。</p>
	<p>数値 16 進文字列変換関数</p> <p>☞「21.11.7 数値 16 進文字列変換」(21-85 ページ)</p> <p>バイナリデータを 16 進数文字列に変換する関数です。</p>
	<p>部分文字列関数</p> <p>☞「21.11.8 部分文字列」(21-86 ページ)</p> <p>指定した文字列長分を抽出して別のバッファに格納します。</p>
	<p>文字列書き込み</p> <p>☞「21.11.9 文字列設定」(21-87 ページ)</p> <p>固定文字列をデータバッファに格納します。</p>
	<p>文字列長さ取得関数</p> <p>☞「21.11.10 文字列長さ」(21-88 ページ)</p> <p>格納されている文字列の長さを得ます。</p>
	<p>文字列連結関数</p> <p>☞「21.11.11 文字列連結」(21-89 ページ)</p> <p>文字列または文字コードを文字列バッファに連結します。</p>



21.11.1 10 進文字列数値変換

項目	内容
概要	10 進数の文字列を整数値に変換する関数です。パラメータ 2 の 10 進数文字列を整数値に変換し、パラメータ 1 に格納します。
書式	<p><code>_decasc2bin (変換先アドレス, 変換元データバッファ)</code></p>  <p>パラメータ 1 : 内部デバイス、テンポラリアドレス パラメータ 2 : データバッファ</p>

記述例 1 (ビット長が 16 ビットの場合)

`_decasc2bin ([w:[#INTERNAL]LS0100], databuf0)`

databuf0 の内容が次のような場合

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

上記データを変換すると次のようになります。

	16bit
LS0100	1234

記述例 2 (ビット長が 32 ビットの場合)

```
_decasc2bin ([w:[#INTERNAL]LS0100], databuf0)
```

databuf0 の内容が次のような場合

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	35h	'5'
databuf0[5]	36h	'6'
databuf0[6]	37h	'7'
databuf0[7]	38h	'8'
databuf0[8]	00h	NULL

上記データを変換すると次のようになります。

	32bit
LS0100	12345678
LS0102	

重要

- 変換後のビット長が、D スクリプトエディタのビット長を超えるような場合はエラーになります。

例) スクリプトのビット長が 16 ビット長の場合

```
_strset (databuf0, "123456") // 誤って 6 桁の 10 進文字列をセット
_decasc2bin ([w:[#INTERNAL]LS0100], databuf0)
```

上記式を実行したときに、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 2 (文字列変換エラー) が発生します。ただし、エラーが発生するとメイン関数の先頭に戻るため、_decasc2bin() 実行直後に参照はできません。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻ります。)

- 変換する文字列に“0”～“9”以外の文字を含む文字列データを変換したときはエラーとなります。

例) スクリプトのビット長が 16 ビット長の場合

```
_strset (databuf0, "12AB") // 誤って 10 進数以外の文字列をセット
_decasc2bin ([w:[#INTERNAL]LS0100], databuf0)
```

上記式を実行したときに、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 2 (文字列変換エラー) が発生します。ただし、エラーが発生するとメイン関数の先頭に戻るため、_decasc2bin () 実行直後に参照はできません。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻ります。)

- エラーが発生した段階で処理が終了し、メイン関数の先頭に戻ります。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻ります。)

21.11.2 16 進文字列数値変換

項目	内容
概要	16 進数の文字列をバイナリデータに変換する関数です。パラメータ 2 の 16 進数文字列を整数値に変換し、パラメータ 1 に格納します。
書式	<p><code>_hexasc2bin (変換先アドレス, 変換元データバッファ)</code></p>  <p>パラメータ 1：内部デバイス、テンポラリアドレス パラメータ 2：データバッファ</p>

記述例 1 (ビット長が 16 ビットの場合)

`_hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)`

databuf0 の内容が次のような場合

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

上記データを変換すると次のようになります。

	16bit
LS0100	1234h

記述例 2 (ビット長が 32 ビットの場合)

```
_hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)
```

databuf0 の内容が次のような場合

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	35h	'5'
databuf0[5]	36h	'6'
databuf0[6]	37h	'7'
databuf0[7]	38h	'8'
databuf0[8]	00h	NULL

上記データを変換すると次のようになります。

	32bit
LS0100	12345678h
LS0102	

重要

- 変換する文字列が 16 ビット、32 ビット以上のデータになるときはエラーになります。
 - 例) スクリプトのビット長が 16 ビット長の場合


```
_strset (databuf0, "123456")
_hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)
```

 上記式を実行したときに、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 2 (文字列変換エラー) が発生します。
- 変換する文字列に "0" ~ "9"、"A" ~ "F"、"a" ~ "f" の文字以外の文字を含む文字列データを変換したときはエラーとなります。
 - 例) スクリプトのビット長が 16 ビット長の場合


```
_strset (databuf0, "123G")
_hexasc2bin ([w:[#INTERNAL]LS0100], databuf0)
```

 上記式を実行したときに、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 2 (文字列変換エラー) が発生します。
- エラーが発生した段階で処理が終了し、メイン関数の先頭に戻ります。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻ります。)

21.11.3 内部デバイスからデータバッファへ

項目	内容
概要	内部デバイスに格納されている文字列データを 1 バイトずつデータバッファに文字列数分コピーします。パラメータ 2 で指定した文字列をパラメータ 3 の長さ分、パラメータ 1 のデータバッファに格納します。
書式	<p><code>_ldcopy (コピー先データバッファ , コピー元アドレス , ワード数)</code></p>  <p>パラメータ 1 : データバッファ パラメータ 2 : 内部デバイス パラメータ 3 : 整数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 1 ~ 1024 です。</p>

記述例 1

`_ldcopy (databuf0, [w:[#INTERNAL]LS0100], 4)`

	16bit
LS0100	31h
LS0101	32h
LS0102	33h
LS0103	34h

LS0100 ~ LS103 のデータを databuf0 から連続して 4 バイト書き込みます。内部デバイスは 1 バイト単位（下位ビット）で読み込まれます。

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

重要

- 内部デバイスの下位 1 バイトを読み出してデータバッファに指定データ数分書き込みます。
- パラメータ 3 に設定できる最大数は 1024 です。これ以上の値を設定した場合は、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 1 (文字列オーバーフロー) が発生します。
- 内部デバイスの上位バイトにデータがあった場合でも下位 1 バイトしかデータを読み出しません。
- エラーが発生した段階で処理が終了し、メイン関数の先頭に戻ります。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻りません。)

`_ldcopy (databuf0, [w:[#INTERNAL]LS0100], 4)`

	16bit
LS0100	3132h
LS0101	3334h
LS0102	3536h
LS0103	3738h

上記のようにデータが格納されていた場合は、下位 1 バイトのデータを読み出してデータバッファに書き込みます。

	8bit	
databuf0[0]	32h	'2'
databuf0[1]	34h	'4'
databuf0[2]	36h	'6'
databuf0[3]	38h	'8'
databuf0[4]	00h	NULL

21.11.4 データバッファから内部デバイスへ

項目	内容
概要	データバッファのオフセットから格納されている文字列データを1バイトずつ内部デバイスに文字列数分コピーします。 パラメータ2で指定した文字列のパラメータ3で指定したオフセット値からパラメータ4で指定した長さ分の文字列をパラメータ1で指定した内部デバイスに格納します。
書式	<p>_dlcopy (コピー先アドレス, コピー元データバッファ, コピー元のオフセット値, コピーする文字数)</p>  <p>パラメータ1: 内部デバイス パラメータ2: データバッファ パラメータ3: 数値、内部デバイス、テンポラリアドレス パラメータ3に設定できる範囲は0～1024です。 パラメータ4: 数値、内部デバイス、テンポラリアドレス パラメータ4に設定できる範囲は1～1024です。</p>

記述例 1

_dlcopy ([w:[#INTERNAL]LS0100], databuf0, 2, 4)

8bit	
databuf0[0]	31h
databuf0[1]	32h
databuf0[2]	33h
databuf0[3]	34h
databuf0[4]	35h
databuf0[5]	36h
databuf0[6]	37h
databuf0[7]	38h

databuf0のオフセット2から4バイト分抽出したデータをLS0100～LS0103に書き込みます。内部デバイスには1バイト単位で書き込まれます。

16bit	
LS0100	33h
LS0101	34h
LS0102	35h
LS0103	36h

重要

- データバッファから 1 バイト読み出して内部デバイスに書き込みます。このため、内部デバイスには下位 8 ビット (1 バイト) 分しか使用しません。上位 8 ビット (1 バイト) は、0 でクリアされます。
 - コピー元のオフセット値 + コピーする文字数がデータバッファを超えるような指定をした場合は、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 3 (文字列抽出エラー) が発生します。
 - エラーが発生した段階で処理が終了し、メイン関数の先頭に戻ります。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻りません。)
-

21.11.5 文字列操作エラーステータス

文字列操作を行った場合にエラーが発生した場合には、文字列操作エラーステータス [e:STR_ERR_STAT] にエラーがセットされます。[e:STR_ERR_STAT] が 0 の場合は正常であり、0 以外の値が格納されているとエラーです。文字列操作エラーステータス [e:STR_ERR_STAT] には、一番最後に発生したエラーが格納されます。文字列操作エラーステータスの設定は、D スクリプトツールボックスの「SIO ポート操作 / ラベル設定」で行います。文字列操作エラーには、次のようなエラーがあります。

エラー番号	エラー名称	内容
0	正常	エラー無し
1	文字列オーバーフロー	_strset(), _strlen(), _strcat(), _strmid(), _IO_READ_WAIT() 関数に 256 バイト以上の文字列を直接引数に与えた。 または、_strcat(), _ldcopy() 関数実行時、データバッファを超える文字列を作成した。 例) _strcat(databuf0, databuf1) databuf0 に 1020 バイトの文字列、databuf1 に 60 バイトの文字列があるときに上記関数を実行した (データバッファ 0 のサイズである 1024 バイトを超えるためエラーとなる)
2	文字列変換エラー	_hexasc2bin(), _decasc2bin() 関数に指定外の文字コードを与えた。 例) _hexasc2bin() の第 2 引数に 0 ~ 9, A ~ F, a ~ f 以外の文字コード (G など) をセットした
3	文字列抽出エラー	_strmid() 関数で指定した文字列より長い文字列を抽出しようとした。または、指定した文字列より大きいオフセット値を指定した。 例) _strmid(databuf0, "12345678", 2, 8) オフセット 2 から 8 文字分抽出しようとした

文字列操作エラーステータスを D スクリプト、グローバル D スクリプトで使用することはできません。誤って読み出した場合には、0 を読み込みます。

各関数実行時にエラーステータスに格納されます。

[e:STR_ERR_STAT] は、メイン関数の先頭でエラーステータス確認用のチェックを記述してください。次のように記述することでエラーが確認できます。

記述例

```

if ([e:STR_ERR_STAT] <> 0)           // エラーステータスをチェック
{
    set ([b:[#INTERNAL]LS005000])    // エラー表示用ランプのビットセット
}
endif

```

重要

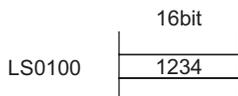
- エラーが発生した段階で処理が終了し、メイン関数の先頭に戻ります。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻りません。)

21.11.6 数値 10 進文字列変換

項目	内容
概要	整数値を 10 進文字列に変換する関数です。パラメータ 2 の整数値を 10 進数文字列に変換し、パラメータ 1 に格納します。
書式	<p><code>_bin2decasc (変換先データバッファ , 変換元アドレス)</code></p>  <p>パラメータ 1 : データバッファ パラメータ 2 : 内部デバイス、テンポラリアドレス</p>

記述例 1 (ビット長が 16 ビットの場合)

`_bin2decasc (databuf0, [w:[#INTERNAL]LS0100])`

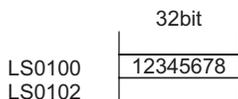


上記データを変換すると次のようになります。NULL (0x00) が付加されます。

	8bit	
databuf0[0]		31h '1'
databuf0[1]		32h '2'
databuf0[2]		33h '3'
databuf0[3]		34h '4'
databuf0[4]		00h NULL

記述例 2 (ビット長が 32 ビットの場合)

`_bin2decasc (databuf0, [w:[#INTERNAL]LS0100])`



上記データを変換すると次のようになります。

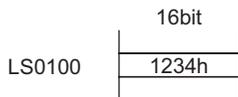
	8bit	
databuf0[0]		31h '1'
databuf0[1]		32h '2'
databuf0[2]		33h '3'
databuf0[3]		34h '4'
databuf0[4]		35h '5'
databuf0[5]		36h '6'
databuf0[6]		37h '7'
databuf0[7]		38h '8'
databuf0[8]		00h NULL

21.11.7 数値 16 進文字列変換

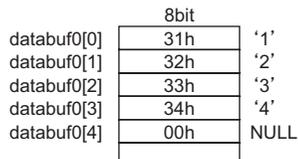
項目	内容
概要	バイナリデータを 16 進数文字列に変換する関数です。パラメータ 2 の整数値を 16 進数文字列に変換し、パラメータ 1 に格納します。
書式	<p><code>_bin2hexasc (変換先データバッファ , 変換元アドレス)</code></p>  <p>パラメータ 1: データバッファ パラメータ 2: 内部デバイス、テンポラリアドレス</p>

記述例 1 (ビット長が 16 ビットの場合)

`_bin2hexasc (databuf0, [w:[#INTERNAL]LS0100])`

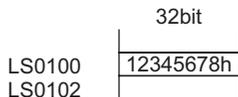


上記データを変換すると次のようになります。NULL (0x00) が付加されます。

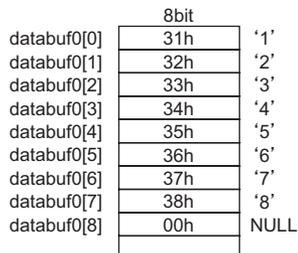


記述例 2 (ビット長が 32 ビットの場合)

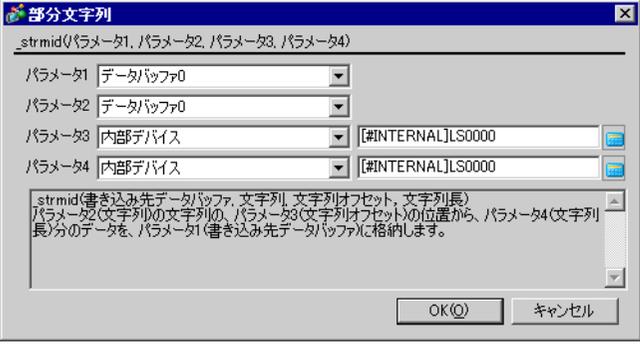
`_bin2hexasc (databuf0, [w:[#INTERNAL]LS0100])`



上記データを変換すると次のようになります。



21.11.8 部分文字列

項目	内容
概要	文字列の指定したオフセットから文字列長分抽出して別のバッファに取り出します。パラメータ 2 で指定した文字列のパラメータ 3 で指定したオフセット値からパラメータ 4 で指定した長さ分の文字列をパラメータ 1 で指定したデータバッファに格納します。
書式	<p><code>_strmid (書き込み先データバッファ, 文字列, 文字列オフセット, 文字列長)</code></p>  <p>パラメータ 1: データバッファ パラメータ 2: 文字列、データバッファ パラメータ 3: 数値、内部デバイス、テンポラリアドレス パラメータ 3 に設定できる範囲は 0 ~ 1024 です。 パラメータ 4: 数値、内部デバイス、テンポラリアドレス パラメータ 4 に設定できる範囲は 1 ~ 1024 です。</p>

記述例

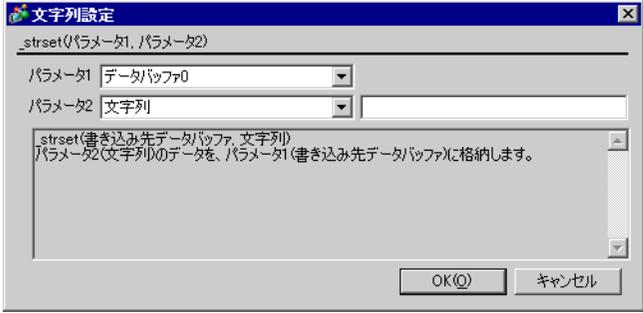
`_strmid (databuf0, "12345678", 2, 4)`

文字列 “12345678” のオフセット 2 から 4 バイト分抽出したデータを databuf0 に格納します。

	8bit	
databuf0[0]	33h	'3'
databuf0[1]	34h	'4'
databuf0[2]	35h	'5'
databuf0[3]	36h	'6'
databuf0[4]	00h	NULL

- | |
|----|
| 重要 |
|----|
- `_strmid ()` 関数で指定した文字列より長い文字列を抽出しようとしたり、指定した文字列より大きいオフセット値を指定した場合には、文字列エラーステータス [e:STR_ERR_STAT] のエラー番号 3 (文字列抽出エラー) が発生します。
 - エラーが発生した段階で処理が終了し、メイン関数の先頭に戻ります。(命令が呼び出された関数の中にある場合、呼び出し元の関数の呼び出された次の行に戻ります。)

21.11.9 文字列設定

項目	内容
概要	固定文字列をデータバッファに格納します。パラメータ 2 で指定した文字列をパラメータ 1 に格納します。
書式	<p><code>_strset (書き込み先データバッファ, 文字列)</code></p>  <p>パラメータ 1: データバッファ パラメータ 2: 文字列、数値 (文字コード) パラメータ 2 に設定できる範囲は 0、1 ~ 255 です。</p>

記述例

```
_strset (databuf0, "ABCD")
```

データバッファには次のように格納されます。

	8bit	
databuf0[0]	41h	'A'
databuf0[1]	42h	'B'
databuf0[2]	43h	'C'
databuf0[3]	44h	'D'
databuf0[4]	00h	NULL

重要

- 設定できる文字列は最大 255 文字までです。これ以上の文字列を設定する場合は、一度別のデータバッファに文字列を格納して、文字列連結関数 (`_strcat`) を使用して連結してください。
- データバッファのクリアを行う場合には、空文字列 "" または数値 0 を設定することでバッファをクリアできます。

例: `_strset (databuf0, "")`

`_strset (databuf0, 0)`

21.11.10 文字列長さ

項目	内容
概要	格納されている文字列の長さを得ます。パラメータ 2 にデータバッファを指定した場合には、NULL 文字までの文字の個数をパラメータ 1 に格納します。(NULL 文字は含みません。)
書式	<p><code>_strlen (文字列長書き込み先アドレス, 文字列)</code></p>  <p>パラメータ 1 : 内部デバイス、テンポラリアドレス パラメータ 2 : 文字列、データバッファ</p>

記述例 1

```
_strlen ([w:[#INTERNAL]LS0100], "ABCD")
```

上記式を実行すると次のように、LS0100 に文字列長さが書き込まれます。



記述例 2

```
_strlen ([t:0000], databuf0)
```

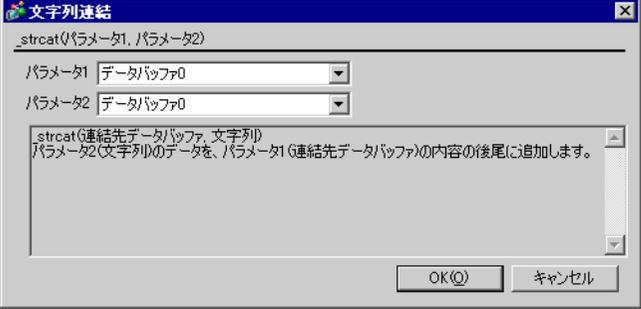
databuf0 の内容が次のような場合

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

上記式を実行すると次のように、[t:0000] に文字列長さが書き込まれます。



21.11.11 文字列連結

項目	内容
概要	文字列または文字コードを文字列バッファに連結します。パラメータ 2 で指定した文字列または文字コードをパラメータ 1 の後ろに連結します。
書式	<p><code>_strcat (連結先データバッファ , 文字列)</code></p>  <p>パラメータ 1 : データバッファ パラメータ 2 : 文字列、数値 (文字コード)、データバッファ パラメータ 2 に設定できる範囲は 0、1 ~ 255 です。</p>

記述例 1

`_strcat (databuf0, "ABCD")`

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	00h	NULL

上記の状態に “ABCD” を連結する場合は以下の結果となります。NULL (0x00) が付加されます。

	8bit	
databuf0[0]	31h	'1'
databuf0[1]	32h	'2'
databuf0[2]	33h	'3'
databuf0[3]	34h	'4'
databuf0[4]	41h	'A'
databuf0[5]	42h	'B'
databuf0[6]	43h	'C'
databuf0[7]	44h	'D'
databuf0[8]	00h	NULL

重要

- 設定できる文字列は最大 255 文字までです。
- パラメータ 2 に空文字列 "" または数値 0 を設定した場合、パラメータ 1 のデータバッファは変化しません。
 例 : `_strcat (databuf0, "")`
`_strcat (databuf0, 0)`

21.12 演算例

21.12.1 論理演算子を用いた計算例

論理演算子をもちいた計算例を以下に示します。

$((100 > 99) \text{ and } (200 <> 100))$

結果：ON

$((100 > 99) \text{ and } (200 <> 200))$

結果：OFF

$((100 > 99) \text{ or } (200 <> 200))$

結果：ON

$((100 < 99) \text{ or } (200 <> 200))$

結果：OFF

$\text{not } (100 > 99)$

結果：OFF

$\text{not } (100 < 99)$

結果：ON

$[w:D200] < 10$

結果：D200 が 10 より小さければ真

$\text{not } [w:D200]$

結果：D200 が 0 のとき真

$([w:D200] == 2) \text{ or } ([w:D200] == 5)$

結果：D200 が 2 または 5 のとき真

$([w:D200] < 5) \text{ and } ([w:D300] < 8)$

結果：D200 が 5 より小さくかつ D300 が 8 より小さいとき真

$[w:D200] < 10$

結果：D200 が 10 より小さければ真

$\text{not } [w:D200]$

結果：D200 が 0 のとき真

$([w:D200] == 2) \text{ or } ([w:D200] == 5)$

結果：D200 が 2 または 5 のとき真

$([w:D200] < 5) \text{ and } ([w:D300] < 8)$

結果：D200 が 5 より小さくかつ D300 が 8 より小さいとき真

21.12.2 ビット操作を用いた計算例

ビット操作をもちいた計算例を以下に示します。

```
[ w:D200 ] << 4
```

結果：D200 の内容を 4 ビット左にシフトする。

```
[ w:D200 ] >> 4
```

結果：D200 の内容を 4 ビット右にシフトする。

データ形式 BIN、D301 に 12(0000Ch) を格納

```
[ w:D200 ] = [ w:D300 ] >> [ w:D301 ]
```

結果：D300 の内容を 12 ビット右にシフトして D200 に代入する。

```
[ w:D200 ] << 4
```

結果：D200 の内容を 4 ビット左にシフトする。

```
[ w:D200 ] >> 4
```

結果：D200 の内容を 4 ビット右にシフトする。

データ形式 BIN、D310 に 12(0000Ch) を格納

```
[ w:D200 ] = [ w:D300 ] >> [ w:D310 ]
```

結果：D300 の内容を 12 ビット右にシフトして D200 に代入する。

ビットの論理積

0 & 0	結果：0
0 & 1	結果：0
1 & 1	結果：1
0x1234 & 0xF0F0	結果：0x1030

ビットの論理和

0 0	結果：0
0 1	結果：1
1 1	結果：1
0x1234 0x9999	結果：0x9BBD

ビットの排他的論理和

0 ^ 0	結果：0
0 ^ 1	結果：1
1 ^ 1	結果：0

ビットの 1 の補数 (データ形式 Bin16 + の場合)

~ 0	結果：0xFFFF
~ 1	結果：0xFFFE

21.12.3 条件分岐を用いた計算例

制御の流れを分岐させる、if-endif、if-else-endif を以下に示します。

if-endif

```
if ( 条件 )
  { 処理 1 }
endif
```

条件が成立した場合は処理 1 を実行し、成立しなかった場合は処理 1 を無視します。

例)

```
if ( [ w:D200 ] < 5 )
  {
    [ w:D100 ] = 1
  }
endif
```

D200 のデータが 5 未満の場合、D100 に 1 を代入します。

if-else-endif

```
if ( 条件 )
  { 処理 1 }
else
  { 処理 2 }
endif
```

条件が成立した場合は処理 1 を実行し、成立しなかった場合は処理 2 を実行します。

例)

```
if ( [ w:D200 ] < 5 )
  {
    [ w:D100 ] = 1
  }
else
  {
    [ w:D100 ] = 0
  }
endif
```

D200 のデータが 5 未満の場合、D100 に 1 を代入し、それ以外は D100 に 0 を代入します。

21.12.4 オフセットアドレスを用いた計算例

オフセット指定 : [w:D00100]#[t:0000] を用いた特殊な計算例を以下に示します。

スクリプト設定 : 16 ビット符号無しで [t:0000]= 65526 の時、指定アドレスは [w:D00090] となる。

$$100 + 65526 = 64(\text{Hex}) + \text{FFF6}(\text{Hex}) = \underline{\underline{1005A}}(\text{Hex}) \quad 005A(\text{Hex}) = 90$$

下位 16 ビットが有効

スクリプト設定 : 16 ビット符号有り度 [t:0000]= -10 の時、指定アドレスは [w:D00090] となる。

$$100 + (-10) = 64(\text{Hex}) + \text{FFF6}(\text{Hex}) = \underline{\underline{1005A}}(\text{Hex}) \quad 005A(\text{Hex}) = 90$$

下位 16 ビットが有効

スクリプト設定 : 32 ビット符号無しで [t:0000]= 4294901840 の時、指定アドレスは [w:D00180] となる。

$$100 + 4294901840 = 64(\text{Hex}) + \text{FFFF0050}(\text{Hex}) = \text{FFFF}\underline{\underline{00B4}}(\text{Hex}) \quad 00B4(\text{Hex}) = 180$$

下位 16 ビットが有効

スクリプト設定 : 32 ビット符号有り度 [t:0000]= -65456 の時、指定アドレスは [w:D00180] となる。

$$100 + (-65456) = 64(\text{Hex}) + \text{FFFF0050}(\text{Hex}) = \text{FFFF}\underline{\underline{00B4}}(\text{Hex}) \quad 00B4(\text{Hex}) = 180$$

下位 16 ビットが有効

重要

- オフセットアドレスはスクリプトのビット長、データ形式の設定に関係なく、常に 16 ビット Bin で扱われます。もし演算結果が 16 ビット (最大値 : 65535) を超えるような場合、15 ビット目までを有効なビットとし扱い、16 ビット目以上は切り捨てられます。

21.13 命令一覧

項目	命令、関数など	D スクリプト/ グローバルD スクリプト	拡張スクリプト
データ形式	Bin、BCD		Bin のみ
ビット長	16 ビット、32 ビット		
符号	有無		
トリガ	タイマ設定		×
	ビット立ち上がり		×
	ビット立ち下がり		×
	ビット両動作		×
	条件式成立時		×
	条件式不成立時		×
描画	画面呼び出し		×
	ドット		
	直線		
	円		
	四角		
演算子	加算 +		
	減算 -		
	余り %		
	掛け算 *		
	割り算 /		
	代入 =		
比較	論理積 and		
	論理和 or		
	否定 not		
	未満 <		
	以下 <=		
	等しくない <>		
	超える >		
	以上 >=		
	等しい =		

次のページに続きます。

項目	命令、関数など	D スクリプト/ グローバルD スクリプト	拡張スクリプト
メモリ操作	メモリコピー memcpy ()		
	メモリ初期化 memset ()		
	メモリコピー (可変指定) _memcpy_EX ()		
	メモリ初期化 (可変指定) _memset_EX ()		
	オフセットアドレス		
	メモリシフト		
	メモリアリキ		
	メモリ検索		
	メモリ比較		
ビット操作	左シフト <<		
	右シフト >>		
	論理積 &		
	論理和		
	排他的論理和 ^		
	1 の補数		
	セットビット set ()		
	クリアビット clear ()		
	トグルビット toggle ()		
記述式	if ()		
	if () else		
	loop (), break		
	loop () 無限ループ	×	
アドレス	ビットアドレス		内部デバイス
	ワードアドレス		内部デバイス
	テンポラリワークアドレス		1
定数	Dec, Hex, Oct		

次のページに続きます。

項目	命令、関数など	D スクリプト/ グローバル D スクリプト	拡張スクリプト
SIO 関数	受信 IO_READ ([p:SIO])		
	送信 IO_WRITE ([p:SIO])		
	拡張受信 _IO_READ_EX ()	×	
	拡張送信 _IO_WRITE_EX ()	×	
	待ち受け受信関数 _IO_READ_WAIT ()	×	
	コントロール [c:EXT_SIO_CTRL]		
	ステータス [s:EXT_SIO_STAT]		
	受信データ数 [r:EXT_SIO_RCV]		
	待機関数 _wait ()	×	
文字列操作	文字列	×	
	データバッファ databuf0、databuf1、 databuf2、databuf3	×	
	文字列書き込み _strset ()	×	
	データバッファから内部デバ イスにコピー _dlcopy ()	×	
	内部デバイスからデータバッ ファにコピー _ldcopy ()	×	
	16 進文字列数値変換関数 _hexasc2bin ()	×	
	10 進文字列数値変換関数 _decasc2bin ()	×	
	数値 16 進文字列変換 _bin2hexasc ()	×	
	数値 10 進文字列変換 _bin2decasc ()	×	
	文字列長さ取得関数 _strlen ()	×	
	文字列連結関数 _strcat ()	×	
	部分文字列関数 _strmid ()	×	
ステータス [e:STR_ERR_STAT]	×		

次のページに続きます。

項目	命令、関数など	D スクリプト/ グローバルD スクリプト	拡張スクリプト
関数	呼出 Call		
	return	×	
CF ファイル 操作	CSV ファイルリード		
	ファイルリスト出力 _CF_dir ()		
	ファイルリード _CF_read ()		
	ファイルライト _CF_write ()		
	ファイル削除 _CF_delete ()		
	ファイル名変更 _CF_rename ()		
プリンタ操作	COM ポート出力 IO_WRITE ([p:PRN])		
デバッグ	_debug ()		

1 テンポラリアドレスは従来の D スクリプト、グローバル D スクリプトとは別に存在します。

