

eV+3

ユーザーズマニュアル

おことわり

- 本マニュアルの内容の一部または全部を無断で複写、複製、転載することを禁じます。
- 本マニュアルの内容に関しては、改良のため予告なしに仕様などを変更することがあります。
あらかじめご了承ください。
- 本マニュアルの内容に関しては、万全を期しておりますが、万一ご不審な点や誤りなど、お気づきのことがありましたら、お手数ですが、弊社支店または営業所までご連絡ください。
その際、巻末記載の Man.No.（マニュアル No.）も併せてお知らせください。

商標

- Microsoft、Windows は、米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。
 - ODVA、CIP、DeviceNet、EtherNet/IP は ODVA の商標です。
 - SD、SDHC ロゴは、SD-3C, LLC の商標です。 
- その他、本文中に掲載している会社名および製品名は、それぞれ各社の商標または登録商標です。

著作権

スクリーンショットはマイクロソフトの許可を得て使用しています。

表記についての留意事項

本マニュアルでは、「Automation Control Environment（オートメーション コントロール エンバイロメント）」をすべて「ACE」と略して記述しています。

ご承諾事項

「当社商品」について特別の合意がない場合には、お客様のご購入先にかかわらず、本ご承諾事項記載の条件を適用いたします。

● 定義

本ご承諾事項中の用語の定義は次のとおりです。

- 「当社商品」：「当社」の FA システム機器、汎用制御機器、センシング機器、電子・機構部品
- 「カタログ等」：「当社商品」に関する、ベスト制御機器オムロン、電子・機構部品総合カタログ、その他のカタログ、仕様書、取扱説明書、マニュアル等であって電磁的方法で提供されるものも含みます。
- 「利用条件等」：「カタログ等」に記載の、「当社商品」の利用条件、定格、性能、動作環境、取り扱い方法、利用上の注意、禁止事項その他
- 「お客様用途」：「当社商品」のお客様におけるご利用方法であって、お客様が製造する部品、電子基板、機器、設備またはシステム等への「当社商品」の組み込み、または利用を含みます。
- 「適合性等」：「お客様用途」での「当社商品」の (a) 適合性、(b) 動作、(c) 第三者の知的財産の非侵害、(d) 法令の遵守および (e) 各種規格の遵守

● 記載事項のご注意

「カタログ等」の記載内容については次の点をご理解ください。

- 定格値および性能値は、単独試験における各条件のもとで得られた値であり、各定格値および性能値の複合条件のもとで得られる値を保証するものではありません。
- 参考データはご参考として提供するもので、その範囲で常に正常に動作することを保証するものではありません。
- 利用事例はご参考ですので、「当社」は「適合性等」について保証いたしかねます。
- 「当社」は、改善や当社都合等により、「当社商品」の生産を中止し、または「当社商品」の仕様を変更することがあります。

● ご利用にあたってのご注意

ご採用およびご利用に際しては次の点をご理解ください。

- 定格・性能ほか「利用条件等」を遵守しご利用ください。
 - お客様ご自身にて「適合性等」をご確認いただき、「当社商品」のご利用の可否をご判断ください。「当社」は「適合性等」を一切保証いたしかねます。
 - 「当社商品」がお客様のシステム全体の中で意図した用途に対して、適切に配電・設置されていることをお客様ご自身で、必ず事前に確認してください。
 - 「当社商品」をご使用の際には、(i) 定格および性能に対し余裕のある「当社商品」のご利用、冗長設計などの安全設計、(ii) 「当社商品」が故障しても、「お客様用途」の危険を最小にする安全設計、(iii) 利用者に危険を知らせるための、安全対策のシステム全体としての構築、(iv) 「当社商品」および「お客様用途」の定期的な保守、の各事項を実施してください。
 - 「当社」はDDoS攻撃（分散型DoS攻撃）、コンピュータウイルスその他の技術的な有害プログラム、不正アクセスにより、「当社商品」、インストールされたソフトウェア、またはすべてのコンピュータ機器、コンピュータプログラム、ネットワーク、データベースが感染したとしても、そのことにより直接または間接的に生じた損失、損害その他の費用について一切責任を負わないものとします。
- お客様ご自身にて、（i）アンチウイルス保護、（ii）データ入出力、（iii）紛失デー

タの復元、（iv）「当社商品」またはインストールされたソフトウェアに対するコンピュータウイルス感染防止、（v）「当社商品」に対する不正アクセス防止についての十分な措置を講じてください。

- 「当社商品」は、一般工業製品向けの汎用品として設計製造されています。従いまして、次に掲げる用途での使用は意図しておらず、お客様が「当社商品」をこれらの用途に使用される際には、「当社」は「当社商品」に対して一切保証をいたしません。ただし、次に掲げる用途であっても「当社」の意図した特別な商品用途の場合や特別の合意がある場合は除きます。
 - （a）高い安全性が必要とされる用途（例：原子力制御設備、燃焼設備、航空・宇宙設備、鉄道設備、昇降設備、娯楽設備、医用機器、安全装置、その他生命・身体に危険が及ぶ用途）
 - （b）高い信頼性が必要な用途（例：ガス・水道・電気等の供給システム、24時間連続運転システム、決済システムほか権利・財産を取扱う用途など）
 - （c）厳しい条件または環境での用途（例：屋外に設置する設備、化学的汚染を被る設備、電磁的妨害を被る設備、振動・衝撃を受ける設備など）
 - （d）「カタログ等」に記載のない条件や環境での用途
- 上記の（a）から（d）に記載されている他、「本カタログ等記載の商品」は自動車（二輪車含む。以下同じ）向けではありません。自動車に搭載する用途には利用しないで下さい。自動車搭載用商品については当社営業担当者にご相談ください。

● 保証条件

「当社商品」の保証条件は次のとおりです。

- 保証期間 ご購入後1年間といたします。
（ただし「カタログ等」に別途記載がある場合を除きます。）
- 保証内容 故障した「当社商品」について、以下のいずれかを「当社」の任意の判断で実施します。
 - （a）当社保守サービス拠点における故障した「当社商品」の無償修理
（ただし、電子・機構部品については、修理対応は行いません。）
 - （b）故障した「当社商品」と同数の代替品の無償提供
- 保証対象外 故障の原因が次のいずれかに該当する場合は、保証いたしません。
 - （a）「当社商品」本来の使い方以外のご利用
 - （b）「利用条件等」から外れたご利用
 - （c）本ご承諾事項「ご利用にあたってのご注意」に反するご利用
 - （d）「当社」以外による改造、修理による場合
 - （e）「当社」以外の者によるソフトウェアプログラムによる場合
 - （f）「当社」からの出荷時の科学・技術の水準では予見できなかった原因
 - （g）上記のほか「当社」または「当社商品」以外の原因（天災等の不可抗力を含む）

● 責任の制限

本ご承諾事項に記載の保証が、「当社商品」に関する保証のすべてです。

「当社商品」に関連して生じた損害について、「当社」および「当社商品」の販売店は責任を負いません。

● 輸出管理

「当社商品」または技術資料を、輸出または非居住者に提供する場合は、安全保障貿易管理に関する日本および関係各国の法令・規制を遵守ください。お客様が法令・規則に違反する場合には、「当社商品」または技術資料をご提供できない場合があります。

労働安全衛生規則第36条第31号および32号において、産業用ロボットの教示等を行う作業や検査・修理・調整等を行う作業は、労働安全衛生法で定める「危険又は有害な業務」に該当します。労働安全衛生法第59条により、事業者は労働者に対し「安全又は衛生のための特別の教育」を行う義務があります。

お客様の責任において、リスクアセスメントの結果から適切な安全対策を実施してください。ロボットセーフティガイドおよび弊社ロボットシステム商品関連資料の記載事項を全て遵守しても、産業用ロボットが引き起こす人身事故または機器の損傷を回避する保証はできません。

第1章: はじめに	13
1.1 対象となる読者	13
1.2 関連マニュアル	13
1.3 システムの概要	13
eV+ オペレーティングシステム	14
eV+ 言語	14
1.4 eV+ キーワード	15
1.5 モニタコマンドプログラム と V+ プログラム	16
1.6 eV+ のファイル管理	16
ファイル名の要件	16
ファイルの種類	17
プログラムとサブルーチン	18
第2章: キーワードの使用方法	19
2.1 関数キーワードの使用	19
関数キーワードの構文	19
変数割り当てとデータタイプ	19
関数をキーワードパラメータとして使用	19
式で使用する関数キーワード	20
2.2 モニタコマンドキーワードの使用	20
モニタコマンドキーワードの構文	20
2.3 プログラム命令キーワードの使用	21
プログラム命令キーワードの構文	21
2.4 システムパラメータキーワードの使用	22
使用可能なシステムパラメータ	22
システムパラメータの表示と設定	23
2.5 システムスイッチキーワードの使用	24
使用可能なシステムスイッチ	24
システムスイッチの表示と設定	25
第3章: eV+システムの動作	27
3.1 システムメッセージ	27
メッセージタイプ	27

メッセージ処理	27
3.2 デジタル I/O 制御	28
I/O 使用上の留意事項	28
基本的な I/O 制御	28
ソフトウェア信号	29
3.3 ディスク I/O の操作	29
論理ユニット番号	29
I/O 操作のエラーステータス	29
論理ユニットのアタッチとデタッチ	30
ディスクデバイスのアタッチとデタッチ	31
I/O デバイスの読み込みと書き込み	31
入力ウェイトモード	32
出力ウェイトモード	32
ディスクファイル操作	32
より高度なディスク操作	35
第4章: モーション制御操作	39
4.1 モーション制御の概要	39
4.2 ジョイント補間モーションと直線モーション	39
4.3 安全な接近（アプローチ）と離脱（ディパーチャ）	39
4.4 連続パス軌道	40
4.5 連続パスの中断	41
4.6 手続き型動作	42
4.7 モーション制御タイミングの留意事項	43
4.8 ロボットの速度と性能	43
ロボット速度	43
ロボットエンドエフェクタ／ツール先端速度に関する留意事項	44
ロボットの性能	44
4.9 モーション修飾子	44
第5章: 変数とデータタイプ	47
5.1 変数の作成	47
5.2 変数の分類	47
変数のスコープ	47
外部変数	49
グローバル変数	50
ローカル変数	51

オート変数	52
5.3 変数名の要件	52
5.4 変数の初期化	53
5.5 変数演算子	53
代入演算子	53
算術演算子	54
関係演算子	54
論理演算子	55
文字列演算子	55
演算子の優先順位	55
5.6 数値表現	56
数式	57
5.7 文字列のデータタイプ	57
ASCII 値	57
5.8 実数型整数のデータタイプ	58
実数値と整数値の範囲	58
5.9 論理定数	58
5.10 ロケーションデータタイプ	59
ロケーション変数の作成	59
ロケーション変数の修正	59
5.11 配列	61
グローバル配列へのアクセス制限	62
5.12 変数のコンテキスト	62
変数のコンテキスト指定の解釈	64
第6章: モニタコマンドプログラム	65
6.1 Monitor Command Program の概要	65
6.2 モニタコマンドプログラム 名	65
6.3 モニタコマンドプログラム の内容	65
6.4 モニタコマンドプログラム の実行	66
モニタコマンドプログラム からモニタコマンドプログラムを実行する	66
モニタコマンドプログラム の自動スタート	66
モニタコマンドプログラム の処理	67
第7章: V+ プログラム	69
7.1 V+ プログラム のフォーマット	69

プログラム行	69
プログラム構造	70
7.2 V+ プログラムの実行	70
プログラムタスク	71
V+ プログラム の処理	72
7.3 プログラム制御	73
無条件分岐	73
条件分岐	74
ループ構造	76
システムメモリへのプログラムと変数の読み込み	78
システムメモリからプログラムと変数を削除	78
7.4 V+ プログラムの割り込み	78
WAIT プログラム割り込み	79
WAIT.EVENT プログラム割り込み	79
REACT、REACTI プログラム割り込み	80
REACTE プログラム割り込み	81
HALT、STOP、PAUSE プログラム割り込み	81
追加のプログラム割り込みキーワード	81
プログラム割り込みの例	81
7.5 V+ プログラム のスタック	83
V+ プログラムのスタック要件	83
7.6 V+ プログラム 間の情報交換	84
グローバル変数を使用した情報の交換	85
ソフトウェア信号を使用した情報の交換	85
プログラム引数リストを使用した情報の交換	85
7.7 再入可能プログラムと再帰的プログラム	87
再入可能プログラム	87
再帰的プログラム	88
7.8 非同期処理	88
エラートラップ	89
第8章: T20 ペンダントのプログラミング	91
8.1 T20 ペンダントのアタッチとデタッチ	91
8.2 T20 ペンダントディスプレイへの書き込み	91
8.3 ユーザー入力の検出	92
ペンダントキー押下の検出	92
8.4 その他のペンダントプログラミング機能	93

付録	95
A.1 警告メッセージ、情報メッセージ、エラーメッセージ	95
システムの情報メッセージ	95

第1章: はじめに

本項では、eV+ プログラミング言語と制御システムの特徴と基本的な機能について説明します。

1.1 対象となる読者

このマニュアルの対象読者は、次の職種であって、一般的なプログラミング技法およびロボット制御方法に対する知識も有している方です。

- FA システムの導入担当者
- FA システムの設計担当者
- FA システムの設置/保守担当者
- FA システムおよび施設の管理担当者

1.2 関連マニュアル

参考資料として次の関連マニュアルを使用します。

表1-1. 関連マニュアル

マニュアル	説明
『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』	eV+ キーワードの使用方法和機能について説明します。
『Sysmac Studio ロボット統合システム構築機能オペレーションマニュアル ロボット統合CPUユニット編 (SBCE-464)』	ロボット統合CPU ユニットを使用したロボット統合システムを設定するための、Sysmac Studioの操作手順や機能を説明します。
ロボットユーザーズマニュアル	使用するロボットのユーザーズマニュアルです。
『ティーチングペンダント T20 ユーザーズマニュアル (SBCE-392)』	オプションのT20手動制御ペンダントの使用方法について記載されています。
『NJ シリーズ ロボット統合 CPU ユニット ユーザーズマニュアル (SBCE-436)』	CPUユニットの設定、操作、およびオムロン製ロボットの制御に関するプログラミングコンセプトについて記載されています。
『ロボットセーフティガイド (SBCE-381)』	オムロン製産業用ロボットの安全に関する情報が記載されています。

1.3 システムの概要

eV+ は、OMRON 産業ロボット用インタプリタプログラミング言語および制御システムです。このシステムは、ロボットの動作、I/O、ビジョン、ファイル操作、他のデバイスとの通信をサ

ポートするさまざまなキーワードを提供します。並列タスクのプログラムをリアルタイムで解釈し、先読み処理するため、連続パス軌道の生成、オンラインでのプログラムの生成、デバッグ、修正が可能です。

eV+ システムを NJ シリーズ ロボット統合 CPU ユニットと共に使用すれば、IEC 61131-3 プログラミング言語仕様に基づく I/O制御、ロジック制御、安全制御およびモーション制御ファンク機能を作成できます。V+ プログラムの実行に相互作用する共有変数、共有信号、ファンクションブロックを追加することにより、Sysmac Studio ソフトウェアで制御ソリューション全体を開発できます。

eV+ オペレーティングシステムは、オムロン製ロボットで動作し、プログラミングやランタイムの機能性を高めるため、他のシステムオブジェクトと通信できます。次の図にシステム全体の概要を示します。

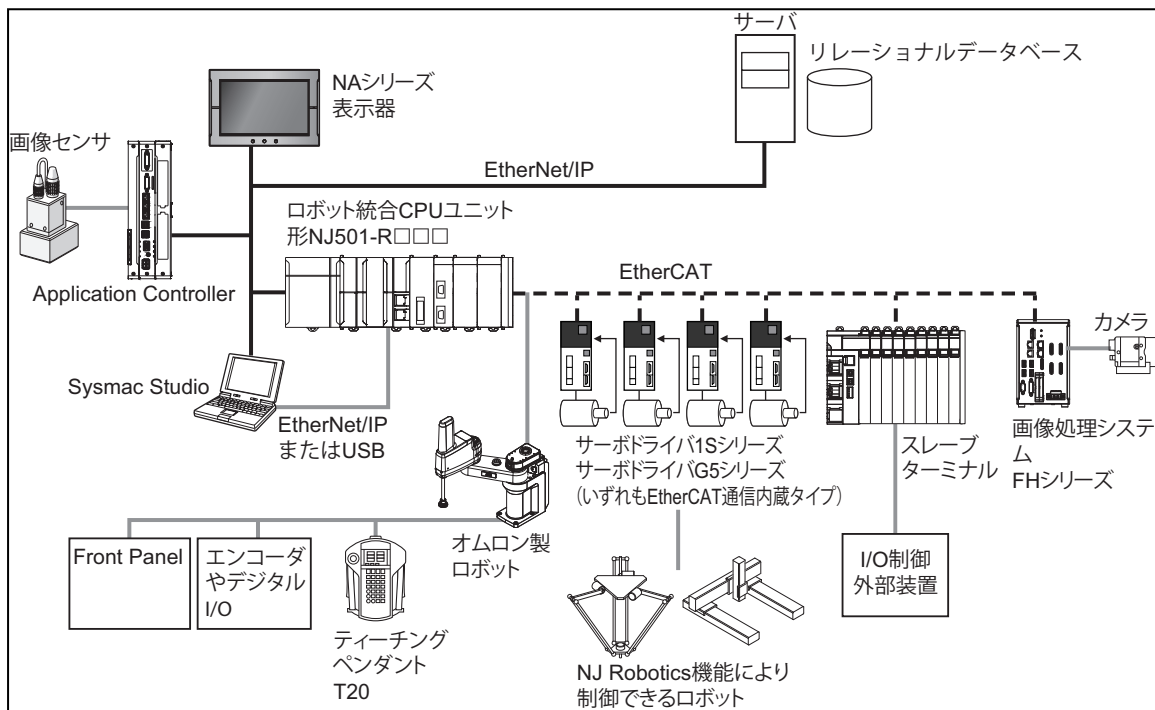


図1-1. システムの概要

eV+ オペレーティングシステム

eV+オペレーティングシステムは、ロボットコントローラと NJシリーズロボット統合CPUユニットの電源を ON にすると自動的に起動します。オペレーティングシステムは、アプリケーションプログラムからの命令、ワークセル周辺デバイスからの入力、ペンダントからのオペレータ入力を受け付けます。オペレーティングシステムが実行するタスクには次の事項が含まれます。

- ・アプリケーションプログラム実行の管理
- ・ストレージデバイスの情報の入出力管理
- ・コントローラと接続された外部デバイスのモニタリング
- ・処理中に発生したエラーの報告

eV+ 言語

eV+ 言語は、制御、データ操作、他のアプリケーション固有の機能を提供するキーワードから構成されています。これらのキーワードについては、本マニュアルの各ステートメントの例で構文

とともに説明します。各ステートメントは [.END] ステートメントに達するまで連続で実行されます。これはプログラムサイクルと呼ばれます。キーワードを使用して、V+ プログラム を作成したり、モニタウィンドウで各コマンドを発行できます。その他の構文とステートメントの例については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

重要: ホストシステムとして NJ シリーズロボット統合CPU ユニット を含まないシステムでキーワードを発行した場合、"Command not supported"エラーが返されます。

1.4 eV+ キーワード

eV+ キーワードは、ロボットベースのアプリケーションでさまざまな操作を実行するために使用する命令セットです。キーワードを使用して、次に示すプログラム機能を実行するためのステートメントを作成します。

- ロボット動作の制御
- エラー処理
- 算術関数
- システムパラメータとスイッチの操作
- データ管理
- I/O 制御と外部デバイス通信
- 論理条件と他の条件の評価
- サブルーチンの実行と制御
- さまざまなシステム関数の実行
- T20 ペンダント関数

キーワードの種類に応じて、プログラム作成に使用するものと、Sysmac Studio のモニタウィンドウに個別に表示するものがあります。

次に示すように、キーワードは 6 種類のカテゴリーに分けられます。

表1-2. eV+ キーワードの種類

キーワードの種類	用途
関数キーワード	eV+ オペレーティングシステムの値を返すために使用されます。 詳細については、関数キーワードの使用 19 ページを参照してください。
モニタコマンドキーワード	モニタウィンドウで個々の演算を実行したり、Monitor Command Program を作成したりするために使用します。 詳細については、モニタコマンドキーワードの使用 20 ページを参照してください。
その他のキーワード	[SPEED] プログラム命令キーワードを使用するとき、単位の指定に使用します。 詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。
プログラムコマキーワード	V+ プログラム のコマンド操作に使用します。 詳細については、プログラム命令キーワードの使用 21 ページを参照してください。

キーワードの種類	用途
システムパラメータ キーワード	V+ プログラム またはモニタウィンドウでシステムパラメータを操作するとき に使用します。 詳細については、システムパラメータキーワードの使用 22 ページを参照して ください。
システムスイッチ キーワード	V+ プログラム またはモニタウィンドウでシステムスイッチを操作するとき に使用します。 詳細については、システムスイッチキーワードの使用 24 ページを参照して ください。

キーワードと構文

キーワードをeV+ オペレーティングシステムで機能させるためには、指定の構文に従って発行する必要があります。通常、この構文は、一つのキーワードと複数の関連パラメータで構成され、パラメータはそれぞれ、コンマ、角カッコ、丸カッコなどで区切られ、順番が指定されています。この構文については、本マニュアルと『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』に詳しく記載されています。次に、キーワードの構文の例を示します。

```
SET loc_name = SHIFT(loc_value BY 5, 5, 5)
```

Sysmac Studio を使用して V+ プログラム を作成する場合、文字の入力時にチェックとフォーマット化が行われます。詳細については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』を参照してください。

1.5 モニタコマンドプログラム と V+ プログラム

eV+ オペレーティングシステムでは、モニタコマンドプログラム と V+ プログラム の 2 種類のプログラムを実行できます。

V+ プログラム には、実行時間中にロボットを制御するロジック、モーション制御およびビジョンのキーワードが含まれます。詳細については、「V+ プログラム 69 ページを参照してください。

モニタコマンドプログラム は、ファイルの読み込み、デフォルトのディレクトリパスの変更、システム起動後のメインの V+ プログラムの実行など、システムレベルの機能実行に使用されます。詳細については、「モニタコマンドプログラム 65 ページを参照してください。

1.6 eV+ のファイル管理

eV+ オペレーティングシステムのファイル管理システムは、他のオペレーティングシステムと類似しています。サブディレクトリ内の各ファイルには、固有の名前を付ける必要があります。ファイルの種類 17 ページに示すように、eV+ システムのファイルには、さまざまなファイル拡張子が使用されています。

Sysmac Studio から、あるいは固有のキーワードを発行して、ファイルの新規作成ができます。詳細については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』と『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

ファイル名の要件

次に eV+ のファイル名についての要件を示します。

- ・ ファイル名には最大 8 文字まで使用できます。
- ・ ファイル名には、1～3 文字長の拡張子（ファイルの種類）を指定する必要があります。
詳細については、ファイルの種類 17 ページを参照してください。
- ・ ファイル名には、英数文字とアンダースコアのみ使用できます。それ以外の特殊文字は使用できません。
- ・ ファイル名に空白文字は使用できません。
- ・ キーワードは、ファイル名として使用できません。ファイル名をキーワードと同じ名前にしないでください。

注: ファイル名は、大文字と小文字を区別しません。

ファイルの種類

eV+ オペレーティングシステムは、複数の種類のファイルを使用します。システム開発、試運転、デバッグ、他の通常運転で使用するファイルの種類を下表に示します。

重要: 通常の使用では、これらのファイルに直接アクセスしたり、編集する必要はありません。

表1-3. eV+ ファイルの種類の説明

ファイルの種類	説明
.v2	V+ プログラム プログラムやサブルーチンが参照するグローバル変数を格納します。 特定のプログラムが（直接または間接的に）参照するすべてのサブルーチンを格納します。 STORE キーワードを使用してプログラムや変数をディスクファイルに保存する詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。 プログラムや変数のコントローラへの保存の詳細については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCE-464)』を参照してください。
.cal	システムキャリブレーションデータファイルです。
.xml	ロボット仕様および他のパラメータファイルです。
.rtf	テキストファイル（Readme ファイル）です。
.pg	V+ プログラム、またはプログラムグループ（モジュール） 指定のプログラムに加え、これらのプログラムが参照するサブルーチンもファイルに保存できます。 STOREP、STOREM キーワードを使用してプログラムや変数をディスクファイルに保存する詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

プログラムとサブルーチン

V+ は、インタープリタ型の言語のため、リンクとコンパイルは必要ありません。メインプログラムとサブルーチンは、別のプログラムとして常に存在します。eV+ のファイル構造では、メインプログラムが呼び出す、または実行するすべてのサブルーチンをメインプログラムと同一のファイルに保存できるため、メインプログラムの読み込みと同時に、メインプログラムが呼び出すすべてのサブルーチンも読み込まれます。また、プログラムとサブルーチンを同時に格納するこのファイルはモジュールと呼ばれます。

プログラムがシステムメモリにないサブルーチンを呼び出すと、エラー **Undefined program or variable name**が表示されます。

追加情報: 詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』の STORE_ と MODULE キーワードを参照してください。

プログラムファイル作成の詳細については、Sysmac 『Studio ロボット統合システム制御機能 オペレーションマニュアル(SBCA-464)』を参照してください。

第2章: キーワードの使用方法

本章では、eV+ システムでさまざまなキーワードを使用する方法について説明します。

2.1 関数キーワードの使用

V+ プログラムステートメント内で関数キーワードを発行し、次に示すさまざまな方法で使用できます。

追加情報: 特定の関数キーワードの詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

関数キーワードの構文

eV+ には、文字列データ、数値データ、その他のデータを操作する、さまざまな定義済みの関数キーワードが用意されています。ほとんどの場合、関数キーワードにデータを入力する必要があります。キーワードは、そのデータに対する特定の演算処理をした値を返します。関数キーワードは、値や式を使用する場所ならどこでも使用できます。

V+ プログラムでステートメントを作成する場合、関数キーワードの正しい構文に従う必要があります。構文に誤りがある場合、eV+ は、*Illegal monitor command*エラーを返します。

変数割り当てとデータタイプ

関数キーワードを発行すると、eV+ オペレーティングシステムは値を返します。戻り値のデータタイプは、関数に割り当てられた変数のデータタイプと一致する必要があります。例えば、\$TIME 関数キーワードを発行した場合、eV+ は文字列データタイプの値を返します。次の例では、変数割り当てで使用するさまざまなデータタイプを示します。

```
$current_time = $MID($TIME(,time),11,8)

SET #pos1 = #PHERE

SETBELT %main.belt = BELT(%main.belt)
```

追加情報: 詳細については変数とデータタイプ 47 ページを参照してください。

関数をキーワードパラメータとして使用

関数の戻り値のデータタイプが正しい場合に限り、関数キーワードをパラメータとして使用することもできます。例として、次に示す SQRT 関数ステートメントについて説明します。SQRT 関数キーワードの構文は、「SQRT(value)」です。

```
i = SQRT(SQR(x))
```

上記ステートメントは、変数「x」の絶対値を返し、その値を変数「i」に代入します。上記ステートメントで「SQR(x)」の代わりに、数値を返す関数を使用することもできます。

式で使用する関数キーワード

関数の戻り値のデータタイプが正しい場合に限り、関数キーワードを式として使用することもできます。例として、次に示す IF ... THEN 条件ステートメントについて説明します。IF ... THEN キーワードは、構文「IF logical_exp THEN」を使用します。

```
IF LEN($string_variable) > 12 THEN
```

上記ステートメントは、「LEN(\$string_variable) > 12」を検証します。この式には、変数「\$string_variable」の文字数を返す LEN 関数キーワードが含まれています。上記ステートメントで LEN(\$string_variable) の代わりに、数値を返す関数を使用することもできます。

2.2 モニタコマンドキーワードの使用

モニタコマンドキーワードは、モニタウィンドウで発行したり、コマンドプログラムで使用できます。ほとんどのモニタコマンドキーワードには、eV+ システムの命令実行方法を制御する 1 つ以上のパラメータが含まれます。モニタウィンドウでキーワードを発行するとき、この追加情報を指定します。パラメータは、リストにある順序で入力し、正しい構文を使用する必要があります。

追加情報: 特定のモニタコマンドキーワードの詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

モニタウィンドウの詳細については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCE-464)』を参照してください。

モニタコマンドキーワードの構文

V+ プログラムでステートメントを作成する場合、モニタコマンドキーワードの正しい構文に従う必要があります。構文に誤りがある場合、eV+ は、*Illegal monitor command*エラーを返します。

一般的なモニタコマンドキーワードのフォーマットを次に示します。キーワードは大文字、引数は小文字です。必須キーワード、パラメータ、等号や丸かっこなどの記号は太字です。オプションのキーワード、パラメータ、記号は標準体の文字です。

KEYWORD

KEYWORD req_param

KEYWORD (req_param)

KEYWORD opt_param

KEYWORD opt_param1, opt_param2, ..., opt_paramX

KEYWORD req_param, opt_param = value

KEYWORD @task:program_step(expression)

KEYWORD req_param1 = req_param2

KEYWORD req_param1 = req_param2 **KEYWORD** opt_param

追加情報: モニタコマンドキーワードのすべての必須あるいはオプションのパラメータ、および関連する構文については、『eV+ 3 キーワードリファレンスマ

マニュアル (SBCE-487)』を参照してください。特定のキーワードの詳細については、前述のマニュアルに記載されている個別のキーワードの箇所を参照してください。

モニタコマンドキーワードのパラメータ入力

モニタコマンドキーワードのパラメータは、オプションまたは必須です。パラメータが必要な場合、コマンドラインで値を入力する必要があります。入力しない場合、コマンドは正確に実行されません。キーワードとキーワードに続くパラメータの間には、空白文字が必要です。通常、パラメータの区切りにはコンマを使用しますが、一部のモニタコマンドキーワードでは、等号 (=) 代入演算子が必要になります。

パラメータがオプションの場合、その値を省略することができます。省略した場合、システムはデフォルトの値を使用します。例えば、次のように STATUS モニタコマンドキーワードを発行できます。STATUS の構文は、「STATUS select」です。「select」はオプションのパラメータです。

```
STATUS
```

上記のようにして STATUS モニタコマンドを発行すると、最後にリセットしてから使用されたシステムタスクに使用されるすべてのプログラムタスクに対するステータス情報が返されます。

次に示すように、パラメータ値を指定して STATUS モニタコマンドキーワードを発行すると、システムタスク番号 1 のステータス情報のみが返されます。

```
STATUS 1
```

パラメータの区切り文字の前後の空白文字は、オプションです。省略されたパラメータの後に、1 つ以上のパラメータを続ける場合、パラメータの区切り文字を入力する必要があります。省略されたパラメータの後のすべてのパラメータがオプションで、これらのパラメータも省略した場合、区切り文字を入力する必要はありません。次の例では、この概念を示しています。XSTEP の構文は、「XSTEP task program (param_list), cycles, step」です。すべてのパラメータはオプションです。

```
XSTEP assembly, , 23
```

2.3 プログラム命令キーワードの使用

プログラム命令キーワードを使用して、ロボットや I/O の制御、ファイル操作、エラー処理、データ管理などの機能の実行のための V+ プログラムを作成します。本項では、V+ プログラムを作成するときのプログラム命令キーワードの使用法について説明します。

プログラム命令キーワードの構文

V+ プログラムでステートメントを作成する場合、プログラム命令キーワードの正しい構文に従う必要があります。構文に誤りがある場合、V+ エディタでステートメントエラーが発生し、プログラムは実行されません。

パラメータがある場合、プログラム命令キーワードの構文でコンマ、角かっこ、丸かっこなどの記号を使うことが必須です。プログラム命令キーワードには、演算に必要な情報を定義または指定するパラメータが含まれる場合があります。一部のプログラム命令キーワードではパラメータは不要ですが、それ以外のプログラム命令キーワードではパラメータ必須またはオプションです。

一般的なプログラム命令キーワードの形式 (一部) を次に示します。キーワードは大文字、引数は小文字です。必須のキーワード、パラメータ、等号や丸かっこなどの記号は**太字**です。オプションのキーワード、パラメータ、記号は標準体の文字です。

KEYWORD

KEYWORD req_param

KEYWORD opt_param

KEYWORD (parameter)opt_param

KEYWORD opt_param1, opt_param2, ..., opt_paramX

KEYWORD req_param, opt_param = value

KEYWORD req_param1 = req_param2

KEYWORD req_param1 = req_param2 KEYWORD opt_param

追加情報: プログラム命令キーワードのすべての必須あるいはオプションのパラメータ、関連する構文については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。特定のキーワードの詳細については、前述のマニュアルに記載されている個別のキーワードの箇所を参照してください。

パラメータ不要

LEFTY、RIGHTY、ABOVE、BELOWなど、一部のプログラム命令キーワードではパラメータは不要です。パラメータ不要のキーワードは、以下のように使用します。

```
ABOVE
MOVE point1
BREAK
```

必須またはオプションのパラメータ

一部のプログラム命令キーワードでは、パラメータは必須またはオプションです。複数のパラメータを持つキーワードを使用する場合、コンマ、角かっこ、丸かっこなどの記号が必要になる場合があります。必須のパラメータまたは構文に誤りがある場合、エラーが発生します。エラー情報の詳細については、警告メッセージ、情報メッセージ、エラーメッセージ 95 ページを参照してください。また、eV+ の構文を正確に記述するための記号の使い方については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

注: プログラム行を入力するとき、行内の要素間に空白スペースを追加できます。eV+ エディタは、プログラム行のスペースを追加または削除して、標準の間隔にします。また、エディタは、すべてのキーワードを大文字、すべてのユーザー定義名を小文字に自動的に変換します。

2.4 システムパラメータキーワードの使用

システムパラメータは、eV+ システムの任意の動作特性を決定します。これらのパラメータの数値は、特定のシステム設定やニーズに合わせて、モニタウィンドウやプログラム内で変更できます。本項では、パラメータ値を表示、変更する操作とともに、さまざまなパラメータについて説明します。

使用可能なシステムパラメータ

使用可能なシステムパラメータと基本的な機能について次の表に示します。設定の詳細、例、他の情報については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してくだ

さい。

表2-1. システムパラメータの詳細

パラメータ	説明
BELT.MODE	eV+ システムのコンベアトラッキング機能の特性を設定します。
DEVIATION	ロボットが特異点にあるとき、特異領域のモーションに 1～100% のパス偏差を追加します。
NOT.CALIBRATED	eV+ システムが制御するロボットのキャリブレーション状態を示します。
VTIMEOUT	ビジョンコマンド後に応答が無い場合、エラーメッセージを返すタイムアウト値を設定します。
JOG.TIME	ジョグ運転のキープアライブタイムを設定します。 このパラメータ設定は、ジョグ運転が実行されるたびに、軸またはジョイントが移動する時間を指定します。

システムパラメータの表示と設定

モニタウィンドウでシステムパラメータの表示と設定ができます。また、システムパラメータは、V+ プログラムでも制御できます。本項では、システムパラメータの表示方法および設定方法について説明します。

モニタウィンドウを使用したシステムパラメータの表示と設定

次の例に示すように、PARAMETER モニタコマンドキーワードを使用して、パラメータの値を表示、設定します。

パラメータの値の表示

次の例では、モニタウィンドウにすべてのパラメータと現在の値を表示します。

```
PARAMETER
```

次の例では、モニタウィンドウに BELT.MODE パラメータの現在の値を表示します。

```
PARAMETER BELT.MODE
```

パラメータの値の設定

次の例では、BELT.MODE パラメータに 4 の値を設定します。

```
PARAMETER BELT.MODE = 4
```

V+ プログラムを使用したシステムパラメータの読み取り／書き込み

V+ プログラムの実行中に [PARAMETER] プログラム命令キーワードを使ってパラメータの設定をすることができます。

注: 通常、パラメータの設定は Monitor Command Program を使用します。詳細については、「モニタコマンドプログラム 65 ページを参照してください。

次のプログラムステートメントは、PARAMETER プログラム命令キーワードを使用して、BELT.MODE システムパラメータのビット 1 とビット 3 を 1（マスク値 1 + 4）に設定します。

```
PARAMETER BELT.MODE = 5
```

V+ プログラムの実行中、またはモニタウィンドウを使用して、パラメータ値を返すことができます。

次のプログラムステートメントは、BELT.MODE システムパラメータの現在の設定値をモニタウィンドウに返します。

```
TYPE "The BELT.MODE parameter is set to", PARAMETER(BELT.MODE)
```

2.5 システムスイッチキーワードの使用

システムスイッチは、eV+ システム固有の動作特性を決定します。モニタウィンドウやプログラム内でこれらのスイッチを ON/OFF することができます。本項では、システムスイッチの表示・制御操作とともに、さまざまなシステムスイッチについて説明します。

使用可能なシステムスイッチ

使用可能なシステムスイッチと基本的な機能について以下の表に示します。設定、例、その他の詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

スイッチ	説明
AUTO.POWER.OFF	このスイッチが ON のときに以下のエラーが発生した場合、ハイパワーは無効になります。このスイッチが OFF のときにエラーが発生した場合、ロボットを停止し、eV+ システムに信号を送りますが、ハイパワーのスイッチは OFF になりません。 次にエラーの定義を示します。 • (-624) *force protect limit exceeded* • (-1003) *Time-out nulling errors* Mtr • (-1006) *Soft envelope error* Mtr
CP	連続パスの動作処理を有効／無効にします。詳細については、連続パス軌道 40 ページを参照してください。
DECEL.100	ACCEL プログラム命令キーワードに対して 100% の最大減速を有効または無効にします。
DELAY.IN.TOL	eV+ が動作区間(位置決め許容範囲)を完了した後に、COARSE または FINE を無効にするタイミングを制御します。詳細については、COARSE、FINE プログラム命令キーワードを参照してください。
DRY.RUN	このスイッチは、ロボットへの動作コマンドの送信を有効／無効にします。プログラムの適切な論理フローや外部通信機能のテストをするときは、衝突を防ぐためにこのスイッチを ON してください。 注: DRY.RUN が有効な場合、T20 ペンダントを使用してロボットを移動できます（ロボットに割り付けられているタスクがな

スイッチ	説明
	ければ。そうではない場合、「Robot Interlocked」（ロボットがインターロックされています、-621）エラーが発生します。
MESSAGES	このスイッチを ON にすると、TYPE プログラム命令キーワードを使って、モニタウィンドウにメッセージを表示できます。 このスイッチを OFF にすると、TYPE プログラム命令キーワードを使って、モニタウィンドウにメッセージを表示することは出来ません。
OBSTACLE	ロボットに定義された最大 4 つの異なる障害物を有効／無効にします。
POWER	このスイッチはロボットのハイパワーを制御、表示します。ロボットのハイパワーのスイッチがON になると、本スイッチは自動的に有効になります。 システム設定において、ロボットのタイミングが 0 以外に指定されている場合、ハイパワーを有効にするには 2 段階の手順が必要になります。この場合、T20 ペンダント、モニタウィンドウ、ソフトウェアのいずれかからハイパワーを有効にした後、システムによりタイミングの設定値（秒）の間、ハイパワーインジケータが点滅します。タイムアウトが発生した場合、メッセージ *HIGH POWER button not pressed* が表示されます。 詳細については警告メッセージ、情報メッセージ、エラーメッセージ 95 ページを参照してください。 ロボットのタイミング設定値の詳細については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』を参照してください。
ROBOT	1 台のロボットまたはすべてのロボットを有効、または無効にします。
SCALE.ACCEL	プログラム速度が事前に設定されたしきい値を下回っている限り、プログラム速度の関数として加速度および減速度のスケーリングを有効または無効にします。
SCALE.ACCEL.ROT	直線モーション中、SCALE.ACCEL システムスイッチが直交回転速度を考慮するかを指定します。
UPPER	文字列の値を比較するとき、小文字と大文字を区別するかどうかを指定します。このスイッチが ON の場合、すべての小文字は大文字とみなされます。

システムスイッチの表示と設定

システムスイッチは、モニタウィンドウで表示したり、設定できます。また、V+ プログラムで制御することもできます。本項では、システムスイッチの表示方法および設定方法について説明します。

モニタウィンドウを使用したシステムスイッチの表示と設定

次の例に示すように、SWITCH モニタコマンドキーワードを使用して、スイッチの値を表示、設定します。

システムスイッチの表示

次の例では、モニタウィンドウにすべてのスイッチとその現在の値を表示します。

SWITCH

次の例では、モニタウィンドウに POWER スイッチの現在の状態を表示します。

SWITCH POWER

システムスイッチの設定

次の例では、AUTO.POWER.OFF スイッチを ON にします。

ENABLE AUTO.POWER.OFF

次の例では、AUTO.POWER.OFF スイッチを OFF にします。

DISABLE AUTO.POWER.OFF

第3章: eV+システムの動作

以下の項ではeV+システムの各種動作について説明します。

3.1 システムメッセージ

システムメッセージには、識別のために指定された番号および文字列があります。これらを使用することで、V+プログラムやモニタウィンドウを使用して、各種eV+状態の検出および復旧をすることが来ます。

追加情報: 詳細については警告メッセージ、情報メッセージ、エラーメッセージ
95 ページを参照してください。

メッセージタイプ

メッセージタイプは次の 3 種類があります。

- 情報メッセージ (メッセージ番号 0~49)
- 警告メッセージ (メッセージ番号 50~299)
- エラーメッセージ (マイナス値を持つメッセージ)

システムの挙動

eV+ システムは、メッセージのタイプおよびそのメッセージが発生したときの eV+ の状態によってさまざまな挙動をとります。

情報メッセージおよび警告メッセージはシステムの動作に影響を与えず、監視する目的で使用することができます。

エラーメッセージはシステムの動作に影響を与えます。eV+ システムを完全に停止することはしませんが、そのエラーに関連する個別タスクの実行は停止します。

エラー発生後のシステムの挙動は、実行中のタスク、ロボットの状態およびエラータイプによって異なります。ロボットが動作中で、エラーの発生したタスクにあった場合、ロボットは最大の減速度で停止し、エラーメッセージが発生します。ロボットが動作中ではないタスクでエラーが発生した場合、そのタスクだけが停止して、ロボットは影響を受けません。

メッセージ処理

eV+ はシステムメッセージ情報を内部に保存して、ERROR および REACTE キーワード操作によってアクセスできるようにしています。

ERROR キーワード操作を使用してアクセスするために保存されているのは次の情報です。

- (現在の、または選択した) タスクの最後のシステムメッセージ
- 割り付けられているロボットそれぞれの最後のシステムメッセージ
- eV+ システムの最後のメッセージ
- それぞれのシステムメッセージのカテゴリ、ソースおよび名前

追加情報: ERROR および REACTE キーワード操作の詳細については、『eV+ 3
キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

3.2 デジタル I/O 制御

eV+ 起動時、システムが検出する内部と外部のデジタル I/O に対してシステムメモリブロックが割り当てられます。

デジタル I/O の制御に複数のプログラム命令キーワードを使用することができます。デジタル入力信号の変化をモニタしたり、条件に応じて出力信号を ON/OFF できます。

デフォルトの信号割り当てに関する詳細についてはロボットユーザズマニュアルを参照してください。

I/O 使用上の留意事項

RESET、DEF.DIO プログラム命令キーワードは、Host I/O 信号番号 4001～4999 に影響しません。

IO モニタコマンドキーワードは、ホストにマッピングされている信号値を返しますが、マッピングされていない信号に対しては「-」を返します。

SIG.INS 関数キーワードは、ホスト信号を使用できる場合は TRUE、使用できない（マッピングされていない）場合は FALSE を返します。

詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

基本的な I/O 制御

プログラム命令キーワードを使用した基本的な I/O 制御の例を示します。

入力信号の例

次の例では、デジタル入力チャンネル 1001 にアタッチされているスイッチデバイスが閉じるまで、プログラムの実行を停止します。

```
WAIT SIG(1001)
```

次の例では、入力信号 1002 を評価し、信号が ON のときプログラム「service.feeder」を呼び出します。

```
IF SIG(1002) THEN
    CALL service.feeder()
END
```

注: 上記の例ではこの条件が評価された時に信号1002 がON であった場合に限って、「service.feeder」を呼び出します。

出力信号の例

次の例では、デジタル出力信号 33 を OFF にします。

```
SIGNAL(-33)
```

次の例では、デジタル出力信号 33 を ON にします。

```
SIGNAL(33)
```

次の例では、SIGNAL ステートメントで複数のデジタル出力信号を制御します。

```
SIGNAL(33), (-34), (35)
```

ソフトウェア信号

ソフトウェア信号は、2001～2999 の範囲で提供されます。これらの信号は、すべての V+ プログラム で利用でき、SIG、SIGNAL キーワードを使用して、V+ プログラムのさまざまなタスク間、または V+ プログラムとアプリケーションマネージャの機能との間で相互作用として使用できます。

3.3 ディスク I/O の操作

以下の項ではディスク I/O 操作に共通する基本手順および機能について説明します。

論理ユニット番号

すべての eV+ ディスク I/O 操作では、論理ユニット番号 (LUN) と呼ばれる整数値を参照します。LUN を使用することで、I/O 操作で参照されるデバイスやファイルを特定できます。LUN デバイスは、ロボット、ディスク、TCP プロトコルデバイスドライバ、TFTP サーバまたは UDP プロトコルデバイスドライバを参照できます。

LUN 指定子は、次に示すキーワードで使用されるパラメータです。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

これらのプログラム命令キーワードは LUN パラメータを使用します。

- ATTACH
- DETACH
- FCMND
- FOPEN
- FSET
- READ
- WRITE

I/O 操作のエラーステータス

多くの他の eV+ キーワードと異なり、I/O 操作は、一定の状況下で失敗することが想定されています。たとえば、ファイルの読み込みの場合、ファイルの最後に達したことを示すために、エラーステータスがプログラムに返されます。プログラムは、このエラーに対処し、実行を継続する必要があります。

このような理由から、eV+ の I/O キーワードは、通常、エラーが発生したときにプログラムの実行を停止しません。代わりに、操作の成功／失敗は内部的に保存され、IOSTAT 実数値関数でアクセスできます。例えば、IOSTAT(5) は、LUN 5 に対して最後に実行した I/O 操作のステータスを示す値を返します。IOSTAT が返す値は、次の 3 つに分類されます。

表3-1. IOSTAT キーワードの戻り値

値	説明
1	I/O 操作は正常に完了。
0	I/O 操作が未完了。この値は先行読取またはノーウェイトの I/O 操作が実行されている場合のみ表示されます。
<0	I/O 操作が完了し、エラーが発生。エラーコードは発生したエラーの種

値	説明
	類を示します。

追加情報: 警告メッセージ、情報メッセージ、エラーメッセージ 95 ページを参照してください。プログラムで \$ERROR 文字列関数キーワード（または LISTS モニタコマンドキーワード）を使用して、ほとんどの I/O エラーに関連するテキストを生成できます。

注: GETC キーワードによってエラーが直接返されるので、GETC キーワードの使用後は IOSTAT を使用する必要はありません。

論理ユニットのアタッチとデタッチ

I/O デバイスがプログラムからアクセスされる前に、ATTACH プログラム命令キーワードを使用してその I/O デバイスをアタッチする必要があります。一度、あるプログラムタスクによって特定のデバイス（ティーチペンダント等）がアタッチされると、他のプログラムタスクはそのデバイスを使用できません。

参照した論理ユニット番号に連携するデバイスがアタッチされていない場合、ほとんどの I/O リクエストは失敗します。プログラムがデバイスを利用し終えたときは、DETACH プログラム命令キーワードでデバイスをデタッチします。これにより、他のプログラムが保留している I/O 操作を処理できるようになります。

論理ユニットにアタッチするときに、物理デバイスの種類を指定できます。デバイスの種類が指定されると、アタッチされている論理ユニットに限り、デフォルトを変更します。指定したデバイスの種類は、論理ユニットがデタッチされるまで扱われたままになります。

制御プログラムが正常に実行を完了すると、そのプログラムにアタッチされているすべての I/O デバイスは自動的にデタッチされます。プログラムが異常停止した場合、ほとんどのデバイスのアタッチは保持されます。制御プログラムタスクが再開し、これらの論理ユニットに再度アタッチする場合、そのアタッチがまだ有効なため、失敗する可能性があります。

追加情報: KILL モニタコマンドは、プログラムにアタッチされているすべてのデバイスをデタッチするようにプログラムに強制します。

プログラムによってアタッチされた場合、エラー状態やシングルステップモードを含む何らかの理由でプログラムが休止または停止したときは、モニタウィンドウとティーチペンダントはデタッチされます。プログラムが再開すると、終了前にアタッチされたモニタウィンドウとティーチペンダントは自動的に再度アタッチされます。

アタッチ要求応答モード

アタッチ要求はオプションとして即時モードを指定できます。通常、アタッチ要求はキューに入れられ、別の制御プログラムタスクがデバイスにアタッチされている場合、呼び出しプログラムが一時停止されます。デバイスがデタッチされると、キュー内にある次のアタッチが処理されます。即時モードでは、要求されたデバイスがすでに別の制御プログラムタスクでアタッチされている場合、ATTACH プログラム命令キーワードは、エラーを伴って直ちに完了します。

また、アタッチ要求はノーウェイトモードも指定できます。このモードは、プログラムにアタッチ要求の完了を待つことを強制せず、アタッチ要求がキューに入ることを許可します。IOSTAT 関数を使用して、いつアタッチが完了したかを判断する必要があります。

タスクがすでに論理ユニットにアタッチされている場合、デタッチせずに再度アタッチが試行されるとウェイトモードの種類にかかわらず、即時にエラーが発生します。

ディスクデバイスのアタッチとデタッチ

次の項に、ディスクデバイスのアタッチとデタッチの方法を示します。

ディスクデバイスのアタッチ

アクセスするデバイスの種類は、DEFAULT モニタコマンドキーワード、または ATTACH プログラム命令キーワードを使用して決定します。DEFAULT キーワードで設定されているデフォルトのデバイスの種類が、ある時点で適当ではない場合、ATTACH キーワードを使用してデフォルトの種類を無効にできます。

次の例は、使用可能なディスク論理ユニットにアタッチして、変数「dlun」に論理ユニット番号を返します。その番号を他のディスクI/O操作で使用することができます。

```
ATTACH (dlun, 4) "DISK"
```

キーワード構文でデバイス名を省略した場合、指定された LUN に対するデフォルトのデバイスが使用されます。必ず ATTACH キーワードを使用してデバイス名を指定してください。デバイス SYSTEM は、DEFAULT モニタコマンドで指定されたデバイスを意味します。一度アタッチされると、論理ユニットをデタッチするまでデバイスを変更することはできません。しかし、ファイルを開くときは、デバイスで使用可能な任意のユニットを指定できます。例えば eV+ DISK ユニットの A、C および D です。DISK デバイス LUN をアタッチした後、LUN をデタッチするまでは、プログラムでこれらのディスクユニットのファイルを開いたり閉じたりすることができます。

ディスクデバイスのデタッチ

ディスク論理ユニットをデタッチする場合、そのユニットで開いているディスクファイルは自動的に閉じます。しかし、閉じる操作によって検出されたエラー状態はレポートされない可能性があります。そのため、FCLOSE キーワードを使用してファイルを閉じ、その後エラーステータスを確認することが重要です。FCLOSE は、そのファイル用のすべてのバッファデータがファイルに書き込まれていることを確認し、ディスクディレクトリを更新してファイルの変更を反映させます。DETACH キーワードは、論理ユニットを解放します。

次の例では、ファイルを閉じ、ディスク LUN をデタッチします。

```
FLCLOSE(dlun)
  IF IOSTAT(dlun) THEN
    TYPE $ERROR(IOSTAT(dlun))
  END
DETACH(dlun)
```

I/O デバイスの読み込みと書き込み

READ プログラム命令キーワードは、すべてのデバイスからの入力を処理します。基本的な READ キーワードは、指示された LUN にアタッチされたデバイスにリクエストを発行し、プログラムの実行を続行する前に、完全なデータレコードを受信するまで待機します。IOSTAT 関数の 2 番目の引数に 2 の値を設定すると、最後のレコード読み取り長を取得できます。

GETC 実数値関数は、完全なデータレコードを待機することなく、I/O デバイスから取得した次のデータバイトを返します。これは一般的に、シリアルラインまたはモニタウィンドウからデータを読み取るために使用されます。また、バイト単位でディスクファイルの読み込みにも使用できます。モニタウィンドウの読み込み操作では、エコーなしに読み込むことができる特殊なモードビットをサポートしています。

モニタウィンドウの入力には PROMPT プログラム命令キーワードを使用します。GET.EVENT 実数値関数を使用して、モニタウィンドウから入力を読み込むことができます。これは、グラフィックおよび非グラフィックベースのシステムの両方で動作するプログラムの作成に有効で

す。ディスクデバイスのデータを読み込むには、対象となる論理ユニットでファイルを開く必要があります。FOPEN_ プログラム命令キーワードはディスクファイルを開きます。

WRITE プログラム命令キーワードはディスクデバイスならびにモニタウィンドウへ至る出力を処理します。基本的な WRITE キーワードは、指示された LUN にアタッチされているデバイスにリクエストを発行し、プログラムの実行を継続する前に、完全なデータレコードが出力されるまで待機します。WRITE キーワードでは、出力データのフォーマット方法、およびレコード終了マークをレコードの末尾に記述すべきかを決定する、フォーマット制御指定子を使用できます。

モニタウィンドウの出力には PROMPT または TYPE キーワードを使用します。データをディスクデバイスに書き込む前に、FOPENW または FOPENA キーワードでファイルを開く必要があります。FOPENW は新規ファイルを開き、一方 FOPENA は、既存のファイルを開きそのファイルにデータを入れます。

追加情報: 上記キーワードの使用上の注意事項、詳細、および例については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

入力ウェイトモード

通常、eV+ はプログラムの実行を継続する前に、入力キーワードからのデータが使用可能になるまで待機します。ただし、READ キーワードや GETC キーワードはノーウェイトモードを指定するオプションのパラメータを受け付けます。

ノーウェイトモードでは、使用可能なデータがない場合、これらのキーワードはただちにエラーステータス -526 (受信データなし) を返します。読み込みが正常に完了するまで、または他のエラーを受信するまで、プログラムはループしてこれらの操作を繰り返し使用できます。ディスクデバイスは入力についてノーウェイトモードを認識しないため、これらのリクエストを通常の待機有効の入力リクエストとして処理します。

出力ウェイトモード

通常、eV+ は、次のプログラムステートメントを継続する前に、各 I/O 操作が完了されるまで待ちます。

次の例では、eV+ は次のプログラムステートメントを継続する前に、50 スペースあるレコード全体が転送されるまで (伝送速度 9600 で約 50 ミリ秒) 待つようにします。

```
TYPE /X50
```

同様に、ディスクファイルへの WRITE プログラム命令キーワードは必要な物理的出力が完了するのを待ってから次に進みます。

出力キーワードに /N (ノーウェイト) フォーマット制御指定子が使用されている場合、この待機は実行されません。その代わりに、eV+ はただちに次のプログラムステートメントを実行します。IOSTAT 関数は、出力が完了しているかを確認し、前回の I/O が完了していない場合、ゼロ値を返します。最初のノーウェイト操作が完了する前に、特定の論理ユニット番号に対する 2 番目の出力キーワードが実行された場合、最初の操作が完了するまで 2 番目のステートメントは自動的に待機状態になります。この構造はノーウェイト出力が実質的にダブルバッファ状態にあることを意味します。最初の操作でエラーが発生した場合、2 番目の操作は取り消され、IOSTAT の値は最初の命令に対して正しい値となります。IOSTAT 関数の 2 番目のパラメータを 3 に設定して使用すると、ノーウェイト書き込みの完了を明確に確認できます。

ディスクファイル操作

次に示すように、eV+ システムでは複数のディスクファイル操作を使用できます。

FDIRECTORY モニタコマンドキーワードを使用して、ディレクトリのファイル名を表示できます。ディスクファイルには、シーケンシャル（ファイルの最初から最後まで順番にデータレコードにアクセス）、またはランダム（任意の順番でアクセス）のいずれかでアクセスできます。ディスクファイルのシーケンシャルアクセスとランダムアクセスの詳細については、より高度なディスク操作 35 ページを参照してください。

FCMND プログラム命令キーワードを使用して、次の操作を実行できます。

- ファイル名の変更
- サブディレクトリの作成
- サブディレクトリの削除

追加情報: FCMND キーワードは、他のディスク I/O キーワードと同様に、論理ユニットとのアタッチが必要です。コマンドの成功または失敗の結果は、IOSTAT 実数値関数が返します。IOSTAT 関数の詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

ディスクディレクトリへのアクセス

eV+ のディレクトリ構造は、IBM PC DOS オペレーティングシステムで使用されている構造と同じです。ディレクトリ構造には、各ファイルのファイル名、属性、作成日時、ファイルサイズが含まれます。FOPEN キーワードの実行に成功すると、ディレクトリのエントリを読み取ることができます。

READ キーワード操作が返す各ディレクトリのレコードには、次の表に示す ASCII 文字で記載された情報が含まれます。

表3-2. ディスクディレクトリフォーマットの説明

バイト	サイズ (バイト)	説明
1～7	7	右側を空白文字で埋められた属性コードです。特定の属性が指定されていない場合、属性フィールドは空白になります。 次の文字を使用できます。 • D: サブディレクトリ • P: 保護ファイルのため、読み取り、修正不可。 • R: 読み取り専用ファイルのため、修正不可。 • S: システムファイル
9	1	ASCII タブ文字 (10 進数の9)
10～19	10	セクタの ASCII ファイルサイズ (右揃え)
20	1	ASCII タブ文字 (10 進数の9)
20～28	9	ファイル更新日 (dd-mm-yy フォーマット) 注: ファイルが作成または最後に変更されたときにシステムの日付および時刻が設定されていない場合、ファイル更新日は空白になります。
29	1	ASCII タブ文字 (10 進数の9)

バイト	サイズ (バイト)	説明
30～38	9	ファイル更新時刻 (hh:mm:ss フォーマット) 注: ファイルが作成または最後に変更されたときにシステムの日付および時刻が設定されていない場合、ファイル更新時刻は空白になります。
39	1	ASCII タブ文字 (10 進数の9)
40+	---	ASCII ファイル名と拡張子 (サイズはファイル名の長さによって異なる)

ディスクファイルを開く

ディスクファイルを開く前に、ファイルがあるディスクとアタッチする必要があります。FOPEN_ キーワードを使用して、ディスクファイル（およびファイルディレクトリ）を開きます。これらのキーワードは、ディスクファイルの LUN と関連付けられています。ファイルを開いた後、READ、GETC、WRITE キーワードを使用して、ファイルにアクセスできます。同じディスクにある複数のファイルを開いたときに、異なるディスクファイルに対する I/O 操作が互いに影響しないように、これらのキーワードは、割り当てられた LUN を使用してファイルにアクセスします。

注: 書込や追記アクセスのためにファイルを開いている間は、別の制御プログラムタスクは、そのファイルにアクセスできません。しかし、読み取り専用モードでは、複数の制御プログラムタスクが同時にファイルにアクセスできます。

ディスクへの書き込み

次の例では、「\$in.string」に格納されている文字列を、「dlun」に開いているディスクファイルに書き込みます。

```
WRITE (dlun) $in.string
```

次の例では、上記書き込み操作で発生したエラーを変数「error」に返します。

```
error = IOSTAT(dlun)
```

ディスクからの読み込み

次の例では、「dlun」に開いているファイルから、最初の CR/LF またはファイルの最後まで読み込み、結果を「\$in.string」に格納します。ファイルの最後に達すると、eV+ はエラー番号 -504（予期せぬファイルの最後）を発生します。このエラーを認識し、ファイルの読み込みを停止するには、IOSTAT 関数を使用する必要があります。

```
DO
  READ (dlun) $in.string
  TYPE $in.string
UNTIL IOSTAT(dlun) == -504
```

追加情報: ファイルの各バイトを確認したい場合やファイルが CR/LF で区切られていない場合、GETC 関数を使用してファイルをバイト単位で読み込みます。

複数機能の例

次の例では、ディスクファイルを作成し、ファイルに書き込み、ファイルを閉じて、再度開き、その内容を読み戻します。

```

      AUTO dlun, i
      AUTO $file.name
      $file.name = "data.tst"

      ATTACH (dlun, 4) "DISK"
      IF IOSTAT(dlun) < 0 GOTO 100

      FOPENW (dlun) $file.name
      IF IOSTAT(dlun) < 0 GOTO 100

      FOR i = 1 TO 10
        WRITE (dlun) "Line "+$ENCODE(i)
        IF IOSTAT(dlun) < 0 GOTO 100
      END

      FCLOSE (dlun)
      IF IOSTAT(dlun) < 0 GOTO 100

      FOPENR (dlun) $file.name
      IF IOSTAT(dlun) < 0 GOTO 100
      READ (dlun) $txt
      WHILE IOSTAT(dlun) > 0 DO
        TYPE $txt
        READ (dlun) $txt
      END
      IF (IOSTAT(dlun) < 0) AND (IOSTAT(dlun) <> -504) THEN
100 TYPE $ERROR(IOSTAT(dlun))
      END
      FCLOSE (dlun)
      IF IOSTAT(dlun) < 0 THEN
        TYPE $ERROR(IOSTAT(dlun))
      END
      DETACH (dlun)

```

より高度なディスク操作

本項では、eV+ システムでより高度なディスク操作を実行する FOPEN と FOPENR プログラム命令に関する追加情報を示します。

可変長レコード

デフォルトのディスクファイルアクセスモードは、可変長レコードモードです。このモードでは、レコードの長さは自由（最大 512 バイト）であり、512 バイトセクタの境界を超えることができます。レコードの最後は、ラインフィード文字（10 進数の 10）で表されます。ファイルの最後は、Ctrl+Z 文字（10 進数の 26）を置くことで表されます。

ラインフィード文字や Ctrl+Z 文字をデータとして可変長レコードに含めないでください。この形式は、標準の ASCII ファイル形式と互換性があり、V+ プログラムの読み込みや保存に使用されます。可変長レコードモードを選択するには、FOPEN_ キーワードのレコード長さパラメータをゼロに設定するか、またはこのパラメータを完全に省略します。このモードでは、WRITE キーワードは改行文字（10 進数の 13）とラインフィード文字を出力データに自動的に付加し

て、完全なレコードにします。出力指定で /S 形式制御が指定されたときは、リターン/ラインフィードは追加されません。次に続く WRITE はその データを現在のデータと連結して同じレコードの一部とします。/Cn 形式制御が指定されたときは、n 個のリターン/ラインフィードが書き込まれ、1 つの WRITE で複数のレコードを作成します。

READ キーワードで可変長レコードを読み込む場合、eV+ プログラムにデータを返す前に、最後のリターン/ラインフィードシーケンスは削除されます。GETC キーワードを使用してディスクファイルから読み込む場合、リターン文字、ラインフィード文字、Ctrl+Z 文字を含むすべての文字はファイルで表示されるままの形で返されます。

固定長レコード

固定長レコードモードでは、ディスクファイルのすべてのレコードは、指定された同じ長さです。レコードの開始または終了を示す特殊文字の埋め込みはありません。レコードは連続しており、512 バイトセクタの境界を自由に超えることができます。

固定長レコードモードを選択するには、FOPEN_ キーワードのレコード長さパラメータをレコードのサイズ (バイト単位) に設定します。WRITE キーワードは、必要に応じてデータレコードにゼロバイトを埋め込むか、または長いレコードを切り詰めて、指定されたレコードの長さに調整します。ほかのデータバイトは付加されないため、/S 形式制御も影響しません。

固定長モードでは、READ キーワードは常に指定された長さのレコードを返します。偶数のレコードに分割できない長さのファイルの場合、最後のレコードの READ にゼロバイトが埋め込まれ、正しい長さに調整されます。

シーケンシャルアクセスファイル

通常、ディスクファイル内のレコードは、最初から最後まで途中のレコードを抜かすことなく順番にアクセスされます。このようなファイルをシーケンシャルファイルと呼びます。シーケンシャルアクセスファイルは、可変長または固定長レコードのいずれかを含むことができます。

ランダムアクセスファイル

eV+ は、固定長レコードのファイルのみ、ランダムアクセスをサポートしています。ランダムアクセスを選択するには、FOPEN_ キーワードのモードパラメータのランダムアクセスビットを設定します。また、ゼロ以外のレコード長も指定する必要があります。

指定のレコードにアクセスするには、READ、WRITE キーワードでレコード番号を指定します。レコード番号が省略された、または 0 の場合、最後にアクセスしたレコードの次のレコードにアクセスします。

レコード番号は 1 から始まります。次のステートメントでは、ランダムアクセスレコードの最初のバイト位置を計算します。

```
byte_position = 1 + (record_number - 1) * record_length
```

バッファリングと I/O オーバーラップ

すべての物理ディスク I/O は、512 バイトごとに読み込み、書き込みされます。レコードは、入力時にセクタバッファから取り出され、レコードを完了するために追加のセクタが必要に応じて読み込まれます。読み込み操作速度を上げるために、eV+ は現在のセクタを処理しながら、次のセクタの読み込みリクエストを自動的に発行します。このリクエストは、先読みと呼ばれます。

先読みは、シーケンシャルアクセスとランダムアクセスモードの両方でデフォルトとして選択されています。FOPEN_ キーワードのモードパラメータのビットを設定して、先読みを無効にすることができます。先読みが有効な場合、読み取りアクセス用のファイルを開くと、ファイルの最初のセクタに対してただちに読み取りを発行します。

実際にレコードにランダムアクセスする場合、先読み操作は一度も使用されることがないセクタも読み込むため、システムの性能を低下させる可能性があります。この場合、先読みを無効にし、FSEEK キーワードを使用して、使用する次のレコードの先読みを開始してください。

追加情報: IOSTAT(lun, 1) 関数は、保留中の先読みまたは FSEEK 操作の完了ステータスを返します。

出力では、レコードはセクタバッファに格納され、バッファが一杯になった後で書き込まれます。/N 形式制御を指定した書き込み操作でノーウェイトモードを選択した場合、WRITE キーワードはセクタが書き込まれるまで待機せずプログラム実行の継続を許可します。

ランダムアクセスモードでは、通常、セクタバッファに含まれないレコードがアクセスされるまで、そのセクタバッファの内容をディスクに書き込みません。FEMPTY キーワードは、セクタバッファの内容をただちにディスクに書き込み、現在のセクタバッファを空にします。

非バッファモードでファイルを開くこともできます。この場合、通常のバッファモードよりもかなり低速になりますが、システム障害や停電でも書き込み情報が損失することはありません。このモードは、長時間開いた状態に置かれ断続的に更新されるログファイルに主に使用されることを目的としています。この種のファイルでこのモードを使用した場合、オーバーヘッドが増えますが、その利点に比べれば重要ではありません。

作成されたファイルを閉じるまで、ファイルサイズ情報はディスクディレクトリに保存されません。また、ファイルを閉じると、一部のセクタバッファの内容が強制的にディスクに書き込まれます。プログラムを中断しても、関連ファイルを強制的に閉じることにはなりません。KILL キーワードを実行する、または中断しているプログラムを再実行するまで、ファイルは開かれたままでディレクトリも更新されません。

第4章: モーション制御操作

本章では、eV+ システムを使用したモーション制御の操作について説明します。

4.1 モーション制御の概要

eV+ は、モーション制御ステートメントをただちに実行し、モーションの完了を待たずに次のステートメントを続行します。この先読み処理はモーション制御において通常行われる基本的な方法です。

追加情報: 連続パス動作を中断することで、先読み処理動作を変更することができます。詳細については、連続パス軌道 40 ページを参照してください。

次の例では、CP システムスイッチが ON の場合、最初の MOVE ステートメントで指令された動作をロボットが開始した後ただちに出力信号 1 は ON になります。最初のロケーション「part.1」への移動という MOVE ステートメントが完了するまで、2 番目の MOVE ステートメントは実行されません。ロケーション「part.2」への移動をロボットが開始した後ただちに信号 2 は ON になります。

```
MOVE part.1  
SIGNAL 1  
BREAK  
MOVE part.2  
SIGNAL 2
```

4.2 ジョイント補間モーションと直線モーション

あるロケーションから別のロケーションまでロボットが移動する際にとる経路は、ジョイント補間モーションまたは直線モーションのいずれになります。ジョイント補間モーションでは、一定の速度で各ジョイントを動かしますが（加速フェーズ/減速フェーズを除く）、その速度は最も低速のジョイントに制限されます。

直線モーションでは、経路が重要であるアプリケーションにおいて、ロボットツール先端がロケーション間を直線でたどることを確実にすることができます。

次のステートメントは、ジョイント補間モーションを使用して、ロケーション「pick」までロボットを移動させます。

```
MOVE pick
```

次のステートメントは、直線モーションを使用して、ロケーション「pick」までロボットを移動させます。

```
MOVES pick
```

4.3 安全な接近（アプローチ）と離脱（ディパーチャ）

移動においては、まずツールZ軸に沿って距離オフセットからある位置へ接近、またはある位置からツールZ軸に沿って離脱し、それから次の位置に移動することが多くあります。

次の例では、ロボットを位置「place」からZ軸方向に50 mm離れたところで、安全な接近と離脱をします。ロボットは、次の順に従って、ジョイント補間モーションで移動します。

1. 位置「place」の上方 50 mm の位置に移動します。
2. 位置「place」まで Z 軸方向に下降移動します。
3. 位置「place」の上方 50 mm の位置まで Z 軸方向に上昇移動します。

```
APPRO place, 50
MOVE place
DEPART 50
```

重要:APPRO 動作は指定の位置に対して相対的、DEPART 動作は現在のツール方向に対して相対的動作を行います。X、Y、Z、Q、180、Roll フォーマットのツールオフセットを使用する4自由度のロボットの場合、これらのキーワードは、ロボットワールド座標のZ軸で調整されたモーションと同様に動作します。6自由度のロボット、または yaw と pitch の方向が異なるツールオフセットを使用する自4自由度のロボットの場合、これらのキーワードの動作は異なり、モーションはロボットワールド座標のZ軸と一致しない場合があります。

注: 上記例で APPROS、DEPARTS、MOVES キーワードを使用する場合、順序は同じですが、直線モーションになります。

4.4 連続パス軌道

ロボットの動作を停止することなく、なめらかに動作区間を移行することを連続パスモーションと呼びます。これは、eV+ がロボットモーションの実行に使用する通常の方法です。必要に応じて、CP システムスイッチキーワードで連続パスモーションを無効にすることができます。CP スイッチが無効な場合、次の位置に移動し始める前に、ロボットは減速し、各動作区間の最後で停止します。

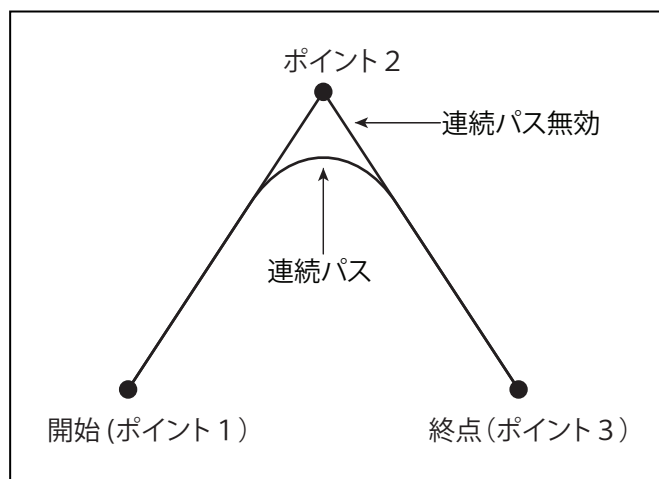


図4-1. 連続パス有効/無効

注: 連続パスモーションを無効にしても、先読み処理には影響しません。

eV+ が MOVE などの単一の動作キーワードを実行すると、ロボットは命令された速度までなめらかに加速し、位置に向かって移動を開始します。ロボットは目標位置に近づくと、なめらかに減速し、指定の位置で停止します。この動作を単一動作区間と呼びます。次のステートメントは、「pick」として定義された位置に対して単一動作区間を生成します。

```
MOVE pick
```

eV+ が動作キーワードシーケンスを実行すると、ロボットは命令された速度までなめらかに加速し、最初の位置に向かって移動を開始します。ロボットが最初の目標位置に近づいたとしても、減速停止はしません。次の目標位置に向かって軌道をシームレスに移行します。この動作をブレンドと呼びます。ロボットは、最初の動作区間の減速部分に到達すると、次の動作区間の加速部分を開始します。ブレンド動作の形態は、2つの動作区間の長さ、加速プロファイル、加速度、減速度、速度を含む多くのパラメータで決定します。シーケンスの最終ロケーションに達すると、ロボットは減速して停止します。次のステートメントは、これらの動作を示し、最初の位置を通過して、次の位置で減速停止するブレンド動作で、連続パス軌跡を生成します。

```
MOVE loc.1
MOVE loc.2
```

追加情報: 連続パスの移行は、直線モーションとジョイント補間モーションの組み合わせの間で発生する可能性があります。例えば、連続モーションは、DEPARTS キーワードによる直線モーションに続く APPRO キーワードによるジョイント補間モーションと MOVES キーワードによる最終の直線モーションで構成される場合があります。このようにして、任意の数の動作区間を組み合わせることができません。

注: 上記の例では、CPシステムスイッチが無効である場合、各動作の間で減速停止します。詳細については、システムスイッチキーワードの使用 24 ページを参照してください。

4.5 連続パスの中断

特定のプログラム命令キーワードは、ロボットの現在の動作が目標位置に到達し停止するまで、プログラムの実行を一時停止します。これを連続パスの中断と呼びます。この機能は、ある動作（エンドエフェクタを閉じる場合など）の実行中にロボットを停止する必要がある場合に有効です。

次の例は連続パスの中断を示します。

```
MOVE loc.1
BREAK
SIGNAL 1
```

上記の例では、MOVE キーワードでロボットを「loc.1」まで移動させます。プログラムの実行は継続し、BREAK 命令が実行されます。これにより、「loc.1」への移動が完了するまで、それより先のプログラムの実行は一時停止されます。移動の完了後、SIGNAL 1 命令が実行され、信号が ON になります。

追加情報: BREAK、CPOFF、DETACH(0)、HALT、OPENI、PAUSE、TOOL キーワードによって、eV+ はロボットが停止するまでプログラムの実行を一時停止します。

BRAKE キーワード、または REACTI 命令が実行されると、ロボットは減速停止します。これらのイベントは、モーション区間内でいつでも発生する可能性があります。

現在の動作が完了する前に新しいモーション制御動作が実行されない場合、ロボットは減速停止します。この状況は次の条件下で発生する可能性があります。

- ロボットの動作が完了する前で条件が満たされない時に、WAIT または WAIT.EVENT キーワードが実行された。
- 動作が完了する前に PROMPT キーワードが実行され応答がない。

- ・ モーション動作間に実行されたキーワードの実行時間が、ロボットがその指令の動作を実行する時間よりも長い。

4.6 手続き型動作

直線およびジョイント補間弧で移動する能力は、eV+ の基本動作に組み込まれています。また、ロボットツールもあらかじめ記録されている経路、または数式で記述されている経路に沿って移動できます。これらの動作は、手続き型動作で実行できます。

手続き型動作は、多くの短い動作を計算するプログラムループを使用して、適切な動作リクエストを発行します。通常、ループの開始前に、経路が計算され、配列に保存されます。ロボット動作と非動作キーワードを平行して実行するため、ロボットは停止することなく、定義されている各連続動作を実行できます。eV+ の連続経路機能は、計算された動作区間の移行を自動的にスムーズにします。

以下の例に、配列 `path[]` に保存されている位置で示される軌道に沿ってロボットツールを移動する手続き型動作を示します。LAST キーワードを使用して、配列のサイズを決定します。ロボットツールは、毎秒 0.75 インチの一定速度で配列 `path[]` に定義されている各位置を移動します。

```
SPEED 0.75 IPS ALWAYS
FOR index = 0 TO LAST(path[])
  MOVES path[index]
END
```

あるいは、経路に沿って位置を等間隔とし、加速度／減速度を増加し、DURATION キーワードを使用して、次のポイントへの移動完了に必要な動作時間を制御する方法が一般的です。これにより、経路上にある任意の 2 つのポイント間の距離に基づき、速度を効果的に定義します。

以下の例に、ロボットツールを円弧に沿って移動する手続き型動作を示します。経路は事前に記録されておらず、アークの半径と中心に基づき、数学的に記述されています。

以下の部分的なプログラムでは、実変数 `radius` に目的のアークの半径、「`x.center`」と「`y.center`」に曲率中心の各座標がすでに割り当てられています。

変数 `start` と `last` は、円の一部分を描くために定義されています。最後に、変数「`angle.step`」は、各増加動作で横断する角度の増分を指定するために定義されています。DURATION キーワードを使用するため、プログラムは、0.5 秒ごとに弧の上を「`angle.step`」度で、ロボットツールを移動させます。

このプログラムを実行すると、弧の上のポイントの X、Y 座標を繰り返し計算し、次のロボット動作区間の目標位置を定義する変換の作成に使用します。

```
DURATION 0.5 ALWAYS
FOR angle = start TO last STEP angle.step
  x = radius*COS(angle)+x.center
  y = radius*SIN(angle)+y.center
  MOVE TRANS(x, y, 0, 0, 180, 0)
END
DURATION 0 ALWAYS
```

追加情報: 手続き型動作中に加速度を増加させることが、プログラムの加速度が動作を制約しないようにすることに有効である場合があります。

4.7 モーション制御タイミングの留意事項

eV+ がモーションセグメント間の移行実行の計算時間を必要とするため、指令の位置間隔をどこまで近づけるかには制限があります。位置間隔が近すぎる場合、eV+ があるモーションから次のモーションへの移行を計算し実行するための時間が十分でないため、連続パスの動作は途切れます。これはロボットが中間地点で一時的に停止することを意味します。

このような状況を発生させない最小の位置間隔は、ある位置から次の位置への動作の完了時間によって決まります。動作区間がそれぞれ軌道時間の2倍を超える場合、直線モーションを使用することができます。1つの軌道時間秒程の短い動作区間では、ジョイント補間モーションを使用できます。

4.8 ロボットの速度と性能

本マニュアルでは、加速区間と減速区間の間でのロボットの移動速度をロボットの速度としています。また、ロボットの速度は速度プロファイルの等速区間の速度の大きさとみなすことができます。さらに、より一般的なある位置から別の位置に移動する速度を表す場合もあります（本マニュアルでは、ロボットの性能と呼びます）。

ロボット速度

次に示すように、最高速度に基づくロボットの実際の移動速度は、プログラム速度とモニタ速度で決まります。モニタ速度とプログラム速度を100に設定した場合、ロボットは最高速度で移動します。モニタ速度とプログラム速度を50に設定した場合、ロボットは最高速度の25%で移動します。

- プログラム速度は、SPEED プログラム命令キーワードで設定します。プログラムの実行開始時、この速度は100に設定されます。
- モニタ速度は、SPEEDモニタコマンドキーワード、または、unitsパラメータに「MONITOR」を指定したSPEEDプログラム命令キーワードで設定します。通常、この速度はシステムの起動時に50に設定されます。Sysmac Studio でもモニタ速度を設定できます。

追加情報: 上記の2つのSPEED指令はそれぞれ異なった効果を発揮します。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。起動時に有効となるモニタ速度の初期値の設定については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』を参照してください。

ロボットの移動には、次に示す3つの段階があります。

1. 指定の最高速度までロボットを加速する加速段階。
2. 指定の最高速度を超えない速度でロボットを移動させる速度段階。ロボットは加減速値と移動距離に基づいて、指定された最高速度に到達しない場合があります。
3. ロボットを減速停止（または次の動作へ移行）させる減速段階。

加速段階と減速段階のロボットの速度は、最高速度の割合（%）またはロボットツール先端が移動する絶対速度のいずれかで指定します。デフォルトでは最高速度の割合（%）として設定されます。

ロボットエンドエフェクタ／ツール先端速度に関する留意事項

ロボットツール先端を絶対速度で移動させるには、次の例に示すように、SPEED プログラム命令キーワードで速度を inch/s または mm/s 単位で指定します。例では、次の SPEED キーワードが実行されるまで、すべてのロボット動作に対してツール先端の絶対速度を 25 mm/s に指定します。

```
SPEED 25 MMPS ALWAYS
```

ツール先端を指定の速度で移動させるために、次の事項を検討する必要があります。

- モニタ速度を 100 にする必要があります。
- ロボットが目標とする速度まで加速し、かつ動作終了時点で減速停止できるように、移動距離を十分に確保する必要があります。

ロボットの性能

ロボットの性能は、SPEED キーワードの設定と次の要素から決定されます。

- ロボットの加速プロファイルと ACCEL 設定値。ロボットの加減速に十分な時間が取れるように、最高速度を ACCEL キーワードによって減少させることができます。
- SCALE.ACCEL と SCALE.ACCEL.ROT システムスイッチの状態はロボットの性能に影響を与えます。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。
- オプションの加速プロファイルを定義して、加減速の変化率を変更することもできます。
- 移動に関する位置許容範囲の設定値 (COARSE/FINE, NULL/NULL)。ロボットの実際の位置をより正確な位置に到達させるためには、より多くの移動時間が必要になります。
- DURATION の設定。DURATION は、SPEED の設定値にかかわらず、最短時間でロボット移動を完了させます。
- 最大許容速度。最大速度は工場出荷時に設定されています。
- ロボットの慣性負荷とロボットの調整を考慮する必要があります。
- 直線モーションとジョイント補間モーションを考慮する必要があります。直線パスと補間パスでは、動的応答が異なるため動作時間も異なります。

追加情報: 特定のアプリケーションでは、これらの設定状態によっては、ロボットの性能が著しく向上することも大幅に低下する可能性もあります。例えば、高負荷のロボットの場合、SPEED と ACCEL の設定値を下げることで、移動終了時のオーバーシュートを減少させロボットをより早く停止させることができますという、より優れた性能を得ることができます。真空グripperで製品の入った袋を取り上げるようなアプリケーションなどは高い精度を要求しないので通常、COARSE 許容誤差で高速に移動できます。

4.9 モーション修飾子

次のキーワードは、各動作の特性を変更します。

- ABOVE / BELOW
- ACCEL

- BREAK
- COARSE / FINE
- CPON / CPOFF
- DURATION
- FLIP / NOFLIP
- LEFTY / RIGHTY
- NOOVERLAP / OVERLAP
- NULL / NONULL
- BRAKE
- SINGLE / MULTIPLE
- SPEED

追加情報: 詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

第5章: 変数とデータタイプ

本項では、eV+ システムで使用する変数の作成方法とその使用方法について説明します。

5.1 変数の作成

変数とデータタイプは、明示的に宣言したり、プログラムで初めて使用することで宣言できます。変数を初めて使用するときに変数のデータタイプを決めて、その変数の保存領域を割り当てます。この方法は動的割り当てと呼ばれます。数値データ、文字列データ、最大 3 次元までの変換配列は動的割り当て方法で作成できます。

動的割り当てを使用せずに新しい変数を明示的に作成するには、次に示すような単純なステートメントを使用できます。変数が定義済みの場合、次のステートメントはその変数に新しい値を代入するだけです。

```
real_var = 13.65  
$string1 = "ABC123"
```

5.2 変数の分類

変数は、外部変数、グローバル変数、ローカル変数、オート変数に分類することができます。分類別にプログラムのさまざまな呼び出しインスタンスによる変数の使用方法や変更方法が決まります。

変数定義ステートメントは、その他のキーワードステートメントより前となるように、プログラムの最初に記述する必要があります。.PROGRAM ステートメント、コメント、空白行のみ、変数定義ステートメントの前に記述することができます。

追加情報: ある変数の分類を決めたあとでその変数を別の分類に割り当てようとすると、エラー *Attempt to redefine variable class*が発生します。

注: AUTO キーワードを使用した変数宣言ステートメントをスタックにあるタスクとして現在実行中のプログラムには追加することはできません。オート変数を追加する前に、プログラムを停止しスタックから削除する必要があります。

本項の記述により変数の分類について理解してください。

変数のスコープ

変数のスコープは、その変数にアクセスできるプログラムの範囲です。次の図にさまざまな変数のスコープを示します。

変数は、それが含まれているボックスの領域に示されているプログラムに加えて、より小さなボックス内にあるプログラムによって変更できます。

注: プログラムが外部変数、オート変数、ローカル変数を定義した場合、他のプログラムが作成した同じ名前のグローバル変数にはアクセスできません。

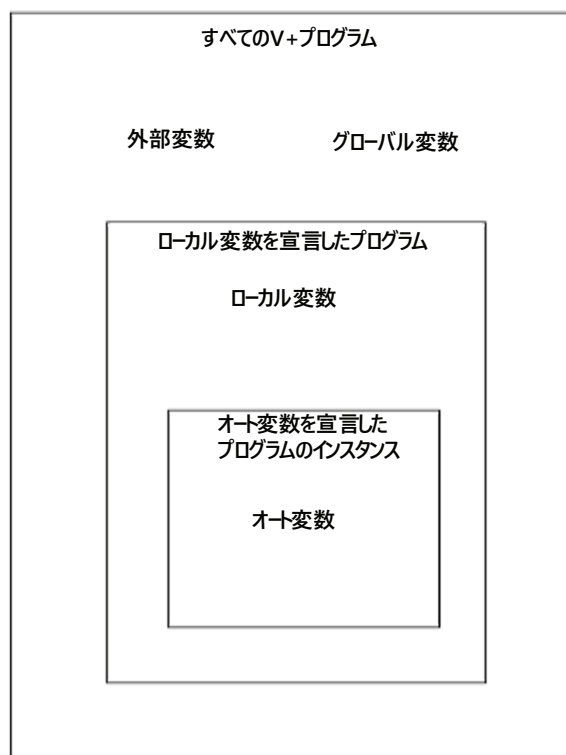


図5-1. 変数のスコープ

注: 外部変数は、NJ シリーズロボット統合CPU ユニット において定義されている場合に限って、V+ プログラムで使用することができます。詳細については外部変数 49 ページを参照してください。

次の図を使用して、さまざまなプログラムの呼び出しをする場合のグローバル、ローカルおよびオート変数の定義について理解してください。

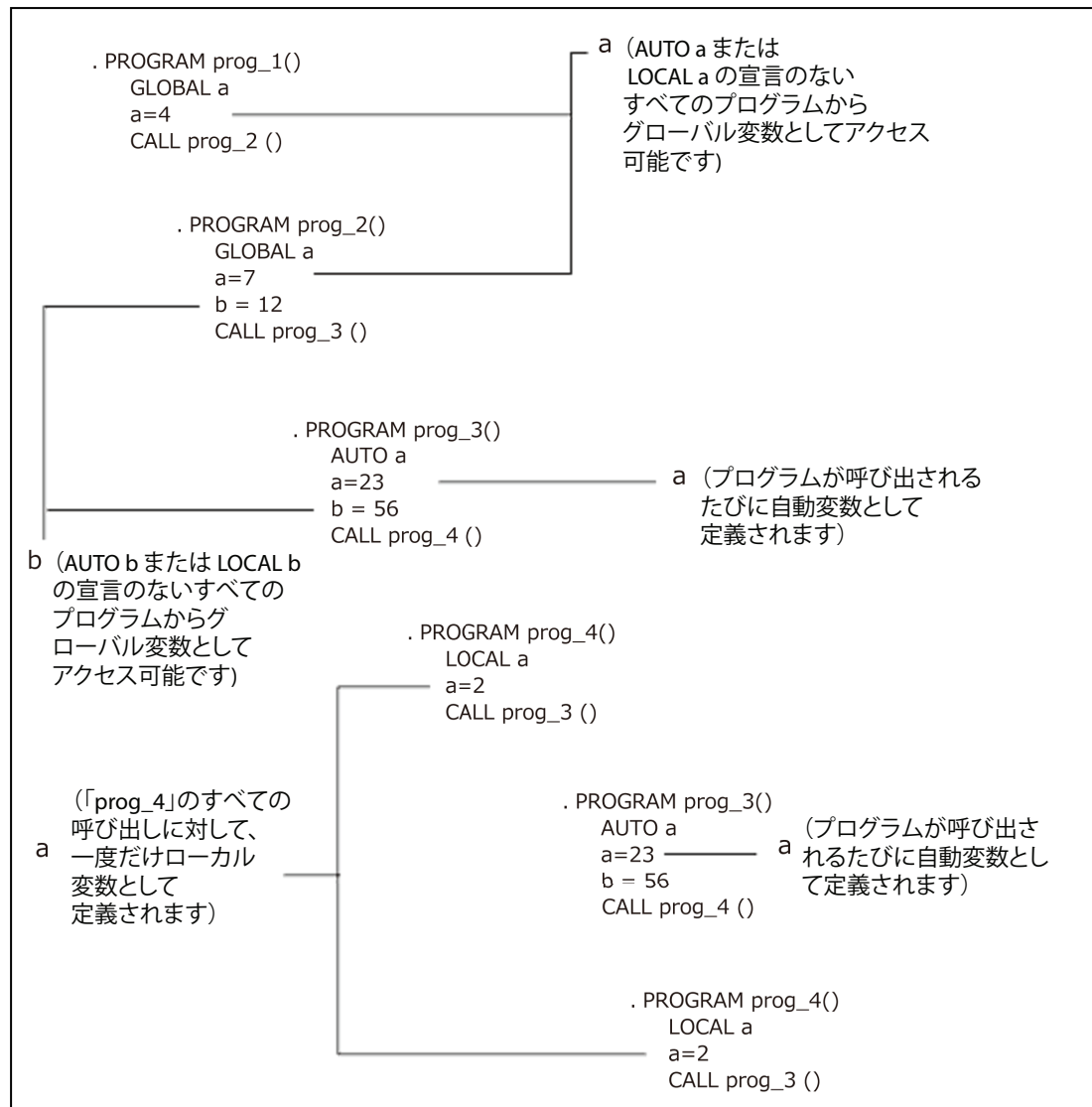


図5-2. 変数のスコープの例

外部変数

外部変数はeV+ システムとNJシリーズロボット統合CPUユニットの間で共有されます。外部変数を使用するには、NJシリーズロボット統合CPUユニットとeV+ システムで外部変数を定義する必要があります。

EXTERNAL プログラム命令キーワードを使用して、eV+ システムで外部変数を定義します。外部変数を定義した後、タスクで実行される V+ プログラムで使用できます。

EXTERNAL プログラム命令キーワードの使用の詳細については、『eV+3 キーワードリファレンスマニュアル(SBCE-487)』を参照してください。NJシリーズロボット統合CPUユニットでの外部変数の使用については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)』を参照してください。

外部変数を作成する場合、次の事項を考慮する必要があります。

- 外部変数が削除されている、または外部変数を参照するすべての V+ プログラムがシステムメモリから削除されている場合 (DELETE または ZERO キーワードを使用)、以前に定義した外部変数は使用できなくなります。

- ・ 外部変数は、グローバル変数と同じ方法でV+プログラムやモニタウィンドウからアクセスできます。
- ・ 変数名は、eV+ システムとNJシリーズロボット統合CPUユニットで同じ名前にする必要があります。
- ・ 変数名は文字から始める必要があります。
- ・ 変数名は、0～9、a-z、ピリオド、またはアンダースコアのみ使用できます。
- ・ 変数名の最大長は 15 バイトです。変数名の最初の 15 文字のみ使用されます。
- ・ 同じプログラム内では、他の変数分類（GLOBAL、AUTO、LOCAL）と同じ名前でも外部変数を定義することはできません。eV+ システムで外部変数とグローバル変数を同じ名前で定義した場合、変数名を重複して定義したことになり、エラーが発生します。
- ・ V+ プログラム内で新しい外部変数を定義する場合、EXTERNAL ステートメントは、コメント行や他の変数型定義ステートメント以外の実行可能なプログラムステートメントより前に記述する必要があります。
- ・ 配列として定義する外部変数の配列サイズは0～99 する必要があります。配列サイズは、NJシリーズロボット統合CPUユニットで定義します。詳細については、『*Sysmac Studio* ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCE-464) 』を参照してください。

外部変数のデータタイプ

外部変数のタイプと配列サイズは、NJシリーズロボット統合CPUユニットで定義します。eV+システムでは、自動的に外部変数を倍精度データタイプとして使用します。文字列、構造体、可変長配列、その他のデータタイプはサポートされていません。

外部変数に関連するエラー

次の条件が存在する場合、V+ プログラムの実行中にエラーが発生します。

- ・ 外部変数がNJシリーズロボット統合CPUユニットに存在しない場合 **Undefined program or variable name** (-406) エラーが発生します。
- ・ 配列の上限 (0～99) を超えた場合、**Illegal array index** (-404) エラーが発生します。
- ・ 変数のタイプがeV+ システムとNJシリーズロボット統合CPUユニットの間で一致しない場合、**Variable type mismatch** (-465) エラーが発生します。
- ・ 変数のタイプが未サポートの場合、**Variable type mismatch** (-465) エラーが発生します。

グローバル変数

すべての実行可能なプログラムでグローバル変数を使用できます。オート変数、ローカル変数、外部変数として定義されていない限り、その変数はグローバル変数とみなされますが、次の事項を検討する必要があります。

- ・ 同一プログラムで同じ変数をローカル変数やオート変数として定義した場合、その変数のグローバル変数としての分類は無効になります。

- DELETE、ZERO キーワード操作によってシステムメモリから、その変数を削除するかその変数を参照しているすべてのプログラムを削除するまで、グローバル変数はメモリに存在し、実行中のプログラムで使用できます。

次の例に示すように、GLOBAL プログラム命令キーワードでグローバル変数を明示的に定義できます。

```
GLOBAL LOC my.loc
GLOBAL REAL my.real
GLOBAL DOUBLE my.double
GLOBAL $my.string
GLOBAL %my.beltvar
```

重要: グローバル変数は非常に強い効力を持つため、選択的に使用する必要があります。可能な限り、グローバル変数ではなく、ローカル変数やオート変数を使用することを推奨します。

ローカル変数

ローカル変数はローカル変数を定義している実行プログラム内でのみ使用できます。ローカル変数には次の特性があります。

- ローカル変数は、プログラムの特定の呼び出しインスタンスに対してだけでなく、プログラムのすべてのコピー（呼び出しインスタンス）に対してローカルです。
- ローカル変数の参照はそれを定義したプログラム内でのみ可能です。
- LOCAL ステートメントで変数を記述した場合、そのプログラムから同じ名前のグローバル変数に直接アクセスすることはできません。
- ローカル変数名は、ほかのプログラムで定義されているローカル変数名を考慮することなく選ぶことができます。
- ローカル変数の変更はそれを定義しているプログラムによってのみ可能です。
- ローカル変数はプログラム実行中に 1 回だけ割り当てられ、その値は連続するサブルーチンの呼び出し中も保持されます。複数のプログラムタスクによって同じプログラムが実行される場合もその値が共有されます。
- ローカル変数の値は、STORE、または LOAD モニタコマンドキーワードによって保存、復元することはできません。

重要: ローカル変数を使用するプログラムが、複数の異なるプログラムタスクによって呼び出されたり、同一のタスクから再帰的に呼び出される場合、変数の値はさまざまなプログラムインスタンスによって変更され、結果的に予期せぬプログラムエラーが発生ことがあります。そのため、エラーの発生を最小限に抑えるためには、一時的なローカル変数の代わりにオート変数を使用してください。詳細については「オート変数 52 ページを参照してください。

注: ロケーション変数（LOC）とローカル変数（LOCAL）を混同しないでください。

次の例に示すように、LOCAL プログラム命令キーワードでローカル変数を明示的に定義できます。

```
LOCAL LOC my.loc
LOCAL REAL my.real
```

```

LOCAL DOUBLE my.double
LOCAL $my.string
LOCAL %my.beltvar

```

オート変数

オート変数はそれを定義したプログラムが実行されるときに自動的に作成されます。オート変数は現在のプログラム内でのみ定義できます。オート変数の特性を以下に示します。

- オート変数の参照はプログラムの特定の呼び出しインスタンスによってのみ可能です。
- プログラムが最初に実行されるときオート変数の値は未定です。プログラムが終了した後、値はなくなります。
- AUTO ステートメントで変数を記述した場合、プログラムから同じ名前のグローバル変数に直接アクセスすることはできません。
- オート変数の値は、STORE、または LOAD モニタコマンドキーワードによって、保存、復元することはできません。
- オート変数名は、ほかのプログラムで定義されている変数名を考慮することなく選ぶことができます。
- オート変数はプログラムが呼び出されるたびに割り当てられ、その値は次のサブルーチン呼び出しまでの間、保持されることはありません。
- 複数の異なるプログラムタスクで同時に呼び出したり、同一のタスクで再帰的に呼び出した場合でも、プログラムが呼び出されるたびにオート変数のコピーが作成されます。
- オート変数の保存スペースは、プログラムの実行スタックに割り当てられます。定義されたオート変数の数に対してスタックが小さい場合、タスクの実行を停止し、エラーメッセージ「Too many subroutine calls」（サブルーチンの呼び出しが多すぎます）が表示されます。
- DELETE_ プログラム命令キーワードによってオート変数を削除することはできません。

次の例に示すように、AUTO プログラム命令キーワードでオート変数を明示的に定義できます。

```

AUTO LOC my.loc
AUTO REAL my.real
AUTO DOUBLE my.double
AUTO $my.string
AUTO %my.beltvar

```

5.3 変数名の要件

新しい変数名を作成する場合、次の要件に従ってください。

- キーワードは変数名として使用できません。変数にはキーワードと同じ名前を使用できません。
- 変数名は文字から始める必要があります。
- 変数名には、文字、数字、ピリオド、アンダースコア文字（_）のみ使用できます。
- 変数名の長さは、最大 15 文字です。16 文字以上使用した場合、警告なしに自動的に 15 文字に切り詰められます。例えば、変数名「variable_name_too_long」は自動的に「variable_name_t」に切り詰められます。

- eV+ システムはデフォルトのシステム変数名を自動的に作成するので、変数名が偶然重複することがないようにするため、2 または 3 文字の後にピリオドを付けた変数名は作成しないようにしてください。「sv.error」、「tsk.idx」、「tp.pos1」などは避けるべき変数名です。

5.4 変数の初期化

変数は使用する前に、初期化が必要です。文字列変数と数値変数は、代入ステートメントの左側に配置することで初期化できます。変数を初期化する場合、次の留意事項を考慮する必要があります。

- ステートメントの右側に配置しても変数は初期化できません。
- 文字列変数、数値変数、ロケーション変数は、ディスクファイルから読み込むことで初期化できます。
- 文字列変数と数値変数は、PROMPT キーワードで初期化できます。
- 変換とプレジジョンポイントは、SET または HERE プログラム命令キーワードで初期化できます。また HERE モニタコマンドやティーチペンダントでも初期化できます。

次のステートメントは、変数「var_one」と「\$var_two」を初期化します。

```
var_one = 36
$var_two = "two"
```

次のステートメントは、「var_two」がすでに初期化されている場合、「var_one」を初期化します。「var_two」が初期化されていない場合にこのステートメントを実行すると、エラー *Undefined value* が返されます。

```
var_one = var_two
```

次のステートメントは、前のステートメントで「var_one」が初期化されている場合のみ有効です。

```
var_one = var_one + 10
```

5.5 変数演算子

本項では変数演算子について説明します。

代入演算子

数値変数と文字列変数に値を代入するとき、等号(=)を使用します。変数に値を代入する場合、変数を等号の左辺に表記する必要があります。等号の右辺には、初期化された変数、左辺と同じデータタイプの値、結果が左辺と同じデータタイプになる式を使用できます。

代入演算子の右辺で使用する変数はあらかじめ初期化しておく必要があります。ロケーション変数の場合、有効な代入ステートメントに SET キーワードを使用する必要があります。

ロケーション変数やプレジジョンポイント変数の場合、次のステートメントは使用できません。

```
loc_var1 = loc_var2
```

eV+ では、代入演算子の左辺にある、複合変換に使用されている不明な変換の解を求めることができます。これは、他のフレームを基準とするロケーション変数の初期化に役立ちます。次の例では、「slot.offset」を TRANS(200,200,200,0,0,0) として初期化します。

```
SET pallet_frame = TRANS(100,100,100,0,0,0)
SET taught_loc = TRANS(300,300,300,0,0,0)
SET pallet_frame:slot.offset = taught_loc
```

算術演算子

次の算術演算子を使用できます。

表5-1. 算術演算子

演算子	機能
+	加算
-	減算または単項マイナス
*	乗算
/	除算
MOD	剰余除算

関係演算子

関係演算子は、ブール値を求める式に使用します。関係演算子を含む式の解は、常に -1 (TRUE) または 0 (FALSE) になり、式に記述された特定の関係が真か偽かを示します。

注: 関係演算子は、両側にある変数の値を変更することはありません。

関係演算子は一般的に制御構造で使用されます。eV+ は次の表に示すような標準的な関係演算子を使用します。

表5-2. 関係演算子

演算子	機能
==	等しい
<	未満
>	より大きい
<= または =<	以下
>= または =>	以上
<>	等しくない

次のブール値ステートメントでは、「x」の値が 6、「y」の値が 10 のとき、解は -1 (TRUE) になります。

```
x < y
y >= x
y <> x
```

次のブール値ステートメントでは、「x」の値が 6、「y」の値が 10 のとき、解は 0 (FALSE) になります。

```
x > y
x <> 6
x == y
```

次の例では、代入演算子「=」と関係演算子「==」の違いを示します。このステートメントでは、ブール式「x==y」は FALSE で解は 0 のため、「z」には 0 の値が代入されます。

```
z = x == y
```

論理演算子

論理演算子は、ブール型変数や式の解に作用し、1 つのブール値を解くために複数のブール式を組み合わせます。次の表に示すように、eV+ では標準の論理演算子を使用します。

表5-3. 論理演算子

演算子	機能
NOT	式または値を補完します。TRUE の式や値を FALSE にします。FALSE の式や値を TRUE にします。
AND	式の両方が TRUE の場合のみ、式全体が TRUE になります。
OR	式のいずれかが TRUE の場合、式全体が TRUE になります。
XOR	式のいずれかが TRUE で他方が FALSE の場合のみ、式全体が TRUE になります。

次のステートメントでは、「x」の値が 6、「y」の値が 10 のとき、解は -1 (TRUE) になります。

```
NOT(x == 7)
(x > 2) AND (y <= 10)
```

次のステートメントでは、「x」の値が 6、「y」の値が 10 のとき、解は 0 (FALSE) になります。

```
NOT(x == 6)
(x < 2) OR (y > 10)
```

文字列演算子

プラス記号を使用して文字列を連結（結合）できます。

次のステートメントでは、変数「\$full_name」の値は「Omron Robotics and Safety Technologies, Inc.」になります。

```
$name = "Omron Robotics "
$incorp = ", Inc."
$full_name = $name + "and Safety Technologies" + $incorp
```

演算子の優先順位

式に複数の演算子が含まれている場合、単純に左から右順に評価されとは限りません。演算子が評価される優先順位を次の表に示します。

式では、次の表に示すとおり、優先度の高い演算子を含む関数が最初に評価されます。評価順序は、丸かっこを使用して変更することもできます。評価はまず丸かっこ内の演算子で最も深くネスティングされたペアから始まり、次にその丸かっこの外の演算子という順に、次の表に示した規則にしたがって行われます。表の同じ行にある演算子については、厳密に左から右に評価されます。

表5-4. 演算子の優先度

優先度	演算子
高	NOT、COM
...	-（単項マイナス）
	＊、／、MOD、AND、BAND
	＋、－、OR、BOR、XOR、BXOR
低	＝、＜＝、＞＝、＜、＞、＜＞

演算子の評価順位の例

式での演算子の評価順序を例を使って説明します。

次の例は異なる優先度をもつ演算子の評価順位を表しています。このステートメントを実行すると、モニタウィンドウに「1 + 1 x 3 = 4」が表示されます。

```
TYPE "1 + 1 x 3 = ", 1 + 1 * 3
```

次の例は、異なる優先度をもつ演算子の評価順位を、丸かっこを使って変更する式を示します。このステートメントを実行すると、モニタウィンドウに「(1 + 1) x 3 = 6」が表示されます。

```
TYPE "(1 + 1) x 3 = ", (1 + 1) * 3
```

次の例は、同じ優先度の演算子を単純に左から右の順に評価する式を示します。このステートメントは、左から右に評価され、モニタウィンドウに「10 MOD 3 AND 10 MOD 3 = 2」が表示されます。

```
TYPE "10 MOD 3 AND 10 MOD 3 = ", 10 MOD 3 AND 10 MOD 3
```

5.6 数値表現

前述の項で説明したように、数値は標準の 10 進数または指数表記で表すことができます。また、数値を 8 進数、2 進数および 16 進数で表すこともできます。各整数表現に必要な形式を次の表に示します。

プレフィックス	表記法	例
なし	10 進数	-193
^B	最大 8 ビットのバイナリ値	^B1001
^	8 進数	^346
^H	16 進数	^H23FF
^D	倍精度	^D20000000

注: eV+ には、特殊な論理データタイプ（ブール型）はありません。数値、変数、式は論理データ型タイプとして使用できます。eV+ は、0 を false、それ以外の値を true とみなします。詳細については、論理定数 58 ページを参照してください。

数式

変数の数値を使用できるほぼすべての状況で、数式も使用できます。次の例では、変数「x」の値はすべて 3 になります。

```
x = 3
x = 6/2
x = SQRT(9)
x = SQR(2) - 1
x = 9 MOD 6
```

5.7 文字列のデータタイプ

文字列データを含む変数の先頭にはドル記号 (\$) を付けます。文字列変数に代入する値は、ダブルクォーテーションで囲む必要があります。文字列変数の値には、次の制約があります。

- 34 (") を除く、32 (スペース) ~126 (~) の ASCII 文字を文字列変数に代入することができます。
- 文字列には最大 128 文字まで使用できます。
- 文字列変数には 0~255 の値を使用できます。

注: 変数名に対する 15 文字の制限については、ドル記号は含まれません。

次のステートメントは、その変数がまだ存在しない場合、新しい変数「string_variable」を作成し、値「ABC123」を代入します。

```
$string_variable = "ABC123"
```

次のステートメントは、文字列変数に実数値を割り当てようとするため、エラーを返します（詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』の \$ENCODE キーワードを参照）。

```
$string_variable = 123.0
```

ASCII 値

ASCII 値は、それぞれ 1 つの ASCII 文字を示す数値表現です。ASCII 値を指定するには、文字の前にアポストロフィ (') を付けます。スペース文字 (10 進数の 32) からチルダ文字 (16 進数の 7e、10 進数の 126) までの ASCII 文字を ASCII 定数として使用できます。次に有効な ASCII 定数を示します。

```
'A'    '1'    'v'    '%'
```

追加情報: ASCII 値 '1 (10 進数の 49) は、整数値 1 (10 進数の 1) や文字列の「1」とも異なります。

5.8 実数型整数のデータタイプ

整数部と小数部（または値が指数表記で表現されている場合、仮数部と指数部）を持つ数値のデータタイプは、実数になります。整数部のみの数値のデータタイプは、整数になります。通常、eV+ では、この 2 種類のデータタイプを区別する必要はありません。

整数が必要で実数値を使用した場合、eV+ は四捨五入（切り捨てではなく）を行い実数を整数に変換します。実数が必要な場合、eV+ は、整数を小数部のない特殊なケースの実数とみなします。

デフォルトの実数値タイプは、IEEE 754 規格の符号付き 32 ビット単精度値です。実数値は、DOUBLE キーワードで明確にタイプを指定された場合、IEEE 754 規格の 64 ビット倍精度値としても保存できます。

実数値と整数値の範囲

次に、実数値変数と整数値変数に値を割り当てる場合の範囲を示します。

表5-5. 実数値と整数値の範囲

データタイプ	範囲
整数	-16,777,216～16,777,215
単精度実数値	24 ビット精度の -1E+38～1E+38
倍精度実数値	52 ビット精度の -1E+307～1E+307

5.9 論理定数

eV+ では、次の論理定数を使用できます。ブール式の使用が想定されているところでこれらの定数を使用できます。

- TRUE (-1)
- ON (-1)
- FALSE (0)
- OFF (0)

評価が必要なあらゆる場所で、論理値、変数、式を使用できます。次の例では、入力信号を評価します。信号が ON (HIGH) の場合、変数「dio.sample」は TRUE となり、IF 節が実行されます。それ以外の場合は、ELSE 節が実行されます。

```

dio.sample = SIG(1001)
IF dio.sample THEN
  TYPE "The signal is ON"
ELSE
  TYPE "The signal is OFF"
END

```

上記例では、最初の 2 行を組み合わせ、次のステートメントに置き換えることができます。

```

IF SIG(1001) THEN

```

5.10 ロケーションデータタイプ

ロケーションは、変換またはプレジジョンポイントとして指定できます。

変換は、直交座標空間のロケーションとそのロケーションにあるモーションデバイスの腕先ツースリングの方向を特定する 6 つの構成要素からなるセットです。また、任意のローカルな基準フレームの位置を指すこともあります。

相対変換では、あるロケーションを別のロケーションの相対位置としたり、変換を相対的なものにするローカルな基準フレームを作成できます。例えば、ワークセルのロケーションを定期的に変更するアセンブリを作成できます。アセンブリのすべてのロケーションをワールド座標フレームに対して相対的にティーチし、毎回ワークセルの異なった位置にアセンブリを配置する場合、すべてのロケーションを再度ティーチする必要があります。しかし、アセンブリの識別可能な特徴に基づきフレームを作成すれば、フレームを定義するロケーションのみを再度ティーチするだけで済みます。詳細については、「ロケーション変数の修正 59 ページを参照してください。

プレジジョンポイントは、メカニズムの運動学に基づく各ジョイント位置の要素を含めることで、スペース内のロケーションを定義します。4 つの自由度を持つメカニズムには、4 つの要素を持つプレジジョンポイント、6 つの自由度を持つメカニズムには、6 つの要素が必要になります。回転ジョイントの値は度で測定され、並進ジョイントの値は mm で測定されます。これらの値は、ロボットハードウェアのキャリブレーションで決定するジョイントのゼロポジションに対する絶対値です。

ロケーション変数の作成

ロケーション変数を作成する最も簡単な方法は、ある位置にロボットを配置し、次のステートメントを実行します。

```
HERE loc_var
```

6 つの構成要素のいずれかを使用するか、または位置に到達したときのロボットのジョイントの状態を指定することによって、位置を指定できます。前者の方法の場合、変換変数になります。さまざまなアーム構成で使用できるため、変換は最も柔軟性が高く、効率的なロケーション変数です。また、異なるアーム位置で使用できるツールオフセットやスロットオフセットなど、さまざまなコンテキストで使用したり、同じパレットレイアウトのさまざまなパレットフレームにアクセスしながら使用できます。

プレジジョンポイントは、ロボットの各ジョイントのジョイント値を記録します。プレジジョンポイントは、より正確に各ジョイントを動かすことができるジョイント情報を抽出する唯一の方法です。先頭のシャープ記号 (#) でプレジジョンポイントを識別できます。

次のステートメントは、現在のロボットのジョイント値と等しいプレジジョンポイント「pick」を作成します。

```
HERE #pick
```

ロケーション変数の修正

次の例では、ロケーション変数を修正するさまざまな方法を示します。

追加情報: 変換とプレジジョンポイントは、SET または HERE プログラム命令キーワードで初期化できます。

ロケーション変数のコピー

次の例では、既存のロケーション変数をコピーし、「loc_value」の値を変数「loc_name」に代入します。

```
SET loc_name = loc_value
```

既存の変数の修正を元にロケーション変数を作成

次の例では、ロケーション変数「loc_name」を作成し、変数「loc_value」のロケーションから、X、YおよびZの正方向にそれぞれ5 mm シフトします。

```
SET loc_name = SHIFT(loc_value BY 5, 5, 5)
```

相対変換の使用

次のステートメントを使用して、図5-3. に示すロケーションを生成します。この図では、変換「loc_b」は、「loc_a」で定義されたローカルな基準フレームから「loc_c」で定義されたローカルな基準フレームまで移動するために必要な変換を定義しています。

次の例で使用しているコロンの(:) 演算子は、複合変換演算を表します。例えば、loc_a:loc_b は、loc_a に対する変換 loc_b です。loc_b は、図5-3. に示すように、完全に固有の独立したロケーション変換ですが、loc_a（ピッチ: 180度）に対して使用する場合は、loc_c まで移動する変換になります。

次のステートメントは、単純な変換を定義し、そのロケーションまでロボットを移動させます。

```
SET loc_a = TRANS(300,50,350,0,180,0)
MOVE loc_a
BREAK
```

次のステートメントは、「loc_a」の位置と方向を基準にして、X方向に-50 mm、Y方向に20 mm、Z方向に30 mmオフセットしたロケーションにロボットを移動します。

```
MOVE loc_a:TRANS(-50, 20, 30)
BREAK
```

次のステートメントは、「loc_a」を基準とした現在のロケーションを「loc_b」として定義します。「loc_b」は-50、20、30、0、0、0となります。

```
HERE loc_a:loc_b
BREAK
```

次のステートメントは、「loc_c」を「loc_a」と「loc_b」のベクトルとして定義します。「loc_c」は350、70、320、0、180、0となります。

```
SET loc_c = loc_a:loc_b
```

上記ステートメントを実行した後、「loc_b」は、「loc_a」と完全に独立した変換として存在します。次のステートメントは、「loc_c」に対して X 方向に -50 mm、Y 方向に 20 mm、Z 方向に 30 mm の別のロケーションにロボットを移動させます。

```
MOVE loc_c:loc_b
```

複数の変換を関連付けることができます。「loc_d」の値を 0, 50, 0, 0, 0, 0 に定義した場合、次のステートメントは、「loc_a」に対して X 方向に -50 mm、Y 方向に 70 mm、Z 方向に 30 mm のロケーションにロボットを移動させます。

```
SET loc_d = TRANS(0,50)
MOVE loc_a:loc_b:loc_d
```

変換を反対方向に進める必要がある場合（「loc_c」から「loc_a」）、次のステートメントで計算できます。

```
INVERSE(loc_b)
```

次のステートメントは、ロボットを「loc_a」に戻します。

```
MOVE loc_c:INVERSE(loc_b)
```

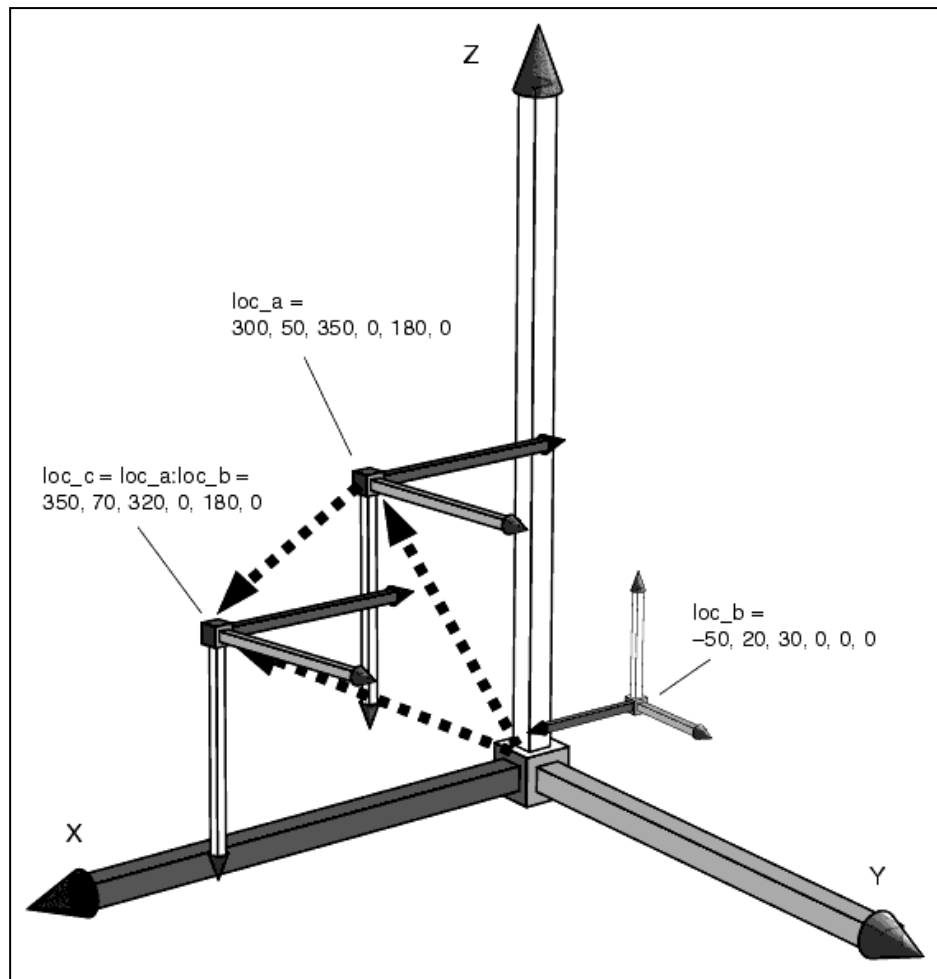


図5-3. 相対変換

ロケーション変数の分解

ロケーション変数の要素にアクセスするためには、ロケーション変数を配列に分解する必要があります。これはプレジジョンポイントと変換変数の両方に適用されます。

```
DECOMPOSE jnt.angles[0] = #PHERE
DECOMPOSE curr.pos[0] = HERE
curr.z.pos = curr.pos[2]
```

5.11 配列

eV+ は、最大 3 次元の配列までサポートしています。配列にはどの eV+ データタイプも保存できます。配列をオートとして宣言していない限り、配列サイズを宣言する必要はありません。この文書では、行が 1 番目の要素、列が 2 番目の要素、範囲が 3 番目の要素とします。

注: 配列の割り当ては、範囲の最上位のインデックスが列の最上位のインデックスを超え、列の最上位のインデックスが行の最上位のインデックスを超えた場合、最も効果的になります。

次のステートメントでは、1 次元配列「array.one」用のスペースを割り当て、配列の要素 2 に値 36 を格納します。角括弧 ([]) 内の数字は、インデックスとして参照されます。また、配列のインデックスには変数や式も使用できます。

```
array.one[2] = 36
```

次のステートメントでは、2 次元配列「array.two」用のスペースを割り当て、行4, 列5に「row 4, col 5」を格納します。

```
$array.two[4,5] = "row 4, col 5"
```

次のステートメントでは、3 次元配列「array.three」用のスペースを割り当て、行 2、列 2、範囲 4 に 10.5 の値を格納します。

```
array.three[2,2,4] = 10.5
```

配列がすでに宣言されている状態で上記のステートメントのいずれかを実行した場合、ステートメントは、単に適切な位置に値を格納するだけです。値のデータタイプが配列のデータタイプと異なる場合、エラーが発生します。

次のステートメントの結果として、配列要素0~15 が割り当てられます

```
any_array[2] = 50
```

次のステートメントの結果として、配列要素0~31 が割り当てられます。

```
any_array[2] = 50
any_array[20] = 75
```

グローバル配列へのアクセス制限

eV+ には、グローバル配列やローカル配列を使用する時に自動的に拡張する機能があります。効率性を維持するために、複数のタスク間で配列拡張処理のインターロックはありません。あるタスクが配列にアクセスしている間、別のタスクが配列を拡張したり削除すると、クラッシュが発生する可能性があります。以下の方法を使用してカスタム eV+ プログラムをコーディングし、この問題を回避する必要があります。

方法 1

配列の次元について、妥当な上限が既知である場合、任意の値を配列の最大要素に割り当てて定義します。多次元配列の場合、可能なサブ配列ごとに最大要素を割り当てます。このように割り当てることで、配列の拡張を防ぐことができます。

方法 2

TAS 関数キーワードを使用して、配列のアクセスをインターロックします。この場合、配列へのアクセスは、配列へのアクセスを制御する TAS を含む 1 つまたは 2 つのサブルーチンから排他的に処理されます。

5.12 変数のコンテキスト

キーワードを使用してローカル変数やオート変数にアクセスする場合、eV+ システムに対して、どのプログラムの変数にアクセスするかを知らせる必要があります。そのため、使用されるプログラムのコンテキストを指定する必要があります。ローカル変数やオート変数にアクセスできるコマンドの一般的な構文を次に示します。「command」はキーワード名、「parameters」はキーワードのパラメータです。

```
command @task:program parameters
```

オプションの要素「@task:program」は、キーワードパラメータで参照される変数のプログラムコンテキストを指定します。「task」の部分は、システムのプログラムタスクの 1 つを指定する整数です。コンテキストの「program」の部分は、システムメモリにあるプログラム名です。

コンテキストは次に示す形式で指定できます。使用可能なさまざまなコンテキスト指定の概要については、変数のコンテキスト指定の解釈 64 ページを参照してください。

1. 「task:program」は空白です（構文として「@」を使用）。参照される変数のコンテキストは、次のいずれかになります。
 - MCS キーワードでは空白のコンテキストを指定します。次の例では、変数が MCS キーワードを含むプログラム内で参照されるかのように処理されることを示します。この例で使用されるキーワードは、どのタスクがプログラムの実行に割り当てられているかにかかわらず、ルーチンで使用されるローカル変数を削除できます。

```
MCS "DELETED @ parts[]"
```

 - 変数は、ロボット制御タスク（STATUS キーワードでリストされた最後のプログラム、例えば、「STATUS 0」）の最上位のスタックにあるプログラム内で参照されるかのように処理されます。
2. 「Task」をタスク番号（0、1、2 など）として指定し、「:program」の部分は省略します。例えば、「@1」を入力します。この場合、変数は、指定のプログラムタスク（STATUS キーワードでそのタスク用に示される最後のプログラム、例えば、「STATUS 1」）の最上位のスタックにあるプログラム内で参照されるかのように処理されます。
3. 「Task」を省略し、「program」を指定します（例えば、構文では「@sample」を使用）。この場合、変数は、次のように決定されるタスクスタックで最近発生した指定のプログラム内で参照されるかのように処理されます。
 - Sysmac Studio でプログラムデバッガを使用していない場合、メインの制御タスク（task 0）のスタックが使用されます。
 - Sysmac Studio でプログラムデバッガを使用している場合、デバッガがアクセスしているスタックが使用されます。
4. 「task」にタスク番号（0、1、2 など）、「program」にプログラム名（例えば、構文では「@1:sample」を使用）を入力し、「Task:program」を両方指定します。この場合、変数は、指定のプログラムタスクのスタックで発生した最も最近のプログラム内で参照されるかのように処理されます。

コンテキストが指定されると、該当するものがある場合、ステートメントで参照されるすべての変数は、そのプログラム内ではローカル変数またはオート変数とみなされます。ステートメントで参照される変数がそのプログラム内にない場合、グローバル変数とみなされます。オート変数とサブルーチンの引数は特定のプログラムインスタンス内にのみ存在し、プログラムが終了すると変数もクリアされるため、モニタコマンドキーワードからこれらの変数へのアクセスは、次の条件により制限されます。

- タスクスタックにあるプログラムであること。つまり、STATUS キーワードで示されるプログラムリストに存在しているプログラムである必要があります。
- 参照されるスタックのプログラムタスクは、アクティブでないこと。これはアクセス中に変数がクリアされることを防ぐためです。

注: 上記の条件を満たさない場合、エラーメッセージ *Undefined value in this context*が表示されます。

デバッグ中にオート変数やサブルーチンの引数にアクセスする必要がある場合、目的のプログラムタスクを一時停止し、変数にアクセスした後にタスクの実行を再開することができます。

変数のコンテキスト指定の解釈

以下の表を使用して、変数のコンテキスト指定について説明します。

- 「T」は、アクセスされるタスクを示します。
- 「P」は、アクセスされるプログラムを示します。
- 「None」は、指定のタスクやプログラム（グローバル変数にアクセス）がないことを示します。
- 「現在のデバッグ」は、Sysmac Studio のデバッガでアクセス中のタスクまたはプログラムを示します。
- 「スタックの最上位」は、示されたタスクの実行スタックの最上位にあるプログラムを示します。

表5-6. 変数のコンテキスト指定

コンテキスト指定	デバッガを使用しない場合	デバッガを使用する場合
None	T = なし	T = 現在のデバッグ
	P = なし	P = 現在のデバッグ
@（注記参照）	T = タスク 0	T = 現在のデバッグ
	P = スタックの最上位	P = 現在のデバッグ
@number	T = タスク番号	T = タスク番号
	P = スタックの最上位	P = スタックの最上位
@program	T = タスク 0	T = 現在のデバッグ
	P = プログラム	P = プログラム
@number:program	T = タスク番号	T = タスク番号
	P = プログラム	P = プログラム

注: プログラム内の MCS キーワードで「@」が指定されている場合、T は現在のタスク、P は現在のプログラム（スタックの最上位）を指します。

第6章: モニタコマンドプログラム

本章では、モニタコマンドプログラムの機能とその使用方法について説明します。

6.1 Monitor Command Program の概要

モニタコマンドプログラムは、通常、連続したモニタコマンドキーワードで構成される、特殊なプログラムです。システムの起動や頻繁に実行する他のキーワードシーケンスの自動化に使用されます。モニタコマンドプログラムは、eV+ モニタウィンドウやティーチペンダントから開始できます。最初の行から最後の行までトップダウンで実行されます。

追加情報: モニタコマンドプログラムでプログラム命令キーワードを使用できません。詳細については、「モニタコマンドプログラムの内容 65 ページを参照してください。

オートスタート機能を使用すれば、システムを ON した時に モニタコマンドプログラム を自動的に開始できます。次の例に示すように、モニタコマンドプログラムは V+ ベースのアプリケーションで最も一般的に使用されます。

- システムの起動時、コントローラのオートスタートスイッチは、D:¥AUTO.V2 にある モニタコマンドプログラム「auto」を起動します。詳細については、「モニタコマンドプログラムの自動スタート 66 ページを参照してください。
- モニタコマンドプログラムを使用して、V+ モジュールやアプリケーション変数を保存できます。モニタコマンドプログラムは、モジュールのプログラムを定義し、バックアップ／保存操作を実行します。

Sysmac Studio を使用する場合、V+ モジュールオブジェクトがプロジェクトにプログラムを保存するため、上記の操作は必要ありません。同じことがV+ グローバル変数コレクションオブジェクトに含まれる V+ アプリケーション変数にも適用されます。

追加情報: V+ モジュールと V+ グローバル変数の詳細については、『Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合 CPUユニット編 (SBCA-464)』を参照してください。

6.2 モニタコマンドプログラム名

モニタコマンドプログラム名には最大 15 文字まで使用できます。モニタコマンドプログラム名は、英文字、数字、ピリオド、アンダースコアをから始める必要があります。

モニタコマンドプログラムは、eV+ のディスクファイルにオブジェクトとして保存されます。ディスクファイル名（コマンドプログラム名ではない）は、eV+ ディスクファイル名の規定に準拠する必要があります。詳細については、「ファイル名の要件 16 ページを参照してください。

6.3 モニタコマンドプログラムの内容

通常、モニタコマンドプログラムには、モニタコマンドキーワードのみ使用します。必要に応じて D0 モニタコマンドキーワードを使用して、プログラム命令キーワードを使用することができ

ます。モニタコマンドプログラム でプログラム命令キーワードを使用するには、次の構文のステートメントを使用します。

```
MC DO keyword parameter, ...
```

モニタコマンドプログラムの作成においては、次の事項を考慮する必要があります。

- モニタコマンドプログラム では、DONE、TEACH、ZERO キーワードを除くすべてのモニタコマンドキーワードを使用できます。
- モニタコマンドプログラム では、制御キーワード（GOTO、IFなど）は無視されます。
- モニタコマンドプログラムのすべての非空白行にはモニタコマンドキーワード、またはコメントを記述する必要があります。

重要:モニタコマンドプログラム が、オペレータの応答や介入がなくても実行できることを確認してください。

6.4 モニタコマンドプログラムの実行

モニタコマンドプログラム は、以下に示す方法で実行できます。

モニタコマンドプログラム からモニタコマンドプログラムを実行する

COMMANDS モニタコマンドキーワードを使用して、モニタウィンドウから モニタコマンドプログラム を実行します。

次のステートメントを使用して、ファイル名「my_cmd.pg」の モニタコマンドプログラム「my.cmd.program」を実行します。

```
LOAD my_cmd.pg  
COMMANDS my.cmd.program
```

注: EXECUTE キーワードは、モニタコマンドキーワードではなく、プログラム命令キーワードを含むプログラムを対象とします。EXECUTE キーワードを使用して「MC」で始まる行があるプログラムを実行しようとした場合、プログラムは停止し、プログラム命令キーワードとモニタコマンドキーワードを混合してはいけないことを示すメッセージが返されます。

モニタコマンドプログラムの自動スタート

次の条件を満たす場合、コントローラの電源を ON にしたとき、モニタコマンドプログラムが自動的に読み込まれスタートします。

- コントローラの設定で自動スタートオプションが有効に設定されている。
- デフォルトのディスクにディスクファイル「AUTO.V2」が存在する。
- ディスクファイル「AUTO.V2」には、ファイル名が「auto」という Monitor Command Program が存在する。ほかのプログラムやデータをディスクファイル「AUTO.V2」に格納することもできます。

上記条件を満たしている場合、コントローラの電源を ON にすると次の機能が実行されます。

- eV+ オペレーティングシステムが読み込まれます。
- デフォルトのディスク仕様が設定されます。

- 次のステートメントが自動的に発行されます。

```
LOAD auto.v2
COMMANDS auto
```

eV+ は、ハイパワーの ON やそれ以外のイベントまたは条件を待ちません。自動スタートしたモニタコマンドプログラムがこれらの機能を明示的に呼び出すか、またはモニタコマンドプログラムが呼び出すプログラムでこれらの機能を実行する必要があります。例えば、モニタコマンドプログラムを処理するときはオペレーターがコントローラのプログラムスタートボタンを押すようにしたい場合、モニタコマンドプログラムに WAIT.START ステートメントを記述する必要があります。

追加情報: 自動スタートしたモニタコマンドプログラムに最小限の情報を読み込ませた後で、より複雑なロジックやエラーを処理できる V+ プログラムを実行してください。

モニタコマンドプログラムの処理

モニタコマンドプログラムを実行すると、次のイベントのいずれかが発生するまで、すべてのステートメントを処理します。

- モニタコマンドプログラムの最後に達した。
- モニタコマンドプログラム内の CYCLE.END ステートメントが処理され、参照されたプログラムタスクが実行された。これにより、参照されたプログラムタスクの実行が終了するまで、モニタコマンドプログラムの処理は一時停止します。

注: EXECUTE ステートメントの処理中はモニタコマンドプログラムは一時停止しません。モニタコマンドプログラムは指定のアプリケーションプログラムの実行を開始し、ただちに次のステートメントを続行します。

- CTRL+C が押された。これにより、モニタコマンドプログラムの処理が終了します。処理中のステートメント、またはモニタコマンドプログラムが呼び出したプログラムは完了するまで継続します。
- アクティブなコマンドプログラム内から別のモニタコマンドプログラムが呼び出された。実行可能なプログラムを開始する場合とは異なり、2 番目のモニタコマンドプログラムが完了したとき、最初のモニタコマンドプログラムに制御には戻りません。
- モニタコマンドプログラムでステートメントの処理中にエラー条件が発生した。

第7章: V+ プログラム

本章では、V+ プログラム の作成とその使用方法について説明します。

7.1 V+ プログラム のフォーマット

V+ プログラムでは、ロボットの動作を生成するために使用され、I/O 制御、データ管理、さまざまなランタイムタスクなど他の操作のロジックと機能を含みます。V+ プログラムは、プログラム命令と関数キーワードを使用して作成します。

本項では、eV+ プログラムで順守すべきフォーマットについて記述します。プログラムの全体的構成に沿って、各行のフォーマットを説明します。この情報は、プログラムの使用目的にかかわらず、すべての V+ プログラム に適用されます。

プログラム行

eV+ システムは、プログラムのそれぞれの行やステップを一つのプログラムステートメントとして解釈します。eV+ プログラム の一般的なステップのフォーマットは、次に示す構文を使用します。

```
step_number step_label operation ;Comment
```

上記構文の各項目はオプションです。詳細を次の表に示します。

表7-1. プログラム行の各項目の説明

項目	説明
step_number(ステップ番号)	プログラム内の各ステップには自動的にステップ番号が割り当てられます。ステップ番号には連番が振られ、ステップが挿入または削除されると番号は自動的に調整されます。ステップ番号は自動的に V+ プログラム に表示されます。この番号は、V+ プログラム のエラーや他のプログラムの参照に使用されます。また、ステップ番号は行番号としても参照されます。
step_label(ステップラベル)	ステップ番号はプログラムの作成中に変更されることがあるため、プログラム制御分岐ステップを特定するのにステップ番号を使うことは有効的ではありません。そのため、プログラムステップにステップラベルを付けます。ステップラベルは、整数値（0～65535）で、参照するプログラム行の開始場所としてプログラム内のどの位置にも配置できます。通常、参照には GOTO ステートメントが使用されます。
operation(演算)	各ステップの演算部分は正しいステートメントでなければなりません。パラメータや追加のキーワードを含めることができます。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください
comment(コメント)	プログラム行の残りの部分をコメントアウトする場合、セミコロン (;) を使用します。eV+ は、コメントを無視します。プログラムステップのすべての要素を省略した場合、空白行となります。 プログラムステップがコメント要素のみの場合、そのステップはコメント行となり

項目	説明
	ます。コメントは、そのプログラム部分の目的を説明するために使用します。このようにプログラムに注釈を付けると、プログラムの修正やデバッグに役立ちます。 注: eV+ プログラムでは、空白行を使用できます。空白行は、プログラムステップを区切り、プログラムの可読性を高めます。

空白スペースと大文字／小文字

プログラム行を入力するとき、行内の要素間に空白スペースを追加できます。eV+ エディタは、プログラム行のスペースを追加または削除して、標準の間隔に直します。また、エディタはすべてのキーワードを大文字にすべてのユーザー定義名を小文字に自動的に変換します。

追加情報: プログラム行の入力後、改行を入力する、別の行に移動する、エディタを終了すると、エディタはその行の構文を確認します。実行できない行は、Sysmac Studio V+ エディタ内で赤色に表示されます。詳細については、『*Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合 CPUユニット編 (SBCE-464)*』を参照してください。

プログラム構造

すべての V+ プログラム の最初のステップは、.PROGRAM キーワードにする必要があります。このキーワードにより、プログラムに名前が付けられ、プログラムが受け取る、または返す引数が定義されます。次にフォーマットを示します。

```
.PROGRAM program_name(parameter_list) ;Comment
```

program_name 項目は必須ですが、パラメータリストとコメントはオプションです。

最初の行の .PROGRAM ステートメントに続くほかのステートメントの順序については次に示す 2 つの制限があります。

1. AUTO、EXTERNAL、LOCAL、GLOBAL キーワードステートメントは、プログラムの上部に、他の標準のキーワードステートメントより前に記述する必要があります。コメント行、空白行、その他の AUTO、LOCAL、GLOBAL キーワードのみ、.PROGRAM ステップと AUTO、LOCAL、GLOBAL キーワードステートメント間に記述できます。
2. プログラムの最終行は .END で始めます。V+ エディタはプログラムの最終行に自動的に .END 行を追加します。

注: Sysmac Studio プログラムエディタは、.PROGRAM と .END ステートメントを自動的に挿入します。別のプログラムエディタを使用する場合、この 2 行を手作業で入力する必要があります。通常、プログラミングにはフォーマットされていない ASCII ファイル を生成するエディタを使用できます。

7.2 V+ プログラムの実行

eV+ システムでは、最大 64 個のプログラムまで同時に実行できます。各プログラムの実行は、別々のプログラムタスクとしてシステムで管理されます。プログラムの開始方法は、使用するプログラムタスクと実行するプログラムの種類で異なります。

次の項では、プログラム実行の詳細について説明します。

プログラムタスク

V+ プログラム は 64 個の使用可能タスクの 1 つで実行されます。タスク番号 0 は最も優先度が高く、通常、主要なアプリケーションプログラムに使用されます。タスク 0 は、通常、主なロボット制御プログラムの実行に使用されます。

タスク番号 0 の実行が開始すると、タスクは自動的にロボット 1 を選択し、モーション関連キーワードが実行されるとすぐにロボットをアタッチします。通常、EXECUTE モニタコマンドキーワードを発行してタスク 0 の実行を開始します。ABORT モニタコマンドまたはプログラム命令キーワードは、現在のロボットの動作が完了した後、タスク番号 0 を停止します。CYCLE.END モニタコマンドまたはプログラム命令キーワードを使用して、現在の実行サイクルの終了時にプログラムを停止できます。

通常、ロボット制御プログラムを実行する場合、システムスイッチキーワード DRY.RUN を無効にして、ロボット制御プログラムでロボットをアタッチする必要があります。エラー復元プログラムを構築していない限り、ロボット関連のエラーによってプログラムの実行は停止されます。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』の「REACTE」を参照してください。

エラーや PAUSE、ABORT ステートメントのためにプログラムの実行が停止した場合、モニタコマンドキーワードの PROCEED や RETRY で実行を再開できます。

追加情報: 実行が停止している間、DO モニタコマンドキーワードを使用して、あるプログラムステートメントをあたかも停止しているプログラムの次のステートメントであるかのように手動で実行することができます。

タスク 0 以外のプログラムタスクの実行もタスク 0 の場合とほぼ同じです。異なる点を次に示します。

- EXECUTE、ABORT、PROCEED、RETRY などプログラムの実行に影響するすべてのステートメントに、明示的にタスク番号を記述する必要があります。
- プログラムがロボットを制御する場合、ロボットを制御するステートメントを実行する前に、SELECT と ATTACH ステートメントを明示的に記述する必要があります。

ロボットの排他的制御

次に、タスクでロボットを排他的制御する場合の制限について示します。

- アクティブなタスクでロボットをアタッチする場合、REACTI と BRAKE キーワードを除き、他のタスクでそのロボットをアタッチしたり、ロボットに影響するステートメントを実行できません。詳細については、V+ プログラムの割り込み 78 ページを参照してください。
- 何らかの理由でロボット制御タスクが停止する場合、タスクが自動的にロボットを再度アタッチしてタスクが再開するまで、ロボットはデタッチされます。その間に別のタスクがそのロボットをアタッチした場合、最初のタスクは再開できません。
- プログラムの実行が開始すると、タスク 0 は常にロボット 1 とのアタッチを試みます。SELECT と ATTACH ステートメントを明示的に実行しない限り、ほかのどのタスクもどのロボットにもアタッチできません。
- タスク 0 がロボット 1 とのアタッチを試みたとき、すでに別のタスクがそのロボットとアタッチされている場合、タスク 0 は実行できません。別のタスクがロボットを制御する必

要があり、タスク 0 を実行したい場合、次に示すイベントシーケンスに従う必要があります。

1. タスク 0 を開始します。
2. タスク 0 をロボット 1 からデタッチします。
3. ロボット 1 を制御するタスクを開始します。タスク 0 として実行するプログラムで別のタスクを開始できます。
4. そのもう一つのタスクをロボットにアタッチします。

注: DRY.RUN モードでもロボットをアタッチできます。この場合、タスクが発行したモーションステートメントは無視され、ほかのタスクはロボットにアクセスできません。

EtherCAT通信の異常

通信の異常は次に説明するようにタスクおよびロボットの挙動に影響を与えます。

EtherCAT ネットワーク通信エラー

EtherCAT ネットワーク通信エラーは次に示すシステム挙動を引き起こします。

- すべての実行中のタスクは停止し、ホストシステムとの通信を必要とするタスクは実行できなくなります。すべてのロボットが切り離されます。
- ネットワーク内のすべてのロボットが最大減速度で停止し、ハイパワーが無効になります。

EtherCAT ノード通信エラー

個々のノードに対してネットワーク通信が停止すると、次のイベントが発生します。

- 問題のあるノードに関連する実行中のタスクが停止し、関連するロボットが切り離され、ホストシステムとの通信を必要とするタスクは実行できなくなります。
- 通信の失われたロボットノードは割り付けられているタスクの実行を停止します。

V+ プログラム の処理

通常、プログラムのステートメントはプログラムの最初から最後まで順番に実行されます。この実行順は GOTO、IF...GOTO ステートメント、制御構造がある場合、変更されます。

CALL ステートメントは別のプログラムを呼び出し実行しますが、呼び出されたプログラムが RETURN または RETURNE ステートメントを実行すると、呼び出したプログラムは前回の停止位置から実行を再開するため、呼び出したプログラムの実行順序が変わることはありません。

WAIT ステートメントは、条件が満たされるまで現在のプログラムの実行を一時停止します。WAIT.EVENT ステートメントは、指定のイベントが発生するか、または指定の時間が経過するまで、現在のプログラムの実行を一時停止します。

PAUSE、HALT ステートメントは、現在のプログラムの実行を終了します。PAUSE ステートメントの後に、PROCEED モニタコマンドを使ってプログラムの実行を再開することができます。HALT プログラム命令はブレークを発生し、未完了のプログラムループの有無に関わらずアプリケーションプログラムの実行を終了します。HALT コマンドでプログラムを終了した後、PROCEED または RETRY コマンドでプログラムを再開することはできません。

STOP ステートメントでは、プログラムの実行を終了できる場合とできない場合があります。複数のプログラム実行サイクルが動作中の場合、STOP ステートメントはサブルーチンに記述されたとしても、メインのプログラムを最初のステップから再開します。残された実行ループが無い場合、STOP ステートメントは現在のプログラムを終了します。詳細については、「V+ プログラムの割り込み 78 ページを参照してください。

7.3 プログラム制御

本項では、プログラムの実行を制御するためにeV+ で利用可能な構造やその他の方法について説明します。eV+ では、多くの高級言語で一般的なループ分岐構造や eV+ 独自のキーワードをサポートしています。

無条件分岐

本項では、無条件分岐で使用する次のキーワードについて説明します。

- GOTO
- CALL
- CALLS

GOTO 無条件分岐

GOTO キーワードは、プログラムの実行をプログラム内の別の場所にあるラベルに、ただちに分岐します。GOTO ステートメントの構文を次に示します。

```
GOTO label
```

上記の label は、プログラムコード行の先頭に入力する整数です。ラベルはプログラムのステップ番号とは異なります。ステップ番号はシステムによって割り当てられますが、ラベルはプログラマーがプログラムコード行の先頭に入力します。

次の例では、ラベル 100 に移動し、TYPE ステートメント「The GOTO keyword got me here.」を実行します。

```
GOTO 100

100 TYPE "The GOTO Keyword got me here."
```

GOTO ステートメントは、GOTO キーワードの前後にあるラベルに分岐できます。

重要:GOTO キーワードを使用する場合注意が必要です。GOTO ステートメントは、特に多くのサブルーチンの呼び出しがある長く複雑なプログラムの場合、プログラムのロジックの追跡やデバッグを困難にします。GOTO は、一般的にルーチンの終了、またはエラーステートメントの終了に使用します。

CALL 無条件分岐

CALL キーワードは、プログラムの実行を一時停止し、新しいプログラムの実行を開始します。新しいプログラムの実行が完了すると、呼び出し元プログラムの CALL キーワードの次のステートメントから実行を再開します。

CALL ステートメントの簡単な構文を以下に示します。

```
CALL program_1(arg_list)
```

上記の「program_1」は呼び出されるプログラムの名前です。プログラム名は正確に指定する必要があります、呼び出されるプログラムはシステムメモリ上に存在していなければなりません。

上記の「arg_list」項目は、サブルーチンに渡される引数のリストです。これらの引数は、値渡し、または参照渡しが可能であり、呼び出されるプログラムに必要な引数と一致している必要があります。詳細については、「プログラム引数リストを使用した情報の交換 85 ページを参照してください。

次の例では、呼び出しプログラムの実行を一時停止して、引数「locx」、「locy」、「length」をプログラム「check_data」に渡し、「check_data」を実行します。「check_data」の実行が完了した後、呼び出しプログラムの CALL 命令に続くステートメントから実行を再開します。

```
CALL check_data(locx, locy, length)
```

CALLS 無条件分岐

CALLS キーワードはプログラムの指定方法以外は CALL キーワードと同じです。CALLS キーワードの場合、文字列パラメータは文字列値、変数、式です。そのため、同じコード行を使用して異なる条件でさまざまなサブルーチンを呼び出すことができます。これらのサブルーチンは、引数リストを同じにする必要があります。

CALLS ステートメントの簡略化した構文を以下に示します。

```
CALLS string(arg_list)
```

次の例では、呼び出しプログラムの実行を一時停止して、変数「length」と「width」を配列「\$program_list」の配列のインデックス「program_select」で指定されたプログラムに渡し、指定のプログラムを実行します。その後、呼び出しプログラムの次のステップから実行を再開します。

```
$program_name = $program_list[program_select]
```

```
CALLS $program_name(length, width)
```

条件分岐

本項では、ブール値（TRUE または FALSE の変数、または解が TRUE または FALSE となる式）を持つ式や変数に応じて実行が異なるプログラム制御構造について説明します。式の最終結果がブール値である限り、式では、任意の数の変数またはデジタル入力信号を入れることができます。

条件分岐キーワードでは、プログラム変数または式の現在の値に基づいてコードブロックを実行できます。eV+ には、次に示す 3 つの条件分岐キーワードがあります。

1. IF...GOTO
2. IF...THEN...ELSE
3. CASE value OF

どの数値（実数または整数値）でもこの要件を満たすことができます。0 は false、0 以外は true とみなされます。システムの定数は 4 種類あり、TRUE と ON は -1、FALSE と OFF は 0 を表します。

IF...GOTO 条件分岐

IF...GOTO の動作は GOTO と似ていますが、条件に従って分岐する点異なります。

次の例では、「logical_expression」が true の場合、ラベル 100 に分岐します。

```
IF logical_expression GOTO 100
```

IF...THEN...ELSE 条件分岐

IF...THEN...ELSE は、論理式の結果に応じて、2 つの異なるアクションのうち 1 つのアクションを条件付きで実行します。

次に示すように、このキーワードには 2 つの型があります。

```
IF expression THEN
    statement
END
```

```
IF expression THEN
    statement
ELSE
    statement
END
```

次の例では、「num_parts」が 75 より大きい場合、CALL check_num(num_parts) を実行します。「num_parts」が 75 以下の場合、END に続くステートメントからプログラムの実行を継続します。

```
IF num_parts > 75 THEN
    CALL check_num(num_parts)
END
```

次の例では、信号 1033 が ON で「need_part」が true の場合、プログラムは MOVE ステートメントと DEPART ステートメントを実行します。IF ステートメントが false の場合、TYPE ステートメントを実行し、END に続くステートメントからプログラムの実行を継続します。

```
IF SIG(1033) AND need_part THEN
    MOVE loc1
    DEPART 50
ELSE
    TYPE "Part not picked up"
END
```

CASE value OF 条件分岐

CASE 構造では、変数の値に基づき、多くの異なるアクションのうち 1 つのアクションを実行できます。変数には、実数または整数値を使用する必要があります。

次に CASE 構造の型を示します。実数値リスト「list_of_values」（コンマ区切り）が「target」と等しい場合、VALUE ステートメントに続くステートメントが実行されます。「target」が「list_of_values」のリストにない場合、ANY ステップの後のステートメントが実行されます。

```
CASE target OF
    VALUE list_of_values
        statements
    VALUE list_of_values
        statements
ANY
    statements
END
```

ループ構造

多くの場合、プログラムで同じステートメントを複数回実行する必要があります。次に示すように、eV+ にはステートメントを複数回実行できる 3 種類のループ構造があります。

FOR ループ構造

FOR キーワードは、所定のステートメントブロックを指定された回数実行する実行ループを作成します。

FOR ループの基本形式を次に示します。このループは、最初に「start_val」の値を「index」に設定します。次に、「index」の値が「start_val」から「end_val」の範囲内にあるか確認します。範囲内であれば、FOR ループ内のステートメントを実行します。ループを実行した後、「index」の値を「incr」の値分インクリメントします。「index」の値が「start_val」から「end_val」の範囲外に達するまで、この動作を継続します。

```
FOR index = start_val TO end_val STEP incr
    statement
    ...
    statement
END
```

以下の例では、インデックスが 32 になるまで配列「\$names」の偶数要素を出力します。

```
FOR i = 2 TO 32 STEP 2
    TYPE $names[i]
END
```

以下の例では、2 次元配列「values」の値を 10 行 × 10 列分出力します。

```
FOR i = 1 TO 10
    FOR j = 1 to 10
        TYPE values[i,j], /S
    END
    TYPE " ", /C1
END
```

以下の例では、ステップインクリメントに負の値を設定することで、10 から 1 までカウントダウンします。

```
FOR i = 10 TO 1 STEP -1
    TYPE i
END
```

FOR ループ内でインデックスの値を変更すると、不適切なループ動作になります。インデックスに関する問題を回避するためには、インデックスの変数をオート変数とし、FOR ループ内でインデックスを変更しないことが重要です。実行後に開始と終了の変数を変更しても、FOR ループには影響しません。

注: FOR ループを必要以上に頻繁に実行した場合、処理時間を浪費することになります。これを回避するためには、FOR ループ内に WAIT または WAIT.EVENT ステートメントを挿入してください。

DO...UNTIL ループ構造

DO...UNTIL は、不特定回数、ステートメントグループを実行するループ構造です。ループを制御するブール式または変数が true になるとき、ループは終了します。ステートメントグループを 1 回実行するごとにブール値を評価し、式が true になれば、ループの実行は停止します。ステートメントを実行するまで式を評価しないため、コードブロックは最低 1 回は実行されます。

注: 式はブール型ステートメントであり、最終的に true になる必要があります。そうでない場合、無限にループを実行します。

このループ構造の形式を次に示します。

```
DO
    Statement
    ...
    Statement
UNTIL expression
```

以下の例では、モニタウィンドウに数字の 1~100 を出力します。

```
x = 1
DO
    TYPE x
    x = x + 1
UNTIL x > 100
```

以下の例では、モニタウィンドウに最大 15 文字までエコーします。このループは、15 文字エコーしたとき、または「#」文字が入力されたとき停止します。

```
x = 1
DO
    PROMPT "Enter a character: ", $ans
    TYPE $ans
    x = x + 1
UNTIL (x > 15) or ($ans == "#")
```

WHILE...DO ループ構造

WHILE...DO は、ループの最後ではなく最初にブール式を評価する点を除き、DO...UNTIL と同様のループ構造です。WHILE...DO ステートメントの式が示す条件が true の場合、ループ内のコードが実行されます。WHILE...DO ループは、DO...UNTIL ループと同じように無限ループになる可能性があります。ループを終了するためには、ループを制御する式が最終的に false になる必要があります。

WHILE...DO ループ構造の形式を次に示します。

```
WHILE expression DO
    statement(s)
END
```

入力を有効にするために WHILE...DO ループを使用する例を次に示します。ループを実行する前にブール式が評価されるため、許容できない値を入力されたときのみ、ループ内のコードを実行します。

```
PROMPT "Enter a number in the range 32 to 64.", ans
WHILE (ans < 32) OR (ans > 64) DO
    PROMPT "Number must be in the range 32-64.", ans
END
```

上記のコードで、数値以外の値を入力した場合、プログラムの実行は停止します。よりロバストな設計にするために、PROMPT ステートメントで文字列変数を使用し、\$DECODE、VAL キーワードで入力を評価してください。

以下の例では、WHILE ステートメントでデジタル信号 1033 が ON の場合、ループは実行されず、END キーワードの次からプログラムは継続します。デジタル信号 1033 が OFF の場合、信号が ON になるまでループを継続的に実行します。

```
WHILE NOT SIG(1033) DO  
  
END
```

システムメモリへのプログラムと変数の読み込み

eV+ は、ディスクファイルに保存されているプログラムと情報をシステムメモリ（RAM）に読み込むまで、プログラムと情報を使用できません。ファイルは、Sysmac Studio または LOAD モニタコマンドキーワードを使用して読み込むことができます。詳細については、『*Sysmac Studio ロボット統合システム構築機能 オペレーションマニュアル ロボット統合CPUユニット編 (SBCA-464)*』を参照してください。

LOAD キーワードはプログラムの実行を開始しません。別のコマンドがプログラムを開始、変更したり、プログラム変数の値を変更できるように、単にディスクファイルの内容のコピーをシステム RAM に配置するだけです。

LOAD キーワードの構文を次に示します。

```
LOAD my_file
```

このステートメントは、「my_file」の内容を RAM に配置します。

注: 1つのディスクファイルには、複数のプログラムと変数情報を格納できます。詳細については、eV+ のファイル管理 16 ページを参照してください。

追加情報: メモリに読み込まれたすべてのプログラムを表示するには、DIRECTORY キーワードを使用します。

SD カードのすべてのファイルを表示するには、FDIRECTORY キーワードを使用します。

システムメモリからプログラムと変数を削除

プログラムが終了しても、プログラムはシステムメモリ（RAM）から自動的に削除されません。ディスクファイルの読み込み中に *Out of memory* エラーメッセージが表示された場合、読み込み操作を完了する唯一の方法は、システムメモリからオブジェクトを削除することです。他のプログラムとデータがシステムメモリを使用できるようにするためには、DELETE キーワードでメモリ内のオブジェクトを明確に削除する必要があります。

注: LOAD キーワードは、システムメモリ内のプログラムを上書きしません。そのため、同じ名前の新しいプログラムを読み込む場合、現在システムメモリにあるプログラムを削除する必要があります。

すでにシステムメモリにあるプログラムと同じ名前のプログラムを読み込む場合以外は、別のディスクファイルを読み込んだり、他のプログラムを実行する前に、オブジェクトを削除する必要はありません。

DELETE キーワードは、実行中のプログラム、またはタスクのスタックにあるプログラムを削除しません。DELETE キーワードは、オブジェクトの読み込み元であるディスクファイルには影響しません。システムメモリからオブジェクトを削除した後、LOAD コマンドでそのオブジェクトを再度読み込むことができます。

7.4 V+ プログラムの割り込み

eV+ には、プログラムの実行を一時停止、または終了する複数の方法があります。

- WAIT キーワードを使用すると、特定の条件が TRUE になるまでプログラムを一時停止できますが、プロセッサの時間を効率的に使用できない可能性があります。詳細については、WAIT.EVENT プログラム割り込み 79 ページを参照してください。
- WAIT.EVENT キーワードで、指定の時間、または別のタスクにイベントが発生するまでプログラムを一時停止できます。
- REACT および REACTI キーワードにより、デジタル信号の状態の変化に基づいてプログラムを中断することができます。詳細については、REACT、REACTI プログラム割り込み 80 ページを参照してください。
- REACTE キーワードでプログラムエラーをインターセプトし、処理できます。
- HALT、STOP、PAUSE キーワードでプログラムの実行を終了できます。

これらのキーワードはそれらが記述されているプログラムに割り込むことができます。プログラムを実行している他のタスクには影響しません。BRAKE、DELAY キーワードでロボットの動作を制御できます。ABORT、PROCEED モニタコマンドキーワードを使用して、プログラムを一時停止したり、再開することもできます。

以下に、さまざまなプログラム割り込み方法を示します。

WAIT プログラム割り込み

WAIT キーワードは、条件が TRUE になるまでプログラムの実行を一時停止します。

WAIT キーワードの構文を以下に示します。

```
WAIT 条件
```

次の例では、デジタル入力信号 1032 が ON、1028 が OFF になるまで実行を一時停止します。

```
WAIT SIG(1032, -1028)
```

次の例では、タイマー 1 が 10 より大きい値を返すまでプログラムの実行を一時停止します。

```
WAIT TIMER(1) > 10
```

WAIT.EVENT プログラム割り込み

WAIT.EVENT キーワードは、指定のイベントが発生するまで、または指定の時間が経過するまでプログラムの実行を一時停止します。

WAIT.EVENT キーワードの構文を以下に示します。

```
WAIT.EVENT mask, timeout
```

追加情報: このキーワードは、タイマーの値を継続的に確認するのにタスクがループする必要がないため、タイマー (WAIT TIMER(1) > 10) を待機するよりも効果的です。

次の例では、3.7 秒間、プログラムの実行を一時停止します。

```
WAIT.EVENT, 3.7
```

次の例では、待機中のタスクに対して別のタスクが SET.EVENT ステートメントを発行するまで、プログラムの実行を一時停止します。

```
WAIT.EVENT
```

注: SET.EVENT 動作が発生しない場合、タスクは無限に待機します。

REACT、REACTI プログラム割り込み

REACT、REACTI キーワードが実行されると、プログラムは REACT ステートメントで指定されたデジタル入力信号のモニタを開始します。指定の信号が遷移するまでプログラムの実行は通常通り継続し、バックグラウンドでこの信号をモニタします。遷移を検出すると、プログラムは現在の実行ステップで実行を一時停止し、指定のサブルーチンを呼び出します。

さらに、REACTI は BRAKE 命令を発行し、ただちに現在のロボットの動作を停止します。どちらのキーワードでも、デジタル入力信号の遷移を検出したときに実行するサブルーチンを指定します。指定のサブルーチンが完了した後、デジタル入力信号の遷移を検出したときに実行していたステップからプログラムの実行を再開します。

デジタル信号を REACT キーワードと共に使用することができます。REACT/REACTI で開始した信号のモニタリングは、別の REACT/REACTI または IGNORE キーワードが実行されるまで有効です。IGNORE または 2 番目の REACT/REACTI キーワードが実行される前に、指定の信号の遷移を検出しない場合、REACT/REACTI キーワードはプログラムの実行に影響しません。次の信号を使用することができます。

信号タイプ	範囲	追加情報
デジタル入力	1001～1999	その信号が実際に存在し、設定されている場合に限って信号番号を使用することができます。
ソフトウェア	2001～2999	
外部	4001～4999	信号番号はマッピングされていれば使用できます。

追加情報: REACT または REACTI サブルーチンが完了した後、LOCK キーワードを使用してプログラムの実行を制御できます。

REACTE と REACTI キーワードの構文を以下に示します。

```
REACT signal_number, program, priority
REACTI signal_number, program, priority
```

次の例では、信号 1001 の状態が ON から OFF に遷移するかを評価する REACT ルーチンを実行します。遷移を検出した場合、サブルーチン「alarm」を呼び出し、優先度を 10 に設定し、次のステップで実行を再開します。メインのプログラムのデフォルトの優先度は 0 です。

MOVE c ステートメントの実行中に信号 1001 が遷移した場合、プログラムの優先度が REACT より高いため、MOVE c、MOVE d、LOCK 0 は完了します。サブルーチン「alarm」が呼び出され、MOVE e ステートメントで実行を再開します。LOCK ステートメントは、プログラムの応答ロックアウトの優先度を 20 の値に設定します。

```
REACT -1001, alarm, 10

MOVE a
MOVE b
LOCK 20
MOVE c
MOVE d
LOCK 0
MOVE e

IGNORE -1001
```

REACTE プログラム割り込み

REACTE は、プログラムの実行を停止するシステムエラーが発生した場合、応答プログラムの実行を有効にします。システムエラーには、すべてのロボットエラー、ハードウェアエラー、大部分のシステムエラー（I/O エラー以外）が含まれます。REACT、REACTI とは異なり、優先度の留意事項に基づき、REACTE を保留することはできません。

REACTE キーワードの構文を次に示します。

```
REACTE program_name
```

次の例では、システムエラーをモニタします。システムエラーが発生した場合、プログラム「trouble」を実行します。

```
REACTE trouble
```

HALT、STOP、PAUSE プログラム割り込み

HALT キーワードに遭遇すると、プログラムの実行が停止し、FCLOSE操作によって開いているディスクユニットが切り離されて閉じます。PROCEED または RETRY は実行を再開しません。

STOP キーワードが実行されると、現在実行中のプログラムサイクルは終了し、プログラムの最初のステップから次の実行サイクルを再開します。最後の実行サイクルで STOP キーワードに遭遇すると、プログラムの実行が停止し、FCLOSE操作によって開いているディスクユニットが切り離されて閉じます。PROCEED または RETRY は実行を再開しません。

PAUSE キーワードが実行されると、プログラムの実行は一時停止します。一時停止した後、システムプロンプトが表示され、モニタコマンドキーワードを実行できます。ここでプログラム変数の値を確認したり、システムパラメータを設定できます。これはプログラムのデバッグに役立ちます。モニタコマンドキーワード PROCEED は、PAUSE キーワードで割り込まれたプログラムの実行を再開します。

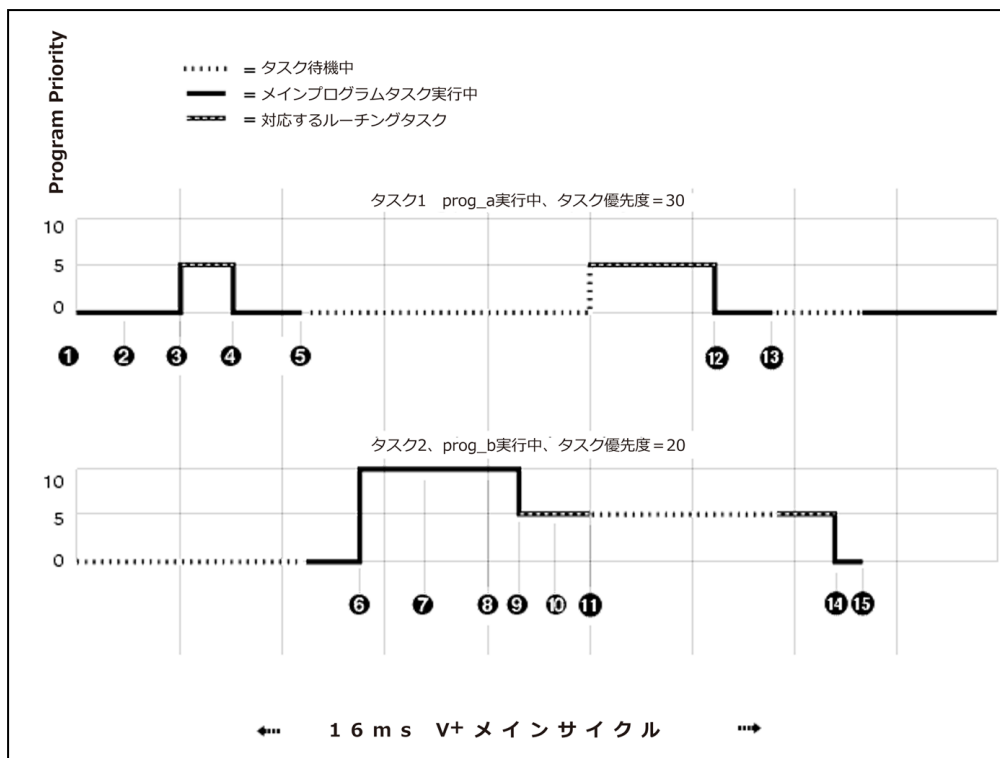
追加のプログラム割り込みキーワード

I/O キーワード ATTACH、READ、GETC、WRITE のパラメータを指定することで、I/O リクエストが完了するまでプログラムを一時停止することができます。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

プログラム割り込みの例

次の図にタスクとプログラムの優先度がどのように機能するかを示します。また、非同期のプログラム割り込みステートメントが優先度の枠組みの中でどのように機能するかについても示します。この例では、次の状態を想定しています。

- タスク 1 はすべてのタイムスライスで優先度 30 で実行する。
- タスク 2 はすべてのタイムスライスで優先度 20 で実行する。
- すべてのシステムタスクは無視される。
- すべてのシステム割り込みは無視される。



プログラム実行のタイミングを上図に示します。実線は、プログラムが実行していること、点線はプログラムが待機していることを示します。Y 軸はプログラムの優先度を示し、X 軸は主要サイクルを 16 ms ごとに分割しています。

この例では、各タスクで REACT ルーチンを有効にして、2 つのタスクを同時に実行しています。LOCK キーワードと REACT ルーチンのトリガはプログラムの優先度を変更します。この例のイベントシーケンスを次に示します。

1. タスク 1 は、プログラム「prog_a」をプログラム優先度 0 で実行しています。信号 1003 に基づく応答プログラムは優先度 5 で有効になっています。
2. 信号 1003 は外部からアサートされます。信号の遷移は、次の主要サイクルまで検出されません。
3. 信号 1003 の遷移が検出されます。タスク 1 の応答プログラムが実行を開始し、「prog_a」に割り込みます。
4. タスク 1 の応答プログラムは再度有効になり、RETURN 命令を発行して完了します。「prog_a」はタスク 1 の実行を再開します。
5. タスク 1「prog_a」は CLEAR.EVENT 命令を発行し、次に WAIT.EVENT 命令を発行してイベントフラグがセットされるまで待機します。タスク 1 は一時停止し、タスク 2 は「prog_b」の実行を再開します。タスク 2 は、優先度 5 で有効になる信号 1010 に基づく応答プログラムです。
6. タスク 2「prog_b」は LOCK 10 ステートメントを発行し、プログラムの優先度のレベルを 10 に上げます。
7. 信号 1010 は外部からアサートされます。信号の遷移は、次の主要サイクルまで検出されません。

8. 信号 1010 の遷移が検出され、タスク 2 の応答がトリガーされます。しかし、応答のレベルは 5 で、現在のプログラム優先度は 10 のため、応答の実行は保留になります。
9. タスク 2 「prog_b」は LOCK 0 ステートメントを発行し、プログラムの優先度をレベル 0 に下げます。レベル 5 の応答プログラムが保留中のため、ただちに実行を開始し、プログラムの優先度を 5 にセットします。
10. 信号 1003 は外部からアサートされます。信号の遷移は、次の主要サイクルまで検出されません。
11. 信号 1003 の遷移が検出され、タスク 1 の応答ルーチンをトリガーし、タスク 1 のイベントフラグをセットします。タスク 1 の優先度 (30) は、タスク 2 (20) よりも高いため、タスク 1 は REACT ルーチンの実行を開始し、タスク 2 は一時停止状態になります。
12. RETURN キーワードでタスク 1 の応答ルーチンが完了し、タスク 1 の「prog_a」に制御が戻ります。
13. タスク 1 「prog_a」は CLEAR.EVENT 命令を発行し、次に WAIT.EVENT 命令を発行してイベントフラグがセットされるまで待機します。タスク 1 は一時停止状態になり、タスク 2 は REACT ルーチンの実行を再開します。RETURN キーワードを発行してタスク 2 の REACT ルーチンは完了します。タスク 2 の「prog_b」に制御が戻ります。
14. タスク 2 「prog_b」は SET.EVENT 1 ステートメントを発行し、タスク 1 のイベントフラグをセットします。タスク 2 は RELEASE ステートメントを発行し、CPU に制御を戻します。
15. タスク 1 のイベントフラグがセットされ、優先度がタスク 2 よりも高いため、タスク 1 は実行を再開し、タスク 2 は一時停止状態になります。

7.5 V+ プログラムのスタック

プログラムが呼び出されると、eV+ は、実行プログラムが必要とする情報をスタックと呼ばれる内部の記憶領域に保存します。保存される情報を次に示します。

- 呼び出しプログラムの名前とステップ番号
- サブルーチンの引数に必要なデータ
- 呼び出されるプログラムで指定されたオート変数の値

eV+ システムでは、各プログラムタスクのスタックに記憶領域を明示的に割り当てることができます。特定のアプリケーション用にスタック領域の容量を調整し、システムメモリの使用を最適化できます。スタックは、任意の大きさに増やしたり、システムで使用できるメモリ量に制限することができます。

V+ プログラムのスタック要件

V+ プログラム を所定のタスク内で実行すると、各プログラムスタックに 32 キロバイトのメモリが割り当てられます。必要なスタック要件を決定した後、STACK モニタコマンドキーワードを使用してこの容量を調整できます。通常、これは モニタコマンドプログラム の起動時に実行されます。

スタック要件の決定

プログラムタスクのスタック要件を判別する 1 つの簡単な方法は、そのプログラムを実行することです。スタック領域がプログラムでいっぱいになった場合、エラーメッセージ*Not enough program stack space*が表示され、プログラムは停止します。スタックサイズを増やした後、プログラムが最初から再開しないように RETRY キーワードを発行してプログラムの実行を継続してください。

追加情報: STATUS キーワードは、タスク要求が失敗したときのスタック容量を返します。

別の方法としてプログラムを実行する前に、スタックサイズを大きく設定することができます。プログラムを実行し、すべての実行パスを進んだ後、STATUS モニタコマンドキーワードを使用して、プログラムタスクのスタック統計を確認します。スタックの最大値は、プログラムタスクを実行するために必要なスタック容量を示します。スタックサイズを調整し、タスクのスタック容量要件を満たすことができます（スタックサイズは、余裕を持たせるために 10～20% 程度多く設定することを推奨します）。

すべての実行可能なパスを呼び出すことができない場合、次の表を参考に、理論的なスタックの上限を計算することができます。一度に有効にできるすべてのプログラム呼び出しのオーバーヘッドを合計して、ワーストケースのスタックサイズを計算できます。合計値を 1024 で除算して、サイズをキロバイト単位で算出します。STACK モニタコマンドステートメントでこの数値を使用して、サイズを設定します。

表7-2. スタック容量の要件

項目	スタック容量の要件 (バイト)
サブルーチンの呼び出し	20
各サブルーチン引数（および以下のいずれか 1 つ）	32
実数値の各サブルーチンの引数またはオート変数	4
倍精度実数値の各サブルーチンの引数またはオート変数	8
変換の各サブルーチンの引数またはオート変数	48
プレジジョンポイントの各サブルーチンの引数またはオート変数	ロボットの各ジョイントに対して 4 バイト必要です。ロボットシステムが複数ある場合、最もジョイント数の多いロボットを使用します。
各ベルト変数の引数またはオート変数	84
各文字列変数の引数またはオート変数	132

注: サブルーチンの引数、またはオート変数が配列の場合、表に示すサイズに配列のサイズを乗算する必要があります（配列のインデックスは 0 から開始）。

サブルーチンの引数が常に参照渡しの場合、その引数の値を省略できます。

7.6 V+ プログラム 間の情報交換

V+ プログラム 間で情報を交換するには、次に示す 3 つの方法があります

1. グローバル変数
2. ソフトウェア信号
3. プログラム引数リスト

グローバル変数を使用した情報の交換

グローバル変数を使用して V+ プログラム 間で情報を交換する場合、単純に異なるプログラムで同じ変数名を使用します。

この方法はプログラムの実行によって想定外の結果を招いたり、デバッグが難しくなる可能性があるため注意が必要です。また、メインプログラムとサブルーチンで使用する変数名を必ず同じにする必要があるため、汎用性のあるサブルーチンの作成は難しくなります。

ソフトウェア信号を使用した情報の交換

ソフトウェア信号は、すべてのタスクとプログラムが読み取り、設定できる状態のデジタルソフトウェアスイッチです。詳細については、「ソフトウェア信号 29 ページを参照してください。

プログラム引数リストを使用した情報の交換

プログラム引数を使用して情報を交換すれば、変数の変化をより管理しやすくなります。また、呼び出しプログラムとサブルーチンで使用している変数名を同じにする必要はありません。

以下の項では、プログラム引数リストでデータを交換する方法について示します。

引数の受け渡し

呼び出しプログラムからサブルーチンに引数リストを渡すとき、2 つの重要事項を考慮する必要があります。

1. サブルーチンが期待する引数の受け取り方法（マッピング）で、呼び出しプログラムが確実に引数を渡すこと。
2. サブルーチンが変数を変更できる方法（値渡しや参照渡し）を決定すること。

引数リストのマッピング

引数リストは、コンマで区切られた変数または値のリストです。呼び出しプログラムに渡される引数リストとサブルーチンの引数リストで、引数の数、各引数のデータタイプが一致している必要があります。一致していない場合にエラーが発生します。

追加情報: 呼び出しプログラムが、引数リストを空白にしたり (arg_1, , arg_3 など)、または引数リストの最後で引数を省略 (arg_1, arg_2 など) した場合、引数は未定義のまま渡されます。引数リストを受け取ったサブルーチンは、DEFINED 関数キーワードを使用してこの値を検証し、適切なアクションを実行できます。

変数名を一致させる必要はありません。呼び出しプログラムがサブルーチンに引数を渡すとき、サブルーチンはリストの変数名ではなく、リストにある引数の位置のみを評価します。CALL ステートメント内の引数リストは、サブルーチンの引数リストにマッピングされます。このマッピング機能によって、呼び出しプログラムが使用する実際の値や変数名に関わらず、複数の異なるプログラムが呼び出すことができる汎用のサブルーチンを記述することができます。

次の図に、サブルーチンの引数リストに CALL ステートメントの引数リストをマッピングする様子を示します。矢印は、リスト内の各項目の位置とデータタイプを一致させる必要があることを

示しています。名前は必ずしも一致させる必要はありません。CALL ステートメントの引数リストには、値、式、変数名を使用できます。

この図に示すように、メインプログラムが CALL ステートメントに達すると、サブルーチン「a_routine」が呼び出され、引数リストが渡されます。

呼び出しプログラムのステートメント:

```
CALL a_routine(loc_var_a, real_var_a, 43.654, $string_var_a)
```

サブルーチンの1行目:

```
.PROGRAM a_routine(any_loc, any_real_x, any_real_y, $any_string)
```

呼び出しプログラムのステートメント:

```
CALL a_routine(loc_var_a:NULL, (real_var_a), real_var_b, $string_var_a+"")
```

サブルーチンの1行目:

```
.PROGRAM a_routine(any_loc, any_real_x, any_real_y, $any_string)
```

追加情報: 配列の受け渡しの詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』のCALL キーワードを参照してください。

引数の値渡しと参照渡し

サブルーチンで変数を変更し、変更した値を呼び出しプログラムに戻すことができます。値で引数を渡した場合、値のコピーが呼び出されたサブルーチンに渡されますが、サブルーチンで変更された値は呼び出しプログラムに戻されません。これは次に示す方法で、強制的に行うことができます。これらの方法は、サブルーチンが複数回呼び出されても、サブルーチンの修飾子は1つのインスタンスでメインプログラムにのみ渡される場合に有効です。

呼び出しプログラムがサブルーチンに変数を渡しても、サブルーチンが変数を変更した状態で戻すことができない場合、変数は、値渡しとみなされます。サブルーチンで変更が必要な変数は、参照渡しにする必要があります。

実変数「real_1」の値を強制的に呼び出しルーチン「program」に渡さない場合、次のステートメントのように、変数を丸かっこで囲む必要があります。

```
CALL program((real_1))
```

文字列変数「string_1」の値を強制的に呼び出しルーチン「program」に戻さないようにする場合、次のステートメントのように、変数に空の文字列を追加する必要があります。

```
CALL program($string_1+"")
```

変換変数「location_1」の値を強制的に呼び出しルーチン「program」に戻さないようにする場合、次のステートメントのように、null で変換を複合する必要があります。

```
CALL program(location_1:NULL)
```

呼び出しプログラムがサブルーチンに変数を渡し、サブルーチンが変数を変更し、変更した変数を呼び出しプログラムに戻される場合、変数は参照渡しとみなされます。

サブルーチンで変数を変更すると、呼び出しプログラムの変数の状態に反映されます。サブルーチンで変更され、呼び出しプログラムに渡される引数は、（式や値ではなく）変数として指定する必要があります。

サブルーチンを呼び出すとき、一部の引数を値渡しとし、それ以外は参照渡しとすることが出来ます。値渡しの場合、実際の変数ではなく変数のコピーがサブルーチンに渡されます。サブルーチンは、変数を変更できますが、その変更は呼び出しプログラムに返されません（呼び出しプログラムの変数は、サブルーチンが呼び出されたときの値と同じ）。

次の図に、さまざまな種類の変数を値渡しする方法を示します。実数と整数は、丸かっこで囲みます。ロケーション変数には NULL、文字列変数には「+」を付加します。

この図では、「real_var_b」は参照渡しされ、サブルーチンでの変更が呼び出しプログラムに反映されます。サブルーチンは、その他の変数を変更できません。サブルーチンに渡されたこれらの変数のコピーに対してのみ変更できます。呼び出しルーチンに戻す場合以外に、サブルーチンで引数を変更することは、プログラミング的に好ましくありません。入力引数を変更する必要がある場合、引数を自動的にコピーし、そのコピーを操作することを推奨します。

呼び出しプログラムのステートメント:

```
CALL a_routing(loc_var_a, real_var_a, 43.654, $string_var_a)
```

サブルーチンの1行目:

```
.PROGRAM a_routine(any_loc, any_real_x, any_real_y, $any_string)
```

呼び出しプログラムのステートメント:

```
CALL a_routing(loc_var_a:NULL, (real_var_a), real_var_b, $string_var_a+"")
```

サブルーチンの1行目:

```
.PROGRAM a_routine(any_loc, any_real_x, any_real_y, $any_string)
```

CALL ステートメントで変数だけでなく値も渡すことができます。次の例のように、「a_routing」を呼び出すことができます。

```
CALL a_routing(loc_1, 17.5, 121, "some string")
```

7.7 再入可能プログラムと再帰的プログラム

本項では、再入可能プログラムと再帰的プログラムについて説明します。

再入可能プログラム

eV+ システムでは、同時に複数のプログラムタスクが同じプログラムを実行できます。プログラムの実行中に、そのプログラムに再入することができます。これによって、同時に実行される異なるタスクが同じ汎用サブルーチンを使用できるようになります。

プログラムを再入させるためには、プログラムの作成時に次のガイドラインに従う必要があります。

- ・ グローバル変数を読み込むことができますが、変更はできません。
- ・ ローカル変数は使用しないでください。
- ・ オート変数とサブルーチンの引数のみ変更できます。

特殊な状況の場合に、ローカル変数を使用したり、グローバル変数を変更することもできますが、プログラムでこれらの変数へのアクセスをインターロックするプログラムロジックを明示する必要があります。このような状況では TAS 実数値関数が有用です。

再帰的プログラム

再帰的プログラムは、直接または間接的に自分自身を呼び出すサブルーチンです。プログラムが自分自身を呼び出す場合、直接呼出しになります。

間接呼び出しがより一般的で、プログラム A がプログラム B を呼び出す場合、つづいてプログラム B が戻る前にプログラム A を呼び出すという展開になります。例えば、出力ルーチンがエラーを検出し、エラー処理ルーチンを呼び出します。エラー処理ルーチンは、元出力ルーチンを呼び出し、エラーを報告します。再帰的サブルーチン呼び出しを使用する場合、プログラムは再入可能プログラムと同じガイドラインに従う必要があります。詳細については、「再入可能プログラム 87 ページを参照してください。

重要:再帰的呼び出しが無限に継続しないことを確認してください。継続した場合、プログラムのタスクはスタックスペースを使い果たします。

7.8 非同期処理

eV+ システムの重要な特徴は、プログラムの制御構造とは独立して発生する外部信号やエラー条件などのイベントに対応できることです。この種類のプログラム実行は、通常のプログラムの流れとは同期していないため、非同期とみなされます。eV+ は、イベント処理が適切に有効になっている場合のみ、CALL ステートメントが実行されたかのように指定のプログラムを呼び出し、イベントに対応します。

非同期処理を有効にするには、REACT、REACTE、REACTI キーワードを使用します。各プログラムタスクでこれらのキーワードを使用して、イベントの独立した処理を準備できます。さらに、WINDOW キーワードでは、ロボットがコンベアベルトを追跡しているとき、ベルトウィンドウの違反を非同期で処理ができます。

重要なプログラムセクションが完了するまで、応答を待機させる必要がある場合があります。一部のイベントは、他のイベントよりも重要性が高いため、プログラムでイベントに優先度を付ける必要があります。eV+ では、プログラムの優先度の値を 1~127 に設定して、応答の相対的重要性を指定することができます。優先度の値が高いほど、応答の重要性も高くなります。

応答サブルーチンは、メインプログラムの優先度よりも高い場合のみ呼び出されます。メインプログラムの優先度が応答サブルーチン以上の場合、メインプログラムの優先度が下がるまで応答サブルーチンの実行は保留されます。通常、メインプログラム（ロボット制御プログラムなど）の優先度は 0、応答プログラムの優先度は最低でも 1 で実行するため、どの応答でもメインプログラムに割り込むことができます。

LOCK プログラム命令キーワードでメインプログラムの優先度を上げ下げしたり、PRIORITY 実数値関数で現在の優先度の値を確認することができます。メインプログラムの優先度を一定の値まで上げた場合、その値以下のすべての応答はロックアウトされます。

応答サブルーチンが呼び出されると、メインプログラムの優先度は自動的にその応答プログラムの優先度に設定されるため、その優先度以下の応答は割り込むことができなくなります。応答プログラムで RETURN ステートメントを実行すると、メインプログラムの優先順位は、応答サブルーチンが呼び出される前のレベルに自動的にリセットされます。

追加情報: 応答とプログラムの優先度の詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』の LOCK、PRIORITY、REACT、REACTI を参照してください。

エラートラップ

通常、プログラムの実行中にエラーが発生した場合、プログラムは終了し、モニタウィンドウにエラーメッセージが表示されます。しかし、REACTE キーワードでエラートラッププログラムを有効にした場合、eV+ システムは、エラーになったプログラムを終了させる代わりに、サブルーチンとしてそのプログラムを呼び出します。各プログラムタスクは、独自のエラートラップを有効にできます。

エラートラップサブルーチンを呼び出す前に、eV+ はメインプログラムの優先度を 254 に上げて、その他のすべての応答をロックアウトします。

追加情報: エラートラップの詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』の REACTE キーワードの説明を参照してください。

第8章: T20 ペンダントのプログラミング

本項では、オプションのT20 ペンダントのカスタマイズ機能について説明します。ペンダントのインストールと操作については、『ティーチングペンダント T20 ユーザーズマニュアル (SBCE-392)』を参照してください。

8.1 T20 ペンダントのアタッチとデタッチ

V+ プログラム と通信する前に、ATTACH キーワードでペンダントをアタッチする必要があります。関連する V+ プログラム が終了したときに、ペンダントをデタッチしてください。

注: 次の例では、ペンダントの論理ユニット番号 (LUN) は 1 です。

すべての I/O デバイスと同様に、各 I/O 操作の後に IOSTAT キーワードでエラーの有無を確認してください。

V+ プログラム でペンダントがアタッチされている場合、マニュアルモードを使うことなくペンダントと情報を交換できます。

次の例では、ATTACH キーワードを使用してペンダントをアタッチします。

```
t20_lun = 1
ATTACH (t20_lun)
```

以下の例では、DETACH キーワードを使用してペンダントをデタッチします。

```
DETACH (t20_lun)
```

8.2 T20 ペンダントディスプレイへの書き込み

ペンダントでは、タイトルとメッセージ本文を表示したり、F1～F4 のファンクションキーのラベルを変更できます。どのフィールドにも空文字 (“”) を使用できます。メッセージ本文に HTML タグコードを使用できます。

次に、ペンダントディスプレイにメッセージを書き込む方法とクリアする方法を例を上げて説明します。

次の例では、PDNT.WRITEキーワードを使用して、図に示すように、タイトル「Operator Control」、メッセージ「Select Option from buttons below」を表示し、F1とF2ボタンのラベルを作成します。

```
$p.title = "Operator Control"
$p.msg[0] = "Select Option from buttons below"
$p.f[1] = "Apps"
$p.f[2] = "Status"
$p.f[3] = ""
$p.f[4] = ""
PDNT.WRITE $p.title, $p.msg[], $p.f[1], $p.f[2], $p.f[3], $p.f[4]
```

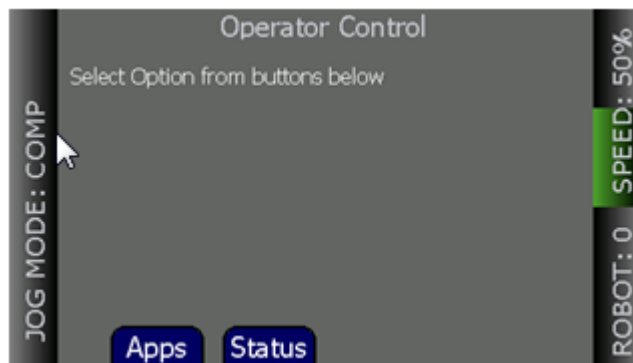


図8-1. PDNT.WRITE を使用したペンダントへの書き込み

次の例では、PDNT.NOTIFY キーワードを使用して、図に示すように"Waiting on Parts"と "Please add parts to Feeder"を表示します。

```
PDNT.NOTIFY "Waiting on Parts", "Please add parts to Feeder"
```



図8-2. PDNT.NOTIFY を使用したペンダントへの書き込み

追加情報: PDNT.CLEAR キーワードは、ペンダントの表示をクリアし、Home 1 画面に戻ります。

8.3 ユーザー入力の検出

検出可能なキーを一つ押すこと（シングルキープレス）で、ペンダントからの入力を受信できます。本項で示すように、シングルキープレスは3つの異なるモードでモニタすることが可能です。

1. キーボードモード: 通常のキーボードのキーのように、キーをモニタできます。
2. トグルモード: トグルモード（ON または OFF）でキーをモニタできます。キーを押すたびにキーの状態は ON/OFF と変化します。
3. レベルモード: レベルモードでキーをモニタできます。キーを押し続けたときだけ、キーの状態は ON とみなされます。

ペンダントキー押下の検出

個々のペンダントキーが押されたかどうかは、PENDANT プログラム命令キーワードで検出されます。このキーワードは、最初に押されたキーの番号を返します。

ペンダントには3種類のキー押下モードがあります。KEYMODE キーワードでどのモードを使用するかを設定します。使用可能なキー押下モードを次に示します。

1. キーボードモード
2. トグルモード
3. レベルモード

次の図は T20 ペンダントのキー番号を表しています。



図8-3. T20 ペンダントキーのマップ

注: カスタムウィンドウ (PDNT.WRITE) の場合でも、T20 ペンダントにあるすべての緑色のキーと Select Robot キーは正常に機能します。それ以外のすべてのキーの主な機能は無効になります。これらのキーは PENDANT プログラム命令キーワードを使ってV+ プログラムから読み取ります。

8.4 その他のペンダントプログラミング機能

次に示すその他のペンダント機能を PENDANT プログラム命令と一緒に使用できます。詳細については、『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』を参照してください。

- キーの状態をモニタする
- ペンダントの状態をモニタする
- キーボードモードを設定する
- 速度信号設定をモニタする
- 現在表示されている画面の番号をモニタする
- ペンダントのバージョン番号をモニタする

A.1 警告メッセージ、情報メッセージ、エラーメッセージ

この項では、システムの動作中に発生する可能性のあるシステムのエラーメッセージ、警告メッセージおよび情報メッセージについて説明します。

追加情報: メッセージの管理に関する詳細については、エラートラップ 89 ページおよび『eV+ 3 キーワードリファレンスマニュアル (SBCE-487)』の REACTE キーワードを参照してください。

システムの警告メッセージ

システムの警告メッセージの詳細については以下を参照してください。

表9-1. eV+システムの警告メッセージ

メッセージ No.	エラー定義名	エラー No.	内容
1	IE_WTMV	1010	Waiting for motion to complete
2	IE_WRNPOW	202	Power is now ON
3	IE_WRNPOF	203	Power is now OFF
4	IE_WRNCLS	101	Network connection closed
5	IE_WRNOPN	100	Network connection opened
6	IE_WRNHTR	58	Press and hold pendant enable switch
7	IE_WRNVFP	57	Press HIGH POWER button to enable power
8	IE_WNOHID	52	*Warning* Protected and read-only programs are not stored
9	IE_WNOCAL	51	*Warning* Not calibrated
10	IE_WRNDRY	50	Executing in DRY.RUN mode

システムの情報メッセージ

システムの情報メッセージの詳細については以下を参照してください。

表9-2. eV+システムの情報メッセージ

メッセージ No.	エラー定義名	エラー No.	内容
11	IE_SSDONE	13	Flag single-step done

メッセージ No.	エラー定義名	エラー No.	内容
12	IE_BPTMES	17	Breakpoint hit
13	IE_PAUSEM	9	PAUSE instruction hit
14	IE_FHALTM	8	HALT instruction hit
15	IE_DOHALT	6	DO done
16	IE_DOSTAR	5	Starting a DO command
17	IE_UHALT	4	Program task stopped
18	IE_FINI	3	Program completed
19	IE_USTART	2	Starting program execution
20	IE_SUCCESS	1	Success
21	IE_WARN	0	Warning

システムのエラーメッセージ

システムのエラーメッセージの詳細については以下を参照してください。

表9-3. eV+システムのエラーメッセージ

メッセージ No.	エラー定義名	エラー No.	内容
22	IE_NOTYET	-1	Not implemented
23	IE_OBSOL	-2	Obsolete Instruction
24	IE_ILLCMD	-300	Illegal monitor command
1	IE_NOEXIST	-3	Instruction does not exist
25	IE_NOPROG	-301	No program specified
26	IE_NODO	-302	DO not primed
27	IE_NOTAUTO	-303	Controller not in automatic mode
28	IE_BADFRE	-305	Storage area format error
29	IE_PRGNEX	-307	Program not executable
30	IE_PRGILK	-308	Program interlocked
31	IE_PRGDEF	-309	Program already exists
32	IE_HIDERR	-310	Can't access protected or read-only program

メッセージ No.	エラー定義名	エラー No.	内容
33	IE_PRGACT	-311	Invalid when program task active
34	IE_CANTPR	-312	Can't start while program running
35	IE_CNTPRO	-313	Can't go on, use EXECUTE or PRIME
36	IE_NOTENB	-314	Switch can't be enabled
37	IE_BADCFG	-315	Invalid software configuration
38	IE_BADCHK	-316	Software checksum error
39	IE_PRNACT	-318	Program task not active
40	IE_PRNUSE	-319	Program task not in use
41	IE_NODPRG	-350	Can't delete .PROGRAM statement
42	IE_PRG1ST	-351	First statement must be .PROGRAM
43	IE_READER	-352	Invalid in read-only mode
44	IE_ONLYCU	-353	No other program referenced
45	IE_TOLONG	-354	Line too long
46	IE_NOEXIT	-355	Can't exit while lines attached
47	IE_NOFIND	-356	Not found
48	IE_DMMES	-357	Recursive macros illegal
49	IE_SCANCL	-358	Cancelled
50	IE_DEBUGE	-359	Illegal in debug monitor mode
51	IE_NOTDBG	-360	Must be in debug mode
52	IE_CNTSWC	-361	Can't change modes while task running
53	IE_NWPRGI	-362	Can't execute from SEE program instruction
54	IE_UNKECM	-363	Unknown editor command
55	IE_NOCCOM	-364	Can't create program in read-only mode
56	IE_WRIOER	-365	Illegal in read-write mode
57	IE_PONSTK	-366	Invalid when program on stack
58	IE_BPTNAL	-380	Breakpoint not allowed here

メッセージ No.	エラー定義名	エラー No.	内容
59	IE_ABORT	-400	Abort
60	IE_UNDVAR	-401	Undefined value
61	IE_BADNUM	-402	Illegal value
62	IE_BADASN	-403	Illegal assignment
63	IE_BADARR	-404	Illegal array index
64	IE_BADSIG	-405	Illegal digital signal
65	IE_UNDNAM	-406	Undefined program or variable name
66	IE_ILLARG	-407	Invalid argument
67	IE_SUBAMM	-408	Program argument mismatch
68	IE_ARITHO	-409	Arithmetic overflow
69	IE_NEGSQR	-410	Negative square root
70	IE_NOFRES	-411	Not enough storage area
71	IE_MISLBL	-412	*Branch to undefined label* Step
72	IE_SUBERR	-413	Not enough program stack space
73	IE_MCERR	-414	Can't mix MC & program instructions
74	IE_STGOVF	-416	String variable overflow
75	IE_STRSHT	-417	String too short
76	IE_NXMERR	-418	Illegal memory reference
77	IE_CMDACT	-419	Illegal when command program active
78	IE_UNDVCX	-420	Undefined value in this context
79	IE_PRGNOT	-421	Program not on top of stack
80	IE_FALENB	-422	Function already enabled
81	IE_ILLOPR	-423	Illegal operation
82	IE_NOCALP	-425	Calibration program not loaded
83	IE_NOCALF	-426	Can't find calibration program file
2	IE_SYSABORT	-429	Aborted by system
84	IE_BADLIN	-450	Can't interpret line

メッセージ No.	エラー定義名	エラー No.	内容
85	IE_SWPERR	-451	Unexpected text at end of line
86	IE_ILLINS	-452	Unknown instruction
87	IE_AMBIG	-453	Ambiguous name
88	IE_NOARGU	-454	Missing argument
89	IE_INVNME	-455	Invalid program or variable name
90	IE_BADNFM	-456	Invalid number format
91	IE_RESWRD	-457	Reserved word illegal
92	IE_SYNERR	-458	Illegal expression syntax
93	IE_MISPAR	-459	Missing parenthesis
94	IE_MISQUO	-460	Missing quote mark
95	IE_ILLFMT	-461	Invalid format specifier
96	IE_UNKFUN	-462	Unknown function
97	IE_BADLBL	-463	Invalid statement label
98	IE_DUPLBL	-464	Duplicate statement label
99	IE_BADTYP	-465	Variable type mismatch
100	IE_ILLBLV	-466	Illegal use of belt variable
101	IE_BADPRG	-467	Illegal .PROGRAM statement
102	IE_DUPARG	-468	Duplicate .PROGRAM arguments
103	IE_REDECL	-469	Attempt to redefine variable type
104	IE_RDFCLS	-470	Attempt to redefine variable class
105	IE_MISPCD	-471	Misplaced declaration statement
106	IE_CSFERR	-472	*Control structure error* Step
107	IE_BADCON	-473	Control structure error
108	IE_BADDIM	-474	Too many array indices
109	IE_MISBRK	-475	Missing bracket
110	IE_ILLSQL	-476	Invalid qualifier
111	IE_AMBAUT	-477	Ambiguous AUTO invalid

メッセージ No.	エラー定義名	エラー No.	内容
112	IE_FSTFPE	-500	File already exists
113	IE_NOFILE	-501	Nonexistent file
114	IE_BADIOC	-502	Illegal I/O device command
115	IE_CEFULL	-503	Device full
116	IE_IEEOF	-504	Unexpected end of file
117	IE_IEINI	-505	Initialization error
118	IE_IEFAO	-506	File already opened
119	IE_DEVNTR	-508	Device not ready
120	IE_IECHK	-510	Data checksum error
121	IE_BADFIL	-512	File format error
122	IE_IEFNO	-513	File not opened
123	IE_SUBNAME	-514	File or subdirectory name error
124	IE_LUNBSY	-515	Already attached to logical unit
125	IE_NOTATT	-516	Not attached to logical unit
126	IE_ILLCHAN	-518	Illegal I/O channel number
127	IE_ICE	-519	Driver internal consistency error
128	IE_IEDAT	-522	Data error on device
129	IE_IEDAO	-524	Communications overrun
130	IE_ILGRED	-525	Illegal I/O redirection specified
131	IE_NODATA	-526	No data received
132	IE_BADLUN	-527	Illegal user LUN specified
133	IE_BADRLN	-528	Illegal record length
134	IE_IOOVER	-529	Output record too long
135	IE_IVHCFG	-533	Invalid hardware configuration
136	IE_IEPRO	-530	Protection error
137	IE_IETMO	-531	Communication time-out
138	IE_IENER	-536	Too many network errors

メッセージ No.	エラー定義名	エラー No.	内容
139	IE_IEUNN	-537	Unknown network node
140	IE_IEILC	-540	Invalid connection specified
141	IE_IEINP	-541	Invalid network protocol
142	IE_NETINE	-543	Illegal when network enabled
143	IE_NOTCFA	-544	Not configured as accessed
144	IE_NOSUBDIR	-545	Nonexistent subdirectory
145	IE_SUBINUSE	-547	Subdirectory in use
146	IE_IETMW	-550	Too many windows
147	IE_TOOMNY	-553	Too many arguments
148	IE_IENoor	-559	Out of network resources
149	IE_ERNTIM	-562	Network timeout
150	IE_IENFTL	-569	File too large
151	IE_DNNOTCFG	-586	DeviceNet not configured
152	IE_HDWSTP	-600	Stopped due to servoing error
153	IE_NOARM	-601	Robot not attached to this program
154	IE_ARMALA	-602	Robot already attached to program
155	IE_NOTCMP	-603	COMP mode disabled
156	IE_PWISOF	-604	Robot power off
157	IE_NOTCAL	-605	Robot not calibrated
158	IE_AIRERR	-607	No air pressure
159	IE_PANBUT	-608	External E-STOP
160	IE_BADJTN	-609	Illegal joint number
161	IE_OUTRNG	-610	Location out of range
162	IE_CNTJI	-611	Must use straight-line motion
163	IE_CANCFG	-612	Straight-line motion can't alter configuration
164	IE_CNTMVE	-613	Illegal motion from here

メッセージ No.	エラー定義名	エラー No.	内容
165	IE_BLTER	-614	Attempt to modify active belt
166	IE_NOBELT	-615	Belt not enabled
167	IE_WINERR	-616	Belt window violation
168	IE_BLTDED	-617	Belt servo dead
169	IE_LTOCLS	-618	Location too close
170	IE_BADORI	-619	Invalid orientation
171	IE_MANBTT	-620	Speed pot or STEP not pressed
172	IE_ROBILK	-621	Robot interlocked
173	IE_NOROBT	-622	No robot connected to system
174	IE_FPBSOF	-623	Stop-on-force triggered
175	IE_FBPRT	-624	Force protect limit exceeded
176	IE_INVSDT	-625	Invalid servo initialization data
177	IE_CTBLAL	-626	Can't ALTER and track belt
178	IE_PONERR	-627	Robot power on
179	IE_MODERR	-628	*Robot module not loaded* ID:
180	IE_SYFERR	-629	SYSFAIL detected by CPU
181	IE_ESIERR	-630	Backplane E-STOP detected by CPU
182	IE_CTLHOT	-631	Controller overheating
183	IE_ROBPWF	-632	Power failure detected by robot
184	IE_PANCMD	-633	PANIC command
185	IE_FRCCER	-634	Force sensor communication error
186	IE_NOCTCL	-635	Cartesian control of robot not possible
187	IE_HSTOVRE	-636	overrun detected by host system
188	IE_SPNERR	-637	Illegal while joints SPIN'ing
189	IE_CNTSPN	-638	SPIN motion not permitted
190	IE_MNBRAK	-639	Manual brake release
191	IE_ESTROB	-640	E-STOP from robot

メッセージ No.	エラー定義名	エラー No.	内容
192	IE_ESTAMP	-641	E-STOP from amplifier
193	IE_ESTSYF	-642	SYSFAIL detected by robot
194	IE_ESTBAC	-643	E-STOP detected by robot
195	IE_MOVING	-644	Illegal while robot is moving
196	IE_PWDSBL	-645	Power disabled: Manual/Auto changed
197	IE_PWENBL	-646	HIGH POWER button not pressed
198	IE_EC3DED	-649	Timeout: MCP enable switch not toggled
199	IE_TPFAIL	-650	Manual control pendant failure
200	IE_RSCDWN	-651	RSC communications failure
201	IE_RSCRST	-652	RSC reset
202	IE_RSCTMO	-653	RSC time-out
203	IE_RSCTER	-654	RSC transmission garbled
204	IE_RSCBAD	-655	RSC bad packet format
205	IE_RSCNCL	-656	RSC calibration load failure
206	IE_PENCON	-657	Pendant not connected
207	IE_HDWNPR	-658	Device hardware not present
208	IE_IEDERR	-660	Device error
209	IE_EEPCHK	-661	NVRAM data invalid
210	IE_DEVSER	-662	Device sensor error
211	IE_IEDRES	-663	Device reset
212	IE_INVCAL	-664	Invalid calibration data
213	IE_EEPBAT	-665	NVRAM battery failure
214	IE_NOTSYS	-666	Must use CPU #1
215	IE_PWFERR	-667	Power failure detected
216	IE_DEVINU	-668	Device in use
217	IE_ERSCHW	-669	RSC hardware failure
218	IE_ERSCPW	-670	RSC power failure

メッセージ No.	エラー定義名	エラー No.	内容
219	IE_ESR12F	-671	Servo board 12V fuse open
220	IE_ESRSLF	-672	Servo board solenoid fuse open
221	IE_ESRESF	-673	Servo board E-STOP fuse open
222	IE_SEROVR	-674	Servo task overloaded
223	IE_MOTPWF	-675	Timeout enabling power
224	IE_MIDWRG	-676	RSC module ID doesn't match robot
225	IE_IESVIN	-677	Servo protocol incompatible
226	IE_IEDUPS	-678	Duplicate servo node ID
227	IE_IENFSV	-679	Expected servo node not found
228	IE_IESNDN	-680	Servo node not downloaded
229	IE_VINOOB	-704	No objects seen
230	IE_VINRUN	-705	Camera not running
231	IE_VIRUN	-706	Invalid request while camera running
232	IE_VIPROT	-707	Unknown prototype
233	IE_VINCAL	-713	Vision not calibrated
234	IE_VIONAL	-714	Camera already running
235	IE_VIOFAL	-719	Camera already off
236	IE_VININS	-720	Vision option not installed
237	IE_VINOCI	-722	Camera interface board absent
238	IE_VINPPD	-723	No picture data available
239	IE_VIILDM	-724	Illegal display mode
240	IE_VIILCL	-726	Bad camera calibration
241	IE_IEVTRN	-729	Invalid request while vision training
242	IE_IENINF	-730	Information not available
243	IE_IESUBP	-731	Unknown sub-prototype
244	IE_VIIVNM	-732	Invalid model name
245	IE_VINOMM	-733	Vision system out of memory

メッセージ No.	エラー定義名	エラー No.	内容
246	IE_ILVARG	-735	Invalid vision argument
247	IE_VABORT	-749	Vision aborted
248	IE_NVSEL	-751	No vision system selected
249	IE_VINTER	-767	Vision internal error
250	IE_NOVISOPT	-770	The vision option is not licensed
251	IE_ILLASS	-771	Invalid Robot Vision Manager sequence
252	IE_ILLAST	-772	Invalid Robot Vision Manager tool index
253	IE_ILLASP	-773	Invalid Robot Vision Manager parameter ID
254	IE_ILLASP2	-774	Invalid Robot Vision Manager parameter index
255	IE_NOASSEQ	-775	Robot Vision Manager instance not found
256	IE_NOCAMIM	-776	Unable to acquire an image from the camera
257	IE_NOROBLAT	-777	Unable to read the robot position latch
258	IE_NOBELLAT	-778	No belt latch was detected
259	IE_UNKNER	-800	Unknown error code
260	IE_BDVFEA	-801	Invalid VFEATURE access
261	IE_NCMARM	-802	Invalid camera calibration
262	IE_ILLCAM	-803	Illegal camera number
263	IE_NOOPT	-804	Option not installed
264	IE_NOHDWR	-805	Hardware not in system
265	IE_DBERR	-859	Database manager internal error
266	IE_COLDET	-901	Obstacle collision detected
267	IE_MSECME	-902	Interrupted multi-segment motion
268	IE_ESTUCK	-904	[Fatal] E-STOP signals are stuck off
269	IE_NOHPWR	-906	Robot power off requested
270	IE_ESTPWR	-907	E-STOP circuit relay failure

メッセージ No.	エラー定義名	エラー No.	内容
271	IE_ESTFP	-908	E-STOP from front panel button
272	IE_ESTMCP	-909	E-STOP from MCP E-STOP button
273	IE_CP44P3	-910	E-STOP from user E-STOP button
274	IE_CP44P5	-912	E-STOP from user enable switch
275	IE_ESTCPU	-919	E-STOP asserted by CPU
276	IE_AMSTUK	-920	[Fatal] Auto mode switch stuck on
277	IE_CP44P7	-921	E-STOP from user muted safety gate
278	IE_ESTINC	-922	E-STOP channels 1 and 2 do not match
279	IE_ESTSHR	-923	E-STOP circuit is shorted
280	IE_BADLIT	-924	Front panel HIGH POWER lamp failure
281	IE_CP48P7	-929	E-STOP from Line E-STOP input
282	IE_PWROVLD	-933	Power output overloaded on Belt/EXPIO
283	IE_HPSTUK	-934	[Fatal] HIGH POWER button stuck on
284	IE_NOSOL1	-935	Orientation out of range
285	IE_NOSOL2	-936	Kinematic solution not found
286	IE_BADKEY	-937	Robot already under manual control
287	IE_NOJNT	-938	Joint control of robot not possible
288	IE_ESTOPUNS TABLE	-939	E-STOP unstable
289	IE_ABORTX	-999	Aborted
290	IE_UNKSVE	-1001	Invalid servo error
291	IE_NOSOL	-1002	Position out of range
292	IE_NOTIME	-1003	Time-out nulling errors
293	IE_NOREF	-1004	No zero index
294	IE_NOSYNC	-1005	Unexpected zero index
295	IE_EVLERR	-1006	Soft envelope error
296	IE_MSTALL	-1007	Motor stalled
297	IE_QADERR	-1008	Encoder quadrature error

メッセージ No.	エラー定義名	エラー No.	内容
298	IE_AMPERR	-1009	Timeout enabling amplifier
299	IE_MOTHOT	-1016	Motor overheating
300	IE_AMPERR1	-1018	Motor amplifier fault
301	IE_DUTCYC	-1021	Duty-cycle exceeded
302	IE_SKWERR	-1022	Skew envelope error
303	IE_NOSOLM	-1023	Position out of range, motor
304	IE_ENCFLT	-1025	Encoder fault
305	IE_SWINC	-1026	Software incompatible
306	IE_HEVLER	-1027	Hard envelope error
307	IE_SSPDER	-1028	Soft overspeed error
308	IE_HSPDER	-1029	Hard overspeed error
309	IE_NGOVER	-1032	Negative overtravel
310	IE_PSOVER	-1033	Positive overtravel
311	IE_RBOVER	-1034	Overtravel
312	IE_SVDEAD	-1104	[Fatal] Servo dead* Mtr
313	IE_NOCTRK	-1106	Calibration sensor failure* Mtr
314	IE_SAFETYSYS TEMFAULT	-1109	*Safety System Fault*
315	IE_EC3EEP	-1111	E-STOP from safety system* Code
316	IE_POWERSYS TEMFAILURE	-1115	*Power System Failure*
317	ENDMARKER	-3000	Unknown error code

●本誌に記載のない条件や環境での使用、および原子力制御・鉄道・航空・車両・燃焼装置・医療機器・娯楽機械・安全機器、その他人命や財産に大きな影響が予測されるなど、特に安全性が要求される用途に使用される際には、当社の意図した特別な商品用途の場合や特別の合意がある場合を除き、当社は当社商品に対して一切保証をいたしません。
●本製品の内、外国為替及び外国貿易法に定める輸出許可、承認対象貨物（又は技術）に該当するものを輸出（又は非居住者に提供）する場合は同法に基づく輸出許可、承認（又は役務取引許可）が必要です。

オムロン株式会社 インダストリアルオートメーションビジネスカンパニー

●製品に関するお問い合わせ先

お客様相談室

フリー 01 20-91 9-066

携帯電話・PHS・IP電話などではご利用いただけませんので、下記の電話番号へおかけください。

電話 055-982-5015（通話料がかかります）

■営業時間：8:00～21:00 ■営業日：365日

●FAXやWebページでもお問い合わせいただけます。

FAX 055-982-5051 / www.fa.omron.co.jp

●その他のお問い合わせ

納期・価格・サンプル・仕様書は貴社のお取引先、または貴社担当オムロン販売員にご相談ください。

オムロン制御機器販売店やオムロン販売拠点は、Webページでご案内しています。

オムロン制御機器の最新情報をご覧ください。

www.fa.omron.co.jp

緊急時のご購入にもご利用ください。

オムロン商品のご用命は