

SIMOTION

SIMOTION SCOUT 基本機能

機能マニュアル

はじめに

システムの概要

1

テクノロジーパッケージおよびテクノロジーオブジェクト

2

テクノロジーオブジェクトを使用したプログラミング

3

テクノロジーオブジェクトでのエラー処理

4

実行システム、タスク、およびシステムサイクルクロック

5

実行システム / タスク / システムサイクルクロックのプログラミング

6

一般的な標準ファンクションのプログラミング

7

一般的なシステムファンクションブロックのプログラミング

8

エラーの原因および効率的なプログラミング

9

付録A

A

安全性に関する基準

本書には、ユーザーの安全性を確保し製品の損傷を防止するうえ守るべき注意事項が記載されています。ユーザーの安全性に関する注意事項は、安全警告サインで強調表示されています。このサインは、物的損傷に関する注意事項には表示されません。

 危険
回避しなければ、直接的な死または重傷に至る危険状態を示します。

 警告
回避しなければ、死または重傷に至るおそれのある危険な状況を示します。

 注意
回避しなければ、軽度または中度の人身傷害を引き起こすおそれのある危険な状況を示します（安全警告サイン付き）。

注意
回避しなければ、物的損傷を引き起こすおそれのある危険な状況を示します（安全警告サインなし）。

通知
回避しなければ、望ましくない結果や状態が生じ得る状況を示します（安全警告サインなし）。

複数の危険レベルに相当する場合は、通常、最も危険度の高い（番号の低い）事項が表示されることになっています。安全警告サイン付きの人身傷害に関する注意事項があれば、物的損傷に関する警告が付加されます。

有資格者

装置/システムのセットアップおよび使用にあたっては必ず本マニュアルを参照してください。機器のインストールおよび操作は有資格者のみが行うものとします。有資格者とは、法的な安全規制/規格に準拠してアースの取り付け、電気回路、設備およびシステムの設定に携わることを承認されている技術者のことをいいます。

使用目的

以下の事項に注意してください。

 警告
本装置およびコンポーネントはカタログまたは技術的な解説に詳述されている用途にのみ使用するものとします。また、Siemens社の承認または推奨するメーカーの装置またはコンポーネントのみを使用してください。本製品は輸送、据付け、セットアップ、インストールを正しく行い、推奨のとおりにより操作および維持した場合にのみ、正確かつ安全に作動します。

商標

®マークのついた称号はすべてSiemens AGの商標です。本書に記載するその他の称号は商標であり、第三者が自己の目的において使用した場合、所有者の権利を侵害することになります。

免責事項

本書のハードウェアおよびソフトウェアに関する記述と、実際の製品内容との一致については検証済みです。しかしなお、本書の記述が実際の製品内容と異なる可能性もあり、完全な一致が保証されているわけではありません。記載内容については定期的に検証し、訂正が必要な場合は次の版で更新いたします。

はじめに

内容

本書は、『System and Function Descriptions documentation package』の一部です。

対象範囲の確認

本書は、SIMOTION SCOUT V4.1の以下のバージョンを対象としています。

- SIMOTION SCOUT V4.1 (SIMOTION製品シリーズのエンジニアリングシステム)、
- SIMOTION Kernel V4.1、V4.0、V3.2、V3.1、またはV3.0
- SIMOTIONテクノロジーパッケージのCam、Cam_ext (Kernel V3.2以降)および、それぞれのカーネル用バージョンのTControl (Kernel V3.0までの装置、位置および基本モーションコントロールのテクノロジーパッケージを含む)。

本書に掲載されている章

本書では、SIMOTIONとテクノロジーオブジェクトの一般的に使用される機能について説明します。

- システムの概要

SIMOTIONに関する一般情報です。

- テクノロジーパッケージおよびテクノロジーオブジェクト

テクノロジーパッケージおよびテクノロジーオブジェクトに関する基本情報です。

- テクノロジーオブジェクトのプログラミング

テクノロジーオブジェクトのプログラミングに使用するシステムファンクションと、プログラミングの方法に関する情報です。

- テクノロジーオブジェクトでのエラー処理

テクノロジーカルアラームとコマンド戻り値に関する一般情報です。

- 実行システム、タスク、およびシステムサイクルクロック

実行システムとタスクに関する一般情報です。

- 実行システム / タスク / システムサイクルクロックのプログラミング

実行システム、タスク、およびシステムサイクルクロックに関する情報です。

- 一般的なシステムファンクションのプログラミング

利用できる一般的なシステムファンクションとその使用方法に関する情報です。

- 一般的なシステムファンクションブロックのプログラミング

利用できる一般的なシステムファンクションブロックとその使用方法に関する情報です。

- **索引**

情報を検索するためのキーワードの索引

SIMOTIONドキュメンテーション

SIMOTIONドキュメンテーションの一覧は、別途、参考文献一覧として掲載されています。

このマニュアルは、提供されるSIMOTION SCOUTとともに電子マニュアルとして収録されます。

SIMOTION取扱説明書は9個のマニュアルパッケージで構成され、そのパッケージには約60のSIMOTIONマニュアルとその他の製品(たとえばSINAMICS)に関するマニュアルが含まれています。

SIMOTION V4.1では、以下のドキュメンテーションパッケージを使用できます。

- SIMOTIONエンジニアリングシステム
- SIMOTIONシステムおよび機能
- SIMOTION診断
- SIMOTIONプログラミング
- SIMOTIONプログラミング - リファレンス
- SIMOTION C2xx
- SIMOTION P350
- SIMOTION D4xx
- SIMOTION追加ドキュメンテーション

ホットラインおよびインターネットアドレス

技術上のご質問がある場合は、弊社のホットライン(世界中どこでも可能です)にお問い合わせください。

A&Dテクニカルサポート:

- 電話番号: +49 (180) 50 50 222
- FAX番号: +49 (180) 50 50 223
- 電子メール: adsupport@siemens.com
- インターネット: <http://www.siemens.de/automation/support-request>

ご質問やご提案がある場合や、ドキュメンテーションの間違いにお気づきの場合は、次の連絡先宛にファックスまたは電子メールでお知らせください。

- FAX番号: +49 (9131) 98 63315
- 電子メール: docu.motioncontrol@siemens.com

Siemensインターネットアドレス

SIMOTION製品、製品サポート、およびFAQに関する情報は、インターネットの次のアドレスに掲載されています。

- 一般情報:
 - <http://www.siemens.de/simotion>(ドイツ)
 - <http://www.siemens.com/simotion>(世界共通)
- 製品サポート:
 - <http://support.automation.siemens.com/WW/view/en/10805436>

その他のサポート

弊社は、SIMOTIONの習得のための入門コースも提供しています。

お客様の地域のトレーニングセンターか、D-90027 Nuremberg/Germany、Tel +49 (911) 895 3202の本部トレーニングセンターにお問い合わせください。

目次

	はじめに	3
1	システムの概要	15
1.1	SIMOTIONモーションコントロール	15
1.2	用途	16
1.3	PLCとモーションコントロールの統合	17
1.4	統合オートメーション(Totally Integrated Automation)	18
1.5	ハードウェアプラットフォーム	19
1.6	システムアーキテクチャ	21
1.6.1	SIMOTIONのシステムアーキテクチャ	21
1.6.2	SIMOTION SCOUTエンジニアリングシステム	22
1.6.3	SIMOTIONプロジェクト	23
1.6.4	オフライン / オンラインモード	25
1.6.5	プログラミング	25
2	テクノロジーパッケージおよびテクノロジーオブジェクト	27
2.1	概要	27
2.2	テクノロジーパッケージ	30
2.3	テクノロジーオブジェクト(TO)	31
2.3.1	インスタンス化および設定	31
2.3.2	プログラミング	32
2.3.3	プログラミング	33
2.3.4	相互接続	36
2.3.5	テクノロジーオブジェクトおよびDCC	37
2.3.6	使用できるテクノロジーオブジェクト	39
2.4	SIMOTIONのメモリコンセプト(対象システム)	41
2.4.1	対象システムでのストレージコンセプト	41
2.5	エキスパートリスト	43
2.5.1	コンフィグレーションデータおよびシステム変数のエキスパートリスト	43
2.5.2	エキスパートリストの使用	46
2.6	テクノロジーオブジェクトの相互接続(エキスパート用)	52
2.6.1	SCOUTの汎用相互接続画面フォームによる相互接続	52
2.6.2	相互接続インターフェースの一般的な特性	54
2.6.3	TO上の相互接続インターフェース	55
2.6.4	"Motion"タイプの相互接続インターフェース	58
2.6.5	"LREAL"タイプの相互接続インターフェース	60
3	テクノロジーオブジェクトを使用したプログラミング	63
3.1	定義	63
3.2	テクノロジーオブジェクト(TO)のプログラミング	64
3.2.1	テクノロジーファンクションのプログラムでの使用	64

3.2.2	周期的プログラミングと逐次的プログラミングの相違点	67
3.2.3	テクノロジーファンクションの入力パラメータ	68
3.2.4	移行およびステップ有効化条件	73
3.2.5	コマンド実行の診断	78
3.2.6	テクノロジーオブジェクトインスタンスの識別子	80
3.2.7	TOデータタイプの変換	81
3.2.8	システム変数	83
3.2.9	コンフィグレーションデータ	87
3.2.10	テクノロジーオブジェクトのリセット	90
3.2.11	テクノロジーオブジェクトのライブラリでの使用	91
3.3	障害およびイベントへの応答	93
3.3.1	障害およびイベントの評価	93
3.3.2	プログラムでの実行エラー	94
3.3.3	システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセ スエラー	95
3.3.4	プロセスイメージの生成時のエラー	96
3.3.5	Taskstartinfoの使用	97
4	テクノロジーオブジェクトでのエラー処理	109
4.1	テクノロジーオブジェクトで発生しうるエラー	109
4.2	プロセスアラーム	110
4.2.1	ローカル応答	110
4.2.2	グローバル応答	111
4.2.3	エラーの有効化	111
4.2.4	テクノロジーカルアラームの設定	113
4.2.5	テクノロジーカルアラームの表示とリセット	115
4.2.6	ユーザープログラムを介したリセット	116
4.2.7	ユーザープログラムでの評価	120
4.3	コマンドの戻り値	122
4.4	システムデータへのアクセス時のエラー	124
5	実行システム、タスク、およびシステムサイクルクロック	127
5.1	実行システム	127
5.1.1	実行レベル / タスク	128
5.1.2	SIMOTION SCOUTの実行システム	131
5.1.3	タスク優先度	133
5.1.4	SIMOTIONのランタイムモデル	136
5.2	ユーザープログラムタスクの概要	138
5.2.1	StartupTask	138
5.2.2	MotionTasks	140
5.2.3	BackgroundTask	144
5.2.4	TimerInterruptTasks	147
5.2.5	SynchronousTasks	151
5.2.6	SystemInterruptTasks	157
5.2.7	UserInterruptTasks	162
5.2.8	ShutdownTask	167
5.3	実行システムのコンフィグレーション	170
5.3.1	実行レベル / タスクへのプログラムの割り当て	170
5.3.2	サイクルクロックソースの選択	174
5.3.3	システムサイクルクロックの定義	175
5.3.4	タスクの実行時間	181

5.3.5	タイムアウトとレベルオーバーフロー	182
5.3.6	タスクの起動に関する情報: TaskStartInfo(TSI)	185
5.3.7	ウォッチドッグ	185
5.4	ラウンドロビン実行レベルでの時間割り当て	187
5.4.1	時間割り当ての設定	189
5.4.2	設定(例)	190
5.4.3	タスクの処理(例)	193
5.5	フィールドバスシステム上のアイソクロナスI/O処理	197
5.5.1	PROFIBUS DP上のデータプロトコル	197
5.5.2	PROFINET IO上のプロトコル	198
5.5.3	アイソクロナスデータ処理	199
5.5.4	コントロールのデータ処理の動的応答	201
5.5.5	データ伝送(取得から処理まで)に関する動的応答	203
5.5.6	データ取得およびデータ出力の動的応答	203
5.5.7	PROFIBUSのET 200 I/Oデバイス用HWコンフィグレーションによるTdp、Ti、およびToの識別	204
5.5.8	サイクルクロックスケーリングの処理	205
5.5.9	ランタイムを最適化するためのHW ConfigでのPROFIBUS DPのコンフィグレーション	206
5.6	SIMOTION実行システムへのDCCの統合	208
5.6.1	概要	208
5.6.2	DCCブロック(DCB)のシーケンスモデル	208
5.6.3	サーボレベルのservoDccTask	210
5.6.4	IPOレベルのipoDccタスク	211
5.6.5	IPO2レベルのipoDcc_2タスク	212
5.6.6	DccAuxおよびDccAux_2の実行レベル	213
5.6.7	ブロック間のデータ交換	214
5.6.7.1	ブロック間のデータ交換(概要)	214
5.6.7.2	同じレベルのブロック間のデータ交換	214
5.6.7.3	低優先度レベルからのブロック用データ	216
5.6.7.4	高優先度レベルからのブロック用データ	219
5.6.8	ブロックと変数の相互接続	220
5.6.8.1	変数との相互接続	220
5.6.8.2	FPU例外時の動作	222
5.7	ドライバI/Oのインクルード	223
5.7.1	増設I/OモジュールTM15およびTM17 High Feature	223
5.7.2	TM31、TM41、TM15 DI/DO増設I/Oモジュール、TB30増設I/Oカード、SIMOTION DまたはCU310/CU320およびCX32のオンボードI/O	223
6	実行システム / タスク / システムサイクルクロックのプログラミング	227
6.1	実行システム	227
6.1.1	実行システム概要	227
6.1.2	実行レベルとタスク	227
6.1.3	タスク起動シーケンス	229
6.1.4	実行システムのコンフィグレーション	229
6.1.4.1	コンフィグレーション時の設定	229
6.1.4.2	タスクへのプログラムの割り当て	230
6.1.5	変数初期化に対するタスク実行動作の影響	230
6.1.5.1	ローカルプログラム変数を初期化するタイミング	230
6.1.5.2	ユニット変数への初期値の割り当て	231
6.1.5.3	複数のVAR_GLOBAL、VAR_GLOBAL RETAINブロックの使用	232
6.1.5.4	変数初期化に対するコンパイラの影響	232
6.1.5.5	HMI関連データのマーキング	234
6.1.6	タスクのステータス	235

6.1.6.1	タスク状態の検索と意味.....	235
6.1.6.2	タスク状態の組み合わせ.....	236
6.1.6.3	タスク状態の使用例.....	237
6.1.7	MotionTaskで条件が満たされるのを待機する.....	237
6.1.7.1	EXPRESSIONの条件の構文.....	237
6.1.7.2	WAITFORCONDITIONステートメントの構文.....	238
6.1.7.3	WAITFORCONDITIONステートメントの効果.....	239
6.1.7.4	WAITFORCONDITIONの使用例.....	240
6.1.7.5	FB経由の時間監視の例.....	241
6.1.7.6	非周期的タスク / MotionTaskで直接に時間監視を行うWAITFORCONDITIONの使用例.....	242
6.1.8	タスクを指定した時間だけ待機させる.....	242
6.1.9	RUN時のダウンロードの手順.....	244
6.1.9.1	RUN時のダウンロード.....	244
6.1.9.2	RUN時のダウンロードのオプション.....	246
6.1.9.3	Whileループの例.....	251
6.1.9.4	Block_Init_on_Changeの構文の例.....	251
6.2	タスク制御コマンド.....	252
6.2.1	タスク制御コマンドの一覧.....	252
6.2.2	タスク制御コマンドの使用例.....	254
6.2.3	ファンクション_getStateOfTaskId.....	254
6.2.4	ファンクション_resetTaskId.....	256
6.2.5	ファンクション_restartTaskId.....	258
6.2.6	ファンクション_resumeTaskId.....	259
6.2.7	ファンクション_retriggerTaskIdControlTime.....	260
6.2.8	ファンクション_startTaskId.....	262
6.2.9	ファンクション_suspendTaskId.....	263
6.2.10	ファンクション_getTaskId.....	265
6.2.11	ファンクション_checkEqualTask.....	266
6.3	タスクのランタイム測定用ファンクション.....	267
6.3.1	タスクのランタイム測定用ファンクション - 概要.....	267
6.3.2	ファンクション_getMaximalTaskIdRunTime.....	268
6.3.3	ファンクション_getMinimalTaskIdRunTime.....	269
6.3.4	ファンクション_getCurrentTaskIdRunTime.....	271
6.3.5	ファンクション_getAverageTaskIdRunTime.....	272
6.3.6	タスクの正確なランタイム測定用のファンクション.....	274
6.3.7	ファンクション_getInternalTimeStamp.....	274
6.3.8	ファンクション_getTimeDifferenceOfInternalTimeStamp.....	275
6.4	メッセージプログラミング用ファンクション(AlarmS).....	276
6.4.1	メッセージプログラミングに関する全般情報.....	276
6.4.2	ファンクション_alarmSId.....	277
6.4.3	ファンクション_alarmSqlId.....	280
6.4.4	ファンクション_alarmScId.....	284
6.4.5	ファンクション_getAlarmId.....	285
6.4.6	ファンクション_getPendingAlarms.....	287
6.4.7	ファンクション_resetAlarmIdおよび_reset_AllAlarmId.....	287
7	一般的な標準ファンクションのプログラミング.....	289
7.1	一般的な標準ファンクションのプログラミング - 概要.....	289
7.2	数値標準ファンクション.....	290
7.2.1	数値ファンクションの特性.....	290
7.2.2	一般数値標準ファンクション.....	290
7.2.3	対数標準ファンクション.....	290
7.2.4	三角関数標準ファンクション.....	291

7.2.5	ビット列標準ファンクション	292
7.3	ビット列のビットへのアクセス	294
7.3.1	ファンクション_getBit	294
7.3.2	ファンクション_setBit	295
7.3.3	ファンクション_toggleBit	297
7.4	数値データタイプに対するビット操作	299
7.5	文字列処理(V4.0以降)	300
7.5.1	文字列編集用ファンクション	300
7.5.2	文字列編集時エラーの分析	302
7.6	データタイプ変換用の標準ファンクション	304
7.6.1	数値データタイプおよびビットデータタイプの変換用ファンクション	304
7.6.2	日付および時刻データタイプの変換用ファンクション	309
7.6.3	枚挙データタイプの変換用ファンクション	309
7.6.4	BYTEとSTRING間の変換	310
7.6.5	INT/FLOATおよびSTRINGデータタイプの変換用ファンクション	311
7.7	任意のデータタイプとバイト配列との間の変換	313
7.7.1	一般	313
7.7.2	ファンクションAnyType_to_BigByteArray、ファンクションAnyType_to_LittleByteArray	313
7.7.3	ファンクションBigByteArray_to_AnyType、ファンクションLittleByteArray_to_AnyType	315
7.8	ビット列データタイプの結合	318
7.8.1	ビット列データタイプの結合に関する全般的な情報	318
7.8.2	ファンクション_BYTE_FROM_8BOOL	318
7.8.3	ファンクション_WORD_FROM_2BYTE	319
7.8.4	ファンクション_DWORD_FROM_2WORD	319
7.8.5	ファンクション_DWORD_FROM_4BYTE	320
7.9	テクノロジーオブジェクトデータタイプの変換	322
7.9.1	ファンクションAnyObject_to_Object	322
7.10	浮動小数点数の検証用ファンクション	323
7.10.1	ファンクション_finite	323
7.10.2	ファンクション_isNaN	324
7.11	選択用ファンクション	326
7.11.1	ファンクションSEL	326
7.11.2	ファンクションMUX	327
7.11.3	ファンクションMAX	328
7.11.4	ファンクションMIN	328
7.11.5	ファンクションLIMIT	329
7.12	変数の操作	331
7.12.1	一般	331
7.12.2	[_testAndSetSemaphore]ファンクション	331
7.12.3	[_releaseSemaphore]ファンクション	332
7.13	システム変数および入/出力へのアクセス	334
7.13.1	システム変数および入/出力へのアクセスの概要	334
7.13.2	[_getSafeValue]ファンクション	334
7.13.3	[_setSafeValue]ファンクション	337
7.13.4	[_getInOutByte]ファンクション	341
7.14	ユーザープログラムからのデータのバックアップ	344
7.14.1	ユーザープログラムからのデータのバックアップの概要	344
7.14.2	[_saveUnitDataSet]ファンクション	344
7.14.3	[_loadUnitDataSet]ファンクション	349

7.14.4	[_exportUnitDataSet]ファンクション(カーネルV3.2以降)	352
7.14.5	[_importUnitDataSet]ファンクション(カーネルV3.2以降)	356
7.14.6	[_deleteUnitDataSet]ファンクション	360
7.14.7	[_getStateOfUnitDataSetCommand]ファンクション	362
7.14.8	[_checkExistingUnitDataSet]ファンクション	363
7.14.9	[_deleteAllUnitDataSets]ファンクション	365
7.15	commandId用ファンクション	368
7.15.1	[_getCommandId]ファンクション	368
7.15.2	[_getSyncCommandId]ファンクション	369
7.16	待機時間の定義	370
7.16.1	[_waitTime]ファンクション	370
7.17	デバイス固有のファンクション	372
7.17.1	[_getDeviceId]ファンクション	372
7.17.2	[_getMemoryCardId]ファンクション	373
7.17.3	[_setDeviceErrorLED]ファンクション	374
7.18	データタイプまたは変数のメモリサイズの特定	375
7.18.1	[_sizeOf]ファンクション	375
7.19	使用可能な追加のファンクション	377
7.20	特定のシステムファンクションの適用	378
7.20.1	メッセージのプログラム	378
7.20.1.1	一般	378
7.20.1.2	ファンクションの概要	378
7.20.1.3	アラームID	379
7.20.1.4	AlarmSのバッファ管理	380
7.20.1.5	メッセージ生成の例	381
7.20.1.6	故障番号とメッセージの状態のチェック(戻り値のフィルタ)	382
7.20.2	変数の一貫した読み取りおよび書き込み(セマフォ)	383
7.20.2.1	一貫したデータアクセス	383
7.20.2.2	セマフォ	384
7.20.2.3	例: セマフォによる一貫したデータアクセス	384
7.20.3	ユーザープログラムからのデータのバックアップおよび初期化	385
7.20.3.1	ユーザープログラムからのデータバックアップとデータ初期化 - 機能と手順	385
7.20.3.2	入力パラメータ	387
7.20.3.3	戻り値	389
7.20.3.4	保存場所およびメモリ要件	389
7.20.3.5	ステップの有効化条件	390
7.20.4	任意のデータタイプとバイト配列との間の変換(マーシャリング)	391
7.20.5	通信ファンクション	395
7.20.5.1	使用可能なファンクション	395
7.20.5.2	[_Xsend]のパラメータリスト	396
7.20.5.3	[_Xreceive]のパラメータリスト	397
7.20.5.4	[_GetStateOfXCommand]のパラメータリスト	398
7.20.5.5	SIMOTIONとSIMATIC S7デバイス間の通信	398
7.20.5.6	送受信プログラムの例	400
7.20.5.7	Ethernetを介したTCP/IPプロトコルによる通信	402
7.20.5.8	Ethernetを介したUDPプロトコルによる通信	402
7.20.6	同期起動	403
7.21	HMI (マンマシンインターフェース)接続	405
7.21.1	HMIとSCOUTまたはSIMOTIONとの間のインターフェース	405
7.21.2	ProToolアプリケーションの例	405
7.21.3	HMIデバイスへの一貫性のあるデータアクセス(例)	405

8	一般的なシステムファンクションブロックのプログラミング	409
8.1	ファンクションブロックの概要	409
8.2	双安定要素(フリップフロップのセット)	412
8.3	エッジ検出	414
8.4	COUNTER	416
8.4.1	カウンタの概要	416
8.4.2	CTUアップカウンタ	416
8.4.3	CTU_DINTアップカウンタ	417
8.4.4	CTU_UDINTアップカウンタ	418
8.4.5	CTDダウンカウンタ	418
8.4.6	CTD_DINTダウンカウンタ	419
8.4.7	CTD_UDINTダウンカウンタ	419
8.4.8	CTUDアップ/ダウンカウンタ	419
8.4.9	CTUD_DINTアップ/ダウンカウンタ	420
8.4.10	CTUD_UDINTアップ/ダウンカウンタ	421
8.5	タイマー	422
8.6	ビット列データタイプの分割	426
8.6.1	ビット文字列データタイプの分割の概要	426
8.6.2	_BYTE_TO_8BOOLファンクションブロック	426
8.6.3	_WORD_TO_2BYTEファンクションブロック	427
8.6.4	_DWORD_TO_2WORDファンクションブロック	428
8.6.5	_DWORD_TO_4BYTEファンクションブロック	429
8.7	SIMATIC S7コマンドのエミュレーション	431
8.7.1	一般	431
8.7.2	_S7_COUNTERファンクションブロック	431
8.7.3	_S7_TIMERファンクションブロック	432
9	エラーの原因および効率的なプログラミング	435
9.1	プログラミング中のエラーソース	435
9.1.1	プログラミング中のエラーソース	435
9.1.2	算術式を割り当てるときのデータタイプのチェック	435
9.1.3	周期的タスクのファンクションの起動の毎回のチェック	436
9.1.4	周期的タスクでの待機時間	436
9.1.5	正しいcommandIdパラメータの使用	437
9.1.6	エラーの追跡(STプログラム)	438
9.1.7	ダウンロード時のエラー	438
9.1.8	CPUがRUNに切り替わらない	439
9.1.9	CPUがSTOP状態に移行	439
9.1.10	システムクロックのチェックおよび設定	440
9.1.11	REALまたはLREAL変数の比較	440
9.1.12	範囲違反のチェック	441
9.1.13	ローカルデータスタックのサイズの設定	441
9.2	効率的なプログラミング	442
9.2.1	効率的なプログラミング - 概要	442
9.2.2	実行時間に最適化されたプログラミング	442
9.2.2.1	実行時間に最適化されたプログラミング	442
9.2.2.2	入力および出力へのアクセスの最適化	442
9.2.2.3	システム変数へのアクセスの最適化	442
9.2.2.4	最適な変数宣言	443
9.2.2.5	ファンクションブロックのパラメータへのアクセスの最適化	443
9.2.2.6	ファンクションブロック呼び出しの最適化	443

9.2.2.7	プログラム構造の最適化.....	443
9.2.2.8	実行システムの最適化	443
9.2.3	変更に最適化されたプログラミング	444
9.2.3.1	変更に最適化されたプログラミング	444
9.2.3.2	1つのユニットでの保持される変数の宣言	444
9.2.3.3	1つのユニットでのHMI変数.....	444
9.2.3.4	デバイスグローバル変数およびユニットグローバル変数の使用	445
9.2.3.5	MotionTaskの集中起動およびリセット	446
A	付録A.....	447
A.1	シンボリック定数	447
A.2	予約済みの識別子	449
	索引	477

システムの概要

このセクションでは、SIMOTIONモーションコントロールシステムの構成と用途の概要を説明します。

1.1 SIMOTIONモーションコントロール

SIMOTIONでは、モーションコントロールアプリケーションとテクノロジタスクに重点を置いた、自動化 / ドライブソリューション向けの最適化されたシステムプラットフォームを提供します。

SIMOTIONは、生産機械の最新動向に対する答えです。

- **メカトロニクス**

メカトロニクスとは、機械を力学的側面からのみ評価するのではなく、機械機器、電気機器、および制御 / ソフトウェア技術が均一に統合された1つの完成システムとして評価することです。

メカトロニクスの一環として、比較的柔軟性の低い機械機器(カム、ギア、ラインシャフトなど)はインテリジェントなソフトウェアソリューションで置き換えられます。

- **PLC機能とモーションコントロールテクノロジの統合**

従来からある純粋な自動化機能とモーション機能の区別を解消します。

これらの機能は、ハードウェア側とソフトウェア側の両方で一体化されます。

- **ユーザビリティ**

一貫したエンジニアリングシステム(SIMOTION

SCOUT)は、設定、パラメータ設定、およびプログラミングの一貫性を保証します。

自動化タスクとモーションコントロールタスクは、同じ言語でプログラミングされます。

- **標準化**

工業オートメーションは、ますますMicrosoft

WindowsやEthernetなどのPC環境で管理されるようになってきています。

標準化された世界共通のプログラミング言語は、お客様のシステム操作性を大幅に改善しました。

- **モジュール方式の機械設計**

また、標準化に向けた動きには、機械設計も含まれています。

そのために、機械設計を複数のサブコンポーネントに分割する試みが行われています。

モジュール方式の長所は、個々のサブコンポーネントを標準化して、標準コンポーネントとして異なるタイプの機械に組み込めることです。

1.2 用途

効率的なコントロールシステムにするには、オープン制御のコンセプトを使用して現在のタスクおよび生産機械分野の将来動向に対応する必要があります。

SIMOTIONは、生産機械の自動化を目的とした、一貫したモーションコントロールシステムです。

この一貫性には、エンジニアリング、プログラミング、通信、データ管理、およびマンマシンインターフェース(HMI)が含まれます。したがって、異なるハードウェアプラットフォーム上であっても、システム全体の一貫性が確保されます。

これらの動向は主に機械工学の以下の分野に関係しており、またシーメンスのモーションコントロールシステムはこれらの分野に特に適しています。

- 包装機類
- プラスチック加工機械
- 金属成型技術
- 繊維機械
- 印刷機
- 木材、ガラス、および陶器製造業で使用する機械
- およびその他の機械

標準論理およびドライブ関連のタスクに加えて、この分野の自動化ソリューションでは、統合された一貫性のあるモーションコントロールとテクノロジータスクの取り込みもますます必要になってきています。

SIMOTIONモジュラシステムは、SCOUTエンジニアリングシステムと、さまざまなハードウェアプラットフォームに対応する共通ランタイムシステムで構成されています。

SCOUTエンジニアリングシステムは、すべてのハードウェアプラットフォームで同一です。

。コンフィグレーション、パラメータ割り当て、およびプログラミングは、グラフィックベースまたはテキストベースの方法を使用して実行されます。

また、SCOUTエンジニアリングシステムは業界特有のソリューションの基礎を提供します。

。基本機能を拡張するためにどのランタイムソフトウェア(位置、ギアなど)をロードするかは、使用するアプリケーションに基づいて決定します。

SIMOTIONはさまざまなハードウェアプラットフォームで動作することから、すべての必要条件を満たす多目的のソリューションを提供できます。

1.3 PLCとモーションコントロールの統合

SIMOTIONでは、開ループ制御、テクノロジー、およびモーションコントロールが統合されます。

したがって、必要な制御の間にはハードウェアインターフェースおよびソフトウェアインターフェースは存在しません。

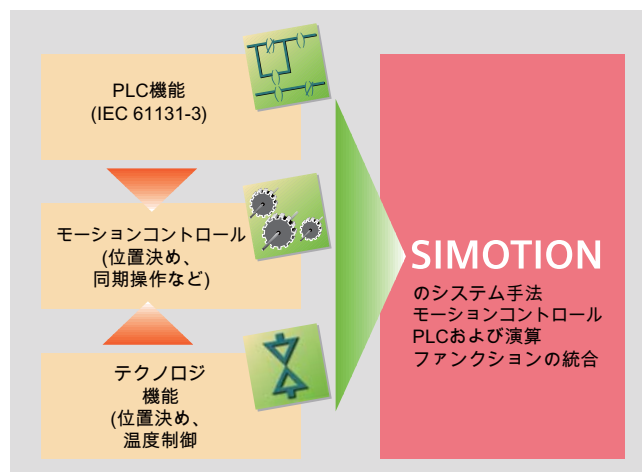


図 1-1 PLCとモーションコントロールの統合

この統合は、ハードウェア側では、モーション機能をプログラマブルコントローラで処理できることを意味します。ハードウェアプラットフォームは個別に選択できます。

ソフトウェア側では、自動化機能とモーション機能の統合によりエンジニアリングが簡素化されます。これは設定ら、パラメータ割り当ておよびプログラミングにまで及びます。

SIMATICとの整合性はもう1つの基本的な特徴であり、1つのプラントで両方のシステムを一緒に使用することができます。

下記も参照

ハードウェアプラットフォーム (ページ 19)

1.4 統合オートメーション(Totally Integrated Automation)

統合オートメーションの核心部分は、プログラミングと設定、データ管理、および通信の3つにわたる一貫性です。

つまり、SIMATICにより解決されるすべての自動化タスクでは、統合オートメーションの以下のような多くの利点を完全に利用することができます。

- エンジニアリングコストの大幅な削減
- 自動化環境でのシステムの無停止稼働
- すべてのコンポーネントの基盤になる1つのソフトウェア

機械やシステムにカスタマイズされたソリューションを実装する必要がある自動化タスクと、小規模な自動化タスクの間には違いはありません。

SIMATICを使用した統合オートメーションには、自動化環境に必要なすべてのテクノロジーが組み込まれています。これらのテクノロジーには、プログラマブルコントローラ、PCベースのコントローラ、自動化コンピュータ、リモートI/O、HMIシステム、通信ネットワーク、またはプロセス制御システムが含まれます。

モジュール方式であるため、一貫性のある1つの完全なシステムを使用して、処理の必要条件に適合および採算性のある的確なソリューションを実装できます。

SIMOTIONとSIMATICは、統合オートメーションという意味で統合されています。

この一貫性は、2つの方法を通して保証されます。1つはSIMOTION

SCOUTのSIMATICへの統合、もう1つは相互に適切な動作を行うために同じ工学理念を使用することです。

SIMATICで利用できないエンジニアリングプロセス(モーションコントロール、出力カムコントローラなど)、または分散システムの要求に適応するためにSIMOTIONで必要とされるエンジニアリングプロセスは、ユーザビリティが最適になるように選択されます。

さらに、SIMOTIONはPROFINETに統合されており、このために必要なシステム機能がすべて含まれています。

1.5 ハードウェアプラットフォーム

SIMOTIONでは、さまざまなハードウェアプラットフォームをサポートしています。使用されるハードウェアコンポーネントは、主に必要条件に応じて決まります。また、自動化タスクは、異なる対象システムに配布できます。

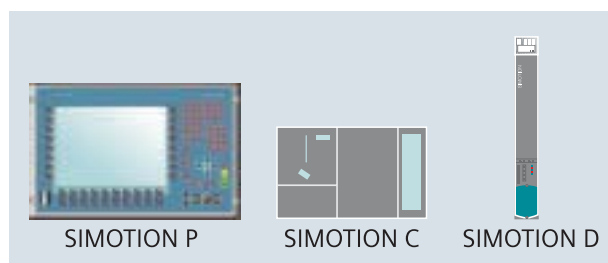


図 1-2 SIMOTIONのハードウェアプラットフォーム

以下のプラットフォームが使用できます。

- PCベース(**SIMOTION P350**)

SIMOTION用のリアルタイム拡張が搭載されているSIMOTION P3xxは、PCベースのモーションコントロールシステムとしてWindows XPオペレーティングシステム上で動作します。

SIMOTIONアプリケーションとは別に、他のPCアプリケーションも起動できます。

SIMOTION P350は、以下の用途に適しています。

- PC環境向けのオープンアーキテクチャが必要なアプリケーション
- ハードウェアベースの制御と視覚化が必要なアプリケーション
- 大規模なデータストレージ、評価、およびロギング

- コントローラベース(**SIMOTION C2xx**)

このコントローラは、SIMATIC S7-300の実装技術の中では、統合されたアナログドライブインターフェースおよび複数のデジタル入力 / 出力で構成されています。

さらに、コントローラは、SIMATIC S7-300製品群のI/Oモジュールで拡張できます。PROFIdriveインターフェースによる2つのPROFIBUS接続、および1つの工業用Ethernetを使用すると、他の機械部品との通信が可能になります。

また、PROFIBUSは、操作パネル(たとえばSIMATIC HMI製品群からの)または上位にある制御システム(SIMATIC S7など)との通信にも使用できます。

ProTool/Pro、WinCC

flexible、またはOPCインターフェースを搭載したPCと同様に、SIMATIC HMIパネルはOperator Systemに適しています。

SIMOTION C2xxでは、以下のことが可能です。

- ドライブの自由な選択
- 広範囲にわたるプロセス信号
- 内蔵アナログインターフェースを介したアプリケーションのレトロフィット
- アナログドライブとステッピングドライブの直接接続

- ドライブベース(SIMOTION D4xx)

SIMOTION Dは、SINAMICS S120多軸ドライブシステムの閉ループ制御モジュールに直接統合されています。

PROFIdriveインターフェースによる2つのPROFIBUS接続、および2つの工業用Ethernetインターフェースを、オンボードで利用できます。

SIMOTION D4xxでは、SIMOTION Cxxの機能を提供し、また制御とドライブを1つのコンポーネントに統合することを可能にします。

SIMOTION D4xxでは、SIMOTIONの完全な機能を提供し、また制御とドライブを1つのコンポーネントに統合することを可能にします。

1.6 システムアーキテクチャ

SIMOTIONシステムアーキテクチャでは、分散化、異種の対象システム、および分散知能の実装を進める動きに対応できます。

下記も参照

テクノロジーオブジェクト(TO) (ページ 31)

SIMOTION SCOUTエンジニアリングシステム (ページ 22)

SIMOTIONプロジェクト (ページ 23)

オフライン / オンラインモード (ページ 25)

プログラミング (ページ 25)

SIMOTIONのシステムアーキテクチャ (ページ 21)

実行システム、タスク、およびシステムサイクルクロック (ページ 127)

1.6.1 SIMOTIONのシステムアーキテクチャ

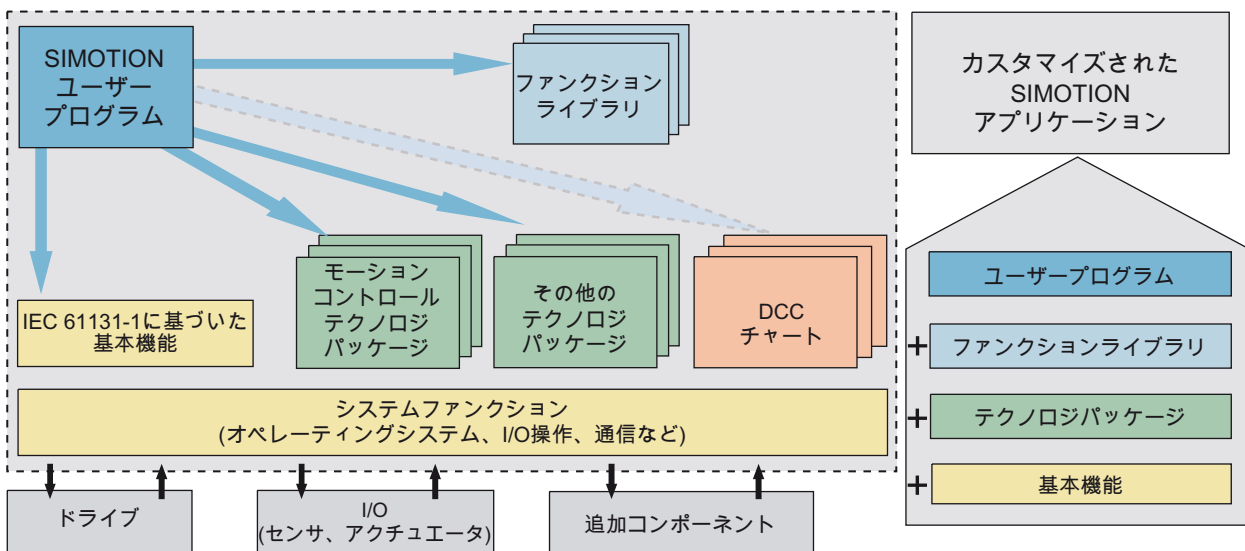


図 1-3 SIMOTIONのシステムアーキテクチャ

デバイスの基本機能(SIMOTION Kernel)には、論理演算と算術演算の機能だけではなく、開ループ制御と閉ループ制御のための機能も含まれています。プログラムは、周期的、時間駆動、または割り込み駆動で実行できます。そのため、SIMOTION Kernelには、ほぼすべてのアプリケーションに必要な機能が含まれています。また、SIMOTION Kernelは基本的に、IEC 61131-3コマンドセットを搭載したPLCと、さまざまなコンポーネントを制御するシステムファンクション(入力や出力など)に対応しています。

デバイスのSIMOTION

Kernelは、テクノロジーパッケージをダウンロードすることで拡張できます。

テクノロジーパッケージには、SIMOTION

Kernelへのアクセスと同じようにユーザープログラムから特別な言語コマンドを使用してアクセスします。

特定のタスクに対しては、既存のアプリケーションを使用するか、または必要なアプリケーションをユーザー自身がプログラムしてリンクできます。アプリケーションはIEC 61131-3に従ってプログラムし、個別のタスクに適合させることができます。

さらに、SIMOTIONでは、システムファンクションとモーションファンクションを含むファンクションライブラリが提供されます。

ファンクションライブラリには、ファンクションおよびテクノロジーオブジェクトのシステム変数へのアクセスが含まれます。また、ファンクションライブラリは、SIMOTION SCOUTの関連するデバイスおよびテクノロジーパッケージにリンクされます。

閉ループ制御ファンクションなどの特殊なタスクに対しては、配線図を使用できます。配線図は、グラフィックツールにより接続されたブロックを使用して、対応するDCC実行タスク内で実行します。

1.6.2 SIMOTION SCOUTエンジニアリングシステム

機械の自動化には、以下の作業が必要です。

- ハードウェアとソフトウェアコンポーネントの選択 / 設定、ハードウェアとソフトウェアの設定(HW Configによる通信ネットワークを含む)
- テクノロジーオブジェクトの作成と設定
- ユーザープログラムの作成
- ドライブユニットのテストと試運転
- 機械操作制御(HMI)のリンク
- 手順の完了(機械に関するドキュメントの作成など)

SIMOTION

SCOUTエンジニアリングシステムでは、一貫性のあるユーザービューと柔軟性のある機能が提供されます。

生産機械用の個々の自動化タスクは、一貫性と整合性のあるユーザーインターフェースで編成されます。

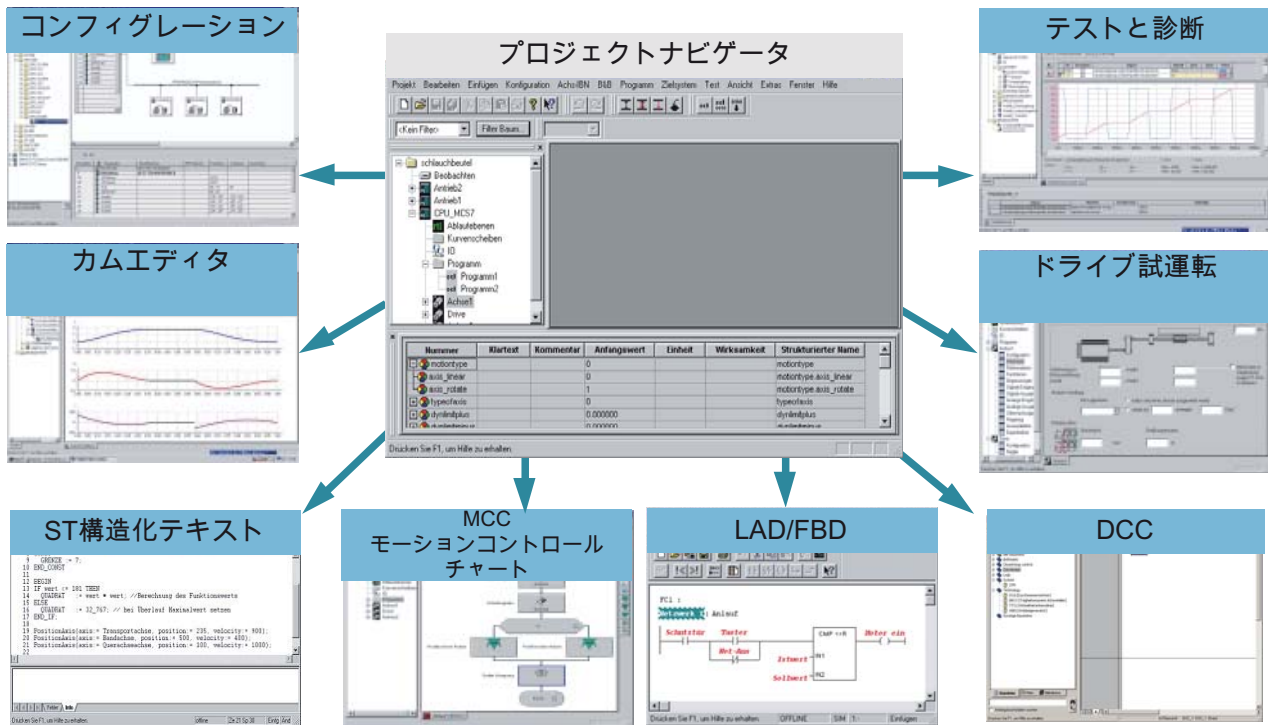


図 1-4 SIMOTION SCOUTエンジニアリングシステム

SIMOTION

SCOUTエンジニアリングシステムは強力なツールで、必要な工学的手順の最適なサポートをユーザーフレンドリな方法で提供するPC開発環境として機能します。SIMOTION SCOUTエンジニアリングシステムはSIMATICの環境に統合されており、STEP7用のオプションパッケージとして動作します。また、自動化タスクの最適なユーザビリティおよび包括的で機能指向のビューを提供することに、特別な注意が払われています。

1.6.3 SIMOTIONプロジェクト

プロジェクトには、すべてのSIMOTIONデバイスとドライブなどが含まれます。関連するデバイスは、テクノロジーオブジェクトとプログラムに階層的に割り当てられます。この階層構造は、プロジェクトナビゲータに表示されます。

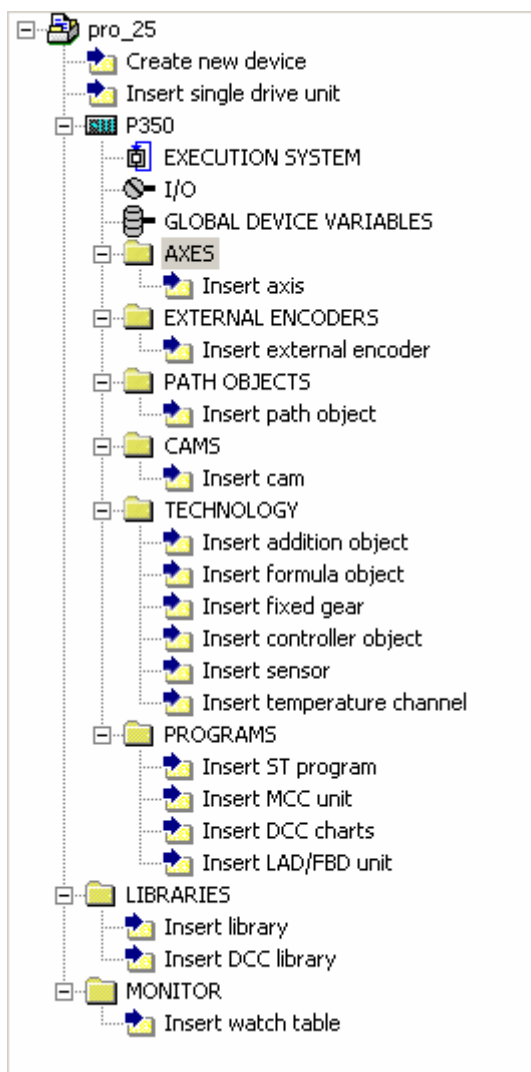


図 1-5 SIMOTION SCOUTのプロジェクトナビゲータ

プロジェクトは、データ管理階層構造の最上位レベルにあります。たとえば、SIMOTION SCOUTでは、生産機械に属しているすべてのデータはプロジェクトディレクトリに保存されます。

1.6.4 オフライン / オンラインモード

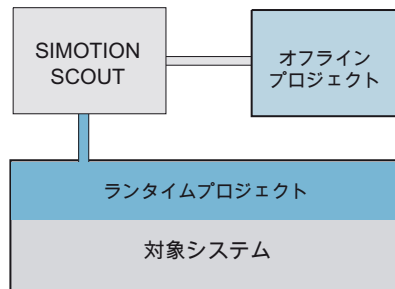


図 1-6 SIMOTION SCOUTによるオフラインとランタイムプロジェクト間の接続

SIMOTION SCOUTは、以下の2つのモードで使用できます。

- **オフラインモード**では、プロジェクトとユーザープログラムを作成できます。

オフラインモードでは、SCOUTエンジニアリングシステムをランタイムシステムに接続しない状態で操作します。

データとテクノロジーオブジェクトは、エンジニアリングシステム内で維持管理されます。

オフラインモードは、先行操作モードです。つまり、システム変数コンフィグレーションデータの変更は、オフラインで行われます。

- **オンラインモード**では、プロジェクトとデータを対象システムにロードして、アプリケーションを制御および監視できます。

SCOUTエンジニアリングシステム内のデータは、オフラインプロジェクトとして保持されます。

オンラインで変更されたコンフィグレーションデータは、オフラインプロジェクトに再ロードできます。一方、オンラインで変更されたシステム変数は再ロードできません。

1.6.5 プログラミング

開ループ制御タスクとモーションコントロールタスクは、ユーザープログラムの中で事前に定義されます。

ユーザープログラムの作成には、以下のプログラミング言語が使用できます。

- **MCC - モーションコントロールチャート**

グラフィックベースのプログラミング言語で、ロジックとモーションコントロール機能のプログラミングに使用します。MCCチャートが作成されます。

- **ST - 構造化テキスト**

IEC 61131-

3に準拠したテキストベースのプログラミング言語で、複数のプログラムで構成されるSTソースを作成できます。

また、外部のSTエディタを使用してSTソースファイルを編集できます。

MCCを使用して作成されたプログラムはSTに変換できますが、逆の変換できません。

- **LAD/FBD - ラダーロジック / ファンクションブロックダイアグラム**

IEC 61131-

3に従ったグラフィックベースのプログラミング言語で、PLCopenブロックによるロジックとモーションコントロールのプログラミングに使用します。

- **DCC - ドライブコントロールチャート**

グラフィックベースのプログラミング言語で、ドライブ機能のプログラミングに使用します。

また、外部のDCCエディタを使用してドライブコントロールチャートを編集できます。

プログラミング言語の詳細については、**SIMOTION MCC**、**SIMOTION ST**、および**SIMOTION LAD/FBD**の各プログラミングマニュアルに記載されています。

SIMOTIONでのプログラミングには、以下の利点があります。

- 1つのプロジェクトで、異なるプログラミング言語のプログラムを使用できます(MCC、ST、およびLAD/FBD)。
- プログラミングは、ハードウェアプラットフォームからは独立しています。
- 1つのプログラムには、PLC、モーションコントロール、およびテクノロジーを実装できます。
- PROFIDRIVEで利用できるドライブパラメータに直接アクセスする機能があります。

モジュール方式の機械設計は、以下の方法でサポートされています。

- ライブラリを使用したモジュール方式のソフトウェア開発
- 個別の機械モジュールへの分割
- DPスレーブとテクノロジーオブジェクトの有効化 / 無効化
- 同期制御用のコマンド

テクノロジーパッケージおよびテクノロジーオブジェクト

2

このセクションには、テクノロジーオブジェクトおよびその基本機能と特性に関する一般情報が記載されています。

2.1 概要

SIMOTIONの基本機能

テクノロジーパッケージとテクノロジーオブジェクトがないSIMOTIONシステムでは、実行システムおよびIEC 61131-3に基づいたコントロールシステムの基本機能が提供されます。

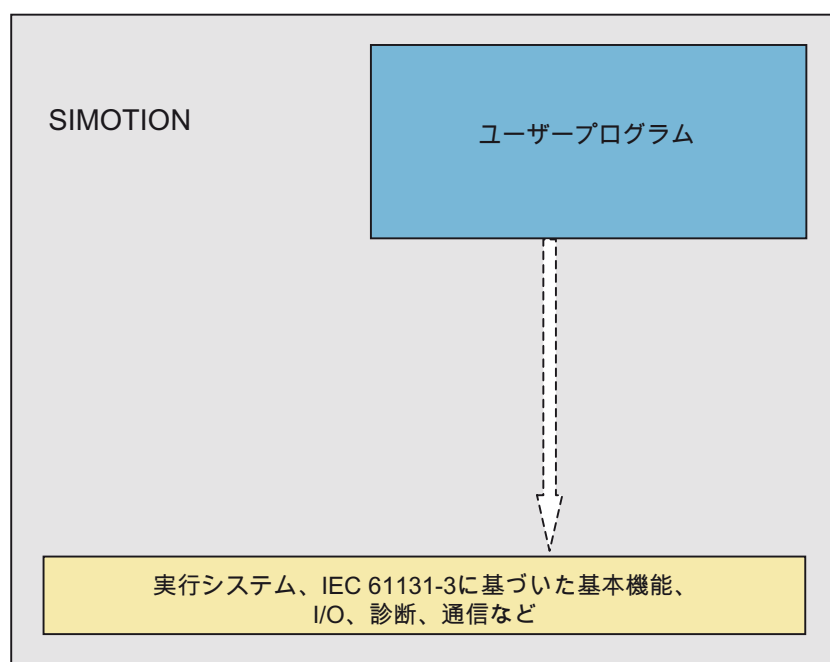


図 2-1 SIMOTIONの基本機能

ユーザープログラムは、さまざまなタスクに割り当てることができます。

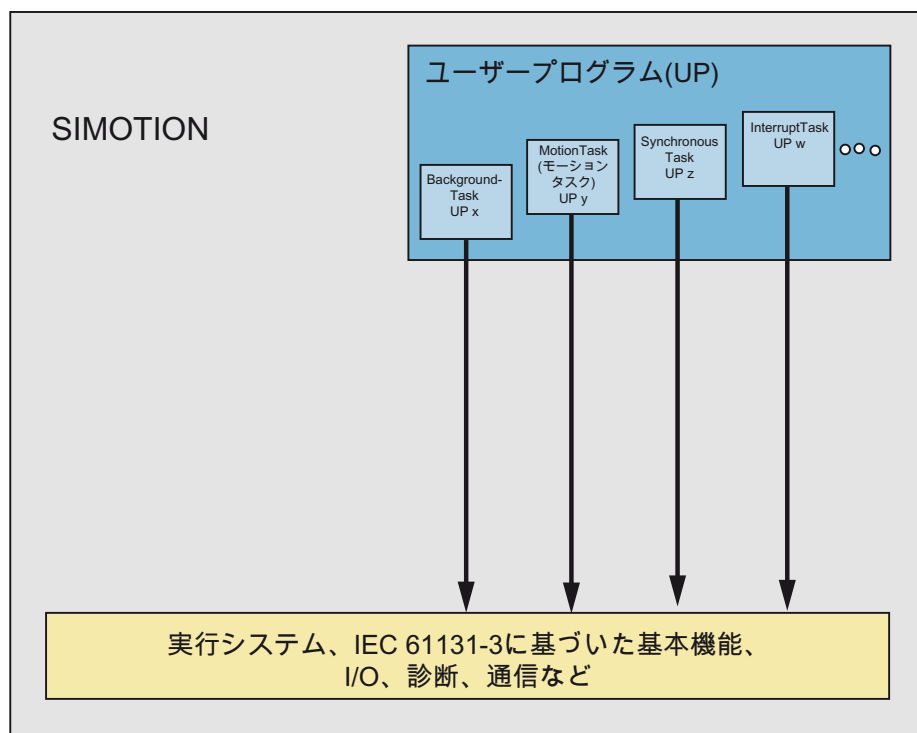


図 2-2 ユーザープログラムのタスクへの割り当て

テクノロジーオブジェクト

SIMOTIONでは、テクノロジーとモーションコントロール用のテクノロジーオブジェクトが提供されます。

テクノロジーオブジェクトはアクチュエータとセンサのテクノロジービューをユーザーに提供し、さらに、これらに対して以下の例のようなテクノロジー機能を提供します。

- ドライブとエンコーダ用のTO軸
- 1つのエンコーダのみに使用するTO External Encoder
- 規定の方法で切り替えられる出力に使用するTO outputCam / camTrack
- 測定入力用のTO measuringInput

さらに、システムレベルでテクノロジーデータを用意するために、以下の例のようなテクノロジーオブジェクトも使用できます。

- 2つの軸間または1つの軸をエンコーダ値に同期して動作させるためのTO FollowingObject
- パスに沿って移動するパス軸およびパスに同期した位置決め軸に使用するTO path
- 複雑なプログラマブルファンクションを表すためのTO cam
- システム側でモーションデータとテクノロジーデータを処理するためのTO additionObjectおよびTO formulaObject

テクノロジーオブジェクトは、システムにより、IPO task、IPO_2 task、またはサーボタスクなどのシステムタスクに実装されます。

インスタンス化

システムでは、テクノロジーオブジェクトを、インスタンス化を使用して特定のアプリケーションに適合させた**テクノロジーオブジェクトタイプ**として提供します。

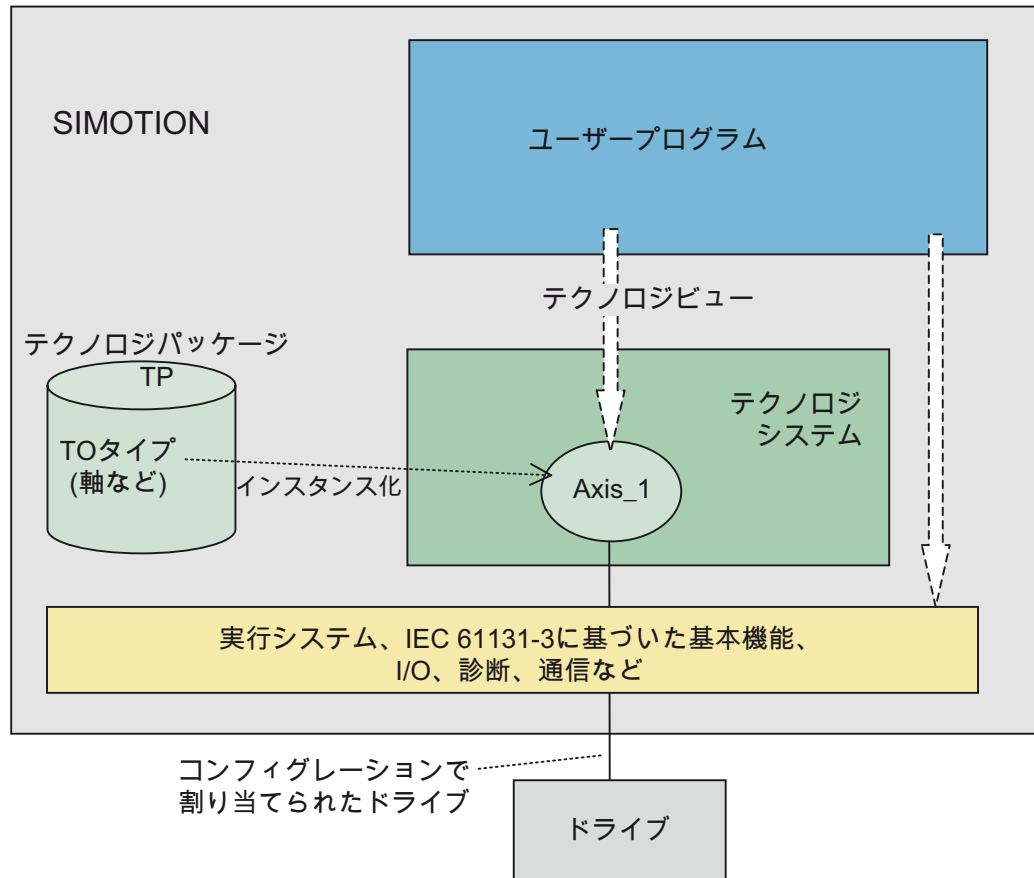


図 2-3 TO axisなどのテクノロジーオブジェクトインスタンス上でのプログラミングモデル

テクノロジーオブジェクトタイプは1つのテクノロジーパッケージに組み込まれており、ランタイムシステムにロードされます。

2.2 テクノロジーパッケージ

SIMOTIONでは、テクノロジーオブジェクトタイプはテクノロジーパッケージにより提供されます。

テクノロジーオブジェクトタイプは、プロジェクトとともにSIMOTIONランタイムシステムにロードされます。

以下のテクノロジーパッケージが使用できます。

- **TP**
Camには、テクノロジーとモーションコントロール用の基本テクノロジーオブジェクトタイプ(driveAxis、posAxis、followingAxis、followingObject、cam、outputCam、camTrack、measuringInput)が含まれています。
- さらに、**TP**
Cam_extには、システムレベルでテクノロジーデータを用意するためのオブジェクト(additionObjectやformulaObjectなど)が含まれています。
- **TControl**には、温度コントローラテクノロジーオブジェクトタイプが含まれています。
- **TP Path**には、パステクノロジーオブジェクトタイプが含まれています。

2.3 テクノロジーオブジェクト(TO)

テクノロジーオブジェクト / テクノロジーオブジェクトタイプでは、テクノロジーとモーションコントロール用の機能が提供されます。これらの機能は、テクノロジーシステムファンクションで構成され、強固なハードウェア結合を隠蔽します。

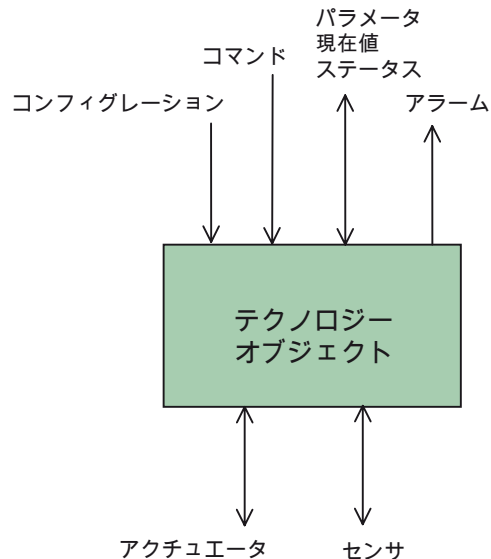


図 2-4 テクノロジーオブジェクトで使えるインターフェース

システムレベルでTOをまたぐテクノロジーデータの処理を行うために、テクノロジーオブジェクトでは定義済みの入力と出力インターフェースが提供されています。

下記も参照

使用できるテクノロジーオブジェクト (ページ 39)

相互接続 (ページ 36)

2.3.1 インスタンス化および設定

Add <テクノロジーオブジェクト>を使用して、SIMOTION SCOUT内で対応するTOタイプのインスタンスを作成します (データ、パラメータ、アラームリストなどが、システム内で作成されます)。

必要に応じて、テクノロジーオブジェクトタイプを指定する必要があります。たとえば、ドライブ軸、位置決め軸、従動軸などの軸に対して指定します。

アクチュエータ / センサへのハードウェア / ソフトウェアインターフェースが定義されます。たとえば、PROFIdriveメッセージフレームでのハードウェアアドレス、データ幅、メッセージフレームタイプが定義されます。

基本設定が定義されます。たとえば、IPOまたはIPO_2 サイクルクロックのテクノロジーシステムファンクションの処理が定義されます。

2.3 テクノロジーオブジェクト(TO)

SIMOTION

SCOUTでは、テクノロジーオブジェクトを設定するためのウィザードとダイアログボックスが提供されます。

テクノロジーオブジェクトの基本機能は、コンフィグレーションデータにより定義されます。

設定画面フォームとパラメータ設定画面フォームにあるすべてのエントリは、コンフィグレーションデータとシステム変数に出力されます。これらのデータと変数は、画面フォームにツールヒントとして表示されます。

コンフィグレーションデータとシステム変数の詳細については、『SIMOTION Reference Lists』を参照してください。

個別のテクノロジーオブジェクトを作成および設定する手順については、それぞれの機能マニュアルを参照してください。

コンフィグレーションデータの大部分は、実行時にユーザープログラムまたはエキスパートリストを通して修正できます。

コンフィグレーションデータとシステム変数の詳細については、『SIMOTION Reference Lists』を参照してください。

下記も参照

一般的な標準ファンクションのプログラミング - 概要 (ページ 289)

システム変数 (ページ 83)

コンフィグレーションデータ (ページ 87)

2.3.2 プログラミング

システム変数は、テクノロジーオブジェクトのテクノロジーパラメータと表示値で構成されています。

通常、SIMOTION

SCOUTでは、テクノロジーパラメータと標準値 / デフォルト値を設定するための画面フォームが提供されます。

システム変数は、連続して読み取られる個別の値または構造体です。

注記

オブジェクトは、テクノロジーオブジェクトが再開した後に再初期化されます。

これが理由で、たとえば、再開が完了するまではシステム変数を読み取ることはできません。つまり、再開が完了するまでシステム変数は使用できません。

システム変数の関連情報については、システム変数 (ページ 83)を参照してください。

2.3.3 プログラミング

テクノロジーカル機能は、特定のコマンドを使用して有効 / 無効にされます。

同期 / 非同期コマンド実行

コマンドは、*同期*または*非同期*になるように設定できます。

モーションコマンドを同期設定にすると、特定のモーションステータスになるかまたはモーション(位置決めなど)が完了するまで、コマンドを待機させることができます。
この設定は、特に、連続タスク(モーションタスク)のモーションシーケンスをプログラミングする際に役立ちます。

戻り値

戻り値では、ファンクション呼び出し(関数呼び出し)の結果が返されます。たとえば、ファンクションが計画に従って実行 / 実行されたか、またはエラーがあるかが示されます。

CommandId

CommandIdは、TOコマンドを一意に識別または追跡するのに使用されます。

パラメータ、条件を有効にする手順、および診断などの詳細については、以下を参照してください。:

- 本書に記載されている「CommandID関連のファンクション (ページ 368)」および各種の例を参照してください。
 - SIMOTIONのMCCモーションコントロールチャート、MCC Chartsのプログラミング
- TOコマンドの詳細については、『SIMOTION Reference Lists』を参照してください。

プログラミングモデル

コマンドは、タスクシステムの実行レベルに順に割り当てられるタスクに割り当てられます。

また、コマンドは、システムのすべてのユーザープログラムタスクから発行できます。

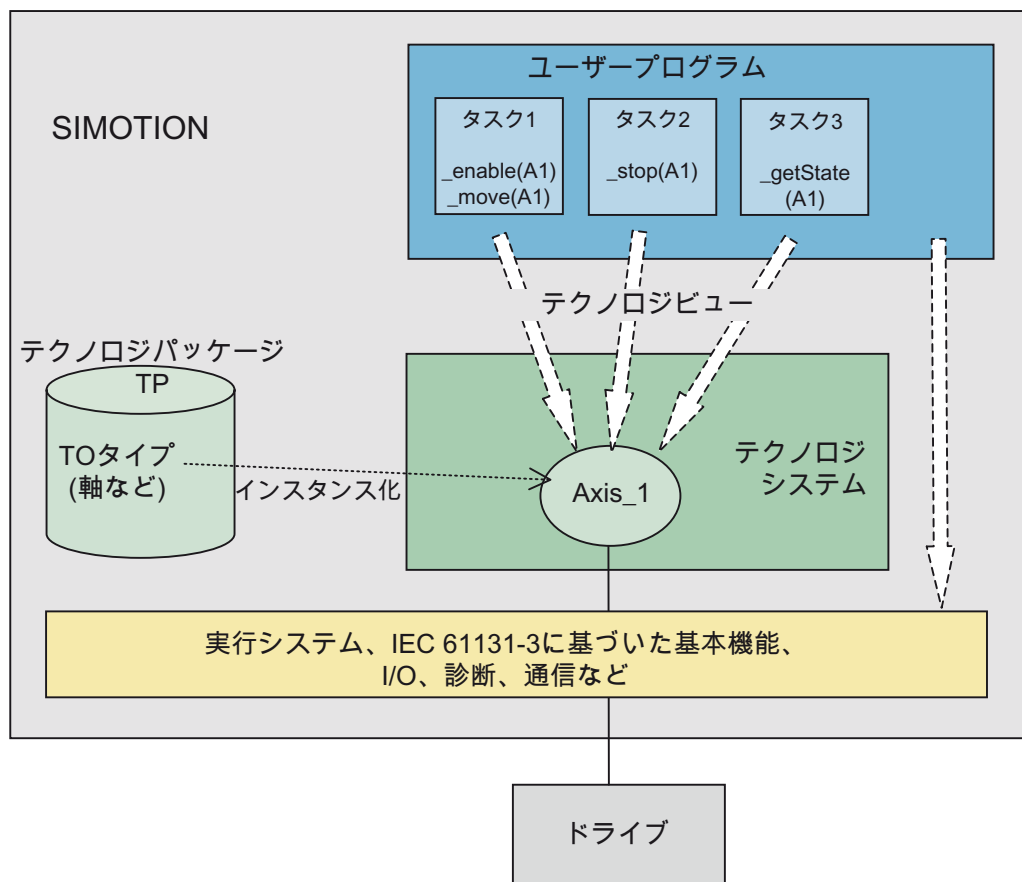


図 2-5 TO axisなどのテクノロジーオブジェクトインスタンス上でのプログラミングモデル

テクノロジーオブジェクト上でのコマンドの実行時間は、コマンドの有効性を計る指標ではありません。

複数のタスクからコマンドが発行される場合、プログラミングの一貫性は、ユーザープログラム側で確保する必要があります。

TO上でのコマンド実行 / TOコマンドの有効性

ユーザープログラム(ユーザープログラムタスク)からTOに発行されるコマンドは、一般に、以下のように分類されます。

- ユーザープログラムタスクのコンテキスト / シーケンスで直ちに実行されるコマンド
これらのコマンドは、ユーザープログラムタスク内のファンクションと同じように処理されます。

ユーザープログラムはファンクションの結果が返された場合にのみ継続されるので、これらのコマンドは同期して実行されます。

例: TO値の読み取り

- コマンドバッファに入力され、互いに上書きするコマンド
- コマンドバッファに入力され、コマンドバッファに空きがないと拒否されるコマンド

TOコマンドでは、2つの処理サイクルクロックの間で2つ以上のコマンドがコマンドグループに対して発行された場合の、TO上でのコマンドの応答を指定します。たとえば、置換 / 上書きまたはコマンド拒否を指定します。

コマンドバッファおよびコマンドの有効化については、個別のテクノロジーオブジェクトの機能マニュアルを参照してください。

実行特性

テクノロジーオブジェクトの実行特性は、オブジェクトに固有のものです。

テクノロジーオブジェクトの実行では、DPサイクルクロック、サーボサイクルクロック、およびIPO / IPO_2サイクルクロック(温度コントローラを除く)といった同期実行レベルが提供されます。

テクノロジーオブジェクトのインスタンスは、別の実行レベルで処理されます。

- コマンド評価およびモーションコントロールは、IPO/IPO_2サイクルクロックで処理されます。
- 位置と設定値制御は、サーボサイクルクロックで処理されます。
- ドライブとの通信は、DPサイクルクロックでPROFIBUS DPを経由して処理されるか、またはPNサイクルクロックでIRT使用のPROFINET IOを経由して処理されます。

この処理は、システム設定とオブジェクト設定により影響を受けます。

- システムサイクルクロック設定
システムサイクルクロックを設定すると、軸(IPO、IPO_2)のモーションコントロール、位置制御(サーボ)、およびPROFIBUS DPを経由またはIRT使用のPROFINET IOを経由する通信での、サンプリング時間も設定されます。
- オブジェクトを設定する際に、モーションコントロールをIPOまたはIPO_2サイクルクロックのどちらで実行するかを指定できます。
これにより、タイムクリティカルな動作とそうではない動作を区別できます。
- テクノロジーオブジェクトは、ユーザープログラムからシステムファンクション呼び出しを通して制御されます。

アラーム

テクノロジーオブジェクトでは、TOに必要なI/Oだけではなくテクノロジー機能の実行と実行可能性も監視し、必要に応じて**テクノロジーカルアラーム**を作成します。

テクノロジーカルアラームには、TOローカルのアラーム反応とグローバルのアラーム反応の2種類があります。たとえば、TOローカルの反応としては、モーションの停止などがあり、またグローバルの反応としては、システムの停止または追加的な反応を指定できるアラームタスク呼び出しがあります。

ローカルとグローバルのアラーム反応は、特定のアラームに対して設定できます。

TOアラームの詳細については、『SIMOTION Reference Lists』を参照してください。

下記も参照

実行システム (ページ 127)

周期的プログラミングと逐次的プログラミングの相違点 (ページ 67)

プロセスアラーム (ページ 110)

2.3.4 相互接続

テクノロジーオブジェクトでは、テクノロジーオブジェクト間でシステムレベルのデータ / 情報交換を行うために、相互接続インターフェースが定義されています。
一般に、これらのインターフェースを使用すると、個別のテクノロジーオブジェクト間で双方向のデータ交換ができます。

外部インターフェースのタイプは、各TOタイプと異なっていてもかまいません。
また、相互接続やアクチュエータ / センサは必要ありません。

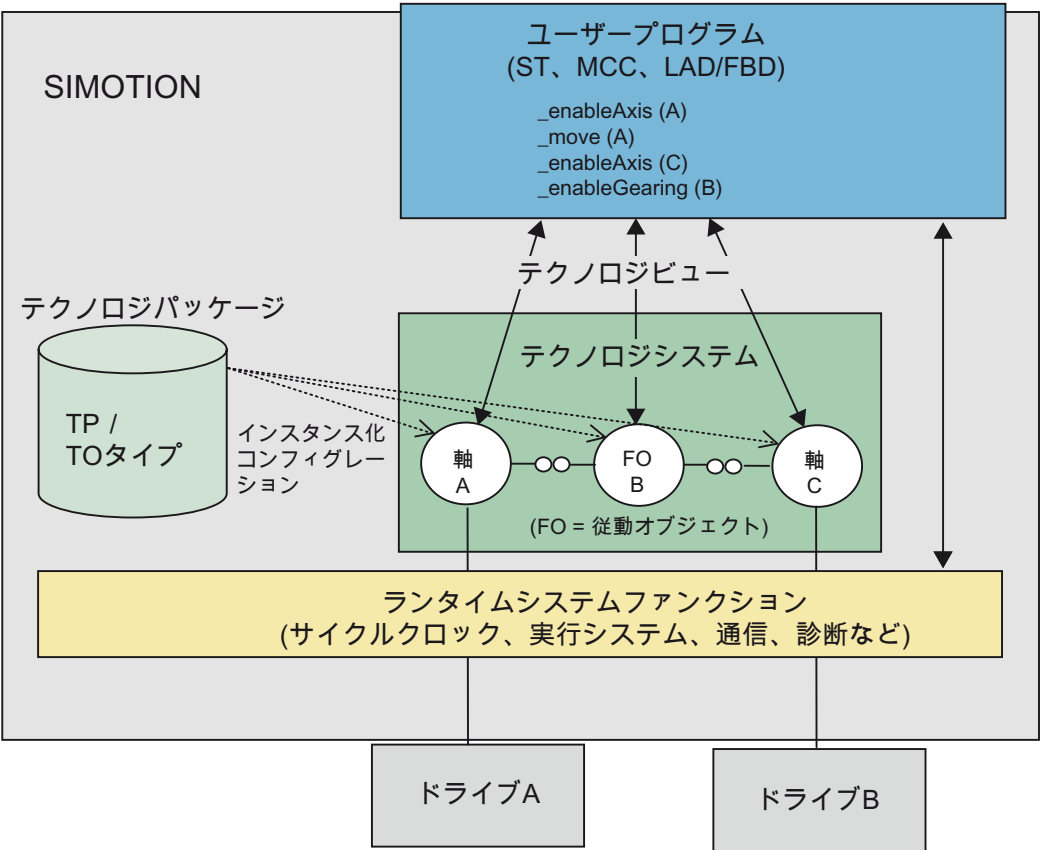


図 2-6 テクノロジーオブジェクト間の相互接続
例: マスター軸 - 従動オブジェクト - スレーブ軸

インターフェースのタイプによっては、相互接続インターフェースで、異なるデータを周期モード(タイプモーションのモーションベクトルのs、v、aなど、周期的モーションデータ)および初期化中(モジュール情報、単位など)に交換できます。

暗黙的な相互接続

相互接続が明らかに必須である場合、相互接続はSIMOTION SCOUTエンジニアリングシステムにより暗黙的に作成されます。たとえば、測定入力と軸、カムと軸、および同期操作と軸の相互接続が作成されます。
対応するTOタイプは、軸の下で提供されます。

テクノロジー相互接続画面フォームを使用した相互接続

追加的な相互接続のために、以下のような個別の画面フォームが提供されています。

- 従動軸とマスター軸 / マスター値
- 従動オブジェクトとカム
- プロファイル入力とカム

汎用相互接続画面フォームを使用した相互接続(エキスパート専用)

特殊な相互接続のために、以下のような汎用相互接続画面フォームが提供されています。

- MotionInインターフェース相互接続(モーションベクトル用)
- トルク入力インターフェース相互接続

下記も参照

"Motion"タイプの相互接続インターフェース (ページ 58)

"LREAL"タイプの相互接続インターフェース (ページ 60)

テクノロジーオブジェクトの相互接続(エキスパート用) (ページ 52)

2.3.5 テクノロジーオブジェクトおよびDCC

説明

DCCとTO間のデータ交換は、DCCダイアグラムにより実行されます。

- ブロック入力とブロック出力をTOのシステム変数に直接相互接続する方法を使用します。
TO
axisの場合は、ユーザープログラムで毎回コマンドを発行しなくても、システム変数で指定されたデータ(オーバーライド値や設定値)を周期的に転送できます。
- または、ユーザープログラムでコマンドと変数を使用して、DCCで計算された値をTOに転送します。

TOには、特定のファンクションに関連した相互接続インターフェースがあります。

TO自体は、DCCダイアグラムでブロックとして使用することはできません。

2.3 テクノロジーオブジェクト(TO)

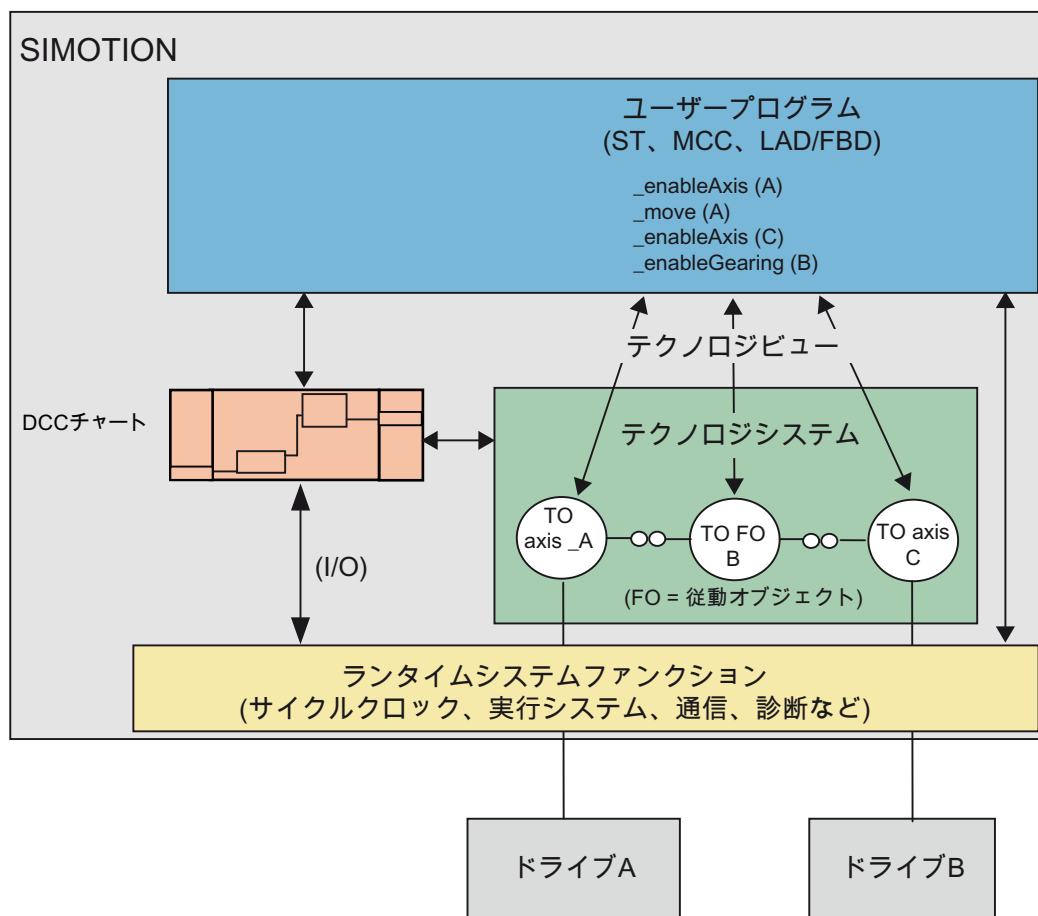


図 2-7 テクノロジーおよびDCC

下記も参照

DCCブロック(DCB)のシーケンスモデル (ページ 208)

2.3.6 使用できるテクノロジーオブジェクト

表 2-1 SIMOTIONで使用できるテクノロジーオブジェクトの概要

テクノロジーオブジェクト	シンボル	説明
軸		さまざまなバージョンの軸が使用できます。以下の軸があります。 • 実軸または仮想軸 • 位置決め軸またはドライブ軸 • 電気軸または油圧軸 • 基準軸または制御対象の推力 / 圧力 • モジュール軸 • 従動軸 • パス軸 詳細については、『Motion Control, TO axis, Electrical/Hydraulic, external encoder』機能マニュアルを参照してください。
従動オブジェクト		同期操作テクノロジーで軸を作成すると、軸の下に従動オブジェクトが作成されます。 同期操作作用の設定は、従動オブジェクトに格納されます。 詳細については、『Motion Control TO followingObject, TO cam』機能マニュアルを参照してください。
パスオブジェクト (V4.1以降)		パスオブジェクトテクノロジーオブジェクトを使用すると、パス補間を設定できます。 詳細については、『Motion Control Path Interpolation』機能マニュアルを参照してください。
測定入力		測定入力は、実際の位置を高速かつ正確に測定するために使用します。 詳細については、『Motion Control TO outputCam, TO measuringInput』機能マニュアルを参照してください。
出力カム		TO outputCamでは、位置依存の切り替え信号を作成して、位置決め軸、従動軸、または外部エンコーダに割り当てることができます。 詳細については、『Motion Control TO outputCam, TO measuringInput』機能マニュアルを参照してください。
カムトラック(camTrackType) (V3.2以降)		カムトラックは、複数の出力カムをグループ化してテクノロジーオブジェクトを形成するのに使用します。 詳細については、『Motion Control TO outputCam, TO measuringInput』機能マニュアルを参照してください。
外部エンコーダ		TO External Encoderでは、軸がない外部エンコーダ(プレス上のシャフトエンコーダなど)を接続する機能をシステム内で提供します。 詳細については、『Motion Control, TO axis, Electrical/Hydraulic, external encoder』機能マニュアルを参照してください。
カム		TO camを使用すると、転送機能を定義して他のテクノロジーオブジェクトで割り当てることができます。 詳細については、『Motion Control TO followingObject, TO cam』機能マニュアルを参照してください。

2.3 テクノロジーオブジェクト(TO)

テクノロジーオブジェクト	シンボル	説明
温度チャンネル		TO temperatureChannelを使用すると、SIMOTIONで温度制御を設定できます。 詳細については、『 Motion Control Additional Technology Objects 』機能マニュアルを参照してください。
追加のテクノロジーオブジェクト(エキスパート用) これらのテクノロジーオブジェクトの利点は、他のTOのようにシステムレベルで実行されることです。 相互接続された信号は、システムサイクルクロック内で、テクノロジーオブジェクトにおいて直接処理されます。 対照的に、構造化テキストの応用ソリューションでは、システムとアプリケーション間での変更が頻繁に起こり、むだ時間が発生します。追加のテクノロジーオブジェクトは、たとえば巻上げ機アプリケーションなどの実装に使用されます。		
固定ギア (V3.2以降)		TO fixedGearを使用すると、指定されたギア比を用いた固定同期操作(同期化 / 非同期化がない)を実装できます。 詳細については、『 Motion Control Additional Technology Objects 』機能マニュアルを参照してください。
追加オブジェクト (V3.2以降)		追加オブジェクトを使用すると、最大4つの入力ベクトルを追加して出力ベクトルを生成できます。 詳細については、『 Motion Control Additional Technology Objects 』機能マニュアルを参照してください。
数式オブジェクト (V3.2以降)		数式オブジェクトでは、Motionタイプのスカラー(LREAL、DINT)またはモーションベクトルで数学演算を使用できます。 詳細については、『 Additional Technology Objects 』機能マニュアルを参照してください。
センサ (V3.2以降)		TO sensorを使用すると、スカラーの測定値を記録できます。 詳細については、『 Motion Control Additional Technology Objects 』機能マニュアルを参照してください。
コントローラオブジェクト (V3.2以降)		コントローラオブジェクトは、スカラー変数の準備と制御に使用できます。 詳細については、『 Motion Control Additional Technology Objects 』機能マニュアルを参照してください。

2.4 SIMOTIONのメモリコンセプト(対象システム)

2.4.1 対象システムでのストレージコンセプト

メモリのタイプ

SIMOTIONでは、スタガ型のメモリ管理コンセプトを採用しています。

メインメモリは**RAM**です。

データは、ダウンロード中にRAMに書き込まれ、アップロード中にRAMから読み取られます。**ROM**は、恒久的な保管に使用されます(電源切断後も保持されます)。

対象デバイスがシャットダウンされた後も、プロジェクトデータは引き続きROMに保存されます。

対象デバイスが起動すると、データは**ROM**から**RAM**にコピーされ、さらにRAMから作業メモリである**カレントデータメモリ**にコピーされます。

RAMの内容は、RUNモードへの切り替え中に is copied to the **カレントデータメモリ**にコピーされます。このメモリは、RUNモードで使用されます。

RUNモードでのメモリ管理

システム変数またはコンフィグレーションデータの値を変更すると、これらのシステム変数の値は、**カレントデータメモリ**内で直ちに有効になります。

コンフィグレーションデータの新しい値は、最初に**ネクストメモリ**にバッファされます。

コンフィグレーションデータは直ちに有効になり、**カレントデータメモリ**に取り込まれます。**RESTART**

(システム変数restartactivationの値を**ACTIVATE_RESTART**に設定)の後にのみテクノロジーオブジェクト上で有効になるコンフィグレーションデータは、**RESTART**の後にのみ**カレントデータメモリ**に書き込まれます。

試運転段階

The **カレントデータメモリ**は、機械が本稼働中の場合に機能します。

試運転段階では、**実際のメモリ**に格納されている変数値が、**RAMメモリ**にある値と異なる場合があります(たとえば、RUNモードでコンフィグレーションデータを変更すると、その変更は**カレントデータメモリ**のみに反映されます)。

試運転段階が完了した後は(すべてのコンフィグレーションデータが要求どおりに設定された場合)、**[Target system | Copy current data to**

RAM]コマンドを使用して**カレントデータメモリ**の内容を**RAM**に取り込む必要があります。このコピー操作によってのみ、現在有効なコンフィグレーションデータは**RAM**に転送されます。

変更したコンフィグレーションデータの**RAM**への転送が完了すると、SCOUTの設定は対象システムの設定と一致しなくなります。これは、対象システムに修正したコンフィグレーションデータが格納されているからです。メニューコマンド**[Target system | Load | Configuration data to**

PG]を使用して、**RAM**からデータを読み込むことで、一貫性のあるシステム状態を回復する必要があります。

また、コンフィグレーションデータを停電から確実に保護するために、メニューコマンド**[Target system | Copy RAM to ROM]**を実行する必要があります。

注記

[Copy current to

RAM]メニューコマンドでは、システム変数の値はRAMメモリへ転送されません。したがって、"メモリカードへの保存(RAMからROM)"または"エンジニアリングプロジェクトへの保存(コンフィグレーションデータのPGへのロード)"はできません。

システム変数の値もエンジニアリングプロジェクトとメモリカードに保存できるようにするには、システム変数の値をOFFLINEモードで変更してから、ダウンロードと保存により対象システムにロードする必要があります。

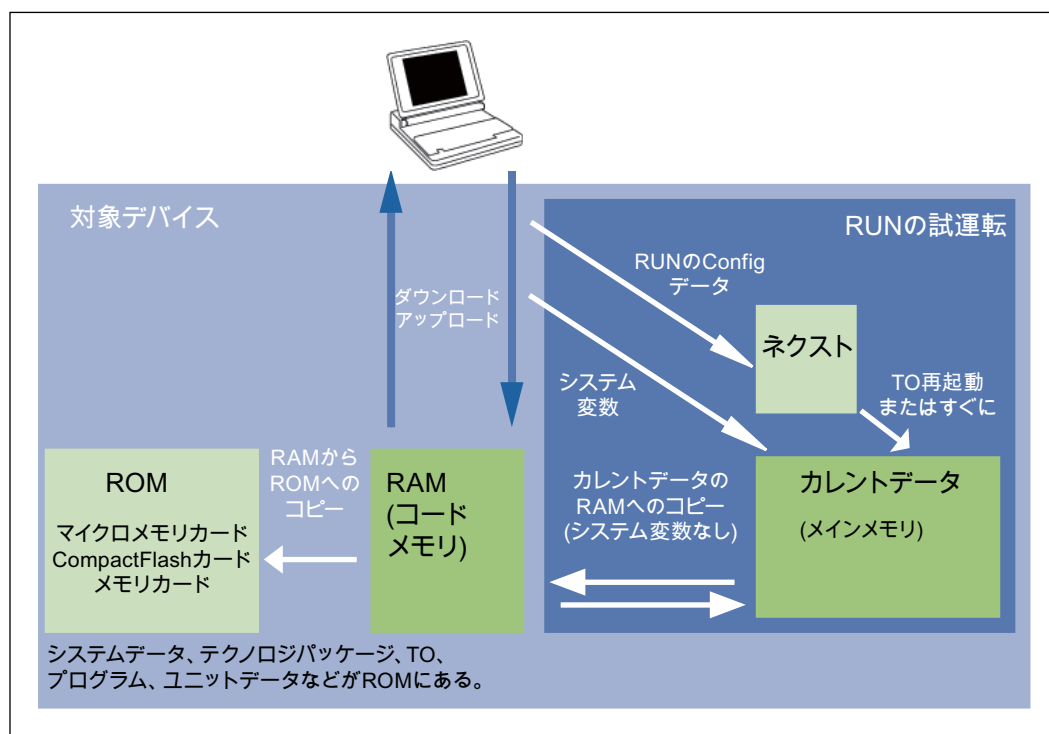


図 2-8 メモリコンセプト(全般的な説明)

以下に示すメモリ領域は、特定の動作により影響を受けます。

- 電源投入および全初期化

データ(コンフィグレーションデータ、システム変数のデフォルト値)はROM (メモリカード)からRAMにロードされ、その後RAMから実際のメモリにロードされます。

- テクノロジーオブジェクトの再開

TOのデータ(コンフィグレーションデータ、システム変数のデフォルト値、カム)はROM (メモリカード)からRAMにロードされ、その後RAMから実際のメモリにロードされます。

2.5 エキスパートリスト

ウィザードおよびパラメータ設定画面フォームによりコンフィグレーションデータとシステム変数へのアクセスに加えて、エキスパートリストを使用して直接アクセスするオプションもあります。

すべての重要なコンフィグレーションデータとシステム変数のパラメータはダイアログボックスを使用して設定できるので、エキスパートリストが必要になるのは例外的なケースです(コンフィグレーションデータのオンラインでの修正など)。

V4.1以降には、呼び出される最も重要なコンフィグレーションデータとシステム変数(メインパラメータ)を含むエキスパートリストがあります。

これらのリストには、軸、従動オブジェクト、外部エンコーダのプログラミングと診断に使用される最も重要なコンフィグレーションデータおよびシステム変数が記述されています。

注記

エキスパートリストのパラメータを修正するには、システムに関する特別な知識が必要です。

- エキスパートリストを使用して作成したエントリは、ウィザードおよびパラメータ設定画面フォームを呼び出して上書きすることができます。
- 他のパラメータとの依存関係に関するチェックは行われません。

エキスパートリストの呼び出し

1. プロジェクトナビゲータで、必要なテクノロジーオブジェクトを強調表示にします。
2. オブジェクトコンテキストメニューで[Expert | Expert List]を選択するか、またはキーボードでCtrl + Eキーを押します(たとえば、軸のダイアログボックスが開いた場合)。

2.5.1 コンフィグレーションデータおよびシステム変数のエキスパートリスト

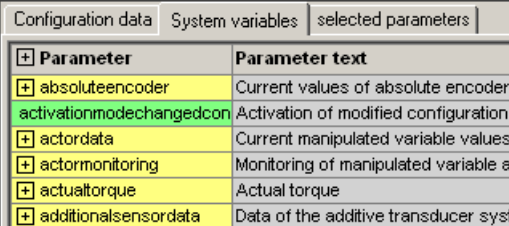
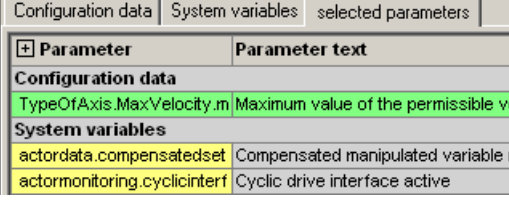
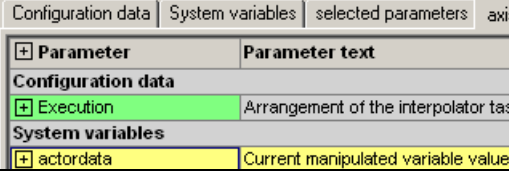
テクノロジーオブジェクトに関するシステム変数とコンフィグレーションデータは、エキスパートリストの最初の2つのタブに表示されます。[Selected Parameters]タブでは、ユーザー定義リストが保存および個別に調整されます。

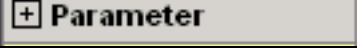
コンフィグレーションデータの値は、OFFLINEモードおよびRUNモードの間はONLINEモードで変更できます。

書き込みができるパラメータは緑色で表示され、書き込みができないパラメータは黄色で表示されます。グレー表示の値は変更できません。

表 2-2 エキスパートリストには、以下の情報が表示されます。

ボタン / テーブル列	意味 / 注
[Configuration data] Configuration data System variables selected parameters Parameter Parameter text Execution LeadScrew Modulo Restart Arrangement of the interpolator ta Leadscrew pitch of a linear axis Modulo settings of a rotary axis Modes for the restart of the techn	[Configuration data] data]タブには、TOのコンフィグレーションデータがアルファベット順に一覧表示されます。 通常、ユーザーはすべてのコンフィグレーションデータ(緑色で表示)を修正できます。

ボタン / テーブル列	意味 / 注
[System variables] 	[System Variables] タブには、TOのコンフィグレーションデータがアルファベット順に一覧表示されます。 書き込みができるシステム変数は緑色で表示され、書き込みができないシステム変数は黄色で表示されます。
[selected parameters] 	[selected parameters]タブでは、TOの最も重要なシステム変数とコンフィグレーションデータが、システム変数とコンフィグレーションデータに分けて、アルファベット順で一覧表示されます。 選択されているのは工場出荷時の設定なので、変更はできません。ただし、リストは閉じることができ、次回呼び出したときに再表示されます。
ユーザ定義パラメータリスト 	ユーザー定義リストからのコンフィグレーションデータまたはシステム変数は、追加のタブに表示されます。 コンフィグレーションデータとシステム変数は、複数のブロックに分割されます。
[Create new tab] 	ユーザー定義リストを新規に作成、または既存のユーザー定義リストやデフォルトパラメータリストを開く必要がある場合は、このボタンをクリックします。
[Quick search] 	このテキストフィールドでは、エキスパートリスト内のクイック検索を実行できます。[Parameter]、[Parameter text]、[Value]の各列が検索されます。
[Collect changes] 	このボタンが有効になっている間は、ユーザーにより変更(RUNモードで)されたコンフィグレーションデータはすべて収集されますが、新しい値は対象システムでは有効になりません。[Activate changes]がクリックされた場合にのみ、コンフィグレーションデータに対するすべての変更が対象システムで有効になります。
[Activate changes] 	収集されたすべてのコンフィグレーションデータ変更を対象システムで有効にする場合は、このボタンをクリックします。
コンフィグレーション選択リスト	リニア軸または回転軸のパラメータが、エキスパートリストのテーブル内で基準軸、推力、圧力のいずれに関連して表示または変更されるかを選択します。
[New user-defined list] 	このボタンをクリックすると、新規のユーザー定義リストが作成されます。
[Open user-defined list] 	このボタンをクリックすると、既存のユーザー定義リストが開きます。
[Save user-defined list] 	このボタンをクリックすると、現在開いているユーザー定義リストが保存されます。
[Edit user-defined list] 	このボタンをクリックすると、現在開いているユーザー定義リストの編集モードにアクセスできます。

ボタン / テーブル列		意味 / 注
[Cancel editing mode] 		このボタンをクリックすると、編集モードがキャンセルされます。変更は破棄されます。
[Enter comment lines] (編集モード) 		このボタンをクリックすると、新規に作成した行のフォーマットを"コメント"に変更できます。
[Enter headlines] (編集モード) 		このボタンをクリックすると、新規に作成した行のフォーマットを"見出し"に変更できます。
[Enter parameter lines] (編集モード) 		このボタンをクリックすると、新規に作成した行のフォーマットを"パラメータ"に変更できます。
[Parameter] 		ここには、コンフィグレーションデータ項目またはシステム変数の名前が表示されます。 [+]記号をクリックすると、全体の構造が表示されます。
[Parameter text]		システム変数またはコンフィグレーションデータの簡単な説明が表示されます。
[Offline value]		ここには、コンフィグレーションデータ項目またはシステム変数の値が表示されます。パラメータのタイプに応じて、値を数値で直接入力すること、または選択リストから記号識別子を選択することができます。
[Configured value]	(ONLINEモードのみ、コンフィグレーションデータ)	ここには、コンフィグレーションデータ項目の設定済み値が表示されます。この値は、対象デバイスのRAMに保存されます。
[Current value]	(ONLINEモードのみ)	ここには、システム変数またはコンフィグレーションデータの現在値が表示されます。 システム変数の値は、ONLINEモードで直接変更できます。 コンフィグレーションデータの変更は、[Next value]列でのみ行うことができます。 現在値はカレントデータメモリに格納されます。 これらの値をRAMに転送するには、最初にメニューで[Target system Copy current data to RAM]を選択する必要があります。 注: ONLINEモードで変更されたシステム変数は、RAMにコピーできません。 したがって、メモ리카ードへの保存(RAMからROM)または"エンジニアリングプロジェクトへの保存(コンフィグレーションデータのPGへのロード)はできません。 「SIMOTIONのメモリコンセプト」を参照してください。
[Next value]	(ONLINEモードのみ、コンフィグレーションデータ)	ここでは、コンフィグレーションデータ項目の次の値をONLINEモードで入力できます。 その後、コンフィグレーションデータ項目が有効になった場合に、この値は[Current value] (現在値)の値として取り込まれます。 直後にENTERキーを押すか、または値を選択すると、対象システムで新しい値が直ちに有効になります。
[Units]		システム変数またはコンフィグレーションデータの単位が表示されます。
[Effectiveness]	(ONLINEモードのみ、コンフィグレーションデータ)	これは、変更したコンフィグレーションデータ項目の値が有効になった場合に表示されます(たとえば、再起動により)。対象システムでは、変更した値は再起動後に有効になります。 特定のコンフィグレーションデータは、ONLINEモードでは変更できません。

ボタン / テーブル列	意味 / 注
[Data type]	ここには、システム変数またはコンフィグレーションデータ項目のデータタイプが表示されます(たとえば、整数値に対してはINT)。
[Minimum]	システム変数またはコンフィグレーションデータ項目の最小値が表示されます。
[Maximum]	システム変数またはコンフィグレーションデータ項目の最大値が表示されます。

下記も参照

対象システムでのストレージコンセプト (ページ 41)

2.5.2 エキスパートリストの使用

システム変数およびコンフィグレーションデータのヘルプの表示

1. [Help | Context-sensitive help]を選択するか、またはShift + F1キーを押します。
カーソルが疑問符の形に変わります。
2. エキスパートリスト内で、ヘルプを表示する必要があるコンフィグレーションデータ項目またはシステム変数をクリックします。
関連するヘルプが表示されます。

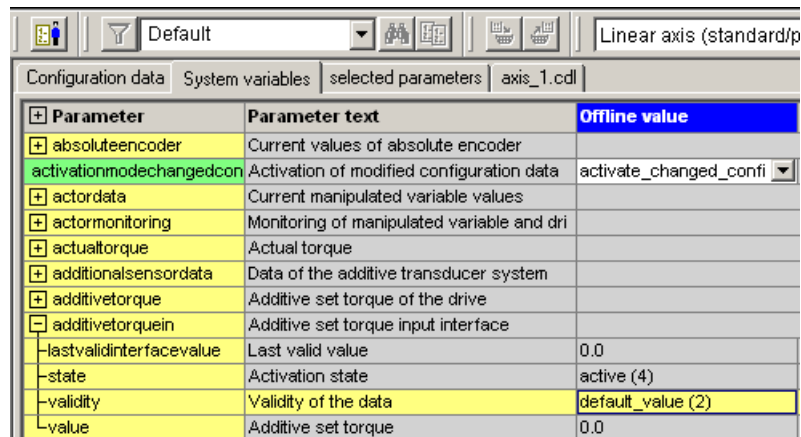
エキスパートリストにあるデータのオフラインでの変更

1. エキスパートリストで、[Configuration data]タブまたは[System variables]タブを選択します。
タブの全体が表示されます。
2. エキスパートリストの[Parameter]列で、エントリの左側にある[+]記号をクリックしてサブエントリを表示します。
黄色で強調表示されているエントリは変更できません。
3. 変更するエントリの[Offline value]列をクリックします。
4. 値を直接入力するか、または選択リストが表示された場合は値を選択します。

クイック検索の実行

検索語を入力すると、カーソルは検索語が見つかった最初の行に移動します。

1. 検索語を入力してENTERキーを押します。



Parameter	Parameter text	Offline value
absoluteencoder	Current values of absolute encoder	
activationmodechangedcon	Activation of modified configuration data	activate_changed_confi
actordata	Current manipulated variable values	
actormonitoring	Monitoring of manipulated variable and dri	
actualtorque	Actual torque	
additionalsensordata	Data of the additive transducer system	
additivatorque	Additive set torque of the drive	
additivatorquein	Additive set torque input interface	
-lastvalidinterfacevalue	Last valid value	0.0
-state	Activation state	active (4)
-validity	Validity of the data	default_value (2)
-value	Additive set torque	0.0

図 2-9 クイック検索の例

検索語が書き込み可能なパラメータである場合、カーソルは値フィールドに直接移動します。

検索された構造シエルに別のパラメータ値がなければ、カーソルは検索語が見つかったセルに直接移動します。

連続検索

1. F3キーを押します。

検索語を含む次のセルが検索されます。

検索機能がリスト内でそれ以上のエントリを発見できなかった場合は、メッセージが出力されます。

エキスパートリスト内での矢印キーによる移動

エキスパートリスト内の各セルは、矢印キーを使用して選択できます。

1. 矢印キーを押します。

隣りにあるセルが選択されます。

2. Ctrl +

矢印キーショートカットを使用すると、カーソルは各行や列の先頭または最後に移動します。

キーボードによるセルの選択

非表示領域のセルを含むエキスパートリスト全体を選択する場合

1. Ctrl + Aキーを押します。

オンラインモードでこのショートカットを使用すると、すべてのパラメータ値の(V4.0時点のテクノロジーオブジェクトを使用した)読み取り処理が開始されます。

以下のように、個々のセル範囲は、矢印キーまたはマウスを使用した移動により選択できます。

1. セルは、Shiftキーを押しながらカーソルを移動すると選択されます。

以下のように、セル範囲はマウスを使用しても選択できます。

1. カーソルを行に置き、マウスの左ボタンを押しながらマウスポインタを選択するセル範囲をドラッグします。

エキスパートリスト内でのセルの内容のコピー

選択した行は、Windowsのクリップボード経由で他のWindowsアプリケーションにコピーできます。

1. **Ctrl + C**キーを押すか、またはコンテキストメニューで[Copy to clipboard]を選択します。

たとえば、ExcelまたはWordドキュメントで、クリップボードからセルの内容を挿入できます。

コンテキストメニュー項目[Copy parameter path to clipboard]を使用すると、完全な構造パスをクリップボードにコピーしてから、他のSCOUTアプリケーション(STプログラム、監視テーブルなど)にコピーできます。

[Copy cell content to clipboard]を使用して、エキスパートリストのセルの内容をクリップボードにコピーします。

ユーザー定義パラメータリストの作成

1. [Create new tab]ボタンをクリックします。



以下の選択ダイアログボックスが表示されます。

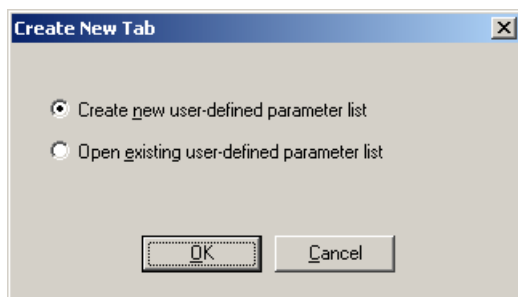


図 2-10 ユーザー定義パラメータリストの選択ダイアログボックス

2. [Create new user-defined parameter list]を選択し、[OK]をクリックします。

新規に作成されたリストが含まれる追加のタブが表示されます。

このタブには、最初はコンフィグレーションデータとシステム変数の2つに分けるセルのみが含まれています。

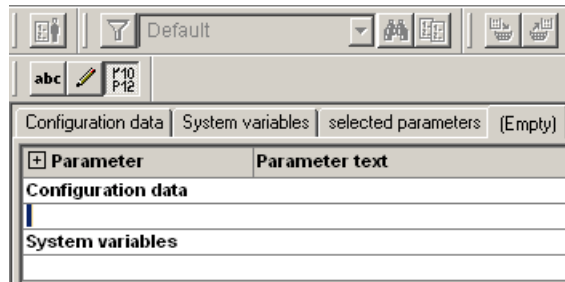


図 2-11 新規のユーザー定義パラメータリスト

3. コピーペースト(コンテキストメニューまたはCtrl + CとCtrl + V)を使用して、目的のパラメータを他のタブから転送します。

例:

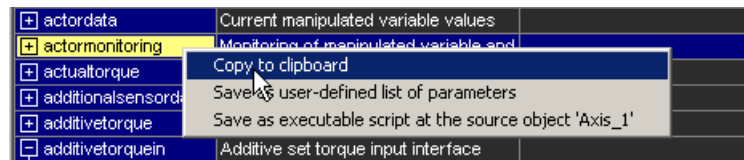


図 2-12 パラメータのコピー

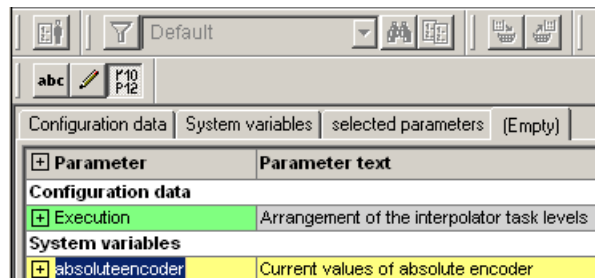


図 2-13 ユーザ定義パラメータリスト

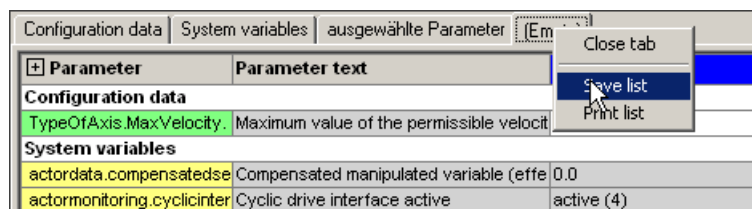


図 2-14 ユーザー定義パラメータリストの保存または印刷

4. リストを保存するには、タブのコンテキストメニューで[Save]ボタン  [Save list]を選択し、[OK]をクリックして確定します。

保存したリストの名前は、タブのラベルとして使用されます。

Configuration data	System variables	ausgewählte Parameter	test.cdl
Parameter	Parameter text	Offline value	
Configuration data			
TypeOfAxis.MaxVelocity	Maximum value of the permissible velocity	500.0	
System variables			
actordata.compensatedse	Compensated manipulated variable (effe	0.0	

図 2-15 保存されたユーザー定義パラメータリスト

5. リストを印刷するには、タブのコンテキストメニューで[Print list]を選択します。

パラメータリストからの実行可能スクリプトの作成

バージョンV4.0以降では、エクスポートリストで、パラメータセットをスクリプトフォーム内の一連の書き込み / 読み取り要求として保存できます。

1. タブのコンテキストメニューで、[Save list]を選択します。

以下の選択ダイアログボックスが表示されます。

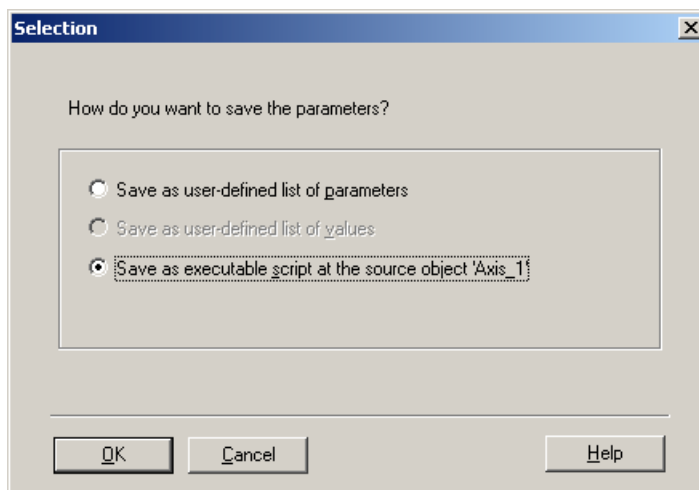


図 2-16 リストを保存するための選択ダイアログボックス

2. 表示された選択ダイアログボックスで、[Save as executable script ...]オプションを選択し、[OK]をクリックして確定します。
3. [Add script]ダイアログボックスで、スクリプトの名前を入力します。
[OK]をクリックして確定します。

スクリプトは、オブジェクトの下にある[SCRIPTS]フォルダに保存されます。スクリプトフォルダが存在しない場合は、フォルダが作成されます。

オプションとして、スクリプトをASCIIテキストとしてエクスポートすることもできます。

1. [Save as]ダイアログボックスで、テキストファイルの保存場所と名前を選択できます。次に、[OK]をクリックして操作を確定するか、またはスクリプトをエクスポートしない場合は[Cancel]をクリックします。

スクリプトの構造に関する留意点

- 変換された各スクリプトは、日付と作成元を含んだ見出しを受け取り、次に空白行を受け取ります。
- スクリプトエンジンのロギングメカニズムは、その次の行で有効になります。この結果、書き込みと読み取りの各動作がスクリプト出力ウィンドウに出力されます。
- ユーザ定義パラメータリスト内の見出しが、空白行に続くコメントとしてスクリプトに入力されます。
- ユーザー定義リスト内のコメントは、空白行は追加されずに単純なコメントとしてスクリプトに入力されます。
- ユーザ定義パラメータリストの各書き込みパラメータに対しては、書き込み要求がスクリプトに入力されます。番号と列挙値は、整数値として書き込まれます。パラメータのパラメータテキストが、書き込み要求の右隣りにコメントとして追加されます。
- ユーザー定義パラメータリストの読み取り専用パラメータに対しては、値 = (パラメータ(x, y))の形式で割り当てが入力されます。
この結果、パラメータは読み取られ、値は[Scripting]出力ウィンドウに出力されます。

スクリプトの使用例

- 変更されたユニットシステムがある場合、スクリプトを実行する際にそのユニットシステムは使用されません。
受け取ったパラメータ値が現在のコンテキストで適切であるかは、ユーザー側で確認する必要があります。
- 構造を変更するパラメータ(TypeOfAxis.typeOfAxisなど)に対しても、処理は変わりません。
これらのパラメータがパラメータ選択の最上位にあることを、ユーザー側で確認する必要があります。
- 現在値のみが書き込まれるので、スクリプトにコンフィグレーションデータが含まれていると、スクリプトが実行されないことがあります。

用途:

スクリプトを使用して、パラメータの割り当てを1つの軸から別の軸に転送できます。

エキスパートリストのシンボルブラウザへの表示

プロジェクトナビゲータでTOを選択すると、関連するエキスパートリストがシンボルブラウザに自動的に表示されます。

ただし、このリストの機能範囲は限定されているので、たとえば、ユーザー定義リストの表示はできませんが、リストの作成はできません。

2.6 テクノロジーオブジェクトの相互接続(エキスパート用)

通常の暗黙的な相互接続およびテクノロジー相互接続画面フォームによる相互接続とは別に、汎用相互接続画面フォームを使用してシステムレベルでテクノロジーオブジェクトを他のテクノロジーオブジェクトに相互接続することもできます。

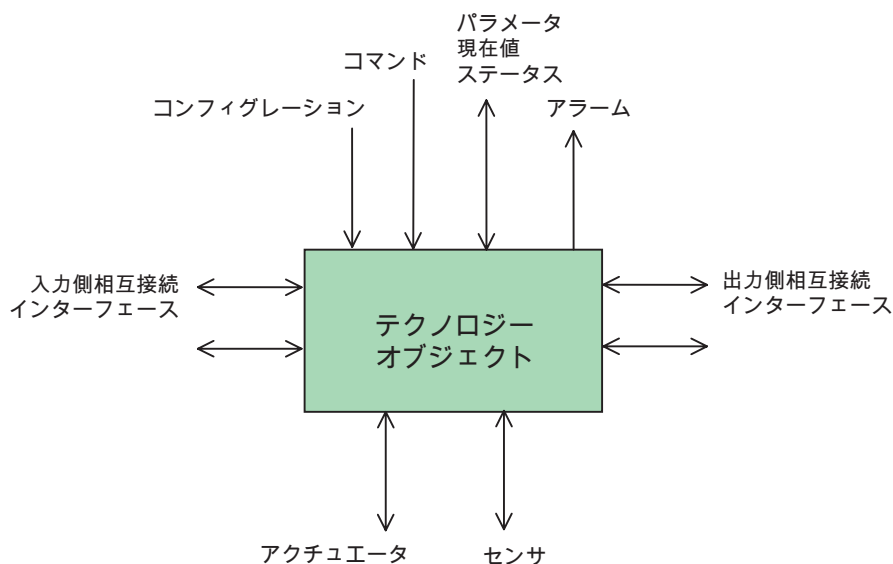


図 2-17 テクノロジーオブジェクトの相互接続インターフェース

2.6.1 SCOUTの汎用相互接続画面フォームによる相互接続

異なるテクノロジーオブジェクトを相互接続するために、SCOUTでは、以下のテクノロジーオブジェクトで利用できる**相互接続表示**を提供しています。

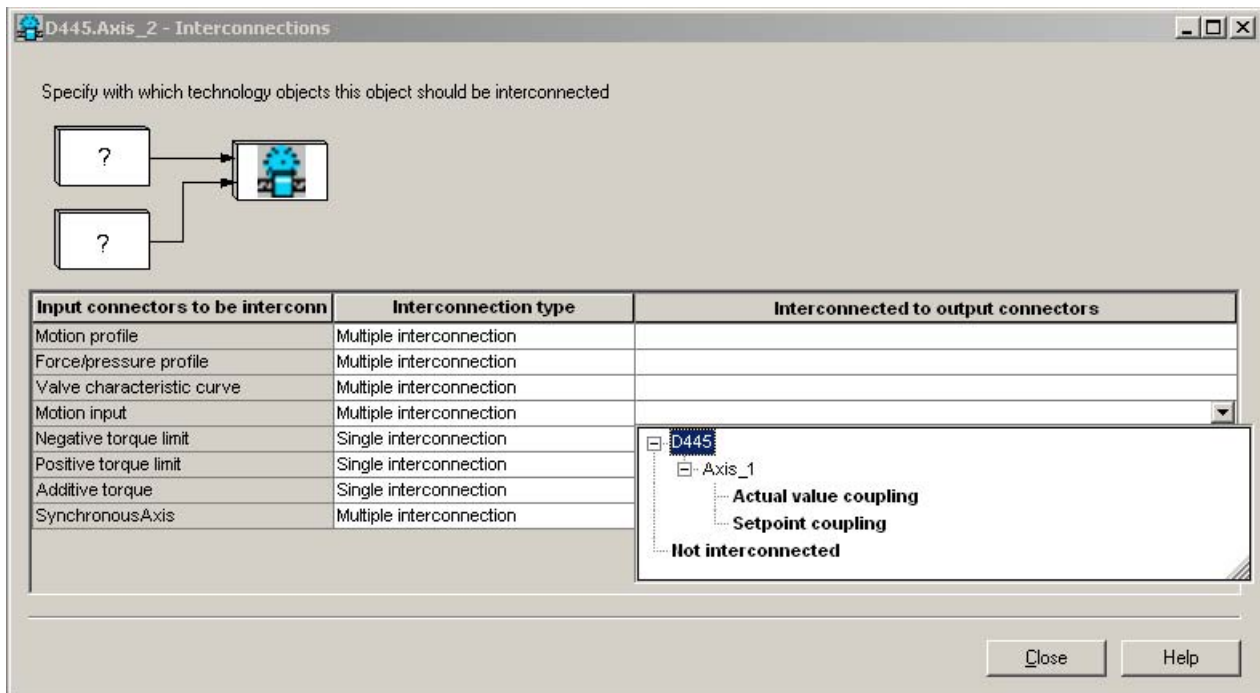


図 2-18 SCOUTでの相互接続 - 軸の例

左側には、選択したオブジェクトの入力側にあるすべての相互接続インターフェースが一覧表示され、右側には、出力側にある同じタイプの相互接続インターフェースが一覧表示されます。

この一覧では、太字で表示されている出力側のインターフェースのみが選択できます。

- これらの行の1つを強調表示(クリック)にするとツリーが開き、可能性のあるすべての相互接続が表示されます。

このツリーでは、太字で表示されている要素のみが選択できます。

デバイスとテクノロジーオブジェクトの表示は、確認の目的でのみ使用されます。

- 直接入力、入力行で行います。

完全に正しい入力のみが受け付けられます。

入力が誤りがある場合は、入力フィールドを終了できません。

入力側相互接続インターフェースに複数の相互接続に対応する機能があり、出力側の複数の相互接続インターフェースと相互接続できる場合、出力側相互接続インターフェースとの間で1つの相互接続が確立されると、入力側相互接続インターフェースに対して別の行が追加されます。

- この行を消去すると、相互接続は削除されます。

入力行が終了すると、値は取り込まれます。

2.6.2 相互接続インターフェースの一般的な特性

相互接続インターフェースの設定

入力側相互接続インターフェースは、以下のように作成されます。

- TOのインスタンス化にともない暗示的に作成
- 各テクノロジーに固有の相互接続画面フォーム
- 汎用相互接続画面フォームによる

有効化 / 無効化

入力側相互接続インターフェースは、関連するTO上でコマンドを使用して有効にするか、またはデフォルトで有効になっています。

入力側相互接続インターフェースが複数の相互接続に対応している場合は、以下の例のように、テクノロジカル機能用のコマンドを使用して有効化 / 無効化を実行します。

- `_enableCamming()` コマンドで `cam` パラメータを指定してカムを選択
- `_setMaster()` コマンドで `master` パラメータを指定して、マスター値用のテクノロジーオブジェクトを選択
- `_run...basedMotionIn...()` コマンドで `mastertype.reference` パラメータを指定して、MotionIn インターフェースの基準値用のテクノロジーオブジェクトを選択
- `_run...Profile()` コマンドで `profile` パラメータを指定して、プロファイルまたはカムを選択

このトピックの詳細については、各テクノロジーオブジェクトの機能マニュアルを参照してください。

ステータスおよび相互接続値のインジケータ

入力側相互接続のステータスおよび相互接続値に対するインジケータは、個別のインターフェイスタイプに対応するTO上で定義されます。

このトピックの詳細については、各テクノロジーオブジェクトの機能マニュアルを参照してください。

有効性

相互接続が確立されて相互接続値のステータスが"有効"になっている場合、相互接続インターフェースは有効な値を保持しています。なお、相互接続 / 相互接続値が不正な場合、ステータスの値は0になります。

軸(仮想軸)がシミュレーションに設定されると、出力側の設定値は引き続き更新されますが、現在値は一時的に無効にされます。

システム変数で利用できるステータスを読み取ると、ランプアップ後の相互接続値または置換値が有効かどうかを確認できます。

アラーム応答

相互接続に障害が発生している場合、相互接続上でファンクションが有効になると、テクノロジーアラームのみが発行されます。

入力側相互接続インターフェースが複数の相互接続に対応している場合は、参照TOを選択する必要があります。

プログラムされたファンクションはSTOPUからSTOPへ移行するときにキャンセルされますが、相互接続ファンクションはSTOPUからSTOPへ移行しても引き続き有効です。

2.6.3 TO上の相互接続インターフェース

入力側相互接続インターフェースは、同じタイプの出力側相互接続インターフェースにのみ接続できます。

TO	インターフェース		タイプ
ドライブ軸(driveAxis)			
	入力側相互接続インターフェース		
		運動入力	Motion
		トルク制限(正側)	LREAL
		トルク制限(負側)	LREAL
		追加トルク	LREAL
		動作プロファイル	MotionProfile
		推力 / 圧力プロファイル	ForceProfile
		バルブ特性(油圧軸)	ValveCharacteristics
	出力側相互接続インターフェース		
		動作の設定値	Motion
		動作の外挿による現在値	Motion
		実トルク	LREAL
位置決め軸			
	入力側相互接続インターフェース		
		ドライブ軸用として	
	出力側相互接続インターフェース		
		ドライブ軸用および追加機能として	
		出力カム / カムトラック用インターフェース (出力カムまたはカムトラックを作成する際に、システムにより暗黙的に軸に相互接続されます)	固有
		測定入力用インターフェース (測定入力を作成する際に、システムにより暗黙的に軸に相互接続されます)	固有
従動軸			
	入力側相互接続インターフェース		
		位置決め軸用および追加機能として	
		同期オブジェクト用のインターフェース (従動軸 / 従動オブジェクトを作成する際に、システムにより暗黙的に相互接続されます)	固有

	出力側相互接続インターフェース	
	位置決め軸用および追加機能として	
パス軸		
	入力側相互接続インターフェース	
	従動軸用インターフェースおよび追加機能	
	パス動作	固有
	出力側相互接続インターフェース	
	位置決め軸用として	
従動オブジェクト		
	入力側相互接続インターフェース	
	運動入力	Motion
	カム	CAM
	出力側相互接続インターフェース	
	スレーブ軸用インターフェイス (従動軸 / 従動オブジェクトを作成する際に、システムにより暗黙的に相互接続されます)	固有
パスオブジェクト		
	入力側相互接続インターフェース	
	速度プロファイル	MotionProfile
	出力側相互接続インターフェース	
	パス動作(パス軸1)	固有
	パス動作(パス軸2)	固有
	パス動作(パス軸3)	固有
	パス同期動作	固有
	運動出力(パスオブジェクト.x)	Motion
	運動出力(パスオブジェクト.y)	Motion
	運動出力(パスオブジェクト.z)	Motion
外部エンコーダ		
	入力側相互接続インターフェース	
	なし	
	出力側相互接続インターフェース	
	出力カム / カムトラック用インターフェース (出力カムまたはカムトラックを作成する際に、システムにより暗黙的に相互接続されます)	固有
	測定入力用インターフェース (測定入力を作成する際に、システムにより暗黙的に相互接続されます)	固有
	動作の設定値	Motion
	動作の外挿による現在値	Motion
測定入力		
	入力側相互接続インターフェース	
	測定入力インターフェース (測定入力を作成する際に、システムにより暗黙的に軸または外部エンコーダに相互接続されます)	固有

		"リスニング測定入力"用の入力インターフェース (V4.0以降) (汎用相互接続画面フォームを使用)	固有
		出力側相互接続インターフェース	
		"リスニング測定入力"用の出力インターフェース (V4.0以降)	固有
出力カム、カムトラック			
		入力側相互接続インターフェース	
		出力カムインターフェース (出力カムまたはカムトラックを作成する際に、システムにより暗黙的に軸または外部エンコーダに相互接続されます)	固有
カム			
		入力側相互接続インターフェース	
		なし	
		出力側相互接続インターフェース	
		カム	Cam
		動作プロファイル	MotionProfile
		推力 / 圧力プロファイル	ForceProfile
		バルブ特性(油圧軸)	ValveCharacteristics
温度コントローラ			
		相互接続インターフェースなし	
追加オブジェクト			
		入力側相互接続インターフェース	
		運動出力1	Motion
		運動入力2	Motion
		運動入力3	Motion
		運動入力4	Motion
		出力側相互接続インターフェース	
		運動出力	Motion
数式オブジェクト			
		入力側相互接続インターフェース	
		運動入力1	Motion
		運動入力2	Motion
		運動入力3	Motion
		LREAL入力1	LREAL
		LREAL入力2	LREAL
		LREAL入力3	LREAL
		LREAL入力4	LREAL
		DINT入力1	DINT
		DINT入力2	DINT
		DINT入力3	DINT
		DINT入力4	DINT
		出力側相互接続インターフェース	
		運動出力1	Motion
		運動出力2	Motion

		運動出力3	Motion
		LREAL入力1	LREAL
		LREAL入力2	LREAL
		LREAL入力3	LREAL
		LREAL入力4	LREAL
		DINT出力1	DINT
		DINT出力2	DINT
		DINT出力3	DINT
		DINT出力4	DINT
固定ギア			
	入力側相互接続インターフェース		
		運動入力	Motion
	出力側相互接続インターフェース		
		運動出力	Motion
センサ			
	入力側相互接続インターフェース		
		なし	
	出力側相互接続インターフェース		
		センサ値	LREAL
		センサ値の微分	LREAL
		運動出力	Motion
コントローラオブジェクト			
	入力側相互接続インターフェース		
		設定値	LREAL
		現在値	LREAL
		フィードフォワード制御	LREAL
		運動入力の設定値	Motion
		運動入力の現在値	Motion
	出力側相互接続インターフェース		
		運動出力の出力値	Motion
		出力値	LREAL

2.6.4 "Motion"タイプの相互接続インターフェース

相互接続インターフェースタイプMotionタイプの相互接続インターフェースには、運動ベクトル(s, v, a)、有効性ステータスなどの内部情報、およびモジュール情報(該当する場合)が含まれていません。

運動成分

運動ベクトルには、異なる運動成分(位置または速度)が含まれることがあります。

"運動速度"成分では、位置成分はゼロです。

ドライブ軸では、運動速度成分を持つ運動ベクトルのみを出力側で提供します。

運動成分の検討 / 設定には、コマンド / コマンドパラメータ(TO axis)または設定(TO additionObject、TO fixedGear)が使用されます。

運動ベクトルの成分の整合性は、TO axisとTO fixedGearの出力側では維持されます(位置値の微分が速度値になり、速度値の微分が加速度になります)。

TO additionObjectとTO formulaObjectの出力側では、運動ベクトルの成分の整合性は維持されません。

この要素では、速度と加速度は位置に基づいて厳密に決定されます。

"Motion"タイプの入力側相互接続インターフェース"MotionInput"

TOタイプによって、"Motion"タイプの入力側相互接続インターフェース"MotionInput"で可能な相互接続の回数が、1回または複数回のいずれになるかが決まります。

たとえば、以下のテクノロジーオブジェクトには、"Motion"タイプの入力側相互接続インターフェース"MotionInput"があります。

- ドライブ軸
- 位置決め軸
- 従動軸
- 固定ギア
- 追加オブジェクト
- 数式オブジェクト
- 従動オブジェクト(マスター値)

"Motion"タイプの出力側相互接続インターフェース

たとえば、以下のテクノロジーオブジェクトには、"Motion"タイプの出力側相互接続インターフェースがあります。

- ドライブ軸
- 位置決め軸
- 従動軸
- パスオブジェクト
- 固定ギア
- 追加オブジェクト
- 数式オブジェクト
- 外部エンコーダ

2.6.5 "LREAL"タイプの相互接続インターフェース

LREALタイプの相互接続インターフェースは、スカラー変数に対して使用できます。

軸上の相互接続インターフェース"TorqueLimitPositive"および"TorqueLimitNegative"によるトルク制限の指定

B+ / B-

トルク制限を指定するには、軸の入力側相互接続インターフェース"TorqueLimitPositive"および"TorqueLimitNegative"を、他のTOのLREALタイプの出力側相互接続インターフェースに相互接続します。

必要条件は、TO axis 上のプロセス関連フィールドを有効にすることです。

入力側相互接続インターフェース"TorqueLimitPositive"および"TorqueLimitNegative"は、複数回の相互接続には対応していません。したがって、これらのインターフェースを分散させることはできません。

たとえば、以下のテクノロジーオブジェクトには、LREALタイプの出力側相互接続インターフェースがあります。

- 出力側相互接続インターフェース"Torque"を含むTO axis
- 出力側相互接続インターフェース"LRealOut"を含むTO formulaObjectType
- センサTO
- コントローラオブジェクトTO

軸上の"Torque"出力側相互接続インターフェースによる実トルクの提供

軸上の"Torque"出力側相互接続インターフェースは、他のTOへの実トルクの発行に使用できます。

必要条件は、TO axis 上のプロセス関連フィールドを有効にすることです。

ドライブにより提供される実トルクは、追加のプロセス関連フィールド内で発行されます。

これは、軸上のユニットセット内で発行されます。

出力側相互接続インターフェースは、複数回の相互接続に対応しています。

出力側相互接続インターフェース"ActualTorque"を分散させることはできません。

軸上の入力側相互接続インターフェース"AdditiveTorque"による追加トルクの指定

LREALタイプの入力側相互接続インターフェース"AdditiveTorque"は、軸上で使用できます。このインターフェースでは、追加のプロセス関連フィールドを使用して、ドライブに対して追加トルクを指定します。

他のTOに依存する追加トルクを指定するには、入力側相互接続インターフェース"AdditiveTorque"を、軸上の他のTOにあるLREALタイプの出力側相互接続インターフェースに相互接続します。

入力側相互接続インターフェース"AdditiveTorque"は、複数回の相互接続には対応していません。したがって、このインターフェースを分散させることはできません。

たとえば、以下のテクノロジーオブジェクトには、LREALタイプの出力側相互接続インターフェースがあります。

- 出力側相互接続インターフェース"Torque"を含むTO axis

- 出力側相互接続インターフェース"LRealOut"を含むTO formulaObjectType
- センサTO
- コントローラオブジェクトTO

テクノロジーオブジェクトを使用したプログラミング

3

3.1 定義

以下のセクションでは、モーションコンポーネントについて、主にSTプログラミング言語の視点から紹介します。

これは、システムファンクションとシステム変数に密接に関連しています。
ここでの説明は、以下の定義に基づいています。

- システムファンクションは、システム管理に使用されるファンクションです。
このファンクションでは、デバイスのテクノロジーに中立な機能にアクセスできます。
システムファンクションはいつでも使用できます。
- テクノロジーオブジェクト(TO)は、SIMOTIONユーザープログラムのテクノロジーカル機能(たとえば、軸の位置決め、出力カム用パラメータの割り当て)を表しています。
- TOファンクションまたはテクノロジーコマンドは、たとえばテクノロジーオブジェクト用のファンクションなど、個々のTOに使用できる言語コマンドです。 .
- テクノロジーパッケージには、1つまたは複数のテクノロジーオブジェクトが含まれています。
- システム変数は、テクノロジーオブジェクトと基本システムの属性です。
システム変数を使用すると、テクノロジーオブジェクトと基本システム用のパラメータの割り当て、またはそれらのステータスを読み取ることができます。

注記

モーションコントロールテクノロジーおよび、特にテクノロジーオブジェクトの基本的な情報については、SIMOTIONモーションコントロールテクノロジーオブジェクトの機能マニュアルを参照してください。

テクノロジーオブジェクトの設定については、SIMOTIONモーションコントロールファンクションの説明を参照してください。

本書では、前記のトピックの概要のみを示します。

テクノロジーオブジェクトのプログラミングの詳細については、SIMOTIONモーションコントロール - テクノロジーオブジェクトの機能マニュアルに記載されています。

3.2 テクノロジーオブジェクト(TO)のプログラミング

3.2.1 テクノロジファクションのプログラムでの使用

テクノロジファクション(TOファクション)を使用するには、テクノロジパッケージを1回選択する必要があります。

これは、テクノロジーオブジェクト(TO)とTOファクションが、テクノロジパッケージに含まれているためです。

正式には、TOとTOファクションは、ファクション名、入力パラメータ、および戻り値(たいていの場合)を持つファクションです。

この後は、テクノロジーオブジェクトおよびTOファクションのコードとコンポーネントの概要について説明します。ただし、「テクノロジファクションの入力パラメータ」セクションに記載されている入力パラメータは除きます。

以下の説明は、STプログラミングの視点からのものです。

LAD/FBDとMCCのプログラミングについては、それぞれのマニュアルを参照してください。

テクノロジパッケージの選択

テクノロジパッケージの選択には、以下のコマンドを使用します。

```
USEPACKAGE tp-name AS name space
```

ここでの *tp-name* は、テクノロジパッケージの名前です(以下の表を参照)。

オプションの *namespace* アドオンASを使用すると、名前空間を定義できます。

さらに、STソースファイルにあるものと同じ名前を持つテクノロジパッケージのタイプ、変数、ファクション、およびファクションブロックにアクセスすることもできます。

表 3-1 V4.1のテクノロジパッケージ

テクノロジパッケージ	内容の説明
Cam ¹	- 基本的なモーションコントロールコマンド、位置決め、ギアリング、不連続同期操作(カム) - 外部エンコーダ、測定入力、出力カム、およびカムトラック用のコマンド
CAM_EXT ²	補助的なテクノロジーオブジェクト(固定ギア、追加オブジェクト、数式オブジェクト、センサ、コントローラオブジェクト)用のコマンド
TControl	温度コントローラ用のコマンド
¹ バージョンV3.1以降には、BasicMC、Position、およびGearテクノロジパッケージが含まれています。 USEPACKAGEコマンドでは、これらのテクノロジパッケージはCamテクノロジパッケージに置き換えられます。 ² V3.2以降で使用可能	

以下の例で、Camテクノロジパッケージの選択方法、Camテクノロジパッケージに名前空間Cam1を割り当てる方法、およびその名前空間の使用方法を示します。

表 3-2 テクノロジパッケージの選択および名前空間の使用に関する例


```
INTERFACE
    USEPACKAGE Cam AS Cam1;
    USES ST_2;
    FUNCTION function1;
END_INTERFACE

IMPLEMENTATION
    FUNCTION function1 : VOID
    VAR _INPUT
        _Axis : posAxis;
    END_VAR
    VAR
        retVal : DINT;
    END_VAR

    retVal:= Cam1._enableAxis (
        axis := axis,
        nextCommand := Cam1.WHEN_COMMAND_DONE,
        commandId := _getCommandId() );
    END_FUNCTION
END_IMPLEMENTATION
```

テクノロジーオブジェクト(TO)のコード

テクノロジーパッケージで提供されたテクノロジーオブジェクト(TO)は、SIMOTION SCOUTによりインスタンス化されます。
インスタンス化されたTOは、その名前で識別(参照)されます。

テクノロジーファンクション(TOファンクション)のコード

TOファンクションは、テクノロジーオブジェクト上で特定の動作を実行する主要なコマンドです。そのため、TOコマンドとも呼ばれます。
また、TOファンクションはユーザー定義FBで使用できます。
ユーザー定義FCおよびFBの書式規則の詳細については、『SIMOTION STプログラミングマニュアル』を参照してください。
以下の表に、TOファンクションの正式なコードの概要を示します。

表 3-3 TOファンクションのコード

コード	説明
名前	すべてのTOファンクションの名前(この例では <i>_enableAxis</i>)は、SIMOTION システム内で定義される識別子で、常に <i>_</i> (アンダスコア)で始まります。これらのTOファンクションとユーザー定義FBおよびFCを区別できるように、 <i>_</i> 文字で始まるソースファイルセクションは作成しないでください。
入力パラメータ	TOファンクションが呼び出され場合、そのTOファンクションには1つ以上の入力パラメータが含まれ、また常に呼び出し位置に戻り値を返します。出力パラメータはサポートされていません。 詳細については、サブセクション8.1.2.を参照してください。
戻り値	通常、すべてのコマンドは、倍精度整数(DINTデータタイプ)を返します。これは、コマンドがシステムに問題なく転送されて正常に処理されたかどうか(戻り値がゼロ)、またはエラーが発生したかどうか(戻り値がゼロ以外)を示します。

例

仮想軸を有効にする必要があるアプリケーションでは、ソースファイルを以下の図に示されているようにプログラムできます。

以下の条件を満たす必要があります。

- 位置決め軸または速度軸のインスタンスが、SIMOTION SCOUT内でAxis_1と呼ばれる仮想軸として作成されている必要があります。
- myPosプログラムが、たとえばMotionTask_1に割り当てられている必要があります。また、[Activation after StartupTask]オプションが、MotionTask_1のタスク設定で選択されている必要があります。
- ソースファイルが対象システムにダウンロードされている必要があります。

CPUがRUNモードに切り替わると、Axis_1ではイネーブルを発行します。軸イネーブルのステータスは、システム変数Axis_1.controlで確認できます。

表 3-4 TOファンクションのプログラムでの使用例

```
INTERFACE
    USEPACKAGE cam;
    PROGRAM myPos;
END_INTERFACE
IMPLEMENTATION
    // 以下のプログラムは、MotionTaskに割り当てる
    //   必要があります。
    //   タスクの設定で、[Activation
    //   after StartupTask] オプションを選択すること。
    PROGRAM myPos
        VAR
            retVal : DINT;
        END_VAR
        // 軸は位置決めに対して有効になります。
        retVal := _enableAxis (
            axis := Axis_1,
            // TOインスタンスの識別子
            nextCommand := WHEN_COMMAND_DONE,
            // プログラムを進めるための条件
            commandId := _getCommandId() );
        // 一意なコマンドID
    END_PROGRAM
END_IMPLEMENTATION
```

注記

パラメータ(値が割り当てられた)の受け渡しには、「テクノロジーファンクションの入力パラメータ」で説明されているロングフォームを使用します。このフォームは、明確さと柔軟性の点でより優れています。

システムファンクションでパラメータを効率的に使用するためのヒントについて、「誤差原因および効率的なプログラミング」に記載されています。

下記も参照

テクノロジーファンクションの入力パラメータ (ページ 68)

効率的なプログラミング - 概要 (ページ 442)

3.2.2 周期的プログラミングと逐次的プログラミングの相違点

周期的タスク

周期的タスク(BackgroundTaskなど)は、1つのタスクが完了するごとに起動されるか、または一定時間の経過後に自動的に起動されます。
割り当てたプログラムの静的変数の値は、保持されます。
周期的タスクでは時間監視が行われ、一定時間を超えると定義済みのエラー応答を返します。
そのため、周期的タスクに含まれるコードでは、そのタスクを迅速かつ効率的に実行する必要があります。
待ち特性を持つタスク(たとえば、軸の位置決めを待ちます)は、周期的タスクの複数の呼び出しサイクルにわたって実行されることになります。
したがって、通常、TOシステムコマンドを呼び出す際には、nextCommandパラメータでステップ有効化条件としてIMMEDIATELYを指定する必要があります。
その後の呼び出しサイクルでは、システムコマンドの結果が処理の成功またはエラーであるかをチェックする必要があります。この手順は、非同期実行とも呼ばれます。

逐次的タスク

起動後に、シーケンシャルタスク(MotionTaskなど)は一度だけ実行された後、終了します。
起動のたびに、割り当てたプログラムのローカル変数はすべて初期化されます(変数初期化に対するコンパイラの影響 (ページ 232)変数の初期化については～を参照)。
逐次的タスクでは時間監視は一切行われないため、実行時間の長さには制限はありません。
逐次的タスクは、アプリケーションの制御のみを目的としています。
つまり、アプリケーションでは、タスクを起動、終了、休止、および再開することができます。
逐次的タスクに含まれるコードは、連続的にタスクを処理します。ただし、以下のタスクは、通常、前のタスクが完了した場合にのみ実行されます。
たとえば、軸を最初に開放し、次に原点へ移動します。
そのため、TOシステムコマンドを呼び出す際には、nextCommandパラメータでステップ有効化条件としてWHEN_COMMAND_DONEを指定します。
システムファンクションが逐次的タスクの呼び出しに復帰するのは、コマンドの処理が完了した場合のみです。これは同期実行とも呼ばれます。

一般的な手順

一般に、逐次シーケンスはMotionTaskでプログラムする必要があります。
MotionTaskタスクでの逐次的プログラミングとBackgroundTaskでの周期的プログラミングは、以下の点が異なります。

- MotionTaskでは、継続のために、常に1つのステップ有効化条件(ただし、複数の条件を組み合わせることは可能)のみを待ちます。
ステップ有効化条件は、補間サイクルクロック内で優先的にチェックされます。
MotionTaskを継続するための応答時間を短縮する必要がある場合は、ステップ有効化条件のチェックの優先度を一時的に上げることができます。(-> WAITFORCONDITION)。
- 通常、周期的プログラムでは、各サイクルで多数の信号状態とステップ有効化条件をチェックします。これは、サイクルタイムに対して重い負荷となります。
逐次的プログラミングと比較した場合の長所は、要求とシーケンスの並列処理です。

同期および非同期コマンド実行

周期的または逐次的プログラムを作成した場合、コマンドは同期または非同期で処理されます。

詳細については、「移行およびステップ有効化条件 (ページ 73)」を参照してください。

3.2.3 テクノロジファクションの入力パラメータ

TOファクションには、以下のファクションパラメータがあります。

- **必須** - これらのパラメータは必ず指定する必要があります。たとえば、TOインスタンスおよび位置決め目標位置がこれに該当します。
- **定義済みオプション** - これらのパラメータは指定することができます。指定しなかった場合は、システムのデフォルト設定が有効になります。たとえば、ステップ有効化条件では、IMMEDIATELYが有効になります。
- **ユーザー定義オプション** - これらのパラメータは指定することができます。指定しなかった場合は、ユーザー定義のデフォルト動作が有効になります(関連づけられているuser Default変数でプログラムまたは設定が可能)。
モーションコントロールプログラムでのパラメータ値の使用については、各テクノロジーオブジェクト(TO)のマニュアルを参照してください。

実際には、TOファクションでは、オブジェクトを指定するために、常に少なくとも1つのパラメータを入力する必要があります。

「必ず使用する変数の例」にあるように、有効にする軸を指定する必要があります(axis := myaxis)。

入力パラメータの指定時に必ず適用される規則

以下の表に、入力パラメータを指定する際に従う必要がある規則を示します。

表 3-5 入力パラメータの指定時に適用される規則

規則	例
割り付け	
実際のパラメータは、常に仮パラメータに割り当てられます。	「必ず使用する変数の例」では、 <i>myAxis</i> (変数)およびIMMEDIATELY (値)の2つのカレントパラメータが仮パラメータの <i>axis</i> および <i>nextCommand</i> に割り当てられ、ファクションでデータが処理できるようになります。
複数の入力パラメータ	

規則	例
入力パラメータが複数ある場合はコンマで区切ります。	「必ず使用する変数の例」では、以下のようになっています。 <code>_enableAxis (axis:=myAxis, nextCommand := IMMEDIATELY)</code>
入力パラメータの順序	
入力パラメータの順序は、以下の最初にある2つの規則に従っている限り任意です。	「必ず使用する変数の例」では、 <code>_enableAxis (nextCommand:=IMMEDIATELY, axis:=myAxis)</code> も可能です。
デフォルト値を含む入力パラメータ	
デフォルト値を含むパラメータを省略した場合、システムによりデフォルト値が自動的に設定されます。	「必ず使用する変数の例」では、入力パラメータ <code>nextCommand := IMMEDIATELY</code> はデフォルト値なので省略することができます。
変数としての実パラメータ	
実パラメータを変数として入力する場合(本書の例では、TOパラメータ <code>myAxis</code>)は、最初に変数の定義と宣言を行う必要があります。 長所は、これらの変数を使用するソースファイルのセクションを再利用できます。 たとえば、ユーザー定義ファンクションブロック(FB)内でTOファンクションを呼び出す必要がある場合は、可変の軸識別子を入力パラメータとして使用します。 不変の軸名を使用してファンクションを複数回呼び出すことは困難であり、またFBが再利用できないことを意味します。 他の実パラメータを変数として入力することもできます。	「必ず使用する変数の例」では、ユーザー定義ファンクションブロック <code>posFB</code> でTOインスタンス <code>myAxis</code> (<code>myAxis:PosAxis</code>)が定義されています。 このFBからTOファンクションを呼び出す際には、TOインスタンスが実パラメータ(<code>Axis := myAxis</code>)として使用されます。 TOインスタンス用の値は、ユーザー定義FBからではなく、このFBにより <code>myFB (myAxis := Axis)</code> を用いて呼び出されるプログラムから取得されます。 したがって、FBを呼び出して、<code>myAxis := Axis2</code> または <code>myAxis := Axis3</code> などでTOファンクションを呼び出すこともできます。 つまり、FBは、STソースファイルのすべてのプログラムで使用でき、また他のソースファイルのプログラムではエクスポートファンクションを用いることで使用できます。
変数としての実パラメータ(列挙子)	
実パラメータを変数として入力した場合(本書の例では、<code>nextCommand := IMMEDIATELY</code>)、さまざまな実際の値(定義済みの列挙子)から少なくとも1つを選択する必要があります。 列挙子は、派生データタイプです。 列挙子の基本情報については、サブセクション3.4.2に記載されています。	「必ず使用する変数の例」では、IMMEDIATELYおよびWHEN_COMMAND_DONE列挙子を <code>_enableAxis</code> TOファンクションの <code>nextCommand</code> 仮パラメータで使用できます。 他の値は、プログラムのコンパイル中にコンパイラにより拒否されます。
移行およびステップ有効化条件用のパラメータ	
多くのモーションコマンドでは、TOコマンドがテクノロジーオブジェクト上でいつ実行されるか、および次の命令文がいつ処理されるかを指定できます(「移行およびステップ有効化条件」を参照)。	
コマンド識別用のパラメータ	
すべてのモーションコマンドでは、コマンド識別用のパラメータが指定されている必要があります。このパラメータを使用すると、コマンド実行のステータスを追跡できます(「コマンド実行の診断」を参照)。	

注記

システムファンクション内およびシステム変数内にある列挙体用のすべてのシステムデータタイプは、ワード *Enum* で始まります。たとえば、仮パラメータ *nextCommand* では、列挙型データタイプは *EnumNextCommandEnable* (値として *IMMEDIATELY* または *WHEN_COMMAND_DONE* を含む) になります。

システムファンクション内およびシステム変数内にある構造体のすべてのシステムデータタイプは、ワード *Struct* で始まります。

ユーザー定義データタイプに *Enum* または *Struct* 以外の文字列で始まる名前を指定した場合は、名前をオーバーラップさせることができません。

本書の「付録D」には、ST(構造化テキスト)プログラミング言語、システムファンクション、システムブロック、およびSIMOTIONデバイスの予約済み識別子がすべて記載されています。

SIMOTIONテクノロジーパッケージの予約済み識別子については、各SIMOTIONテクノロジーパッケージのパラメータマニュアルに記載されています。

運動変数用の参照パラメータおよび値パラメータ

運動変数のパラメータ(速度、加速度など)は、参照パラメータと値パラメータを使用して定義されます。

- 参照パラメータでは、運動変数が転送されるシステム変数のどれを参照するかを指定し、さらに必要であればその後に続く値パラメータをどのように解釈するかを指定します。

参照パラメータの識別子は、サフィックスが *Type* (たとえば、*velocityType*) の運動変数の識別子から生成されます。

- 値パラメータでは、以下の指定を行います。
 - 参照パラメータ = DIRECT の場合: 運動変数の数値。
 - 参照パラメータ = USER_DEFAULT の場合: システム変数に格納されているデフォルト値のスケーリング計数。

値パラメータは、他の参照パラメータに影響することはありません。

値パラメータの識別子は、運動変数の識別子です(たとえば、*velocity*)。

表 3-6 よく使用される参照パラメータおよび関連する値パラメータの効果

参照パラメータ	値パラメータ	運動変数の値
DIRECT	絶対値	値パラメータの内容(テクノロジーオブジェクトの設定中に指定される運動変数の単位を含む)。 以下の表にある例を参照してください。
USER_DEFAULT	相対値[%]	デフォルト設定×値パラメータ / 100 運動変数のデフォルト設定は、システム変数に格納されています。 以下の表にある例を参照してください。
ACTUAL	-1	現在の現在値

参照パラメータ	値パラメータ	運動変数の値
CURRENT	- ¹	インターポレータの現在の設定値
EFFECTIVE	- ¹	最後にプログラムされた値
¹ 値パラメータは無視されます。		

表 3-7 velocityType (参照パラメータ)およびvelocity (値パラメータ)の例

velocityType	velocity	運動変数の値
DIRECT	(該当なし)	100 [設定された単位]
	50.0	50 [設定された単位]
	200.0	200 [設定された単位]
USER-DEFAULT	(該当なし)	システム変数値の100% ¹
	50.0	システム変数値の50% ¹
	200.0	システム変数値の200% ¹
¹ 軸のモーションコマンドの場合: システム変数 <i>userDefault.velocity</i>		

ショートフォームのパラメータ転送

ショートフォームのパラメータ転送では、仮パラメータ(パラメータ識別子)は省略され、実パラメータ(パラメータ値)のみが使用されます。

通知
ショートフォームのパラメータ転送では、すべてのパラメータ値(オプションのパラメータ値であっても)をコマンドで区切って、正しい順序で列べます。

呼び出された際にショートフォームのパラメータ転送を必要とするファンクションは、ほんのわずかです(「タスクのランタイム測定用のファンクション」、「タスク制御コマンド」、および「メッセージ設定用のファンクション」を参照)。
これは、関連するファンクションに対して明示的に指定されます。

注記

前記のロングフォームを使用してください(値の割り当てに)。
このフォームは、明確さと柔軟性の点でより優れています。

TOインスタンスのデータタイプ

たとえば、TOインスタンスは、TOファンクション呼び出しでのオブジェクト指定に使用できます。
最初に変数を宣言する必要がありますが、ただし、データタイプを定義済みの選択肢から選択する必要があります(「テクノロジーオブジェクトのデータタイプ」の表を参照)。
このセクションの「プログラム内でのTOファンクションの使用」にある例(必ず使用する変数の例)では、データタイプは*PosAxis*になっています。これは、位置決め軸用のTOインスタンスを作成する必要があるからです。選択した*Cam*

(不連続同期操作)テクノロジーパッケージには、*Position*
Axisテクノロジーオブジェクト用の*PosAxis*データタイプが含まれています。

表 3-8 テクノロジーオブジェクトのデータタイプ(テクノロジーオブジェクトの名前)

テクノロジーオブジェクト	データタイプ	含まれているテクノロジーパッケージ
ドライブ軸	driveAxis	CAM ^{1 2} 、CAM_EXT
外部エンコーダ	externalEncoderType	CAM ^{1 2} 、CAM_EXT
測定入力	measuringInputType	CAM ^{1 2} 、CAM_EXT
出力カム	outputCamType	CAM ^{1 2} 、CAM_EXT
カムトラック(V3.2以降)	_camTrackType	CAM、CAM_EXT
位置決め軸	posAxis	CAM ^{1 3} 、CAM_EXT
従動軸	followingAxis	CAM ^{1 4} 、CAM_EXT
従動オブジェクト	followingObjectType	CAM ^{1 4} 、CAM_EXT
Cam	camType	CAM、CAM_EXT
固定ギア(V3.2以降)	_fixedGearType	CAM_EXT
追加オブジェクト(V3.2以降)	_additionObjectType	CAM_EXT
数式オブジェクト(V3.2以降)	_formulaObjectType_	CAM_EXT
センサ(V3.2以降)	_sensorType	CAM_EXT
コントローラオブジェクト(V3.2以降)	_controllerObjectType_	CAM_EXT
温度チャンネル	temperatureControllerType	TControl
汎用データタイプ(すべてのTOに割り当て可能)	ANYOBJECT	
¹ バージョンV3.1以降には、BasicMC、Position、およびGearテクノロジーパッケージが含まれています。 ² バージョンV3.0以前では、BasicMC、Position、およびGearテクノロジーパッケージにも含まれています。 ³ バージョンV3.0以前では、PositionおよびGearテクノロジーパッケージにも含まれています。 ⁴ バージョンV3.0以前では、Gearテクノロジーパッケージにも含まれています。		

テクノロジーオブジェクト(TO)データタイプの変数は、デフォルトでは値TO#NILで初期化されます。

これは、変数に有効なTOが割り当てられたかどうかをチェックするのに使用できます。詳細については、「**必ず使用する変数の例**」を参照してください。

必ず使用する変数の例

必ず使用するテクノロジーオブジェクトデータタイプを含む変数の例を、以下に示します(オプションで使用する場合の例については、「テクノロジーオブジェクトのデータタイプ」を参照)。この例では、再使用可能なFBを記述して、任意の軸を有効にします。軸の名前は変数なので、軸のデータタイプを含む変数を定義する必要があります。

有効にされた軸は、FBが呼び出された際に提供されます。

この軸は、安全上の理由で検証されます。

TOが利用できない場合(TO#NIL)、FBの実行は中断されます。

表 3-9 データタイプの指定に必ず使用するテクノロジーオブジェクトの変数の例

```
// ...
FUNCTION_BLOCK posFB
  VAR_INPUT
    myAxis: posAxis;
  END_VAR

  VAR_OUTPUT
    // TOファンクションの値と、
    // FBの出力パラメータを返します。
    return_value: DINT:= -1;
  END_VAR

  // 有効なTOをチェック
  IF myAxis = TO#NIL THEN RETURN; END_IF;
  // TOのデータタイプの変数を使用する呼び出しの例
  return_value := _enableAxis (
    axis:= myAxis, // TOファンクション
    nextCommand:= IMMEDIATELY, //オプション
    commandId    := _getCommandId() );
END_FUNCTION_BLOCK
PROGRAM EXAMPLE
  VAR
    myFB: posFB;
  END_VAR
  myFB (myaxis := Axis1);
  // 名前は、SIMOTION SCOUT内で起動時に作成されます。
  myFB (myaxis := Axis2);
  // 名前は、SIMOTION SCOUT内で起動時に作成されます。

END PROGRAM
//...
```

3.2.4 移行およびステップ有効化条件

TOファンクションの処理(コマンド実行)の基本原則

TOファンクションは、実行のために、コマンドとしてテクノロジーファンクションに転送されます。
TOでは、設定中に指定された処理サイクルクロック(IPOサイクルクロックなど)で、これらのコマンドを実行または有効にします。

outputCam、measuringInput、externalEncoder、およびcamテクノロジーオブジェクトでは、コマンドを直接実行できます。
同じテクノロジーオブジェクト上の新しいコマンドは、そこで有効になっているコマンドに優先します。

直接コマンド実行用のコマンドに加えて、以下のテクノロジーオブジェクト上ではモーションコマンドも発行できます。driveAxis、posAxis、followingAxis、およびfollowingObject
テクノロジーオブジェクトには、コマンド管理用の構造体要素があります。

以下の例では、軸のコマンド管理について説明されています。

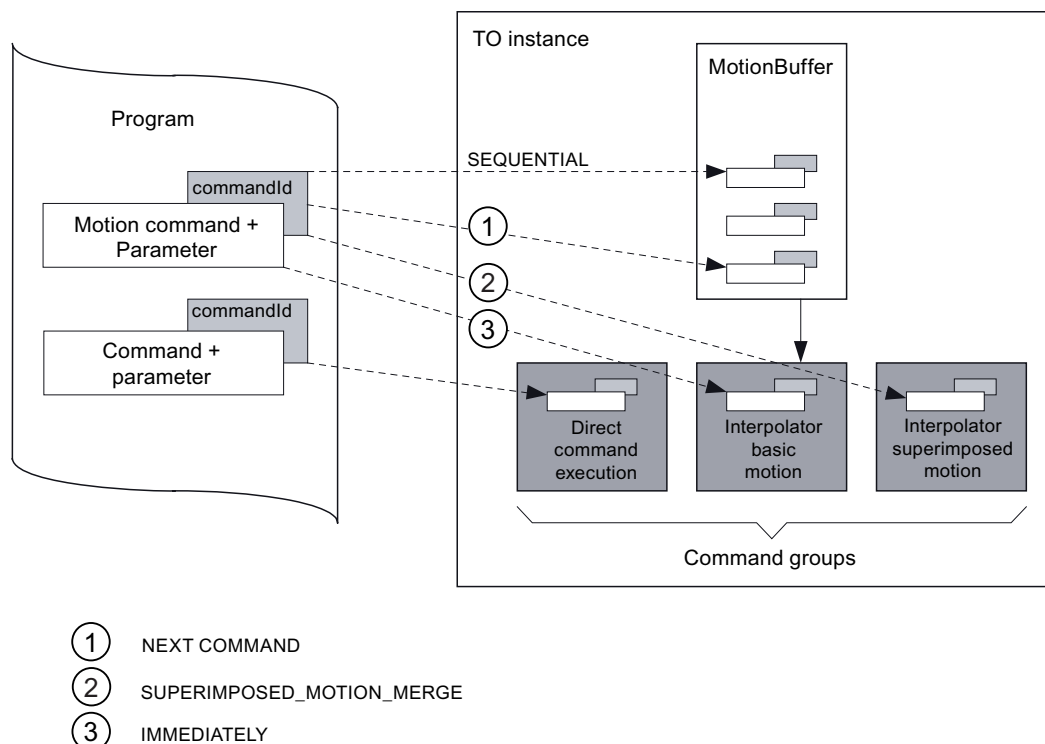


図 3-1 軸上でのコマンドの実行

MotionBufferには、インターポレータにより逐次的に実行されるモーションコマンドが格納されています。

モーションバッファに格納できるモーションコマンドの数は、コンフィグレーションデータ *TypeOfAxis.DecodingConfig.numberOfMaxbufferedCommandId* によりされます。したがって、テクノロジーオブジェクト上では、動作中のコマンドの実行ステータスに関係なく複数のモーションコマンドを発行できます。

コマンドが発行されると、そのコマンドには **commandId** が割り当てられます。これはコマンド内に格納され、発行されたコマンドへの参照を提供します。

コマンドは、コマンドグループに割り当てられます。既存のコマンドグループ内にある保留コマンドは、補間により並列で処理されます。特定のコマンドはテクノロジーオブジェクト上で直ちに有効になりますが、重畳コマンドなどの他のコマンドはモーションバッファ内のキューから有効になります。

現在有効なモーションコマンドの移行動作

TOファンクション内のテクノロジーオブジェクト上で現在有効になっているモーションコマンドの移行動作は、パラメータ *MergeMode* で指定します。このパラメータでは、テクノロジーオブジェクト上のコマンド実行シーケンスでTOファンクションをどのように配置するか、またはTOファンクションをどのコマンドグループに割り当てるかを指定します。

表 3-10 よく使用されるマージモード

マージモード	説明
IMMEDIATELY	コマンド内で指定された動作は、直ちに有効になります。既に有効になっている動作は置き換えられ、また保留コマンド / 動作は中断されます。
NEXT_MOTION	有効な動作の後に実行し、その他の保留コマンド / 動作を削除します。
SEQUENTIAL	前のコマンド / 動作に追加します。
SUPERIMPOSED_MOTION_MERGE	軸には、基本動作に重畳する動作を追加できます。

コマンドのステップ有効化条件

インターポレータで実行されるTOファンクションには、ステップ有効化条件をプログラムできます。これは、プログラムソース内の次の命令文がいつ実行されるかを指定します。
*nextCommand*パラメータはこの目的で使用されます。
各コマンドの許容条件については、『SIMOTION TP Cam System Functions Parameter Manual』(リファレンスリスト)を参照してください。

非同期および同期コマンド実行は、主に以下の点が異なります。

- 非同期コマンド実行

TOファンクションはテクノロジーオブジェクトに転送され、プログラムは直ちに続行されます。そのためには、*nextCommand*= IMMEDIATELYを設定します。

この場合、TOファンクションが1回だけ発行されること、および軸またはコマンドステータスのチェックにより点検信号が明示的に評価されることを、アプリケーション側で保証する必要があります(「**コマンド実行の診断**」を参照)。

例: 以下の2つの図を参照

このモーションコントロールの方法は、**周期的プログラミング**と呼ばれます。
これはすべてのシステムタスクで可能で、特に周期的タスクのプログラミングを目的としています。

非同期コマンド実行は、パラメータ*nextCommand*が指定されていない場合のデフォルト設定です。

- 同期コマンド実行

ステップ有効化用のパラメータが設定されているTOファンクションは、テクノロジーオブジェクトおよび停止された呼び出しタスクに転送されます。
テクノロジーオブジェクトではファンクションを実行し、指定したステップ有効化条件に適合するか、またはコマンドが中断すると、直ちにプログラム実行の再開を要求します。このために、*nextCommand*が目的のステップ有効化条件に設定されます。

例: 「**非同期プログラム実行(逐次的プログラミング)**」を参照してください。

このモーションコントロールの方法は、**逐次的プログラミング**と呼ばれます。
これは、特にMotionTasksでサポートされています。

周期的タスクのプログラミングコマンドシーケンスは、タスクタイムアウトおよびそれによるランタイムエラーの原因になります。

表 3-11 非同期プログラム実行(周期的プログラミング)の例 - パート1

```

INTERFACE
USEPACKAGE CAM;
    PROGRAM ProgramCycle;
END_INTERFACE

IMPLEMENTATION

PROGRAM ProgramCycle
    VAR
        boStartCommand : BOOL; // コマンド - コマンドを発行
        boCommandStarted : BOOL; // 補助変数 - コマンド発行済み
        boCommandDone : BOOL; // 補助変数 - コマンド実行済み
        i32Ret : DINT; // システムファンクションの戻り値
        sCommandId : CommandIdType; // CommandId
        // 戻り値 - _getStateOfAxisCommand
        sRetCommandState : StructRetCommandState;
        // エッジ検出用のシステムFBのインスタンス
        r_trig_1 : R_TRIG;
    END_VAR

    r_trig_1 (boStartCommand); // エッジ検出を呼び出し

    IF r_trig_1.Q THEN
        // システム全体で一意的なcommandIdを要求
        sCommandId := _getCommandId ();
        // TOでcommandIdを登録
        // --> コマンドの終了または中断の診断が可能
        i32Ret := _bufferAxisCommandId (
            axis := Axis_1,
            commandId := sCommandId );
        // システムファンクションの戻り値の評価
        // ...
        // コマンドの発行- USER_DEFAULT値に基づく動作
        i32Ret := _move(
            axis := Axis_1,
            nextCommand := IMMEDIATELY,
            commandId := sCommandId );
        // システムファンクションの戻り値の評価
        // ...
        // コマンド実行を調整するための補助変数
        boCommandStarted := TRUE;
        boCommandDone := FALSE;
    //-----
    // 以下に続く

```

表 3-12 非同期プログラム実行(周期的プログラミング)の例 - パート2

```
// 続き
//-----
ELSIF boCommandStarted AND NOT boCommandDone THEN
    // コマンド実行ステータスのクエリー
    sRetCommandState := _getStateOfAxisCommand(
        axis := Axis_1,
        commandId := sCommandId );

    IF sRetCommandState.functionResult = 0 THEN
        IF sRetCommandState.commandIdState = EXECUTED THEN
            // コマンドは実行済み(完了)
            boCommandStarted := FALSE;
            boCommandDone := TRUE;
            // TO上で登録済みcommandIdを削除
            i32Ret := _removeBufferedAxisCommandId(
                axis := Axis_1,
                commandId := sCommandId );
        END_IF;
    ELSE
        // _getStateOfAxisCommand function呼び出し用のエラー処理
        // ...
        ;
    END_IF;

    ELSIF boCommandDone THEN
        // コマンド実行後の実行
        // ...
        ;
    END_IF;

// これ以降は追加的なユーザープログラム
// ...

END_PROGRAM

END_IMPLEMENTATION
```

表 3-13 同期プログラム実行(逐次的プログラミング)の例

```
INTERFACE
    USEPACKAGE CAM;
    VAR_GLOBAL
        g_boCommandStarted : BOOL; // 補助変数 - コマンド発行済み
        g_boCommandDone : BOOL; // 補助変数 - コマンド実行済み
    END_VAR
    PROGRAM ProgramSequential;
END_INTERFACE

IMPLEMENTATION
    PROGRAM ProgramSequential
        VAR
```

```

        i32Ret : DINT: // システムファンクションの戻り値
END_VAR;
g_boCommandStarted := TRUE;
g_boCommandDone := FALSE;
// 動作の前に実行される命令文
// ...
i32Ret := _move(
axis := Axis_1,
nextCommand:= WHEN_MOTION_DONE,
commandId:= _getCommandId () );

// 動作の後に実行される命令文
// ...
// システムファンクションの戻り値の評価
// ...

g_boCommandStarted := FALSE;
g_boCommandDone := TRUE;

END_PROGRAM

```

3.2.5 コマンド実行の診断

コマンド識別 – commandId

システムファンクションがコマンドを発行すると、commandIdが転送されます。コマンドがテクノロジーオブジェクトにより実行されている間、テクノロジーオブジェクトはコマンドを識別するためにcommandIdをそのコマンド内に保存します。

プロジェクト全体で一意的なcommandIdは、`_getCommandId`システムファンクションを使用して取得されます。これにより、同じcommandIdを持つ別のコマンドがシステム内に存在しないことが保証されます(コマンドへの一意的参照)。

表 3-14 commandIdの使用例

```

//...
VAR
    myCommandId : CommandIdType;
END_VAR
//...
// 一意のIDの保存
myCommandId := _getCommandId ();
// IDを使用したファンクションの実行
myFC := _pos (axis := myAxis,
              position := position_1,
              nextCommand := IMMEDIATELY,
              commandId := myCommandId);
//...

```

この後は、commandIdを使用してコマンドの実行ステータスを追跡する方法について説明します。

コマンド / 実行ステータスをチェックするためのシステムファンクション

複数の実行するコマンドが発行されるテクノロジーオブジェクトには、`_getStateOf...Command`システムファンクション(`_getStateOfAxisCommand`など)があります。

`StructRetCommandState`データタイプの戻り値では、`EnumCommandIdState`コンポーネント内のインターポレータによるモーションコマンドの実行ステータスに関する情報が提供されます。

SIMOTION KernelのバージョンV3.2以降では、中断理由は`EnumCommandIdState = ABORTED`の`abortId`コンポーネント内で指定されます。

この値は、テクノロジーアラーム"30002 Command aborted"の理由に対応します。

コマンドが完了または中断した後のステータスチェック

コマンドの実行が中断または完了すると、通常そのコマンドはテクノロジーオブジェクトの内部コマンド管理システムから削除されます。

そのため、上記のシステムファンクションを呼び出してコマンドステータス`End`または`Abort`を診断することはできません。

コマンドの実行が中断または完了した後でもそのステータスをチェックできるようにするには、関連するコマンドの`commandId`がテクノロジーオブジェクトの内部コマンド管理システムで認識されるようにする必要があります。

これは、`_buffer...CommandId`システムファンクション(`_bufferAxisCommandId`など)を使用して実行されます。

`End`または`Abort`ステータスが評価された後、`commandId`はテクノロジーオブジェクトのコマンド管理システムから明示的に削除する必要があります。

これは、システムファンクション`_removeBuffered...CommandId` (`_removeBufferedAxisCommandId`など)により実行されます。

例:

「非同期プログラム実行(周期的プログラミング)、パート1およびパート2」を参照してください。

テクノロジーオブジェクトをリセットしたときの動作については、以下の「テクノロジーオブジェクトをリセットした後のステータスチェック」を参照してください。

同時に管理する必要があるコマンドの最大数をテクノロジーオブジェクトに通知するために、コンフィグレーションデータ `TypeOfAxis.DecodingConfig.numberOfMaxbufferedCommandId`を設定する必要があります。

テクノロジーオブジェクトをリセットした後のステータスのチェック

SIMOTION

KernelのバージョンV3.2以降では、`commandId`はバッファに格納されるので、テクノロジーオブジェクトがリセットされても削除されません。

したがって、テクノロジーオブジェクトがリセットされた後も`commandId`を使用できます。

- これを行うには、TOファンクション `_buffer...CommandId`を `deleteCommandIdWithReset = NO`パラメータを指定して呼び出します。

その後で、commandIdを削除するには、`_removeBuffered...CommandId`コマンドを使用して明示的に削除する必要があります。

- また、`_buffer...CommandId`TOファンクションを `deleteCommandIdWithReset= YES` (デフォルト設定)パラメータを指定して呼び出した場合、テクノロジーオブジェクトをリセットする際にcommandIdは `_reset...ファンクション` (`_resetAxis`など)でも削除されます。

なお、これはSIMOTION KernelのバージョンV3.1以前の動作です。

3.2.6 テクノロジーオブジェクトインスタンスの識別子

テクノロジーオブジェクトインスタンスの識別子は、SIMOTION SCOUTでこれらのインスタンスの設定中に定義されます。

識別子は、SIMOTIONデバイス内で一意に定義する必要があります。

一般に、テクノロジーオブジェクトのインスタンスは、その識別子を使用して呼び出すことができます。

ただし、STソースファイルでデータタイプ、変数、ファンクションや、ファンクションブロックに同じ識別子が指定された場合、または同じ名前のグローバルデバイス変数やI/O変数が作成された場合、テクノロジーオブジェクトインスタンスはこれらの識別子の対象になります。

定義済みの名前空間 `_to` を使用しても、テクノロジーオブジェクトインスタンスにアクセスできます。たとえば、インスタンスのシステム変数またはコンフィグレーションデータにアクセスできます(『STプログラミングマニュアル』の「**Name spaces**」を参照)。
名前空間は、対応する名前の前にピリオドで区切って入力してください。たとえば、`_to.to-name` のように入力します。

別のSIMOTIONデバイス上にあるテクノロジーオブジェクトのインスタンスにアクセスする必要がある場合は、SIMOTIONデバイスの名前をインスタンス識別子の前にピリオドで区切って入力してください。たとえば、`dev-name.to-name` または `dev-name._to.to-name` のように入力します。

デバイスの識別子が対象になっている場合は、定義済みの名前空間 `_project` を使用できます。たとえば、`_project.dev-name.to-name` または `_project.dev-name._to.to-name` のように指定します。

注記

プロジェクト全体で一意なテクノロジーオブジェクトインスタンスの識別子を使用する場合、定義済みの名前空間 `_project` もインスタンスの識別に使用できます。

これにより、SIMOTION

SCOUTの旧バージョン(V3.2以前)で作成されたプロジェクトとの互換性が保証されます。

3.2.7 TOデータタイプの変換

階層的なデータタイプ内でのタイプ変換

driveAxis、*posAxis*、*followingAxis*

TOデータタイプは、機能範囲に基づく階層構造を持っています。

- 位置決め軸(*posAxis*データタイプ)には、速度制御軸(*driveAxis*データタイプ)の機能が含まれています。
- 従動軸(*followingAxis*データタイプ)には、位置決め軸(*posAxis*データタイプ)の機能が含まれているので、したがって速度制御軸(*driveAxis*データタイプ)の機能も含まれます

タイプ変換は、これらの階層的なデータタイプ内および汎用のANYOBJECTテクノロジーオブジェクトデータタイプとの間でのみ可能です。

注記

他のタイプの変換はできません(たとえば、測定入力と従動オブジェクト間の変換)。

暗黙的なタイプ変換

変数(TOデータタイプを含む)またはTOインスタンスは、変換ファンクションを指定しなくても以下の変数に割り当てることができます。

- 下位層にあるTOデータタイプの変数
 - *followingAxis* から *posAxis* または *driveAxis*
 - *posAxis* から *driveAxis*
- 汎用ANYOBJECT TOデータタイプの変数

以下の表を参照してください。

表 3-15 暗黙的なタイプ変換の例

```
// 以下のTOインスタンス(軸)はプロジェクトナビゲータで設定します。 //  
従動軸としてfol_axis_real  
  
VAR  
    drv_axis1 : driveAxis;  
    pos_axis1 : posAxis;  
    any_obj1  : ANYOBJECT;  
END_VAR  
  
drv_axis1 := pos_axis1;  
any_obj1  := fol_axis_real;...
```

明示的なタイプ変換

*anyObject_to_Object*タイプ変換ファンクションは、変数(TOデータタイプを含む)を上位層にあるTOデータタイプに割り当てるために使用されます。

そのための必要条件は、ソース変数(下位層にあるTOデータタイプを含む)が、対象変数のTOデータタイプに少なくとも階層的に対応しているTOインスタンスを参照していることです。

表 3-16 タイプ変換の成功例

```
// 以下のTOインスタンス(軸)はプロジェクトナビゲータで
// 設定します。
// 位置決め軸としてpos_axis_real
// 従動軸としてfol_axis_real

VAR
    drv_axis1 : driveAxis;
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR

// 暗黙的なタイプ変換
drv_axis1 := pos_axis_real;
any_obj1 := fol_axis_real;

// 正常に完了したタイプ変換

pos_axis1 := anyObject_to_Object (in := drv_axis1);
// タイプ変換が正常に完了しました。
// drv_axis1は位置決め軸を参照しています。

pos_axis1 := anyObject_to_Object (in := any_obj1);
// タイプ変換が正常に完了しました。
// any_obj1は従動軸を参照しています。

//...
```

タイプ変換が失敗した場合、対象変数には値としてTO#NILが割り当てられます。

表 3-17 タイプ変換の失敗例

```
// 以下のTOインスタンス(軸)はプロジェクトナビゲータで
// 設定します。
// ドライブ軸としてdrv_axis_real

VAR
    pos_axis1 : posAxis;
    any_obj1 : ANYOBJECT;
END_VAR
```

```
// 暗黙的なタイプ変換
any_obj1 := drv_axis_real;

// タイプ変換に失敗
pos_axis1 := anyObject_to_Object (in := any_obj1);
// タイプ変換は必要ありません。
// any_obj1は速度制御軸を参照しています。
// pos_axis1は、値としてTO#NILを保持しています。
```

3.2.8 システム変数

システム変数を使用すると、テクノロジーオブジェクトと基本システム用のパラメータの割り当て、またはそれらのステータスを読み取ることができます。

システム変数の構文

システム変数のクエリーには、以下の構文のストラクチャアクセスが使用されます。以下の表に、個々の構文要素の意味を示します。

`[_to.]TO name.variable.component` または

`[_device.]variable.component`

表 3-18 システム変数の構文要素

構文要素	重要度
<i>TO name</i>	<i>TO name</i>は、テクノロジーオブジェクト(TO)の名前を表しています(たとえば、軸の名前)。以下のいずれかが実行されている必要があります。 - SIMOTION SCOUTでテクノロジーオブジェクトを挿入 - またはプログラム内でこのテクノロジーオブジェクトのデータタイプの変数を宣言 テクノロジーオブジェクトのすべてのデータタイプは、「テクノロジーファンクションの入力パラメータ」のリストに記載されています。
<i>_to</i>	オプションワード <i>_to</i>では、テクノロジーオブジェクトの定義済み名前空間を特定します。これは、<i>TO name</i>でテクノロジーオブジェクトを一義的に指定するために使用します。 『STプログラミングマニュアル』の「名前空間」も参照してください。
<i>_device</i>	オプションワード <i>_device</i>では、デバイス固有変数(SIMOTIONデバイスのシステム変数、I/O変数、グローバルデバイス変数)の定義済み名前空間を特定します。 これは、変数をSIMOTIONデバイスのシステム変数として一義的に指定するために使用します。 これらのシステム変数は、テクノロジーオブジェクトがロードされていない場合でも使用できます。 『STプログラミングマニュアル』の「名前空間」も参照してください。

構文要素	重要度
<i>Variable</i>	<i>Variable</i> は、すべてのシステム変数のリストに記載されているシステム変数を表しています(SIMOTIONテクノロジーパッケージシステム変数のパラメータマニュアルを参照)。
<i>Component</i>	<i>Component</i> は、クエリーする必要がある構造体の部分を表しています。これは、他のコンポーネントを含む追加的な構造体にもできます。構造体の階層の深さはシステム変数によって決まり、またゼロにすることもできます。

システム変数の使用

システム変数への読み取りアクセスは、常に可能です。

システム変数は、以下の2つの方法で変数に割り当てることができます。

- 値を割り当てる場合(『STプログラミングマニュアル』の「**値の割り当て**」を参照)。
この場合、エラーのイベントでExecutionFaultTaskが呼び出されます。
エラー応答の詳細については、「**システム変数へのアクセス**」を参照してください。
- `_getSafeValue`システムファンクションを使用する場合(「`_getSafeValue` (ページ 334)」および「システム変数および入力 / 出力へのアクセス (ページ 334)」を参照)。
この場合、エラーのイベントに必要な応答を得るようにプログラムできます。

式での使用、またはファンクションやファンクションブロックでのパラメータとしての使用が可能です。

この場合、エラーのイベントでExecutionFaultTaskも呼び出されます(「**システム変数へのアクセス**」を参照)。

システム変数への書き込みができるかどうかは、テクノロジーオブジェクト(TO)のシステム変数のパラメータマニュアル(リファレンスリスト)にある以下のエントリで指定されています。

- Effective: *immediately* (読み取りと書き込みが可能)
- Effective: *read only* (読み取りのみが可能)

システム変数への書き込みが可能な場合、システム変数には以下の2つの方法で値を割り当てることができます。

- 値を割り当てる場合(『STプログラミングマニュアル』の「**値の割り当て**」を参照)。
この場合、エラーのイベントでExecutionFaultTaskが呼び出されます。
エラー応答の詳細については、システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー (ページ 95)を参照してください。
- システムファンクション`_setSafeValue`を使用する場合(ファンクション「`_setSafeValue` (ページ 337)」を参照)。
この場合、エラーのイベントに必要な応答を得るようにプログラムできます。

TO#NILは、特別な変数値です。

この値は、有効なテクノロジーオブジェクトが存在しているかどうかをチェックするのに使用できます(「**テクノロジーファンクションの入力パラメータ**」にある例を参照)。

注記

性能上の理由で、システム変数には絶対に必要な場合にのみアクセスするようにしなければなりません。
代わりに、システム変数の内容を同じデータタイプのローカル変数に保存する方法もあります。
ローカル変数へのアクセスでは、必要なリソースははるかに少なく済みます。これは、使用されるプロセッサ時間が少ないからです。
詳細については、「**効率的なプログラミング**」を参照してください。

また、エラーの原因は、REAL変数、LREAL変数、およびシステム変数(軸位置など)を比較することで確認できます(「**REAL変数またはLREAL変数の比較**」を参照)。

システム変数の範囲

1. システム変数(たとえば、ステータスフラグ)は、1つのIPOサイクルクロックの間だけ存在する場合があります。
2. すべてのシステム変数には、文書化されたUpdateプロパティがあります。
3. より短いサイクル時間でタスクのステータスをクエリーする必要がある場合は、ステータスをそれに続くORステータスにリンクします。
OR演算では、アプリケーションのすべての状態が確実に使用されるようにします。
また、この演算は、ステータスフラグの後に続くENUMSが確実にグループ化されるようにします。

システム変数の例

Axis1の軸位置と動的な軸状態をチェックする必要があると仮定します。

前提条件

- Axis1が、SIMOTION SCOUTで作成されているか、またはプログラムで定義および初期化されている必要があります(たとえば、PosAxisデータタイプのmyAxis変数を使用)。
- プログラム内で、軸位置および動的な軸状態を記録するための変数を定義してある必要があります。
これらの変数のデータタイプは、以下の例のように、チェックされる変数のデータタイプと一致していなければなりません。

```
VAR
act_pos : LREAL;
act_motionState : EnumAxisMotionState;
END_VAR
```

基本データタイプの構造体要素によるシステム変数へのアクセスの例

```
act_pos := Axis1.positioningState.actualPosition;
```

PositioningStateはシステム変数で、またactualPositionはクエリーされるデータタイプLREALの構造体要素です。

列挙体要素によるシステム変数へのクエリーの例

```
act_motionState := Axis1.motionStateData.motionState;
```

motionStateDataはシステム変数で、またmotionStateはクエリーされる列挙子データタイプEnumAxisMotionStateの構造体要素です。

システム変数の初期化

プロジェクトをダウンロードする際の規則として、システム変数は初期化されません。つまり、システム変数の現在値は、初期値にリセットされません。
通常は、SIMOTIONデバイスが再起動された場合にのみ、システム変数が再初期化されます。

再起同時における置換値またはシステム変数の最新値の読み込み(V4.1以降)

また、TOをRESTARTまたはTOを有効化する際には、システムをSTOPに移行しなくてもシステム変数にアクセスできます。
ファンクション_getSaveValueを介してシステム変数を読み取る代わりに、コンフィグレーションデータ内のエントリ(restart info)を使用して以下の設定を行うこともできます。

- 最新値の読み込み(LAST_VALUE = デフォルト値)
- デフォルト値の読み込み(=プロジェクトのロード時の値であるDEFAULT_VALUE)
- STOPへの移行(STOP_DEVICE)

例

```
restartInfo : RestartInfo
  behaviorInvalidSysvarAccess : EnumInvalidSysvarAccess
  [
    LASTVALUE I
    DEFAULT_VALUE I
    STOP_VALUE
  ]
```

例外

現在のTOステータスを提供する変数では、RESTARTのときに正しいステータスも返します。
これは、_getSaveValueを介してアクセスできるシステム変数restartActivationに影響を及ぼします。

注

TOシステム変数への書き込みは、RESTARTの間も可能です。
値は、RESTARTの後に適用または有効になります。

注記

ONLINEで保存したシステム変数は、[Save to memory card (Ram2Rom)]では保存できません。また、[Save in the engineering project (コンフィグレーションデータのPGへのロード)]で保存することもできません。

システム変数の値もエンジニアリングプロジェクトとメモリカードに保存できるようにするには、システム変数の値をOFFLINEモードで変更してから、ダウンロードと保存により対象システムにロードする必要があります。「対象システムでのストレージコンセプト (ページ 41)」を参照してください。

3.2.9 コンフィグレーションデータ

コンフィグレーションデータでは、テクノロジーオブジェクトの基本機能を定義します。
通常、コンフィグレーションデータはSIMOTION
SCOUTでテクノロジーオブジェクトを設定している間に決定され、その大部分はランタイム中に変更することはできません。

ただし、このコンフィグレーションデータの重要な部分は、プログラムが実行中であっても変更できます。

コンフィグレーションデータ項目がランタイム中に変更できるかどうかは、コンフィグレーションデータ項目ごとに異なります。これについては、SIMOTIONテクノロジーパッケージコンフィグレーションデータのパラメータマニュアルに記載されています。

コンフィグレーションデータ項目によっては、以下のオプションが使用できます。

- オンラインで変更不可

これらのコンフィグレーションデータは、SIMOTION
SCOUTでテクノロジーオブジェクトを設定している間だけ変更できます。

- オンラインで変更可能、再起動後に有効化

これらのコンフィグレーションデータは、ユーザープログラムからアクセスする変数により変更できます。

この変更は、テクノロジーオブジェクトが再起動されるまでは有効になりません(「テクノロジーオブジェクトのリセット」を参照)。

- オンラインで変更可能、直ちに有効化

これらのコンフィグレーションデータは、ユーザープログラムからアクセスする変数により変更できます。変更は直ちに有効になります。

注記

ONLINEで変更されたコンフィグレーションデータは、[Copy current data to RAM]とそれに続く[Copy from RAM to ROM]を使用して保存すること、または[Load configuration data to PG]を使用して保存することができます。

「対象システムでのストレージコンセプト (ページ 41)」を参照してください。

コンフィグレーションデータの読み取り

テクノロジーオブジェクトの各コンフィグレーションデータは、以下の例のように、読み取りおよび変数に割り当てることができます。

- SIMOTIONデバイスのRAMに保存されている値

これには、構文 *TO-name.setconfigdata.config-date* を使用します。

- テクノロジーオブジェクト上で現在有効な値

これには、構文 *TO-name.activeconfigdata.config-date* を使用します。

以下の表にある例を参照してください。

コンフィグレーションデータの個々の変数へのアクセスのみが許可されます。

表 3-19 テクノロジーオブジェクトのコンフィグレーションデータ項目への読み取りアクセス

```

VAR
    lreal_var : LREAL;
    drive_var : driveaxis;
END_VAR

lreal_var :=
    drive_var.setconfigdata.TypeOfAxis.MaxJerk.maximum;
    // 保存されている値への読み取りアクセス
lreal_var :=
    drive_var.activeconfigdata.TypeOfAxis.MaxJerk.maximum;
    // 現在有効になっている値への読み取りアクセス

```

コンフィグレーションデータ項目は、以下の2つの方法で変数に割り当てることができます。

- 前出の表にある例のように値を割り当てる方法(『STプログラミングマニュアル』の「**値の割り当て**」も参照)。
この場合は、エラーのイベントで *ExecutionFaultTask* が呼び出されます。
エラー応答の詳細については、「**システム変数へのアクセス**」を参照してください。
- *_getSafeValue* システムファンクションを使用する方法(「*_getSafeValue* (ページ 334)」ファンクションを参照)。
この場合、エラーのイベントに必要な応答を得るようにプログラムできます。

式での使用、またはファンクションやファンクションブロックでのパラメータとしての使用が可能です。この場合は、エラーのイベントで *ExecutionFaultTask* が呼び出されます。

ランタイムにおけるコンフィグレーションデータの変更(オンライン変更)

コンフィグレーションデータは、RAMに保存されている値への単純な変数書き込みアクセスによりオンラインで変更できます。これには、構文 *TO-name.setconfigdata.config-date* を使用します。

有効性(テクノロジーオブジェクト上で有効な現在値としての採用)は、コンフィグレーションデータ項目により決められます(直ちに有効化 / 再起動後に有効化)。

「**コンフィグレーションデータ項目への書き込みアクセス**」にある例を参照してください。
コンフィグレーションデータの個々の変数へのアクセスのみが許可されます。

表 3-20 テクノロジーオブジェクトのコンフィグレーションデータ項目への書き込みアクセス

```

VAR
    lreal_var : LREAL;
    drive_var : driveaxis;
END_VAR

drive_var.setconfigdata.TypeOfAxis.MaxJerk.maximum :=
    200000.0;
    // 保存されている値への書き込みアクセス

```

コンフィグレーションデータ項目には、以下の2つの方法で値を割り当てることができます。

- 前出の表にある例のように値を割り当てる方法(『STプログラミングマニュアル』の「**値の割り当て**」も参照)。
この場合は、エラーのイベントで *ExecutionFaultTask* が呼び出されます。
エラー応答の詳細については、「**システム変数へのアクセス**」を参照してください。
- システムファンクション *_setSafeValue* を使用する方法(「*_setSafeValue* (ページ 337)」ファンクションを参照)。
この場合、エラーのイベントで必要な応答を得るようにプログラムできます。

さらに、システム変数を介して有効化を制御するオプションもあります。

テクノロジーオブジェクトシステム変数 *activationModeChangedConfigData* を使用して、変更されたコンフィグレーションデータをいつ有効にするかを指定します。

- *activationModeChangedConfigData* = *ACTIVATE_CHANGED_CONFIG_DATA* を設定すると、変更されたデータは直ちに有効になります。
また、システム変数をこの値に設定すると、この時点までに収集されたデータも有効になります。
- *activationModeChangedConfigData* = *COLLECT_CHANGED_CONFIG_DATA* を設定すると、変更されたデータが収集されます。

この場合、システム変数 *ACTIVATE_CHANGED_CONFIG_DATA* に設定すると、収集されたデータ全体が直ちに有効になります。

収集された変更後のコンフィグレーションデータは、対応するテクノロジーオブジェクトシステムファンクション(たとえば、*_resetAxisConfigDataBuffer*)を呼び出すことで削除(有効化せずに)できます。

通知

activationModeChangedConfigData = *ACTIVATE_CHANGED_CONFIG_DATA* の場合、変更されたコンフィグレーションデータ項目が書き込まれてから有効になるまである程度時間がかかります。

特に、同時に複数のコンフィグレーションデータが変更されると、タスク内でタイムアウトが発生する可能性があります。
コンフィグレーションデータを *SynchronousTasks* 内で変更することはできません。

注記

複数のコンフィグレーションデータを同時に変更する必要がある場合は、先に *activationModeChangedConfigData* = *COLLECT_CHANGED_CONFIG_DATA* で収集を行ってから、*ACTIVATE_CHANGED_CONFIG_DATA* 設定を使用して逐次的タスク内で収集したデータ全体を有効にすることを推奨します。

ランタイムに変更されたコンフィグレーションデータは、メモリカードおよびESプロジェクトに保存できます(「対象システムでのストレージコンセプト (ページ 41)」を参照)。

注記

ユーザープログラムからコンフィグレーションデータにアクセスする場合(たとえば、一般的なFBで)、TOタイプによっては個別のコンフィグレーションデータも使用できます。

TO参照および軸配列用のコンフィグレーションデータへのアクセス

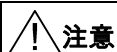
TO参照用のコンフィグレーションデータにアクセスできます。
 軸配列のコンフィグレーションデータにアクセスするには、中間媒介変数を使用する必要があります。
 軸配列を直接使用しようとすると中間媒介変数を使用する必要があることを示すコンパイルメッセージを受け取ります。

例

```
Vartemp
  myPosAxis   :posaxis;
End_Var
myPosAxis     :=Pos[2]

myPosAxis.setconfigdata.TypeOfAxis.MaxJerk.maximum :=
  200000.0;
```

3.2.10 テクノロジーオブジェクトのリセット



注意

テクノロジーオブジェクトをリセットすると、エラーメッセージなしで現在の動作が中断されます。

テクノロジーオブジェクトの再起動を必要とするコンフィグレーションデータを統合するには、テクノロジーオブジェクトをリセットする必要があります。

手順は、テクノロジーオブジェクトの *restart.restartActivationSetting* コンフィグレーションデータ項目により異なります。

- *restart.restartActivationSetting* = RESTART_BY_COMMANDを設定

テクノロジーオブジェクトは、対応するシステムファンクション(たとえば、*_restartAxis*)を使用した場合にのみ再起動できます。再起動するには、パラメータ *activateRestart* = ACTIVATE_RESTARTを設定します。

- *restart.restartActivationSetting* = RESTART_BY_SYSVAR_AND_COMMANDを設定

再起動は、以下の2つの方法で実行できます

- 対応するシステムファンクションを呼び出す方法(たとえば、*_restartAxis*)。パラメータ *activateRestart* = ACTIVATE_RESTARTを設定する方法。
- テクノロジーオブジェクトシステム変数 *restartActivation* = ACTIVATE_RESTARTに値を割り当てる方法。
システム変数は初期化され、テクノロジーオブジェクトにあるすべてのステータス情報(原点復帰した軸など)は失われます。

再起動は、常に非同期で実行されます。

テクノロジーオブジェクトの再移動が正常に完了すると、このシステム変数の値は NO_RESTART_ACTIVATION になります。

3.2.11 テクノロジーオブジェクトのライブラリでの使用

ライブラリには、TOファンクションと、テクノロジーオブジェクトのシステム変数へのアクセスを含めることができます。

ライブラリがコンパイルされるSIMOTIONデバイスとテクノロジーパッケージを、ライブラリのオブジェクトプロパティで指定します。

1. プロジェクトナビゲータで、ライブラリを選択します。
2. [Edit | Object Properties]メニューコマンドを選択します。
3. [TPs/TOs]タブを選択します。
4. ライブラリがコンパイルされるSIMOTIONデバイス(バージョン番号を含む)とテクノロジーパッケージを選択します。

通知

エラーを出さずにプロジェクトをコンパイルするために、以下の表に記載されているSIMOTIONデバイスとテクノロジーパッケージの選択に関する規則に従ってください。

また、必要であれば、ライブラリのSTソースファイル内でテクノロジーパッケージを指定できます(USEPACKAGEコマンドを使用)。ただし、これは必須ではありません。

表 3-21 ライブラリでのデバイスおよびテクノロジーパッケージの選択

選択	説明
デバイスに非依存	以下の選択も行う必要があります。 • テクノロジーパッケージ • 選択したテクノロジーパッケージのバージョン番号 注: 1. ライブラリは、SIMOTIONデバイスまたはSIMOTION Kernelのバージョンを参照しないでコンパイルされます。 このため、以下のものは使用できません。 – SIMOTIONデバイスのシステムファンクション – SIMOTIONデバイスのシステム変数 – バージョン依存のシステムファンクション – テクノロジーオブジェクトのコンフィグレーションデータ 2. ライブラリは、選択したバージョンに正確に対応するようにコンパイルされます。 選択したバージョンで利用できないシステムファンクションまたは変数を使用すると、コンパイルエラーが発生します。 3. デバイス非依存のライブラリを別のバージョンでできるようにする場合は、ライブラリをコピーして別の名前で挿入します。 このコピーは、別のバージョン参照を使用して再コンパイルする必要があります。

選択	説明
バージョンを含むSIMOTIONデバイス(複数選択も可能)	選択したデバイスのすべてで利用できるテクノロジーパッケージのみが表示されます。 注: 1. ライブラリは、選択したデバイスとテクノロジーパッケージ(選択したデバイスバージョンの)のすべてに対してコンパイルされます。 2. 選択したデバイスの1つまたは個別のデバイスバージョンのテクノロジーパッケージで利用できないシステムファンクションや変数を使用すると、コンパイルエラーが発生します。 3. ライブラリは、選択したデバイスとテクノロジーパッケージにのみ使用できます。 ライブラリをSTソースファイル内で使用する場合は、以下のことがチェックされます。 - ライブラリが、インポートしたSTソースファイルを収容しているSIMOTIONデバイス(バージョンを含む)用にコンパイルされているかどうか。 - テクノロジーパッケージがSIMOTIONデバイス上に設定されているかどうか。およびライブラリ内のSTソースファイルでUSEPACKAGEコマンドを使用してテクノロジーパッケージが指定されているかどうか。 なんらかの矛盾があると、コンパイルエラーが発生します。

3.3 障害およびイベントへの応答

3.3.1 障害およびイベントの評価

SIMOTIONシステムには、スケジューリングプログラム用のさまざまな実行レベルがあります。これらの実行レベルの1つは割り込み実行レベルで、これはエラーなどの特定のイベントへの応答の中で起動されます。

障害およびイベントへの応答に関するオプション

割り込み実行レベルでタスクを起動できるイベントには、2種類あります。

- イベントがユーザー定義の場合、これらはUserInterruptTasksタスクと呼ばれ、またこれらのタスクに割り当てられるプログラムもユーザー定義になります。
- イベントがシステムまたはテクノロジーに関連したイベント(システムエラーまたはテクノロジーオブジェクト)の場合は、SystemInterruptTasksと呼ばれます。以下の表に、使用可能なSystemInterruptTasksの一覧を示します。

システムにより生成されるメッセージは、アラームと呼ばれます。この反応は、割り込み設定で定義されます(以下を参照)。

表 3-22 指定されたSystemInterruptTasks

SystemInterruptTask	以下のイベントの場合に呼び出し
ExecutionFaultTask	プログラムでの実行エラー
PeripheralFaultTask	I/Oモジュールからのプロセスアラームと診断アラーム
TechnologicalFaultTask	テクノロジーオブジェクトからのアラーム、警告、および注意
TimeFaultBackgroundTask	BackgroundTaskのタイムアウト
TimeFaultTask	TimerInterruptTasksのタイムアウト

注記

メッセージ生成は、別のシステムフィードバックです。特定のイベント(タンクが空など)が発生した場合などに、ユーザーが割り当てたメッセージをプログラムで使用できます。詳細については、「プログラミングメッセージ(ページ 378)」を参照してください。

アラーム設定

テクノロジカルアラームのシステム動作をSIMOTION SCOUTで定義できます(アラーム設定)。以下のシステム動作を選択できます。

- STOP: STOPモードへ移行(すべてのシステムおよびユーザータスクが停止します)。
- STOP U: STOP Uモードへの移行(ユーザープログラムタスクのみが停止します)。
- START TechnologicalFaultTask: 関連するSystemInterruptTaskを起動します。
- NONE: 応答なし

各アラームには、デフォルトの応答があります。

アラーム設定の詳細については、「テクノロジーオブジェクトのエラー処理」に記載されています。

START

TechnologicalFaultTaskを選択した場合は、関連付けられているアラームに応答するTechnologicalFaultTaskにプログラムを割り当てる必要があります。

アラームは実行内容をプログラムして応答を設定できるだけでなく、テクノロジーオブジェクトに応答させることもできます(SIMOTIONモーションコントロールテクノロジーオブジェクトの機能マニュアルを参照)。

3.3.2 プログラムでの実行エラー

タスク設定のプログラムで、実行エラーへの応答を設定します(「実行システムの設定」を参照)。

これは、次のような操作に作用します。

- 不動小数点数に関する不正な操作(負数の対数など)。
- 不正なタイプ変換
- 0での除算
- 配列の制限違反

次の表に発生しうるエラー応答を示します。

表 3-23 プログラムで実行エラーが発生した場合のエラー対応

エラー対応	タスクタイプ	説明
CPU to STOP	すべてのタスク	SIMOTIONデバイスはSTOPモードに切り替わり、ShutdownTaskが起動します。
ExecutionFaultTask	シーケンシャル(非周期)	ExecutionFaultTaskが起動されます。 エラーが発生したタスクは中止されます。中止されたタスクは、ユーザーが再度使用することができます。
	サイクリック	ExecutionFaultTaskが起動されます。 エラーが発生したタスクは中止されます。 ExecutionFaultTaskが終了すると、SIMOTIONデバイスはSTOPモードに切り替えられます。ShutdownTaskが起動されます。

下記も参照

コンフィグレーション時の設定 (ページ 229)

3.3.3 システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー

このセクションでは、通常の方法で(式の変数の識別子または変数割り当てを使用して)システム変数、コンフィグレーションデータ、またはI/O変数にアクセスしたときにエラーが発生した場合の応答について説明します。

- システム変数およびコンフィグレーションデータのエラー:

ExecutionFaultTaskが起動されます。さらなるエラー応答はエラーが発生するタスクの種類(シーケンシャルまたは周期的)により異なります(次の表を参照)。

表 3-24

システム変数およびコンフィグレーションデータに不正にアクセスした際のExecutionFaultTaskの起動時の応答

タスクタイプ	説明
シーケンシャル(非周期)	ExecutionFaultTaskが起動されます。 エラーが発生したタスクは中止されます。中止されたタスクは、ユーザーが再度使用することができます。
サイクリック	ExecutionFaultTaskが起動されます。 エラーが発生したタスクは中止されます。 ExecutionFaultTaskが終了すると、SIMOTIONデバイスはSTOPモードに切り替えられます。ShutdownTaskが起動されます。

- I/O変数のエラー(入出力への直接アクセス)

エラー応答は、I/O変数を定義するときに指定します(『STプログラミングマニュアル』の「**周期的タスクの直接アクセスおよびプロセスイメージ**」を参照してください)。

- CPU停止: ExecutionFaultTaskが起動されます。
SIMOTIONデバイスはSTOPモードに切り替わり、ShutdownTaskが起動します。
- 置換値
I/O変数が定義したときに指定された置換値が使用され、タスクが続行されます。
- 最新値

読み出しアクセス(入力部または出力部への)では
最新値が適用され、タスクが続行されます。

書き込みアクセス(出力部への)では 値が変数に書き込まれます。
ただし、出力が再度使用できるようになるまで、出力部で有効化されません。
タスクは続行されます。

特定のケースでは、エラーに別の応答をしたり、定義されたエラー応答と異なる応答をする必要があります。ファンクション_getSafeValue、_setSafeValue、およびgetInOutByteは、そのために使用します。

ファンクション_getSafeValueおよび_setSafeValueは非常に実行時間がかかります。

注記

RESTART時のシステム変数の最新値や置換値を読み取ることもできます。「システム変数(ページ 83)」を参照してください。

下記も参照

[_getSafeValue]ファンクション (ページ 334)

[_setSafeValue]ファンクション (ページ 337)

[_getInOutByte]ファンクション (ページ 341)

システム変数および入/出力へのアクセスの概要 (ページ 334)

3.3.4 プロセスイメージの生成時のエラー

このセクションでは、割り当てられたタスクのプロセスイメージの更新中にエラーが発生したときの応答について説明します。

- 周期的タスクのプロセスイメージの場合(『STプログラミングマニュアル』の「**周期的タスクの直接アクセスおよびプロセスイメージ**」を参照してください)

エラー応答は、I/O変数を定義するときに指定します。

- CPU停止: 応答の詳細は次の表を参照して下さい。

- 置換値

I/O変数を定義したときに指定された置換値が使用され、周期的タスクが続行されます。

- 最新値

プロセス入力イメージ(入力の読み取り):

そのアドレスのプロセスイメージの値は変更されません。周期的タスクは続行されます。

プロセス出力イメージ(出力への書き込み):

出力が再度使用可能になったときにのみ、そのアドレスの出力でこの値が有効になり、周期的タスクは続行されます。

- BackgroundTaskの固定プロセスイメージの場合(『STプログラミングマニュアル』の「**BackgroundTaskの固定プロセスイメージへのアクセス**」を参照してください)

エラー応答は、直接アクセスがI/O変数を使用して同じアドレスに定義されているかどうかにより異なります。

- 直接アクセスの定義なし: エラー応答は、必ずCPU

STOPとなります。応答については、次の表を参照してください。

- 直接アクセスの定義あり:

I/O変数の定義で指定されたエラー処理が適用されます(上の周期的タスクのプロセスイメージなどを参照)。

表 3-25 プロセスイメージ更新時のCPU STOP応答のエラー応答

イベント	説明
エラーの発生	- 受信メッセージが一度生成されます。 - PeripheralFaultTaskにリンクするプログラムがない場合、SIMOTIONデバイスはSTOPモードに移行し、ShutdownTaskが起動されます。 - それ以外の場合: - PeripheralFaultTaskが、すぐに一度起動されます(次のIPOサイクルクロックではなく): <code>TSI#interruptId = _SC_IMAGE_UPDATE_FAILED</code> <code>TSI#logBaseAdrIn</code>または <code>TSI#logBaseAdrOut</code>には、エラーが発生したアドレスが含まれます。 - プロセス入カイメージ: そのアドレスでのプロセスイメージの値は変更されません。 プロセス出カイメージ: 出力が使用できるようになるまで、そのアドレスの出力で値が有効になりません。 - エラーが発生した周期的タスクは継続されます。
エラーの再発	- 追加のメッセージは生成されません。 - PeripheralFaultTaskは再起動されません。 - プロセス入カイメージ: そのアドレスでのプロセスイメージの値は変更されません。 プロセス出カイメージ: 出力が使用できるようになるまで、そのアドレスの出力で値が有効になりません。
エラーの消滅	- 送信メッセージが一度生成されます。 - PeripheralFaultTaskが、すぐに一度起動されます(次のIPOサイクルクロックではなく): <code>TSI#interruptId = _SC_IMAGE_UPDATE_OK</code> - 周期的タスクは続行されます。

3.3.5 Taskstartinfoの使用

次のようなタスクの起動に関する重要な情報が、各タスクのTaskstartinfoに格納されます。

- タスクの起動時刻
- TechnologicalFaultTaskの場合:
テクノロジーオブジェクトのトリガインスタンスおよびアラーム番号
- TimeFaultTaskの場合: タイムアウトエラーの原因となったTimerInterruptTask。

タスク内で、このタスクの関連TaskStartInfoを検索することができます。

そのためには、`TSI#<info>`システム変数を使用します。ここで、`<info>`はクエリーするための特定情報を示します。

タスク起動情報および関連するシステム変数の内容と範囲は、関連するタスクにより異なります(次の表を参照)。

Taskstartinfoのクエリーは、通常、SystemInterruptTaskで使用されます(TechnologicalFaultTaskのTaskstartinfoのクエリーやTimeFaultTaskのTaskstartinfoのクエリーの例を参照)。

表 3-26 ユーザープログラムタスクのTaskstartinfo

タスク	Taskstartinfo			重要度
	説明	:	データタイプ	
StartupTask				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)
MotionTasks				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)
BackgroundTask				
	TSI#startTime	:	DT	サイクル制御ポイントの時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	コンフィグレーションされたタスクのサイクルタイム(= 0、タスクは非同期的で非周期的であるため)
TimerInterruptTasks				
	TSI#startTime	:	DT	サイクル制御ポイントの時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム
SynchronousTasks				
	TSI#startTime	:	DT	タスクの起動時刻(=タスクが同期して実行されるサイクルクロック)
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム - ServoSynchronousTask:位置制御サイクル - IPOsynchronousTask: 補間サイクルクロック(IPO) - IPOsynchronousTask_2: 補間サイクルクロック(IPO_2) - PWMsynchronousTask: パルス幅変調サイクルクロック - InputSynchronousTask_1: Input1サイクルクロック - InputSynchronousTask_2: Input2サイクルクロック - PostControlTask_1: Control1サイクルクロック - PostControlTask_2: Control2サイクルクロック
TimeFaultTask				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)
	TSI#interruptId	:	UDINT	トリガする事象: _SC_CYCLE_TIMER_OVERFLOW (= 301) 現時点では、追加の事象は定義されていません。
	TSI#taskId	:	StructTaskId	時間監視機能で応答したTimerInterruptTaskのTaskId。

表 3-27 ユーザープログラムタスクのTaskstartinfo (継続)

タスク	Taskstartinfo			重要度
	説明	:	データタイプ	
TimeFaultBackgroundTask				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(=0、タスクがシーケンシャルのため)
	TSI#interruptId	:	UDINT	トリガする事象: • _SC_BACKGROUND_TIMER_OVERFLOW (= 300) 現時点では、追加の事象は定義されていません。
TechnologicalFaultTask				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(=0、タスクがシーケンシャルのため)
	TSI#alarmNumber	:	DINT	トリガされたアラームの番号(『SIMOTIONアラーム診断マニュアル』の説明を参照) アラームメッセージのパラメータ出力は TSI#alarmP1_DINT ~ TSI#alarmP5_LREAL で使用可能です。(たとえば、TSI#alarmP3_UDINT はデータタイプ UDINT のパラメータ3)。
	TSI#toInst	:	ANYOBJECT	エラーの原因のTOインスタンスは、AnyObject_to_Object フังก์ションを使用して変換することができます。 テクノロジーオブジェクトのインスタンスと比較することもできます(TechnologicalFaultTaskのTaskstartinfoのクエリーの例を参照)。
	TSI#commandId.low	:	UDINT	トリガするコマンドのCommandId(下位ワード)
	TSI#commandId.high	:	UDINT	トリガするコマンドのCommandId(上位ワード)
	TSI#alarmP1_DINT TSI#alarmP1_UDINT TSI#alarmP1_LREAL	: : :	DINT UDINT LREAL	各データタイプのトリガされたアラームのメッセージのパラメータ1~5 (補助値)。 例: TSI#alarmP3_UDINT にはUDINTデータタイプのパラメータ3が含まれます)。 パラメータの意味は、『SIMOTIONアラーム診断マニュアル』のアラームの説明を参照してください。 ここでは、次の構文を持つパラメータの番号とデータタイプについて説明されています。 /n/%A 位置: • /n/: パラメータの番号 • %A: データタイプの略称 - %dDINT
	TSI#alarmP2_DINT TSI#alarmP2_UDINT TSI#alarmP2_LREAL	: : :	DINT UDINT LREAL	
	TSI#alarmP3_DINT TSI#alarmP3_UDINT TSI#alarmP3_LREAL	: : :	DINT UDINT LREAL	
	TSI#alarmP4_DINT TSI#alarmP4_UDINT TSI#alarmP4_LREAL	: : :	DINT UDINT LREAL	

3.3 障害およびイベントへの応答

	TSl#alarmP5_DINT	:	DINT	- %X: UDINT
	TSl#alarmP5_UDINT	:	UDINT	- %If:LREAL
	TSl#alarmP5_LREAL	:	LREAL	例: /3/%XはUDINTデータタイプを持つパラメータ3を意味します。 『SIMOTIONアラーム診断マニュアル』で説明されているトリガされるアラームのパラメータは、使用可能なもののみ公開されています。

表 3-28 ユーザープログラムタスクのTaskstartinfo (継続)

タスク	Taskstartinfo			重要度
	説明	:	データタイプ	
ExecutionFaultTask				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)

	TSI#executionFault Type	:	UDINT	ユーザープログラムの実行エラーのタイプ • <code>_SC_DIVISION_BY_ZERO</code> (= 500) 整数の除算またはモジュール生成時のエラー(データタイプANY_INT): 除数が0 • <code>_SC_INVALID_FLOATING_POINT_OPERATION</code> (= 501) 浮動小数点数の処理時のエラー: - 加算: 両方の加数が無限大ですが、符号が異なります - 減算: 減数と被減数が無限大ですが、符号が同じです - 乗算: 片方の因子が0で、もう一方が無限大です - 除算: 除数が0、または両方の因子が無限大です - STシステムファンクションSQRT: 被開法数($\sqrt{\quad}$内の数)が0未満です - STシステムファンクションLNおよびLOG: 引数≤ 0です。 - STシステムファンクションSIN、COS、およびTAN: 引数が無限大です - STシステムファンクションASINおよびACOS: 引数>1です - STシステムファンクションEXP、EXPD、または演算子<code>***</code>: 底≤ 0です • <code>_SC_ARRAY_BOUND_ERROR_READ</code> (= 502) 読み取りアクセス時に配列の限界を超えました • <code>_SC_ARRAY_BOUND_ERROR_WRITE</code> (= 503) 書き込みアクセス時に配列の限界を超えました • <code>_SC_VARIABLE_ACCESS_ERROR_READ</code> (= 504) 次の読み取りアクセス時のエラー - テクノロジーオブジェクトのシステム変数へ: - 配列の制限違反 - テクノロジーオブジェクトのリセット(RESET)中のアクセス - TO#NILの値を持つテクノロジーオブジェクトデータタイプ変数へのアクセス - 直接アクセスによるI/O変数へのアクセス • <code>_SC_VARIABLE_ACCESS_ERROR_WRITE</code> (= 505) 次の書き込みアクセス時のエラー - テクノロジーオブジェクトのシステム変数へ: 配列の限界を超えました。 テクノロジーオブジェクトのリセット(RESET)中のアクセス。 TO#NIL値を持つテクノロジーオブジェクトデータタイプの変数へのアクセス。 直接アクセスを使用したI/O変数へのアクセス。 • <code>_SC_TO_INSTANCE_NOT_EXISTENT</code> (= 506) 存在しないTOインスタンスへのアクセス。 TO#NILの値を持つテクノロジーオブジェクトデータタイプ変数がTOファンクションで使用されています。
	TSI#taskId	:	StructTaskId	実行エラーが発生したタスクのTaskId。

表 3-29 ユーザープログラムタスクのTaskstartinfo (継続)

タスク	Taskstartinfo		重要度
	説明	: データタイプ	

PeripheralFaultTask				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)
	TSI#interruptId	:	UDINT	• _SC_PROCESS_INTERRUPT (= 200) プロセス割り込みがI/Oモジュールで発生しました 次のTSIの追加情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut - TSI#details - TSI#eventClass - TSI#faultId • _SC_DIAGNOSTIC_INTERRUPT (= 201) 診断割り込みがI/Oモジュールで発生しました 次のTSIの追加情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut - TSI#logDiagAdri - TSI#details - TSI#eventClass - TSI#faultId • _SC_STATION_DISCONNECTED (= 202) PROFIBUS DP: DPスレーブステーションで障害が発生しました PROFINET IO: IOデバイスのステーションの障害 次のTSIの追加情報 - TSI#logDiagAdri - TSI#eventClass - TSI#faultId • _SC_STATION_RECONNECTED (= 203) PROFIBUS: DPスレーブのステーションの再接続 PROFINET IO: IOデバイスのステーションの戻り 次のTSIの追加情報 - TSI#logDiagAdri - TSI#eventClass - TSI#faultId • _SC_IMAGE_UPDATE_FAILED (= 204) プロセスイメージの更新時のエラー(DPスレーブ: ステーションの障害と関連) 次のTSIの追加情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut - TSI#logDiagAdri • _SC_PC_INTERNAL_FAILURE (= 205) ローカルコントローラモジュールのエラー 次のTSIの追加情報 - TSI#details

表 3-30 ユーザープログラムタスクのTaskstartinfo (継続)

タスク	Taskstartinfo		重要度
	説明	: データタイプ	
PeripheralFaultTask (継続)			
	TSI#interruptId (つづき)	: UDINT	• _SC_IMAGE_UPDATE_OK (= 206) プロセスイメージの更新は再び機能するようになりました(DPスレープ: ステーションの回復と関連) 次のTSIの追加情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut - TSI#logDiagAdri • _SC_DP_CLOCK_DETECTED (= 207) 最初のクロック信号が検出され、有効なPRMメッセージフレームが受信されます • _SC_DP_SYNCHRONIZATION_LOST (= 208) 複数のサイクルの失敗またはPLLがスリップしました(内部DP_INTERFACES_SYNCHRONIZEDステータス)。 PLLは非管理モードに切り替わります • _SC_DP_SLAVE_SYNCHRONIZED (= 209) PLLは同期操作にスリップしました • _SC_DP_SLAVE_NOT_SYNCHRONIZED (= 210) 複数のサイクルの失敗またはPLLがスリップしました(内部DP_SLAVE_SYNCHRONIZEDステータス)。 PLLは管理モードのままです • _SC_BATTERY_WARNING_LEVEL1 (= 211) D435 V3.0のみ: バッテリー電圧がプレアラームレベルを下回りました。 • _SC_BATTERY_WARNING_LEVEL2 (= 212) D435 V3.0のみ: バッテリー電圧が警告レベルを下回りました。オフに切り替えると、スーパーキャップのみ定義されたバックアップ期間バックアップリングされます。 • _SC_BATTERY_NOT_PRESENT (=213) バッテリーモジュールが存在しないか取り外されています。 • _SC_IO_MODULE_SYNCHRONIZED (= 214) TM15 / TM17 High Feature: 動機制御が成功しました。 次のTSIの詳細情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) TM15 / TM17 High Feature: 動機制御が失敗しました。 次のTSIの詳細情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut • _SC_PULL_PLUG_INTERRUPT (= 216) PROFINET IO: IOデバイスのモジュールの挿入または取り外し。 次のTSIの詳細情報 - TSI#logBaseAdriIn - TSI#logBaseAdriOut - TSI#eventClass - TSI#faultId

3.3 障害およびイベントへの応答

	TSI#logBaseAdrIn	:	DINT	割り込みの原因がモジュールの入力領域にある場合の次の <i>TSI#InterruptId</i> の論理ベースアドレス • _SC_PROCESS_INTERRUPT (= 200) • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) Otherwise _SC_INVALID_ADDRESS (= -1)

ユーザープログラムタスクのTaskstartinfo (継続)

タスク	Taskstartinfo			重要度
	説明	:	データタイプ	
PeripheralFaultTask (継続)				
	TSI#logBaseAdrOut	:	DINT	割り込みの原因がモジュールの出力領域にある場合の次の TSI#InterruptId の論理ベースアドレス • _SC_PROCESS_INTERRUPT (= 200) • _SC_DIAGNOSTIC_INTERRUPT (= 201) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) • _SC_PULL_PLUG_INTERRUPT (= 216) • _SC_IO_MODULE_SYNCHRONIZED (= 214) • _SC_IO_MODULE_NOT_SYNCHRONIZED (= 215) Otherwise _SC_INVALID_ADDRESS (= -1)
	TSI#logDiagAdr	:	DINT	次の TSI#InterruptId の DP スレーブまたは IO デバイスの診断アドレス • _SC_STATION_DISCONNECTED (= 202) • _SC_STATION_RECONNECTED (= 203) • _SC_IMAGE_UPDATE_FAILED (= 204) • _SC_IMAGE_UPDATE_OK (= 206) Otherwise _SC_INVALID_ADDRESS (= -1)
	TSI#details	:	DWORD	TSI#InterruptId に依存する詳細情報(TSI#detailsの意味の表を参照)
	TSI#eventClass	:	UINT	TSI#InterruptId に依存するイベントクラス TSI#faultId についての説明: 表「Significance of the TSI#eventClass および TSI#faultId の意味」を参照
	TSI#faultId	:	UINT	TSI#InterruptId に依存する faultId TSI#eventClass についての説明: 表「Significance of the TSI#eventClass および TSI#faultId の意味」を参照
UserInterruptTasks				
	TSI#startTime	:	DT	タスクの起動時間
	TSI#currentTaskId	:	StructTaskId	タスクのTaskId
	TSI#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)
ShutdownTask				

	TSl#startTime	:	DT	タスクの起動時間
	TSl#currentTaskId	:	StructTaskId	タスクのTaskId
	TSl#cycleTime	:	TIME	タスクの設定されたサイクルタイム(= 0、タスクがシーケンシャルのため)
	TSl#shutDownInitiator	:	UDINT	STOPへの初期移行 • _SC_MODE_SELECTOR (= 400): モードセレクトスイッチ • _SC_DEVICE_COMMAND (= 401): ユーザープログラムのシステムファンクション • _SC_EXTERNAL_COMMAND (= 402): SIMOTION SCOUTのコマンド • _SC_EXCEPTION (= 403): 例外 • _SC_ALARM_CONFIGURATION (= 404) テクノロジカルアラームへの設定された応答 例外の例 • プログラムでの実行エラー • レベルオーバーフロー • セントラルモジュールのRUNモードでの取り外し

表 3-31 TSl#InterruptId (PeripheralFaultTask)に依存するTSl#detailsの意味

TSl#InterruptId	TSl#detailsの意味	
_SC_PROCESS_INTERRUPT (= 200)	• モジュール信号割り込みの割り込みデータ これらのデータの構造は、モジュールのマニュアルに説明があります。 • 割り込みのソースがSIMOTION Iスレーブにある場合 _sendProcessInterrupt()システムファンクションの呼び出しパラメータ	
_SC_DIAGNOSTIC_INTERRUPT (= 201)	モジュール/ステーション信号割り込みのDS0 (= DS1のバイト0~3) DS0の構造は、モジュールのマニュアルに説明があります。	
_SC_PC_INTERNAL_FAILURE (= 205)	割り込みの原因は、次の原因の値のOR演算として示されます(16進値の合計)。	
	規格値	原因
	16#00000001	ホストオペレーティングシステムの"フェイタルエラー" (Windowsのブルースクリーン- P350)。
	16#00000002	温度エラー(P350、D4xx)
	16#00000004	温度が通常に戻りました(P350、D4xx)
	16#00000008	バッテリー警告(バッテリー電圧が低下していますが、まだ足りています-P350、D4xx)
	16#00000010	バッテリー電圧が通常に戻りました(P350、D4xx)
	16#00000020	バッテリーエラー(バッテリー電圧が低すぎます-P350、D435)
	16#00000040	バッテリーモジュールが取り外されました(P350、D4xx)
	16#00000080	ファンの故障(P350、D4xx)
	16#00000100	UPSがバッファリング状態です(P350)
	16#00000200	UPSの充電不可期間が終了しました。PCを起動しています(P350)

3.3 障害およびイベントへの応答

TSI#InterruptId	TSI#detailsの意味	
	16#00000400	UPSは再び稼動しています(P350)
	16#00000800	UPS/バッテリー警告(P350)
	16#00001000	UPS/バッテリーエラー(バッテリーが動作していません- P350)
	16#00002000	UPS/バッテリー警告(P350)

)

表 3-32 TSI#InterruptIdに依存するTSI#eventClassおよびTSI#faultIdの意味(PeripheralFaultTask)

TSI#InterruptId	TSI#eventClass	TSI#faultId	バスシステム¹	重要度
_SC_PROCESS_INTERRUPT (= 200)	16#11	16#41	PROFIBUS DP PROFINET IO P-Bus	プロセス割り込み
_SC_DIAGNOSTIC_INTERRUPT (= 201)	16#39²	16#42	PROFIBUS DP PROFINET IO P-Bus	入力診断割り込み
	16#38³	16#42	PROFIBUS DP PROFINET IO P-Bus	出力診断割り込み
_SC_STATION_DISCONNECTED (= 202)	16#39²	16#C4	PROFIBUS DP	DPスレーブステーションに障害が発生しました
		16#CA	PROFINET IO	システムエラーPROFINET IO⁴
		16#CB	PROFINET IO	IOデバイスのステーションの障害
		16#CC	PROFINET IO	IOデバイスの障害。 チャンネルの診断または製造者固有の診断の保留中。
_SC_STATION_RECONNECTED (= 203)	16#38³	16#C4	PROFIBUS DP	DPスレーブステーションは再接続されました
		16#CB	PROFINET IO	IOデバイスはエラーなしで再接続されました
		16#CC	PROFINET IO	IOデバイスエラーは修正されました
		16#CD	PROFINET IO	IOデバイスは再接続されましたがエラーがありました: 設定されたコンフィグレーション<> 実際のコンフィグレーション
		16#CE	PROFINET IO	IOデバイスは再接続されましたが、モジュールのパラメータ設定時にエラーがありました:
_SC_PULL_PLUG_INTERRUPT (= 216)	16#39²	16#51	PROFINET IO	PROFINET IOモジュールが取り外されたか、処理することができません。
		16#54	PROFINET IO	PROFINET IOサブモジュールが取り外されたか、処理することができません。

TSI#InterruptId	TSI#eventClass	TSI#faultId	バスシステム ¹	重要度
	16#38 ³	16#54	PROFINET IO	PROFINET IOモジュールまたはサブモジュールが挿入されました。モジュールタイプは適合しています(実際のコンフィグレーション=設定されたコンフィグレーション)
		16#55	PROFINET IO	PROFINET IOモジュールまたはサブモジュールが挿入されましたが、モジュールタイプが不正です(実際のコンフィグレーション<>設定されたコンフィグレーション)
		16#56	PROFINET IO	PROFINET IOモジュールまたはサブモジュールが挿入されましたが、モジュールのパラメータ設定時にエラーがありました
		16#58	PROFINET IO	モジュールのIOステータスがBADからGOODに変更されました
¹ 指定されたTSI#eventClassおよびTSI#faultIdに関するTSI#InterruptIdを送信するバスシステム ² 信号入力イベント ³ 信号出力イベント ⁴ 出力イベント(TSI#eventClass = 16#38)はステーションの再接続時に既存の各ステーションに送信されます。 エラーの状態に応じて、TSI#faultIdの値16#CB、16#CD、または16#CEが表示されます。				

次に、TechnologicalFaultTaskのTaskstartinfoを使用してアラームおよびアラーム番号を初期化するTOインスタンスをクエリーする例を示します。

TO_AlarmProgプログラムをTechnologicalFaultTaskに割り当てる必要があります。

表 3-33 TechnologicalFaultTaskのTaskstartinfoにポーリングする例

```

PROGRAM TO_AlarmProg
VAR
  dintVar      : DINT;
  dtVar        : DT;
END_VAR;
dtVar:=TSI#startTime;
dintVar:=TSI#alarmNumber;
  IF TSI#toInst = axis_1 THEN      // コマンドを作成したTOから
    ; //
  END_IF;
  IF TSIalarmNumber = 30002 THEN    // アラームのトリガ
    ; //
  END_IF;
END_PROGRAM

```

次に、TimerInterruptTaskでタイムアウトエラーの原因となった

TimeFaultTaskを特定する例を示します。

そのために_checkEqualTaskファンクションを使用します。

プログラムTimeFaultProgをTimeFaultTaskに割り当てる必要があります。

表 3-34 TimeFaultTaskのTaskstartinfoのクエリーの例

```
PROGRAM TimeFaultProg
  VAR
    beq : BOOL;
  END_VAR;

  beq := _checkEqualTask(TSI#TaskID, TimerInterruptTask_1);
  // トリガするタスクがTimerInterruptTask_1の場合
  // beqはTRUE

END_PROGRAM
```

テクノロジーオブジェクトでのエラー処理

このセクションでは、SIMOTIONテクノロジーオブジェクトのエラー処理について説明します。

4.1 テクノロジーオブジェクトで発生しうるエラー

テクノロジーオブジェクトのプログラミングにおいて、次のような基本的なエラーが発生する可能性があります。

- テクノロジーオブジェクト自身では、アプリケーションやレポートの特定のイベントやステータスに必要な機能を実行できません。
→テクノロジーカルアラームが出力されます。

個々のアラームについては、『SIMOTION Reference Lists』を参照してください。

- テクノロジーオブジェクトに発行したコマンドが実行できません:
→コマンドの戻り値に、その原因に関する情報が含まれています。

コマンドの戻り値については、『SIMOTION Reference Lists』を参照してください。

- コンフィグレーションデータ、システム変数またはI/O変数へのアクセス中のエラー
コンフィグレーションデータまたは変数の読み出し中または書き出し中にエラーが発生すると、ExecutionFaultTaskが呼び出されます。

下記も参照

プロセスアラーム (ページ 110)

コマンドの戻り値 (ページ 122)

システムデータへのアクセス時のエラー (ページ 124)

4.2 プロセスアラーム

テクノロジーオブジェクトでイベント(エラー、警告)が発生したときに、そのオブジェクトは**テクノロジカルアラーム**を発行します。

システムには一度に最大160のテクノロジーオブジェクトアラームを格納できます。

その際に、TOの同一アラーム単位で1つのアラームとカウントされます。

160を超えるアラームが同時に保留になったことが原因でこのバッファがオーバーフローした場合、

システムは**メッセージバッファオーバーフロー診断**に入るとともにSTOPモードに切り替わります。ユーザーはバッファレベルを読み取ることはできません。

同一アラームが連続して発生した場合、最初のアラームの詳細情報のみが表示されます。

最初に発生したアラームがリセットされると、後続のアラームの情報が表示されます。

アラームの影響

テクノロジカルアラームにより、システムの後続の応答が発生します。

以下の区別がされます。

- 影響するテクノロジーオブジェクト自身への影響: **ローカル応答**
- 他のテクノロジーオブジェクトまたは実行システムへの影響: **グローバル応答**

各アラームにの特定の影響には、デフォルト設定があります。

ただし、これらの設定は要件に合わせて補正できます。

- **エラー有効化**を指定して、アラームを即座に有効にするか、エラーの発生が連続したときにするか、または一定時間の経過後にするかを定義できます。
- 一部のアラームを隠すことができます。これにより、不要な情報などを抑制できます。

下記も参照

ローカル応答 (ページ 110)

グローバル応答 (ページ 111)

エラーの有効化 (ページ 111)

テクノロジカルアラームの設定 (ページ 113)

テクノロジカルアラームの表示とリセット (ページ 115)

ユーザープログラムを介したリセット (ページ 116)

ユーザープログラムでの評価 (ページ 120)

4.2.1 ローカル応答

ローカル応答設定により、アラームが発生したときの関連するテクノロジーオブジェクトの応答と、このTOに対する後続のコマンドの処理方法を指定します。

アラーム応答は優先順位付けされます。

次のアラーム応答は、個々のテクノロジーオブジェクトの優先順位の低い順に示されています。任意の時点で1つの応答のみ処理できます。

これは、特定の時点で保留中のアラームの中で最も高い優先度の応答です。

アラーム応答が発生すると(NONEを除く)、コマンドデコーダは必ず停止します。

発行された後続のプログラムのコマンドは拒否されます。

アラームのグローバルエラー応答により自動的に電源投入が必要にならなければ、アラームがリセットされた後にコマンドの実行を続行することができます。

個々のテクノロジーオブジェクトおよびテクノロジーオブジェクトアラームに応じて、特定の応答オプションを選択できます。

個々のローカル応答の詳細については、各テクノロジーオブジェクトの機能マニュアルを参照してください。

4.2.2 グローバル応答

グローバル応答では、実行システムのテクノロジーオブジェクトアラームの作用を記述します。

個々のテクノロジーオブジェクトアラームに応じて、次の応答オプションを選択できます。

- **NONE**

アラームが発生したときにシステムは応答しません。

- **START TechnologicalFaultTask**

TechnologicalFaultTaskが起動されます。

タスクに割り当てられたプログラムが起動します。

これを使用して、ユーザーはTOアラームに対して固有の応答をプログラムできます。

このタスクに割り当てられたプログラムがない場合、システムはSTOPモードに切り替わります。

- **STOP**

システムはSTOPモードに切り替わります。

このモードでは、すべてのテクノロジーオブジェクトは無効になり、ユーザープログラムは実行されず、すべての出力は0になります。

ただし、すべてのシステムサービスは有効で、ユーザープログラムはダウンロードできます。

- **STOPU**

システムはSTOPUモードに切り替わります。

プログラムの実行は終了し、軸イネーブルおよび位置制御は無効になります。

テクノロジーオブジェクトは有効なままで、テストや試運転機能用にリクエストを処理できます。その他はSTOPモードと同じです。

4.2.3 エラーの有効化

エラー有効化のタイプ(TypeOfActivation)

アラームとそれに関連付けるアラーム応答は、さまざまな方法でトリガできます。

エラータイプに応じて、エラーの有効化は次のいずれかの値にプリセットされています。

表 4-1 エラー有効化のタイプ

TypeOfActivation/パラメータ	意味
ACTIVATE_IMMEDIATELY	エラーが発生すると、直ちにアラームが有効化されます。
ACTIVATE_AFTER_NTIME	エラーがn回発生すると、アラームが有効化されます。 エラーの繰り返し数nをContinueパラメータに設定します。エラーの繰り返しの表を参照してください。
ACTIVATE_AFTER_NTIME_CONSECUTIVE	アラームは時間tの経過後に有効になります。エラーは、時間tの期間、中断することなく保留される必要があります。時間はTimeパラメータで設定します。エラー応答の時間の表を参照してください。

エラーの繰り返し(Continue)

表 4-2 エラーの繰り返し

Continue/パラメータ	意味
NO_TIME	エラーが発生すると、直ちにアラームが有効化されます。
TIME_n	エラーがn回発生すると、アラームが有効化されます。

このパラメータは、**TypeOfActivation**パラメータがACTIVATE_AFTER_NTIMEに設定されている場合のみ意味を持ちます。

TIME_nは、アラームに応じて、値TIME_10、TIME_100、およびTIME_1000に設定できます。

エラー応答時間(Time)

表 4-3 エラー応答時間

Time/パラメータ	意味
NO_TIME	エラーが発生すると、直ちにアラームが有効化されます。
TIME_n	一時的に中断することなくnミリ秒間エラーが発生した場合に、アラームがトリガされます。

このパラメータは、**TypeOfActivation**パラメータがACTIVATE_AFTER_NTIME_CONSECUTIVEに設定されている場合のみ意味を持ちます。

TIME_nは、アラームに応じて、値TIME_1、TIME_10、TIME_100、およびTIME_1000に設定できます。

タイプ

Typeパラメータを設定して、特定のTOアラームを表示するかどうかを指定できます。

*Hidden*設定を選択すると、このアラームが発生したときにアラームメッセージは表示されず、エントリは診断バッファに書き込まれません。

そのため、特定のテクノロジカルアラームが頻繁に発行されたときにアラームバッファのオーバーフローを回避したり、不要な通知メッセージを抑制できます。

4.2.4 テクノロジカルアラームの設定

各アラームの個々の応答はプリセットされています。
これらのデフォルト設定内容を変更するには、次の手順を実行します。

1. プロジェクトナビゲータで[Execution System]パスを選択して、[Execution System]ウィンドウを開きます。
パス[SystemInterruptTasks|TechnologicalFaultTask]を選択して、ウィンドウの[Alarm configuration]ボタンをクリックします。

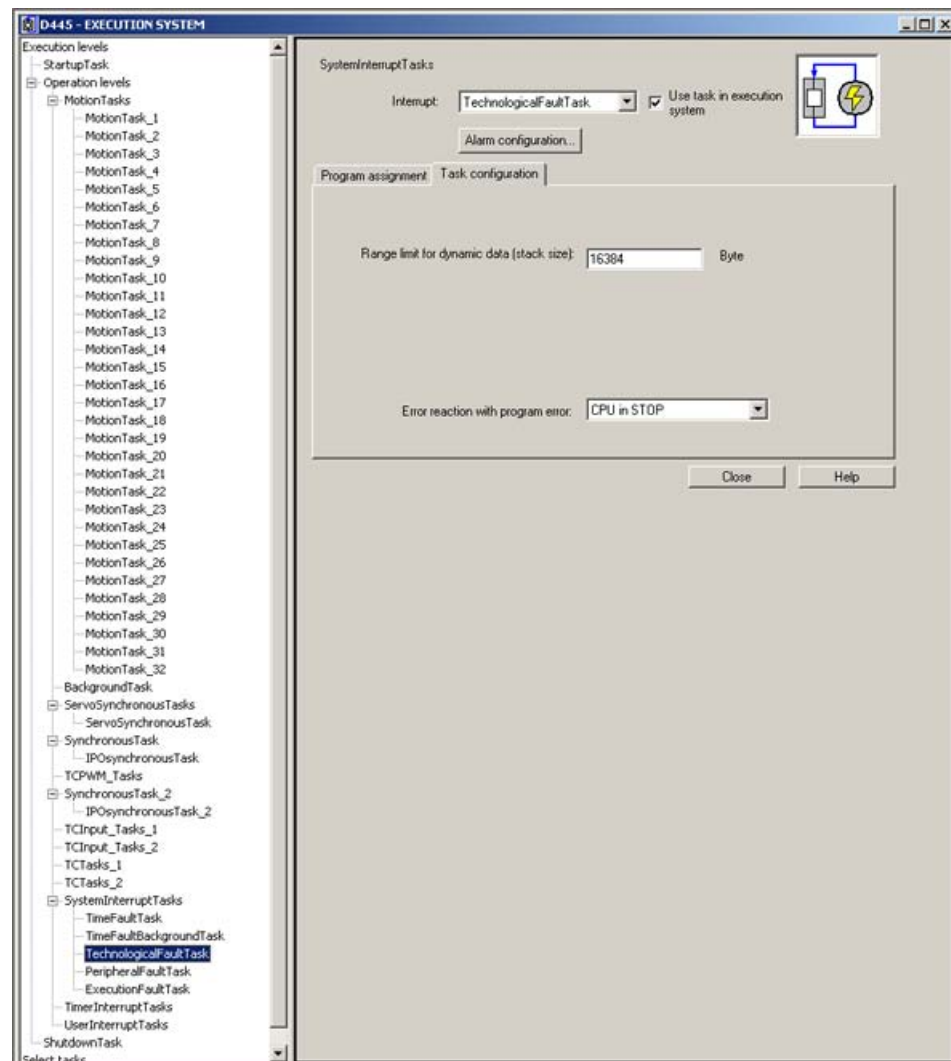


図 4-1 テクノロジアラームの設定

2. コンボボックスで、アラームを設定するテクノロジーオブジェクトを選択して、そのアラームを表示します。
3. 応答を変更するアラームを選択します。
4. 対応するテクノロジーオブジェクトアラームのリストから必要な応答を選択します。
使用可能なオプションは、アラームのタイプにより異なります。

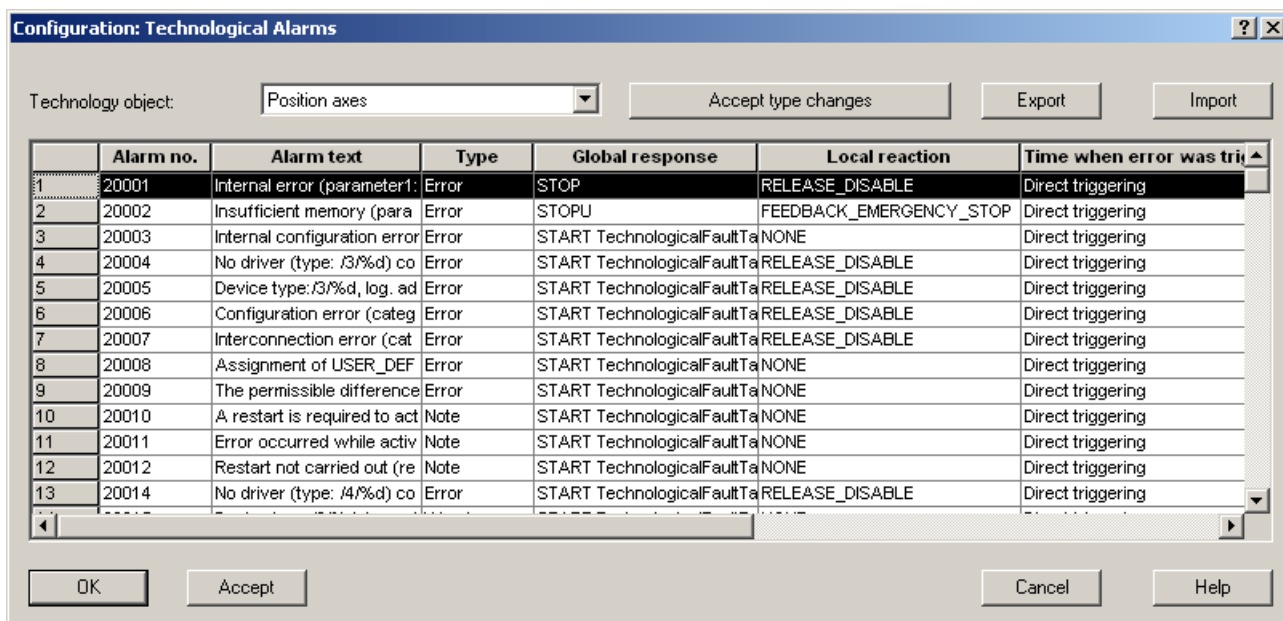


図 4-2 テクノジアラームの選択

注記

設定するアラームのテクノロジーオブジェクトは、すでに設定されている必要があります。

[Alarm

configuration]ダイアログの対応するボタンを使用して(エクスポート/インポート)、アラーム設定を他のテクノロジーオブジェクトに転送することができます。

テクノロジカルアラームのエクスポートまたはインポート

すべてのTOアラームをXMLフォーマットでエクスポートするには、次の手順を実行します。

1. **[Configuration: Technological alarms]**ダイアログを開きます。
2. **[Export]**をクリックして**[Export alarm configuration]**ダイアログを選択します。
3. エクスポートするパスを選択して、**[OK]**で確認します。
アラームは指定されたパスに保存されます。

TOアラームをXMLフォーマットでインポートするには、次の手順を実行します。

1. **[Configuration: Technological alarms]**ダイアログで、**[Import]**をクリックします。
すると、**[Import alarm configuration]**ダイアログが表示されます。
2. **[Source path and source name of the import]**の下で、目的のXMLファイルを選択します。
3. **[OK]**をクリックしてデータをインポートします。
すると、アラームがダイアログに表示されます。

特定のTOタイプのすべてのTOのメッセージの設定

1つのアラーム設定を、特定のタイプのすべてのTOに割り当てることができます。
たとえば、設定をすべての位置決め軸に割り当てられます。

1. [Configuration: Technological alarms]ダイアログを開きます。
2. アラームの設定を補正します。
少なくとも1つのアラームの設定を変更したら、[Technology object]リストのエントリはTOタイプ(位置決め軸など)ごとに展開されます。
3. リストからTOタイプを選択します。
4. [OK]をクリックすると、選択したタイプのすべてのTOにその設定が割り当てられます。

4.2.5 テクノロジカルアラームの表示とリセット

テクノロジカルアラームは、さまざまな方法で評価およびリセットできます。

- SIMOTION
SCOUTがオンラインモードである場合、ワークベンチの詳細ビューの[Alarms]タブにアラームおよびメッセージが表示されます。
[Acknowledge]により、関連するタイプのすべてのアラームは削除されます。
- マンマシンインターフェース(HMI)を介して、アラームを出力、表示、およびリセットできます。
- テクノロジーオブジェクトのすべての保留中のアラームまたは個々に選択したアラームは、ユーザープログラムを介して、クエリー、評価、およびリセットすることもできます。

SIMOTION SCOUTを介したリセット

1. 詳細ビューの[Alarms]タブのアラームを選択します
2. [Acknowledge]をクリックします。

関連するタイプのすべてのアラームは削除されます。

注記

通常、ドライブアラームではテクノロジーオブジェクトアラームも生成されるので、リセット(TO)スイッチによりドライブアラームも削除されます。
ただし、ドライブオブジェクトの原因になるものが存在していれば、新しいTOアラームは即座にトリガされます。この場合、最初にドライブアラームの原因を修正します。

HMIを介した表示およびリセット

1. ProToolまたはWinCC Flexibleを介した接続

設定されたメッセージ行またはメッセージウィンドウにアラームが表示されます。

WinCCまたはProToolデバイスはACKキーまたはユーザー設定のソフトウェアキーまたはボタンを介してリセットされます。

(詳細はWinCC flexibleの説明または ProToolの説明を参照してください)

2. OPCを介したリンク

SP4 OPC Alarms & EventsのV6.0以降では、SimaticNetでアラームを表示およびリセットできます。さらに必要に応じて診断バッファを読み出しおよびリセットできます。

(操作手順は、SIMOTION SCOUT CDドキュメントの「EthernetベースのHMIおよび診断機能」の製品情報を参照してください)

4.2.6 ユーザープログラムを介したリセット

保留されているすべてのテクノロジーオブジェクトアラームのリセット

`_resetTechnologicalErrors`→現在保留されているすべてのテクノロジーオブジェクトアラームをリセットします

ST呼び出しの例: 保留されているすべてのアラームのリセット

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;
        (* すべてのTOアラームをリセットします *)
        s_i_RetVal:= _resetTechnologicalErrors();

    END_PROGRAM
END_IMPLEMENTATION
```

MCCの呼び出し: 保留されているすべてのアラームのリセット

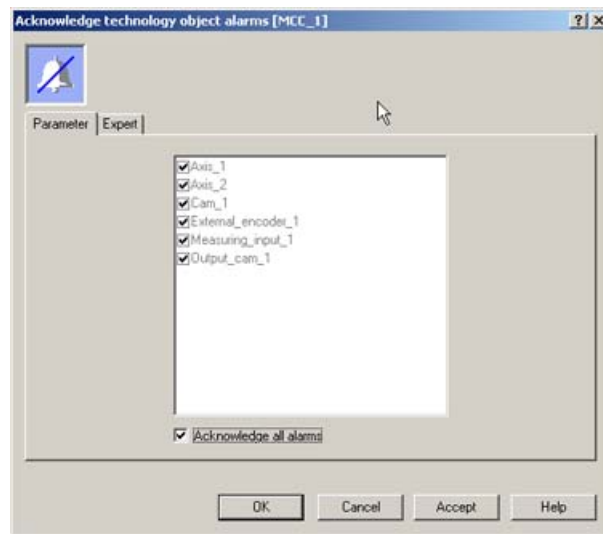


図 4-3 MCCの呼び出し: 保留されているすべてのアラームのリセット

保留されているすべてのテクノロジーオブジェクトのアラームのリセット

ST呼び出しの例: TO軸、TO measuringInput、およびTO outputCamのすべてのアラームのリセット

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END INTERFACE
IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;

        (* TOアラームをリセットします ('ResetTOAlarms') *)
        s_i_RetVal := _resetAxisError(axis:=Axis_1);
        s_i_RetVal := _resetMeasuringInputError(
            measuringInput:=MeasuringInput_1);
        s_i_RetVal := _resetOutputCamError(outputCam:=OutputCam_1);
    END_PROGRAM
END_IMPLEMENTATION
```

MCCの呼び出し: TO軸、TO measuringInput、およびTO outputCamのすべてのアラームのリセット

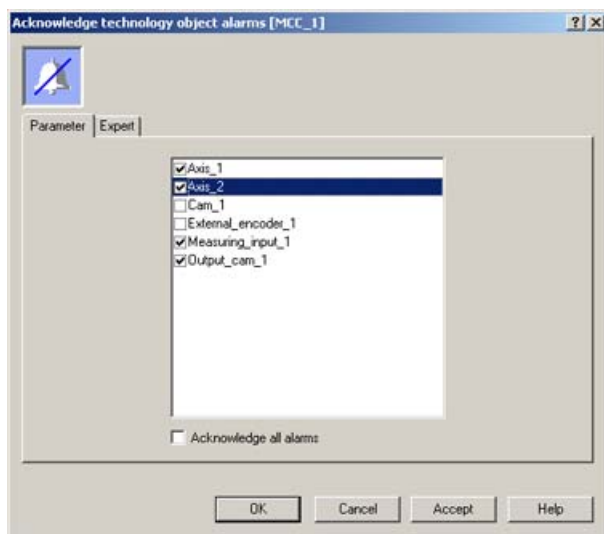


図 4-4 MCCの呼び出し: TO軸、TO measuringInput、およびTO outputCamのすべてのアラームのリセット

テクノロジーオブジェクトの特定のアラームのリセット

errorResetMode := SPECIFIC_ERRORおよび**errorNumber** := XXXXXを_reset..コマンドに追加することにより、特定のアラームを個別にリセットすることができます。

例:

_resetAxisError (... , **errorResetMode** := SPECIFIC_ERROR, **errorNumber** := 30002)

ST呼び出しの例: TO軸上のアラーム30002のリセット

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
    PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
    PROGRAM EXAMPLE
        VAR
            s_i_RetVal : DINT;
        END_VAR;
        (* 特定のTOアラームをリセットします ('ResetSingleTOAlarm') *)
        s_i_RetVal := _resetAxisError(axis:=Axis_1,
            errorResetMode:=SPECIFIC_ERROR,
            errorNumber:=30002);
    END_PROGRAM
END_IMPLEMENTATION
```

MCCの呼び出し: TO軸上のアラーム30002のリセット

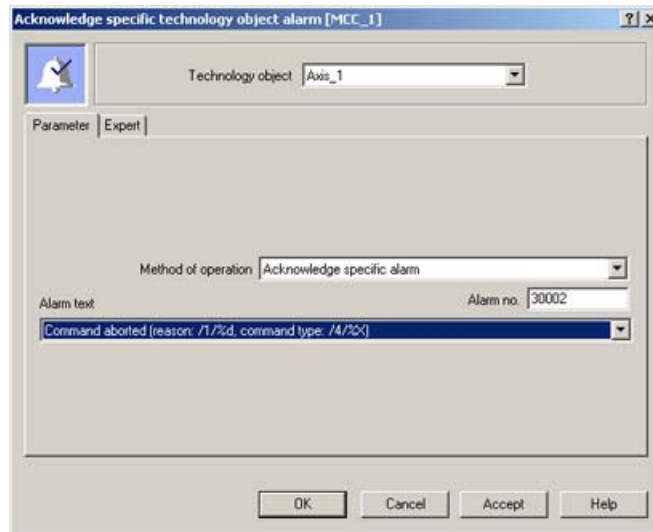


図 4-5 MCCの呼び出し: TO軸上のアラーム30002のリセット

テクノロジーオブジェクトのリセット

テクノロジーオブジェクトは初期状態に設定され、保留されているすべてのアラームはリセットされます。

注記

トラブルシューティング以外にも、リセットのコマンドはTOに別の影響があります。そのため、一般的にエラーは **_reset...Error** コマンドでリセットする必要があります。

たとえば、**_resetCam** コマンドを使用すると、カム全体が削除されます。

呼び出し例: TO軸、TO measuringInput、およびTO outputCamのリセット

```
UNIT ST_1;
INTERFACE
  USEPACKAGE CAM;
  PROGRAM EXAMPLE;
END_INTERFACE
IMPLEMENTATION
  PROGRAM EXAMPLE
    VAR
      s_i_RetVal : DINT;
    END_VAR;
    (* オブジェクトをリセットします('ResetObject') *)
    s_i_RetVal := _resetAxis(axis:=Axis_1,
      userDefaultData:=DO_NOT_CHANGE);
    s_i_RetVal := _resetMeasuringInput(
      measuringInput:=MeasuringInput_1,
      userDefaultData:=DO_NOT_CHANGE);
    s_i_RetVal:= _resetOutputCam(outputCam:=OutputCam_1,
      userDefaultData:=DO_NOT_CHANGE);
  END_PROGRAM
END_IMPLEMENTATION
```

4.2.7 ユーザープログラムでの評価

StartTechnologicalFaultTaskグローバル応答がコンフィグレーションされたアラームの場合は、アラームが発生する度に、必ず、一度、**TechnologicalFaultTask**が呼び出されます。このタスクでは、保留アラームの番号とそのアラームをトリガするテクノロジーオブジェクトをスキャンすることができます。

情報は、タスク起動情報(TSI)経由で、**TechnologicalFaultTask**に渡されます。

TechnologicalFaultTaskにはプログラムを追加することができるため、特定のアラームに対して個々のエラー応答をプログラムしたり、たとえば、アラームを傍受して高レベルのアラーム評価機能に転送することができます。

注記

TechnologicalFaultTaskにプログラムを追加していない場合は、このタスクがアラームによって呼び出されると、CPUは**STOP**モードに切り替わります。

TechnologicalFaultTaskにおける評価のために、以下のパラメータが転送されます。

TSI#startTime	→	アラームが登録された時刻
TSI#alarmNumber	→	アラーム番号
TSI#toInst	→	アラームをトリガしたテクノロジーオブジェクトの名前(たとえば、Axis_1)

プログラミング例

Axis_1でAlarm

30002が発生する度に、カウンタ(s_i_Count)が加算され、アラームが自動的に承認されます。

注:

実行システムでは、このサンプルプログラムを**TechnologicalFaultTask**に追加する必要があります!

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
    PROGRAM TO_AlarmProg;
END_INTERFACE
IMPLEMENTATION
PROGRAM TO_AlarmProg
    VAR
        s_i_Count : INT;
        s_i_RetVal: DINT;
    END_VAR;

    (*Axis_1で、Alarm 30002が保留中であるかどうかのチェック*)
    IF (TSI#alarmNumber = 30002) AND
    (TSI#toInst = Axis_1) THEN
    (*カウンタの加算*)
        s_i_Count := s_i_Count + 1;
```


(*特定のTOアラーム

('ResetSingleTOAlarms')への肯定応答*)

```
s_i_RetVal := _resetAxisError(axis:=Axis_1,  
errorResetMode:=SPECIFIC_ERROR,  
errorNumber:=30002);  
END_IF;  
END_PROGRAM  
END_IMPLEMENTATION
```

4.3 コマンドの戻り値

このユーザープログラムは、コマンドをランタイム時に正常に実行できたかどうかを示します。

すべてのエラー情報は、ブロックの戻り値に収納されます。
このエラー情報への肯定応答は不要です。

戻り値の評価

コマンドが値0を戻す場合は、コマンドがエラーなしで実行されたことを示します。
エラーが発生した場合は、戻り値には、エラーの原因の特定に使用できるエラー番号が収納されます。

各コマンドの可能なエラー番号は、SIMOTIONリファレンスリストにあります。

システムファンクション実行時のエラーについてはユーザープログラムで対処でき、派生的なエラーを回避することができます。

MCCコマンドの場合と同様に、戻り値にコマンドの戻り値を入力して、特定のエラー応答をプログラムすることができます(『MCCプログラミングマニュアル』のセクション「[Expert]タブ」を参照)。

呼び出し例

`_pos`コマンド実行時にエラーが発生すると、`g_bo_error`変数が`true`に設定され、戻り値が`g_i_errornumber`パラメータに入力された後、ブロックが中止されます。

注: 実行システムでは、このサンプルプログラムをMotionTaskに追加する必要があります。

```
UNIT ST_1;
INTERFACE
    USEPACKAGE CAM;
    VAR_GLOBAL
        g_bo_error: BOOL;
        g_i_errornumber: DINT;
        g_i_RetVal: DINT;
    END_VAR
    PROGRAM RETURN_VALUE;
END_INTERFACE
IMPLEMENTATION
    PROGRAM RETURN_VALUE

        (* Position axis ('Pos') *)
        g_i_RetVal:= _pos(axis:=Axis_1,
            direction:=SHORTEST_WAY,
            positioningMode:=ABSOLUTE,
            position:=222,
            velocityType:=DIRECT,
            velocity:=1000,
            velocityProfile:=TRAPEZOIDAL,
            blendingMode:=INACTIVE,
            mergeMode:=IMMEDIATELY,
            nextCommand:=WHEN_MOTION_DONE,
            commandId:=_getCommandId());

        (*戻り値の評価*)
        IF g_i_RetVal <> 0 THEN
```

```
        g_i_errornumber := g_i_RetVal;  
        g_bo_error := true;  
        Return;  // ブロックの終了  
    END_IF;  
    // これ以降は追加的なユーザープログラム  
END_PROGRAM  
END_IMPLEMENTATION
```

4.4 システムデータへのアクセス時のエラー

コンフィグレーションデータ、システム変数またはI/O変数へのアクセス中のエラー

コンフィグレーションデータまたは変数の読み出し中または書き出し中にエラーが発生すると、ExecutionFaultTaskが呼び出されます。

ただし、コンフィグレーションされたエラー応答と異なる応答が必要になる場合もあります。ファンクション_getSafeValueと_setSafeValueはこの目的に役立ちます。

システムデータが使用できない場合の代替値戦略

システム変数の読み取り / 書き込みで、コンフィグレーションデータ項目またはI/O変数がエラーになるさまざまなケースがあります。

表 4-4 読み取り / 書き込みエラーの可能な原因

原因	システム変数	コンフィグレーションデータ項目	I/O変数
変数が一時的に使用できない	TOがリセット中	TOがリセット中	入力エラー
	-	現在のコンフィグレーションで、データが表示されない	出力エラー
不正な値(書き込み時)	値が制限範囲外	値が制限範囲外	-
	値がグリッドにない	値がグリッドにない	-

原因に応じて、以下の応答が可能です。

表 4-5 エラーの場合の可能な応答

原因	可能な応答
変数の読み取りが一時的にできない	STOPへの移行
	読み取りを行わない
	代替値の読み取り
	最後の妥当な値の読み取り
変数の書き込みが一時的にできない	STOPへの移行
	書き込みを行わない
	代替値を承認し、その値が後で有効になる
	現在の値を承認し、その値が後で有効になる
不正な値	STOPへの移行
	書き込みを行わない
	デフォルト値の書き込み

SIMOTIONでは、以下の応答が実装されています。

表 4-6 エラーの場合のシステム応答

必要な応答		システム変数	コンフィグレーション データ項目	I/O変数
読み取り時に一時的にアクセスできない値:				
STOPへの移行	呼び出し	STOP_DEVICE	STOP_DEVICE	STOP_DEVICE
	結果	→ STOP	→ STOP	→ STOP
	規格値	不変	不変	不変
読み取りを行わない	呼び出し	NO_CHANGE	NO_CHANGE	実装されていない
	結果	NO_CHANGE	DEFAULT_VALUE	
	規格値	不定	不定	
代替値の読み取り	呼び出し	実装されていない	実装されていない	DEFAULT_VALUE
	結果			DEFAULT_VALUE
	規格値			置換値
最後の妥当な値の読み 取り	呼び出し	実装されていない	実装されていない	NO_CHANGE
	結果			NO_CHANGE
	規格値			最後の妥当な値
書き込み時に一時的にアクセスできない値:				
STOPへの移行	呼び出し	STOP_DEVICE	STOP_DEVICE	STOP_DEVICE
	結果	→ STOP	→ STOP	→ STOP
	規格値	不変	不変	現在値
書き込みを行わない	呼び出し	NO_CHANGE	NO_CHANGE	実装されていない
	結果	NO_CHANGE	DEFAULT_VALUE	
	規格値	不変	不変	
代替値を承認し、その 値が後で有効になる	呼び出し	実装されていない	実装されていない	DEFAULT_VALUE
	結果			DEFAULT_VALUE
	規格値			置換値
現在の値を承認し、そ の値が後で有効になる	呼び出し	実装されていない	実装されていない	NO_CHANGE
	結果			NO_CHANGE
	規格値			現在値
不正な値の書き込み時:				
STOPへの移行	呼び出し	STOP_DEVICE	STOP_DEVICE	発生しない
	結果	→ STOP	→ STOP	
	規格値	不変	不変	
書き込みを行わない	呼び出し	NO_CHANGE	NO_CHANGE	
	結果	NO_CHANGE	NO_CHANGE	
	規格値	不変	不変	
デフォルト値の書き込 み	呼び出し	DEFAULT_VALUE	DEFAULT_VALUE	
	結果	DEFAULT_VALUE	INVALID_VALUE	
	規格値	デフォルト値	不変	

実行システム、タスク、およびシステムサイクルクロック

5

このセクションでは、実行レベルを持ち、タスクが割り当てられたSIMOTION実行システムについて説明します。

5.1 実行システム

SIMOTION実行システムは、各種の実行レベルを提供しています。

- 実行レベルは、ユーザープログラムタスクに割り当てられます。

- ユーザータスクは、プログラムに割り当てられます。

すべてのプログラムには、また、その結果、タスクにも、PLCとモーションコントロールタスクを収納することができます。

以下の実行レベルがあります。

- 同期実行レベル: コントロールおよび補間サイクルクロックとの同期
- 時間駆動実行レベル
- イベント駆動実行レベル
- 割り込み制御実行レベル
- シーケンシャル実行レベル
- フリーランニング実行レベル

以下のタスクでは、異なる実行プロパティを持つ各種のタスクを使用できます。

- システムタスクは、システムによって定期的に実行されます。

システムサイクルクロックを指定することができます。

システムタスクによって、以下のタスクが実行されます。

- 通信

アイソクロナスPROFIBUS、IRTによるPROFINET IO、またはシステムサイクルクロックへの接続とI/O処理用のシステムタスク

- モーションコントロール

複数のアイソクロナス実行レベルがあり、効率的なモーションコントロール(IPO/IPO_2)、位置制御(サーボ)、およびドライブ通信(DP)で使用できます。

デジタルドライブの場合、これらの実行レベルは、アイソクロナスPROFIBUSまたはIRTによるPROFINET IOと同期化されます。

アイソクロナスPROFIBUSを使用しないアナログドライブの場合、実行レベルは、内部タイマーの設定可能なシステムサイクルクロックによって提供されます。

5.1 実行システム

– 温度制御

TControlテクノロジーパッケージに関して、温度制御:

現在値測定、出力信号の制御およびパルス幅変調で、実行レベルが使用できます。

- ユーザープログラミング(ユーザープログラムタスク):
モーションコントロール、ロジック、およびテクノロジーファンクションでは、タスク関連プログラミング用の実行レベルが使用できます。

下記も参照

コンフィグレーション時の設定 (ページ 229)

5.1.1 実行レベル / タスク

実行レベルは、実行システムのプログラムの時間的順序を定義します。

各実行レベルには、1つまたは複数のタスクが含まれます。

タスクは、プログラムの実行フレームワークを提供します。

各タスクは、特定の条件が満たされたとき実行されます。

各タスクには、1つまたは複数のユーザープログラムを割り当て、タスク内でのそれらの順序を指定することができます。

ユーザープログラム以外に、内容と実行順序が固定の複数のシステムタスクがあります。

実行レベル

以下の図は、タスクを含む実行レベルを示します。

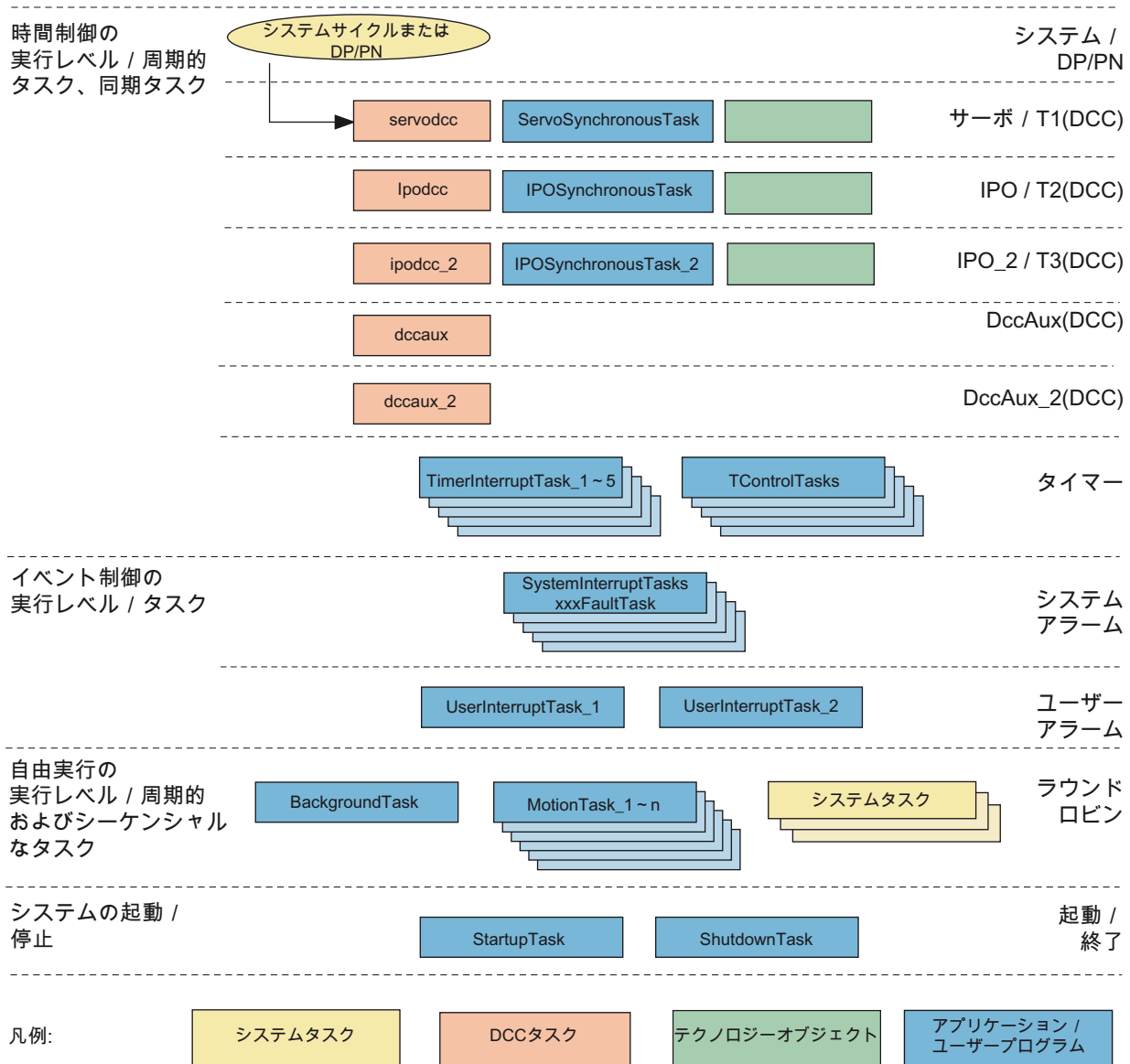


図 5-1 SIMOTIONランタイムシステムの実行レベルとタスク

テクノロジーパッケージを使用するときは、システム提供の実行レベルが自動的に割り当てられます。

ユーザープログラムは、これらの実行レベルのテクノロジーファンクションの処理に影響を与えることはできません。

システムタスクは、たとえば、非周期の通信(Ethernet、PROFIBUS DP、PROFINET IO)、デバッグサービス、またはトレース事前処理です。

DCCレベルとタスクは、SCOUTユーザーインターフェースの実行システムでは表示されません。

DCCプランは、DCCエディタの実行グループに配置されます。「DCCエディタの概要」を参照してください。

タスク

ユーザープログラミング用の各実行レベルでは、1つまたは複数のタスクが使用できます。
タスクの主な特性には、以下のものがあります。

- 起動属性: タスクが起動される時点と条件。(表を参照)
- 優先度: どのタスクがどの他のタスクによって割り込まれるか。
(次のセクションのタスク優先度の表を参照)

次の表は、ユーザープログラムで使用できるタスクを示します。

表 5-1 SIMOTIONランタイムシステムのタスク

タスク		説明
StartupTask		StartupTask(スタートアップタスク)は、STOPまたはSTOPUモードからRUNモードに移行したとき、一度だけ実行されます。 テクノロジーオブジェクトの初期設定およびリセット用です。
フリーランニングタスク:		ラウンドロビン実行レベルでは、タイムスライスプロシージャの背景で、システムによって、MotionTasksとBackgroundTaskが実行されます。
	MotionTasks	MotionTasksは、シーケンスのプログラミングのためのタスクで、プログラムされたモーションコントロールまたは他のシーケンシャル実行で使用されます。 MotionTasksはユーザープログラムによって起動され、一度だけ実行されます。
	BackgroundTask	BackgroundTaskは、固定的な時間フレームなしの周期的シーケンスのプログラミングのために提供されます。 BackgroundTaskは、システムスタートアップ後に起動された後、周期的にフリーランニングで実行されます。
時間駆動タスクと同期タスク:		周期的タスク。 それらは一定の時間フレームで周期的に呼び出され、割り当てられたプログラムの実行後に自動的に再起動されます。
	TimerInterruptTasks	TimerInterruptTasksは、プログラムの周期的起動のためのタスクです。
	SynchronousTasks	SynchronousTasksは、指定されたシステムサイクルクロックと同期して、周期的に起動されます。
イベント駆動タスク:		シーケンシャルタスク。 このタスクは、イベントが発生したときに起動され、一度だけ実行された後、終了します。
	SystemInterruptTasks	SystemInterruptTasksは、システムイベントが発生したとき起動され、一度だけ実行されます。
	UserInterruptTasks	UserInterruptTasksは、ユーザーが定義したイベントが発生したとき起動され、一度だけ実行されます。
ShutdownTask		The ShutdownTask(停止タスク)は、RUNモードからSTOPまたはSTOPUモードに移行したとき、一度だけ実行されます。

注記

ユーザープログラムタスク以外にも、ユーザーが変更できない各種のシステム内部タスクがあります。

たとえば、ControlPanelTaskもこのようなシステム内部タスクの1つです。
それは、デバイス診断のタスクランタイム時には表示されますが、実行システムでは表示されません。

下記も参照

StartupTask (ページ 138)
MotionTasks (ページ 140)
BackgroundTask (ページ 144)
TimerInterruptTasks (ページ 147)
SynchronousTasks (ページ 151)
SystemInterruptTasks (ページ 157)
UserInterruptTasks (ページ 162)
ShutdownTask (ページ 167)

5.1.2 SIMOTION SCOUTの実行システム

SIMOTION

SCOUTでは、実行システムで使用されるすべてのタスクを表示することができます。

- プロジェクトナビゲータで、SIMOTIONデバイスを選択し、メニューの[Target system|Configure execution system]を選択するか、[EXECUTION SYSTEM]をダブルクリックします。

5.1 実行システム

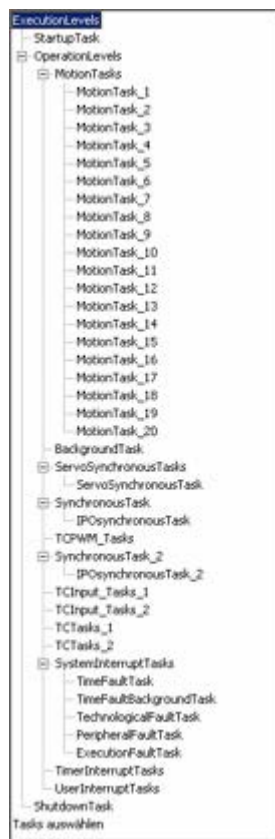


図 5-2 SCOUTのタスクの一覧

実行システム

ここには、実行レベルを持ち、タスクが割り当てられた実行システムが表示されます。

実行レベルツリー

実行レベルツリーには、固定エントリとして使用可能な実行レベル / タスクが表示されます。

[OperationLevel]フォルダには、RUNモードで使用できるタスクが収納されています。

- [OperationLevel]フォルダを開くには、その前の+記号をクリックします。

それぞれの実行レベルまたはタスク名の下には、コンフィグレーションされたタスクと、タスクに割り当てられたプログラムが表示されます。

タスクにプログラムを割り当てると、それらのプログラムは実行レベルツリーに表示されません。

実行システムでのタスクの使用

実行システムで使用する必要があるタスクをリストから選択します。

- 各タスクのチェックボックスを有効にします。

選択したタスクだけが、実行レベルツリーに表示されます。

実行システム - コンテキストメニュー

次のファンクションを選択できます。

ファンクション	意味/注
[Open]	このファンクションを使用して、実行レベルコンフィグレーションを開きます。
エキスパート	
システムサイクルクロックの設定	これを使用して、PROFIBUSまたはPROFINETインターフェースでのパラメータ設定されたアイソクロナス操作に応じて、システムサイクルクロック間(たとえば、補間サイクルクロックとサーボサイクルクロック間)の比率を設定します。
印刷	これを使用して、実行システムの内容を印刷します。 関連コンフィグレーションを持つすべてのタスクが印刷されます。
印刷プレビュー	これを使用して、実行システムの印刷プレビューを開きます。

5.1.3 タスク優先度

タスクとタスクに割り当てられたプログラムは、特定の時刻に起動され、実行されます。複数のタスクが同時に起動され、実行される場合、タスク優先度が最初に実行されるタスクを決定します。

表 5-2 タスク優先度の一覧

優先度	実行レベル	タスク	周期的 / シーケンシャル	重要度
高速	サーボ、DP通信	-	z	
	サーボ / T1(DCC)	Servodcc ServoSynchronousTask ServoTask	z	DCCランタイムグループのタスク(T1) サーボサイクルクロックでのユーザープログラムタスク システムタスク
	IPO / T2(DCC)	Ipodcc	z	DCCランタイムグループのタスク(T2)
		IPOSynchronousTask • UserInterruptのチェック • ユーザープログラム	z	補間サイクルクロックでのユーザープログラムタスク
		IPOTask • WAITFORCONDITIONの条件のチェック(IPO、システム)	z	IPOサイクルクロックでのシステムタスク
	IPO_2/T3(DCC)	Ipodcc_2	z	DCCランタイムグループのタスク(T3)
		IPOSynchronousTask_2	z	補間サイクルクロック2でのユーザープログラムタスク

5.1 実行システム

優先度	実行レベル	タスク	周期的 / シーケンシャル	重要度
↓		IPOTask2	z	IPOサイクルクロックでのシステムタスク
	TControlTasks	PWMSynchronousTask	z	温度制御タスク
		InputSynchronousTask_1	z	温度制御タスク
		InputSynchronousTask_2	z	温度制御タスク
		PostControlTask_1	z	温度制御タスク
		PostControlTask_2	z	温度制御タスク
		+ 関連システムタスク	z	
	DccAux(DCC)	Dccaux	z	DCCランタイムグループのタスク(T4)
	DccAux_2(DCC)	Dccaux_2	z	DCCランタイムグループのタスク(T5)
	SystemInterruptTasks	TimeFaultTask	s	システムアラーム (イベントの順序で)
		TimeFaultBackgroundTask	s	システムアラーム (イベントの順序で)
		TechnologicalFaultTask	s	システムアラーム (イベントの順序で)
		PeripheralFaultTask	s	システムアラーム (イベントの順序で)
		ExecutionFaultTask	s	システムアラーム (イベントの順序で)
	TimerInterruptTasks	TimerInterruptTask1 ... TimerInterruptTask5	z ... z	タイマーアラーム
	UserInterruptTasks	UserInterruptTask_1	s	ユーザーアラーム(イベントの順序で)
		UserInterruptTask_2	s	ユーザーアラーム(イベントの順序で)
	ラウンドロビン	BackgroundTask	z	ユーザープログラムタスク(シーケンスは指定できません)
		MotionTask_1 ... MotionTask_32	s ... s	ユーザープログラムタスク(シーケンスは指定できません)
低速				

優先度

タスクの優先度は、ユーザーが変更することはできません。

注記

タスクは、その優先度に従って実行されます。
その結果、高優先度のタスクは、低優先度のタスクより前に実行されます。
このため、低優先度のタスクの場合、サイクルタイムが変動する場合があります。

- **TimerInterruptTasks:**
タイムスライスが短ければ短いほど、優先度は高くなります。
- **UserInterruptTasks:**
すべてのタスクは同じ優先度を持ち、起動された順序で実行されます。
- **条件の待機 / WAITFORCONDITION**は、MotionTask(モーションタスク)の優先度を一時的に高めます。
 - **UserInterruptTasks**の場合と同様に、同一の優先度を持つ場合に、条件がチェックされます。
 - 条件を満たすと、MotionTask(以前に保留されていた)が再起動されます。
 - **WAITFORCONDITION**と**ENDWAITFORCONDITION**で囲まれたコマンドは、高くされた優先度で実行されます(**SystemInterruptTasks**と**TimerInterruptTasks**の間)。

条件の待機 / WAITFORCONDITIONの詳細は、**SIMOTION MCC**または**SIMOTION ST**のプログラミングマニュアルを参照してください。

UserInterruptTaskの条件のチェック

UserInterruptTasksの条件は、IPOSynchronousTaskのユーザープログラムの実行の前に、IPOSynchronousTaskでチェックされます。

UserInterruptTask条件で発生したエラーは、IPOSynchronousTaskのエラーとして処理されます。

UserInterruptTaskの条件のチェックに必要な時間は、IPOSynchronousTaskのランタイムの一部とされます。

待機条件のチェック

条件の待機 / WAITFORCONDITIONなどの待機条件を使用できる場合、システムがチェックを実行し、チェックのためにタスクは起動されません。

コマンドに関する同期設定(たとえば、モーションエンドまでの待機)は、特に、シーケンシャルタスクの場合に必要です。

WAITFORCONDITION条件は、IPOSynchronousTaskのユーザープログラムの終了後、IPO Taskが起動されたときにチェックされます。

WAITFORCONDITIONで発生しエラーは、関連MotionTaskのエラーとして処理されます。

WAITFORCONDITION条件(複数の場合あり)は、IPO Taskのランタイムの一部とされます。

タスク起動シーケンス

StartupTask(スタートアップタスク)が完了したとき、**RUN**モードになります。

この後、以下のタスクが起動されます。

- SynchronousTasks
- TimerInterruptTasks
- BackgroundTask
- 自動起動の属性が設定されているMotionTasks

注記

実行レベルまたはそのタスクの優先度は、**RUN**モードになった後に、MotionTasks、BackgroundTask、および時間トリガタスクが起動される順序を示すわけではありません。

5.1.4 SIMOTIONのランタイムモデル

次の図は、SIMOTIONにおけるタスクの一般的シーケンスと優先度(高 -> 低)を示します。

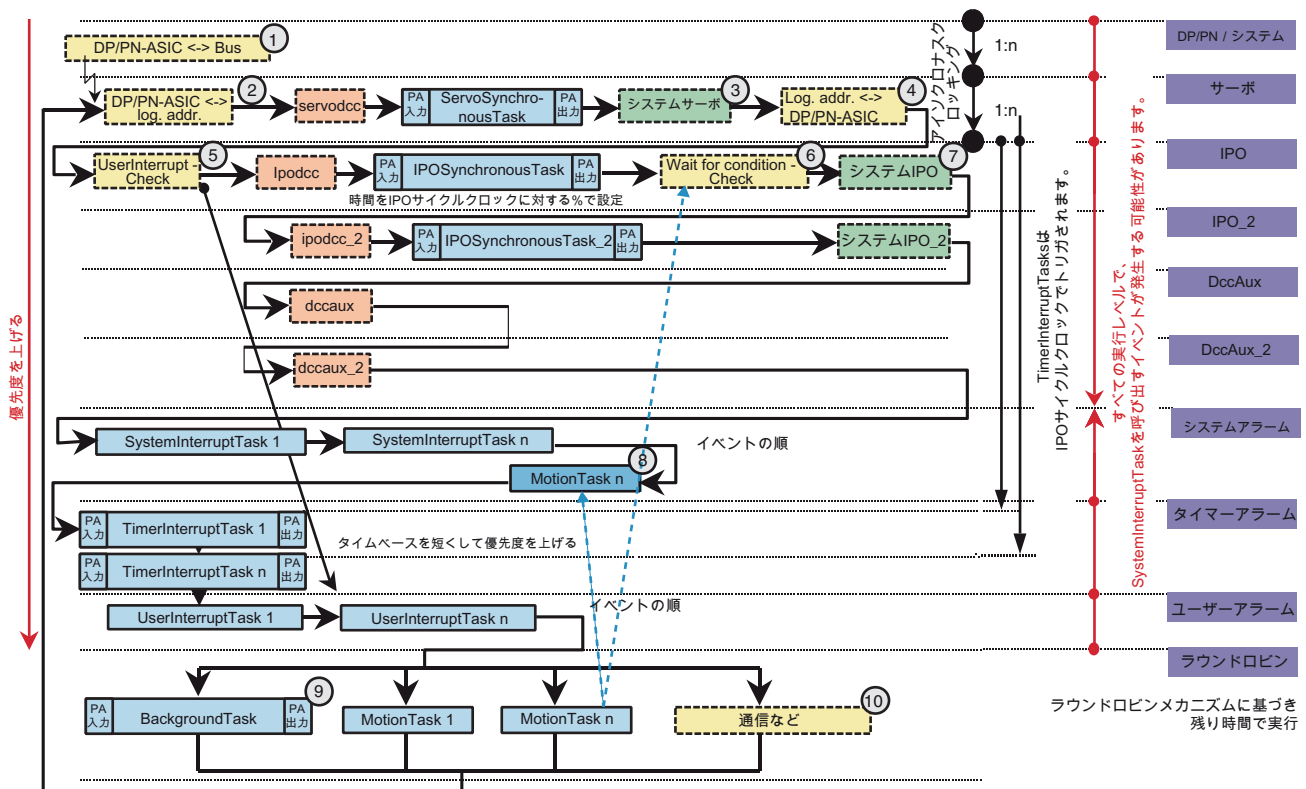


図 5-3 SIMOTIONのランタイムモデル - 一般的シーケンス、優先度

図の説明:

この表示は、DP、サーボ、およびIPOの比率が1:1:1の場合の表示です。

- 色が青 / 緑: ユーザーが使用可能(アプリケーション / テクノロジーオブジェクト)
- 色が黄色(点線): ユーザーは使用不可(システムタスク)

システムタスクの説明

1. DP/PN-ASIC <-> バス:

データが、通信チップから、PROFIBUSまたはIRTによるPROFINET IOへコピーされるか、そこからフェッチされます。
論理アドレス(2)へのI/O入力の転送は、コピー動作が終了したときに起動されます。

コピー動作は、それ自身のプロセッサ上で行われるため、残りの実行システムとは非同期に実行されます。同期ポイントは、バスとASIC間のデータ交換の終了時です。

2. DP/PN-ASIC -> 論理アドレス:

I/O入力が、通信チップからロードされます。

3. システムサーボ:

サーボサイクルクロックでのシステム計算(位置コントローラなど)。

サーボ実行レベルのタスクは、サーボサイクルクロック内で完了する必要があります。
サーボ実行レベルのオーバーフローの場合、CPUはSTOPに移行します。

4. 論理アドレス -> DP/PN-ASIC:

I/O出力が、通信チップに書き込まれます。

5. UserInterrupt - チェック:

2つのユーザー割り込みの条件がチェックされます。

6. 条件の待機 - チェック:

WAITFORCONDITION(軸の待機、信号の待機など)の条件がチェックされます。

7. システムIPO/IPO_2:

IPOサイクルクロックのシステム側の要素が計算されます(モーションコントロール: プロファイルの位置決め、同期操作など)。エラー!ブックマークが定義されていません。

8. MotionTask n:

WAITFORCONDITIONで待機しているMotionTaskは条件が発生したとき優先して起動されます(高優先度の使用)。

9. BackgroundTask:

BackgroundTaskは、ここで実行されます。

バックグラウンドプロセスイメージ(PI)の更新は、BackgroundTask全体の起動時と完了後に行われます。

ランタイムモデルは、通常、入カイメージの読み取りと出カイメージの書き込みの間に複数回実行されます。すなわち、ユーザープログラムのサイズに応じて、BackgroundTaskは高優先度のタスク(ServoTaskで起動)によって複数回割り込まれます。

10. 通信:

通信ファンクション(HMI、PG/PCなど)。

下記も参照

SynchronousTasks (ページ 151)

BackgroundTask (ページ 144)

5.2 ユーザープログラムタスクの概要

5.2.1 StartupTask

StartupTaskは、テクノロジーオブジェクトの一度の初期設定とリセットのために提供されます。

それは、操作モードが**STOP**または**STOPU**から**RUN**に切り替わったときに起動されます。

このタスクは、*nof*プロセススタートアップ、原点復帰、軸の設定のために使用してはいけません(モーションコマンドを使用してはいけません)。

StartupTaskが実行されている間、**SystemInterruptTask**と**UserInterruptTask**を除いて、アクティブである他のユーザープログラムは存在しません。

プロセスイメージとシンボルI/O変数へのアクセスは、禁止されています。

入力のプロセスイメージは、スタートアップタスクの前に更新され、スタートアップタスクの実行中は不変です。

出力のプロセスイメージは、スタートアップタスクの前にゼロに設定され、スタートアップタスクの後に出力されます。入力への直接アクセスでは、現在値が提供差荒れます。

書き込みI/O変数は、初期値に設定することができます。

ただし、これらの値は、端子のすべての出力の有効化機能を持つスタートアップタスクの後にのみ有効になります。

StartupTask(スタートアップタスク)が完了したとき、**RUN**モードになります。

この後、以下のタスクが起動されます。

- **SynchronousTasks**
- **TimerInterruptTasks**
- 自動起動属性が設定されている**MotionTasks**
- **BackgroundTask**

[**Program**

assignment]タブを選択し、作成され、コンパイルされたプログラムをStartupTaskに割り当て、それらのプログラムの実行シーケンスを定義します。

StartupTaskのコンフィグレーション

1. 実行レベルツリーの[StartupTask]をクリックします。

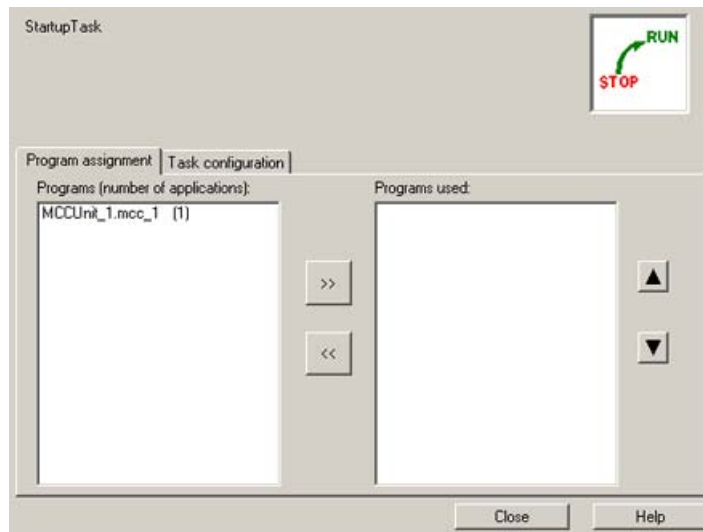


図 5-4 StartupTaskのコンフィグレーション(プログラム割り当て)

2. **[Program assignment]**タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

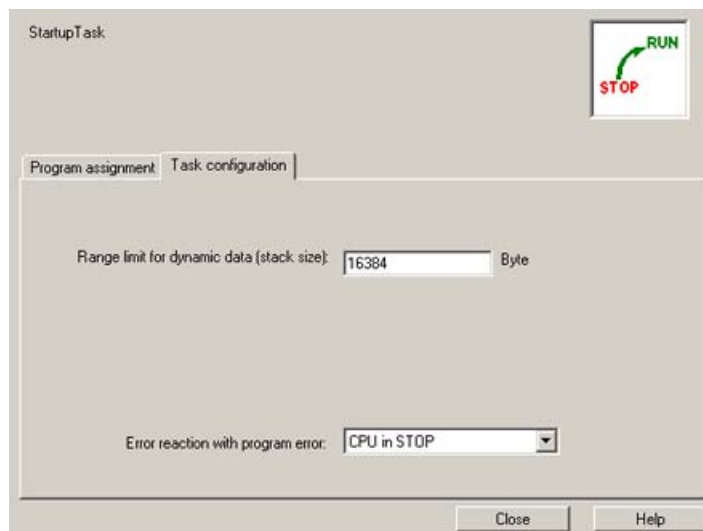


図 5-5 StartupTaskのコンフィグレーション(タスクコンフィグレーション)

3. **[Task Configuration]**タブに切り替えます。
4. 必要な場合、**[Range limit for dynamic data]**(スタックサイズ)を入力します。
5. **[Error reaction with program error]**(たとえば、**ExecutionFaultTask**)を指定します。

タスクコンフィグレーション - StartupTask

[Task configuration]タブで、プログラムエラー時のエラー応答に関するパラメータを設定します。次のパラメータを設定することができます。

フィールド/ボタン	説明/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUがSTOPモードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合は、CPUはSTOPモードに切り替わります。エラーが発生したタスクが終了します。

下記も参照

実行レベル / タスクへのプログラムの割り当て (ページ 170)

SystemInterruptTasks (ページ 157)

5.2.2 MotionTasks

MotionTasksは、シーケンスのプログラミングのためのタスクで、プログラムされたモーションコントロールまたは他のシーケンシャル実行で使用されます。

シーケンシャル実行の例:

軸が目標位置に移動し、使用可能信号を待ってから、次の目標位置に移動します。

複数のMotionTasksが使用できます。

- V3.2まで: 20のMotionTasks(**MotionTask_1** ~ **MotionTask_20**)
- V4.0では、D4xxおよびP350の場合のみ:
32のMotionTasks(**MotionTask_1** ~ **MotionTask_32**)

MotionTasksの名前は変更することができます。タスクへのプログラムの割り当て (ページ 230)を参照してください。

MotionTasksは、ラウンドロビン実行レベルで実行されます。

MotionTasksと**BackgroundTask**は、高優先度のシステムおよびユーザープログラムタスクとは別に、フリータイムを共有します。

両方のレベル間のタイムスライスの関係は、パラメータとして指定することができます。「時間割り当ての設定 (ページ 189)」を参照してください。

MotionTasksとBackgroundTaskには、決まった実行シーケンスは存在しません。

タスク実行に効果を持つ手順については、タスク制御コマンドの一覧 (ページ 252)に提供されています。

MotionTask(モーションタスク)の起動

MotionTasksは、通常、ユーザープログラムで、`_startTaskID`、`_stopTaskID`、...などのタスク制御コマンドを使用して制御します。

該当のコンフィグレーション(属性の設定)が行われたMotionTaskは、RUNモードになったとき自動的に起動されます。

`_getStateOfTaskID`システムコマンドを使用して、現在のタスクステータスをスキャンすることができます。

MotionTaskは時間監視機能を全く持たず、一度起動されると、無期限にアクティブであり続ける場合があります。

同期コマンドを待機するMotionTaskは、そのステータスはアクティブのままです。

MotionTask(モーションタスク)の完了

MotionTaskは、タスクが完了するか、STOPまたはSTOPUモードへの移行時(ShutdownTaskの起動時)に完了します。

タスクを自動的に一時停止する1つの方法は、待機コマンド(条件の待機 / WAITFORCONDITION)を使用することです。

待機コマンドを発行すると、タスクは一時停止されます。

コマンドで指定された条件は、IPOサイクルクロックでチェックされます。

条件が満たされると、MotionTaskが自動的に再開されます。

待機コマンドの配置に応じて、実行が続くときのタスク優先度を設定することができます。

WAITFORCONDITIONとENDWAITFORCONDITIONで囲まれたコマンドは、高くされた優先度で実行されます(SystemInterruptTasksとTimerInterruptTasksの間)。

[Program

assignment]タブを選択し、作成され、コンパイルされたプログラムをMotionTasksに割り当て、実行シーケンスを定義します。

次のパラメータを設定することができます。

フィールド/ボタン	説明/注
MotionTask(モーションタスク)	[MotionTask]のリストされるモーションタスクから、プログラムを割り当てるモーションタスクを選択します。 1つのMotionTaskに複数のプログラムを割り当てることができます。
MotionTask_1 to MotionTask_n	可能なMotionTasksの事前定義された名前。
実行システムでのタスクの使用	実行システムでこのタスクを表示 / 使用する場合は、このチェックボックスを有効にします。 このチェックボックスが無効な場合は、このタスクにプログラムを割り当てることはできません。

MotionTasksのコンフィグレーション

RUNモードになったときに自動的に起動するタスクを定義することができます。定義していない場合は、プログラムされたタスク制御コマンドでMotionTasksを明示的に起動する必要があります。

1. 実行レベルツリーの[MotionTasks]をクリックします。
2. [MotionTask]リストで、必要なタスクを選択します。
タスク名は変更することができます。「実行レベルへのプログラムの割り当て (ページ 170)」を参照してください。

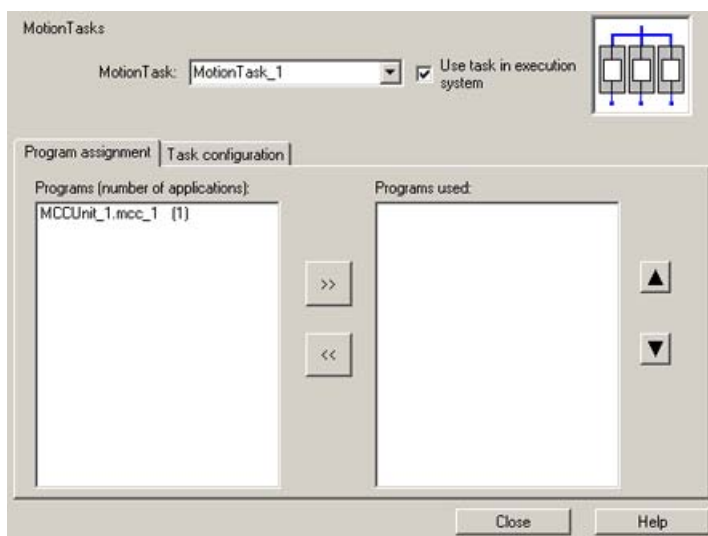


図 5-6 MotionTasksのコンフィグレーション(プログラム割り当て)

3. [Program assignment]タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

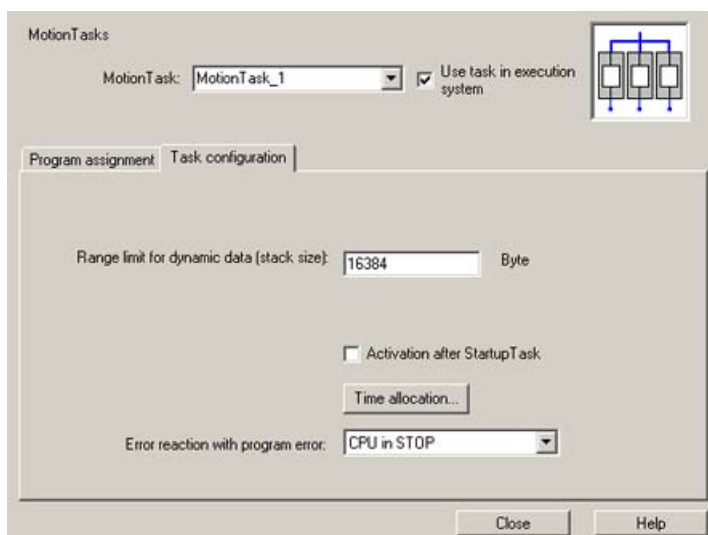


図 5-7 MotionTasksのコンフィグレーション(タスク割り当て)

4. [Task Configuration]タブを選択します。

5. [Activation after StartupTask]オプションを有効にして、StartupTaskの後に一度だけMotionTaskを起動します。
6. 必要な場合、[Range limit for dynamic data](スタックサイズ)を入力します。
7. [Error reaction with program error](たとえば、ExecutionFaultTask)を指定します。
8. コンフィグレーションするすべてのMotionTasksについて、ステップ2~7を繰り返します。
9. MotionTasksとBackgroundTasksのラウンドロビン実行レベルでの時間割り当てを指定します。「時間割り当ての設定 (ページ 189)」を参照してください。

タスクコンフィグレーション - MotionTasks

[Task configuration]タブで、ラウンドロビン実行レベルでのMotionTasksとBackgroundTasksの時間割り当てをパラメータとして指定し、MotionTasksを起動する時点を定義します。

次のパラメータを設定することができます。

フィールド/ボタン	説明/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行される時、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Activation after StartupTask	RUN モードになったとき(StartupTaskが完了したとき)に一度だけMotionTaskを起動する場合は、これを選択します。
Time allocation	このボタンをクリックして、 ラウンドロビン実行レベルでの時間割り当て 用の画面フォームを開きます。 この画面フォームでは、ラウンドロビン実行レベルでのMotionTasksとBackgroundTasksの時間割り当てをパラメータとして指定することができます。
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUが STOP モードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合は、CPUは STOP モードに切り替わります。エラーが発生したタスクが終了します。

下記も参照

BackgroundTask (ページ 144)

SystemInterruptTasks (ページ 157)

ラウンドロビン実行レベルでの時間割り当て (ページ 187)

5.2.3 BackgroundTask

BackgroundTaskは、固定的な時間フレームなしの周期的シーケンスのプログラミングのために提供されます。

それは、ラウンドロビン実行レベルで周期的に実行されます(すなわち、完了時に自動的に再起動されます)。

BackgroundTaskは、周期的に実行する必要があるプログラム(たとえば、インターロックタスク、PLCタスクなど)で使用されます。

BackgroundTaskのサイクルタイムは、監視されます。

サイクルタイム監視が応答すると、**TimeFaultBackgroundTask**(タイムエラーバックグラウンドタスク)が起動されます。

そのタスクがコンフィグレーションされていないか、タスクにプログラムが割り当てられていない場合、CPUは**STOP**モードに切り替わります。

アドレススペース0.0 ~ 63.7の**BackgroundTask**の場合、入力および出力のプロセスイメージが生成されます。**BackgroundTask**が処理されている間、プロセスイメージは不変です。

BackgroundTaskを使用して、低優先度の周期的ロジックファンクション、インターロック、計算、監視機能を実現することができます。

BackgroundTaskは、使用可能なCPU時間を**MotionTasks**と分け合います。

注記

BackgroundTaskは、計算時間を**MotionTasks**および**SystemTasks**(たとえば、通信タスク)と分け合います。

時間割り当てに関しては、ランタイム、すなわち、性能がその設定(ラウンドロビン実行レベルの時間割り当て)によって影響を受けることを考慮する必要があります。

BackgroundTaskの起動

BackgroundTaskは、**RUN**モードになるか、タスクが完了したとき(次のサーボサイクルクロックの使用)、自動的に起動されます。

BackgroundTaskの完了

MotionTaskは、**STOP**または**STOPU**モードへの移行時(**ShutdownTask**の起動時)に完了します。

[Program

assignment]タブを選択し、作成され、コンパイルされたプログラムを**BackgroundTasks**に割り当て、実行シーケンスを定義します。

BackgroundTaskのコンフィグレーション

1. 実行レベルツリーの[BackgroundTask]をクリックします。

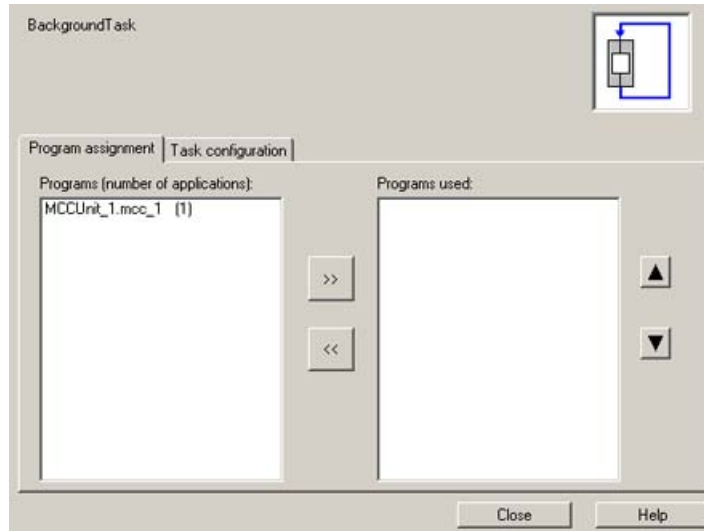


図 5-8 BackgroundTaskのコンフィグレーション(プログラム割り当て)

2. [Program assignment]タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

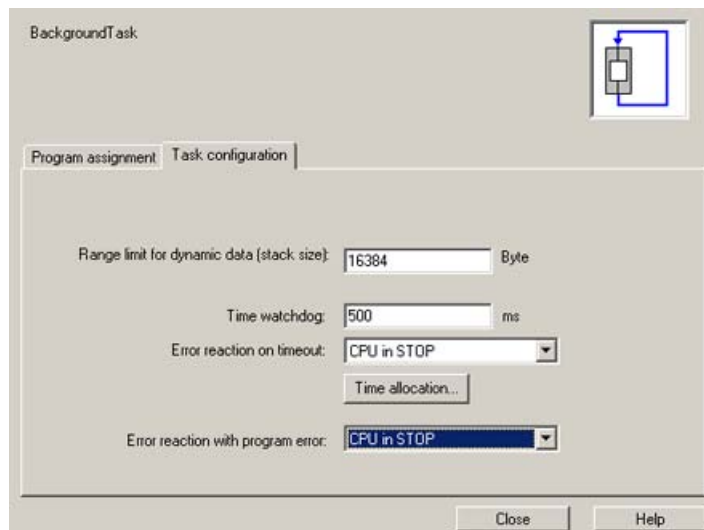


図 5-9 BackgroundTaskのコンフィグレーション(タスク割り当て)

3. [Task configuration]タブに切り替えます。
4. 必要な場合、[Range limit for dynamic data](スタックサイズ)を入力します。
5. 時間監視用の値を入力します。

この時間を超えた場合は、関連SystemInterruptTask(TimeFaultBackgroundTask)を呼び出すか、CPUをSTOPモードにすることができます(0 ms = 監視なし)。

6. [Error reaction on timeout]を指定します。
CPUをSTOPモードにするか、TimeFaultBackgroundTaskを呼び出します。
7. MotionTasksとBackgroundTasksのラウンドロビン実行レベルでの時間割り当てを指定します。「時間割り当ての設定」を参照してください。
8. [Error reaction with program error](たとえば、ExecutionFaultTask)を指定します。

タスクコンフィグレーション - BackgroundTask

[Task configuration]タブでは、時間監視とエラー応答のパラメータを設定することができます。
次のパラメータを設定することができます。

フィールド/ボタン	説明/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
ウォッチドッグ	BackgroundTaskを実行するサイクルタイムをms単位で入力します。 時間監視用の値を入力します。 0を入力するか、値を入力しないと、時間監視は行われません。
Error reaction with program error	BackgroundTaskのサイクルが時間監視で指定された時間フレーム内に終了しない場合のエラー応答を選択することができます。 エラー応答としては、SystemInterruptを選択し、コンフィグレーションすることができます。
TimeFaultBackgroundTask	BackgroundTaskでのタイムアウトシステム割り込み
CPU to Stop	CPUは、STOPモードに切り替わります。
Time allocation	このボタンをクリックして、ラウンドロビン実行レベルでの時間割り当て用の画面フォームを開きます。 この画面フォームでは、ラウンドロビン実行レベルでのMotionTasksとBackgroundTasksの時間割り当てをパラメータとして指定することができます。
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to Stop	CPUがSTOPモードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 この後、CPUはSTOPモードに移行します。

下記も参照

MotionTasks (ページ 140)

実行レベル / タスクへのプログラムの割り当て (ページ 170)

ウォッチドッグ (ページ 185)

時間割り当ての設定 (ページ 189)

SystemInterruptTasks (ページ 157)

ラウンドロビン実行レベルでの時間割り当て (ページ 187)

5.2.4 TimerInterruptTasks

TimerInterruptTasksは、プログラムの周期的起動のためのタスクです。

異なる時間レベルに対して、5つのTimerInterruptTasks(TimerInterruptTask_1 ~ TimerInterruptTask_5)を使用することができます。

TimerInterruptTasksは周期的に起動され、コンフィグレーションされた固定時間フレーム(たとえば、100 ms)で実行されます。

この時間フレームは、補間サイクルクロックの倍数でなければなりません。

このタスクには、I/Oへの直接リンクや軸のモーションコントロールなしの再生可能な時間参照を必要とする閉ループ制御または監視機能を実装することができます。

TControlテクノロジーパッケージ用に、その他のシステムおよびユーザータスクが提供されています。

TControlテクノロジーパッケージはシステムタスクで処理されるのに対し、アプリケーション固有の機能はユーザープログラムタスクで処理されます。

温度コントローラの機能的説明は、『モーションコントロール、追加のテクノロジーオブジェクト』機能マニュアルを参照してください。

TimerInterruptTask(タイマー割り込みタスク)の起動

TimerInterruptTasksは、周期的に起動されます。起動遅延を設定することができます。

TimerInterruptTask(タイマー割り込みタスク)の完了

TimerInterruptTasksは、TimerInterruptTaskに割り当てられたプログラムが完了した後、自動的に完了します。

[Program

assignment]タブでは、作成され、コンパイルされたプログラムを選択したTimerInterruptTaskに割り当て、それらのプログラムの実行シーケンスを定義することができます。

次のパラメータを設定することができます。

フィールド/ボタン	説明/注
For task	これを使用して、5つのTimerInterruptTasksの中で、プログラムを割り当てる1つを選択します。 1つのTimerInterruptTaskに複数のプログラムを割り当てることができます。
TimerInterruptTask_1 to TimerInterruptTask_5	5つのTimerInterruptTasksの事前定義された名前。
Defined time level	これを使用して、TimerInterruptTaskを時間フレームを選択します。 使用可能なリストから時間レベルを選択するか、必要な場合、他の時間レベルを入力することができます。 この値は、IPOサイクルクロックの倍数でなければなりません。

フィールド/ボタン	説明/注
実行システムでのタスクの使用	実行システムでこのタスクを表示 / 使用する場合は、このチェックボックスを有効にします。 このチェックボックスが無効な場合は、このタスクにプログラムを割り当てることはできません。

TimerInterruptTasksのコンフィグレーション

1. 実行レベルツリーの[TimerInterruptTasks]をクリックします。
2. [For task]選択ボックスで、必要なタスクを選択します。
タスク名(TimerInterruptTask_1 ~ TimerInterruptTask_5)は事前設定されています。

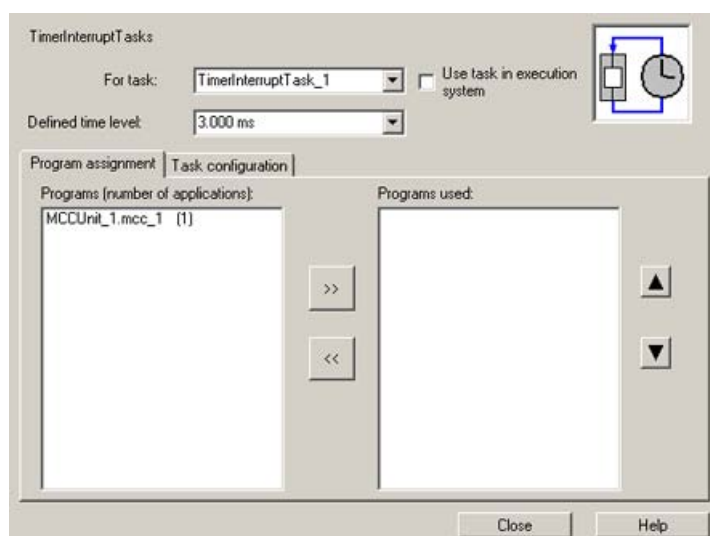


図 5-10 TimerInterruptTasks(プログラム割り当て)

3. [Defined time level]を選択するか、任意の整数値を入力します。
この値は、補間サイクルクロック(IPOサイクルクロック)の倍数でなければなりません。
倍数でない設定は、IPOサイクルクロックの次の整数倍に切り上げられます。
4. [Program assignment]タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

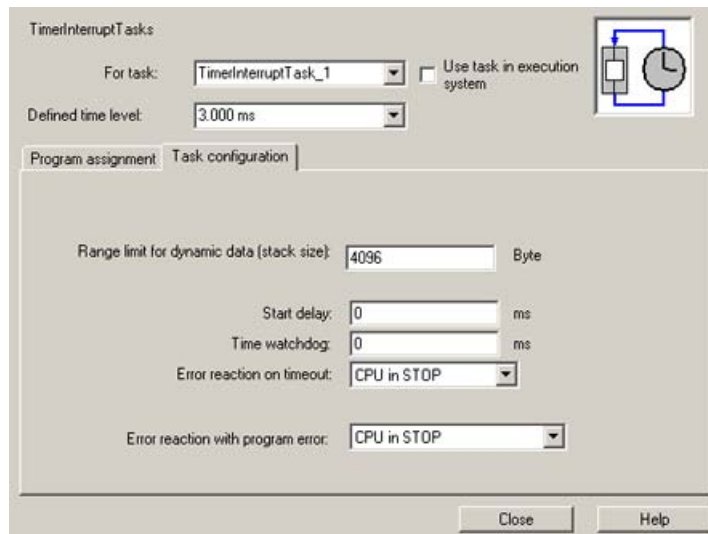


図 5-11 TimerInterruptTasksのコンフィグレーション(タスクコンフィグレーション)

5. [Task Configuration]タブに切り替えます。

6. 必要な場合、[Range limit for dynamic data](スタックサイズ)を入力します。

7. 必要な場合、このタスクの[Start delay]を入力します。

異なる時間トリガタスクが同時に起動されることを防ぐために、タスクごとに、起動遅延を定義することができます。

これによって、異なるTimerInterruptTasksが同時に起動されることがなくなります。

例:

IPOサイクルクロック: 4 ms

TimerInterruptTask_1: 8 ms

TimerInterruptTask_2: 16 ms

この場合、4番目のIPOサイクルクロックごとに、両方のTimerInterruptTasksが同時に起動されます。2つのタスクの中の1つの起動を4

msだけ遅延することによって、それらが同時に実行されることを防ぐことができます。この結果、システムの負荷がより均一に配分され、低優先度のタスクの動作を再現することができます。

8. 時間監視用の値を入力します。

この時間を超えた場合は、関連SystemInterruptTask(TimeFaultTask)を呼び出すか、CPUをSTOPモードにすることができます(0 ms = 監視なし)。

9. [Error reaction on timeout]を指定します。

CPUをSTOPモードにするか、TimeFaultTaskを呼び出します。

10. [Error reaction with program error](たとえば、ExecutionFaultTask)を指定します。

11. コンフィグレーションするすべてのTimerInterruptTasksについて、ステップ3～10を繰り返します。

タスクコンフィグレーション - TimerInterruptTasks

[Task configuration]タブでは、TimerInterruptTasksの起動遅延、時間監視、およびエラー応答を定義することができます。

次のパラメータを設定することができます。

フィールド/ボタン	説明/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Start delay	タスク起動遅延の値をms単位で入力します。 TimerInterruptTaskの起動は、この時間の値だけ遅延されます。 この遅延により、異なるTimerInterruptTasksが同時に起動されることによるタイムアウトが発生しなくなります。
ウォッチドッグ	TimerInterruptTaskを実行するサイクルタイムをms単位で入力します。 時間監視用の値を入力します。 0を入力するか、値を入力しないと、時間監視は行われません。
Error reaction with timeout	TimerInterruptTaskが時間監視で指定された時間フレーム内に終了しない場合のエラー応答を選択することができます。 エラー応答としては、以下を設定することができます。
TimeFaultTask	TimerInterruptTaskでのタイムアウトSystemInterrupt。
CPU to Stop	CPUは、 STOP モードに切り替わります。
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUが STOP モードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合は、CPUは STOP モードに切り替わります。エラーが発生したタスクが終了します。

下記も参照

実行レベル / タスクへのプログラムの割り当て (ページ 170)

ウォッチドッグ (ページ 185)

SystemInterruptTasks (ページ 157)

5.2.5 SynchronousTasks

SynchronousTasksは、指定されたシステムサイクルクロックと同期して起動されます。このタスクは、時間トリガ実行レベルで、高い優先度で実行されます。

以下のSynchronousTasksを使用できます。

- **ServoSynchronousTask**(V4.0以降): サーボサイクルクロックとの同期

サーボ同期タスクでは、I/Oに対するタイムクリティカルな端子 - 端子応答、または、サーボレベルの設定値の高速更新を実装することができます。

PROFIBUS DP上のアイソクロナスI/O処理

高速の端子 - 端子応答などの用途に使用されます。

サーボサイクルクロックでの実行のためにTOがコンフィグレーションされている場合、TOとモーションコマンドも発行することができます。

- **IPOSynchronousTask/IPOSynchronousTask_2**:
補間サイクルクロックIPO/IPO_2との同期

IPO同期タスクでは、テクノロジーオブジェクトのファンクションに直接の効果を持つタイムクリティカルなファンクションを実装することができます。

ユーザープログラムはインタポレータの前に実行されます。すなわち、プログラムされたファンクションは同一のIPOサイクルクロックで効果を持つことができます。

たとえば、イベントに応じた高速起動、イベント端子 - 軸への高速応答などの用途

さらに、TControl用のSynchronousTasksもあります。

- **PWMSynchronousTask**: PWMサイクルクロックとの同期
- **InputSynchronousTask_1/InputSynchronousTask_2**:
温度制御の前に入力1/2サイクルクロックとの同期
- **PostControlTask_1/PostControlTask_2**:
温度制御の後に制御1/2サイクルクロックとの同期

[Program

assignment]タブでは、作成され、コンパイルされたプログラムを選択したSynchronousTaskに割り当て、それらのプログラムの実行シーケンスを定義することができます。

次のパラメータを設定することができます。

フィールド/ボタン	説明/注
For cycle clock level	これを使用して、プログラムを割り当てるSynchronousTaskを選択します。 1つのSynchronousTaskに複数のプログラムを割り当てることができます。
ServoSynchronousTask	サーボサイクルクロックで起動されるSynchronousTask
IPOSynchronousTask	補間サイクルクロック(IPOサイクルクロック)で起動されるSynchronousTask
IPOSynchronousTask_2	2番目の補間サイクルクロック(IPO2サイクルクロック)で起動されるSynchronousTask
PWMSynchronousTask (TCPWM_Tasks)	TO TController(温度チャネル)の操作信号出力で使用するSynchronousTask。 TO TControllerのその他のタスクのための基本サイクルクロックを定義します。

フィールド/ボタン	説明/注
InputSynchronousTask_1/2 (TCInput_Tasks_1/2)	TO TControllerの現在値測定で使用するSynchronousTask
PostControlTask_1/2 (TCTasks_1/2)	TO TControllerの制御で使用するSynchronousTask
実行システムでのタスクの使用	実行システムでこのタスクを表示 / 使用する場合は、このチェックボックスを有効にします。 このチェックボックスが無効な場合は、このタスクにプログラムを割り当てることはできません。

ServoSynchronousTask

ServoSynchronousTasksは、アイソクロナスモード(V4.0以降)による高速プロセスを実装する用途のために提供されます。

ServoSynchronousTasksは1サーボサイクルクロック内で実行され、動作はIPOSynchronousTasksと類似しています。

ServoSynchronousTaskが適している用途には、以下のものがあります。

- 高速応答(たとえば、I/O処理の高速の端子 - 端子時間)
- サーボ効果を持つシステム変数の更新
システム変数のサーボ効果についての情報は、SIMOTIONリファレンスリストのテクノロジーパッケージCAMシステム変数に提供されています。
- サーボサイクルクロックでの実行用にTO(例外的な場合)がコンフィグレーションされている場合のモーションコマンド

以下の用途には適していません。

- 通信
- IPOサイクルクロックでの実行用にTOがコンフィグレーションされている場合のモーションコマンド(IPOサイクルクロックで評価されるため)

サーボサイクルクロックでの実行のためにTOがコンフィグレーションされている場合、TOとモーションコマンドも発行することができます。

IPOSynchronousTaskの場合と同様の特性が適用されますが、許容値を設定することはできません。

ServoSynchronousTaskは、実行システムでコンフィグレーションすることができます。(サーボ実行レベルのサイクルタイムは、サーボで設定されます)。このデフォルト設定は有効です。

他の周期的タスクの場合と同様に、ServoSynchronousTaskでは、プロセスイメージを使用できます。

これは、同一のI/Oアドレスが複数回アクセスされる場合だけ性能の改善をもたらします。

ServoSynchronousTaskのオーバーフローの場合、CPUは**STOP**に移行します。許容値を設定することはできません。

注記

ServoSynchronousTaskでは、タスクのオーバーフローが許されないため、同期コマンドを使用することはできません。

たとえば、_waittimeファンクションを呼び出すと、CPUはSTOPモードに移行します。

IPOSynchronousTask(IPO同期タスク)/IPOSynchronousTask_2

IPOSynchronousTasksは、たとえば、高速で確実な応答や是正動作を必要とする用途のためのタスクです。

この結果、最適な時点での、モーションコントロールシステムへのモーションコマンドの転送が可能になります。

IPOSynchronousTasksは、インタポレータの直前に、1

IPOサイクルクロック内で実行されます。

このため、これらのタスク内のコマンドは、モーションコントロールに直接に効果を与えることができます。

IPOSynchronousTaskはIPOサイクルクロックと同期して起動されるのに対し、IPOSynchronousTask_2はIPOサイクルクロック2(IPOサイクルクロックを短くしたクロック)と同期して起動されます。

IPOSynchronousTaskは内部IPOTaskの前に実行され、IPOSynchronousTask_2はIPOTask_2の前に実行されます。

IPOSynchronousTasksでは、以下のプロパティを指定する必要があります。

- タスクコンフィグレーション
- IPO/IPO_2サイクルクロックでのレベルオーバーフローの合計数

IPOレベルのタスク(IPOSynchronousTask、IPOTask)が1

IPOサイクルクロック内で完了しない場合、オーバーフローが発生します。

サイクルクロック(n)のタスクのオーバーフローは、次のサイクルクロック(n+1)で処理する必要があります。

IPO_2レベル(IPOSynchronousTask_2、IPOTask_2)のタスクにも上記のことが適用されます。

シーケンスで許容するオーバーフロー数(n = 0 ~ 5)を設定することができます。

(オーバーフローがn回生じることができます)。

タスクがオーバーフローなしで実行されたとき、内部監視カウンタがリセットされます。

累積レベルオーバーフロー数は、numberOfSummarizedTaskOverflowシステム変数に表示されます。

- タイムアウト(レベルオーバーフロー)時のエラー応答:
CPUがSTOPモードに移行し、起動ロックアウトが設定されます。
- IPOSynchronousTask/IPOTask: IPOSynchronousTaskとIPOTaskの間の時間比率。

IPO同期タスクがこの比率を超えると、タイムアウトが発生します。

システムサイクルクロック単位で、IPOTaskの時間フレームも定義します。

IPOSynchronousTask/IPOTask/パラメータでは、IPOSynchronousTaskが使用できるサイクルクロックあたりの時間の割合を指定することができます。

たとえば、4

msがIPOサイクルクロックとして設定され、比率として25%が設定されると、IPOSynchronousTaskのランタイムは、高優先度のタスクの可能な割り込みを含めて、1 msを超えてはいけません。

推奨事項:

サーボのギア比: IPO_1 > 1の場合、可能なかぎり大きいパーセンテージ値を入力してください。

注記

IPOSynchronousTaskを一時停止する同期システムファンクションが呼び出されると、CPUは、診断バッファエントリ付きで、**STOP**モードに移行します(V3.2現在)。

時間監視の実行の有無を設定することができます。

SynchronousTasksのコンフィグレーション

1. 実行レベルツリーの適切なタスクをクリックします。
2. **[For cycle clock level]**選択ボックスで、必要なSynchronousTaskを選択します。

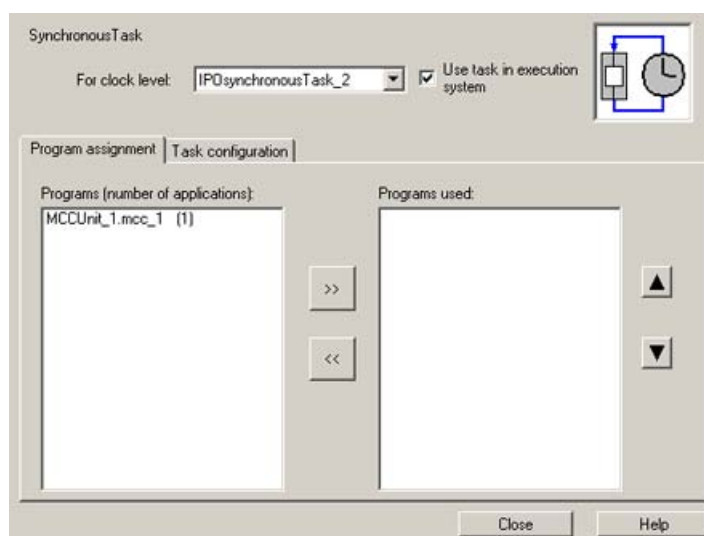


図 5-12 SynchronousTasksのコンフィグレーション(プログラム割り当て)

3. **[Program assignment]**タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

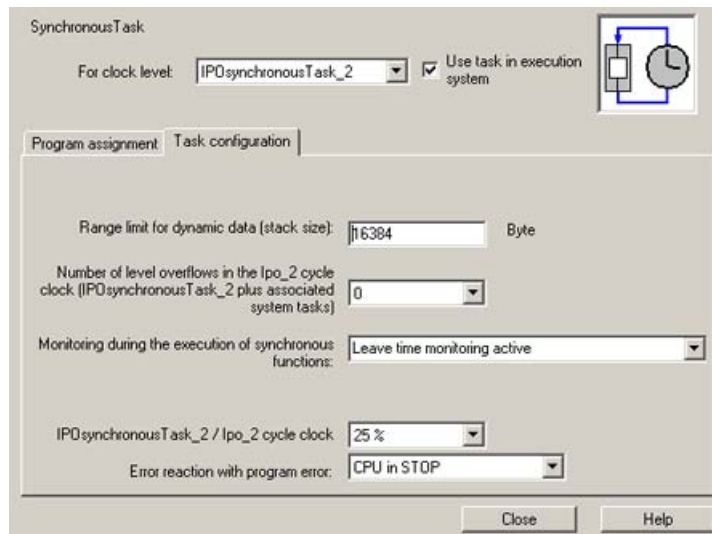


図 5-13 例: IPOsynchronousTaskのコンフィグレーション(タスクコンフィグレーション)

4. [Task Configuration]タブに切り替えます。
5. 以下のオプションも使用できます。
 - スタックサイズ([Task Configuration]タブのオプション)
 - [Level overflows]選択ボックスでの許容レベルオーバーフロー数(0~5)の入力
 - SynchronousTaskとIPOタスクの間の比率の選択
 - [Error reaction with program error]の選択
 - IPOsynchronousTask/IPOTask時間監視のコンフィグレーション
6. コンフィグレーションするすべての同期タスクについて、ステップ3~5を繰り返します。

タスクコンフィグレーション - SynchronousTasks

[Task configuration]タブでは、SynchronousTaskのエラー応答を選択することができます。
次のパラメータを設定することができます。

表 5-3 タスクコンフィグレーション - ServoSynchronousTask

フィールド/ボタン	説明/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Task / cycle clock	ここでは、指定したタスクの継続時間を、指定したサイクルクロックに対するパーセンテージとして選択することができます。 タスクがここで設定された時間より多くの時間を要すると、システムは、エラー応答の設定に従って、応答します。

5.2 ユーザープログラムタスクの概要

フィールド/ボタン	説明/注
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUがSTOPモードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合は、CPUはSTOPモードに切り替わります。エラーが発生したタスクが終了します。

表 5-4 タスクコンフィグレーション - IPOSynchronousTask

フィールド/ボタン	説明/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Number of IPO/IPO2 cycle clock overflows	SynchronousTask用IPO/IPO_2サイクルクロックでの許容レベルオーバーフロー数を入力します。 オーバーフローの数がここで入力されたオーバーフロー数より少ない場合、エラー応答は発生しません。 算術演算を処理するとき、SynchronousTaskおよびSynchronousTask_2実行レベルで、ユーザーが許容できるIPO/IPO_2サイクルクロックでのレベルオーバーフローが発生する場合があります。 実行レベルオーバーフローは、以下の場合に発生します。 - IPOSynchronousTask + IPOタスク > IPOサイクルクロック、または - IPOSynchronousTask_2 + IPO_2タスク > IPO_2サイクルクロック この設定された許容レベルによって、設定されたエラー応答がトリガされないで、次のIPOサイクルクロックが、直前のサイクルクロックの算術演算の終了のために使用されます。 設定されたオーバーフロー数を超えるか、許容値が設定されていない場合、診断バッファにエントリが作成され、エラー応答(CPUが起動ロックスアウト付きでSTOPに移行)が実行されます。 設定できる実行レベルオーバーフロー数は、最大5です。
Monitoring when executing synchronous functions	SynchronousTaskがシステムサイクルクロック内で完了しない場合のエラー応答を選択することができます。
Suspend time monitoring and interrupt task	V3.1.1までの機能と互換性を持ちます 同期ファンクションで、時間監視が一時停止され、1つのIPOサイクルクロックが失われます。すべての古いプロジェクトも、V3.2(CPU置換)に更新された場合、この設定になります。
Leave time monitoring active	時間監視の一時停止は行われず、実際にIPOSynchronousTaskに割り込む同期ファンクションが呼び出されたときに、タイムアウトで、CPUがSTOPに移行します。 IPOオーバーフローが許容されていると、STOPを回避することができません。これは、新しく作成されたV3.2 CPUのデフォルト設定です。 診断バッファエントリ: "STOP by execution system, cause: Timeout".

フィールド/ボタン		説明/注
	CPUがSTOP状態に移行	同期ファンクションは許容されません。 同期ファンクションを呼び出すと、IPOSynchronousTaskが実際には割り込まれていない場合でも、CPUは STOP に移行します。 診断バッファエントリ: "Impermissible calling of a system/package function". この動作は、IPOSynchronousTask、IPOSynchronousTask_2、およびPWMTaskに適用されます。
	Task / cycle clock	ここでは、指定したタスクの継続時間を、サイクルクロック(たとえば、IPOSynchronousTask/IPOサイクルクロック)に対する割合で選択します。 タスクがここで設定された時間より多くの時間を要すると、システムは、エラー応答の設定に従って、応答します。 システムサイクルクロックを設定するには、プロジェクトナビゲータでCPUを選択し、メニューで、[Target system Expert Set system cycle clocks]を選択します。 25%: タスクの最大継続時間は、サイクルクロックの25%です。 50%: タスクの最大継続時間は、サイクルクロックの50%です。 75%: タスクの最大継続時間は、サイクルクロックの75%です。
	Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
	CPU to STOP	CPUが STOP モードに切り替わり、ShutdownTaskが起動されます。
	ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合は、CPUは STOP モードに切り替わります。エラーが発生したタスクが終了します。

下記も参照

フィールドバスシステム上のアイソクロナスI/O処理 (ページ 197)

タイムアウトとレベルオーバーフロー (ページ 182)

実行レベル / タスクへのプログラムの割り当て (ページ 170)

DCCブロック(DCB)のシーケンスモデル (ページ 208)

5.2.6 SystemInterruptTasks

SystemInterruptTasksは、システムイベントが発生したとき起動され、一度だけ実行されます。

以下のSystemInterruptTasksを使用できます。

- **TimeFaultTask:** TimerInterruptTaskタイムアウトが発生したときに起動されます。
- **TimeFaultBackgroundTask:** BackgroundTaskタイムアウトが発生したときに起動されます。

- **TechnologicalFaultTask**: テクノロジーオブジェクトのエラー時に起動されます。
- **PeripheralFaultTask**: I/Oのエラー時に起動されます。
- **ExecutionFaultTask**: プログラム実行時にエラーが発生したときに起動されます。

SystemInterruptTask(システム割り込みタスク)の起動

SystemInterruptTaskは、設定されたイベントが発生した後に自動的に起動されます。

SystemInterruptTaskをトリガするイベントが発生する場合は、実行システムでSystemInterruptTaskを使用し、このタスクに1つのプログラムを割り当てる必要があります。これを行わないと、CPUがSTOPモードに切り替わります。

最大8つの異なる割り込みをバッファに格納することができます。

別の割り込みが発生するとバッファはオーバーフロー状態になり、CPUもSTOPモードに移行します。

SystemInterruptTask呼び出しの原因となったイベントは、このタスクの関連TaskStartInfoを使用して照会することができます。この詳細は、Taskstartinfoの使用(ページ 97)に記載されています。

SystemInterruptTask(システム割り込みタスク)の完了

SystemInterruptTaskは、SystemInterruptTasksに割り当てられたプログラムが完了した後、自動的に完了します。

[Program

assignment]タブでは、作成され、コンパイルされたプログラムを選択したSystemInterruptTaskに割り当て、それらのプログラムの実行シーケンスを定義することができます。

次のパラメータを設定することができます。

フィールド/ボタン	意味/注
For task	これを使用して、SystemInterruptTasksの中で、プログラムを割り当てる1つを選択します。 1つのSystemInterruptTaskに複数のプログラムを割り当てることができます。
ExecutionFaultTask	プログラム処理エラー(たとえば、ゼロによる除算)
PeripheralFaultTask	プロセスアラーム、診断アラーム、モジュールの取り外し / 挿入などのI/Oアラーム
TechnologicalFaultTask	アラーム、警告、およびメモなどのテクノロジーアラーム
TimeFaultBackgroundTask	BackgroundTaskでのタイムアウト
TimeFaultTask	たとえば、SynchronousTasks、TimerInterruptTasks、システムランタイム、およびサイクルタイムでのタイムアウト
アラーム設定	これは、テクノロジーアラームのコンフィグレーション用のウィンドウを表示します。 SystemInterruptTasksのテクノロジーアラームに対するアラーム応答を設定することができます。
実行システムでのタスクの使用	実行システムでこのタスクを表示 / 使用する場合は、このチェックボックスを有効にします。 このチェックボックスが無効な場合は、このタスクにプログラムを割り当てることはできません。

TimeFaultTask

TimeFaultTaskは、**TimerInterruptTask**の時間監視が応答するときに起動されます。**TimeFaultTask**がコンフィグレーションされていないか、タスクにプログラムが割り当てられていない場合、CPUは**STOP**モードに切り替わります。

TimeFaultTaskでは、**TimerInterruptTasks**のタイムアウトに対する応答をプログラミングすることができます。

詳細については、**Taskstartinfo**の使用 (ページ 97)を参照してください。

TimeFaultBackgroundTask

TimeFaultBackgroundTaskは、**BackgroundTask**の時間監視が応答するときに起動されます。

TimeFaultBackgroundTaskがコンフィグレーションされていないか、タスクにプログラムが割り当てられていない場合、CPUは**STOP**モードに切り替わります。

TimeFaultBackgroundTaskでは、**BackgroundTasks**のタイムアウト時の処理をプログラミングすることができます。

TechnologicalFaultTask

TechnologicalFaultTaskは、テクノロジパッケージがアラームまたは情報メッセージを生成したときに起動されます。

TechnologicalFaultTaskがコンフィグレーションされていないか、タスクにプログラムが割り当てられていない場合、CPUは**STOP**モードに切り替わります。

通常、アラームメッセージは、テクノロジパッケージの動作に直接の影響を与えるため、再度テクノロジファンクションを起動する前に、肯定応答する必要があります。

TechnologicalFaultTaskでは、エラーに対して直接に肯定応答したり、マシン上の動作シーケンスに関して、その他の応答を行うことができます。

詳細については、**Taskstartinfo**の使用 (ページ 97)を参照してください。

PeripheralFaultTask

I/Oアクセスエラーの場合、**PeripheralFaultTask**がその優先度に従って迅速に起動されます。

I/Oアクセスエラーは、たとえば、I/Oモジュールの電源異常や、I/Oモジュール上のその他のエラーが起きたときに発生します。

詳細は、*I/Oモジュールの説明*を参照してください。

I/Oアクセス時にエラーが発生したタスクは、**終了しません**。

PeripheralFaultTaskがコンフィグレーションされていないか、タスクにプログラムが割り当てられていない場合、CPUは**STOP**モードに切り替わります。

詳細については、**Taskstartinfo**の使用 (ページ 97)を参照してください。

ExecutionFaultTask

プログラム実行エラーの場合、**ExecutionFaultTask**がその優先度に従って迅速に起動されます。

プログラム実行エラーの例:

- 負の数、不正な数の対数などの、浮動小数点数の演算エラー
- 0での除算
- 配列の制限違反
- システム変数へのアクセス時のエラー

エラーが発生したタスクが終了します。

ExecutionFaultTaskがコンフィグレーションされていないか、タスクにプログラムが割り当てられていない場合、CPUは**STOP**モードに切り替わります。

[CPU to **STOP**]エラー応答はすべてのタスクで可能で、**ShutdownTask**を起動します。SIMOTIONデバイスは、**STOP**モードに切り替わります。

ExecutionFaultTaskのエラー応答は、ExecutionFaultTaskを再起動します。

ExecutionFaultTaskのプログラムのコマンドでは、以下のタスクを再起動することができません。

- StartupTask
- ShutdownTask
- MotionTasks

以下のタスクの場合、ExecutionFaultTaskが完了し、ShutdownTaskが起動されたらすぐに、SIMOTIONデバイスは**STOP**モードに切り替わります。

- BackgroundTask
- TimerInterruptTasks
- SynchronousTasks

詳細については、Taskstartinfoの使用 (ページ 97)を参照してください。

注記

ExecutionFaultTaskと**ShutdownTask**でプログラムエラーが発生すると、システムは直ちに**STOP**モードに切り替わります。

SystemInterruptTasksのコンフィグレーション

1. 実行レベルツリーの[SystemInterruptTasks]をクリックします。
2. [For task]選択ボックスに表示されるタスクの1つを選択します。

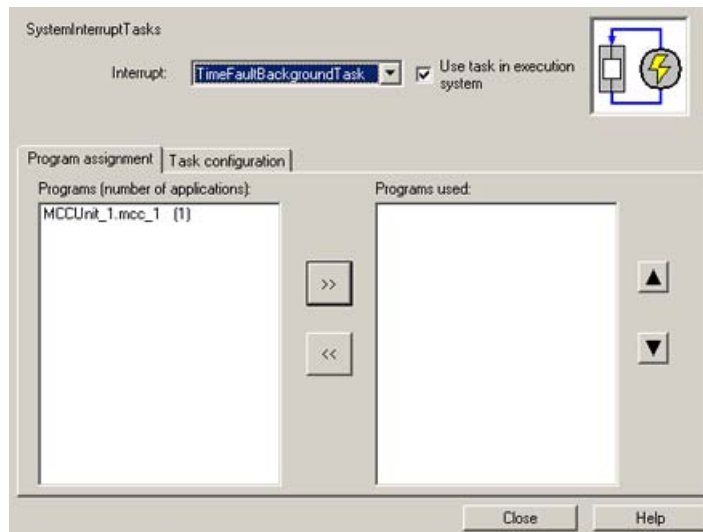


図 5-14 SystemInterruptTasksのコンフィグレーション(プログラム割り当て)

3. **[Program assignment]**タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

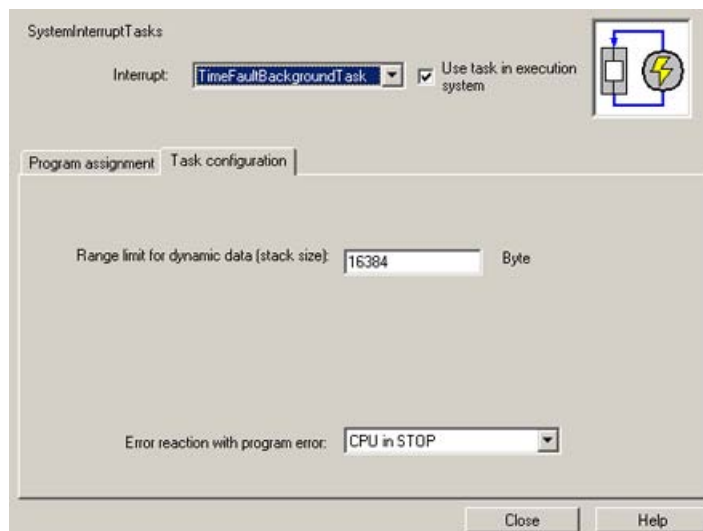


図 5-15 SystemInterruptTasksのコンフィグレーション(タスクコンフィグレーション)

4. **[Task Configuration]**タブに切り替えます。
5. タスクをコンフィグレーションします。
6. コンフィグレーションするすべての同期タスクについて、ステップ3~5を繰り返します。
7. テクノロジアラームをコンフィグレーションするには:
[Alarm configuration]をクリックします。

タスクコンフィグレーション - SystemInterruptTasks

[Task configuration]タブで、プログラムエラーに対するエラー応答のパラメータを設定します。
次のパラメータを設定することができます。

フィールド/ボタン	意味/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUがSTOPモードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合、CPUはSTOPモードに切り替わります。エラーが発生したタスクが終了します。

下記も参照

タスクの起動に関する情報: TaskStartInfo(TSI) (ページ 185)

5.2.7 UserInterruptTasks

UserInterruptTasksは、ユーザー定義アクションのためのタスクです。

2つのUserInterruptTasks: UserInterruptTask_1とUserInterruptTask_2を使用できます。

UserInterruptTaskでは、1つの定義済み条件を指定する必要があります。

その条件が満たされる度に、UserInterruptTaskが起動されます。

UserInterruptTaskを使用する場合は、実行システムで、IPOsynchronousTaskを使用する必要があります。

UserInterruptTasksは、StartupTaskとShutDownTaskの処理中には有効ではありません。

UserInterruptTask(ユーザー割り込みタスク)の起動

UserInterruptTasksは、ユーザー定義割り込み条件が満たされたらすぐに自動的に起動されます。割り込み条件は、補間サイクルクロックでチェックされます。

両方のUserInterruptTasksが同時に起動されると、UserInterruptTask_2の前に、UserInterruptTask_1が処理されます。

注記

停止時にユーザー定義割り込みが発生しても、UserInterruptTaskは起動されません。

停止(ShutdownTask)時のUserInterruptTaskの起動は、_startTask()コマンドを使用することによってのみ行うことができます。

UserInterruptTask(ユーザー割り込みタスク)の完了

UserInterruptTasksは、UserInterruptTaskに割り当てられたプログラムが完了した後、自動的に完了します。

[Program

assignment]タブでは、作成され、コンパイルされたプログラムを選択したUserInterruptTaskに割り当て、それらのプログラムの実行シーケンスを定義することができます。

次のパラメータを設定することができます。

フィールド/ボタン	意味/注
For task	これを使用して、2つのUserInterruptTasksの中で、プログラムを割り当てる1つを選択します。 1つのUserInterruptTaskに複数のプログラムを割り当てることができます。
UserInterruptTask_1 and UserInterruptTask_2	UserInterruptTaskの事前定義された名前
Defined condition	ここでは、選択したUserInterruptTaskのIEC 61131に準拠した条件を定義します。指定された条件は、補間サイクルクロックでチェックされます。 条件が満たされると、UserInterruptTaskが、割り当てられたプログラムとともに起動されます。 シンボルブラウザ(ドラッグ&ドロップによる変数の設定)とプロジェクトナビゲータの[Command library]タブ(ドラッグ&ドロップによる演算子の設定)を使用して、入力フィールドに条件を入力します。 単純な条件(入力とローカルCPU変数の論理演算)とブール変数だけが許容されます。
実行システムでのタスクの使用	実行システムでこのタスクを表示 / 使用する場合は、このチェックボックスを有効にします。 このチェックボックスが無効な場合は、このタスクにプログラムを割り当てることはできません。

UserInterruptTasksのコンフィグレーション

1. 実行レベルツリーの[UserInterruptTasks]をクリックします。
2. [For task]選択ボックスで、1つのUserInterruptTaskを選択します。
3. このタスクを起動する条件を指定します。

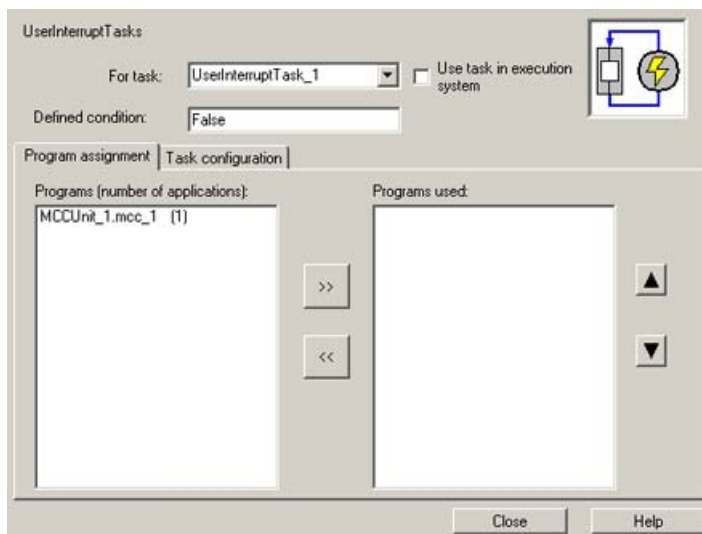


図 5-16 UserInterruptTasksのコンフィグレーション(プログラム割り当て)

4. **[Program assignment]**タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

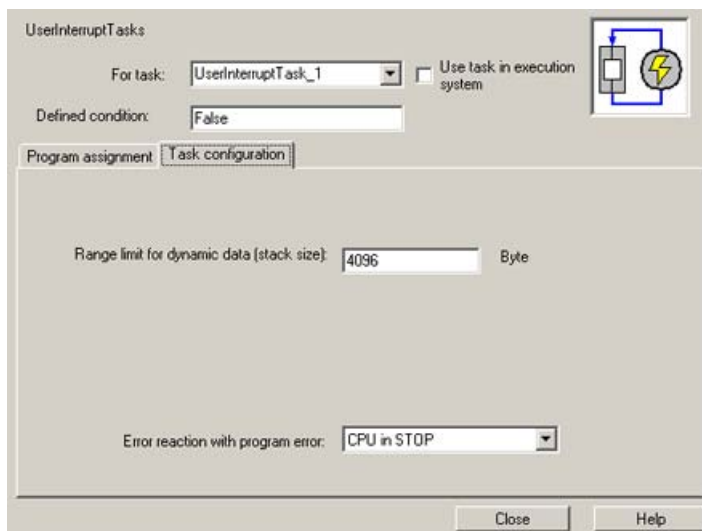


図 5-17 UserInterruptTasksのコンフィグレーション(タスクコンフィグレーション)

5. **[Task Configuration]**タブに切り替えます。
6. タスクをコンフィグレーションします。
7. 2番目のUserInterruptTaskについて、ステップ3～6を繰り返します。

UserInterruptTaskの条件の作成

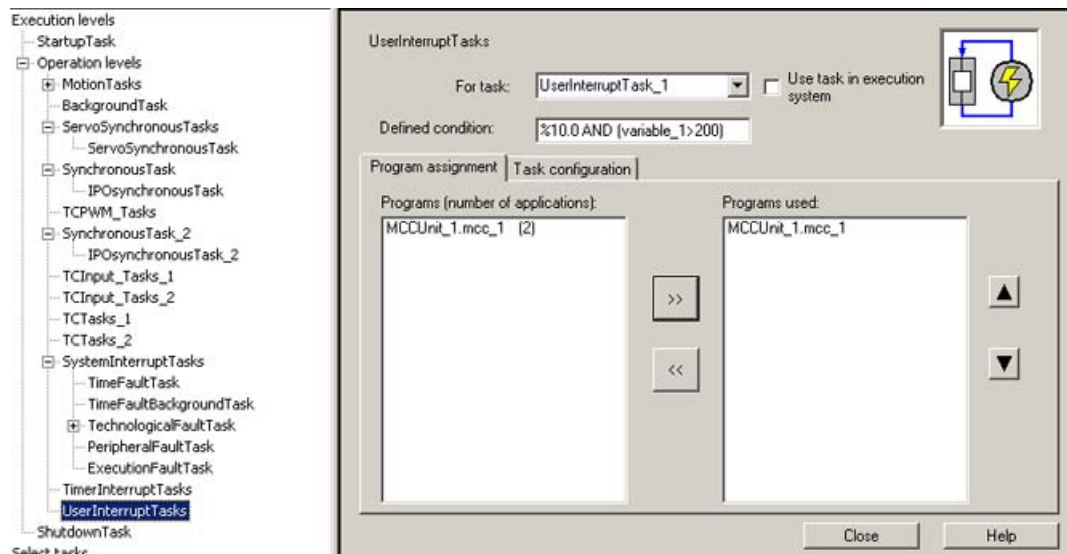


図 5-18 条件付きのUserInterruptTasksのコンフィグレーション

図の凡例 - 例:

デジタル入力0.0 = TRUEで、グローバル変数"variable_1"の値 > 200.0の場合、UserInterruptTask_1が一度だけ実行されます。

UserInterruptTaskを起動する条件は、IEC 61131に準拠した式(構造化テキスト)として入力します。以下の変数を使用することができます。

- グローバルデバイス変数
- ST/MCCまたはLAD/FBDソースファイルのユニット変数

SIMOTION SCOUTは、式の作成を支援します。

UserInterruptTaskを起動する条件を作成するには、以下の手順に従います。

1. **[For task]**で、起動条件を定義するUserInterruptを選択します。
2. 必要な演算子を**[Command library]**タブで見つけ、ドラッグ&ドロップによって、**[Defined condition]**テキストフィールドに挿入します。
その代わりに、テキストフィールドに直接に演算子を入力することもできます。
3. シンボルブラウザでオペランドを見つけ、ドラッグ&ドロップまたはコピー&貼り付けによって、**[Defined condition]**テキストフィールドに挿入します。
その代わりに、テキストフィールドに直接にオペランドを入力することもできます。

コマンドライブラリ

[Command library]タブには、数学的演算および関数の一覧が表示されます。

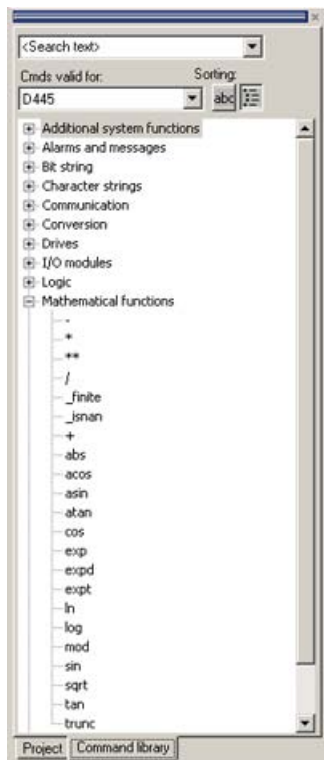


図 5-19 SCOUTのコマンドライブラリ

個々の演算を開くには、フォルダの前の+記号をクリックします。r. 演算子を、ドラッグ&ドロップによって、必要なテキストフィールドにコピーすることができます。これらのコマンドを使用して、たとえば、条件を定義します。

タスクコンフィグレーション - UserInterruptTasks

[Task configuration]タブで、プログラムエラーに対するエラー応答のパラメータを設定します。次のパラメータを設定することができます。

フィールド/ボタン	意味/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このスタックサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
Error reaction to program error	プログラム処理時にエラーが発生したときのエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUがSTOPモードに切り替わり、ShutdownTaskが起動されます。
ExecutionFaultTask	ExecutionFaultTaskが起動されます。 このタスクに割り当てられたすべてのプログラムが起動されます。 プログラムが割り当てられていない場合、CPUはSTOPモードに切り替わります。エラーが発生したタスクが終了します。

下記も参照

実行レベル / タスクへのプログラムの割り当て (ページ 170)

5.2.8 ShutdownTask

ShutdownTaskは、RUNモードからSTOPU/STOPモードに移行するときの選択的介入または緊急停止シーケンスのプログラミング(たとえば、選択された出力設定、軸の定義済み停止)のためのタスクです。

ShutdownTaskは、電源故障の場合には呼び出されません。

ShutdownTaskの場合、[Task configuration]で、**時間監視**を指定する必要があります:
ShutdownTaskの実行の最大継続時間を設定することができます(0 ms = 監視なし)。
この時間の後、CPUはSTOPモードに切り替わります。

ShutdownTaskでは、I/Oに直接にアクセスすることができます。
プロセスイメージとシンボルI/O変数へのアクセスは、禁止されています。
プロセスイメージはもう更新されないため、プロセスイメージによってI/Oにアクセスしても意味がありません。

詳細は、『SIMOTION
ST構造化テキスト』の「入力および出力へのアクセス(プロセスイメージ、I/O変数)」を参照してください。

[Program
assignment]タブを選択し、作成され、コンパイルされたプログラムをShutdownTaskに割り当て、それらのプログラムの実行シーケンスを定義します。

注記

ExecutionFaultTaskとShutdownTaskでプログラムエラーが発生すると、システムは直ちにSTOPモードに切り替わります。

ShutdownTaskのコンフィグレーション

1. 実行レベルツリーの[ShutdownTask]をクリックします。

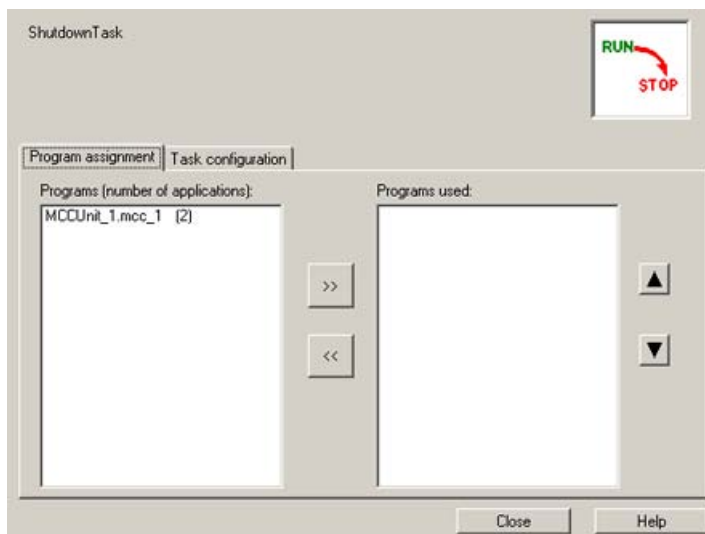


図 5-20 ShutdownTaskのコンフィグレーション(プログラム割り当て)

2. **[Program assignment]**タブで、このタスクに必要なプログラムを割り当て、実行シーケンスを定義します。

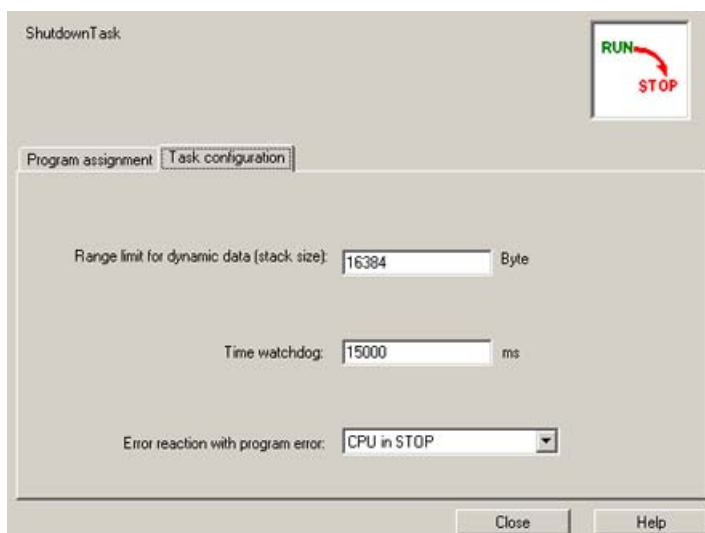


図 5-21 ShutdownTaskのコンフィグレーション(タスクコンフィグレーション)

3. **[Task Configuration]**タブに切り替えます。
4. 必要な場合、**[Range limit for dynamic data]**(スタックサイズ)を入力します。エラーブックマークが定義されていません。
5. 時間監視用の値を入力します。
6. **[Error reaction with program error]**(たとえば、**ExecutionFaultTask**)を指定します。

タスクコンフィグレーション - ShutdownTask

[Task configuration]タブを選択し、時間監視のパラメータを設定します。

次のパラメータを設定することができます。

フィールド/ボタン	意味/注
Limits for dynamic data	このタスクのスタックサイズをバイト単位で入力します。 このタスクに割り当てられたプログラムが実行されるとき、スタックで、このサイズがデータ用として使用可能になります。 標準的な値は、タスクあたり16 KBです。
ウォッチドッグ	ShutdownTaskを実行するサイクルタイムをms単位で入力します。 時間監視用の値を入力します。 0を入力するか、値を入力しないと、時間監視は行われません。
Error reaction with program error	プログラム実行時にエラーが発生した場合のエラー応答を選択します。 プログラムエラーには、たとえば、浮動小数点演算エラー、ゼロ除算、オーバーシュート配列制限エラーなどがあります。
CPU to STOP	CPUは、STOPモードに切り替わります。

下記も参照

実行レベル / タスクへのプログラムの割り当て (ページ 170)

ウォッチドッグ (ページ 185)

5.3 実行システムのコンフィグレーション

実行システムのコンフィグレーションは、以下の手順で行います。

- ユーザープログラムの割り当てとタスクプロパティの定義
- 使用するタスクの有効化
- サイクルクロックソースの選択とシステムサイクルクロックの設定

5.3.1 実行レベル / タスクへのプログラムの割り当て

プログラムは、実行レベルに割り当てる必要があります。

実行レベルに割り当てた場合だけ、プログラムは実行されます。

MCCチャート、STソースファイル、またはLAD/FBDソースファイルのプログラムを作成した後、SIMOTION SCOUTで、1つまたは複数のタスクに割り当てることができます。

1つのタスクに複数のプログラムを割り当てることができます。

この後、割り当てられたプログラムは、リストされた順序で実行されます。この順序は、SIMOTION SCOUTを使用して、指定 / 変更することができます。

複数のプログラムを1つのタスクに割り当てた場合、同一のタスクの次のプログラムを開始する前に、最初のプログラムの実行を終了する必要があります。

たとえば、最初のプログラムが継続ループの場合、2番目のプログラムは決して実行されません。

1つのプログラムを複数のタスクに割り当ててもできます。この場合、プログラムは相互に独立して実行されます。

プログラム割り当てでは、以下を指定します。

- プログラムの実行優先度
- 実行動作: シーケンシャル / 周期的
- プログラム変数の初期化動作

『SIMOTION MCCプログラミングマニュアル』、または、『SIMOTION ST』の「変数初期化の時間」と変数初期化に対するコンパイラの影響 (ページ 232)を参照してください。

プログラムを1つまたは複数のタスクに割り当てるときは、以下の点に注意してください。

- プログラムは、タスクに割り当てる前に、エラーなしでコンパイルする必要があります。
- タスクへの割り当ては、プログラムをターゲットシステムへダウンロードする前に行う必要があります。
- プログラムは、1つのタスクで実行中に、別のタスクが呼び出すことができます(複数のタスクが割り当てられている場合)。
システムでは、データ一貫性を保証する措置は取られていません。
- プログラムを一旦タスクに割り当てると、再コンパイルしても、プログラムは割り当てられたままです。

注記

DCCタスクは、DCCエディタが割り当てられます。DCCエディタの説明を参照してください。

実行システム - プログラム割り当て

[Program assignment]タブでは、作成され、コンパイルされたプログラムを実行レベルの各種のタスクに割り当てることができます。

次のパラメータを設定することができます。

フィールド/ボタン	意味/注
プログラム	プロジェクトで利用できるすべてのコンパイルされたプログラムの一覧が、ここに表示されます。 コンパイルされていないプログラムは表示されません。 プログラム名の後の番号は、そのプログラムが実行レベルの異なるタスクに割り当てられた回数を示します。
割り付け	これを使用して、選択したプログラムをタスクに割り当てます。 [Program]一覧でプログラムを選択し、[Assign]をクリックします。 プログラムがタスクに割り当てられ、タスクリストに表示されます。
削除	これを使用して、タスクに割り当てられたプログラムを割り当て解除します。 [Task]リストでプログラムを選択し、[Remove]をクリックします。 プログラムが、タスクから削除されます。
タスク	このタスクに割り当てられたすべてのプログラムの一覧が、ここに表示されます。 この一覧のプログラムの順序は、プログラムの実行順序に対応します。 一覧の一番上のプログラムは最初に実行されます。
Arrow up	この上矢印を使用して、選択したプログラムをタスク内の1つ上の位置に移動します。 これによって、タスク内のプログラム実行の順序を設定することができます。
矢印、下方向	この下矢印を使用して、選択したプログラムをタスク内の1つ下の位置に移動します。 これによって、タスク内のプログラム実行の順序を設定することができます。

タスクへのプログラムの割り当て

1. プロジェクトナビゲータで、SIMOTIONデバイスを選択し、メニューの[Target system|Configure execution system]を選択するか、[EXECUTION SYSTEM]をダブルクリックします。

[EXECUTION SYSTEM]ウィンドウが、ワークベンチの作業エリアに開かれます。

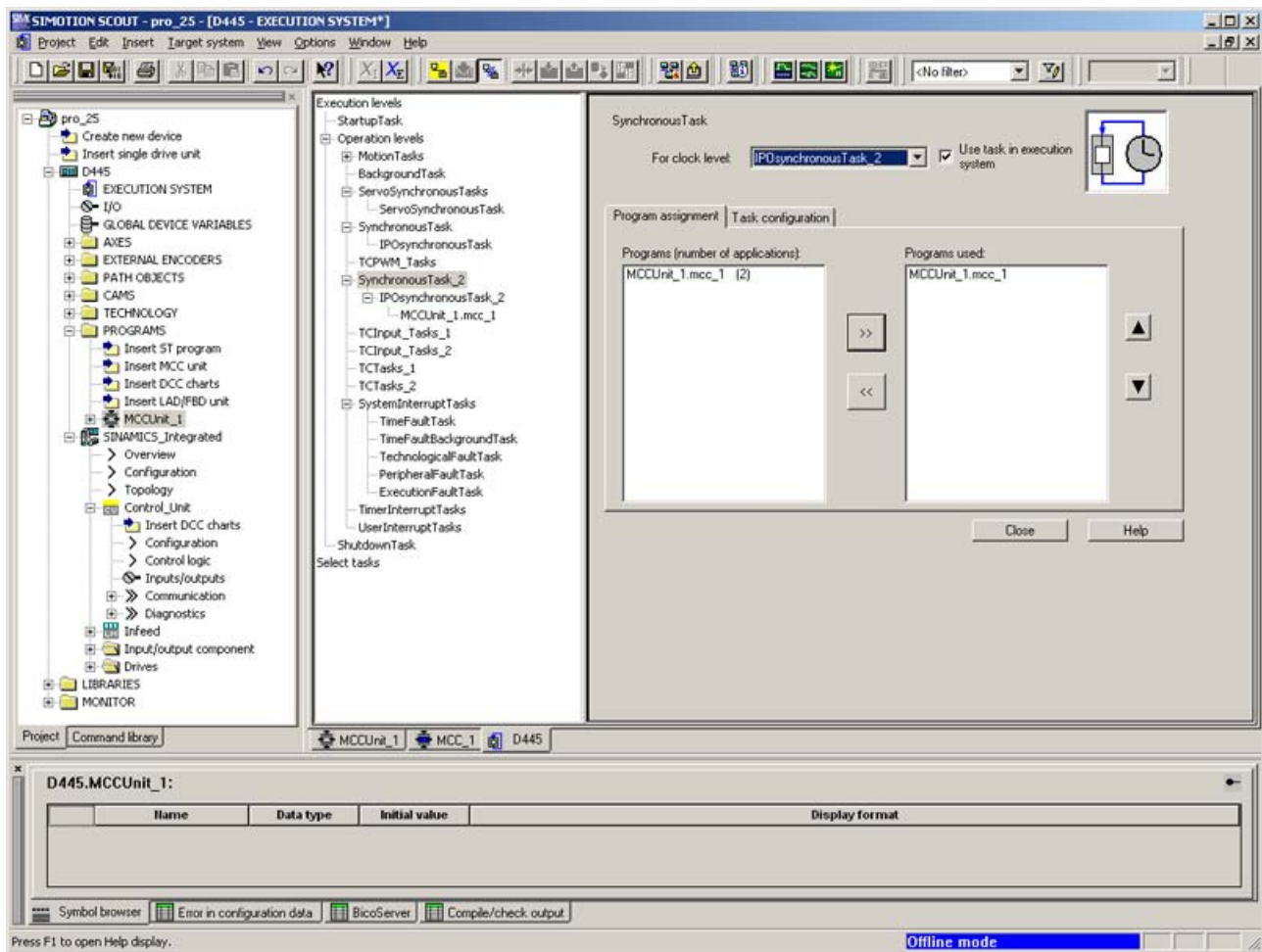


図 5-22 SCOUTでの実行システムのコンフィグレーション

ウィンドウの左のペインには、実行レベルツリーが表示されます。
固定エントリとして使用可能な実行レベル / タスクが表示されます。

[OperationLevels]フォルダには、RUNモードで使用できるタスクが収納されています。

それぞれの実行レベルまたはタスク名の下のリストには、コンフィグレーションされたタスクと、タスクに割り当てられたプログラムが表示されます。

1. コンフィグレーションするタスクを選択します。

2. [Program assignment]タブを選択します。

左の一覧ボックスには、すべての使用可能なプログラム([Task]作成タイプのSTプログラム、MCCチャート、LAD/FBD)が表示されます。

3. この左の一覧ボックスで、タスクに割り当てるプログラムを選択します。

4. >> をクリックします。

右の一覧ボックスに、割り当てられたプログラムが表示されます。

割り当てられたプログラムは、左の一覧ボックスにも表示されたままです。

プログラムは、複数のタスクに割り当てることができます。

割り当てられた回数が、括弧付きで表示されます。

5. [Task configuration]タブを選択し、たとえば以下の追加設定を行います。

- Error reaction with program error
- Time watchdogs for cyclic tasks
- Start behavior of MotionTasks

プログラムを1つまたは複数のタスクに割り当てた後、ターゲットシステムとの接続を確立し、ターゲットシステムにプロジェクトをダウンロードして、それを起動することができます。

実行シーケンスの変更

プログラムは、入力された順序で実行されます。この順序は変更することができます。

1. 右の一覧ボックスで、移動するエントリを選択します。
2. ▲または▼を選択して、エントリを上下に移動します。
3. 必要な数だけ、ステップ2を繰り返します。

タスク名

MotionTasksの名前は、変更することができます。

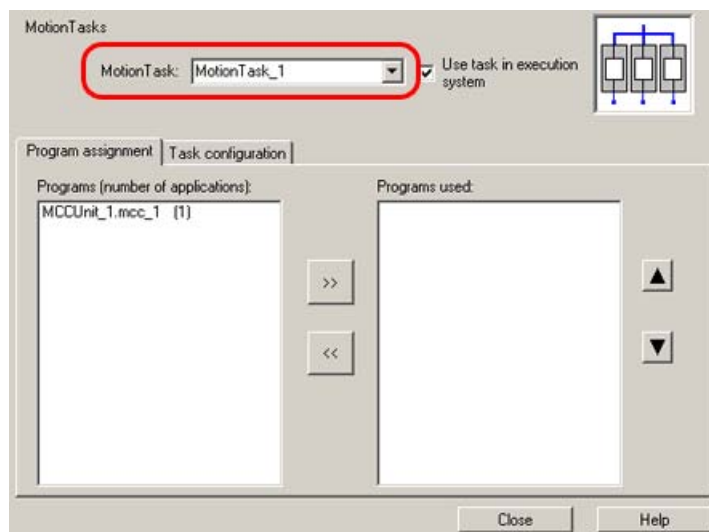


図 5-23 SCOUTのタスク名

1. 左の欄で、必要なタスクを選択します。
2. 右のドロップダウンリストで、新しいタスク名を入力します。

タスク制御

MCC、ST、およびLAD/FBDプログラミング言語では、タスク制御用の複数のコマンドが提供されています(start、stopなど)。

See **SIMOTION MCC**、**SIMOTION ST**、および**SIMOTION LAD/FBD**のプログラミングマニュアルを参照してください。

これらのプログラミングマニュアルでは、タスク制御コマンドの使用方法に関する情報と複数の例も提供されています。

スタックのサイズ

SIMOTION SCOUT(V3.0現在)では、[Task Configuration]ウィンドウの[Task configuration]タブで、タスクのスタックサイズ(動的データの制限値)を設定することができます。タスクごとにデフォルト値が指定されています。

詳細は、『**SIMOTION ST**プログラミングマニュアル』の「ローカルデータスタックのサイズの設定」を参照してください。

下記も参照

タスク優先度 (ページ 133)

5.3.2 サイクルクロックソースの選択

サイクルクロックソースの選択は、[HW configuration]のデバイスコンフィグレーションに実装されています。

インターフェースの1つをアイソクロナスDP/PNインターフェースとして設定すると([Properties]ダイアログボックスでの[Isynchronous mode]の選択)、サイクルクロック設定がバスサイクルクロックとして使用されます。

DP/PN通信レベルと、サーボおよびインタポレータレベルは、バスサイクルクロックと同期化されます。この設定は、PROFIBUS DP/PROFINETで、PROFIDRIVE V3に準拠したデジタルドライブと組み合わせて、TPCam/TPCam_extテクノロジーパッケージのモーションコントロールファンクションを利用する場合、必須です。この通信をサポートし、必要とするドライブにはたとえば: SIMODRIVE 611U、MASTERDRIVES MOTION CONTROL、SINAMICSがあります。アプリケーションからI/Oへアイソクロナスアクセスを行わなければならない場合、バスサイクルクロックとの同期化が必要です。

現在でも、アイソクロナスバス上のドライブ軸としてアイソクロナスモードをサポートしないドライブを使用することもできます。このようなドライブには、Micromaster MM4、MASTERDRIVES VCなどがあります。

アイソクロナスDP/PNインターフェースを使用しない場合は、基本システムサイクルクロックを設定することができます。

サーボおよびインタポレータレベルは、基本サイクルクロックと同期化されます。TPCamテクノロジーパッケージを使用しない場合、または、SIMOTIONのアナログドライブだけと組み合わせてそれを使用する場合は、この設定を選択することができます。

5.3.3 システムサイクルクロックの定義

サイクルクロックソースを選択した後、基本サイクルクロックに基づく、アイソクロナス実行レベルのサンプリング時間を指定します。

これには、次のクロックが含まれます。

- **バスサイクルクロック(DPサイクルクロック / PNサイクルクロック):**
実行システム - システムサイクルクロックの表を参照してください。
- **サーボサイクルクロック(位置制御サイクルクロック)/T1(DCC)サイクルクロック**

入力 / 出力は、サーボサイクルクロックで更新されます。

軸の位置制御と集中 / 分散I/O処理は、サーボサイクルクロックで実行されます。

サーボサイクルクロックは、バスサイクルクロックに対して、1:1または1:2の比率で動作させることができます。

注記

このバスサイクルクロックとサーボサイクルクロックとの間の縮小率は、ドライブ上でも、マスタアプリケーションサイクルとして設定する必要があります。

この設定は、相互寿命監視のために必要です。

詳細は、ドライブの説明書を参照してください。

- **補間サイクルクロック(IPOサイクルクロック)/T2(DCC)サイクルクロック**

軸モーションコントロールは、IPOサイクルクロックで計算されます。

IPOSynchronousTaskは、このサイクルクロックで実行されます。

IPOサイクルクロックは、サーボサイクルクロックに対して、1:1 ~ 1:6の比率で減らすことができます。

- **補間サイクルクロック2(IPO_2サイクルクロック)/T3(DCC)サイクルクロック**

IPOサイクルクロック2は、低優先度の軸のモーションコントロールの基礎となります。

IPOSynchronousTask_2とPWM

Task(TControl)は、このサイクルクロックで実行されます。

IPO_2サイクルクロックは、IPOサイクルクロックに対して、1:2 ~ 1:64の比率で減らすことができます。

- **DccAux(DCC) - サイクルクロック**

DCCAuxサイクルクロックは、T3(DCC)サイクルクロックに対して、1:2 ~ 1:32の比率で減らすことができます。

- **DCCAux_2(DCC)サイクルクロック**

DCCAux_2サイクルクロックは、DCCAuxサイクルクロックに対して、1:2 ~ 1:32の比率で減らすことができます。

- **PWMサイクルクロック:** TControlテクノロジパッケージ用
-

注記

システムサイクルクロックの設定は、システム動作にかなり影響を与えるため、この設定を行うときは、注意が必要です。

システムサイクルクロックの設定

1. メインメニューの[Target system|Expert|Set system cycle clocks...]を使用するか、デバイスのコンテキストメニューまたは[EXECUTION SYSTEM]コンテキストメニューの[Expert|Set system cycle clocks]を使用して、コンフィグレーションウィンドウを呼び出します。

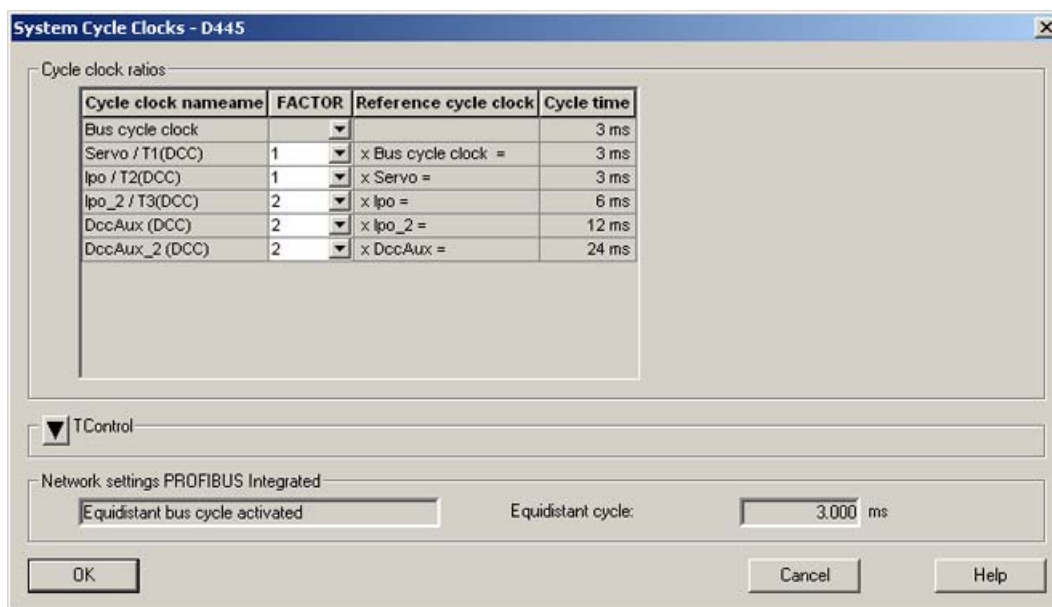


図 5-24 SCOUTでのシステムサイクルクロックの設定

2. 基本 / バスサイクルクロックを指定します(アイソクロナスモードが設定されていない場合だけ)。

可能な値: PROFIBUSの場合1.0 ~ 8.0 ms、

PROFINET付きSIMOTION P(V4.0)およびSIMOTION D445の場合: 0.5 ~ 4.0 ms
、PROFINET付きSIMOTION P(V4.1)の場合: 0.25 ~ 4 msです。

バスサイクルタイムは、PROFIBUSの場合は0.125 msの倍数、C2xxの場合は0.25 msの倍数でなければなりません。

PROFINETのバスサイクルタイムは、0.125

msの倍数です。倍数のなっていない場合は、HWコンフィグレーションで変更します。

3. サイクルクロック間の比率を指定します。

注記

大規模なハードウェア構成で基本 / バスサイクルクロックを短く設定しすぎると、CPUがRUNモードに切り替わらない場合があります。

この場合は、タイムアウト用診断バッファのエントリを調べます。

DCC

DCCのエントリは、チャートをプログラムに追加したときに表示されます。

DCCチャートは、関連タスクが起動されたときに自動的に起動されます。

TControl用システムサイクルクロックの設定

1. **[System Cycle Clocks]**コンフィグレーションウィンドウの**[TControl]**ボタンを使用して、TControlシステムタスクに関する設定を表示することができます。
2. **[Use system tasks for TControl]**チェックボックスを使用して、TControlシステムタスクを有効 / 無効にすることができます。

[Use system tasks for TControl]を有効にすると、温度チャネル用の特殊なタスクが実行システムに表示されます。

温度チャネルを設定していない場合は、不要な計算時間を省くために、TControl用のシステムタスクを無効にする必要があります。
3. さらに、ここでは、TControlタスクのサイクルクロック比率を設定することもできます。
 - **PWM**
サイクルクロックは、位置制御サイクルクロックの倍数でなければなりません。
 - **InputTask_1/2**のサイクルクロックは、PWMサイクルクロックに応じて変わります。
 - **PostTask_1/2**のサイクルクロックは、**InputTask_1/2**のサイクルクロックに応じて変わります。

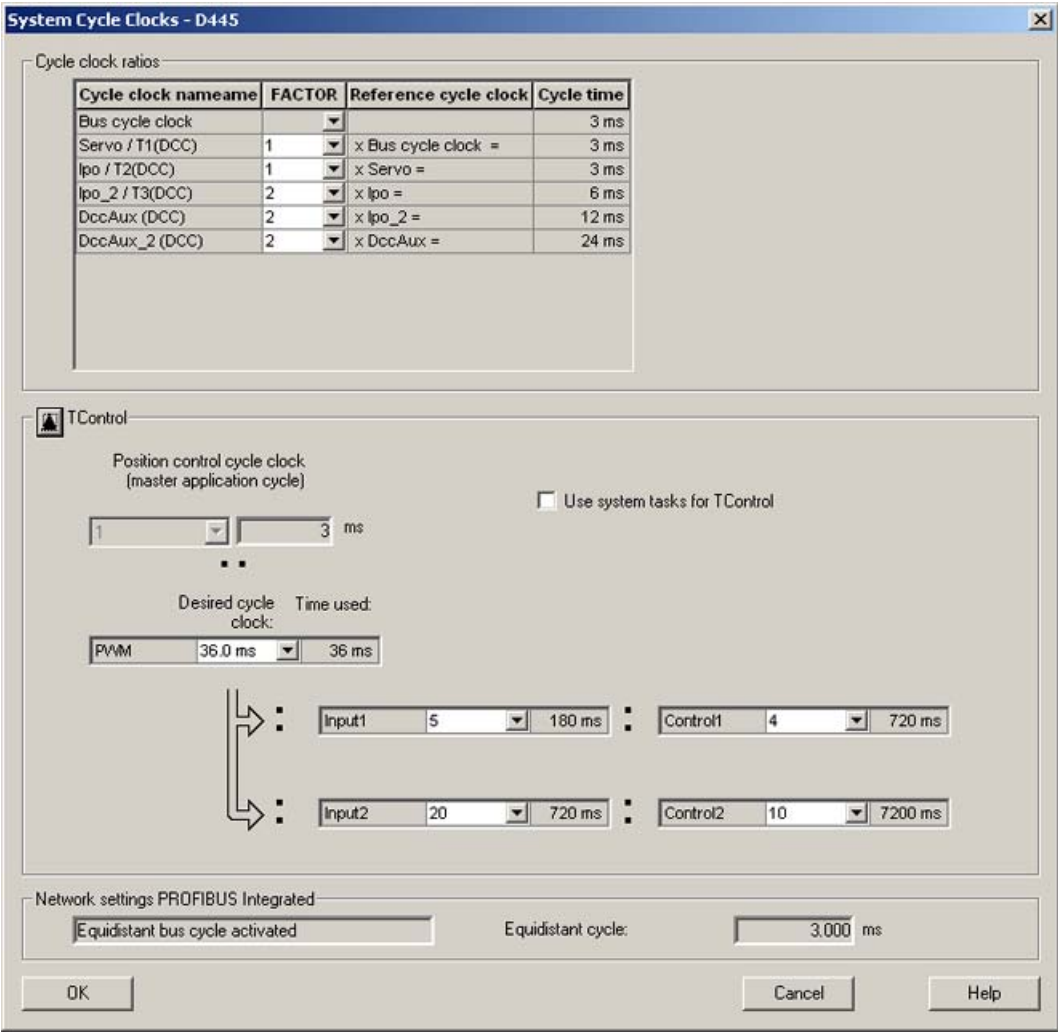


図 5-25 SCOUTでのTControl用システムサイクルクロックの設定

実行システム - システムサイクルクロック

次のパラメータを設定することができます。

フィールド/ボタン	意味/注
サイクルクロック率	
基本サイクルクロックまたはバスサイクルクロック	基本 / バスサイクルクロック(DPサイクルクロックまたはPNサイクルクロック)は、その他のタスクの設定の基礎として使用されます。

フィールド/ボタン	意味/注
サーボ / T1(DCC)	ここでは、サーボサイクルクロックとバスサイクルクロックとの間の整数比率を設定することができます。 たとえば、軸の位置制御はこのサイクルクロックで計算されます。 通常、整数比率1を使用します。 整数比率2を設定すると、コントローラの動的性能は劣化しますが、他のタスクの処理により多くの計算時間を使用できます。 このバスサイクルクロックとサーボサイクルクロックとの間の縮小率は、ドライブの"マスタアプリケーションサイクル"としても設定する必要があります。 この設定は、相互寿命監視のために必要です。 詳細は、ドライブの説明書を参照してください。 アイソクロナスモードが設定されていない場合は、基本時間とサーボサイクルクロックの比率のデフォルト設定は1:1で、このデフォルト設定を変更することはできません!
lpo / T2(DCC)	ここでは、IPOサイクルクロックとサーボサイクルクロックとの間の整数比率を設定することができます。 通常、軸のモーションコントロールは、この補間サイクルクロックで計算されます。 補間サイクルクロックの整数比率は、ドライブ設定値の計算速度を決定します。
lpo2 / T3(DCC)	ここでは、IPO_2サイクルクロックとIPOサイクルクロックとの間の整数比率を設定することができます。 この補間サイクルクロック2は、低優先度の軸のモーションコントロールに使用されます。 この整数比率は、低優先度のドライブ設定値の計算速度を決定します。
DCCAux (DCC)	ここでは、DCCAux DCCサイクルクロックとT3(DCC) DCCサイクルクロックとの間の整数比率を設定することができます。
DCCAux_2(DCC)	ここでは、DCCAux_2 DCCサイクルクロックとDCCAux DCCバスサイクルクロックとの間の整数比率を設定することができます。
TControl	この矢印をクリックして、TP TControl用システムタスクのコンフィグレーションを開きます。 このコンフィグレーションでは、温度チャンネルのタスクのサイクルクロックとサーボサイクルクロックの比率を設定することができます。
Use system tasks for TControl	温度チャンネル用の特殊なタスクを実行システムに表示する場合、このチェックボックスを有効にします。 温度チャンネルを設定していない場合は、不要な計算時間を省くために、TControl用のシステムタスクを無効にする必要があります。
Servo cycle clock (master application cycle)	ここには、参照用に、サーボサイクルクロックが表示されます。 この値を変更するには、ダイアログボックスの上部で、その値を変更する必要があります。 温度チャンネルの他のすべてのサイクルクロックは、サーボサイクルクロックの倍数です。

フィールド/ボタン		意味/注
	PWM (Pulse Width Modulation)	このサイクルクロックは、温度チャネルの操作信号出力のために使用されます。 その他のサイクルクロックのための基本サイクルクロックを指定します。サイクルクロックは、msまたはHz単位で選択します。Hzを選択すると、入力サイクルクロックおよび制御サイクルクロックに対する比率が自動的に指定されます。
	Input1/2	このサイクルクロックは、温度チャネルの現在値測定のために使用されます。 PWMサイクルクロックに対するこのサイクルクロックの比率を選択します。この時間は、淡色表示のフィールドに表示されます。
	Control1/2	このサイクルクロックは、温度チャネルの閉ループ制御のために使用されます。 入力サイクルクロックに対するこのサイクルクロックの比率を選択します。この時間は、淡色表示のフィールドに表示されます。
ネットワーク設定		
	Network settings at the DP master submodule X9	PROFIBUSインターフェースでアイソクロナスモードが設定されたかどうかを示します。この表示は、参照専用です。
	Isochronous bus cycle activated	アイソクロナス動作が、PROFIBUSインターフェースで設定されています。
	Isochronous bus cycle not activated	アイソクロナス動作が、PROFIBUSインターフェースで設定されていません。
Isochronous bus cycle		フィールドバスインターフェースでアイソクロナスモードが設定された場合に、フィールドバスサイクルがms単位で表示されます。 この表示は参照専用で、HWコンフィグレーションで計算されたアイソクロナスモードに対応します。

システムサイクルクロックの割り当て

テクノロジーオブジェクトに優先度を割り当て、システムサイクルクロックを明示的に割り当てることによって、SIMOTION CPUの性能条件を有効に活用することができます。低優先度のタスクの定義によって、高優先度のタスクのために性能条件を利用することができます。

その結果、特に大幅な加速 / 遅延なしの均一のモーションで、TOに対して、高いタイムレベルを使用することができます。

(たとえば、インタポレートサイクルクロック2でのTOの呼び出し)。

以下の場合、標準設定を変更します。

- 長く続きすぎるジョブ処理が検出された場合
- CPUでのテクノロジーの使用率が大きすぎる場合

TO軸と低優先度のタスクを含むTO

externalEncoderには補間サイクルクロック2を割り当て、TO outputCamとTO measuringInputにはIPOサイクルクロックまたは補間サイクルクロック2を割り当てます。

高優先度のタスクを含むテクノロジーオブジェクトには、補間サイクルクロックまたはサーボサイクルクロックを割り当てます。

テクノロジーオブジェクトには、以下のサイクルクロックを割り当てることができます。

モーションコントロールタスク	高優先度	...	低優先度
テクノロジーオブジェクト	サーボサイクルクロック	補間サイクルクロック	補間サイクルクロック2
ドライブ軸	(x)	準拠規格	X
位置決め軸	(x)	準拠規格	X
従動軸	(x)	準拠規格	X
外部エンコーダ	(x)	準拠規格	X
出力カム(camType)	X	X	X
カムトラック(camTrackType)	X	X	X
測定入力	X	X	X

出力カムと測定入力用のシステムサイクルクロックは(V3.2の場合、軸についても)、SIMOTION SCOUTの[Configuration]ダイアログで設定します。

V4.1の場合、例外的なケースですが、サーボでIPOシェアを計算することもできます。この点については、『TO軸マニュアルの「モーションコントロール / インタポレータ」を参照してください。

5.3.4 タスクの実行時間

タスクランタイムを使用して、システムのコンピュータ性能がアプリケーションの要求に十分であるかどうかを確認することができます。

タスクランタイムは、TaskruntimeおよびeffectiveTaskruntimeデバイス変数に表示されます。

以下の区別がされます。

- 1つのレベル内の個々のタスクのランタイム(Taskruntime)

Taskruntimeには、タスクのネットランタイムの合計が表示されます(割り込み時間を含めない)。

- レベルのランタイム(effectiveTaskruntime)

effectiveTaskruntimeには、1つのレベルの有効なランタイムが表示されます(割り込み時間を含めて)。

これは、レベルの起動から、レベルの最後のタスクの終了までの時間です。

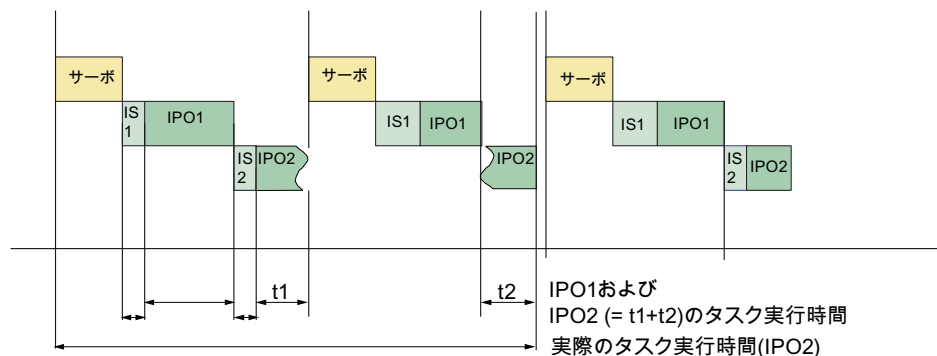


図 5-26 Taskruntimeと effectiveTaskruntime(サイクルタイム比率IPO1 : IPO2 = 2)

凡例:

[Servo]: サーボサイクルクロック

IS1: IPOSynchronousTask

IS2: IPOSynchronousTask_2

IPO1: IPOTask

IPO2: IPOTask2

次の表には、Taskruntime(タスク)とeffectiveTaskruntime(レベル)を測定することができるタスクとレベルを示します。

タスク(Taskruntime)	レベル(effectiveTaskruntime)
servodcc	サーボ / T1(DCC)
servo	
servosynchronous	
lpodcc	lpo / T2(DCC)
ipo	
iposynchronous	
lpodcc_2	lpo_2/T3(DCC)
ipo_2	
iposynchronous_2	
dccaux	DCCAux
dccaux_2	DCCAux_2
background	背景

注記

タスクランタイムを測定するには、システム変数taskRuntimeMonitoringを有効にする必要があります。

SCOUTデバイス診断では、タスクランタイムも表示されます。

下記も参照

タスクのランタイム測定用ファンクション - 概要 (ページ 267)

実行システムの最適化 (ページ 443)

効率的なプログラミング - 概要 (ページ 442)

5.3.5 タイムアウトとレベルオーバーフロー

計算プロセスを実行するとき、タイムアウト / レベルオーバーフローが発生する場合があります。

特定の実行レベルのオーバーフロー時のシステム応答に対して、さまざまな応答をコンフィグレーションすることができます。

タイムアウト

実行レベルBackgroundTask、TimerInterruptTask、SynchronousTasks、およびShutdownTaskの最大実行サイクルを設定することができます。

設定された最大継続時間を超えた場合、関連SystemInterruptTask(TimeFaultTaskまたはBackgroundFaultTask)、または標準ファンクション(たとえば、CPU to STOP)を呼び出すことができます。

レベルオーバーフロー

実行レベルIPOSynchronousTaskおよびIPOSynchronousTask_2に対して、レベルオーバーフローの許容値を設定することができます。

1つのレベル(IPOTask/IPOSynchronousTaskまたはIPOTask2/IPOSynchronousTask_2)が、設定されたシステムサイクルクロック、IPO、またはIPO_2以内に実行されないと、レベルオーバーフローが発生します。

許容値が設定された場合は、次のIPOサイクルクロックが以下の処理のために使用されます。

- 設定されたエラー応答をトリガしないで、直前のサイクルクロックの計算プロセスの完了
- 現在のサイクルクロックの計算プロセスの開始

次のレベルサイクルで別のレベルオーバーフローが発生すると(許容値が設定されていないか、許容値を超えた場合)、システム全体の閉塞を回避するために、CPUがSTOPに設定されます。この結果、診断バッファにエントリが作成され、エラー応答(CPU to STOP)が起動ロックアウト付きで実行されます。

実行システムのタスクコンフィグレーションでは、最大5のレベルオーバーフロー許容値を設定することができます。

例

以下は、レベルオーバーフローの実行関係の例です。

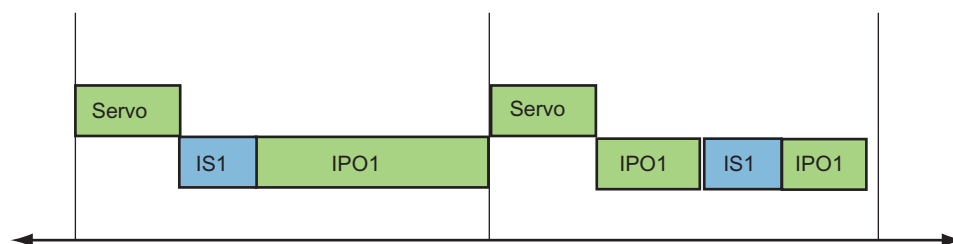


図 5-27 IPOオーバーフローの例

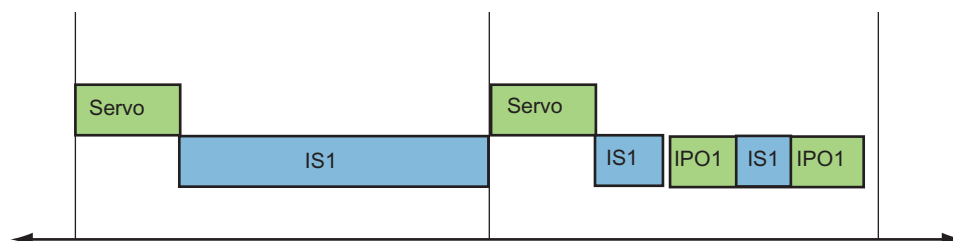


図 5-28 IPOSynchronousTaskオーバーフローの例

[Servo]: サーボサイクルクロック
 IS1: IPO Synchronous Task
 IPO1: IPO Task

高性能のプログラミング

レベルオーバーフローが発生しても、DP/サーボサイクルクロックを長くすることができない場合は、以下の措置を取ることができます。

- IPO Synchronous Taskのタスクコンフィグレーションで
 IPO Synchronous Task : IPO サイクルクロック = 75%
 2回のIPOオーバーフローの許容
- 最適PROFIBUSの使用
 ユーザー定義プロファイル: HSA=2(最大PROFIBUSアドレス)
 たとえば、PROFIBUS上の2つのノード、RetryLimit =
 1、バスパラメータの周期的分散のオフ(ただし、この場合、PGをアイソクロナスPROFI
 BUSに接続することはもうできません)
- BackgroundTask : MotionTasksの比率の変更
 たとえば、MotionTask側から見ると、BackgroundTaskでイベントが発生すると、すぐに、
 MotionTaskが起動され、数少ない割り込みで処理されます。
- システム変数を一度だけ読み取り、それを、将来の使用のためにローカル変数に一時的
 に格納する。

タイムアウトとレベルオーバーフローの監視

システム変数TaskruntimeおよびeffectiveTaskruntimeを使用して、レベルオーバーフローま
 たはタイムアウトが発生しているかどうかを知ることができます。

- IPO/IPO_2サイクルクロックでのTaskruntimeがIPO/IPO_2サイクルクロックでのeffectiv
 eTaskruntimeと同一である場合は、このタスクは高優先度のタスクによって割り込まれ
 ています。
- IPO/IPO_2サイクルクロックでのTaskruntimeがIPO/IPO_2サイクルクロックでのeffectiv
 eTaskruntimeと同一でない場合は、このタスクは高優先度のタスクによって割り込まれ
 ています。

推奨事項: サーボ

: IPO : IPO_2の比率 > 1の場合は、SynchronousTasksのタスクコンフィグレーションで
 、継続時間として、可能な限り大きい、IPOサイクルクロックのパーセンテージ値を
 入力します。

- オーバーフローしたレベルのeffectiveTaskruntimeが設定されたレベルサイクルクロック
 より小さい場合は、タイムアウトが発生しています。

これは、レベルの監視時間をシステム変数effectiveTaskruntimeの値に合わせることによ
 って、回避することができます。

- オーバーフローしたレベルのeffectiveTaskruntimeが設定されたレベルサイクルクロック
 に非常に近いが、大きい場合は、レベルオーバーフローが発生しています。

これは、システムサイクルクロックを調整することによって回避することができます。
 回避しないで、これらのオーバーフローを許容することもできます。

下記も参照

SynchronousTasks (ページ 151)

タスクの実行時間 (ページ 181)

5.3.6 タスクの起動に関する情報: TaskStartInfo(TSI)

タスクが起動される度に、たとえば以下の重要な情報がTaskStartInfoに格納されます。

- タスクの起動時刻
 - **TechnologicalFaultTask**の場合:
テクノロジーオブジェクトのトリガインスタンスとアラーム番号
 - **TimeFaultTask**の場合: タイムアウトエラーを生じさせたTimerInterruptTask
- タスク内で、このタスクの関連TaskStartInfoを検索することができます。

下記も参照

Taskstartinfoの使用 (ページ 97)

5.3.7 ウォッチドッグ

タスクの実行の最大継続時間(サイクルタイム)を設定することができます。
この時間を超えると、関連SystemInterruptTask(**TimeFaultTask**または**TimeFaultBackground Task**)を呼び出すか、応答[CPU to STOP]を呼び出すことができます。

サイクルタイムの監視は、以下のタスクで起動することができます。

- BackgroundTask
- TimerInterruptTasks
- SynchronousTasks
- ShutdownTask

サイクルウォッチドッグのコンフィグレーション

- タスクの[Onfiguration]ウィンドウで、[Task configuration]タブに切り替えます。

BackgroundTask、TimerInterruptTask、ShutdownTaskの場合:

- **[Time monitoring]**フィールドに最大実行時間を入力します(0 ms = 監視なし)。

IPOsynchronousTaskの場合:

- 最大実行時間とIPOサイクルクロックとの間の比率を選択します。

ShutdownTask以外のタスクの場合:

- タイムアウトへのエラー応答として、関連SystemInterruptTaskが、[CPU to STOP]応答を選択します。

注記

時間トリガタスクの場合:

タスクが実行終了の前に再起動される場合、タイムアウトエラーが存在します。

SystemInterruptTaskが起動されるか、CPUがSTOPモードに移行します。

下記も参照

SystemInterruptTasks (ページ 157)

5.4 ラウンドロビン実行レベルでの時間割り当て

高優先度のユーザーおよびシステムタスクが処理された後に残っている時間は、**MotionTasks**と**BackgroundTask**によって使用されます。

この時間は、ラウンドロビン原理に従って、**BackgroundTask**と**MotionTasks**に配分されます。

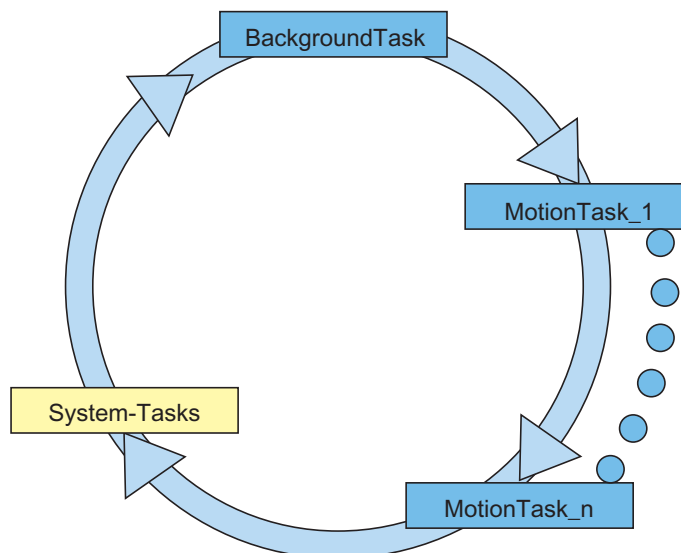


図 5-29 ラウンドロビン実行レベルでの時間割り当ての概要

ラウンドロビン原理

高優先度のタスクによって消費されない計算時間は、残りのタスク(**BackgroundTask**、**MotionTasks**、および**SystemTasks**)にラウンドロビン原理で配分されます。これは、これらの残りのタスクが順々に実行されることを意味します。

この結果、これらのタスクは、並行して処理されるように見えます。

直前のタスクが計算時間を放棄すると(タスクの完了か、タスクの"wait"状態)、常に、次のタスクが起動されます。

さらに、ラウンドロビンタスクの計算時間は、最大サーボサイクルクロック数までに制限されます。次のラウンドロビンサイクルでは、計算時間は割り込み点から続行されます。

システムは、ラウンドロビンタスクが、常に、少なくとも1サーボサイクルクロックの計算時間を得ることを保証します。

時間の割り当てに関して、複数の連続したサーボサイクルクロックの計算時間を提供できるため、**BackgroundTasks**は有利な立場にあります(スライダ実行システム)。

その他のタスク

ユーザープログラムタスク(**BackgroundTask**、**MotionTask**)以外に、ラウンドロビンサイクルでの計算時間を必要とするシステムタスク(たとえば、通信用のシステムタスク)もラウンドロビン実行レベルで実行されます。

シーケンス

通常、タスクは、ラウンドロビンサイクルで、順々に実行されます。
(この順序は完全に固定的なものではなく、たとえば、ダウンロード、電源オンなどによって変更することができます)。

Time allocation

1つのタスクは、次のタスクに計算時間を渡す前に、指定されたサーボサイクルクロック数(残りのランタイム)の間、継続して実行することができます。

ただし、"何もすることがない"場合は、その時間が経過する前に、次のタスクに計算時間を渡すこともできます。

ラウンドロビン実行レベルのすべてのタスクが一度処理された場合は、最初のタスクが再起動されます。これにより、アイドル時間はなくなります。

ユーザープログラムが実行されない場合が存在します。これは、BackgroundTaskとMotionTaskの処理時間が非常に小さいような場合です(1サーボサイクルクロックの残りのランタイムより小さい)。この場合、システムタスクが残りのランタイムを使用します。

再起動 / プロセスイメージ

BackgroundTaskは、終了した後、プロセスイメージが更新されるまで再起動されません。その更新は、**ServoTask**で実行されます。

パフォーマンス

MotionTaskで、waitコマンドや同期モーションコマンドでない継続ループが存在すると、MotionTaskはその最大計算時間を消費してしまいます。

この場合、BackgroundTaskのサイクルも使用されることになります(計算時間の大部分がMotionTaskで消費されます)。

ラウンドロビンレベルのシステムタスクの起動間隔が非常に長くなり、たとえば、通信に影響を与えます。

BackgroundTaskとMotionTasksのラウンドロビン実行レベルの割り当て時には、完全な1ラウンドロビンサイクルのBackgroundTaskが少なくとも1つのタイムスライスを受け取り、MotionTaskは2つのタイムスライスを受け取ります(たとえば、継続ループのために必要な場合)。

周期的MotionTasks

モーションタスクを周期的実行する場合、以下の点に注意してください。

MotionTasksで継続ループを使用したい場合は、たとえば、各サイクルで、**_waitTime(0s)**を発行します。この場合、このMotionTaskは次のMotionTaskに制御を渡します。

注記

MotionTasksに **_waitTime**

(0s)でない継続ループが存在すると、MotionTaskは2つの完全なサーボサイクルクロックを必要とするため、ラウンドロビン実行レベルの負荷が大きくなります。

MotionTaskを周期的に実行する場合は、継続ループや、プログラムの起動時にstepコマンドを使用するのではなく、通常の方法でタスクを終了し、異なるタスク(たとえば、BackgroundTask)からMotionTaskを再起動することをお勧めします。
これは、RUN時のダウンロードの場合にも効果的です。

下記も参照

実行システムの最適化 (ページ 443)

5.4.1 時間割り当ての設定

MotionTasksまたはBackgroundTaskの[Time allocation]ボタンによってアクセス可能なスライダを使用して、ラウンドロビン実行レベルの残りのタスクに対するBackgroundTaskの時間的重みを設定することができます。

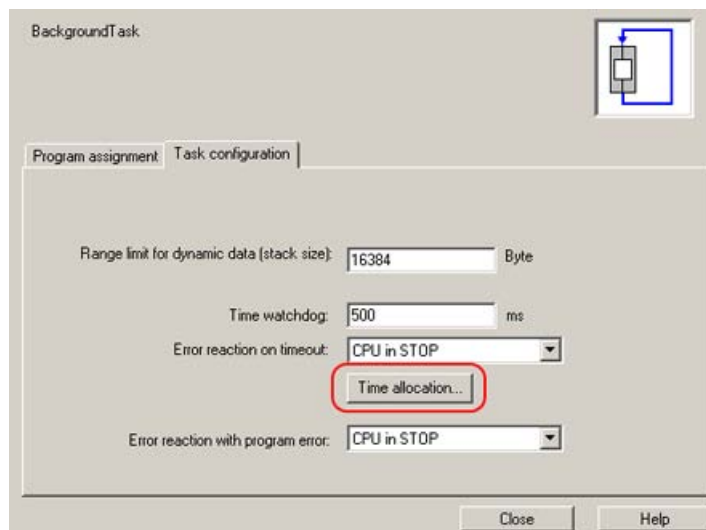


図 5-30 SCOUTでのタスクコンフィグレーション

ラウンドロビン実行レベルでの時間割り当ての指定

1. MotionTaskまたはBackgroundTaskのコンフィグレーションウィンドウで、[Task configuration]タブを選択します。
2. [Time allocation]をクリックします。
[Time Allocation in the Round Robin Execution Level]ウィンドウが開きます。
3. スライダを使用して、計算時間の比率を設定します。
4. [OK]をクリックして作業内容を確認します。

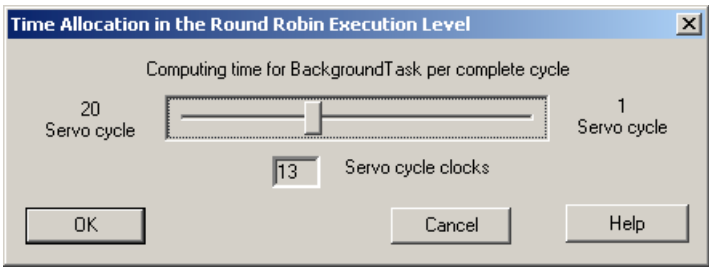


図 5-31 ラウンドロビン実行レベルでの時間割り当て

ラウンドロビン実行レベルでの時間割り当て

ここでは、BackgroundTaskと他のすべてのラウンドロビンタスク(MotionTasksとSystemTasks)の時間割り当ての比率を指定します(「実行レベル / タスク (ページ 128)」を参照)。

BackgroundTaskは、その他の1つのラウンドロビンタスクの最大n倍の計算時間を取得します(他のすべてのラウンドロビンタスク(たとえば、MotionTask_1)は、それぞれが同一の最大計算時間シェアを持ちます)。この係数は、スライダで設定することができます。

BackgroundTaskが最後まで実行されると、早くとも、入力が更新されたときに再起動されます(次のサーボサイクルの後)。

BackgroundTaskの計算時間を長く設定すると、それだけ、他のラウンドロビンタスクに不利になり、たとえば、SCOUT/DPへの通信速度が遅くなります。

このような設定は、非常にタイムクリティカルなアプリケーションだけに限定する必要があります。

表 5-5 MotionTasksとBackgroundTaskに対する計算時間割り当ての効果

計算時間の増加	効果
MotionTasks	BackgroundTaskの開始から終了まで時間が長くなるようになり、極端な場合には、時間監視が応答します。
BackgroundTask	いくつかの状況の下で、MotionTaskのプログラムの実行に長い時間がかかるようになります。
BackgroundTaskの計算時間には、非周期の通信(PROFIBUSまたはIRTによるPROFINET IO経由)に必要な時間も含まれます。	

5.4.2 設定(例)

ラウンドロビン実行レベルでの時間割り当てを説明するために、以下に、スライダを設定する2つの例を提供します。

Case 1: スライダを右端に配置する

BackgroundTaskは、少なくとも1サーボサイクルクロックの間、実行されます。

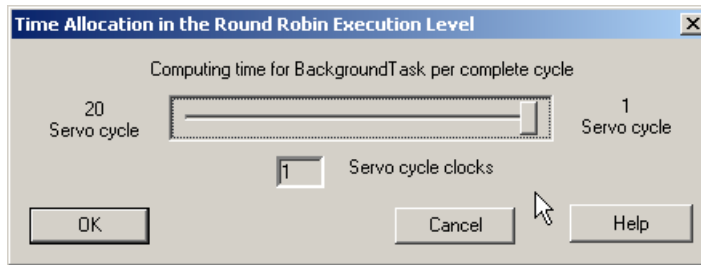


図 5-32 ラウンドロビンでの時間割り当て: MotionTasksの計算時間の設定

この設定では、BackgroundTaskが、1サーボサイクルクロックの間、実行された後、すべてのMotionTasksが実行され(各MotionTaskは、前もって一時停止モードを入力しない場合、最大2サーボサイクルクロックの間、実行されます)、その後に再度、BackgroundTaskが、1サーボサイクルクロックの間、実行されます…。

表 5-6 例:

サーボサイクルクロック1	サーボサイクルクロック2~3	サーボサイクルクロック4~5	サーボサイクルクロック6	サーボサイクルクロック7~8
Background Task	MotionTask 1	MotionTask 2	Background Task	MotionTask 1

サンプルプログラム:

MotionTask 1とMotionTask 2がBackgroundTaskと並行して実行されます、サーボサイクルクロック = 3 ms

- Whileループのカウンタは、BackgroundTaskで加算されます(緑の線)。
- Whileループのカウンタは、MotionTask 1で加算されます(赤の線)。
- Whileループのカウンタは、MotionTask 2で加算されます(青の線)。

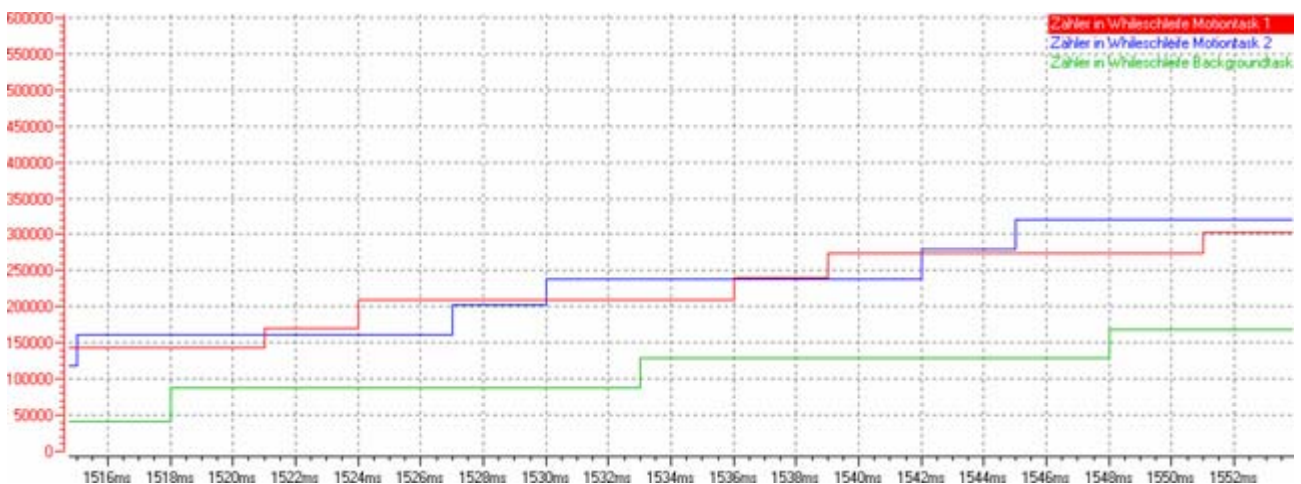


図 5-33 ラウンドロビンでの時間割り当ての実行例 MotionTasksの計算時間の設定

t = 1518 ms	BackgroundTaskが、1サーボサイクルクロックの間、実行されます。
t = 1521 ms、1524 ms	MotionTask 1が、2サーボサイクルクロックの間、実行されます。
t = 1527 ms、1530 ms	MotionTask 2が、2サーボサイクルクロックの間、実行されます。

t = 1533 ms	BackgroundTaskが、1サーボサイクルクロックの間、実行されます。

Case 2: スライダを左端に配置する

BackgroundTaskは、20サーボサイクルクロックの間、実行されます。

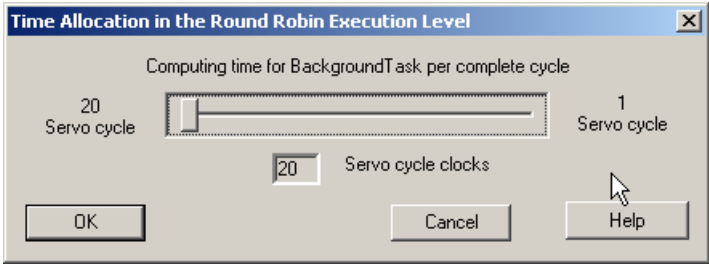


図 5-34 ラウンドロビンでの時間割り当て: BackgroundTaskの計算時間の設定

この設定では、BackgroundTaskが、20サーボサイクルクロックの間、実行された後、すべてのMotionTasksが実行され(各MotionTaskは、前もって一時停止モードを入力しない場合、最大2サーボサイクルクロックの間、実行されます)、その後再度、BackgroundTaskが、20サーボサイクルクロックの間、実行されます...

表 5-7 例:

サーボサイクルクロック1～20	サーボサイクルクロック21～22	サーボサイクルクロック23～24	サーボサイクルクロック25～40	サーボサイクルクロック41～42
Background Task	MotionTask 1	MotionTask 2	Background Task	MotionTask 1

サンプルプログラム:

MotionTask 1とMotionTask 2がBackgroundTaskと並行して実行されます、サーボサイクルクロック = 3 ms

- Whileループのカウンタは、BackgroundTaskで加算されます(緑の線)。
- Whileループのカウンタは、MotionTask 1で加算されます(赤の線)。
- Whileループのカウンタは、MotionTask 2で加算されます(青の線)。

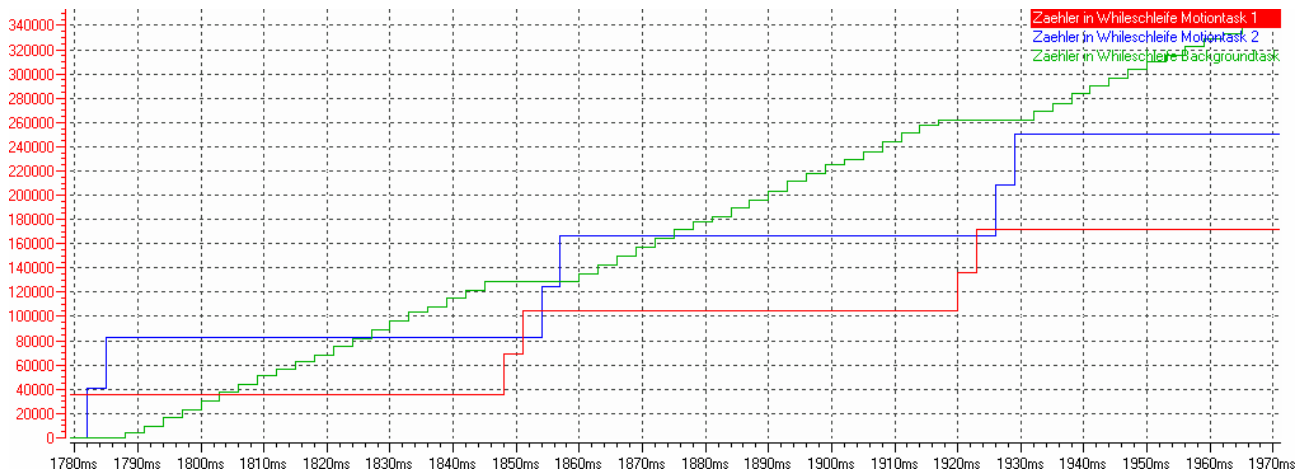


図 5-35 ラウンドロビンでの時間割り当ての実行例 BackgroundTaskの計算時間の設定

t = 1788 ~ 1845 ms	BackgroundTaskは、20サーボサイクルクロックの間、実行されます。
t = 1848 ms、1851 ms	MotionTask 1が、2サーボサイクルクロックの間、実行されます。
t = 1854 ms、1857 ms	MotionTask 2が、2サーボサイクルクロックの間、実行されます。
t = 1860 ~ 1917 ms	BackgroundTaskは、20サーボサイクルクロックの間、実行されます。

5.4.3 タスクの処理(例)

以下の例は、各種タスクの時間軸に沿った実行を示します。

以下の例はPROFIBUSの場合ですが、IRTによるPROFINET IOの場合にも同様に適用できます。

例1: 有効なInterruptTaskがないときの時間軸に沿った実行

デフォルト: DP:[Servo]: IPO = 1 : 1 : 2

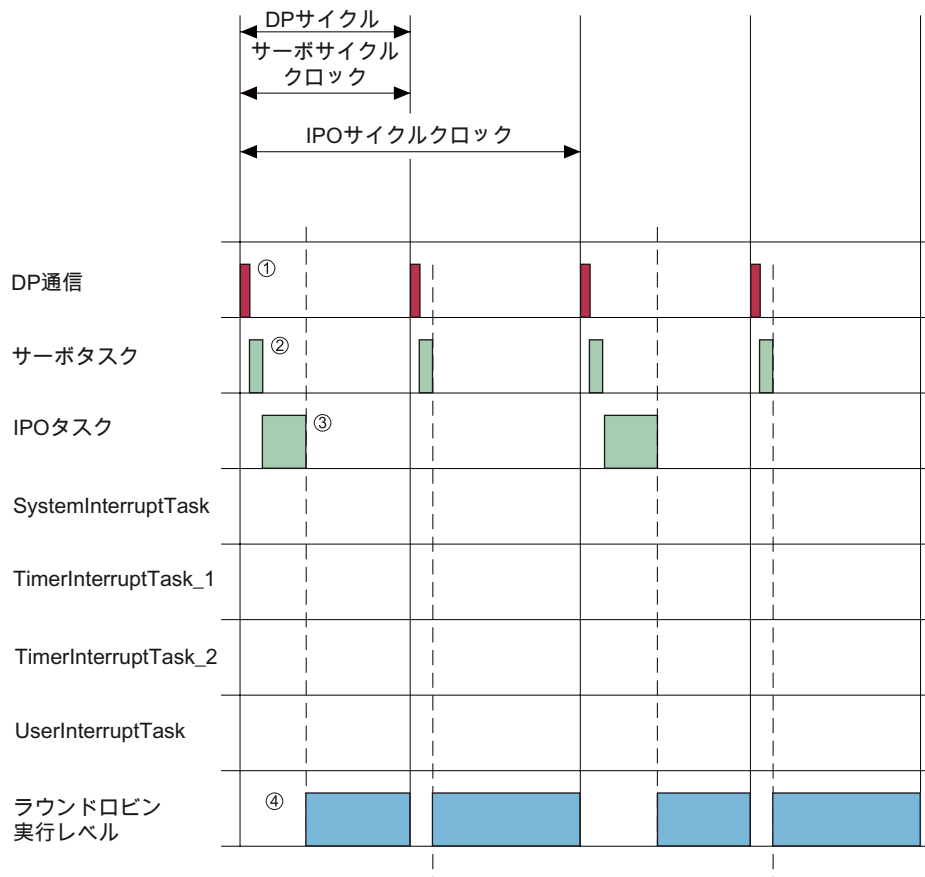


図 5-36 時間軸に沿ったタスクの実行、例: 有効なInterruptTaskなし

- 1 この例の実行システムでは、サイクルクロックが
DP サイクル:サーボサイクルクロック:IPOサイクルクロック: 1:1:2
として選択されています。
サーボタスクはDPサイクルごとに実行され、IPOタスクは2番目のDPサイクルごとに実行されます。
- 2 DP通信が一番高い優先度を持ち、サーボタスクが次の優先度を持ちます。
- 3 IPOタスクは、サーボタスクの後に実行されます。
- 4 ラウンドロビン実行レベルは、残りの時間で実行されます。

例2:
InterruptTaskが有効で、IPOタスクが1サーボサイクルクロックより長い間実行される場合の時間軸に沿った実行

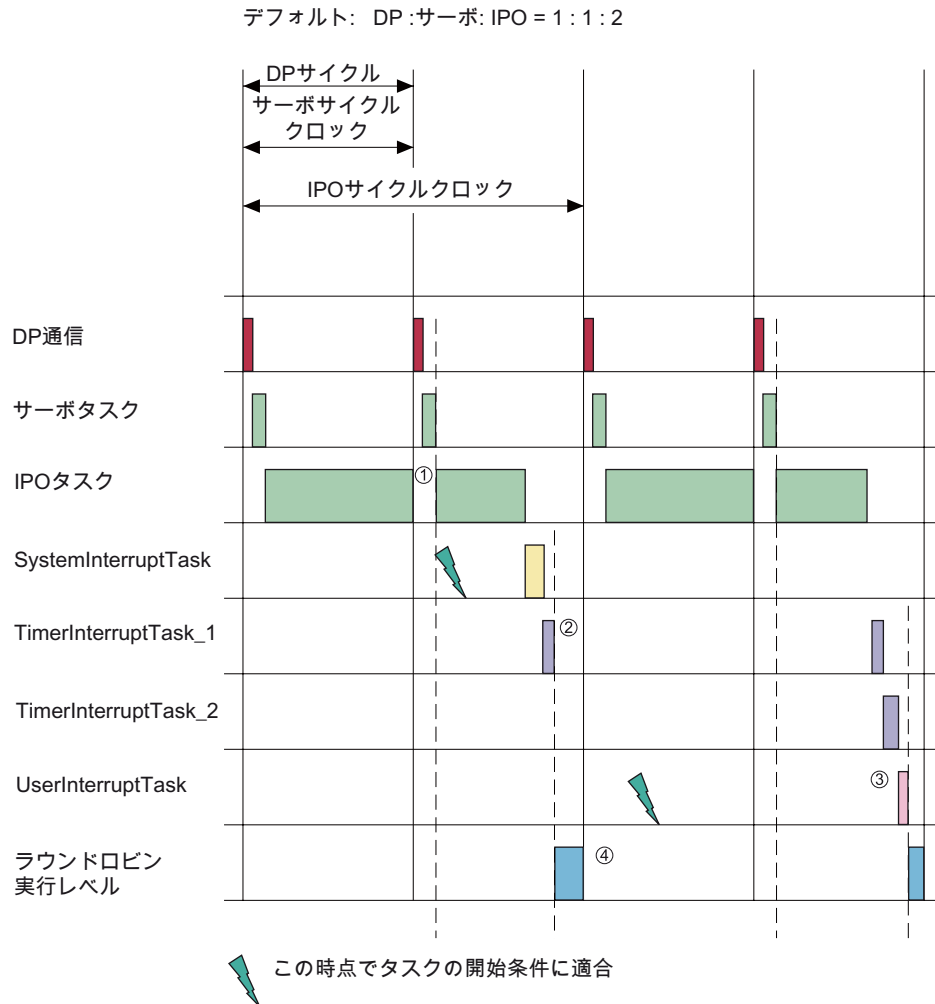


図 5-37 時間軸に沿ったタスクの実行、例:
InterruptTaskが有効で、IPOタスクが1サーボサイクルクロックより長い間実行

- 1 IPOタスクで実行されるプログラムは、サーボサイクルクロックより長い間続きます。この結果、IPOタスクは割り込まれ、DP通信とサーボタスクが実行されます。この後、IPOタスクの実行が再開されます。
- 2 IPOタスクが完了した後、SystemInterruptTaskが実行されます。この後、低優先度のTimerInterruptTaskが起動されます。
- 3 UserInterruptTaskは、実行条件が以前から満たされている場合でも、高優先度のタスクが完了するまでは実行されません。
- 4 ラウンドロビン実行レベルは、残りの時間で実行されます。

例3: マルチタスクにおけるタスク切り替え用の特殊機能

MotionTasksとBackgroundTaskの両方で条件を検索するときは、マルチタスク動作を考慮する必要があります。

例:

表 5-8 例:

	BackgroundTask	MotionTask1	MotionTask2
ソース:	x=5	...	...
	...	if x <> 5	if x = 5
	x=4	_enableAxis	_stopAxis
	...	Endif	Endif
		...	...
	BackgroundTask	MotionTask1	MotionTask2
実行シーケンス:	x=5		
	タスク切り替え →	if x <> 5 条件が満たされていない endif	
		タスク切り替え →	if x = 5 条件が満たされた タスク切り替え →
	x=4		
	タスク切り替え →	...	
		タスク切り替え →	...
			_stopAxis

MotionTask2での処理の実行は明らかであると思われますが、処理の実行がタスク切り替えのために意図どおりでない場合が生じます。

BackgroundTaskで値を変更した後(x=4)、if検索が処理済みで、タスク切り替えの前の検索結果が依然として妥当です。これにより、この時点で条件x=5が満たされない場合でも:

- _enableAxisが、MotionTask1で実行されません。
- _stopAxisが、MotionTask2で実行されます。

この処理の実行は、タスク切り替えが実行される時点に依存しています。

5.5 フィールドバスシステム上のアイソクロナスI/O処理

フィールドバスシステム経由でI/Oが接続された制御システムのアイソクロナスI/O処理の一般的なシーケンスを以下に示します。

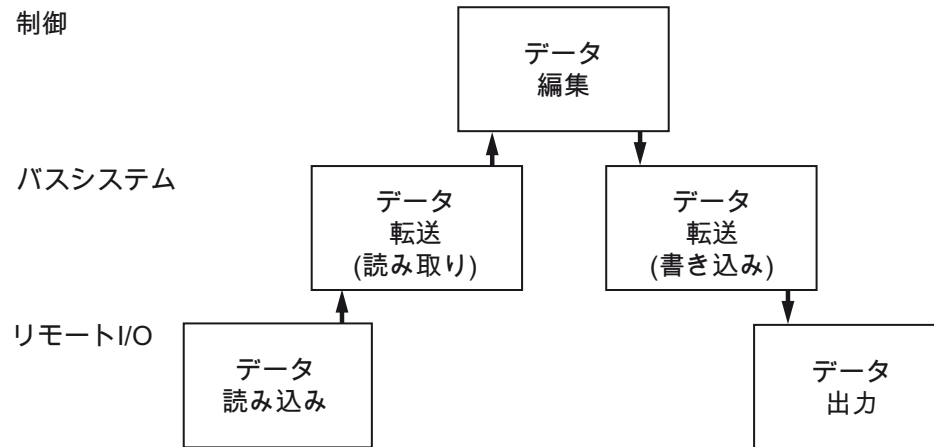


図 5-38 フィールドバスシステム上のアイソクロナスI/O処理のデータフロー

アイソクロナスシステムでは、個々のアクションは時間軸に沿って相互に同期化されます。この結果、短く再現可能な(すなわち、同一の長さの)応答時間が達成されます。

サイクルクロックと時間基準は、アイソクロナス(パルス発振)バスシステムによって事前設定されます。

5.5.1 PROFIBUS DP上のデータプロトコル

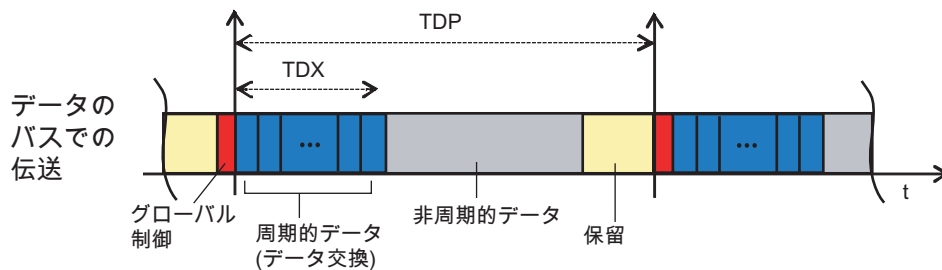


図 5-39 PROFIBUS DP上のデータプロトコル

このバス上のデータプロトコルには、以下が含まれます。

- バスサイクルクロックを定義するアイソクロナスグローバル制御メッセージフレーム(GC)
- 周期的データ: サイクルクロックごとに、ノード間で伝送されるデータ

周期的データ通信は、非常に短い再現可能なプロセス応答時間を可能にします。サイクルごとに、情報の転送が行われます。

ユーザプログラムは、このデータに、プロセスイメージ経由、または、I/Oへの直接アクセスでアクセスします。

- **非周期的データ:** 必要な場合だけ、大量に伝送されるデータ

非周期的伝送は、特に、タイムクリティカルでない大量データの伝送に適しており、データは複数のタスクに分散することができます。

非周期的データの例: アラーム、診断サービス、およびデータセット

ユーザープログラムは、この非周期的データに、たとえば、コマンド `_readrecord`、`_write record` を使用してアクセスします。

- **予備:** 次のグローバル制御までの残りの時間

この残りの時間は現在動作している非周期的通信に応じて異なり、最大 T_{DP-TDX} です。

5.5.2 PROFINET IO上のプロトコル

説明

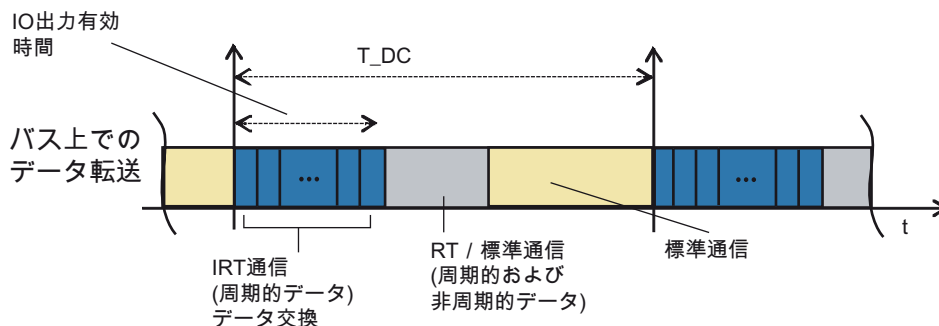


図 5-40 PROFINET IO上のプロトコル

このバス上のデータプロトコルには、以下が含まれます。

- 計画された時間間隔で周期的データを伝送するIRT通信
- 以下のデータを伝送できるRTおよび標準通信:
 - たとえば、IOデータを一緒に伝送できる周期的RTCテレグラム
 - たとえば、アラームデータを一緒に伝送できる非周期的RTAテレグラム
 - NRTデータ(標準通信)、すべての標準Ethernetベースのプロトコルを一緒に伝送できる標準Ethernetテレグラム
- 標準通信(同期テレグラムより前の最後の時間間隔):

ここには、同期テレグラムが使用されるまでに完結する通信だけが可能です。

IO出力妥当時間には、新しい出力データが使用可能な持続時間が記載されます。

5.5.3 アイソクロナスデータ処理

アイソクロナスアプリケーション

アイソクロナスアプリケーションでは、コントローラ、ドライブ、(接続されている場合)分散I/Oの処理サイクルクロックが、バスサイクルと同期化されます。

PROFIBUS DP上のアイソクロナスデータ伝送

PROFIBUSでは、バスサイクルクロックは、グローバル制御テレグラム(GC)と同期化されます。
GCは、バスサイクルクロックを実現し、同時に、システム全体のシステムサイクルクロックです。

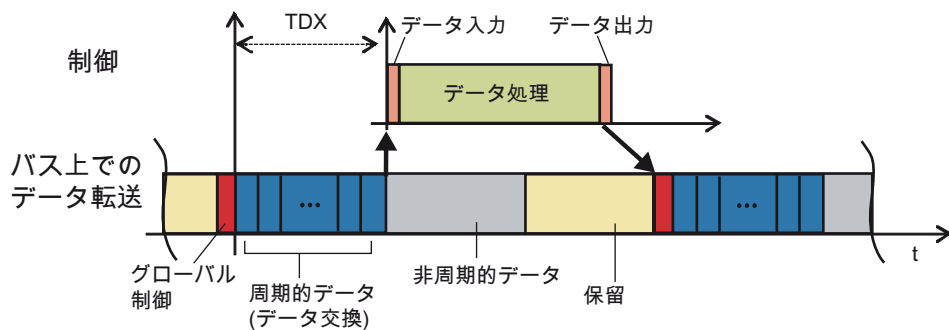


図 5-41 PROFIBUS DP用バスサイクルの構造

PROFINET IO上のアイソクロナスデータ転送

PROFINET IOでは、コントローラの処理サイクルクロックは、IRT通信と同期化されます。

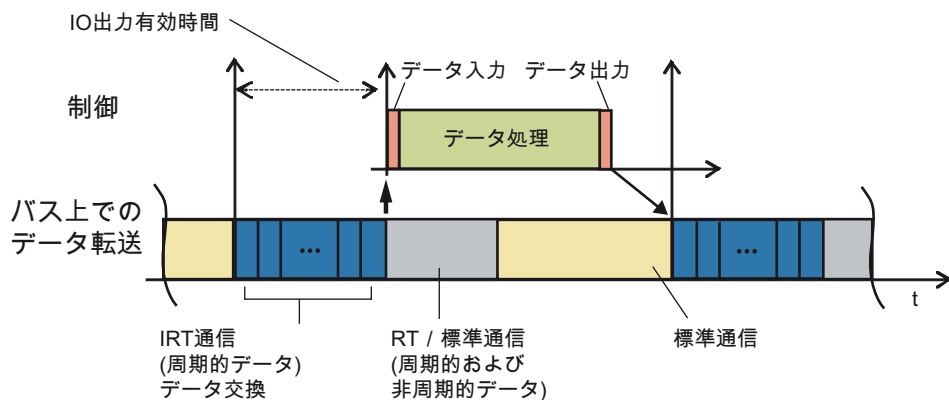


図 5-42 PROFINET IO用バスサイクルの構造

この結果、周期データ通信で伝送されるデータをアイソクロナス処理することができます。

PROFIBUS DPによるアイソクロナスデータ取得

完全なアイソクロナスシステムでは、データ取得、データ処理、およびデータ出力が統合システムサイクルクロックを参照して実行されます。

アプリケーションは、周期的データ伝送が完了した(T_{DX})ときに起動されます。

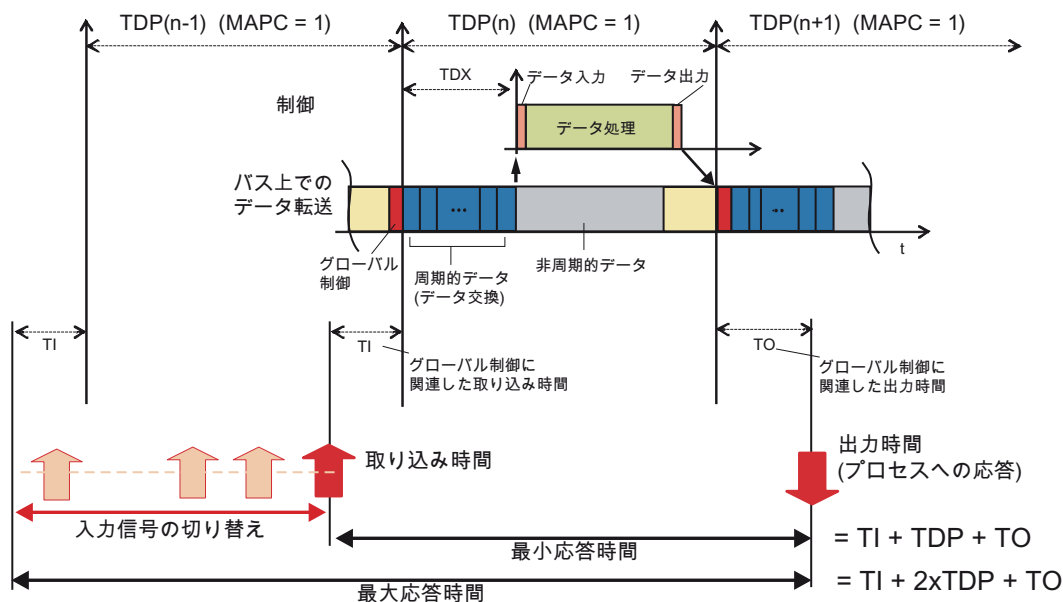


図 5-43 応答時間の計算(PROFIBUS DP)

PROFINET IOによるアイソクロナスデータ伝送

アプリケーションは、周期的データ伝送が完了した(IO出力妥当時間)ときに起動されます。

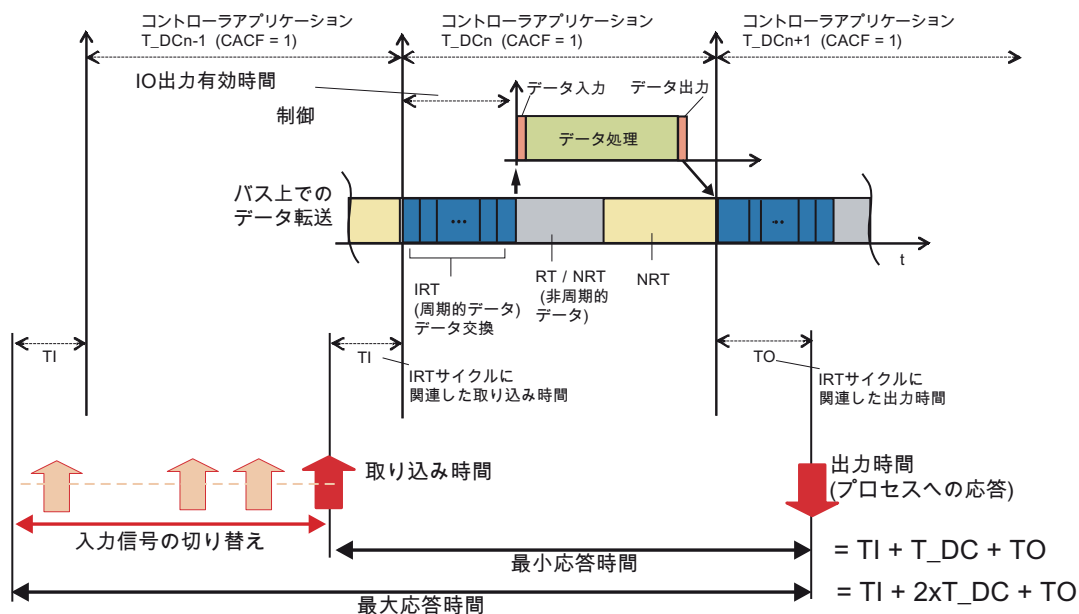


図 5-44 応答時間の計算(PROFINET IO)

応答時間の計算(PROFIBUS DPおよびPROFINET IO)

分散I/Oの読み取りプロセスは、転送が必要で、かつ、新しいバスサイクルの開始時に一貫した状態の入力を読み取るために、時間 T_I によるオフセットが行う必要があります。

時間 T_I は、少なくとも、電子モジュール上の信号処理と変換時間から構成され、さらに、モジュラーET 200 I/Oシステムの場合は、ET 200バックプレーンバス上の伝送時間も含まれます。通常、 T_I はすべてのモジュールについて同一の値が選択され、最も遅いモジュールの時間に合わせられます。

時間 T_O は、ユーザープログラム(データ処理)のプロセス応答が、バス上の分散I/Oの"複数の端子"に同時にかつ正しく到達することを保証します(複数のドライブについて、同時にドライブ制御の設定値が使用可能になる場合と同様です)。

時間 T_O は、少なくとも、バス上のすべてのノードの周期的データ交換用の時間(モジュラーET 200 I/Oシステムの場合、ET 200バックプレーンバス上の出力の伝送時間)、電子モジュール上の信号処理、および変換時間から構成されます。

応答時間またはむだ時間(ドライブの場合)

個々のサイクルを同期化することによって、サイクルクロック" $n-1$ "で入力データを読み取り、サイクルクロック" n "でデータの伝送と処理を行い、サイクルクロック" $n+1$ "の最初に、計算された出力データの伝送を行い、同一サイクルクロックで"複数の端子"への出力データの結合を行うことができます。

PROFIBUS

上記により、実際のプロセス応答時間は以下ようになります。

- $T_I + T_{DP} + T_O$ (最小)
- $T_I + 2 \cdot T_{DP} + T_O$ (最大)

PROFINET

上記により、実際のプロセス応答時間は以下ようになります。

- $T_I + T_{DC} + T_O$ (最小)
- $T_I + 2 \cdot T_{DC} + T_O$ (最大)

5.5.4 コントロールのデータ処理の動的応答

ServoSynchronousTaskのデータ処理

- ServoSynchronousTaskのデータ処理で、バスサイクルクロック : サーボ = 1 : 1の場合、コントローラに1バスサイクルクロックの処理時間を生成することができます。

これは、たとえば、入力信号に応じた出力データ(たとえば、DOまたはAO)の設定に使用されます。

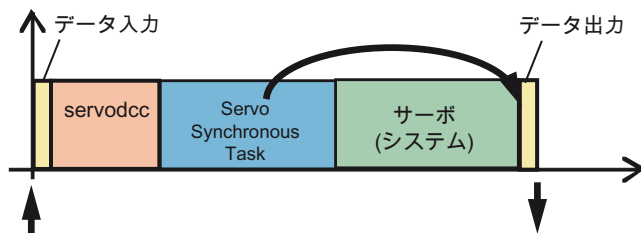


図 5-45 コントロールのデータ処理 - 出力データの設定

- 軸データ / 軸モーションの更新時、処理時間は、データがサーボですでに有効であるかどうかに応じて異なります(スーパーインポーズされた設定値または出力値など)。

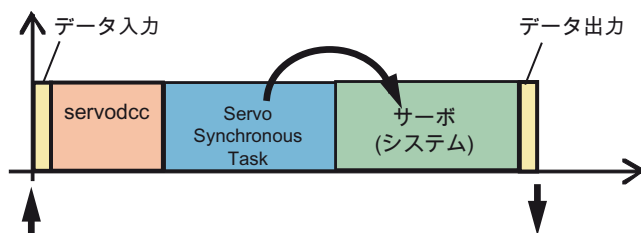


図 5-46 コントロールのデータ処理 - モーションデータの更新

- データまたはコマンドがIPOが起動される(たとえば、モーションコマンドの発行)まで有効でない場合は、出力の出力データは次のサーボでのみ有効になります。

注:

ServoSynchronousTaskのランタイムを少なくするには(その結果、データ出力までの時間を減らすには)、IPOSynchronousTaskに適切なタスクのプログラミングを実装することをお勧めします。

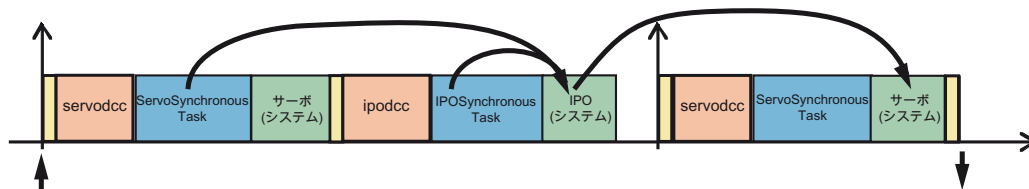


図 5-47 コントロールのデータ処理 - 出力データはIPOで有効

IPOSynchronousTaskのデータ処理

- IPOSynchronousTaskのデータ処理では、I/Oの出力データは次のサーボで有効になります。

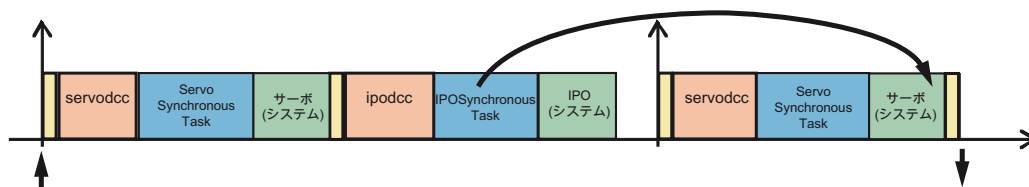


図 5-48 コントロールのデータ処理 - 出力データは次のサーボで有効

- または 次のIPOまたはサーボのモーションデータ(上記を参照)

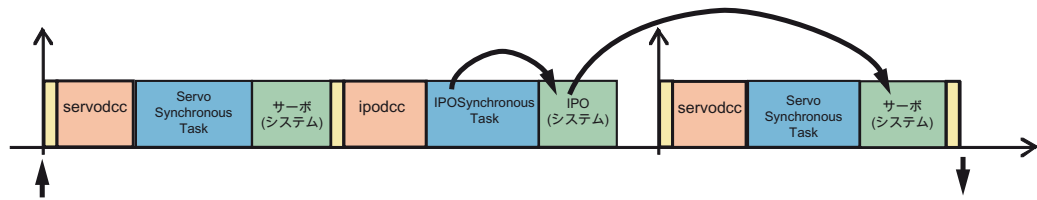


図 5-49 コントロールのデータ処理 - 出力データはIPOで有効 - モーションデータの更新

I/Oの出力データが次のサーボでのみ伝送される場合は、この条件が発生する都度、応答時間は1バスサイクルクロックだけ増えます(バスサイクルクロック：サーボ：IPO = 1 : 1 : 1の場合)。

シナリオ	推奨ユーザープログラムタスク
I/Oの高速端子 - 端子応答	ServosynchronousTaskの使用
設定値の高速更新(サーボレベル)	ServosynchronousTaskの使用
モーションの切り替え / スーパーインポーズ (たとえば、_stop、_move、_pos、...)、 たとえば、マーキングの印刷	IPOSynchronousTaskの使用

V4.1の場合、例外的なケースですが、サーボでIPOシェアを計算することもできます。この点については、『TO軸マニュアルの「モーションコントロール / インタポレータ」を参照してください。

5.5.5 データ伝送(取得から処理まで)に関する動的応答

PROFIBUSまたはPROFINETバスシステムを使用するときは、システムサイクルクロックの計算で、指定されたデータ伝送時間(T_{DX})を考慮に入れる必要があります。 T_{DX} の時間が小さい場合は、サイクルクロックを短くするか、同じ長さのサイクルクロックでアプリケーション用の処理時間を増やすことができます。

- PROFIBUS DPでは、 T_{DX} はHWコンフィグレーションに表示されます。
- 統合されたPROFIBUS(SINAMICSプロジェクトの拡張の程度に依存します)のSIMOTION D(SIMOTION D4xxおよびSIMOTION CX32のI/O拡張)では、 $125 \mu s \leq T_{DX} \leq 375 \mu s$ の範囲の伝送時間が自動的に設定されます。
- PROFINET IOでは、 $T_{DX} = 0.5 \cdot T_{DP}$ が自動的に設定されます。

5.5.6 データ取得およびデータ出力の動的応答

時間 T_O をコンフィグレーションするときは、 T_{DX} を考慮に入れてください。規則: $T_{DX} \leq T_O$ が常に適用されます。

- これらの時間は、HWコンフィグレーションで設定できます。
- 取得および出力でTM15/TM17 High Featureを使用するときは、以下の時間を設定します。
 - 取得の場合: $T_i + 1$ DRIVE-CLiQサイクル(通常、 $125 \mu s$)
 - 出力の場合: $T_o + 1$ DRIVE-CLiQサイクル(通常、 $125 \mu s$)

下記も参照

PROFIBUSのET 200

I/Oデバイス用HWコンフィグレーションによるTdp、Ti、およびToの識別 (ページ 204)

5.5.7 PROFIBUSのET 200

I/Oデバイス用HWコンフィグレーションによるTdp、Ti、およびToの識別

HWコンフィグレーションのアイソクロナスモード画面フォーム(メニュー[Edit]Isochronous mode]の呼び出し)で、T_{DP}、T_i、T_oを識別することができます。

このダイアログボックスでは、関連オブジェクト(PROFIBUS、スレーブ、および各スレーブのモジュール)のアイソクロナスモード用に設定されたパラメータおよび時間の一覧が提供されます。

このダイアログのすべての時間は、ms単位で指定されています。

- T_i/T_o設定
 - ネットワークでは、時間T_iおよびT_oは、"すべてのスレーブに対して同一の値"が集中的に設定されます。
 - スレーブでは、時間T_iおよびT_oは、スレーブごとに個々に設定されます。
- T_i、T_o、T_{DP}

選択したDPマスタシステムの現在の設定値または計算値T_i、T_o、およびT_{DP}が表示されます。

画面フォームおよびすべてのパラメータの詳細は、HWコンフィグレーションのオンラインヘルプを参照してください。

The screenshot shows the 'Isochrone Mode' dialog box with three main sections: PROFIBUS, Slave, and Module. The PROFIBUS table has columns for PROFIBUS, Master system, Constant Bus..., Ti/To Setting, Ti, To, TDP, TDPMin, TDPMax, and Tm. The Slave table has columns for Address, Designation, Synchro..., Ti, To, TiMin, TiMax, ToMin, ToMax, ToMin, TDPMin, and TDPMax. The Module table has columns for Slot, Designation, Synchro..., TCI, TWQ, ToiMin, TDPMin, and TDPMax. Red boxes highlight the TDP column in the PROFIBUS table and the Ti and To columns in the Slave table.

PROFIBUS	Master system	Constant Bus...	Ti/To Setting	Ti	To	TDP	TDPMin	TDPMax	Tm
PROFIBUS Integr..	DP master system (1)	yes	in the slave			1.000	0.000	17470...	
PROFIBUS(1)	DP master system (1)	yes	in the slave			1.000	0.625	32.000	

Address	Designation	Synchro...	Ti	To	TiMin	TiMax	ToMin	ToMax	ToMin	TDPMin	TDPMax
1	IM 153-2, Redundant	yes	0.156	0.313	0.156	0.203	0.375	0.375	0.063	0.370	32.000
3	IM 153-2, Redundant	yes	0.219	0.375	0.219	0.563	0.375	0.719	0.063	0.418	32.000

Slot	Designation	Synchro...	TCI	TWQ	ToiMin	TDPMin	TDPMax
4	DI16xDC24V	yes	0.050				
5	DI16xDC24V	yes	0.050				
6	DO16xDC24V/0.5A	yes		0.050			
7	DO16xDC24V/0.5A	yes		0.050			

図 5-50 HWコンフィグレーションのアイソクロナスモード画面フォーム

注記

アイソクロナスモードの完全な"端子 - 端子"サポートは、シーケンス内のすべてのコンポーネントが"アイソクロナスモード"システムプロパティをサポートする場合だけ可能です。HWコンフィグレーションのPM10カタログまたはハードウェアカタログから選択を行うときは、モジュールの情報フィールドのエントリ[**Isochronous mode**]に注意してください。

現在使用可能なアイソクロナスI/Oモジュールの一覧は、インターネット<http://support.automation.siemens.com/WW/view/de/14747353>を参照してください。

5.5.8 サイクルクロックスケーリングの処理

サイクルクロックスケーリングでは、以下のことに注意してください。

- IPO同期タスクの終了時、プロセスイメージは次の可能なサーボで出力されます(データ出力)(= 応答時間の最適化)。
サーボサイクルクロックをIPOサイクルクロックに合わせると、I/OアクセスがIPOSynchronousTask経由で実行される場合、1つまたは複数のサーボサイクルクロックのデータが、遅かれ、早かれ、1 IPOサイクルクロック以内に出力されます。
- サーボ実行レベルの終了時、ServoSynchronousTaskのプロセスイメージは、次の可能なバスサイクルクロックで出力されます(= 応答時間の最適化)。
 - PROFINETで、PROFINETサイクルクロックをサーボサイクルクロックに縮小すると、I/OアクセスがServoSynchronousTask経由で実行され、サーボ実行レベルのランタイムが1バスサイクルクロックを超えて変動する場合、1つまたは複数のバスサイクルクロックのデータが、遅かれ、早かれ、1サーボサイクルクロック内で出力されます。
 - PROFIBUSの場合、サーボ実行レベルは、常に、最初のバスサイクルクロックで終わる必要があるため、データは、常に、最初のバスサイクルクロックで出力されます。

このため、個々のサイクルクロックのサーボ実行レベルのランタイムが異なることによって、端子 - 端子時間が変わる場合があります。

応答時間を最適化する代わりに、常に一定の応答時間を実現するには、以下を設定する必要があります。

- PROFIBUSの場合:
 - 縮小率 サーボ : IPO = 1 : 1にします。IPOSynchronousTaskからのI/Oアクセスを常にアイソクロナスモードで実現するためです。
 - コメント:
PROFIBUSでは、ServoSynchronousTaskからのI/Oアクセスは常にアイソクロナスモードです。
- PROFINETの場合:
 - 縮小率 バスサイクルクロック : サーボ : IPO = 1 : 1 : 1にします。IPOSynchronousTaskからのI/Oアクセスを常にアイソクロナスモードで実現するためです。

- 縮小率 バスサイクルクロック : サーボ = 1 : 1にします。ServoSynchronousTaskからのI/Oアクセスを常にアイソクロナスモードで実現するためです。

5.5.9 ランタイムを最適化するためのHW ConfigでのPROFIBUS DPのコンフィグレーション

概要

HWコンフィグレーションでの最適化設定で、SIMOTIONファームウェアのランタイムを大幅に改善することができます。

ステーショントポロジに関するランタイム最適化設定

アイソクロナスPROFIBUS

DPでは、TO軸用のアイソクロナスドライブとアイソクロナスDPスレーブに、連続したステーションアドレスを設定する必要があります(たとえば、11、12、13、14、...)。
他のスレーブに当該の番号が割り当てられていない場合は、このアドレスシーケンスに抜けがあってもかまいません(たとえば、11、13、16など)。

iスレーブのランタイム最適化設定

インテリジェントDPスレーブコンフィグレーションのスロット定義で、システムとしての標準規約がないスロットに、連続したスロット番号を割り当てます。

ET200スレーブのランタイム最適化設定

PROFIBUS

DPトポロジを選択することによって、システムサイクルクロックやプロセス応答時間を改善することができます。

- 高速の均一な応答時間を維持するには、アイソクロナス / 非アイソクロナスI/Oを複数の独立したDPマスタシステムに分散する必要があります。
- アイソクロナスモードは、このプロパティを必要とするステーションおよび周辺モジュールの場合だけ有効にします。
- 基本的な負荷時間を減らし、アイソクロナスDPサイクルタイムを最小化するために、スレーブステーションの数を減らします。
同様の理由から、アイソクロナスDPマスタシステムでのOPおよびテキスト表示の使用を避けます。
これらのHMIは、別の独立した非アイソクロナスDPマスタシステムで使用するか、Ethernetインターフェース経由で操作します。
- I/O入力でのフィルタ時間を短くして、プロセス応答時間を減らします(このために、ET 200 S High Featureデジタル入力で最小フィルタ設定を選択します)。
- 多くの周辺モジュールは、サイクルタイムを可能な限り最短にする特別のモードを持っています(たとえば、ET 200Mアナログ入力やET 200S SSIモジュール用の高速モードなど)。

- ステーションあたりI/O数も、 T_i および T_o 時間を決めるファクタです。個々のコンポーネントの中の最大 T_i および T_o 時間が、システム全体の T_i および T_o 時間を決定します。システムアーキテクチャ内での適切なI/O分散によって、 T_i および T_o 時間を最小化します。

以下のアーキテクチャは、動的応答の点で、お勧めできません。

- 少数のインターフェースモジュール(ET 200M、ET 200S)と多数のI/Oサブモジュール
 - 多数のインターフェースモジュール(ET 200M、ET 200S)と少数のI/Oサブモジュール
- 動的応答の点でお勧めするアーキテクチャ:
- DPサブネット上の少数のスレーブ / インターフェースモジュール
 - 少数のI/Oサブモジュールを含むインターフェースモジュール
 - 入力モジュールだけと連携動作するインターフェースモジュール
 - 出力モジュールだけと連携動作するインターフェースモジュール
 - ET 200M: 最長処理時間の出力モジュールは、ET 200Mの左端のスロットから順々に挿入します。
 - ET 200M: 最長処理時間の入力モジュールは、ET 200Mの右端のスロットから順々に挿入します。
 - 高いPROFIBUS転送速度は、DPサイクルタイムを減らします。

HWコンフィグレーションのダイアログボックスでは、コンフィグレーションの結果であるDPサイクルタイムと T_i および T_o 時間が表示されるため、これに従って、DPトポロジの実用的なコンフィグレーションを作成することができます。

下記も参照

タイムアウトとレベルオーバーフロー (ページ 182)

5.6 SIMOTION実行システムへのDCCの統合

5.6.1 概要

概要

ドライブコントロールチャート(DCC)を使用して、ドライブ関連の調整および制御タスクを実装することができます。

このために、コンフィグレーションツール(DCCエディタ)を使用してグラフィカルに(いわゆる、チャート方式で)相互接続が可能なブロックセット(DCB)が用意されています。

DCBは、ライブラリ(DCBLIB)として使用できます。

このため、作成されたチャートは、SINAMICSドライブだけでなく、SIMOTIONプラットフォーム上で実行できます。

その他の参考文献

DCBライブラリ、DCCエディタの説明

5.6.2 DCCブロック(DCB)のシーケンスモデル

説明

個々のDCCブロック(DCB - ドライブコントロールブロック)は、DCCエディタで、ランタイムグループとして編成されます。

ランタイムグループは、DCCエディタで、DCCタスク(タスクT1～T5)に自由に割り当てることができます。

DCCエディタのDCCタスクは、SIMOTION実行システムのDCCタスクに対応します。

DCCエディタのタスク	実行レベル	タスク
T1	サーボ	servoDcc
D2	IPO	ipoDcc
T3	IPO_2	lpoDcc_2
T4	DccAux	dccAux
T5	DccAux_2	dccAux_2

実行グループは、ブロックのバイナリ出力で、有効 / 無効にすることができます。

詳細は、DCCエディタを参照してください。

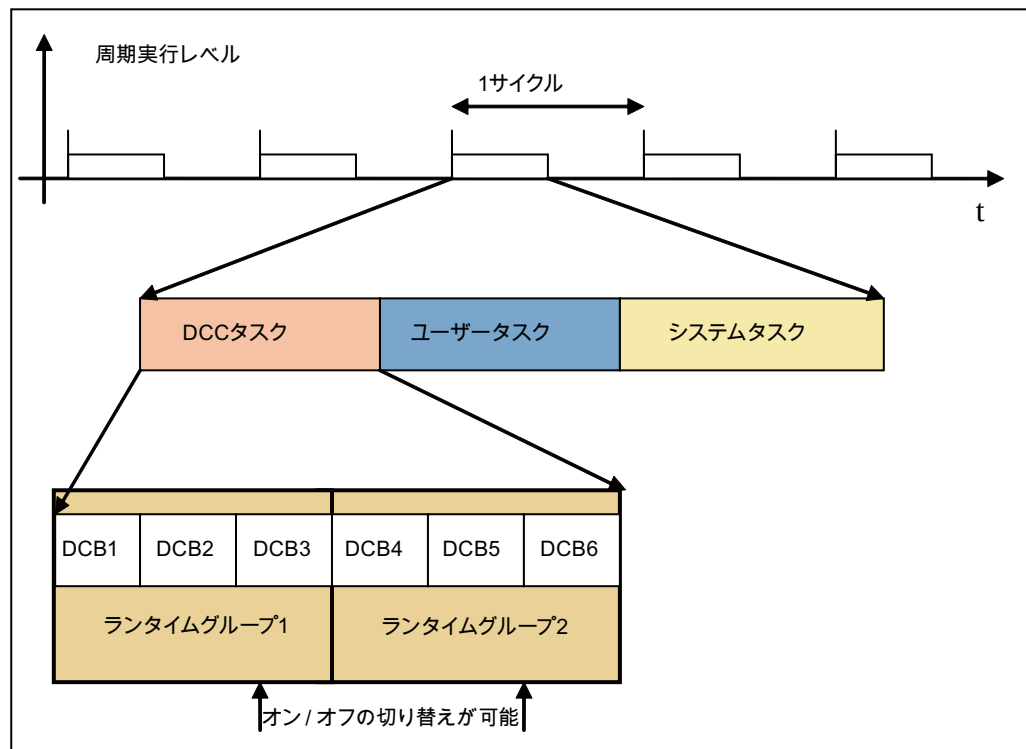


図 5-51 DCBランタイムグループのシーケンスモデル

このユーザーモデルでは、DCCタスクは、ユーザープログラムタスク(ST、MCC、LAD/FBDのユーザープログラムの実行環境)およびシステムタスクの隣に割り当てられています。必要の場合は、エンジニアリングシステム(ES)で、DCCタスクを作成します。これは、DCCタスクがシステムの標準として提供されていないことを示します。

DCCダイアグラムの実行

DCCダイアグラムの実行は、タスクの状態ではなく、タスクに割り当てられたランタイムグループの状態にリンクされます。ランタイムグループは、モード変更に応じて、有効化 / 無効化されます。

モード	アクション
STARTUP->RUN	ランタイムグループをオン
SHUTDOWN->STOPU	ランタイムグループをオフ
SHUTDOWN->STOP	ランタイムグループをオフ

モードの他のすべての変更は、ランタイムグループの変更を生じません。

下記も参照

サーボレベルのservoDccTask (ページ 210)

IPOレベルのipoDccタスク (ページ 211)

IPO2レベルのipoDcc_2タスク (ページ 212)

DccAuxおよびDccAux_2の実行レベル (ページ 213)

5.6.3 サーボレベルのservoDccTask

説明

T1(DCC)タスクは、SIMOTION実行システムのサーボレベルで実行されます。サーボレベルはアイソクロナスI/Oとのデータ交換と同期して呼び出され、主として、高速の制御タスク用に使用されます。

- 開始時に、テクノロジーオブジェクトおよびサーボ同期I/O変数用の周期的I/Oデータが、インターフェースから、実行システムのアイソクロナスI/Oにコピーされます。
- 終了時に、周期的データが再び実行システムからコピーされ、たとえば、PROFIBUS経由でPROFIBUSノードに転送されます。

次の図は、サーボレベルのタスクの時間軸に沿ったシーケンスを示します。

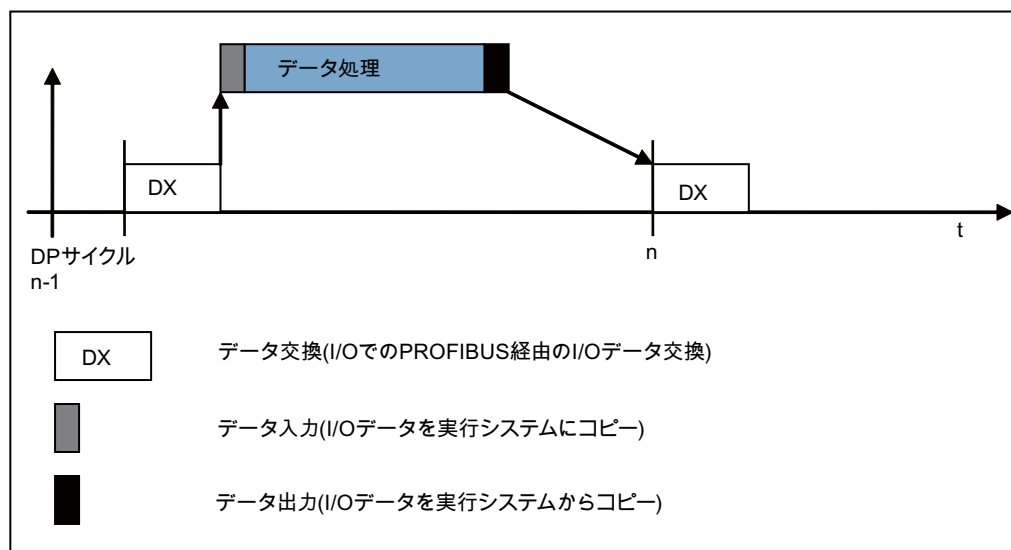


図 5-52 サーボレベルのT1(DCC)タスク

T1(DCC)タスクは、サーボレベルの開始時に、自動的に起動され、実行されます。

次の図は、サーボレベルのタスクのシーケンスを示します。

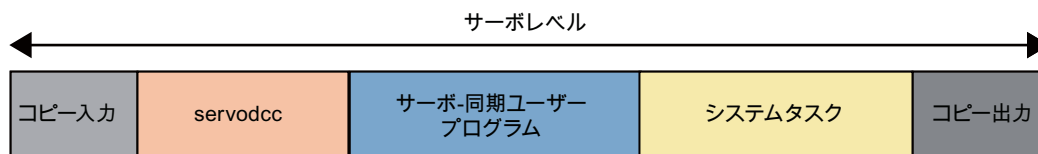


図 5-53 T1(DCC)タスクを含むサーボレベル

サーボレベルのレベルオーバーフロー

サーボレベルのタスク群は同一の優先度で実行されるため、相互に割り込みしません。サーボレベルのすべてのタスクが単一のサイクルで計算できない場合、レベルオーバーフローが発生し、システムはSTOPモードに移行して、スタートアップロックが設定されます。新しいスタートアップ(電源オフ / オン)またはダウンロードの後だけに、システムはRUNモードに戻ることができます。

下記も参照

システムサイクルクロックの定義 (ページ 175)

5.6.4 IPOレベルのipoDccタスク

説明

IPOレベルは、サーボレベルと同一のサイクルタイムまたは整数縮小率で実行されます。サーボレベルのタスクはより高い優先度を持つため、IPOレベルのタスクに割り込むことができます。さらに、IPOレベルでは、すべてのI/Oはサーボレベルにコピーされずに、実行システムにコピーされます。

データは、アイソクロナスI/Oと非アイソクロナスI/Oの両方から取得することができます。

コマンド処理と、テクノロジーオブジェクトの設定値生成は、IPOレベルのシステムタスクで実行されます。

次の図は、IPOレベルの時間軸に沿ったシーケンスを示します。

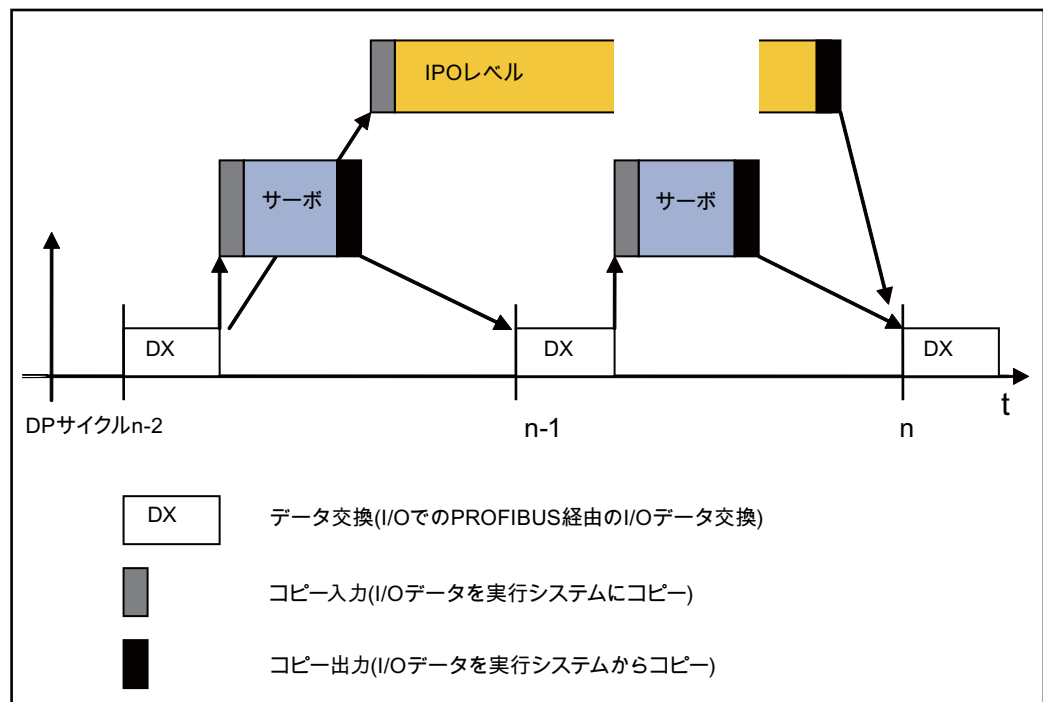


図 5-54 T2(DCC)タスクを含むIPOレベル

T2(DCC)タスクは、IPOレベルの開始時に、自動的に起動され、実行されます。

次の図は、IPOレベルの実行シーケンスを示します。

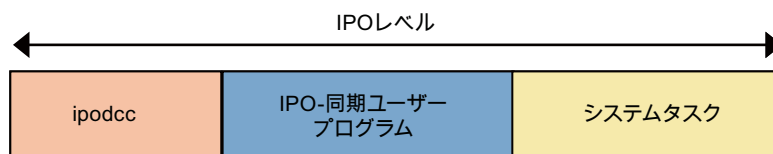


図 5-55 IPOレベルの実行シーケンス

IPOレベルの実行オーバーフロー

IPOレベルのタスク群は同一の優先度で実行されるため、相互に割り込みしません。より高い優先度を持つサーボレベルのタスクは、任意の時点で、IPOレベルの処理に割り込むことができます。これにより、IPOレベルのすべてのタスクの処理を終わらせることができず、レベルオーバーフローが発生することがあります。

この場合、IPOレベルのタスクは、次のサイクルで、再起動され、終わりまで計算されます。このレベルの負荷を軽減し、オーバーフローを回避しやすくするために、DCCタスクは新しいサイクルで再計算されません。

標準動作では、レベルオーバーフローの場合、デバイスはSTOPモードに移行し、スタートアップロックが設定されます。

新しいスタートアップ(電源オフ / オン)またはダウンロードの後にだけ、システムはRUNモードに戻ることができます。

ただし、最大5回の連続したオーバフローを許容するように、システムを設定することもできます。

この許容動作の設定は、タスクコンフィグレーションで正しく行う必要があります。

下記も参照

システムサイクルクロックの定義 (ページ 175)

タイムアウトとレベルオーバーフロー (ページ 182)

SynchronousTasks (ページ 151)

アイソクロナスデータ処理 (ページ 199)

5.6.5 IPO2レベルのipoDcc_2タスク

説明

IPO2レベルのサイクルクロックは、IPOレベルのサイクルクロックの倍数です。ただし、IPO2サイクルクロックとIPOサイクルクロックを等しく設定することはできません。

IPO2レベルのサイクルタイムは、IPOレベルのサイクルタイムより短い時間です。このため、IPOレベルとサーボレベルの両方が、IPO2レベルの処理に割り込むことができます。

T3(DCC)タスクは、IPO2レベルで、次のように割り当てられます。

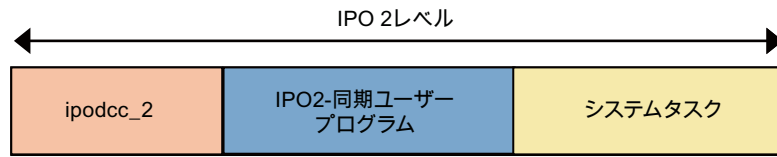


図 5-56 IPO2レベルの処理の順序

IPO2レベルのレベルオーバーフロー

レベルオーバーフローの場合のIPO2レベルの動作は、IPOレベルの動作と同じです。

下記も参照

IPOレベルのipoDccタスク (ページ 211)

システムサイクルクロックの定義 (ページ 175)

5.6.6 DccAuxおよびDccAux_2の実行レベル

説明

2つのDCC実行レベル(DccAuxおよびDCCAux_2)は、システムと同期して呼び出されます。

DCCAuxおよびDCCAux_2レベルは、以下のプロパティを持ちます。

- DccAuxサイクルクロックは、IPOサイクルクロックの倍数です。
- DccAux_2サイクルクロックは、DccAuxサイクルクロックの倍数です。
- DccAuxは、DccAux_2より高い優先度を持ちます。
- 同期トレース処理が可能です。
- 最大5回のオーバーフローが許容されます(デフォルト値は1です)。
現在のレベルオーバーフロー回数を取得することができます。
- 1レベルオーバーフローの場合、DCCタスクは、次のタスクで、最後まで計算されます。

表 5-9 レベルオーバーフロー回数を取得するためのシステム変数

システム変数	データタイプ	重要度
_device.numberOfSummarizedTaskOverflow.DccAux	UDINT	システムスタートアップ後のDccAuxレベルのオーバーフロー回数
_device.numberOfSummarizedTaskOverflow.DccAux_2	UDINT	システムスタートアップ後のDccAux_2レベルのオーバーフロー回数

下記も参照

システムサイクルクロックの定義 (ページ 175)

5.6.7 ブロック間のデータ交換

5.6.7.1 ブロック間のデータ交換(概要)

概要

DCCエディタでは、各種のブロックを相互接続することができます。
ブロック間のデータ交換では、以下の3つのケースを区別する必要があります。

- 同一の実行レベルに存在するブロック間の相互接続
- 1つのブロックの出力と、実行レベルが高い別のブロックの入力との相互接続
- 1つのブロックの出力と、実行レベルが低い別のブロックの入力との相互接続

下記も参照

同じレベルのブロック間のデータ交換 (ページ 214)

低優先度レベルからのブロック用データ (ページ 216)

高優先度レベルからのブロック用データ (ページ 219)

5.6.7.2 同じレベルのブロック間のデータ交換

説明

同じ実行レベルのブロックを相互接続した場合、データは、ブロックの実行シーケンスに従って、伝送されます。実行シーケンスの入力の値が現在の値であることが保証されます。

実行シーケンスをデータフローに従って定義した場合は、むだ時間はレベルで発生しません。

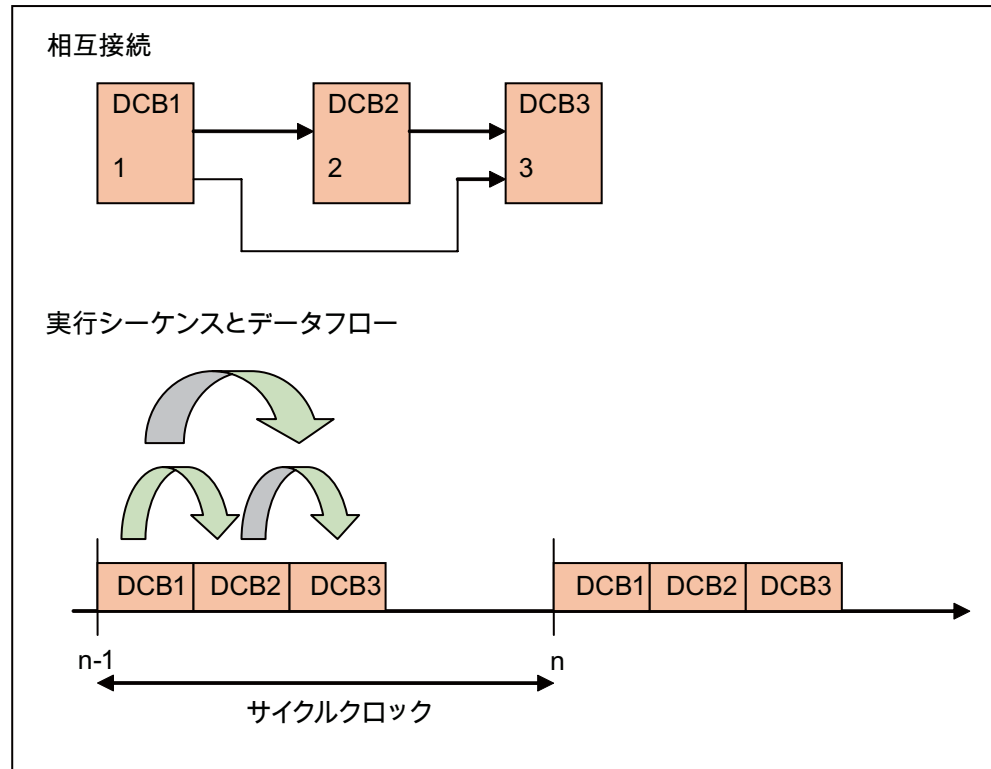


図 5-57 相互接続シーケンスと実行シーケンスが同じ

レベルのブロックの実行シーケンスがデータフローと一致しない場合は、入力で、直前のサイクルクロックの値に対するアクセスが行われるため、むだ時間が発生します。

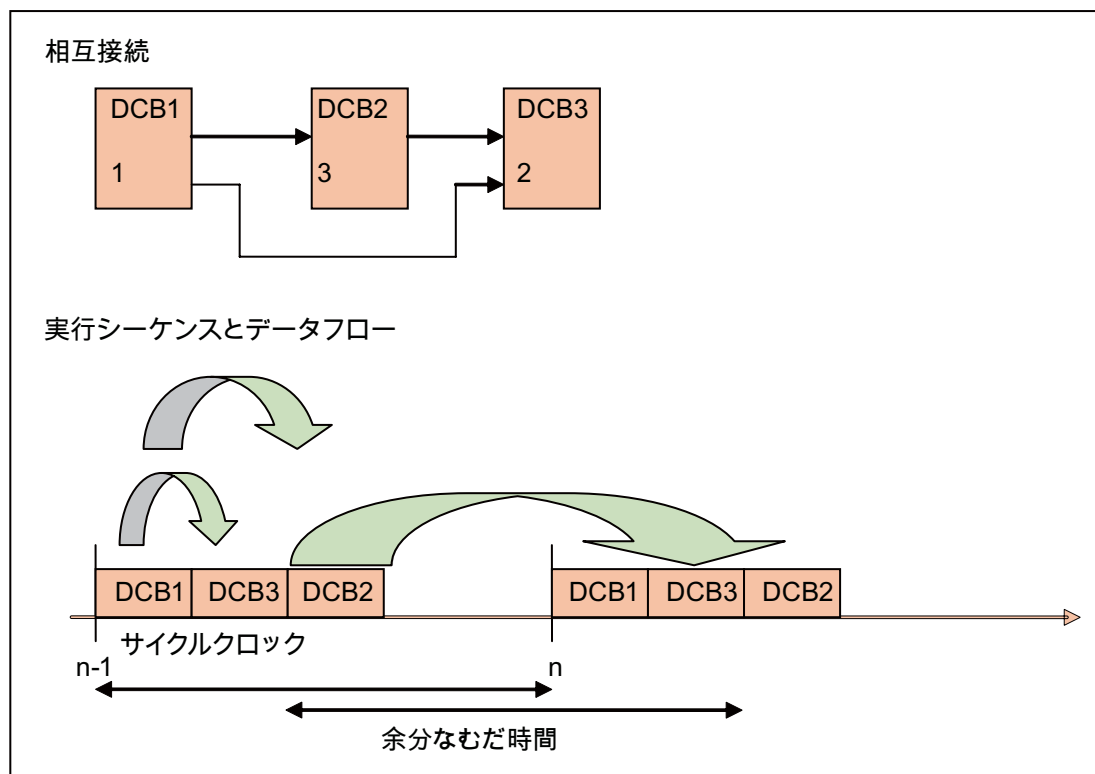


図 5-58 相互接続シーケンスと実行シーケンスが異なる

むだ時間を最適化するには、常に、データフローに合わせて実行シーケンスを設定します。

5.6.7.3 低優先度レベルからのブロック用データ

説明

1つの実行レベルのすべてのブロックの計算を終了して、初めて、それらのブロックの出力値を、異なる実行レベルのブロックの入力で使うことができます。高優先度レベルからの出力値は、入力値の等時値性を確保するために、専用のサイクルクロック(受け取りサイクルクロック)でアクセスされます。次の図は、高優先度レベルでの等時値伝送を示します。

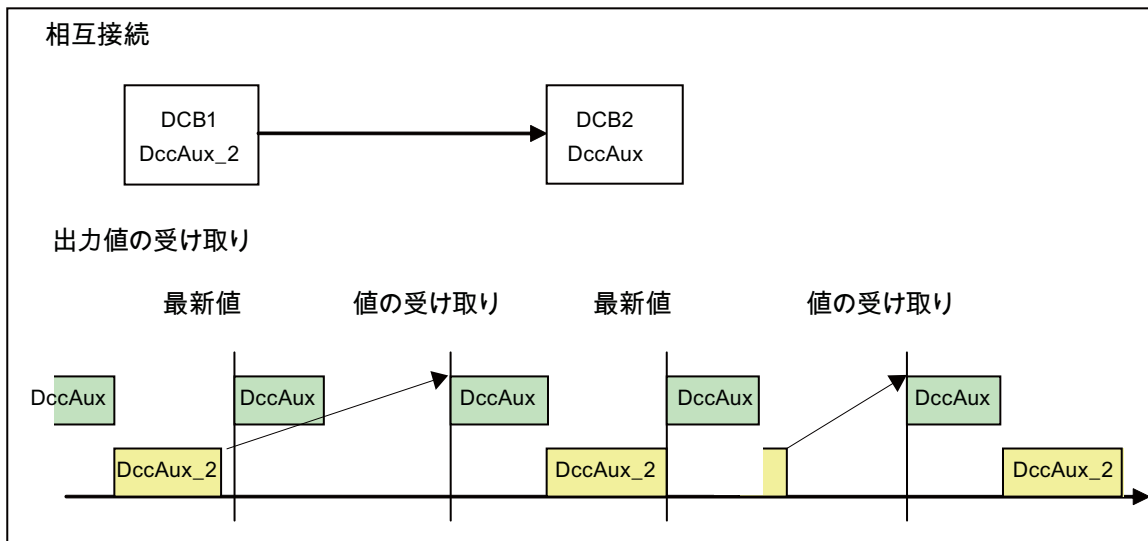


図 5-59 低優先度レベルからのデータ交換の例

レベル間のサイクルクロック比率は、1：2です。高優先度レベルのタスクは、低優先度タスクが直前のサイクルクロックで最後まで計算されたかどうかに関わりなく、常に2番目のサイクルクロックで値を受け取ります。

低優先度レベルのレベルオーバーフロー

低優先度レベルのレベルオーバーフローが発生すると、受け取りサイクルクロックで、現在値がまだ計算されていません。この場合、次の受け取りサイクルクロックまでは、入力の古い値が計算のために使用されます。

次の図は、サイクルクロック比率1：2の場合のオーバーフロー動作を示します。T2で、オーバーフローが発生しています。このため、受け取り時刻(サイクルクロック1)には、古い値が高優先度のタスクに伝送されます。新しい値は次の受け取り時刻に受け取られることになり、結局、2サイクルクロックだけ後になります。

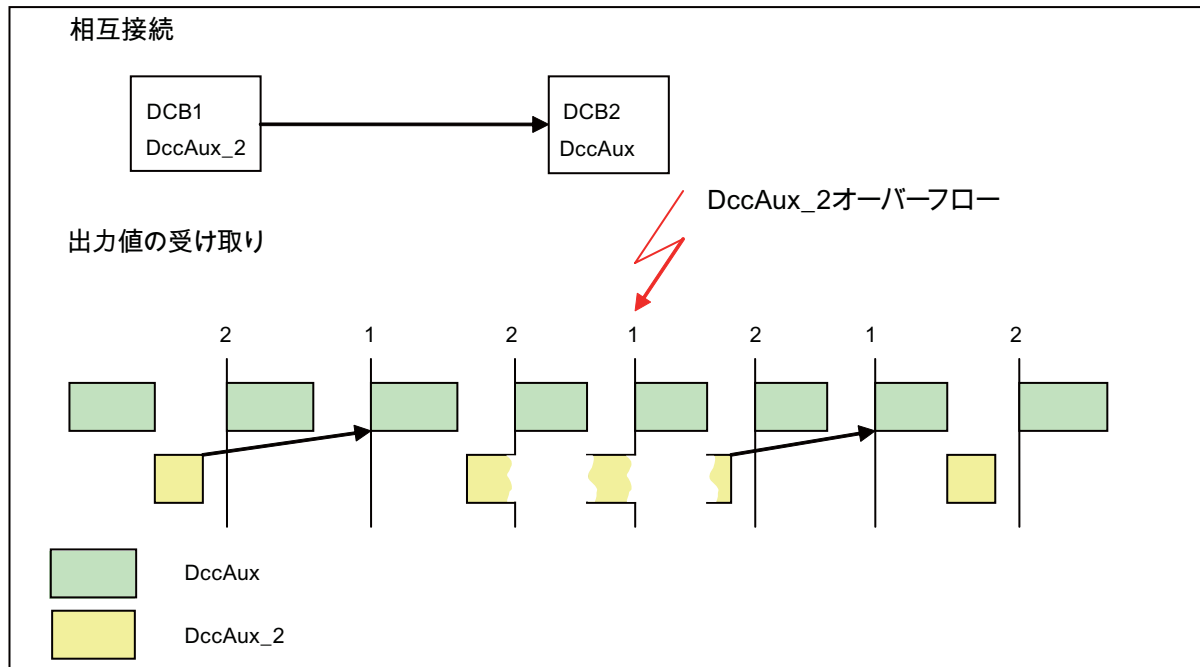


図 5-60 オーバーフローが発生した低優先度レベルからのデータ交換

このオーバーフローのシナリオは、理想的システムにだけ当てはまります。
 実際のシステムでは、DCCタスクだけでなく、ユーザータスクおよびシステムタスクもレベルで実行されています。

ユーザープログラムおよびシステムタスクを考慮に入れた事例

IPOSynchronousユーザープログラムまたはIPOシステムタスクの処理中にオーバーフローが発生した場合は、値がipoDCCタスクで計算済みであるため、高優先度レベルにその値を伝送することができます。
 ただし、次のサイクルクロックでは、新しい値は計算されません。
 ipoDCCタスクでオーバーフローが発生している場合は、上記のように、正しい値は2サイクルクロック後にしか受け取られません。
 2サイクルクロックの間に、値は1つのサイクルクロックによって遅れて受け取られます。

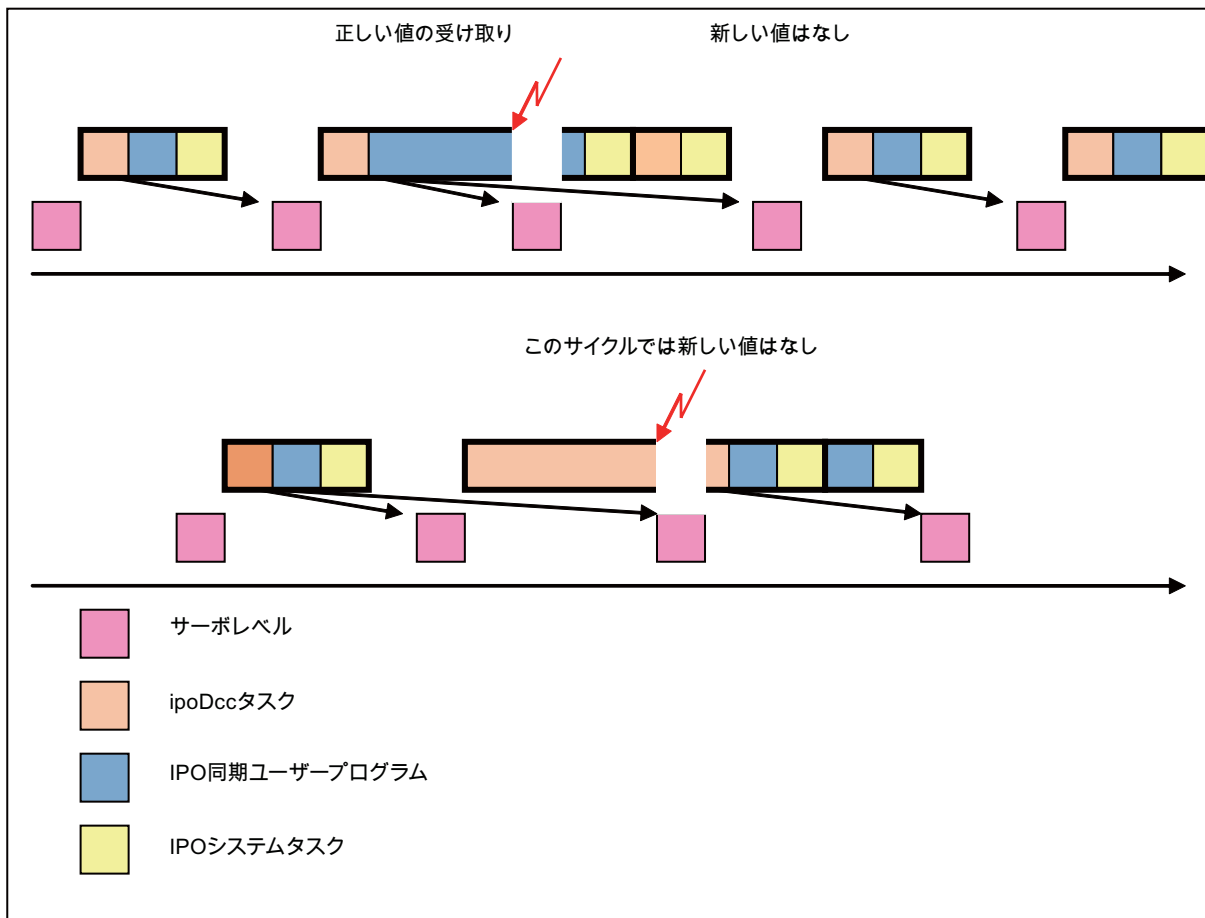


図 5-61 他のタスクを含む実行コンテキストでのオーバーフロー

注記

IPOレベルのipoDCCタスクとサーボレベルのservoDCCの間でのアイソクロナスデータ伝送を確保するには、IPOレベルのipoDccタスクそれ自体のみを考慮するとしても、それがレベルオーバーフローを発生済みであってはいけません。オーバーフローが発生している場合は、高優先度のタスクは、低優先度レベルから伝送された更新値を使用することはできません。この場合、更新された値は、低優先度レベルの次のサイクルの後にしか伝送されません。

5.6.7.4 高優先度レベルからのブロック用データ

説明

高優先度レベルからの出力値にアクセスする場合は、入力値の等時値性を確保するために、高優先度レベルの専用サイクルクロックで値を送送する必要があります。

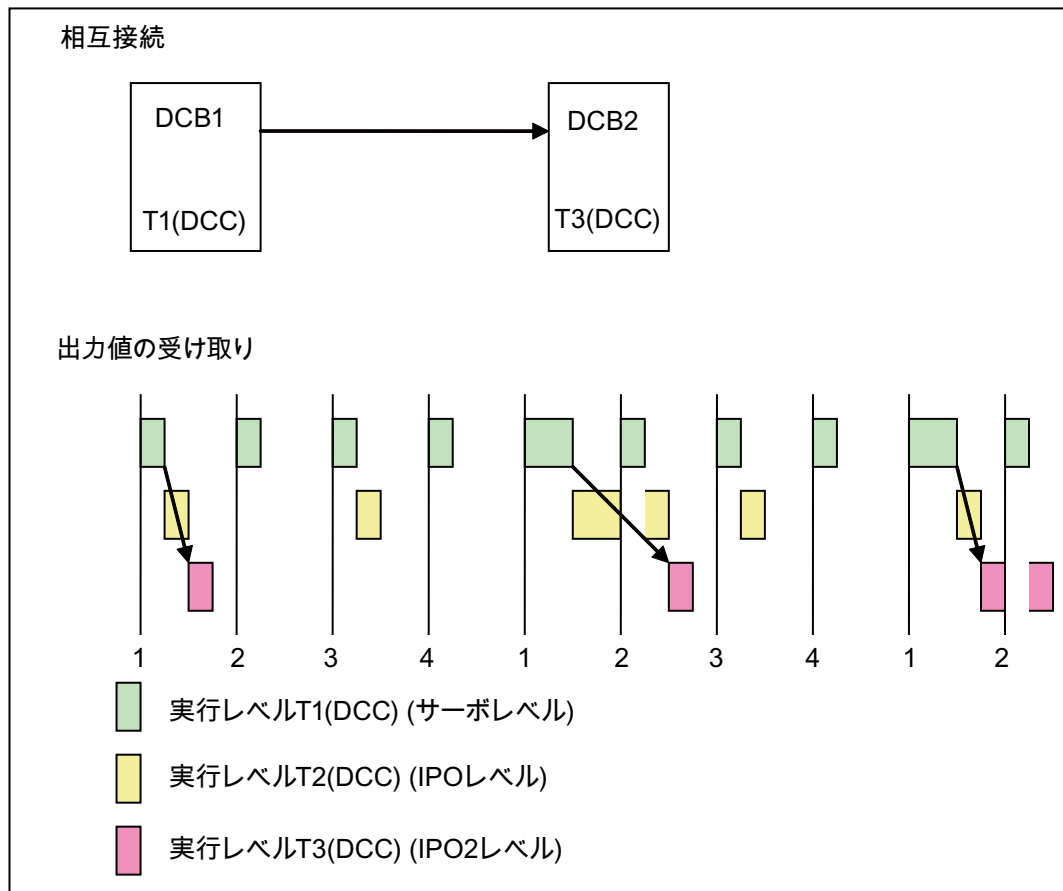


図 5-62 高優先度レベルからのデータ伝送

5.6.8 ブロックと変数の相互接続

5.6.8.1 変数との相互接続

変数との相互接続オプションの概要

ブロックの相互接続オプションに加えて、PINを外部の変数と相互接続することもできます。
 これにより、I/O、テクノロジーオブジェクト、ユーザープログラム、およびHMIとの接続を確立することができます。

以下のユーザー変数をブロックと相互接続することができます。

- I/O変数
- システム変数

- DCCで、以下のユーザー変数を相互接続することができます。
 - グローバルデバイス変数
 - ソースのインターフェースの変数

PINと変数の相互接続

DCCエディタで、各種の入力および出力の相互接続を行うことができます。
詳細は、DCCエディタの説明を参照してください。

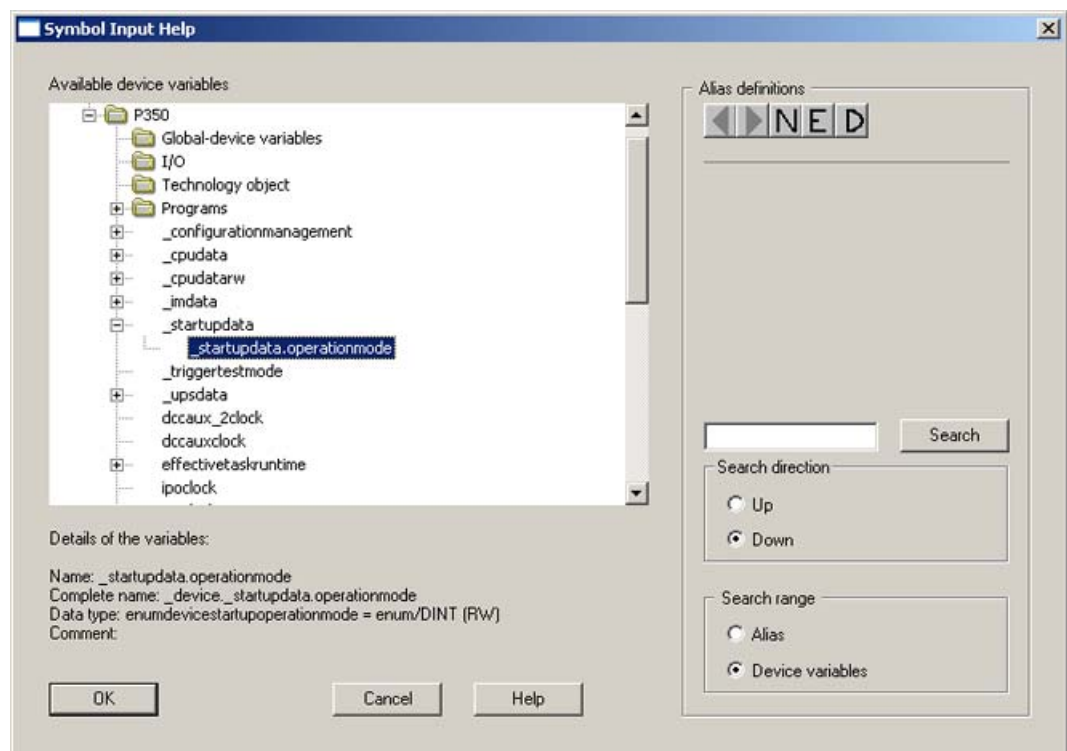


図 5-63 DCCエディタでのユーザー変数の相互接続

注記

1つの出力を変数に相互接続する回数には制限はありません。
最後に計算されたブロックの出力値が常に有効です。

5.6.8.2 FPU例外時の動作

説明

DCCタスクは、FPU(浮動小数点演算ユニット)例外の場合、次の動作を行います。

例外	説明	SIMOTION応答
不正な演算	不正な演算(0/0、INF/INF、INF - INF、SQRT(-1))	デバイスは、STOPモードに切り替わります。
0での除算	0での除算	デバイスは、STOPモードに切り替わります。
オーバーフロー	演算の絶対結果が、abs(+/- Nmax)より大きい値	デバイスは、STOPモードに切り替わります。
アンダーラン	演算の絶対結果が、abs(+/- Nmax)より小さい値	結果は0です。
不正確	結果を正確に表わすことができません。	結果は切り捨てられます。

5.7 ドライブI/Oのインクルード

5.7.1 増設I/OモジュールTM15およびTM17 High Feature

説明

TM15およびTM17 High Feature増設I/Oモジュールは、SIMOTION
モーションコントロールシステムによってアイソクロナスに評価される必要があります。
タイミングの詳細は、TM15/TM17 High
Feature増設I/Oモジュールの試運転マニュアルから得ることができます。

5.7.2 TM31、TM41、TM15 DI/DO増設I/Oモジュール、TB30増設I/Oカード、SIMOTION DまたはCU310/CU320およびCX32のオンボードI/O.

説明

SINAMICSのTM31、TM41、TM15
DI/DO、TB30、およびオンボードI/Oは、フリーランニング(非アイソクロナス)で動作しま
す。

更新サイクルは、以下のドライブパラメータ(デフォルト設定は4 ms)で指定します。

- TM31、TM41、およびTM15 DI/Oの場合、p4099、または
- TB30およびオンボードI/Oの場合、p0799

アクセスは、SINAMICSバックグラウンドタスクで実行されます(たとえば、4 msごとに)。
設定された更新サイクル内の入力および出力へのアクセス時間は、サイクルごとに変わります。
これは、システムが、各サイクル内で更新が行われることだけを保証するためです。更
新サイクルに従った変動範囲があります。
これは、p4099またはp0799を使用して補正することができます。

注記

次の図は、時間の概念図を提供するだけで、絶対量に関する情報は提供しません。

I/Oは、次のタイミングを生成します。

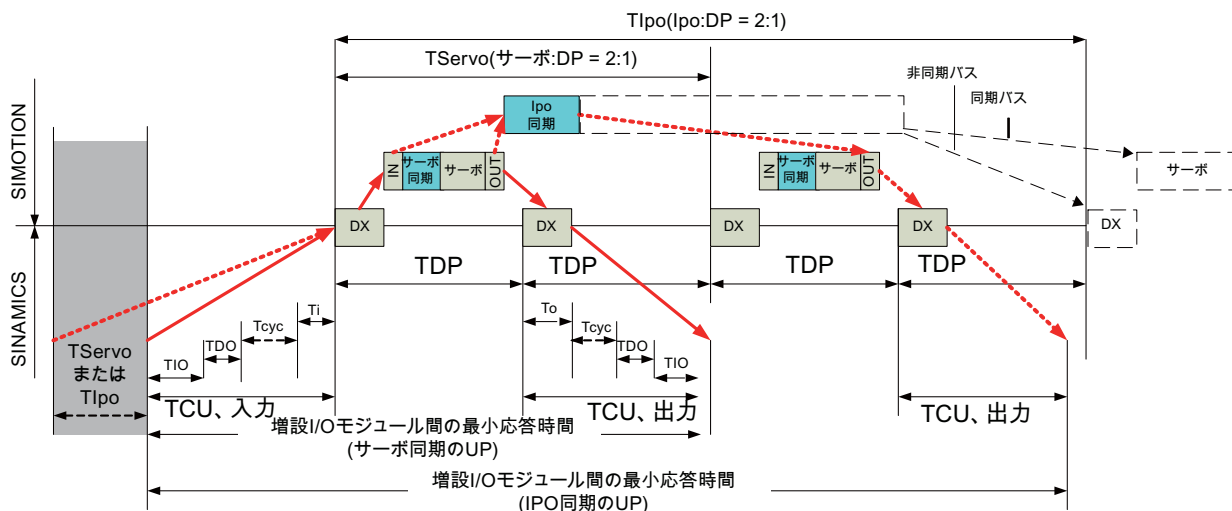


図 5-64 I/Oタイミング

図は、端子 - 端子応答時間に影響するすべての時間を示します。
破線の付いた時間は、以下に示す最悪のケースの時間です。

- 設定された更新サイクル T_{cyc} (= 最大アクセス時間)
- 灰色エリアの T_{servo} または T_{Ipo} 。
入力イベントが発生する時間に応じて、最大端子 - 端子応答時間は、最小端子 - 端子応答時間より最大1サーボサイクルクロックまたは1 I/Oサイクルクロックだけ長くなる可能性があります。

灰色エリアは、入力イベントが、ユーザープログラムの(UP)の次の呼び出しで、取得され、処理される範囲を示します。
この範囲のサイズは、ユーザープログラムが、サーボ同期タスクで実行されるか、I/O同期タスクで実行されるかに応じて異なります。

注記

更新サイクルの減少はSINAMICS閉ループ制御またはSIMOTION D4xxの高負荷を招くため、一定の条件で、ドライブ、増設I/Oモジュールなどのための定量フレームが小さくされる場合があることに注意してください。

SIMOTIONユーザープログラムは、SINAMICSドライブデバイスのI/Oコンポーネントが、BI CO相互接続によって、I/O PZD(プロセスデータ)としてPROFIBUSメッセージフレームに追加されている場合、それらのコンポーネントにアクセスすることができます(『SIMOTION D4xx試運転およびハードウェア据付マニュアル』のセクション「オンボードI/Oおよびドライブオブジェクト用としてコンフィグレーションするメッセージフレーム」を参照)。
V4.1では、特別なテレグラム(39x)を使用することもできます。

I/Oタイミングは、アクセスされるI/Oコンポーネントに応じて異なります。
次の表に、最大遅延時間を示します。端子 - 端子時間を求める場合は、「端子 -> UP」と「UP -> 端子」の時間を追加する必要があります。

注記

IPO同期タスクの図「I/Oタイミング」の破線の付いたエリアは、PROFINET IOにのみ適用されます。このPROFINET IOの場合の時間は、次の表の[IPO同期タスクのUP]列に示します。PROFIBUS DPの場合、[出力]行の T_{IPO} を T_{Servo} に置き換える必要があります。サーボは複数のバスサイクルに渡って実行できないため、破線の付いたエリアはPROFIBUSには無関係です。

			最大遅延時間		
I/Oモジュール			サーボ同期タスクのUP	IPO同期タスクのUP	T_{cyc}
TM31、TM41、TM15 DI/O	Out	UP -> 端子	$T_{DP} + T_{CU\_out}$	$T_{IPO} + T_{DP} + T_{CU\_out}$	p4099
	In	端子 -> UP	$T_{Servo} + T_{CU\_in}$	$T_{IPO} + T_{CU\_in}$	
TB30	Out	UP -> 端子	$T_{DP} + T_o + T_{cyc} + T_{IO}$	$T_{IPO} + T_{DP} + T_o + T_{cyc} + T_{IO}$	p0799 ³⁾
	In	端子 -> UP	$T_{Servo} + T_i + T_{cyc} + T_{IO}$	$T_{IPO} + T_i + T_{cyc} + T_{IO}$	
オンボードI/O D4xx、CX32、CU310、CU320	Out	UP -> 端子 ¹⁾	75μs	75μs	p0799 ³⁾
		UP -> 端子 ²⁾	$T_{DP} + T_o + T_{cyc}$	$T_{IPO} + T_{DP} + T_o + T_{cyc}$	
	In	端子 -> UP	$T_{Servo} + T_i + T_{cyc}$	$T_{IPO} + T_i + T_{cyc}$	

1) 組み込みドライブSIMOTION

D4xxへのアクセス、または、DO1用標準テレグラム39xを使用した場合のCX32へのアクセス。標準テレグラムがない場合のタイミング動作は、CX32の場合と同様です。

2.) CX32へのアクセス、外部PROFIBUSまたはPROFINET IO経由のCU310/CU320アクセス

3) アイソクロナス動作の場合、 $T_{cyc} = \max(T_{DP}, p0799)$ です。

凡例

- T_{DP} : クロックサイクルPROFIBUS(HWコンフィグレーションの設定を参照)
- T_i : アイソクロナスPROFIBUSでの入力のラッチタイム(HWコンフィグレーションのドライブの設定を参照)
- T_o : アイソクロナスPROFIBUSでの出力の遅延(HWコンフィグレーションのドライブの設定を参照)
- T_{IO} : モジュール固有の信号遅延(DRIVE-CLiQサイクル + デジタル入力の入力遅延時間またはデジタル出力の負荷依存出力遅延時間に対応)
- T_{cyc} : SINAMICSの更新サイクル(p4099またはp0799パラメータ)
- [Servo]: SIMOTIONサーボサイクルクロック;
 T_{DP} の倍数(SIMOTIONのサイクルクロック設定を参照)
- IPO: SIMOTION IPOサイクルクロック;
サーボサイクルクロックの倍数(SIMOTIONのサイクルクロック設定を参照)
- T_{DQ} : DRIVE-CLiQサイクルクロック(V4.1では、常に、現在のコントローラのサイクルクロックに対応)

- UP: ユーザープログラム
- CU上の出力時間: $T_{CU\_out} - T_O + T_{cyc} + T_{DQ} + T_{IO}$
- CU上の読み取り時間: $T_{CU\_in} - T_I + T_{cyc} + T_{DQ} + T_{IO}$

実行システム / タスク / システムサイクルクロック のプログラミング

6

6.1 実行システム

6.1.1 実行システム概要

すべてのプログラムは、SIMOTIONデバイス実行システムで実行されます。
実行システムは、各種の実行プロパティを持つ一連の実行レベルを提供します。
プログラムを実行するには、プログラムを実行レベルに割り当てる必要があります。
このために、ソースファイルのプログラムを1つまたは複数のタスクに割り当てます。
これによって、プログラムが実行される順序が定義されます。

注記

プログラムを実行レベルに割り当てるには、ST/MCC/LAD/FBDソースを転送する必要があります。

6.1.2 実行レベルとタスク

実行レベルは、実行システムのプログラムの時間的順序を定義します。
各実行レベルには、1つまたは複数のタスクが含まれます。
タスクは、プログラムの実行フレームワークを提供します。
各タスクには、1つまたは複数のユーザープログラムを割り当て、タスク内でのそれらの順序を指定することができます。
ユーザープログラム以外に、内容と実行順序が固定の複数のシステムタスクがあります。
次の表には、実行レベルと、実行レベルの、ユーザープログラムを割り当てることができるタスク(ユーザープログラムタスク)を示します。
タスク制御コマンドを使用して、実行システムを制御することができます。

表 6-1 SIMOTIONの実行レベル

実行レベル	説明
	実行レベルとタスクの詳細は、『SIMOTIONモーションコントロール基本機能マニュアル』に記載されています。

6.1 実行システム

実行レベル	説明
時間を限定して作成	周期的タスク、割り当てられたプログラムの実行が終了するとすぐに自動的に再起動されます。
SynchronousTasks	タスクは、指定されたシステムサイクルクロックと同期して、周期的に起動されます。 - ServoSynchronousTask: 位置制御サイクルクロックと同期 (SIMOTION Kernel V4.0以降) - IPOSynchronousTask: 補間サイクルクロックIPOと同期 - IPOSynchronousTask_2: 補間サイクルクロックIPO_2と同期 - Tcontrolテクノロジーパッケージ用同期タスク: - PWMSynchronousTask: PWMサイクルクロックと同期 - InputSynchronousTask_1: サイクルクロックInput1と同期 - InputSynchronousTask_2: サイクルクロックInput2と同期 - PostControlTask_1: サイクルクロックControl1と同期 - PostControlTask_2: サイクルクロックControl2と同期
TimerInterruptTasks	タスクは、固定した時間フレームで、周期的に起動されます。この時間フレームは、補間サイクルクロックIPOの倍数でなければなりません。
割り込み	シーケンシャルタスク、起動後に一度だけ実行された後、終了します。
SystemInterruptTasks	システムイベントが発生したときに起動されます。: 詳細 - ExecutionFaultTask: プログラム処理エラー - PeripheralFaultTask: I/Oエラー - TechnologicalFaultTask: テクノロジーオブジェクトのエラー - TimeFaultBackgroundTask: BackgroundTaskタイムアウト - TimeFaultTask: TimerInterruptTaskタイムアウト
UserInterruptTasks	ユーザー定義イベントが発生したエッジトリガで起動されます。
ラウンドロビン	MotionTasksとBackgroundTasksは、高優先度のシステムおよびユーザープログラムタスクが実行された後の残りの空き時間を分け合います。この2つのレベルの比率を割り当てることができます。
MotionTasks	シーケンシャルタスク、起動後に一度だけ実行された後、終了します。タスクの起動は、以下で生じます: - 別のタスクに割り当てられたプログラム内のタスク制御コマンドによって明示的に - RUNモードが取得されたときに自動的に(タスクコンフィグレーションで、該当の属性が設定された場合) MotionTaskの優先度は、システムファンクションWAITFORCONDITIONを使用して、一時的に低くすることができます(「条件を満たすまでMotionTaskを待機させる」を参照)。
BackgroundTask	周期的タスク、割り当てられたプログラムの実行が終了するとすぐに自動的に再起動されます; タスクサイクルタイムはランタイムに応じて異なります。
StartupTask	タスクは、STOPまたはSTOPUモードからRUNモードに移行したとき、一度だけ実行されます。 SystemInterruptTasksは、そのトリガシステムイベントによって起動されます。

実行レベル	説明
ShutdownTask	タスクは、RUNモードからSTOPまたはSTOPUモードに移行したとき、一度だけ実行されます。 以下の場合、STOPまたはSTOPUモードに移行します: - モードセレクトの位置が該当の位置 - SIMOTIONデバイスの該当システムファンクションの呼び出し - 該当エラー応答を持つアラーム(故障)の発生 SystemInterruptTasksとPeripheralFaultTasksは、それらのトリガシステムイベントによって起動されます。
シーケンシャルタスクと周期的タスクの動作に関する情報: - ローカルプログラム変数の初期化については、表「タスク実行動作に応じた、ローカルプログラム変数の初期化」を参照してください。 - プログラムの実行エラーが発生した場合は、セクション「プログラム処理エラー」を参照してください。 プロセスイメージおよびI/Oにアクセスするためのオプションについては、表「直接アクセスおよびプロセスイメージ用の重要なプロパティ」を参照してください。	

6.1.3 タスク起動シーケンス

StartupTask(スタートアップタスク)が完了したとき、RUNモードになります。

この後、以下のタスクが起動されます。

- SynchronousTasks
- TimerInterruptTasks
- BackgroundTask
- スタートアップ属性を持つMotionTasks

注記

RUNモードになった後、これらのタスクが最初に起動される順序は、タスク優先度とは無関係です。

6.1.4 実行システムのコンフィグレーション

6.1.4.1 コンフィグレーション時の設定

実行システムのコンフィグレーション時には、各タスクで実行するプログラムを割り当てます。プログラムを割り当てるとき、以下を指定します。

- プログラムの実行優先度
- 実行動作(シーケンシャル、周期的)

- ローカルプログラム変数の初期化動作

1つまたは複数のタスクへのプログラムの割り当ては、コンパイルが完了した後で、かつ、プログラムをターゲットシステムにロードする前にだけ行うことができます。

プログラムを1つまたは複数のタスクに割り当てた後、ターゲットシステムとの接続を確立し、ターゲットシステムにプログラムをダウンロードして、それを起動することができます。

下記も参照

実行システムのコンフィグレーション (ページ 170)

実行レベル / タスクへのプログラムの割り当て (ページ 170)

6.1.4.2 タスクへのプログラムの割り当て

セクション「サンプルプログラムの実行」では、このサンプルプログラムを使用して、タスク割り当てを設定します。基本的手順を以下に示します。

- プロジェクトナビゲータで、SIMOTIONデバイスを選択し、メニューコマンド[Target system|Configure execution system]を選択します。
SIMOTIONデバイスの実行システム用コンフィグレーションウィンドウが開かれます。
- コンフィグレーションするタスクを選択します。
- [Program assignment]タブを選択し、このタスクに、必要なプログラムを割り当てます。
- [Task configuration]タブを選択し、たとえば以下の追加設定を行います。
 - 動的データフィールド用[Area]制限のローカルデータスタックサイズ(セクション「変数タイプの保管エリア」を参照)
 - プログラムエラー用エラー応答(セクション「プログラム処理エラー」を参照)
 - Time watchdogs for cyclic tasks
 - Start behavior of MotionTasks

下記も参照

実行レベル / タスクへのプログラムの割り当て (ページ 170)

6.1.5 変数初期化に対するタスク実行動作の影響

6.1.5.1 ローカルプログラム変数を初期化するタイミング

タスクの実行動作(シーケンシャルまたは周期的)が、割り当てられたプログラムのローカル変数の初期化タイミングを決定します。
以下の説明は、プログラムのローカル変数として宣言されたファンクションブロックのインスタンスにも適用されます。

すべての変数タイプの一覧とそれらの初期化のタイミングについては、セクション「変数初期化のタイミング」を参照してください。

表 6-2 タスク実行動作に従ったローカルプログラム変数の初期化

実行動作	タスク	ローカルプログラム 変数の初期化
シーケンシャル (周期的でない)	MotionTasks、UserInterruptTasks、SystemInterruptTasks、StartupTask、ShutdownTask	起動後に、シーケンシャルタスクは一度だけ実行された後、終了します。 起動する度に、割り当てられたプログラムのすべてのローカル変数が初期化されます。 データ初期化の時間は、タスクランタイムに含まれます。 コンパイラオプション"1回だけのプログラムデータ初期化"時の動作: 割り当てられたプログラムのローカル変数は、1回だけ初期化されます。
サイクリック	BackgroundTask、SynchronousTasks、TimerInterruptTasks	周期的タスクは、完了時に、自動的に再起動され、割り当てられたプログラムの静的変数の値(VAR / END_VARで宣言)は保持されます。 静的変数は、STOPからRUNへの移行時に、1回だけ初期化されます。 一時的変数(VAR_TEMP / END_VARで宣言)は、タスクが起動される度に初期化されます。 静的変数の1回だけの初期化は、タスクランタイムに含まれません。 コンパイラオプション"1回だけのプログラムデータ初期化"時の動作: 割り当てられたプログラムのローカル変数は、1回だけ初期化されます。

6.1.5.2 ユニット変数への初期値の割り当て

ユニット変数とグローバルデバイス変数は、STOPまたはSTOPUモードからRUNモードへの移行時には、初期化されません(セクション「*変数初期化のタイミング*」を参照)。

それでもなお、これらの変数に初期値を割り当てたい場合はStartupTaskを使用します。

通知

タスク起動シーケンスは、RUNモードへの移行後は、定義できません(セクション「*タスク起動シーケンスTask start sequence*」を参照)。

StartupTask以外のタスクを使用する場合は、正しい初期値の割り当ては保証されません。

6.1.5.3 複数のVAR_GLOBAL、VAR_GLOBAL RETAINブロックの使用

説明

ユニットのインターフェースおよび実装エリア(V4.1)では、複数のVAR_GLOBAL、VAR_GLOBAL RETAINブロックを作成することができます。

インターフェースエリアおよび実装エリアでは、複数の宣言ブロックを任意の順序で指定することができます。

これらのブロックのそれぞれが、独立したバージョン管理の対象になります。

このため、1つのブロック内で変更(直接の変更であれ、データタイプの変更による間接的な変更であれ)が行われると、再ロード時に、これらのブロックが再初期化されます。

さらに、(RETAIN)データ保持はブロックごとに機能します。

RUN時に、既存のデータブロックに影響することなしに、最後に新しいブロックを追加し、変更されたソースを再ロードすることができます。

1つのソースセクション(ステートメントはインターフェースと実装では別々に適用されます)で、1つのブロックを同一タイプ(RETAINまたは非RETAIN)の既存のブロックの前に追加すると、それ以降のすべてのブロックは変更され、ダウンロード時に、再初期化されます。このため、このような変更を行った後は、RUNでダウンロードすることはできません。

ファンクション_saveUnitDataSet/_loadUnitDataSetおよび_exportUnitDataSet/_importUnitDataSetを使用して、ユニットデータブロック情報を保存することができます。

データセット読み取り時に、1つまたは複数のブロックが欠けている場合は、戻りコードで検出することができます。この場合、それ以外のブロックは読み取られます。

初期化によるデータの損失

これらのデータは、前もって、SCOUTの"Save variables"ファンクションで保存し、"Restore variables"ファンクションで復元することができます。

その代わりに、アプリケーション_exportUnitDataSet/_importUnitDataSetのランタイムファンクションを使用することもできます。

6.1.5.4 変数初期化に対するコンパイラの影響

コンパイラオプション"拡張言語要素"(非IEC_conformityの場合)

このコンパイラオプションは各ソースで設定できるため、グローバル設定は上書きされます。

このオプションは作成言語に関わりなく有効であるため、LAD/FBDおよびMCCでも使用することができます。

このコンパイラオプションは、以下のことを可能にします。

- ビット列変数(BOOLを除く)の直接ビットアクセス、ビットアドレス指定
- "ブロック"の外部のファンクションブロックからのINPUT変数の読み取りおよび書き込み
- "プログラム内プログラム"の呼び出し

POU内のグローバルFBインスタンスと同様に、1つのプログラムを異なるプログラム内で呼び出すことができます(たとえば、異なるプログラム内での"myprog"の呼び出し、異なるFB内での呼び出しなど)。ただし、ファンクション内での呼び出しはできません。

インスタンスデータのグローバル可用性は、"プログラム内プログラム"の必須条件です。この条件は、PROGRAMにインスタンスデータが1つも存在しないか、(存在したとしても)呼び出されるPROGRAMのコンパイル時にコンパイラオプション"1回だけのプログラムデータインスタンス化"を使用した場合に満たされます(「1回だけのプログラムデータインスタンス化」を参照)。

注記

このコンパイラオプションを設定しないと、プログラムの動作はV4.0と同じです(DEFAULTTに対応)。

コンパイラオプション"1回だけのプログラムデータインスタンス化"

このコンパイラオプションは各ソースで設定できるため、グローバル設定は上書きされます。

このコンパイラオプションは作成言語に関わりなく有効であるため、LAD/FBDおよびMCCでも使用することができます。

このコンパイラオプションは、ソースに含まれるPROGRAMでのインスタンスデータの作成方法を決定します。

PROGRAMのインスタンスデータは、VAR宣言ブロックの内容によって構築されます。

通常:

1つのプログラムは、タスクあたり1回かぎり、実行システムにフックすることができます。

このコンパイラオプション"1回だけのプログラムデータインスタンス化"を有効化すると(異なるタスクでプログラムを使用するときも)、以下のことが実行されます。

- この方法でコンパイルされたインスタンスデータは、1回だけ作成されます。インスタンスデータは、PROGRAMが宣言されたソースに存在します。
- この方法で作成されたPROGRAMを使用する度に、同一のデータが有効になります。これは、タスク(おそらく、複数のタスク)への割り当てや他のPROGRAMまたはファンクションブロックでの呼び出しに影響を及ぼします。
- 設定"1回だけのプログラムデータインスタンス化"を使用した場合、データ初期化はグローバル初期化のルールに従って行われ、プログラムが宣言されたソース / コードのダウンロードも行われます(プロジェクトの"ダウンロード"設定を参照)。

"1回だけのプログラムデータインスタンス化"を設定しない場合(DEFAULTT)、以下の動作が適用されます。

- 異なるLADから、このプログラムを呼び出すことはできません。
- プログラムのインスタンスデータは、プログラムが割り当てられたタスクごとに別々に作成されます。
- インスタンスデータのデータ初期化は、タスクの起動付きで実行されます(タスク起動を含むシーケンシャルタスク、STOP - RUN移行の周期的タスク)。
- このモードの場合、シーケンシャルタスクのPROGRAMのインスタンスデータをRUNで変更することができます(ローディングが可能な場合にかぎり)。

グローバル変数ブロックの属性

VAR_GLOBAL、VAR_GLOBAL

RETAINブロックでは、ブロックの先頭で、以下のプラグマがサポートされています。

- `HMI_Export := [true|false];`
デフォルト位置から離れたHMIデバイスについて、アドレスエクスポートが行われます(INTERFACEはエクスポートされますが、IMPLEMENTATIONはエクスポートされません)。
- `BlockInit_OnChange := true;`
(IMPLEMENTATIONのみ)RUN時のダウンロード用ブロック構造が変更された場合、ソースで指定された値によるデータの再初期化が実行されます。

このプラグマは、PROGRAMのVAR宣言でも使用することができます。

ただし、VAR宣言のプラグマは、コンパイラオプション"1回だけのプログラムデータインスタンス化"を選択した場合だけ有効です。

6.1.5.5 HMI関連データのマーキング

データを"非HMI関連データ"としてマーキングする

ブロックの先頭でコンパイラプラグマを使用して、インターフェースエリアのブロックを非HMI関連としてマーキングします。

この場合、収納された変数のoperate&observeアドレスは生成されなくなり、このブロックの変更はHMI適合性に対してもう影響を及ぼしません。

表 6-3 例

```
VAR_GLOBAL
{HMI_Export := false; }
x : INT;
y : INT;
END_VAR
```

データを"HMI関連データ"としてマーキングする

ブロックの先頭でコンパイラプラグマを使用して、インターフェースエリアのブロックを非HMI関連としてマーキングします。

この場合、収納された変数のoperate&observeアドレスは生成されなくなり、このブロックの変更はHMI適合性に対してもう影響を及ぼしません。

表 6-4 例

```
VAR_GLOBAL
{HMI_Export := true; }
x : INT;
y : INT;
END_VAR
```

これは、すべてのHMI関連変数が、HMIアクセスの64kbyteアドレス制限以下でなければならないことを示します。

V4.1より前のデバイスバリエーションの場合、これはソース内のブロックの位置に対応しますが、RETAINブロックは、ソースセクションでの位置に関わりなく、動的ブロックの前に配置されます(変更ができなくなります)。

V4.1のデバイスバリエーションでは、HMI関連ブロックだけが以前のバージョンと同様にHMIアドレススペースのエリアを占めます。

64kbyteアドレス制限を超えた場合、警告が出され、HMIアクセスが不能になった変数が表示されます。

HMI用のコンパイラプラグマを指定して明示的にデータブロックをエクスポートする場合に、HMI経由でブロックの変数にアクセスできない場合は、コンパイル時に、エラーメッセージが出されます。

注記

適合性チェック設定なしのHMIからのアクセスは、VAR_GLOBALエリア全体の終わりに変数を追加した場合だけ可能です。これは、変数が最後のVAR-GLOBALブロック変数に追加されるか、VAR-GLOBAL定義ブロック全体が終わりに追加されることを意味します。

6.1.6 タスクのステータス

6.1.6.1 タスク状態の検索と意味

ファンクション_getStateOfTaskId(タスクID)を使用して、タスクステータスをチェックすることができます。

このファンクションは、入力パラメータとしてタスクIDを必要とし、データタイプDWORDの値を戻します。

SIMOTION

KernelバージョンV3.0までは、ファンクション_getStateOfTask(名前)を使用します。

次の表には、タスク状態の可能な組み合わせを、16進表示と記号定数として示します。タスクステータスは組み合わせることができ、16進数値の合計で表示されます。

表 6-5 タスク状態とその意味

シンボリック定数	16進表記	シンボリック定数
TASK_STATE_INVALID	16#0000	タスクが存在しません。
TASK_STATE_STOP_PENDING	16#0001	停止の信号を受け取ったタスクが、TASK_STATE_RUNNINGとTASK_STATE_STOPPEDの中間状態になっています。 タスクが停止するまで、動作が実行されることがあります。
TASK_STATE_STOPPED	16#0002	タスクが停止されたか(ファンクション_resetTaskなどにより)、完了したが、まだ起動されていません。

6.1 実行システム

シンボリック定数	16進表記	シンボリック定数
TASK_STATE_RUNNING	16#0004	以下のような実行中のタスク • ファンクション_startTaskによって起動されたタスク • アクティブな周期的タスクとして実行されているタスク
TASK_STATE_WAITING	16#0010	_waitTimeファンクションまたはWAITFORCONDITIONによって待機しているタスク
TASK_STATE_SUSPENDED	16#0020	_suspendTask関数により一時停止状態にされたタスクです。
TASK_STATE_WAIT_NEXT_CYCLE	16#0040	開始トリガを待っているTimerInterruptTask(タイマ割り込みタスク)です。
TASK_STATE_WAIT_NEXT_INTERRUPT	16#0080	トリガイイベントが発生するのを待っているSystemInterruptTasks(システム割り込みタスク)またはUserInterruptTask(ユーザ割り込みタスク)です。
TASK_STATE_LOCKED	16#0100	_disableScheduler関数によりロックされたタスクです。

6.1.6.2 タスク状態の組み合わせ

_getStateOfTaskId(および_getStateOfTask)は、頻繁に、表「静的および一時変数の宣言のキーワード」に収納されたソースファイルセクションに応じて、OR演算で生成されるタスク状態の組み合わせとして値を戻します。
次の表に、頻繁に起きる組み合わせをリストします。

表 6-6 頻繁に起きるタスクステータスの組み合わせ

組み合わせ	16進表記	意味
TASK_STATE_WAITING OR TASK_STATE_RUNNING	16#0014	タスクは実行中ですが、たとえば、_waitTimeまたはWAITFORCONDITIONなどで待機中です。
TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#0024	タスクは実行中ですが、現在、_suspendTaskによって一時停止されています。
TASK_STATE_WAIT_NEXT_CYCLE OR TASK_STATE_RUNNING	16#0044	TimerInterruptTaskは実行中ですが、現在、その起動トリガを待機しています。
TASK_STATE_WAIT_NEXT_CYCLE OR TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#0064	TimerInterruptTaskは実行中で、その起動トリガを待機中ですが、_suspendTaskによって一時停止されています。
TASK_STATE_WAIT_NEXT_INTERRUPT OR TASK_STATE_RUNNING	16#0084	SystemInterruptTaskまたはUserInterruptTaskが実行中ですが、現在、そのトリガイイベントを待機しています。
TASK_STATE_WAIT_NEXT_INTERRUPT OR TASK_STATE_SUSPENDED OR TASK_STATE_RUNNING	16#00A4	SystemInterruptTaskまたはUserInterruptTaskが実行中で、そのトリガイイベントを待機していますが、_suspendTaskによって一時停止されています。
TASK_STATE_LOCKED OR TASK_STATE_RUNNING	16#0104	タスクは実行中ですが、現在、_disableSchedulerによってロックされています。

他の組み合わせも可能です。

6.1.6.3 タスク状態の使用例

次の例では、起動が可能かどうかを確認するために、MotionTaskのタスクステータスが検索されています。戻り値の関連ビットが評価されます。
評価結果が正の場合、MotionTaskが起動されます。

表 6-7 MotionTaskが起動可能かどうかの検索例

```
ret_dword := _getStateOfTaskId (id := _task.motionTask_1);
IF (ret_dword AND
    (TASK_STATE_STOPPED OR TASK_STATE_STOP_PENDING)
) <> 0 THEN
    // MotionTaskが起動可能
    ret_dword := _restartTaskId (id := _task.motionTask_1);
ELSE
    ; // MotionTaskは起動できません
END_IF;
```

MCCの例は、『MCCプログラミングマニュアル』のコマンド[Task status]を参照してください。

6.1.7 MotionTaskで条件が満たされるのを待機する

6.1.7.1 EXPRESSIONの条件の構文

以下は、STのプログラミング例です。MCCの場合、wait条件の定義はコマンドで直接に行います。

MotionTaskでWAITFORCONDITIONコマンドを使用して、条件が満たされるのを待機することができます(たとえば、イベントの発生)。

このステートメントが呼び出されるMotionTaskは、条件が満たされるまで、TASK_STATE_WAITINGステータスに保持されます。

VAR_IN_OUT入力 / 出力パラメータを使用するとき、WAITFORCONDITIONの時間監視を可能にすることができます。

この条件は、式の形で指定します。

式は、ファンクション宣言の特別なタイプです(構文については、『STプログラミングマニュアル』のセクション「式」を参照してください)。

- 戻り値のデータタイプはBOOLで、明示的には指定されません。
- VAR_IN_OUT入力 / 出力パラメータとVAR_INPUT入力パラメータの使用が可能です。

式の宣言は、ユニットの実装セクションだけで行うことができます。

オプションとして、宣言セクションで、ローカル(一時的)変数を宣言することができます。他の宣言(たとえば、入力パラメータ、定数など)を行うことはできません。

ステートメントセクションでは、以下にアクセスすることができます。

- 式のローカル変数
- ユニット変数
- グローバルデバイス変数、I/O変数、およびプロセスイメージ

BOOLデータタイプの式は、式のステートメントセクションで、式IDに割り当てる必要があります(『STプログラミングマニュアル』のセクション「式」を参照してください)。

注記

式のステートメントセクションに、ファンクション呼び出しやループを含めることはできません。

6.1.7.2 WAITFORCONDITIONステートメントの構文

次の構文は、条件として式を持つ待機タスク用のコマンドを呼び出すために使用されます。

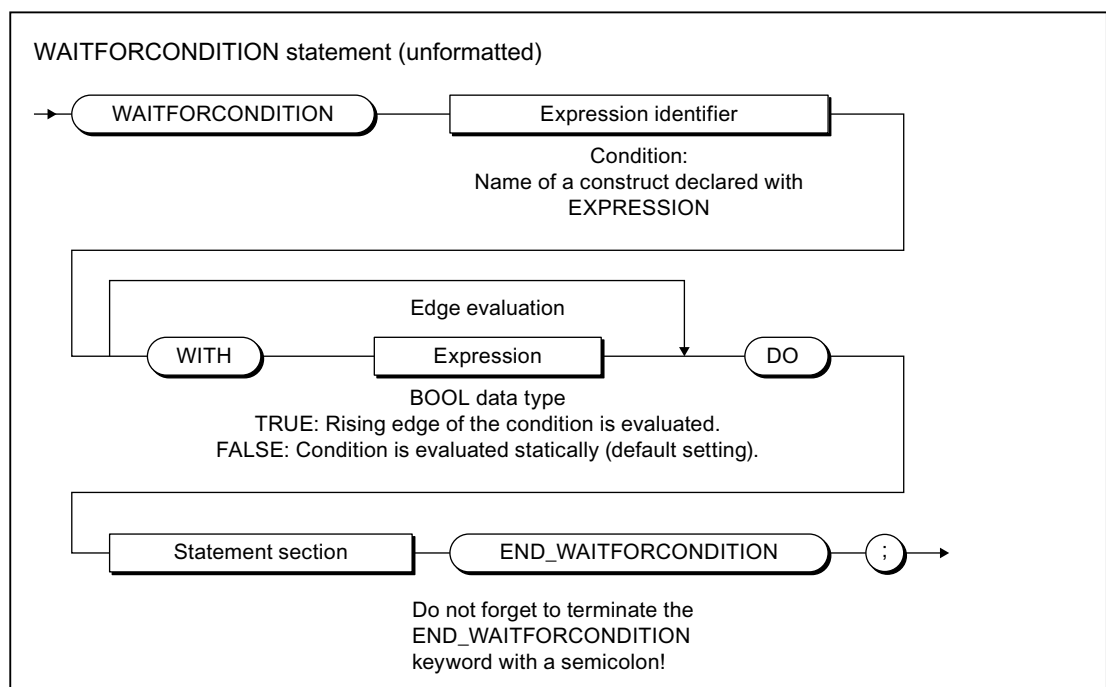


図 6-1 WAITFORCONDITIONステートメントの構文

式IDはEXPRESSIONで宣言されるコンストラクトで、その値(必要な場合、WITHエッジ評価とともに)は条件が満たされたかどうかを定義します。

The WITHエッジ評価シーケンスはオプションです。
エッジ評価はBOOLデータタイプの式で、式IDの値の評価方法を決定します。

- エッジ評価 = TRUE:
式IDの立ち上がりエッジが評価されます。すなわち、式IDの値がFALSEからTRUEに変わったときに、条件が満たされます。

- エッジ評価 = FALSE:
式IDの静的値が評価されます。すなわち、式IDの値がTRUEであるときに、条件が満たされます。

エッジ評価が指定されない場合のデフォルト設定はFALSEです。すなわち、式IDの静的値が評価されます。

NOT 式IDを使用して、立ち下がりエッジをチェックすることもできます。

6.1.7.3 WAITFORCONDITIONステートメントの効果

WAITFORCONDITIONステートメントは、条件(式)がTRUEになるまで、それを呼び出したタスクをTASK_STATE_WAITING ステータスに設定します。

式がTRUEになると、タスクの優先度が増します。

この優先度はUserInterruptTasksより高く、SystemInterruptTasksより低い優先度です。すなわち、このMotionTaskは、ラウンドロビン実行レベルの他のタスク、UserInterruptTasks、およびTimerInterruptTasksより前に起動されます。

高くされたタスク優先度は、WAITFORCONDITIONコマンドとEND_WAITFORCONDITIONコマンドの間のすべてのステートメントに適用されます。

END_WAITFORCONDITIONコマンドで、タスク優先度が元の優先度に戻ります。

ステートメントセクションには、少なくとも1つの空ステートメントを含めなければなりません。

WAITFORCONDITIONコマンドを使用するときは、以下のことに注意してください。

- このコマンドは、MotionTaskでイベントを待機するために使用されます。
別のタスクでこのコマンドがプログラミングされた場合、このコマンドは無視されます。
- MotionTaskには、時間ウォッチドッグは存在しません。
このため、条件が実際にTRUEになることを確認してください。
そうでないと、MotionTaskが待機状態であり続け、タイムアウトエラーが発生します。
- V4.1では、モーションタスクのWAITFORCONDITIONで、時間監視をプログラミングすることができます。
- WAITFORCONDITION ...
END_WAITFORCONDITION構造をネストすることはできません。
- ラウンドレベル内では、待機タスクは、高優先度のタスク(アラームなど)によって妨げられないかぎり、次に実行されます(イベントが発生したとき)。
- ラウンドロビンレベルの有効なタスクのタイムスライスは割り込まれます。
- タスク割り込み時の動作については、_suspendTaskを参照してください。
- 式は、高優先度のIPOサイクルクロックでチェックされます。

下記も参照

タスク優先度 (ページ 133)

MotionTasks (ページ 140)

6.1.7.4 WAITFORCONDITIONの使用例

次の例は、MotionTaskで、*feeder*プログラムが実行中であることを前提にしています。
このMotionTaskに対して、オプション[Activation after StartupTask]が選択されています。

タスクへのプログラムの割り当ては、SIMOTION SCOUTで実行されます(「**実行レベル / タスクへのプログラムの割り当て**」を参照)。

表 6-8 WAITFORCONDITION条件の使用例

```

INTERFACE
  USEPACKAGE cam;
  PROGRAM feeder;
END_INTERFACE

IMPLEMENTATION

  // MotionTask_1のWAITFORCONDITIONの条件
  EXPRESSION automaticExpr
    automaticExpr := IOfeedCam; // デジタル入力
  END_EXPRESSION
  // feeder(MotionTask_1)
  PROGRAM feeder
    VAR
      retVal : DINT;
    END_VAR ;
    retVal := _enableAxis(axis := realAxis,
      enableMode := ALL,
      servoCommandToActualMode := INACTIVE,
      nextCommand := WHEN_COMMAND_DONE,
      commandId := _getCommandId() );
    // 自動的起動の待機
    WAITFORCONDITION automaticExpr WITH TRUE DO
      retVal := _pos(axis := realAxis,
        positioningMode := RELATIVE,
        position := 500,
        velocityType := DIRECT,
        velocity:=300,
        positiveAccelType := DIRECT,
        positiveAccel:= 400,
        negativeAccelType := DIRECT,
        negativeAccel := 400,
        velocityProfile:= TRAPEZOIDAL,
        mergeMode:=IMMEDIATELY,
        nextCommand:=WHEN_MOTION_DONE,
        commandId:= _getCommandId() );
      // WAITFORCONDITION後に優先度を低くする
    END_WAITFORCONDITION;
    retVal := _disableAxis(axis := realAxis,
      disableMode := ALL,
      servoCommandToActualMode := INACTIVE,
      nextCommand := WHEN_COMMAND_DONE,
      commandId := _getCommandId() );
    END_PROGRAM
  END_IMPLEMENTATION

```


6.1.7.5 FB経由の時間監視の例

FB経由の時間監視を行うWAITFORCONDITIONの使用例(V4.1以降)

次の例は、時間監視の式(WAITFORCONDITION)です。
 呼び出し点の式と、時間監視を含む呼び出しファンクションブロックのインスタンスデータとの結合は、右にパラメータとして指定されます。これは、TONタイプの参照変数です。
 変数の呼び出しとその出力の照会は、式内で実行されます。

```

VAR_GLOBAL
  v1, v2 : INT;
  t1, t2 : TIME;
END_VAR

EXPRESSION exp
  VAR_IN_OUT
    v : INT;
    t : TON;
  END_VAR
  t();
  exp := v > 100 and not t.q ;
END_EXPRESSION

FUNCTION_Block waitfb
  VAR_IN_OUT
    refpar1 : INT;
    refpar2 : TIME;
  END_VAR
  VAR_TEMP
    expr_timeout : TON;
  END_VAR
  expr_timeout(pt := refpar2, IN := true);
  // 監視時間の設定と
  // タイマの起動
  WAITFORCONDITION exp(v := refpar1, t := expr_timeout ) DO
  ... ;
  // ステートメント

  IF (expr_timeout.q) THEN
    // タイムアウト時のエラー処理が実行可能
    ... ;
  END_IF;
END_WAITFORCONDITION
END_FUNCTION_BLOCK

```

この後、それぞれの場合に異なる変数を持つwaitfbタイプのインスタンスの呼び出しを任意の時点で実行することができます。

これらのグローバル変数は、その後、周期的タスクで更新することができます。

```

my_waitfb_1(refpar1 := v1, refpar2:=t1); // V1 >
100になるが、時間t1が経過するまで待つ
my_waitfb_2(refpar1 := v2, refpar2:=t1);

```

6.1.7.6 非周期的タスク / MotionTaskで直接に時間監視を行うWAITFORCONDITIONの使用例

説明

式(WAITFORCONDITION)の時間監視の例

その呼び出しは、MotionTaskで直接に実行されます。

この時間監視はTONタイプで、式内で、呼び出しとその出力の照会が行われます。

```

VAR_GLOBAL
    v1, v2 : INT;
    t1, t2 : TIME;
END_VAR

EXPRESSION exp
    VAR_IN_OUT
        v : INT;
        t : TON;
    END_VAR
    t();
    exp := v > 100 and not t.q ;
END_EXPRESSION
VAR_TEMP
    expr_timeout : TON;
END_VAR
.....
    expr_timeout(pt := t1, IN := true); // 監視時間の設定と
                                        // タイマの起動
                                        // v1 > 100になるか、
                                        // 時間t1が経過するまで待つ
WAITFORCONDITION exp(v := v1, t := expr_timeout ) DO
    ... ; // ステートメント
    IF (expr_timeout.q) THEN
    ... ; // タイムアウト時のエラー処理が実行可能
    END_IF;
END_WAITFORCONDITION

.....
v1は、任意の周期的タスクで更新することができます。

```

6.1.8 タスクを指定した時間だけ待機させる

タスクを待機状態にする

特定の時間、タスクをTASK_STATE_WAITINGステータスにすることができます(「タスク状態 (ページ 235)」を参照)。これを行うには、ファンクション_waitTimeを使用します。これは、指定された時間、呼び出しタスクを待機状態にします。

注記

待機状態のタスクは、CPU時間を全く(ほとんど)要求しません。
ターゲットシステムの唯一の負荷は、待機状態が終了したかどうかを確認する周期的チェックだけです。このチェックは、IPOサイクルクロックで行われます。

MotionTaskにおける *_waitTime (timeValue := T#0ms)* の呼び出しは、これらを一時的に不活動にし、プログラム制御をスケジューラに戻します。
これは、たとえば、プログラム制御を意図的に次のラウンドロビンタスク(BackgroundTaskとしても可能)に移行するための長時間ループの場合に、お勧めします。

通知

このファンクションは、MotionTasksのみで使用してください。これを周期的タスクで使用する、時間監視エラーにつながります!

- SynchronousTasksでは SIMOTION Kernel V3.2以降では、時間監視を中断するかを設定することができます。
時間監視は、デフォルト¹で有効です。
IPOsynchronousTaskでは、これ以外に以下を考慮します。
UserInterruptTasksは、トリガ事象では開始されなくなっています!
- 他の周期的タスク(BackgroundTask、TimerInterruptTasks)では
時間監視は常に有効です。

周期的タスクで、タイマシステムファンクションブロック(セクション「タイマを参照)を使用して、待機時間を実装します。

¹ SIMOTION

KernelのバージョンV3.1まででは、SynchronousTasksでは時間監視は中断されます。

データタイプTIMEの式を入力パラメータとして使用します。データタイプDINTの戻り値は常に0です。

このファンクション(構文)の詳細は、ファンクション *_waitTime* を参照してください。

次のサンプルプログラムは、このプログラムが割り当てられたMotionTaskを、10秒間、待機状態にします。

表 6-9 ファンクション *_waitTime* の例

```
INTERFACE
    PROGRAM waitTime;
END_INTERFACE

IMPLEMENTATION
    PROGRAM waitTime
        VAR
            retVal :DINT;
        END_VAR;
        retVal := _waitTime(timeValue := T#10s);
    END_PROGRAM
```

END_IMPLEMENTATION

6.1.9 RUN時のダウンロードの手順

6.1.9.1 RUN時のダウンロード

RUN時のダウンロード

RUN時のダウンロードでは、RUNモードで、プロジェクトソースを選択したターゲットシステムにダウンロードすることができます。

RUN時に、TO、HWコンフィグレーションの設定、I/O変数をダウンロードすることはできません。
同様に、変数は初期化されません(初期化されると、現在のシステム値が上書きされることになります)。

RUN時のダウンロードの補足条件

- ダウンロード時には、ユニットのプログラムは実行中であってははいけません(周期的タスクの場合、サイクル変更時に、ロードの試みが行われます。
指定された時間が経過すると、その手順は中止されます。)
処理中のMotionTasksは変更できません。
- 変更を行うことができる周期的タスクの数は、4つまでに制限されます(すべてのタスクをダウンロードするために許可されている時間内に、ダウンロード時点を見つける必要があります)。
- ソースと関連タスクの数が多すぎると、すべての変更を処理できなくなる場合があります。
この問題は通知され、完全な変更セットはシーケンスに伝送されません。
この場合、古いデータが保持されます。
- VARおよびVAR_GLOBALのプログラム構造は、周期的タスクで使用されている場合、変更してはいけません(周期的タスク関連のデータは、タスクの実行を通じて保持されます)。
VAR_TEMPは、プログラムとFBの場合、変更でき、VARは、FCの場合、変更することができます。
MotionTasks関連のプログラムデータは、起動する度に再初期化されるため、変更することができます。
- 非周期的タスク(MotionTasks)関連のプログラムインスタンスデータは、変更することができます。プログラムインスタンスデータは、プログラム内のVAR変数です。
- 以下のコンパイラオプションとプラグマを使用して、手順を最適化することができます。
 - コンパイラオプション"1回だけのプログラムインスタンスデータの設定"(下記を参照)
 - プラグマ"BlockInit_OnChange"({Blockinit_OnChange := TRUE;})
- [Allow download|copy RAM to ROM during RUN option under Options|Settings|CPU Download]を選択して、SCOUTでのRUN時のダウンロードを可能にします。
ダウンロードを実行するには、デバイスコンテキストメニューで、[Target device|Download]を選択します。

- ダウンロード手順の一部として、ダウンロードを[CPU in RUN]または[STOP]で行うオプションが提供されています。

1回だけのプログラムデータインスタンス化のコンパイラオプション

"1回だけのプログラムインスタンスデータの設定"が設定されていないと(デフォルト)、この手順は旧バージョンの場合と同じです。すなわち、データ初期化はタスク起動時に行われます。プログラムインスタンスデータは、集中的に保管されます。

同一のプログラムに複数のタスクを割り当てると、タスクごとに、別々のインスタンスデータが作成されます。

1つのプログラムのインスタンスデータを変更すると、他のすべてのアクティブなプログラムのインスタンスデータにも影響を及ぼします。これらが他の場所に存在し、RUN時のダウンロードに関して制約を課すためです。

"1回だけのプログラムインスタンスデータの設定"オプションを選択しないと(すなわち、プログラムデータがタスクベースでインスタンス化されます)、プログラムデータがタスク起動時に初期化されるため、インスタンスデータが変更された場合でも、RUN時に非周期的タスクをダウンロードすることができます。

インスタンスデータは、実行システムのシンボルブラウザで、診断目的のためだけに使用できます。

コンパイラ設定"1回だけのプログラムインスタンスデータの設定"を選択すると、データ初期化では、プログラム、または、プログラムが存在するソースのダウンロードも行われます。

この方法でコンパイルされたプログラムのインスタンスデータは、プログラムが複数のタスクで使用される場合でも、一度だけ作成されます。

この場合、インスタンスデータは、プログラムのソースまたはコードに存在します。

これは、RUN時のダウンロードの場合に効果的です。

インスタンスデータは、ユニットのシンボルブラウザで、診断目的のために使用できます。

注意

データ初期化動作は現在、V4.0と同じではありません。シーケンシャルタスクの場合、データはタスク起動時に初期化されるのではなく、ダウンロード時にのみ処理化されます。このため、タスク開始時に、変数を初期化する必要があります。

1つのプログラムに複数のタスクを割り当てると、すべてのタスクで、同一のデータが使用されます。

- グローバル設定の場合、[Options|Settings|Compiler]の[Only put in program instance data once]を選択します。
- ローカル設定の場合は、プログラムソース(MCCおよびLAD/FBD)またはプログラム(ST)を追加するときに、[Insert xxx]ダイアログボックスで、[Compiler]の[Only put in program instance data once]を選択します。
ダウンロードを実行する前に、このコンパイルオプションは、[RUN]に設定する必要があります。

"BlockInit_On_Change"プラグマ

"1回だけのプログラムインスタンスデータの設定"オプションを選択した場合でも、RUN時のダウンロード処理中には、インスタンスデータを変更することはできません。

これを行うには、"BlockInit_OnChange := True;"プラグマを使用する必要があります。

このプラグマは、RUN時のダウンロード処理中のブロック構造変更時に、ソースに指定された値で、データの再初期化を行います。

6.1 実行システム

このプラグマは、プログラムのVAR宣言でも使用することができます。
 ただし、VAR宣言のプラグマは、コンパイラオプション"1回だけのプログラムインスタンスデータの設定"を選択した場合だけ有効です。
 STでのみ使用します。"BlockInit_OnChange"プラグマはMCCおよびLAD/FBDでは使用できません。構文例は、Block_Init_on_Changeの構文の例 (ページ 251)を参照してください。

プラグマは、必要に応じて、任意の時点に設定することができます。

例:

プログラム内のVARブロックの変更のためにRUN時のダウンロードを実行できない場合は、ダウンロードの後に、VARブロックにプラグマを設定します。
 コンパイルの後に、VARブロックの再初期化が完了しているため、RUN時のダウンロードを実行することができます。

次のダウンロードの前にプラグマを取り消して、再初期化を回避することができます。

6.1.9.2 RUN時のダウンロードのオプション

RUN時のダウンロードの有効化

以下の適切なプログラミングを行って、RUNモードでのダウンロード操作を可能にします。

- 可能な場合は常に、実装セクションで、USESステートメントを使用します。
 この方法では、ダウンロードに関連するユニットがほとんどなくなります。
- タスクおよびプログラムの相互割り当て方法に従って、ユニットを作成します。
 これは相互依存関係を最小化し、プログラミングを容易にします。

例:

ユニットに2つのプログラムが含まれ、1つがBackgroundTaskに、もう1つMotionTaskに割り当てられています。
 この場合、MotionTaskの動作中には、RUNモードで、このユニットをダウンロードすることはできません。

- タスクコンフィグレーションでのローカルデータスタックサイズの設定時に(『SIMOTION モーションコントロール基本機能』機能マニュアルの「実行システムのコンフィグレーション」を参照): RUNモードでのダウンロード用の予備を設定します。
- 変更されたユニットのプログラムがMotionTaskに割り当てられている場合、MotionTaskが動作中でないことを保証する措置を取ります:
 - MotionTasksで、継続ループを避けます!
 たとえば、WHILEループの代わりに、ファンクション_restartTaskを使用します。
 - 1つのMotionTask、複数のMotionTasks、またはすべてのMotionTasksをリセットできるように、オペレータコントロールファンクションの用意をしておきます(単一クロックモードなど)。
- 連続して実行されるプログラムセクション(MCCの実行制御など)は、特にMotionTasksに関するかぎり、専用のユニットに保管することをお勧めします(これらのユニットは、未使用のとき、オンにすることができます)。
- コンパイラオプション"1回だけのプログラムインスタンスデータの設定"を使用します。
 この場合は、同一のプログラムを複数のタスクに割り当ててはいけません。
- 追加の変数ブロックまたはコンパイラプラグマ"BlockInit_OnChange := True;"を使用して、ユニットのインターフェースおよび実装セクションのTYPEおよびグローバル変数を変更することができます。

- プログラムインスタンスデータ(VAR)は、"BlockInit_OnChange := True;"コンパイラプラグマを使用して、変更、再初期化することができます。
- 新しいIO変数が必要な場合は、BackgroundTask用のプロセスイメージ(0~63バイト)に配置する必要があります。

例:

```
VAR_GLOBAL
```

```
    anio AT %I2.7 : BOOL;
```

```
END_VAR
```

- 追加の変数ブロックを使用する代わりに、新しいデータで新しいユニットを作成し、US ES(実装セクション)とリンクすることもできます。
これは、MCCおよびLAD/FBDプログラミング言語で実行することもできます。

RUN時のダウンロードのプログラム例

下の画面は、変数の挿入または変更を行う場合に、RUN時のダウンロードが可能になる時点と条件を示す、プロジェクトの実行システムの構造を示します。
表示から明らかなように、実行システムは、それぞれが異なるユニットを含むタスクから構成されます。

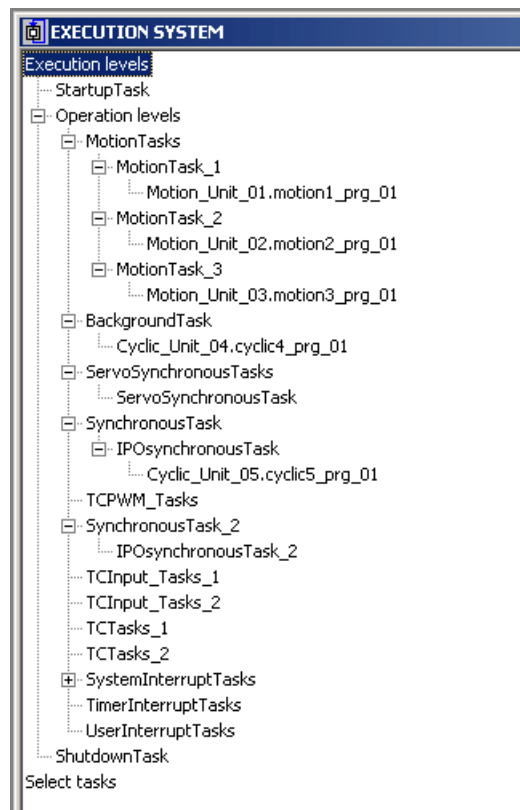


図 6-2 RUN時のダウンロード - プログラム例

この画面は、連続して呼び出すことができるプログラムユニットを示します。
たとえば、*Motion_Unit_01*は*FunctionBlock_01*を呼び出し、*FunctionBlock_01*は2つのファンクション*Function_01*および*Function_02*を呼び出します。

6.1 実行システム

Motion_Unit_01とCyclic_Unit_05は、「RUN時のダウンロードのオプション一覧」に記載されている補足条件によってのみ結合されます。

赤で強調表示されたエリアは、RUN時のダウンロードで問題が発生する可能性があるプログラムブロックです。

Motion_Unit_02には、そこでFunctionBlock_02とFunction_02が順番に呼び出されるWhile TRUEループが含まれます(「Whileループの例 (ページ 251)」を参照)。ループはRUN時のダウンロードができないため(可能なダウンロード点なし)、そこで呼び出されるブロックとファンクションもブロックされます。逆にRUN時のダウンロードが可能な条件を下の表にリストします。

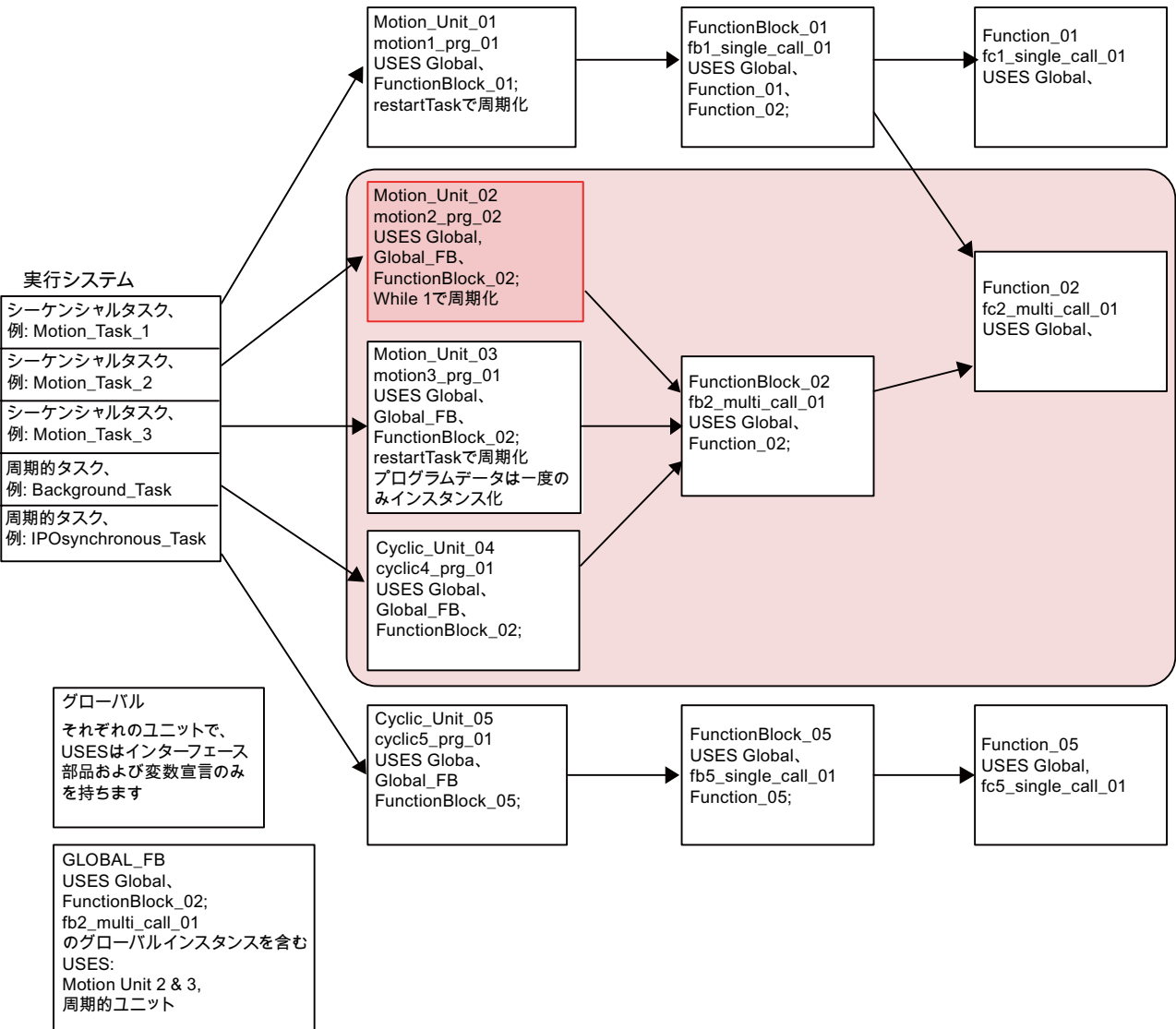


図 6-3 RUN時のダウンロード

RUN時のダウンロードのオプション一覧

以下の表は、変更可能なデータと変更場所を示します。

変更位置	I/O変数	
変更位置	グローバルデバイス変数	

変更位置	インターフェース					
	タイプ	Var_Global	Var_Global Retain	Var_Global Constant	USES	USEPACKAGE
Motion_Unit_01	4) 2)	4)	4)	6)		
Motion_Unit_02	1)	1)	1)	6)	1)	1)
Motion_Unit_03	4) 2)	4)	4)	6)		
Cyclic_Unit_04	4) 2)	4)	4)	6)		
Cyclic_Unit_05	4) 2)	4)	4)	6)		
Function_Block_01	4) 2)	4)	4)	6)		
Function_Block_02	1)	1)	1)	6)	1)	1)
Function_Block_05	4) 2)	4)	4)	6)		
Function_01	4) 2)	4)	4)	6)		
Function_02	1)	1)	1)	6)	1)	1)
Function_05	4) 2)	4)	4)	6)		
グローバル	1)	1)	1)	6)	1)	1)
Global_FB		1)		6)	1)	1)

変更位置	実装					
	タイプ	Var_Global	Var_Global Retain	Var_Global Constant	USES	USEPACKAGE
Motion_Unit_01	4) 2)	4)	4)	6)		
Motion_Unit_02	1)	1)	1)	6)	1)	1)
Motion_Unit_03	4) 2)	4)	4)	6)		
Cyclic_Unit_04	4) 2)	4)	4)	6)		
Cyclic_Unit_05	4) 2)	4)	4)	6)		
Function_Block_01	4) 2)	4)	4)	6)		
Function_Block_02	1)	1)	1)	6)	1)	1)
Function_Block_05	4) 2)	4)	4)	6)		
Function_01	4) 2)	4)	4)	6)		
Function_02	1)	1)	1)	6)	1)	1)
Function_05	4) 2)	4)	4)	6)		

変更位置	プログラム					
		タイプ	Var	Var_Temp	Var_Constant	コード変更
Motion_Unit_01	motion1_prg_01					
Motion_Unit_02	motion2_prg_01	1)	1)	1)		1)

6.1 実行システム

変更位置	プログラム					
		タイプ	Var	Var_Temp	Var_Constant	コード変更
Motion_Unit_03	motion3_prg_01	 2)7)	 3)			
Cyclic_Unit_04	cyclic4_prg_01	 2)5)	 5)			
Cyclic_Unit_05	cyclic5_prg_01	 2)7)	 3)			

変更位置	ファンクションブロック								
		タイプ	Var	Var_Temp	Var_Constant	コード変更	Var_Input	Var_In_Out	Var_Output
FunctionBlock_01	Fb1_single_call_01								
FunctionBlock_02	Fb2_multi_call_01	 1)2)	 1)7)		 6)		 1)7)	 1)7)	 1)7)
FunctionBlock_03	Fb5_single_call_01	 2)7)	 7)		 6)		 7)	 7)	 7)

変更位置	ファンクション							
		タイプ	Var	Var Constant	コード変更	Var_Input	Var_In_Out	戻り値
Function_01	Fc1_single_call_01							
Function_02	Fc2_multi_call_01					 1)	 1)	 1)
Function_03	Fc5_single_call_01							

表の主要項目

脚注	説明
1)	MotionTask_2は、ダウンロードを開始する前に停止する必要があります。 ダウンロードは、プログラムの継続ループ(WHILE TRUE)によって阻止されます。 ダウンロードは、"WAITFORCONDITION"でプログラムが長く待機しすぎる場合、プログラムが"_waitTime"に設定されたままである場合、または、プログラムが同期コマンドの終了を待っている場合にも、阻止されます。
2)	使用という観点から、構造的な変更は、変数の作成 / 変更と同じです。
3)	"BlockInit_OnChange"プラグマを使用します。
4)	追加の変数ブロックまたは"BlockInit_OnChange"プラグマを使用して、追加の変数を挿入する必要があります。
5)	"1回だけのプログラムインスタンスデータの設定"オプションを選択し、"BlockInit_OnChange"プラグマを使用している場合だけ可能です。
6)	使用法に応じて異なる変更機能: コードは常に失われます。 フィールド長は、変数の作成 / 変更時と同じです。"BlockInit_OnChange"プラグマによって可能になります。 初期値はコードと同様に常に失われます。

脚注	説明
7)	再初期化が必要です。"BlockInit_OnChange"プラグマの使用が必要になる場合があります。
	常に、ユニットの内容全体のダウンロードが可能でなければなりません。 ユニットのプログラムの1つ(たとえば、インターフェース、実装など)がダウンロードできない場合は、ユニット全体のダウンロードが阻止されます。
	常に変更可能です。
	制限された変更が可能です。注意事項は、脚注を参照してください。
	変更できません。その理由は、脚注を参照してください。

6.1.9.3 Whileループの例

Whileループの構造の例

Whileループは、While ... DO式がTRUE以外の値になるまで、実行され続けます。

```
//*****
PRG2_Counter1 := PRG2_Counter1 + 1;
WHILE TRUE DO
    PRG2_Counter2 := PRG2_Counter2 + 2;
    my_fb2_multi_call_01(
        fb2_multi_call_01_inp_real_1      := 1.0,
        fb2_multi_call_01_inout_real_1    := Motion2_prg_01_Real_1
    );
END_WHILE;
//*****
```

6.1.9.4 Block_Init_on_Changeの構文の例

Block_Init_on_Changeの構造の例

```
//*****
Var_Global
{BlockInit_OnChange := TRUE;}
Interface_Var_Global1 : INT;
Interface_Var_Global2 : REAL;
END_VAR
//*****
```

6.2 タスク制御コマンド

6.2.1 タスク制御コマンドの一覧

次の表に、タスク制御のために使用できるコマンドを一覧表示します。

これらのファンクションは、SIMOTION
KernelバージョンV3.1以降で 사용할 수 있습니다。

表 6-10 SIMOTION STのタスク制御コマンド

タスク制御コマンド	重要度
<code>_startTaskId(タスクID)</code>	TASK_STATE_STOPPED状態のMotionTaskを起動します。 このタスクのスタートアップコードが実行されます。 MotionTasksに対してだけ使用することができます。 <code>_resetTaskId</code> の直後に使用することはできません。 この場合は、代わりに、 <code>_restartTaskId</code> を使用します。
<code>_resetTaskId(タスクID)</code>	MotionTaskをTASK_STATE_STOPPEDにリセットします。 MotionTasksに対してだけ使用することができます。
<code>_restartTaskId(タスクID)</code>	MotionTaskをTASK_STATE_STOPPEDにリセットし、起動します。このMotionTaskのスタートアップコードが実行されます。 MotionTasksに対してだけ使用することができます。 <code>_startTaskId</code> は、このコマンドの直後に使用することはできません。 この場合は、代わりに、 <code>_restartTaskId</code> を使用します。
<code>_suspendTaskId(タスクID)</code>	該当のタスクが一時停止(TASK_STATE_SUSPENDEDに設定)されます。周期的タスクの場合、時間監視が一時停止されます。 SynchronousTasks、StartupTask、およびShutdownTaskに対して使用することはできません。
<code>_resumeTaskId(タスクID)</code>	TASK_STATE_SUSPENDED状態の該当のタスクが再開されます。時間監視も再実行されます。 SynchronousTasks、StartupTask、およびShutdownTaskに対して使用することはできません。
<code>_getStateOfTaskId(タスクID)</code>	該当のタスクの状態を照会します。
<code>_retriggerTaskIdControlTime(タスクID)</code>	該当のタスクの監視時間が1回だけリセットされます。

タスクコマンド用のMCCブロックについては、『MCCプログラミングマニュアル』に記載されています。

タスク制御コマンドの例は、「タスク制御コマンド (ページ 254)」を参照してください。

「タスク状態 (ページ 235)」も参照。

タスクID

これらのファンクションで、タスクは一意のタスクIDによって指定されます。
タスクのタスクIDは、以下の方法で取得することができます。

- `_task.名前変数`(たとえば、`_task.motionTask_1`など)。用語の説明:
 - `_task`は、タスクID用の事前定義された名前空間です(セクション「[名前空間](#)」を参照)。
名前空間の識別子は、次の識別子とピリオドで区切られます。
 - `名前`は、実行システムで割り当てられたタスクの識別子です(たとえば、`motionTask_1`など)。
- ファンクション `_getTaskId(名前)`の使用。
タスクはその名前によって指定されます(実行システムと同様)。
- ファンクション `_checkEqualTask`を使用して、1つのタスクIDが特定のタスクに属するかどうかを確認することができます(「[ファンクション_checkEqualTask](#)」を参照)。

注記

SIMOTION

KernelバージョンV3.0以前でも、同様のファンクションを使用できます。これらのファンクションでは、タスクはその名前で指定されます(実行システムと同様)。
これらのファンクションは、ライブラリで使用してはいけません。

これらのファンクションは、SIMOTION

KernelのバージョンV3.1では使用できません。これらのファンクションの可用性は、SIMOTION Kernelの将来のバージョンでは保証されていません。

さらに、SIMOTIONデバイスもスケジューラ制御用のシステムファンクションを提供しています。

次の表に、これらのシステムファンクションを示します(詳細は、SIMOTIONデバイスのパラメータマニュアルを参照)。

表 6-11 スケジューラ制御用のコマンド

タスク制御コマンド	重要度
<code>_disableScheduler()</code>	<code>_enableScheduler</code>が呼び出されるまで、すべてのユーザープログラムタスク(SynchronousTasksを除いて)がロードされないようにします。ただし、これはシステムタスクには影響しません。 注記:周期的タスクのための時間監視機能は、一時停止状態にはなりません。 注記: このコマンドは、SystemInterruptTasksとUserInterruptTasksの交換も阻止します!
<code>_enableScheduler()</code>	<code>_disableScheduler</code> の効果を取り消します。

通知

`_task.名前形式`のファンクション `_checkEqualTask` および `_getTaskId`と変数は、ライブラリでは使用してはいけません。

6.2.2 タスク制御コマンドの使用例

次の例は、SIMOTION SCOUTで以下の割り当てが行われていることを前提にしています。

- StartupTaskへの *myStart* プログラムの割り当て
- MotionTask_1への *myMotion* プログラムの割り当て

コントローラのスタートアップ(STOPからRUNへの移行)後に、StartupTaskが自動的に実行されます。このタスクで、*myStart* プログラムが実行されます。

_startTaskId タスク制御コマンドが、MotionTask_1を起動するために *myStart* プログラムで使用されます。MotionTask_1では、*myMotion* プログラムが実行されます。

```
INTERFACE
  PROGRAM myStart;
  PROGRAM myMotion;
END_INTERFACE

IMPLEMENTATION
  VAR
    RetDWord : DWORD;
  END_VAR

  PROGRAM myStart
    RetDWord := _startTaskId (_task.motionTask_1);
    // MotionTask_1の起動

  END_PROGRAM
  // ここには、プログラムのコマンド(軸コマンドなど)を指定します
  PROGRAM myMotion

  END_PROGRAM
END_IMPLEMENTATION
```

6.2.3 ファンクション_getStateOfTaskId

このファンクションは、SIMOTION Kernel V3.1以降で使用できます。

このファンクションは、該当のタスクの状態を戻します。

タスクは、プロジェクト全体で一意的タスクIDによって指定します(「タスク制御コマンドに関する全般情報」を参照)。

このファンクションは、StartupTaskとShutdownTaskを除くすべてのタスクに対して使用することができます。

「タスク状態」も参照。

MotionTaskの起動の可否をタスクステータスに基づいて決定する例も存在します(「タスク状態の使用例」を参照)。

SIMOTION
KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_getStateOfTaskId (  
    { id      : StructTaskId // タスクID  
    }  
    ) : DWORD
```

入力パラメータ

id (オプション)
データタイプ: StructTaskId
デフォルト: このファンクションが呼び出される現在のタスクのタスクID
制御するタスクのタスクID(「タスク制御コマンドに関する全般情報」を参照)

戻り値

データタイプ: DWORD
タスクのステータスが、以下の値をOR演算した値として表示されます(「タスク状態」も参照)。

16#0000	タスクが存在しないか、タスクIDが不正です TASK_STATE_INVALID
16#0001	タスクがRUNからSTOPへ移行中です TASK_STATE_STOP_PENDING
16#0002	タスクが停止されています TASK_STATE_STOPPED
16#0004	タスクが実行中です TASK_STATE_RUNNING
16#0010	タスクが待機中です TASK_STATE_WAITING
16#0020	タスクが一時停止されています TASK_STATE_SUSPENDED
16#0040	TimerInterruptTaskが、起動トリガを待機中です TASK_STATE_WAIT_NEXT_CYCLE
16#0080	SystemInterruptTaskまたはUserInterruptTaskが、起動イベントの発生を待機中です TASK_STATE_WAIT_NEXT_INTERRUPT
16#0100	タスクが(_disableSchedulerによって)ロックされています TASK_STATE_LOCKED

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
getStateOfTask (// 短い形式のみが許されます
    { name      : Task_Name // タスクの名前
    }
) : DWORD
```

このファンクションは、以下の点を除いて、ファンクション_getStateOfTaskIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用してはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

下記も参照

タスク状態の検索と意味 (ページ 235)

6.2.4 ファンクション_resetTaskId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

MotionTaskをTASK_STATE_STOPPEDにリセットします。

タスクは、プロジェクト全体で一意的タスクIDによって指定します(「**タスク制御コマンドに関する全般情報**」を参照)。

このファンクションは、MotionTasksに対してのみ、 사용할 수 있습니다。

ファンクション _startTaskIdは、このファンクションの直後に 사용할 수 없습니다。この場合、代わりに、ファンクション _restartTaskId를 사용합니다。

SIMOTION KernelバージョンV3.0以前の類似ファンクション: ファンクション_resetTask

SIMOTION

KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_resetTaskId (
    id      : StructTaskId // MotionTasksのみ
) : DWORD
```


入カパラメータ

id
データタイプ: StructTaskId
制御するタスクのタスクID

戻り値

データタイプ: DWORD
0 エラーなし
16#FFFF_FFFE タスクIDがMotionTaskを指していません
16#FFFF_FFFF タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_resetTask (// 短い形式のみが許されます
            name : Task_Name // タスクの名前
            ) : VOID // MotionTasksのみ
```

このファンクションは、以下の点を除いて、ファンクション_resetTaskIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、戻り値を持たない。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

下記も参照

ファンクション_restartTaskId (ページ 258)

ファンクション_startTaskId (ページ 262)

6.2.5 ファンクション_restartTaskId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다.

MotionTask를 TASK_STATE_STOPPED에 리셋하고, 재시작합니다. 이 MotionTask의 데이터가 초기화됩니다.

타스크는, 프로젝트 전체에서 하나의 타스크ID에 의해 지정됩니다(「**타스크 제어 명령에 대한 일반 정보**」를 참조).

이 함수는, MotionTasks에 대해서만, 사용할 수 있습니다.

이 함수가 실행 중인 MotionTask를 중지하기 위해 사용될 수 있도록 하기 위해, MotionTask의 상태를 조회, 평가할 수 있습니다(「**타스크 상태**」의 함수와 _getStateOfTaskId와 타스크 상태의 사용 예를 참조).

SIMOTION

Kernel 버전 V3.0 이전의 유사 함수에 대해서는, 섹션의 끝에 설명합니다.

通知

```
_restartTaskId (
    id      : StructTaskId // MotionTasks의
) : DWORD
```

入力パラメータ

id
 데이터 타입: StructTaskId
 제어할 타스크의 타스크ID(「**타스크 제어 명령에 대한 일반 정보**」를 참조)

戻り値

데이터 타입:	DWORD
0	에러なし
16#FFFF_FFFE	타스크ID가 MotionTask를 지칭하지 않습니다
16#FFFF_FFFF	타스크ID가 잘못되었습니다

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```

_resetTask (// 短い形式のみが許されます
            name : Task_Name // タスクの名前
            ) : VOID // MotionTasksのみ

```

このファンクションは、以下の点を除いて、ファンクション_restartTaskIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、戻り値を持たない。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

6.2.6 ファンクション_resumeTaskId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

TASK_STATE_SUSPENDED状態のタスクを再開します。

このステータスには、ファンクション_suspendTaskIdによって移行しています。

タスクは、プロジェクト全体で一意的タスクIDによって指定します(「**タスク制御コマンドに関する全般情報**」を参照)。

このファンクションは、MotionTasks、BackgroundTask、TimerInterruptTasks、UserInterruptTasks、SystemInterruptTasksに対して使用することができます。

周期的タスク(BackgroundTask、TimerInterruptTasks)の場合、タスクの時間監視も再び有効化されます。

SIMOTION

KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```

_resumeTaskId (
    id      : StructTaskId
) : DWORD

```

入力パラメータ

id

データタイプ: StructTaskId

制御するタスクのタスクID(「**タスク制御コマンドに関する全般情報**」を参照)

戻り値

データタイプ:	DWORD
0	エラーなし
16#FFFF_FFFE	タスクIDが不正なタスクを指しています
16#FFFF_FFFF	タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_resumeTask (// 短い形式のみが許されます
              name : Task_Name // タスクの名前
              ) : VOID
```

このファンクションは、以下の点を除いて、ファンクション_resumeTaskIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、戻り値を持たない。
- このファンクションは、ライブラリで使用してはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

下記も参照

ファンクション_suspendTaskId (ページ 263)

6.2.7 ファンクション_retriggerTaskIdControlTime

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

該当のタスクの監視時間を1回だけリセットします。
タスクは、プロジェクト全体で一意的タスクIDによって指定します(「タスク制御コマンドに関する全般情報」を参照)。

このファンクションは、MotionTasks、BackgroundTask、TimerInterruptTasks、UserInterruptTasks、SystemInterruptTasksに対して 사용할 수 있습니다。

SIMOTION KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_retriggerTaskIdControlTime (
    { id      : StructTaskId
    }
) : DWORD
```

入力パラメータ

id (オプション)
 データタイプ: StructTaskId
 デフォルト: このファンクションが呼び出される現在のタスクのタスクID
 制御するタスクのタスクID(「タスク制御コマンドに関する全般情報」を参照)

戻り値

データタイプ: DWORD
 0 ファンクションが正常に実行されました
 非0 ファンクションが正常に実行されませんでした

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_retriggerTaskControlTime ( // 短い形式のみが許されます
    { name      : Task_Name // タスクの名前
    }
) : DWORD
```

このファンクションは、以下の点を除いて、ファンクション_retriggerTaskIdControlTimeと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用してはいけません。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

6.2.8 ファンクション_startTaskId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다.

TASK_STATE_STOPPED状態のMotionTaskを起動します。このタスクのデータが初期化されます。

タスクは、プロジェクト全体で一意的タスクIDによって指定します(「タスク制御コマンドに関する全般情報」を参照)。

このファンクションは、MotionTasksに対してのみ、 사용할 수 있습니다.

以下の状態のMotionTasksには、効果を持ちません。

- TASK_STATE_RUNNING
- TASK_STATE_WAITING
- TASK_STATE_SUSPENDED

このファンクションを使用する前に、MotionTaskのステータスをポーリング、評価할 수 있습니다(「タスク状態」のファンクションと_getStateOfTaskIdとタスク状態の使用例を参照)。

このファンクションは、ファンクション_resetTaskIdの直後に 사용할 수 없습니다. この場合、代わりに、ファンクション_restartTaskId를 사용합니다.

SIMOTION

KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_startTaskId (
    id      : StructTaskId  // MotionTasksのみ
) : DWORD
```

入力パラメータ

id
 데이터 타입: StructTaskId
 制御するタスクのタスクID(「タスク制御コマンドに関する全般情報」を参照 – MotionTasksのみ)

戻り値

데이터 타입:	DWORD
0	エラーなし
16#FFFF_FFFE	タスクID가 MotionTask를 지시하고 있지 않습니다
16#FFFF_FFFF	타스크ID가 잘못되었습니다

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_startTask (// 短い形式のみが許されます  
            name : Task_Name // タスクの名前  
            ) : VOID // MotionTasksのみ
```

このファンクションは、以下の点を除いて、ファンクション_startTaskIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、戻り値を持たない。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

下記も参照

ファンクション_getStateOfTaskId (ページ 254)

ファンクション_resetTaskId (ページ 256)

ファンクション_restartTaskId (ページ 258)

6.2.9 ファンクション_suspendTaskId

このファンクションは、SIMOTION Kernel V3.1以降で使うことができます。

該当のタスクを一時停止します(タスクをTASK_STATE_SUSPENDEDに設定します)。タスクは、プロジェクト全体で一意的タスクIDによって指定します(「**タスク制御コマンドに関する全般情報**」を参照)。

このファンクションは、MotionTasks、BackgroundTask、TimerInterruptTasks、UserInterruptTasks、SystemInterruptTasksに対して使うことができます。

周期的タスク(BackgroundTask、TimerInterruptTasks)の場合、タスクの時間監視も停止されます。

タスクとその時間監視は、ファンクション_resumeTaskIdで再開されます。

SIMOTION KernelバージョンV3.0以前の類似ファンクション:
ファンクション_suspendTask

SIMOTION KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_suspendTaskId (
    id          : StructTaskId
) : DWORD
```

入力パラメータ

id
データタイプ: StructTaskId
制御するタスクのタスクID(「タスク制御コマンドに関する全般情報」を参照)

戻り値

データタイプ:	DWORD
0	エラーなし
16#FFFF_FFFE	タスクIDが不正なタスクを指しています
16#FFFF_FFFF	タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_suspendTask (// 短い形式のみが許されます
    name      : Task_Name    // タスクの名前
) : VOID
```

このファンクションは、以下の点を除いて、ファンクション_suspendTaskIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、戻り値を持たない。
- このファンクションは、ライブラリで使用してはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

下記も参照

ファンクション_resumeTaskId (ページ 259)

6.2.10 ファンクション_getTaskId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다.

このファンクションは、タスク名(実行システムと同じタスク名)からプロジェクト全体で一意的なタスクIDを生成します。

このタスクIDは、データタイプStructTaskIdの変数に割り当て、以下のファンクションの入力パラメータとして 사용할 수 있습니다.

- タスク制御コマンド
- タスクのランタイム測定用ファンクション

通知

このファンクションは、ライブラリで使用するはいけません。

このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストリング、形式パラメータの指定なし)だけで呼び出すことができます。

通知

```
_getTaskId ( // 短い形式のみが許されます
            name      : Task_Name // タスクの名前
            ) : StructTaskId
```

入力パラメータ

name (短い形式のみが許されます)
実行システムで定義されたタスク名

戻り値

データタイプ: StructTaskId
戻り値には、タスクのタスクIDが収納されています。

6.2.11 ファンクション_checkEqualTask

このファンクションは、1つのタスクIDが1つのタスクと関連付けられているかどうかを示します。

通知
このファンクションは、ライブラリで使用するはいけない。 このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。

通知

```
_checkEqualTask (// 短い形式のみが許されます
    id          : StructTaskId,
    名前: TaskName // タスクの名前
) : BOOL
```

入力パラメータ

id
(短い形式のみが許されます)
データタイプ: StructTaskId
チェックするタスクID(たとえば、TSI#taskIdなど)

名前
(短い形式のみが許されます)
データタイプ: TaskName
実行システムで定義されたタスク名

戻り値

データタイプ: BOOL

戻り値は、このタスクIDが指定されたタスクと関連付けられているかどうかを示します。

TRUE : タスクIDが指定されたタスクと関連付けられています。

FALSE タスクIDが指定されたタスクと関連付けられていません。

6.3 タスクのランタイム測定用ファンクション

6.3.1 タスクのランタイム測定用ファンクション - 概要

ランタイム測定用のファンクションは、すべてのタスクに対して使用することができます。ただし、IPOsynchronousTaskおよびShutdownTaskでは、測定はサポートされていません。これらのタスクで呼び出された場合は、測定値T#0msが戻されます。

以下のランタイムを測定することができます。

- 最後のSTOP - RUN移行からの、タスクの最大ランタイム(「ファンクション_getMaximalTaskIdRunTime」を参照)
- 最後のSTOP - RUN移行からの、タスクの最小ランタイム(「ファンクション_getMinimalTaskIdRunTime」を参照)
- タスクの直前のパスからのランタイム(「ファンクション_getCurrentTaskIdRunTime」を参照)
- 最後の10個のパスからの平均タスクランタイム(「ファンクション_getAverageTaskIdRunTime」を参照)

これらのファンクションは、SIMOTION Kernel V3.1以降で使用でき、ライブラリで使用することができます。

これらのファンクションで、タスクは一意的タスクIDによって指定されます。

注記

多くの場合、SIMOTION

KernelバージョンV3.0以前でも、同様のファンクションを使用できます。これらのファンクションでは、タスクはその名前で指定されます(実行システムと同様)。

これらのファンクションは、ライブラリで使用してはいけません。

これらのファンクションは、SIMOTION

KernelのバージョンV3.1では使用できません。これらのファンクションの可用性は、SIMOTION Kernelの将来のバージョンでは保証されていません。

下記も参照

ファンクション_getMaximalTaskIdRunTime (ページ 268)

ファンクション_getMinimalTaskIdRunTime (ページ 269)

ファンクション_getCurrentTaskIdRunTime (ページ 271)

ファンクション_getAverageTaskIdRunTime (ページ 272)

タスクの実行時間 (ページ 181)

6.3.2 ファンクション _getMaximalTaskIdRunTime

このファンクションは、SIMOTION Kernel V3.1以降で使用できます。

このファンクションは、高優先度タスクによるすべての割り込みを含めて、最後のSTOP - RUN移行からの、タスクの最大ランタイムを提供します。
タスクは、プロジェクト全体で一意的タスクIDによって指定します(「ファンクション_startTaskId」を参照)。

以下のファンクションは、測定に割り込みません。

- _suspendTaskId
- _disableScheduler(SIMOTIONデバイス一覧に記載されたマニュアルを参照)

測定されるランタイムはサーボサイクルクロックの倍数で、位置制御サイクルクロックより短いランタイムの場合、T#MIN(= T#0ms)が測定値として戻されます。

このファンクションは、すべてのタスクに対して使用することができます。
ただし、IPOsynchronousTaskおよびShutdownTaskでは、測定はサポートされていません。
これらのタスクで呼び出されたときは、測定値T#MIN(= T#0ms)が戻されます。

SIMOTION
KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_getMaximalTaskIdRunTime (
    { id      : StructTaskId
    }
) : TIME
```

入力パラメータ

id (オプション)
データタイプ: StructTaskId
デフォルト: このファンクションが呼び出される現在のタスクのタスクID
ランタイムを測定するタスクのタスクID(ページ6-330を参照)

戻り値

データタイプ:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	測定がサポートされていないか、まだ完了していません
T#MINより大きく、T#MAXより小さい	発生した最大ランタイム
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_getMaximalTaskRunTime (// 短い形式のみが許されます
{
    name: Task_Name // タスクの名前
}
) : TIME
```

このファンクションは、以下の点を除いて、ファンクション_getMaximalTaskIdRunTimeと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用してはいけません。

これらのファンクションは、SIMOTION KernelのバージョンV3.1では使用できません。これらのファンクションの可用性は、SIMOTION Kernelの将来のバージョンでは保証されていません。

下記も参照

ファンクション_startTaskId (ページ 262)

ファンクション_suspendTaskId (ページ 263)

6.3.3 ファンクション_getMinimalTaskIdRunTime

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

このファンクションは、高優先度タスクによるすべての割り込みを含めて、最後のSTOP - RUN移行からの、タスクの最小ランタイムを提供します。

タスクは、プロジェクト全体で一意的タスクIDによって指定します(「ファンクション_startTaskId」を参照)。

以下のファンクションは、測定に割り込みません。

- _suspendTaskId
- _disableScheduler(SIMOTIONデバイス一覧に記載されたマニュアルを参照)

測定されるランタイムはサーボサイクルクロックの倍数で、位置制御サイクルクロックより短いランタイムの場合、T#MIN(= T#0ms)が測定値として戻されます。

このファンクションは、すべてのタスクに対して 사용할 수 있습니다。

ただし、IPOsynchronousTaskおよびShutdownTaskでは、測定はサポートされていません。これらのタスクで呼び出されたときは、測定値T#MIN(= T#0ms)が戻されます。

SIMOTION

KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_getMinimalTaskIdRunTime (
    { id      : StructTaskId
    }
) : TIME
```

入力パラメータ

id (オプション)
 データタイプ: StructTaskId
 デフォルト: このファンクションが呼び出される現在のタスクのタスクID
 ランタイムを測定するタスクのタスクID(「ファンクション_startTaskId」を参照)

戻り値

データタイプ:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	測定がサポートされていないか、まだ完了していません
T#MINより大きく、T#MAXより小さい	発生した最大ランタイム
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_getMinimalTaskRunTime ( // 短い形式のみが許されます
    { name      : Task_Name // タスクの名前
    }
) : TIME
```

このファンクションは、以下の点を除いて、ファンクション_getMinimalTaskIdRunTimeと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用するはいけない。

これらのファンクションは、SIMOTION KernelのバージョンV3.1では使用できません。これらのファンクションの可用性は、SIMOTION Kernelの将来のバージョンでは保証されていません。

下記も参照

ファンクション_startTaskId (ページ 262)

ファンクション_suspendTaskId (ページ 263)

6.3.4 ファンクション_getCurrentTaskIdRunTime

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다.

このファンクションは、高優先度タスクによるすべての割り込みを含めて、タスクの直前のパスからのランタイムを提供します。

タスクは、プロジェクト全体で一意的なタスクIDによって指定します(「ファンクション_startTaskId」を参照)。

以下のファンクションは、測定に割り込みません。

- _suspendTaskId
- _disableScheduler(SIMOTIONデバイス一覧に記載されたマニュアルを参照)

測定されるランタイムはサーボサイクルクロックの倍数で、位置制御サイクルクロックより短いランタイムの場合、T#MIN(= T#0ms)が測定値として戻されます。

このファンクションは、すべてのタスクに対して 사용할 수 있습니다.

ただし、IPOsynchronousTaskおよびShutdownTaskでは、測定はサポートされていません。これらのタスクで呼び出されたときは、測定値T#MIN(= T#0ms)が戻されます。

SIMOTION

Kernel/バージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_getCurrentTaskIdRunTime (
    { id      : StructTaskId
    }
) : TIME
```

入力パラメータ

id	(オプション)
データタイプ:	StructTaskId
デフォルト:	このファンクションが呼び出される現在のタスクのタスクID
	ランタイムを測定するタスクのタスクID(「ファンクション_startTaskId」を参照)

戻り値

データタイプ:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	測定がサポートされていないか、まだ完了していません
T#MINより大きく、T#MAXより小さい	発生した最大ランタイム
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```

_getCurrentTaskRunTime (// 短い形式のみが許されます
    {
        name      : Task_Name // タスクの名前
    }
    ) : TIME

```

このファンクションは、以下の点を除いて、ファンクション_getCurrentTaskIdRunTimeと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用してはいけません。

これらのファンクションは、SIMOTION KernelのバージョンV3.1では使用できません。これらのファンクションの可用性は、SIMOTION Kernelの将来のバージョンでは保証されていません。

下記も参照

ファンクション_startTaskId (ページ 262)

ファンクション_suspendTaskId (ページ 263)

6.3.5 ファンクション_getAverageTaskIdRunTime

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

このファンクションは、高優先度タスクによるすべての割り込みを含めて、最後の10個のサイクルからの、タスクの平均ランタイムを提供します。
タスクは、プロジェクト全体で一意的タスクIDによって指定します(ファンクション_startTaskIdを参照)。

以下のファンクションは、測定に割り込みません。

- _suspendTask
- _disableScheduler(SIMOTIONデバイス一覧に記載されたマニュアルを参照)

測定されるランタイムはサーボサイクルクロックの倍数で、位置制御サイクルクロックより短いランタイムの場合、T#MIN(= T#0ms)が測定値として戻されます。

このファンクションは、すべてのタスクに対して使用することができます。
ただし、IP0synchronousTaskおよびShutdownTaskでは、測定はサポートされていません。
これらのタスクで呼び出されたときは、測定値T#MIN(= T#0ms)が戻されます。

通知

```
_getAverageTaskRunTime (    // 短い形式のみ許されます
    { id      : : StructTaskId
    }
) : TIME
```

入力パラメータ

id	(オプション)
データタイプ	StructTaskId
デフォルト:	ランタイムを測定するタスクのタスクID(実行システムで定義)
	ランタイムを測定するタスクのタスクID(「ファンクション_startTaskId」を参照)

戻り値

データタイプ:	TIME
T#MIN (= T#0ms = T#1ms * UDINT#0)	測定がサポートされていないか、まだ完了していません
T#MINより大きく、T#MAXより小さい	発生した最大ランタイム
T#MAX (= T#49d_17h_2m_47s_295ms = T#1ms * UDINT#16#FFFF_FFFF)	タスクIDが不正です

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_getAverageTaskRunTime (    // 短い形式のみ許されます
    { name      : Task_Name    // タスクの名前
    }
) : TIME
```

このファンクションは、以下の点を除いて、ファンクション_getAverageTaskIdRunTimeと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。

- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION

KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

下記も参照

ファンクション_startTaskId (ページ 262)

ファンクション_suspendTaskId (ページ 263)

6.3.6 タスクの正確なランタイム測定用のファンクション

説明

以下のシステムファンクションを使用して、システム立ち上がりからの時間を μ s精度で測定することができます。

この結果、アプリケーション内部のセクションで、正確に時間を測定し、アプリケーションを最適化することができます。

以下のファンクションを使用することができます。

- ファンクション_getInternalTimeStamp (ページ 274)
- ファンクション_getTimeDifferenceOfInternalTimeStamp (ページ 275)

6.3.7 ファンクション_getInternalTimeStamp

説明

ファンクション_getInternalTimeStampは、特定の内部時間スタンプをUDINTとして提供します。

タスクの開始時と終了時にタイムスタンプを設定し、ファンクションファンクション_getTimeDifferenceOfInternalTimeStamp (ページ 275)で、UDINT t1およびUDINT t2として使用することができます。

```
_getInternalTimeStamp          :UDINT;
```

6.3.8 ファンクション_getTimeDifferenceOfInternalTimeStamp

説明

ファンクション_getTimeDifferenceOfInternalTimeStampは、2つの内部スタンプ間の時間値を提供します。 ファンクション_getInternalTimeStamp (ページ 274)の戻り値UDINTを使用して、2つのタイムスタンプを生成する必要があります。

```
_getTimeOfDifferenceOfInternalTimeStamps  
(  
    UDINT t1,  
    UDINT t2  
) : UDINT;
```

測定する時間の開始時間はt1で、終了時間はt2です。
このファンクションは、その時間の差(t2 - t1)をUDINT(μs単位)として戻します。

用途

このファンクションを使用して、コードセクションのランタイム(たとえば、iposynchronousタスクの通信時間など)を測定することができます。

6.4 メッセージプログラミング用ファンクション(AlarmS)

6.4.1 メッセージプログラミングに関する全般情報

自由にコンフィグレーションされたメッセージ(たとえば、エラーメッセージ)をログオンされた表示デバイスに送信し、そのステータスをチェックすることができます。

特に、以下のことを行うことができます。

- 肯定応答を必要としないメッセージの送信(「ファンクション_alarmSId」を参照)
- 肯定応答を必要とするメッセージの送信(「ファンクション_alarmSqli」を参照)
- メッセージのステータスとその肯定応答のチェック(「ファンクション_alarmScId」を参照)

これらのファンクションは、SIMOTION Kernel V3.1以降で使用でき、ライブラリで 사용할 ことができます。

- すべての保留アラームの一覧表示(「ファンクション_getPendingAlarms」を参照)(V4.1現在)
- アラームを"送信"に設定(「ファンクション_resetAlarmIdおよび_resetAllAlarmId」を参照)(V4.1現在)

これらのファンクションでは、メッセージを一意のアラームIDによって指定します。

注記

多くの場合、SIMOTION

KernelバージョンV3.0以前でも、同様のファンクションを使用できます。これらのファンクションでは、メッセージはその名前で指定されます(SIMOTION SCOUTで設定された名前)。

これらのファンクションは、ライブラリで使用するはいけません。

これらのファンクションは、SIMOTION

KernelのバージョンV3.1では使用できません。これらのファンクションの可用性は、SIMOTION Kernelの将来のバージョンでは保証されていません。

メッセージプログラミングの例は、プログラミングメッセージに含まれています。(ページ 378)

MCCでのファンクションのプログラミングについては、『MCCプログラミングマニュアル』(セクション「受信メッセージと送信メッセージ」を参照してください。

アラームID

設定したメッセージ名に対するアラームIDを、以下の方法で取得することができます。

- _alarm.名前変数。ここで、
 - _alarmは、アラームIDの事前定義された名前空間です(『STプログラミングマニュアル』の「名前空間」を参照)。
名前空間の識別子は、次の識別子とピリオドで区切られます。
 - 名前 は、SIMOTION SCOUTで設定された、メッセージの識別子です。

- ファンクション_getAlarmId(名前)の使用(「ファンクション_getAlarmId」を参照)。
メッセージは、コンフィグレーションされたメッセージ名によって指定します。

通知

_alarm. 名前形式の_ファンクションgetAlarmIdと変数は、ライブラリでは使用してはいけません。

下記も参照

ファンクション_getAlarmId (ページ 285)

ファンクション_alarmSId (ページ 284)

ファンクション_alarmSqlId (ページ 280)

ファンクション_alarmSId (ページ 277)

6.4.2 ファンクション_alarmSId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

トリガ信号(Sig)でレベル変更が発生したとき、肯定応答を必要としないメッセージが生成され、このような信号を受信するためにログオンされたすべての表示デバイスに送信されます。

生成するメッセージは、プロジェクト全体で一意的なアラームIDによって指定します(「アラームID (ページ 379)」を参照)。

オプションで、代替値を付加することができます。

最大40個のメッセージを同時に処理することができます。

SIMOTION

Kernel/バージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_alarmSId (  
    Sig          : BOOL  
    Ev_Id       : StructAlarmId  
    { sd        : ANY_NUM, ANY_BIT  
    }  
    )           : DWORD
```

入力パラメータ

Sig

データタイプ: BOOL

メッセージをトリガする信号は、以下のように評価されます。

信号が立ち上がりエッジ(このアラームIDによる最後の呼び出しに対して)である場合、受信メッセージが生成されます。

このメッセージ名での最初の呼び出しの信号状態がTRUEの場合、受信メッセージも生成されます。

信号が立ち下がりエッジ(このアラームIDによる最後の呼び出しに対して)である場合、送信メッセージが生成されます。

Ev_Id

データタイプ: StructAlarmId

生成するメッセージのアラームID(「アラームID (ページ 379)」を参照)

sd (オプション)

データタイプ: ANY_NUM, ANY_BIT

デフォルト: 0

代替値: 代替値としては、すべての基本データタイプが許されます。

値を入力することはできません。許容されたデータタイプの1つに属する、変数、または、定数の事前定義された識別子だけが許されます。

SIMOTION

SCOUTでのメッセージコンフィグレーション時に代替値を定義した場合は、その代替値を指定する必要があります。

コンパイル時に、メッセージでコンフィグレーションされた代替値のデータタイプのチェックは実行されません。

戻り値

データタイプ: DWORD

戻り値は、ユーザーに、呼び出しの結果を知らせます。

指定された値は、ALARMS_ERROR定数とDSC_SVS_DEVICE_ALARMS_xxx定数のOR演算として表わすことができます。

16#0000

エラーなし

受信メッセージの場合、メッセージリストにエントリが作成されます。

送信メッセージの場合、メッセージリストからエントリが削除済みです。

16#8001

メッセージ名が不正です

ALARMS_ERROR

OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID

16#8002	オーバーフローによるメッセージの損失(メッセージリストに空きスペースがありません)。 メッセージリストの40個のエントリがすべて占有されています。 メッセージリストへの入力が行われません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
16#8003	オーバーフローによるメッセージの損失(信号が未送信か、信号オーバーフロー)。 クライアントへの通知用送信バッファが最後のイベントによってまだ占有されています。 メッセージリストへの入力が行われません。 ファンクション呼び出しが、立ち上がりエッジおよび立ち下がりエッジで、非常に短い時間隔で次々に行われる場合にも、このエラーが発生します。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#8004	メッセージの重複、メッセージは拒否されます(呼び出しで、シーケンスで2回目の受信メッセージまたは送信メッセージが発生)。 メッセージリストへの入力が行われません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_CALL
16#8005	表示ユニットが未登録です(メッセージは、一応、リストに入力されます)。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#8006	メッセージ名が低優先度レベルで処理済みです。 SIMOTIONでは、発生しません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
16#8007	このメッセージ名を持つジョブがまだ起動されていません(最初の呼び出しでSig = FALSE)。 先行する立ち上がりエッジ(受信メッセージ)なしで到着した立ち下がりエッジ(送信メッセージ) メッセージリストへの入力が行われません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
16#8008	メッセージ名が割り当て済みです。 SIMOTIONでは、発生しません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
16#8009	内部エラー ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR

16#8010 入力が拒否されました。メッセージ肯定応答バッファが満杯です。
。
_alarmSqlIdの場合のみ
ALARMS_ERROR OR
DSC SVS_DEVICE ALARMS NO ENTRY

例

メッセージ生成の例 (ページ 381)を参照してください。

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```

_alarmsS      (      // 短い形式のみが許されます
    Sig        : BOOL
    Al_Name     : Alarm_Name      // メッセージの名前
    { sd        : ANY_NUM, ANY_BIT
    }
    ) :      DWORD

```

このファンクションは、以下の点を除いて、ファンクション `alarmSld`と同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION
KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

6.4.3 ファンクション_alarmSqlId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다。

トリガ信号(Sig)でレベル変更が発生したとき、肯定応答を必要とするメッセージが生成され、このような信号を受信するためにログオンされたすべての表示デバイスに送信されます。

生成するメッセージは、プロジェクト全体で一意的アラームIDによって指定します(「アラームID (ページ 379)」を参照)。

オプションで、代替値を付加することができます。

最大40個のメッセージを同時に処理することができます。

SIMOTION
KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_alarmSqId (  
    Sig          : BOOL  
    Ev_Id       : StructAlarmId  
    { sd        : ANY_NUM, ANY_BIT  
    }  
    )           : DWORD
```

入力パラメータ

Sig

データタイプ: BOOL

メッセージをトリガする信号は、以下のように評価されます。

信号が立ち上がりエッジ(このメッセージ名による最後の呼び出しに対して)である場合、受信メッセージが生成されます。

このメッセージ名での最初の呼び出しの信号状態がTRUEの場合、受信メッセージも生成されます。

信号が立ち下がりエッジ(このメッセージ名による最後の呼び出しに対して)である場合、送信メッセージが生成されます。

Ev_Id

データタイプ: StructAlarmId

生成するメッセージのアラームID(「アラームID (ページ 379)」を参照)

sd

(オプション)

データタイプ: ANY_NUM, ANY_BIT

デフォルト: 0

代替値: 代替値としては、すべての基本データタイプが許されます。

値を入力することはできません。許容されたデータタイプの1つに属する、変数、または、定数の事前定義された識別子だけが許されます。

SIMOTION

SCOUTでのメッセージコンフィグレーション時に代替値を定義した場合は、その代替値を指定する必要があります。

コンパイル時に、メッセージでコンフィグレーションされた代替値のデータタイプのチェックは実行されません。

戻り値

データタイプ: DWORD

戻り値は、ユーザーに、呼び出しの結果を知らせます。

指定された値は、ALARMS_ERROR定数とDSC_SVS_DEVICE_ALARMS_xxx定数のOR演算として表わすことができます。

16#0000	エラーなし 受信メッセージの場合、メッセージリストにエントリが作成されます。 送信メッセージの場合、メッセージリストからエントリが削除済みです。
16#8001	メッセージ名が不正です ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
16#8002	オーバーフローによるメッセージの損失(メッセージリストに空きスペースがありません)。 メッセージリストの40個のエントリがすべて占有されています。メッセージリストへの入力が行われません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
16#8003	オーバーフローによるメッセージの損失(信号が未送信か、信号オーバーフロー)。 クライアントへの通知用送信バッファが最後のイベントによってまだ占有されています。 メッセージリストへの入力が行われません。 ファンクション呼び出しが、立ち上がりエッジおよび立ち下がりエッジで、非常に短い時間隔で次々に行われる場合にも、このエラーが発生します。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
16#8004	メッセージの重複、メッセージは拒否されます(呼び出しで、シーケンスで2回目の受信メッセージまたは送信メッセージが発生)。 メッセージリストへの入力が行われません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_CALL
16#8005	表示ユニットが未登録です(メッセージは、一応、リストに入力されます)。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
16#8006	メッセージ名が低優先度レベルで処理済みです。 SIMOTIONでは、発生しません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
16#8007	このメッセージ名を持つジョブがまだ起動されていません(最初の呼び出しでSig = FALSE)。 先行する立ち上がりエッジ(受信メッセージ)なしで到着した立ち下がりエッジ(送信メッセージ) メッセージリストへの入力が行われません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL

16#8008	メッセージ名が割り当て済みです。 SIMOTIONでは、発生しません。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
16#8009	内部エラー ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
16#8010	入力が拒否されました。メッセージ肯定応答バッファが満杯です。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_NO_ENTRY

例

メッセージ生成の例 (ページ 381)「メッセージのプログラミング」を参照してください。

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_alarmSq ( // 短い形式のみが許されます
    Sig      : BOOL
    Al_Name   : Alarm_Name // メッセージの名前
    { Data    : ANY_NUM, ANY_BIT
    }
) : DWORD
```

このファンクションは、以下の点を除いて、ファンクション_alarmSqlIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

6.4.4 ファンクション_alarmScId

このファンクションは、SIMOTION Kernel V3.1以降で 사용할 수 있습니다.

これは、メッセージのステータスと、メッセージの肯定応答ステータスを表示します。
メッセージは、プロジェクト全体で一意的なアラームIDによって指定します(「アラームID (ページ 379)」を参照)。

SIMOTION
KernelバージョンV3.0以前の類似ファンクションについては、セクションの終わりに説明します。

通知

```
_alarmScId (
    Ev_Id      : StructAlarmId
) : DWORD
```

入力パラメータ

Ev_Id

データタイプ: StructAlarmId

生成するメッセージのアラームID(「アラームID (ページ 379)」を参照)

戻り値

データタイプ: DWORD

戻り値は、ユーザーに、呼び出しの結果とメッセージのステータスを知らせます。
下記の定数値と記号定数は、どちらを使用してもかまいません。

最初のエラーチェック、「メッセージのプログラミング (ページ 378)」も参照:

16#8000	エラー用フィルタ ALARMS_ERROR
16#8001	メッセージ名が不正です。 ALARMS_ERROR OR DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID

2番目のエラーステータスチェック、「メッセージのプログラミング (ページ 378)」も参照:

16#0000	送信メッセージ、肯定応答されていません(a)。
16#0001	受信メッセージ、肯定応答されていません(b)。 ALARMS_STATE

16#0010 このメッセージ名のメッセージが存在しません。
 (3つのオプション:
 メッセージがまだトリガされていない(1)、
 メッセージがAlarmS経由でトリガされ、送信されている(2)、
 メッセージが_AlarmSq経由でトリガされ、送信され、表示デバイスで肯定応答されている(3))。

16#0101 受信メッセージ、肯定応答されています(c)。
 ALARMS_STATE OR ALARMS_QSTATE

メッセージa~c
は_alarmSqlまたは_alarmSqの使用を示し、メッセージbは_alarmScIdまたは_alarmScの使用を示します。

例

サブセクション5.7.1のページ5-261の「メッセージのエラー番号とステータスのチェック(戻り値のフィルタリング)」を参照してください。

SIMOTION KernelバージョンV3.0以前の類似ファンクション

```
_alarmSc (        // 短い形式のみが許されます  
                 Al_Name        : Alarm_Name        // メッセージの名前  
                 ) :        DWORD
```

このファンクションは、以下の点を除いて、ファンクション_alarmScIdと同じです。

- タスクがその名前によって指定される(実行システムに表示される名前と同じ)。
- このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができる。
- このファンクションは、ライブラリで使用するはいけない。

このファンクションは、SIMOTION KernelのバージョンV3.1では使用できず、将来のバージョンでもサポートされません。

6.4.5 ファンクション_getAlarmId

このファンクションは、SIMOTION Kernel V3.1以降で使うことができます。

このファンクションは、コンフィグレーションされたアプリケーション固有のメッセージ名から、メッセージのアラームIDを生成します。
このアラームIDは、データタイプStructAlarmIdの変数に割り当て、以下のファンクションの入力パラメータとして使用することができます。

- ファンクション_alarmScId
- ファンクション_alarmSqlId

- ファンクション_alarmScId

通知
このファンクションは、ライブラリで使用するはいけない。 このファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、形式パラメータの指定なし)だけで呼び出すことができる。

通知

```
_getAlarmId (           // 短い形式のみが許されます
    Al_Name      : Alarm_Name // メッセージの名前
) : StructAlarmId
```

入力パラメータ

Al_Name
これは、メッセージをSIMOTION
SCOUTでコンフィグレーションするときに作成されるアプリケーション固有のメッ
ジ名です。

戻り値

データタイプ: StructAlarmId
戻り値には、コンフィグレーションされたメッセージのアラームIDが収納されます。

下記も参照

- ファンクション_alarmSId (ページ 277)
- ファンクション_alarmScId (ページ 284)
- アラームID (ページ 379)
- ファンクション_alarmSqlId (ページ 280)

6.4.6 ファンクション_getPendingAlarms

説明

このファンクションは、アラームリストの最大エントリ数に従って、最大40の保留アラームを戻します。(保留アラーム)。
保留アラームの数は、*numberOfPendingAlarms*に表示されます。
各アラームのタイプは、*_type*に表示されます。
各アラームの状態は、*state*に表示されます。

構文

```
_getPendingAlarms          :StructRetGetPendingAlarms

StructRetGetPendingAlarms  :STRUCT
    numberOfPendingAlarms  :UINT;
    alarm                  :Array[1..40] of StructPendingAlarmState;
END_STRUCT
StructPendingAlarmState    :STRUCT
    Id                     :StructAlarmId;
    _type                   :EnumAlarmIdType [ALARM_S |ALARM_SQ]
    State                   :EnumAlarmIdState [INCOMING|OUTGOING]
END_STRUCT
```

6.4.7 ファンクション_resetAlarmIdおよび_reset_AllAlarmId

説明

ファンクション_resetAlarmIdまたは_resetAllAlarmIdを使用して、1つまたはすべてのアラームを"送信"に設定します。
以前と同様に、SQアラームは、HMIまたはSCOUTで、肯定応答する必要があります。

注記

2番目のステップで、必要な場合、AlarmSQの"送信"と同時"肯定応答"を設定します。
アラームをリセットすると、AlarmSが削除されます。
AlarmSQには、ユーザー自身が肯定応答する必要があります。

単一のアラームを"送信"に設定する

```
_resetAlarmId    // ( _alarmSid +  
extension、エラーなし、アラームが保留中でない、IDが使用できない、内部エラーを参照)  
(  
    Id            :StructAlarmId;  
) : DWORD
```

すべての保留アラームを"送信"に設定する

```
_resetAllAlarmId : DWORD // ( _alarmSid +  
extension、エラーなし、内部エラーを参照)  
(  
    Id            :StructAlarmId;  
) : DWORD
```


一般的な標準ファンクションのプログラミング

7.1 一般的な標準ファンクションのプログラミング - 概要

SIMOTIONは、ソースで呼び出して汎用的な機能を実現する一連の標準ファンクションを提供しています。

注記

以下の標準ファンクションのいくつかは、短い形式(すなわち、すべてのパラメータ値の完全なリスティング、仮引数の指定なし)だけで呼び出すことができます。

- 結果 := ファンクション名 (最初のパラメータ値, 2番目のパラメータ値)

長い形式

- 結果 := ファンクション名 (仮引数1 := 最初のパラメータ値など)

上記のことは、ファンクションごとの説明に記載されています。

一般的なデータタイプ(たとえば、ANY_INT、ANY_BITなど)については、『STプログラミングマニュアル』の表「基本データタイプ、一般的なデータタイプ」を参照してください。

コマンドライブラリの使用

コマンドライブラリは、プロジェクトナビゲータの1つのタブです。

このタブには、使用可能なシステムファンクション、システムファンクションブロック、および演算子が収納されています。

これらの要素を、ドラッグ&ドロップで、コマンドライブラリからSTエディタのウィンドウにコピーすることができます。

SIMOTIONとSIMATICのシステムファンクションの比較

ユーティリティ&アプリケーションCDの2_FAQに、SIMATIC S7とSIMOTIONのシステムファンクションの比較が掲載されています。

7.2 数値標準関クション

7.2.1 数値関クションの特性

すべての数値標準関クションは、1つの入力パラメータを持ちます。

結果は、常に、戻り値に設定されます。

一般数値、対数、および三角関数標準関クションでは、それぞれ、数値標準関クションのグループ(関クション名で指定)とデータタイプが指定されます。

7.2.2 一般数値標準関クション

一般数値標準関クションは、以下のために使用されます。

- 変数の絶対値の計算
- 変数の平方根の計算
- 変数を整数部分に切り捨てる

表 7-1 一般数値標準関クション

関クション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
ABS	in	ANY_NUM	ANY_NUM ¹	絶対値
SQRT	in	ANY_REAL	ANY_REAL ¹	平方根
TRUNC	in	ANY_REAL	ANY_INT	値を整数部分に切捨てる(0方向)

¹ 入力パラメータ *in* のデータタイプ

次の表に、一般数値標準関クションの呼び出し例とその結果を示します。

表 7-2 一般数値標準関クションの呼び出し

呼び出し	結果
Result:=ABS (-5); Result:=ABS (in := -5);	5
Result:=SQRT (81.0); Result:=SQRT (in := 81.0);	9.0
Result:=TRUNC (-3.141 59); Result:=TRUNC (in := 3.141 59);	-3

7.2.3 対数標準関クション

対数標準関クションは、指数または対数の計算を行う関クションです。

表 7-3 対数標準ファクション

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
EXP	in	ANY_REAL	ANY_REAL ¹	e ^x (eファクション)
EXPD	in	ANY_REAL	ANY_REAL ¹	10 ^x
EXPT	ln 1	ANY_REAL	ANY_REAL ²	指数 (『STプログラミングマニュアル』 で数式の演算子 **も参照)
	ln 2	ANY_NUM		
LN	in	ANY_REAL	ANY_REAL ¹	自然対数
LOG	in	ANY_REAL	ANY_REAL ¹	常用対数
¹ 入力パラメータ <i>in</i> のデータタイプ				
² 入力パラメータ <i>ln1</i> のデータタイプ				

次の表に、対数標準ファクションの呼び出し例とその結果を示します。

表 7-4 対数標準ファクションの呼び出し

呼び出し	結果
Result := EXP (4.1); Result := EXP (in := 4.1);	60.3403 ...
Result := EXPD (3.0); Result := EXPD (in := 3.0);	1_000.0
Result := LN (2.718_281); Result := LN (in := 2.718_281);	1.0
Result := LOG (245); Result := LOG (in := 245);	2.389_166

7.2.4 三角関数標準ファクション

次の表にリストされた三角関数標準ファクションは、ラジアン角度の変数を受け取り、計算します。

表 7-5 三角関数標準ファクション

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
ACOS	in	ANY_REAL	ANY_REAL ^{1,2}	アークコサイン(非ラジアン値)
ASIN	in	ANY_REAL	ANY_REAL ^{1,2}	アークサイン(非ラジアン値)
ATAN	in	ANY_REAL	ANY_REAL ^{1,2}	アークタンジェント(非ラジアン値)
COS	in	ANY_REAL ²	ANY_REAL ¹	コサイン(ラジアン入力)
SIN	in	ANY_REAL ²	ANY_REAL ¹	サイン(ラジアン入力)
TAN	in	ANY_REAL ²	ANY_REAL ¹	タンジェント(ラジアン入力)
¹ 入力パラメータ <i>in</i> のデータタイプ ² ラジアン角度				

次の表に、三角関数標準ファクションの呼び出し例とその結果を示します。

表 7-6 三角関数標準ファンクションの呼び出し

呼び出し	結果
<pre>PI:= 3.141592; // PIは変数です! Result := SIN (PI / 6); Result := SIN (in := PI / 6);</pre>	0.5
<pre>Result := ACOS (0.5); Result := ACOS (in := 0.5);</pre>	1.047_197 // PI/3と等しい

7.2.5 ビット列標準ファンクション

各ビット列標準ファンクションは、2つの入力パラメータを持ちます。

- in(データタイプANY_BIT):

ビット列のビットシフト演算操作

- n(データタイプUSINT):
 - ROLおよびRORの場合、回転桁数
 - SHLおよびSHRの場合、シフト桁数

結果は、常に、戻り値に設定されます。

次の表に、ファンクション名と、2つの入力パラメータおよび戻り値のデータタイプを示します。

表 7-7 ビット列標準ファンクション

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
ROL	in n	ANY_BIT USINT	ANY_BIT ¹	パラメータinのビット列は、パラメータnで指定された桁数だけ左に回転します。
ROR	in n	ANY_BIT USINT	ANY_BIT ¹	パラメータinのビット列は、パラメータnで指定された桁数だけ右に回転します。
SHL	in n	ANY_BIT USINT	ANY_BIT ¹	パラメータinのビット列は、パラメータnで指定された桁数だけ左にシフトされ、右側の桁が0に置換されます。
SHR	in n	ANY_BIT USINT	ANY_BIT ¹	パラメータinのビット列は、パラメータnで指定された桁数だけ右にシフトされ、左側の桁が0に置換されます。
¹ 入力パラメータinのデータタイプ ² パラメータinの最下位5ビットだけが評価され、たとえば、SHL (16#FFFF_FFFF,32) = 16#FFFF_FFFFに、SHR (16#FFFF_FFFF,32) = 16#FFFF_FFFFになります。				

注記

入力パラメータnとして数値が使用された場合、可能な最小のデータタイプと見なされます(たとえば、1の場合BOOL、2の場合BYTE)。

次の表に、ビット列標準ファンクションの呼び出し例とその結果を示します。

表 7-8 ビット列標準ファンクションの呼び出し例

呼び出し	結果
Result := ROL (2#1101_0011, 5); // 最初のパラメータは、10進の211に対応します	2#0111_1010 (= 10進の122)
Result := ROR (2#1101_0011, 2); // 最初のパラメータは、10進の211に対応します	2#1111_0100 (= 10進の244)
Result := SHL (2#1101_0011, 3); // 最初のパラメータは、10進の211に対応します	2#1001_1000 (= 10進の152)
Result := SHR (2#1101_0011, 2); // 最初のパラメータは、10進の211に対応します	2#0011_0100 (= 10進の52)
Result := SHL (1, 3); // 最初のパラメータのデータタイプはBOOL	2#0000_0000 (= 10進の0)
Result := SHL (2, 3); // 最初のパラメータのデータタイプはBYTE	2#0001_0000 (= 10進の16)

7.3 ビット列のビットへのアクセス

7.3.1 ファクション_getBit

このファクションは、ビット列変数の指定されたビットの値を提供します。

通知

```
FUNCTION _getBit (// ビット列変数
    in      : ANY_BIT,    // ビットの番号
    n: USINT
) : BOOL
```

入力パラメータ

in	データタイプ	ANY_BIT
	ビット列変数	
n	データタイプ	USINT
	値を出力するビットの番号	
	許容値:	BYTE[0..7], WORD[0..15], DWORD[0..31]

許容されない値を指定すると、ファクションは戻り値FALSEを戻します。

戻り値

データタイプ:	BOOL
ビット値	

例

```
myBit := _getBit (in := myBitString,
                 n := 5);
```

ユーザー変数`myBit`には、ユーザー変数`myBitString`のビット5が収納されます。

ビットアドレス指定(V4.1)

構造アドレス指定の構文を使用して、ビット列変数(BOOLを除く)のビット番号を指定することができます。
ビット番号の指定は、整数として行うか、タイプANY_INT(ビット列長の範囲内)の記号定数によって行うことができます。
このファクションは、コンパイラオプション"拡張言語要素"によって有効になります。

```
FUNCTION f : VOID
VAR CONSTANT
    BIT_7 : INT := 7;
END_VAR
VAR
    dw : DWORD;
    b: BOOL;
END_VAR
b := dw.BIT_7; // ビット番号7へのアクセス
b := dw.3;     // ビット番号3へのアクセス
b := dw.33;    // コンパイルエラー、不十分なビット幅!
END_FUNCTION
```

注記

別のタスクによって割り込まれる可能性がある独立した読み取り、操作、および再書き込み処理の結果としてのI/Oおよびシステム変数に対してビット列アドレス指定を使用すると、アクセスの適合性は保証されません。
この場合、エラーはシステムによって検出されませんが、読み取りアクセス自体は可能です。

7.3.2 ファクション_setBit

このファクションは、指定されたビットを特定のブール値(TRUE/FALSE)に設定した後のビット列変数の値を提供します。

通知

```
FUNCTION _setBit (
    in      : ANY_BIT,    // ビット列変数
    n: USINT,    // ビット番号
    { value  : BOOL      // ビット値
    }
) : ANY_BIT
```

入力パラメータ

in

データタイプ: ANY_BIT
ビット列変数

n

データタイプ: USINT
値を設定するビットの番号
許容値: BYTE[0..7],
WORD[0..15],
DWORD[0..31]

不正な値を指定した場合(追加のメッセージがない)、ビット列変数の値が変更されな
いまま、戻されます。

値

(オプション)
デフォルト: TRUE
値を設定するビットに割り当てる値

戻り値

データタイプ: ANY_BIT
入力パラメータ *in* のデータタイプ
ビットを変更したビット列変数の値

注: ビット列変数の直接の変更では、戻り値をそれに割り当てることができます。

例

```
myBitString := _setBit (in := myBitString,  
                        n := 5,  
                        value := FALSE);
```

myBitString ユーザー変数のビット5が、FALSE(論理値0)に設定されます。

ビットアドレス指定(V4.1)

構造アドレス指定の構文を使用して、ビット列変数(BOOLを除く)のビット番号を指定することができます。
ビット番号の指定は、整数として行うか、タイプANY_INT(ビット列長の範囲内)の記号定数によって行うことができます。
このファクションは、コンパイラオプション"拡張言語要素"によって有効になりません。


```
FUNCTION f : VOID
VAR CONSTANT
    BIT_7 : INT := 7;
END_VAR
VAR
    dw : DWORD;
    b: BOOL;
    b = 1;
END_VAR
b := dw.BIT_7; // ビット番号7へのアクセス
b := dw.3;     // ビット番号3へのアクセス
b := dw.33;    // コンパイルエラー、不十分なビット幅!
END_FUNCTION
```

注記

別のタスクによって割り込まれる可能性がある独立した読み取り、操作、および再書き込み処理の結果としてのI/Oおよびシステム変数に対してビット列アドレス指定を使用すると、アクセスの適合性は保証されません。
この場合、エラーはシステムによって検出されませんが、読み取りアクセス自体は可能です。

7.3.3 ファンクション_toggleBit

このファンクションは、指定されたビットを反転した後のビット列変数の値を提供します。

通知

```
FUNCTION _toggleBit (
    in      : ANY_BIT,    // ビット列変数
    n: USINT,    // ビット番号
    ) : ANY_BIT
```

入カパラメータ

in	データタイプ:	ANY_BIT
	ビット列変数	
n	データタイプ:	USINT
	値を反転するビットの番号(TRUEからFALSEまたはFALSEからTRUEへの設定)	

許容値: BYTE[0..7]、
 WORD[0..15]、
 DWORD[0..31]

不正な値を指定した場合(追加のメッセージがない)、ビット列変数の値が変更されな
いまま、戻されます。

戻り値

データタイプ: ANY_BIT

入力パラメータ *in* のデータタイプ

ビットを反転したビット列変数の値

注: ビット列変数の直接の変更では、戻り値をそれに割り当てることができます。

例

```
myBitString := _toggleBit (in := myBitString, n := 5);
```

myBitString ユーザー変数のビット5が、反転されます。

7.4 数値データタイプに対するビット操作

以下のファクションを使用して、数値データタイプに対してビット演算を行うことができます。戻り値の各ビットは、入力パラメータの対応ビットから生成されます。

表 7-9 数値データタイプに対するビット演算子

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
_NOT ¹	in	ANY_INT	ANY_INT ²	ビットごとの否定
_AND	in1 in2	ANY_INT ³ ANY_INT ³	ANY_INT ⁴	ビットごとの論理積(AND演算): 戻り値のビットは入力パラメータのすべての対応ビットが1の場合だけ1で、その他の場合は0です。
_OR	in1 in2	ANY_INT ³ ANY_INT ³	ANY_INT ⁴	ビットごとの論理和(OR演算): 戻り値のビットは、入力パラメータの対応ビットの少なくとも1つが1の場合だけ1で、その他の場合は0です。
_XOR	n1 in2	ANY_INT ³ ANY_INT ³	ANY_INT ⁴	排他的論理和(exclusive OR演算): 戻り値のビットは、入力パラメータの対応ビットの1つだけが1のとき1で、その他の場合は0です。
¹ _NOTファンクションは、短い形式(すなわち、すべてのパラメータ値の完全なリストイング、形式パラメータの指定なし)だけで呼び出すことができます。 ² 入力パラメータのデータタイプ ³ in1とin2のデータタイプを暗黙的に一般ANY_INTデータタイプに変換することが可能でなければなりません。 ⁴ 入力パラメータを暗黙的に変換できる最小の一般データタイプ				

7.5 文字列処理(V4.0以降)

7.5.1 文字列編集用ファンクション

以下のファンクションを使用して、データタイプSTRINGの変数の編集を行うことができます。

表 7-10 文字列編集用ファンクション

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
CONCAT	in1 in2	STRING[254] STRING[254]	STRING[254]	文字列in2を文字列in1'に結合することができます。 エラーフラグ: TSI#ERRNO := 1(length(IN1)+length(IN2) > 254の場合)
DELETE	In l p	STRING[254] INT INT	STRING[254]	文字列inから、位置p ^{2 4 6} を先頭とするl文字(複数可)を削除します。 エラーフラグ: TSI#ERRNO := 2(L < 0、P < 1、または P > length(IN)の場合)
FIND	in1 in2	STRING[254] STRING[254]	INT	文字列in1で文字列in2が開始される位置を戻します。 文字列in2が文字列in1に含まれない場合、結果は0です。 エラーフラグ: なし
INSERT	in1 in2 p	STRING[254] STRING[254] INT	STRING[254]	文字列in1を、文字列in2の位置p ^{1 2 5 7} に挿入します。 エラーフラグ: TSI#ERRNO := 2(P < 0またはP > length(IN1)の場合) TSI#ERRNO := 1(length(IN1)+length(IN2) > 254の場合)
左	In l	STRING[254] INT	STRING[254]	文字列in ^{2 3} の最初のl文字を戻します。 エラーフラグ: TSI#ERRNO := 2(L < 0の場合)
LEN	in	STRING[254]	STRING[254]	文字列inの文字数を戻します。 エラーフラグ: なし
MID	In l p	STRING[254] INT INT	STRING[254]	文字列inの位置p ^{2 3} を先頭とするl文字を戻します。 エラーフラグ: TSI#ERRNO := 2(L < 0、P < 1、またはP > length(IN)の場合)

	入力パラメータ		戻り値	
REPLACE	in1 in2 l p	STRING[254] STRING[254] INT INT	STRING[254]	文字列in1の位置p ^{2 4 7} を先頭とするl文字を文字列in2に置き換えます。 エラーフラグ: TSI#ERRNO := 2(L < 0、P < 1、またはP > length(IN1)の場合) TSI#ERRNO := 1(length(IN1)+length(IN2)-L > 254の場合)
RIGHT	in l	STRING[254] INT	STRING[254]	文字列in ^{2 3} の最後のl文字を戻します。 エラーフラグ: TSI#ERRNO := 2(L < 0の場合)
¹ LEN (in1) + LEN (in2) > 254: 文字列が切り捨てられます。 ² l < 0またはp < 0: 空の文字列が戻されます。 ³ l = 0またはp = 0: 空の文字列が戻されます。 ⁴ l = 0またはp = 0: inまたはin1の文字列は変更されません。 ⁵ p = 0: 文字列in1が文字列in2に結合されます ⁶ p > LEN(in): 文字列inは変更されません。 ⁷ p > LEN(in1): 文字列in2がin1に結合されます。				

表 7-11 文字列処理用ファンクションの呼び出し例

呼び出し	結果
A := CONCAT (in1 := 'ASTRING', in2 := '123');	'ASTRING123'
A := DELETE (in1 := 'ASTRING', l := 2, p := 4); A := DELETE (in1 := 'ASTRING', l := 2, p := 0); A := DELETE (in1 := 'ASTRING', l := 2, p := 8); A := DELETE (in1 := 'ASTRING', l := 0, p := 4); A := DELETE (in1 := 'ASTRING', l := 10, p := 4); A := DELETE (in1 := 'ASTRING', l := -1, p := 4); A := DELETE (in1 := 'ASTRING', l := 2, p := -1);	'ASTNG', 'ASTRING', 'ASTRING', 'ASTRING', 'AST', '', ''
B := FIND (in1 := 'ASTRING', in2 := 'RI'); B := FIND (in1 := 'ASTRING', in2 := 'RB');	4. 0.
A := INSERT (in1 := 'ASTRING', in2 := '123', p := 1); A := INSERT (in1 := 'ASTRING', in2 := '123', p := 0); A := INSERT (in1 := 'ASTRING', in2 := '123', p := 10); A := INSERT (in1 := 'ASTRING', in2 := '123', p := -1);	'A123STRING', '123ASTRING', 'ASTRING123', ''
A := LEFT (in := 'ASTRING', l := 3); A := LEFT (in := 'ASTRING', l := 10); A := LEFT (in := 'ASTRING', l := -1);	'AST', 'ASTRING', ''
B := LEN (in := 'ASTRING');	7.
A := MID (in := 'ASTRING', l := 3, p := 2); A := MID (in := 'ASTRING', l := 3, p := 6); A := MID (in := 'ASTRING', l := 3, p := 8); A := MID (in := 'ASTRING', l := 3, p := 0);	'STR', 'NG', '', ''

7.5 文字列処理(V4.0以降)

呼び出し	結果
A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 2); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 1); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 0, p := 2); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 0); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 2, p := 10); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := 5); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := 4, p := -1); A := REPLACE (in1 := 'ASTRING', in2 := '123', l := -1, p := 2);	'A123NG', '123ING', 'ASTRING', 'ASTRING', 'ASTRING', 'ASTRING123'.. 'ASTRI123', '', '',
A := RIGHT (in := 'ASTRING', l := 3); A := RIGHT (in := 'ASTRING', l := 10); A := RIGHT (in := 'ASTRING', l := -1);	'ING', 'ASTRING', '',

STRINGの変換ファンクションについては、INT/FLOATおよびSTRINGデータタイプの変換用ファンクション (ページ 311)を参照してください。

7.5.2 文字列編集時エラーの分析

説明

文字列ファンクションで発生したエラーは、タスクごとに別々に、タスクスタート情報(Taskstartinfo)に保管されます。
このため、これはタスクコンテキストで実装され、これ以降、同一のタスク(たとえば、BackgroundTask)で照会することができます。

Variable:TSI#ERRNO : DINT

値0は、エラーがないことを示します。
文字列ファンクションでは、P(文字列内の位置)とL(文字数)のエラーは最大文字列長の違反とは別に異なる値で保管されます。

- 値0: エラーがない
- 値1: 最大文字列長の違反
- 値2 PまたはLの値が範囲外

注記

TSI#ERRNOを使用して、文字列長を確認することはできません(文字列長の確認は、DINT_TO_STRING、UDINT_TO_STRING、REAL_TO_STRING、またはLREAL_TO_STRINGを使用します)。

この場合は、エラーコード0がTSI#ERRNOに設定されます(文字列長オーバー)。

このため、変換の前に、入力文字列が十分な長さであることを確認するか、ユーザープログラムでの長さの設定をチェックする必要があります。

TSI#ERRNO値のリセット:

- エラーは、TSI#ERRNOの内容を明示的にリセットするまでは、Taskstartinfoに存在します。
- TSI#ERRNOは、新しい文字列ファクションでエラーが発生した場合だけ上書きされ、正常に実行された文字列ファクションでは、TSI#ERRNOに存在するエラーIDは上書きされません。
- エラーIDは、タスクが再起動されたときにリセットされます。
- タスクの周期的実行(たとえば、IPOSynchronousTask)では、エラーIDは上書きされません。エラーIDは、ローカル変数の内容と同様に、保持されます。

表 7-12 例

```
VAR
  A: STRING[254];
END_VAR

A := DELETE (in := 'ASTRING', _l := -1, p := 4);
// 結果は空の文字列です

IF (TSI#ERRNO = 2) then // 文字列処理エラー
// L < 0、P < 1、またはP > length(IN)の場合
  A := 'ERROR';
END_IF;
```

7.6 データタイプ変換用の標準ファンクション

7.6.1 数値データタイプおよびビットデータタイプの変換用ファンクション

情報が失われる可能性がある場合(たとえば、値の範囲が小さくなったり、LREALからREALへの変換のように精度が低くなる場合は)、常に、明示的な変換が必要です。

コンパイラは、精度の損失と関連する変換を検出したとき、警告を出力します。

通知

タイプ変換エラーは、プログラム実行時に発生することがあり、この場合は、タスクコンフィグレーションで設定したエラー応答がトリガされます(「障害とイベントの評価」を参照)。

DWORDをREALに変換するときは、特別な注意が必要です。

DWORDのビット列は、REAL値としてのチェックが行われません。

DWORDのビット列は、必ず、IEEEに準拠した正規化浮動小数点数のビットパターンと一致するようにしなければなりません。

これを行うために、ファンクション_finiteおよび_isNaNを使用することができます(「ファンクション_finite」と「ファンクション_isNaN」を参照)。

これを行うために、ファンクション_finiteとファンクション_isNaNを使用することができます。

IEEEビットパターンと一致しない場合は、(たとえば、プログラム内の、または、シンボルブラウザでの監視時の)算術演算でREAL値が最初に使用されたときにすぐに、エラーがトリガされます。

明示的なデータタイプ変換は、次の表にリストされている標準ファンクションを使用して実行します。

- **入力パラメータ**

データタイプ変換用の各ファンクションは、単一の入力パラメータinを持ちます。

- **戻り値**

結果は、常に、ファンクションの戻り値に設定されます。

次の表に、各ファンクションの変換ルールを記載します。

- **名前**

入力パラメータと戻り値のデータタイプは各ファンクションの名前から明らかであるため、次の表には記載しません。

たとえば、ファンクションBOOL_TO_BYTEの場合、入力パラメータのデータタイプはBOOLで、戻り値のデータタイプはBYTEです。

表 7-13 数値データタイプおよびビットデータタイプの変換用ファンクション

ファンクション名	変換ルール	暗黙的な変換可能
BOOL_TO_BYTE	最下位ビットとして受け取り、その他のビットを0に設定します。	可
BOOL_TO_DWORD	最下位ビットとして受け取り、その他のビットを0に設定します。	可

ファンクション名	変換ルール	暗黙的な変換可能
BOOL_TO_WORD	最下位ビットとして受け取り、その他のビットを0に設定します。	可
BOOL_VALUE_TO_DINT	ブール値をDINT値(0または1)として受け取ります。	不可
BOOL_VALUE_TO_INT	ブール値をINT値(0または1)として受け取ります。	不可
BOOL_VALUE_TO_LREAL	ブール値をLREAL値(0.0または1.0)として受け取ります。	不可
BOOL_VALUE_TO_REAL	ブール値をREAL値(0.0または1.0)として受け取ります。	不可
BOOL_VALUE_TO_SINT	ブール値をSINT値(0または1)として受け取ります。	不可
BOOL_VALUE_TO_UDINT	ブール値をUDINT値(0または1)として受け取ります。	不可
BOOL_VALUE_TO_UINT	ブール値をUINT値(0または1)として受け取ります。	不可
BOOL_VALUE_TO_USINT	ブール値をUSINT値(0または1)として受け取ります。	不可
BYTE_TO_BOOL	最下位ビットを受け取ります。	不可
BYTE_TO_DINT	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
BYTE_TO_DWORD	ビット列を最下位バイトとして受け取り、残りに0を設定します。	可
BYTE_TO_INT	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
BYTE_TO_SINT	ビット列をSINT値として受け取ります。	不可
BYTE_TO_UDINT	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
BYTE_TO_UINT	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
BYTE_TO_USINT	ビット列をUSINT値として受け取ります。	不可
BYTE_TO_WORD	ビット列を最下位バイトとして受け取り、残りに0を設定します。	可
BYTE_VALUE_TO_LREAL	ビット列をUSINT値として評価し、この値を受け取ります。	不可
BYTE_VALUE_TO_REAL	ビット列をUSINT値として評価し、この値を受け取ります。	不可
DINT_TO_BYTE	最下位バイトをビット列として受け取ります。	不可
DINT_TO_DWORD	ビット列を受け取ります。	不可
DINT_TO_INT	最下位2バイトをINT値として受け取ります。	不可
DINT_TO_LREAL	値を受け取ります。	可
DINT_TO_REAL	値を受け取ります(精度が失われる場合があります)。	不可
DINT_TO_SINT	最下位バイトをSINT値として受け取ります。	不可
DINT_TO_UDINT	値をビット列として受け取ります。	不可
DINT_TO_UINT	最下位2バイトをUINT値として受け取ります。	不可
DINT_TO_USINT	最下位バイトをUSINT値として受け取ります。	不可
DINT_TO_WORD	最下位2バイトをビット列として受け取ります。	不可
DINT_VALUE_TO_BOOL	DINT値が0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
DWORD_TO_BOOL	最下位ビットを受け取ります。	不可
DWORD_TO_BYTE	最下位バイトをビット列として受け取ります。	不可
DWORD_TO_DINT	ビット列をDINT値として受け取ります。	不可
DWORD_TO_INT	最下位2バイトをINT値として受け取ります。	不可

ファンクション名	変換ルール	暗黙的な変換可能
DWORD_TO_REAL	ビット列をREAL値として受け取ります (REAL数の妥当性チェックは実行されません)。 重要: ページ6-300の「注意」	不可
DWORD_TO_SINT	最下位バイトをSINT値として受け取ります。	不可
DWORD_TO_UDINT	ビット列をUDINT値として受け取ります。	不可
DWORD_TO_UINT	最下位2バイトをUINT値として受け取ります。	不可
DWORD_TO_USINT	最下位バイトをUSINT値として受け取ります。	不可
DWORD_TO_WORD	最下位2バイトをビット列として受け取ります。	不可
DWORD_VALUE_TO_LREAL	ビット列をUDINT値として評価し、この値を受け取ります。	不可
DWORD_VALUE_TO_REAL	ビット列をUDINT値として評価し、この値を受け取ります(精度が失われることがあります)。	不可
INT_TO_BYTE	最下位バイトをビット列として受け取ります。	不可
INT_TO_DWORD	ビット列を最下位ワード(2バイト)として受け取り、残りを0に設定します。	不可
INT_TO_DINT	値を受け取ります。	可
INT_TO_LREAL	値を受け取ります。	可
INT_TO_REAL	値を受け取ります。	可
INT_TO_SINT	最下位バイトをSINT値として受け取ります。	不可
INT_TO_USINT	最下位バイトをUSINT値として受け取ります。	不可
INT_TO_UDINT	ビット列を最下位ワード(2バイト)として受け取り、残りを0に設定します。	不可
INT_TO_UINT	ビット列をUINT値として受け取ります。	不可
INT_TO_WORD	ビット列を受け取ります。	不可
INT_VALUE_TO_BOOL	INT値が0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
LREAL_TO_DINT	次の整数1まで四捨五入を行います。	不可
LREAL_TO_INT	次の整数1まで四捨五入を行います。	不可
LREAL_TO_REAL	値を受け取ります(精度が失われる場合があります)。	不可
LREAL_TO_SINT	次の整数1まで四捨五入を行います。	不可
LREAL_TO_UDINT	値を次の整数1まで四捨五入します。	不可
LREAL_TO_UINT	値を次の整数1まで四捨五入します。	不可
LREAL_TO_USINT	値を次の整数1まで四捨五入します。	不可
LREAL_VALUE_TO_BOOL	LREAL値が0.0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
LREAL_VALUE_TO_BYTE	次の整数1まで四捨五入を行い、値をビット列として受け取ります。	不可
LREAL_VALUE_TO_DWORD	次の整数1まで四捨五入を行い、値をビット列として受け取ります。	不可
LREAL_VALUE_TO_WORD	次の整数1まで四捨五入を行い、値をビット列として受け取ります。	不可
REAL_TO_DINT	次の整数1まで四捨五入を行います。	不可
REAL_TO_DWORD	ビット列を受け取ります。	不可
REAL_TO_INT	次の整数1まで四捨五入を行います。	不可
REAL_TO_LREAL	値を受け取ります。	可
REAL_TO_SINT	次の整数1まで四捨五入を行います。	不可

ファンクション名	変換ルール	暗黙的な変換可能
REAL_TO_UDINT	値を次の整数 ¹ まで四捨五入します。	不可
REAL_TO_UINT	値を次の整数 ¹ まで四捨五入します。	不可
REAL_TO_USINT	値を次の整数 ¹ まで四捨五入します。	不可
REAL_VALUE_TO_BOOL	REAL値が0.0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
REAL_VALUE_TO_BYTE	次の整数 ¹ まで四捨五入を行い、値をビット列として受け取ります。	不可
REAL_VALUE_TO_DWORD	次の整数 ¹ まで四捨五入を行い、値をビット列として受け取ります。	不可
REAL_VALUE_TO_WORD	次の整数 ¹ まで四捨五入を行い、値をビット列として受け取ります。	不可
SINT_TO_BYTE	ビット列を受け取ります。	不可
SINT_TO_DINT	値を受け取ります。	可
SINT_TO_DWORD	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
SINT_TO_INT	値を受け取ります。	可
SINT_TO_LREAL	値を受け取ります。	不可
SINT_TO_REAL	値を受け取ります。	不可
SINT_TO_UDINT	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
SINT_TO_UINT	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
SINT_TO_USINT	ビット列を受け取ります。	不可
SINT_TO_WORD	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
SINT_VALUE_TO_BOOL	SINT値が0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
StructAlarmId_TO_DINT	ビット列を受け取ります。	不可
UDINT_TO_BYTE	最下位バイトをビット列として受け取ります。	不可
UDINT_TO_DINT	ビット列を受け取ります。	不可
UDINT_TO_DWORD	ビット列を受け取ります。	不可
UDINT_TO_INT	最下位2バイトをINT値として受け取ります。	不可
UDINT_TO_LREAL	値を受け取ります。	可
UDINT_TO_REAL	値を受け取ります(精度が失われる場合があります)。	不可
UDINT_TO_SINT	最下位バイトをSINT値として受け取ります。	不可
UDINT_TO_UINT	最下位2バイトをUINT値として受け取ります。	不可
UDINT_TO_USINT	最下位バイトをUSINT値として受け取ります。	不可
UDINT_TO_WORD	最下位2バイトをビット列として受け取ります。	不可
UDINT_VALUE_TO_BOOL	UDINT値が0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
UINT_TO_BYTE	最下位バイトをビット列として受け取ります。	不可
UINT_TO_DINT	値を受け取ります。	可
UINT_TO_DWORD	ビット列を最下位ワード(2バイト)として受け取り、残りを0に設定します。	不可
UINT_TO_INT	ビット列を受け取ります。	不可
UINT_TO_LREAL	値を受け取ります。	不可

ファクション名	変換ルール	暗黙的な変換可能
UINT_TO_REAL	値を受け取ります。	可
UINT_TO_SINT	最下位バイトをSINT値として受け取ります。	不可
UINT_TO_UDINT	値を受け取ります。	可
UINT_TO_USINT	最下位バイトをUSINT値として受け取ります。	不可
UINT_TO_WORD	ビット列を受け取ります。	不可
UINT_VALUE_TO_BOOL	UINT値が0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
USINT_TO_BYTE	ビット列を受け取ります。	不可
USINT_TO_DINT	値を受け取ります。	不可
USINT_TO_DWORD	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
USINT_TO_INT	値を受け取ります。	可
USINT_TO_LREAL	値を受け取ります。	不可
USINT_TO_REAL	値を受け取ります。	不可
USINT_TO_SINT	ビット列をSINT値として受け取ります。	不可
USINT_TO_UDINT	値を受け取ります。	可
USINT_TO_UINT	値を受け取ります。	可
USINT_TO_WORD	ビット列を最下位バイトとして受け取り、残りに0を設定します。	不可
USINT_VALUE_TO_BOOL	USINT値が0の場合FALSE(0)、それ以外の場合TRUE(1)	不可
WORD_TO_BOOL	最下位ビットを受け取ります。	不可
WORD_TO_BYTE	最下位バイトをビット列として受け取ります。	不可
WORD_TO_DINT	ビット列を最下位ワード(2バイト)として受け取り、残りを0に設定します。	不可
WORD_TO_DWORD	ビット列を最下位ワード(2バイト)として受け取り、残りを0に設定します。	可
WORD_TO_INT	ビット列をINT値として受け取ります。	不可
WORD_TO_SINT	最下位バイトをSINT値として受け取ります。	不可
WORD_TO_UDINT	ビット列を最下位ワード(2バイト)として受け取り、残りを0に設定します。	不可
WORD_TO_UINT	ビット列を受け取ります。	不可
WORD_TO_USINT	最下位バイトをUSINT値として受け取ります。	不可
WORD_VALUE_TO_LREAL	ビット列をUINT値として評価し、この値を受け取ります。	不可
WORD_VALUE_TO_REAL	ビット列をUINT値として評価し、この値を受け取ります。	不可

¹ 次の2つの整数までの距離が等しい場合: 次の偶数に丸めます

下記も参照

障害およびイベントの評価 (ページ 93)

ファクション_finite (ページ 323)

ファクション_isNaN (ページ 324)

7.6.2 日付および時刻データタイプの変換用ファンクション

日付および時刻データタイプでは、以下の標準ファンクションを使用することができます。

データタイプ日付および時刻を含む数学的式に関しては、『STプログラミングマニュアル』の「数学的式」を参照してください。

表 7-14 日付および時刻用の標準ファンクション

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
CONCAT_DATE_TOD	in1 in2	DATE TIME_OF_DAY	DATE_AND_TIME	日付と時刻をTIME_OF_DAY to DATE_AND_TIME(DT)に結合します。
DATE_AND_TIME_TO_TIME_OF_DAY または DT_TO_TOD	in	DATE_AND_TIME	TIME_OF_DAY	時刻を受け取ります。
DATE_AND_TIME_TO_DATE または DT_TO_DATE	in	DATE_AND_TIME	DATE	日付を受け取ります。
INT_TO_TIME	in	INT	TIME	値を時間として受け取ります(ms単位)。
REAL_TO_TIME	in	REAL	TIME	符号なしの自然数成分にまで四捨五入し、その値を時間(ms単位)を示すものとして受け取ります。
TIME_TO_INT	in	TIME	INT	時間(ms単位)を値として受け取ります。
TIME_TO_REAL	in	TIME	REAL	時間(ms単位)を値として受け取ります。
TIME_TO_UDINT	in	TIME	UDINT	時間(ms単位)を値として受け取ります。
UDINT_TO_TIME	in	UDINT	TIME	値を時間として受け取ります(ms単位)。

7.6.3 枚举データタイプの変換用ファンクション

ファンクションENUM_TO_DINTを使用して、枚举データタイプ要素の数値を得ることができます。

ファンクション呼び出し時に、データタイプの識別子と文字#を枚举要素の識別子の前に配置して、枚举要素のデータタイプを指定します(enum_type#enum_value)。

表 7-15 日付および時刻用の標準ファンクション

ファンクション名	入力パラメータ		戻り値	説明
	名前	データタイプ	データタイプ	
ENUM_TO_DINT	in	任意の枚举データタイプ	DINT	枚举要素の数値を戻します。

7.6.4 BYTEとSTRING間の変換

説明

BYTEからSTRING、STRINGからBYTEへの暗黙的変換では、1バイトが1文字として書き込まれるが、1文字から1バイトが読み取られます(ASCIIフォーマット、文字あたり1バイト)。

注記

文字列は、ARRAY[1...文字列サイズ]として評価されます。

BYTEからSTRINGへの変換

変換は、文字列要素nへのバイトの直接の割り当てによって行われます。

```
myString[n] := myByte;
```

変換ルール:

1. $n > \text{len}(\text{myString})$ でかつ $n < \text{maxlen}(\text{myString})$ の場合、文字列の長さが調整されます。
 $\text{myString}[\text{len}(\text{myString})] \sim \text{myString}[n]$ の間のすべての文字には値"0"が割り当てられます。
2. $n > \text{maxlen}(\text{myString})$ の場合、TSI#ERRNOが値2に設定され(妥当な範囲外の値)、myByteに値0が割り当てられます。

STRINGからBYTEへの変換

変換は、バイトへの文字列要素kの直接の割り当てによって行われます。

```
myByte := myString[k];
```

規則

1. $k > \text{len}(\text{myString})$ の場合、TSI#ERRNOが値2に設定され(妥当な範囲外の値)、myByteに値0が割り当てられます。

アプリケーション

- 内部変数を読み取り、必要に応じて、INTからSTRINGまたはDINTからSTRINGへ変換する。
- HMI(たとえば、WIN CC Op)へのSTRINGの直接の出力
- STRINGからバイトARRAYへの変換と、その結果のHMIへの出力

7.6.5 INT/FLOATおよびSTRINGデータタイプの変換用ファクション

説明

以下のファクションは、INTおよびFLOATデータタイプの数の表示で、数値を文字列に変換するために使用されます。

明示的なデータタイプ変換は、次の表にリストされている標準ファクションを使用して実行します。

- 入力パラメータ

データタイプ変換用の各ファクションは、単一の入力パラメータinを持ちます。

- 戻り値

結果は、常に、ファクションの戻り値に設定されます。

DINT/UDINT/REAL/LREALからSTRINGへの変換ルール

- 値は、10進数または実数として、STRINGに左寄せで書き込まれます。
- 符号が存在する場合、数字の前に書き込まれます。
- 文字列長が不足している場合は、右側の数字が切り捨てられ、TSI#ERRNOが値1に設定されます(文字列長の違反)。

STRINGからDINT/UDINT/REAL/LREALへの変換ルール

1. 先行空白は考慮されません。ブランクとタブは空白と見なされます。
2. 変換は、文字列が終了するか、数字でない文字が現れると、停止します。
3. 文字列に妥当な数が収納されていないか、値の範囲の違反が発生すると、TSI#ERRNOが値3に設定され(不正な数表示)、0.0(REAL/LREAL)が出力されます。

ファクション	例	説明
DINT_TO_STRING	myString:=DINT_TO_STRING(myDint)	ルールを順守してください
UDINT_TO_STRING	myString:=UDINT_TO_STRING(myUDint,myUsint))	ルールを順守してください
STRING_TO_DINT	myDint:=STRING_TO_DINT(myString)	妥当な数の形式は、 [空白[符号]数字]です
STRING_TO_UDINT	myDint:=STRING_TO_UDINT(myString)	妥当な数の形式は、 [空白[+][数字]です • 先行ゼロは、無視されます。 • STRINGへの変換では、数表示は10進です。 STRINGからの変換では、数は10進数でなければなりません。 8進数と16進数の表記はサポートされていません。
REAL_TO_STRING LREAL_TO_STRING	myString:=REAL_TO_STRING(myReal) myString:=LREAL_TO_STRING(myLReal)	ルールを順守してください

ファンクション	例	説明
STRING_TO_REAL STRING_TO_LREAL	myReal:=STRING_TO_REAL(myString) myLReal:=STRING_TO_LREAL(myString)	妥当な数の形式は、 [空白][符号][数字][数字][{ e E }[符号]数字]です。

用途

- HMIは、ファイルシステム(レシピメモリ)からテキストを取得し、一定の手順を経て、テキスト用テキストをSIMOTIONの実行システム(ユニット変数)にロードします。
- データは、_saveUnitDataSetまたは_exportUnitDataSetによって、実行システムに保存されます。
現在のSIMOTIONデータでは、STRINGが拡張されています(たとえば、現在の位置など)。
- テキストは、シリアルインターフェース(ET200など)経由で出力されます。

7.7 任意のデータタイプとバイト配列との間の変換

7.7.1 一般

以下のファクションを使用して、任意のデータタイプ(基本データタイプ、テクノロジーパッケージおよびデバイスの標準データタイプ、およびユーザー定義データタイプ)の変数をバイト配列に変換することができます(その逆も可)。

詳細(たとえば、バイト配列の配置、応用例など)は、任意のデータタイプとバイト配列との間の変換(マーシャリング) (ページ 391)を参照してください。

これらのファクションは、通常、各種デバイス間のデータ交換用定義済み伝送フォーマットを作成するために使用されます(「通信ファクション (ページ 395)」も参照)。

7.7.2 ファクションAnyType_to_BigByteArray、ファクションAnyType_to_LittleByteArray

これらのファクションは、任意のデータタイプ(基本データタイプ、テクノロジーパッケージおよびデバイスの標準データタイプ、およびユーザー定義データタイプ)の変数をバイト配列に変換します。

- AnyType_to_BigByteArrayの場合:
Big Endianタイプのバイト配列(低メモリアドレスに最上位バイト)
- AnyType_to_LittleByteArrayの場合:
Little Endianタイプのバイト配列(低メモリアドレスに最下位バイト)

配列の最初の要素の配列添字は、オプションコンスタントオフセット(デフォルトは0)です。それは、配列制限範囲内の値でなければなりません。

Big Endian、Little Endian、応用例、注記、および移植不能な変換可能データタイプについて

STソースファイルのコンパイル時に、オフセットが配列制限範囲内かどうか、また、変数をバイト配列(オフセットと配列の上限値の間)で完全に表示できるかどうかのチェックが行われます。

変換する変数によってカバーされているバイト配列要素だけに値を設定することができます。バイト配列の他の要素は不変のままです。

注記

これらのファクションは1つのタスクのみで呼び出し、処理するか、複数のタスクを使用する場合、ファクションの呼び出し、処理の観点から、これらのタスクを同期化する適切なメソッドを実装する必要があります(たとえば、_testAndSetSemaphore、_releaseSemaphoreなど)。

呼び出しと結果の処理で異なるタスクが使用される場合、未定義の値が生成される場合があります。

通知
データタイプBOOLの各変数(構造化データタイプ内の要素である変数を含めて)は、バイト配列で、1バイトを占めます。

通知

```
FUNCTION AnyType_to_BigByteArray (  
    anydata : ANY, // 任意のデータタイプ  
    { offset : DINT // 定数のみが許されます  
    }  
    ) : ARRAY [..] OF BYTE // Big Endian  
FUNCTION AnyType_to_LittleByteArray (  
    anydata : ANY,          // 任意のデータタイプ  
    { offset : DINT          // 定数のみが許されます  
    }  
    ) : ARRAY [..] OF BYTE // Little Endian
```

入力パラメータ

anydata
データタイプ: ANY
任意のデータタイプの変数
以下のデータタイプが許されます:
• テクノロジーオブジェクト

offset
(オプション)
データタイプ: DINT
デフォルト 0
定数、配列に割り当てる最初の要素を指定します。

戻り値

データタイプ: ARRAY [..] OF BYTE

- AnyType_to_BigByteArrayの場合:
Big Endianでの配置(低メモリアドレスに最上位バイト)
- AnyType_to_LittleByteArrayの場合:
Little Endianでの配置(低メモリアドレスに最下位バイト)

下記も参照

任意のデータタイプとバイト配列との間の変換(マーシャリング) (ページ 391)

7.7.3 ファクションBigByteArray_to_AnyType、ファクションLittleByteArray_to_AnyType

これらのファクションは、バイト配列を、任意のデータタイプ(基本データタイプ、システムデータタイプ、およびユーザー定義データタイプ)の変数に変換します。

- BigByteArray_to_AnyTypeの場合
Big Endianタイプのバイト配列(低メモリアドレスに最上位バイト)
- LittleByteArray_to_AnyTypeの場合
Little Endianタイプのバイト配列(低メモリアドレスに最下位バイト)

配列で評価する最初の要素の配列添字は、オプションコンスタントオフセット(デフォルトは0)です。それは、配列制限範囲内の値でなければなりません。

Big Endian、Little Endian、応用例、注記、および移植不能な変換可能データタイプについて

STソースファイルのコンパイル時に、オフセットが配列制限範囲内かどうか、また、バイト配列(オフセットと配列の上限値の間)が完全に変数をカバーするかどうかのチェックが行われます。

注記

これらのファクションは1つのタスクのみで呼び出し、処理するか、複数のタスクを使用する場合、ファクションの呼び出し、処理の観点から、これらのタスクを同期化する適切なメソッドを実装する必要があります(たとえば、_testAndSetSemaphore、_releaseSemaphoreなど)。呼び出しと結果の処理で異なるタスクが使用される場合、未定義の値が生成される場合があります。

通知

データタイプBOOLの各変数(構造化データタイプ内の要素である変数を含めて)には、バイト配列の1バイトが割り当てられます。

通知

```
FUNCTION BigByteArray_to_AnyType (  
    byteArray : ARRAY [..] OF BYTE, // Big Endian  
    { offset : DINT          // 定数のみが許されます  
    }  
    ) : ANY  
FUNCTION LittleByteArray_to_AnyType (  
    byteArray : ARRAY [..] OF BYTE, // Little Endian  
    { offset : DINT          // 定数のみが許されます  
    }  
    ) : ANY
```

```
byteArray      : ARRAY [...] OF BYTE, // Little Endian
{ offset      : DINT // 定数のみが許されます
}
) : ANY
```

入力パラメータ

byteArray

データタイプ: ARRAY [...] OF BYTE

- BigByteArray_to_AnyTypeの場合
Big Endianでの配置(低メモリアドレスに最上位バイト)
- LittleByteArray_to_AnyTypeの場合
Little Endianでの配置(低メモリアドレスに最下位バイト)

offset

(オプション)

データタイプ: DINT

デフォルト 0

定数、配列に割り当てる最初の要素を指定します。

戻り値

データタイプ: ANY

任意のデータタイプ。以下のデータタイプは許されません:

- テクノロジーオブジェクト

通知

マーシャリングファクションはプログラム実行時にエラーになることがあり、この場合は、タスクコンフィグレーションで設定されたエラー応答がトリガされます(「**プログラムでの実行エラー**」を参照)。

バイト配列を、一般的なANY_REALデータタイプまたはこのデータタイプを含む構造に変換する場合、注意が必要です。

バイト配列のビット列は、ANY_REAL値としてのチェックが行われません。

バイト配列のビット列は、必ず、IEEEに準拠した正規化浮動小数点数のビットパターンと一致するようにしなければなりません。

これを行うために、ファクション_finiteおよび_isNaNを使用することができます。

これを行うために、ファクション_finiteとファクション_isNaNを使用することができます。

IEEEビットパターンと一致しない場合は、(たとえば、プログラム内の、または、シンボルブラウザでの監視時の)算術演算でANY_REAL値が最初に使用されたときにすぐに、エラーがトリガされます。

下記も参照

プログラムでの実行エラー (ページ 94)

関数 `_finite` (ページ 323)

関数 `_isnan` (ページ 324)

任意のデータタイプとバイト配列との間の変換(マーシャリング) (ページ 391)

7.8 ビット列データタイプの結合

7.8.1 ビット列データタイプの結合に関する全般的な情報

以下の関クションを使用して、複数のビット列データタイプの変数を結合して、高度なレベルのデータタイプの変数を設定することができます。

逆関クションも関クションとして実装されています(「ビット列データタイプの分離 (ページ 426)」を参照)。

7.8.2 関クション_BYTE_FROM_8BOOL

この関クションは、データタイプBOOLの8つの変数を、データタイプBYTEの1つの変数に結合します。

通知

```
FUNCTION _BYTE_FROM_8BOOL (
    { bit0,          // 最下位ビット
      bit1, bit2, bit3, bit4, bit5, bit6,
      bit7: BOOL      // 最上位ビット
    }
) : BYTE
```

入力パラメータ

bit0 (オプション)
...
bit7 (オプション)
データタイプ: BOOL
デフォルト FALSE
データタイプBYTEの1つの変数に結合する、最大8つのデータタイプBOOLの変数
bit0: 最下位ビット
...
ビット7: 最上位ビット

戻り値

データタイプ: BYTE
入力パラメータの結合によって生成されるバイト

7.8.3 ファンクション_WORD_FROM_2BYTE

このファンクションは、データタイプBYTEの2つの変数を、データタイプWORDの1つの変数に結合します。

通知

```
FUNCTION _WORD_FROM_2BYTE (  
    { byte0,          // 下位バイト  
      byte1: BYTE     // 上位バイト  
    }  
    ) : WORD
```

入力パラメータ

byte0 (オプション)
byte1 (オプション)
データタイプ: BYTE
デフォルト BYTE#0
データタイプWORDの1つの変数に結合する、最大2つのデータタイプBYTEの変数
byte0: 下位バイト
バイト1: 上位バイト

戻り値

データタイプ: WORD
入力パラメータの結合によって生成されるワード

7.8.4 ファンクション_DWORD_FROM_2WORD

このファンクションは、データタイプWORDの2つの変数を、データタイプDWORDの1つの変数に結合します。

通知

```
FUNCTION _DWORD_FROM_2WORD (  
    { word0,          // 下位ワード  
      word1: WORD     // 上位ワード  
    }  
    ) : DWORD
```

入力パラメータ

word0 (オプション)
word1 (オプション)
データタイプ: WORD
デフォルト WORD#0
データタイプDWORDの1つの変数に結合する、最大2つのデータタイプWORDの変数
word0: 下位ワード
ワード1: 上位ワード

戻り値

データタイプ: DWORD
入力パラメータの結合によって生成されるダブルワード

7.8.5 ファンクション_DWORD_FROM_4BYTE

このファンクションは、データタイプBYTEの4つの変数を、データタイプDWORDの1つの変数に結合します。

通知

```
FUNCTION _DWORD_FROM_4BYTE (  
    { byte0          // 最下位バイト  
      byte1, byte2,  
      byte3: BYTE    // 最上位バイト  
    }  
    ) : DWORD
```


入力パラメータ

byte0 (オプション)
byte1 (オプション)
byte2 (オプション)
byte3 (オプション)

データタイプ: BYTE

デフォルト BYTE#0

データタイプDWORDの1つの変数に結合する、最大4つのデータタイプBYTEの変数

byte0: 最下位バイト

...

バイト3: 最上位バイト

戻り値

データタイプ: DWORD

入力パラメータの結合によって生成されるダブルワード

7.9 テクノロジーオブジェクトデータタイプの変換

7.9.1 ファンクションAnyObject_to_Object

このファンクションは、階層的なTOデータタイプ(driveAxis、posAxis、followingAxis)または一般ANYOBJECTタイプの変数を、互換TOデータタイプに変換します。

追加情報と例は、「TOデータタイプの変換」に記載されています。

通知

```
FUNCTION AnyObject_to_Object (  
    in      : ANYOBJECT  
    ) : ANYOBJECT
```

入力パラメータ

in
データタイプ: ANYOBJECT
TOデータタイプ(またはANYOBJECT)の変数、または、TOインスタンス

戻り値

データタイプ: ANYOBJECT
タイプ変換ができない場合、値はTO#NILです

下記も参照

TOデータタイプの変換 (ページ 81)

7.10 浮動小数点数の検証用ファクション

7.10.1 ファクション_finite

このファクションは、入力パラメータが、IEEE 754に準拠する無限ビットパターンに合致するかどうかをチェックします。

このファクションは、特に、ファクション_isNaNと組み合わせて、浮動小数点数に変換されたビット列が、IEEEに準拠した正規化浮動小数点数のビットパターンと一致するかどうかをチェックするために使用されます。

これは、(たとえば、プログラム内の、または、シンボルブラウザでの監視時の)算術演算で不正な浮動小数点数が最初に使用されたときにすぐに行われる、タスクコンフィギュレーション時に指定されたエラー応答のトリガを阻止します(「プログラムでの実行エラー」を参照)。

通知

```
_finite (  
    in : ANY_REAL  
) : BOOL
```

入力パラメータ

in
データタイプ: ANY_REAL
チェックする変数

戻り値

データタイプ:	BOOL
FALSE	IEEE 754に準拠した無限ビットパターン
TRUE	IEEE 754に準拠した無限ビットパターンなし(すなわち、値範囲内の正しい浮動小数点数か、不正なビットパターンNaN(Not a Number))

例

```

var_real := DWORD_TO_REAL (var_dword);
IF NOT _finite (var_real) OR _isNaN (var_real) THEN
    ; // エラー処理
ELSE
    var_real := SQRT (var_real);
END_IF;

```

下記も参照

関クション_isNaN (ページ 324)

プログラムでの実行エラー (ページ 94)

7.10.2 関クション_isNaN

この関クションは、入力パラメータが、IEEE 754に準拠した浮動小数点数の不正なビットパターン(NaN: Not a Number)に一致するかどうかを検証します。

この関クションは、特に、関クション_*finite*と組み合わせて、浮動小数点数に変換されたビット列が、IEEEに準拠した正規化浮動小数点数のビットパターンと一致するかどうかをチェックするために使用されます。

これは、(たとえば、プログラム内の、または、シンボルブラウザでの監視時の)算術演算で不正な浮動小数点数が最初に使用されたときにすぐに行われる、タスクコンフィギュレーション時に指定されたエラー応答のトリガを阻止します(「**プログラムでの実行エラー**」を参照)。

通知

```

_isNaN (
    in : ANY_REAL
) : BOOL

```

入力パラメータ

in
データタイプ: ANY_REAL
チェックする変数

戻り値

データタイプ:	BOOL
FALSE	正しいビットパターン、または、IEEE 754に準拠した無限ビットパターン
TRUE	IEEE 754に準拠した不正なビットパターン(NaN: Not a Number)

例

```
var_real := DWORD_TO_REAL (var_dword);  
IF NOT _finite (var_real) OR _isNaN (var_real) THEN  
    ; // エラー処理  
ELSE  
    var_real := SQRT (var_real);  
END_IF;
```

下記も参照

関シヨン_finite (ページ 323)
プログラムでの実行エラー (ページ 94)

7.11 選択用関クション

7.11.1 関クションSEL

バイナリ選択

戻り値は、入力パラメータgの値に応じて、入力in0、in1のどちらかです。

通知

```
FUNCTION SEL (  
    g: BOOL,  
    in0    : ANY,  
    in1    : ANY  
    ) : ANY
```

入力パラメータ

g

データタイプ: BOOL

入力パラメータin0またはin1の選択

in0、in1

データタイプ: ANY

入力パラメータin0とin1は同一のデータタイプであるか、暗黙的変換によって1つの共通のデータタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

戻り値

データタイプ: ANY

選択された入力パラメータ

in0 g = 0の場合(FALSE)

in1 g = 1の場合(TRUE)

データタイプは、入力パラメータin0およびin1の共通データタイプと同じです。

7.11.2 ファクションMUX

拡張可能なマルチプレックス機能

戻り値は、入力パラメータ k の値に応じて、 n 個のパラメータ $in_0 \sim in_{n-1}$ のどれか1つになります。

通知

```
FUNCTION MUX (  
    k      : ANY_INT,  
    in0    : ANY,  
    ...  
    inn-1  : ANY  
) : ANY
```

入力パラメータ

K

データタイプ: ANY_INT

入力パラメータ $in_0 \sim in_{n-1}$ の選択

値の範囲は、入力パラメータの数 n に応じて異なります

$in_0 \sim in_{n-1}$: $0 \leq k \leq n-1$

不正な値を指定すると、入力パラメータ in_0 が選択されます。

in₀

...

in_{n-1}

データタイプ: ANY

入力パラメータ $in_0 \sim in_{n-1}$ の数 n は可変です。

すべての入力パラメータ $in_0 \sim in_{n-1}$

1は同一のデータタイプであるか、暗黙的変換によって1つの共通データタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

ファクション呼び出し時に、 n 個の仮引数 $in_0 \sim in_{n-1}$

1を指定する場合は、昇順の途切れない番号で指定する必要があります。たとえば、4つの入力パラメータの場合、仮引数の識別子は: in_0, in_1, in_2, in_3 です。

戻り値

データタイプ: ANY

入力パラメータ k の値が m の場合、入力パラメータ in_m

データタイプは、入力パラメータ $in_0 \sim in_{n-1}$ の共通データタイプと同じです。

7.11.3 ファンクションMAX

拡張可能な最大値機能

戻り値は、n個の入力パラメータin0～in $n-1$ の最大値です。

通知

```
FUNCTION MAX (  
    in0          : ANY_ELEMENTARY,  
    ...  
    inn-1        : ANY_ELEMENTARY  
    ) : ANY_ELEMENTARY
```

入力パラメータ

in0

...

in $n-1$

データタイプ: ANY_ELEMENTARY

入力パラメータin0～in $n-1$ の数nは可変です。

すべての入力パラメータin0～in $n-1$

1は同一のデータタイプであるか、暗黙的変換によって最も強力なデータタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

ファンクション呼び出し時に、n個の仮引数in0～in $n-1$

1を指定する場合は、昇順の途切れない番号で指定する必要があります。たとえば、4つの入力パラメータの場合、仮引数の識別子は: in0, in1, in2, in3です。

戻り値

データタイプ: ANY_ELEMENTARY

入力パラメータの最大値

データタイプは、入力パラメータin0～in $n-1$ の中の最も強力なデータタイプと同じです。

7.11.4 ファンクションMIN

拡張可能な最小値機能

戻り値は、n個の入力パラメータin0～in $n-1$ の最小値です。

通知

```

FUNCTION MIN (
    in0          : ANY_ELEMENTARY,
    ...
    inn-1        : ANY_ELEMENTARY
) : ANY_ELEMENTARY

```

入力パラメータ

in0

...

in $n-1$

データタイプ: ANY_ELEMENTARY

入力パラメータin0～in $n-1$ の数 n は可変です。すべての入力パラメータin0～in $n-1$

1は同一のデータタイプであるか、暗黙的変換によって最も強力なデータタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

ファンクション呼び出し時に、 n 個の仮引数in0～in $n-1$

1を指定する場合は、昇順の途切れない番号で指定する必要があります。たとえば、4つの入力パラメータの場合、仮引数の識別子は: in0, in1, in2, in3です。

戻り値

データタイプ: ANY_ELEMENTARY

入力パラメータの最小値

データタイプは、入力パラメータin0～in $n-1$ の中の最も強力なデータタイプと同じです。

7.11.5 ファンクションLIMIT

制限機能

入力パラメータinの値は、下限値 mn と上限値 mx の間に制限されます。

通知

```

FUNCTION LIMIT (
    mn          : ANY_ELEMENTARY,
    in          : ANY_ELEMENTARY,
    mx          : ANY_ELEMENTARY
) : ANY_ELEMENTARY

```

入力パラメータ

mn

データタイプ: ANY_ELEMENTARY

下限値

すべての入力パラメータは同一のデータタイプであるか、暗黙的変換によって最も強力なデータタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

in

データタイプ: ANY_ELEMENTARY

制限される値

すべての入力パラメータは同一のデータタイプであるか、暗黙的変換によって最も強力なデータタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

mx

データタイプ: ANY_ELEMENTARY

上限値

すべての入力パラメータは同一のデータタイプであるか、暗黙的変換によって最も強力なデータタイプに変換可能でなければなりません(『STプログラミングマニュアル』の「**基本データタイプの変換**」を参照)。

戻り値

データタイプ: ANY_ELEMENTARY

MIN (MAX (in, mn), mx)

データタイプは、入力パラメータの中の最も強力なデータタイプと同じです。

7.12 変数の操作

7.12.1 一般

派生データタイプ(『STプログラミングマニュアル』の「ユーザー定義データタイプ(UDT)」を参照)のグローバル変数にアクセスする場合、複数のタスクが同じユーザー変数にアクセスするときは、データ一貫性を保証する責任はユーザーにあります(I/O変数、システム変数、テクノロジーオブジェクトのシステム変数、グローバルデバイス変数、およびグローバルユニット変数については、プログラミングマニュアルの「**変数モデル**」を参照してください)。

注記

タスク内で常に一貫したデータへのアクセスが確実に行われるようになります。

セマフォを使用して、グローバル変数の一貫性のある書き込みと読み取りを確保します。

データタイプDINTのグローバル変数は、セマフォとして使用することができます。以下のファクションを使って、セマフォのステータスを変更およびテストします。

- `_testAndSetSemaphore`
- `_releaseSemaphore`

グローバル変数への一貫したデータアクセスは、以下の条件で確実に行うことができます。

1. すべてのタスクが、グローバル変数へのアクセスをセマフォを設定することで示すこと。
2. すべてのタスクが、セマフォが有効化された場合のみグローバル変数にアクセスすること。

一貫性を持ったデータアクセスとセマフォの詳細は、「**変数の一貫性のある読み取りと書き込み(セマフォ)**」を参照してください。

下記も参照

一貫したデータアクセス (ページ 383)

7.12.2 `[_testAndSetSemaphore]`ファクション

このファクションを使用して、セマフォが設定されているかチェックします。

ファクション終了時には、セマフォは必ず設定されます。 `[_releaseSemaphore (semaA)]`ファクションが呼び出されるまで、このファクションの追加の呼出しは(他のプログラムからのものも)値FALSEを返します。

通知

```
_testAndSetSemaphore (
    sema      :DINT
) : BOOL
```

入力パラメータ

sema

データタイプ DINT

semaはデータタイプDINTのグローバル変数で、セマフォとして使用されます。

これは、インデックス化してはいけません。

変数が配列の要素の場合は、コンパイル時にインデックスを指定する必要があります(

例: a[2])。

戻り値

データタイプ: BOOL

この戻り値を使用して、セマフォが設定されているかどうかを判別できます。

TRUE セマフォは使用可能です。

FALSE セマフォは設定済みです。

例

「変数の一貫した読み取りおよび書き込み(セマフォ)」を参照してください。

下記も参照

例: セマフォによる一貫したデータアクセス (ページ 384)

7.12.3 [_releaseSemaphore]ファクション

このファクションを使用して、セマフォを使用可能にします。

次に、[_testAndSetSemaphore]ファクションを呼び出すと(別のプログラムから呼び出し
ても)、TRUEの値が返されます。

通知

```
_releaseSemaphore (
    sema      :DINT
) : VOID
```

入力パラメータ

sema

データタイプ: DINT

semaはデータタイプDINTのグローバル変数で、セマフォとして使用されます。

これは、インデックス化してはいけません。

変数が配列の要素の場合は、コンパイル時にインデックスを指定する必要があります(

例: a[2])。

戻り値

なし

例

「変数の一貫した読み取りおよび書き込み(セマフォ)」を参照してください。

下記も参照

例: セマフォによる一貫したデータアクセス (ページ 384)

7.13 システム変数および入/出力へのアクセス

7.13.1 システム変数および入/出力へのアクセスの概要

[_getSafeValue]および[_setSafeValue]ファクションにより、システム変数、コンフィグレーションデータ、またはI/O変数へのアクセス時の特別なエラー応答が可能になります。故障の場合の応答は、設定された応答とは別に設定することができます(「**システム変数およびコンフィグレーションデータへのアクセス時のアクセスエラー**」を参照)。

システムファクション[_getSafeValue]および[_setSafeValue]は、実行時間がかかります。そのため、IF文の中、TOの再開の待機時など、必要な場合にのみ使用します。

V4.1以降では、障害が発生した場合の置換値、またはシステム変数へのアクセス時の最新値も設定できます(TOの再開など)。コンフィグレーションデータrestartInfo.behaviorInvalidSysvarAccessは、これに関連します。

[_getInOutByte]ファクションにより、アドレスを指定して個々のI/Oバイトに直接読み取りアクセスできます。

下記も参照

システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー (ページ 95)

7.13.2 [_getSafeValue]ファクション

このファクションは、指定されたシステム変数(またはコンフィグレーションデータ要素)またはI/O変数を読み取り、別の変数で値を返します。

STOPモードへの移行が発生すると、ExecutionFaultTaskは起動されません。

(変数割り当ての代わりに)このファクションを使用すると、システム変数、コンフィグレーションデータ、またはI/O変数へのアクセス中にエラーが発生した場合、STOPモードへの移行を回避できます(テクノロジーオブジェクトの再起動時やI/Oの障害時など)。

エラー応答の指定

[accessmode]パラメータを使って、エラー応答を制御することができます。

- CONFIGURED (デフォルト):
指定されたエラー応答については、「**システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー**」を参照してください。

- NO_CHANGE:
 - システム変数およびコンフィグレーションデータ
値は変更されません。つまり最後に使用した値です。
 - I/O変数:
読み出しアクセス(入力部または出力部への)では
最後に有効だった値が適用されます。

書き込みアクセス(出力部への)では 値が変数に書き込まれます。
ただし、出力が再度使用できるようになるまで、出力部で有効化されません。
- DEFAULT_VALUE: 置換値または制限値を取ります。
- STOP_DEVICE: SIMOTIONデバイスが、STOPモードに切り替わります。
この戻り値を使って、アクセスが成功したか確認することができます。

通知

```
_getSafeValue (  
    variable      : ANY,           //システム変数,  
                                   // コンフィグレーションデータまたは  
                                   // I/O変数  
    accessMode    : EnumAccessMode,  
    getValue      : ANY  
    )             : EnumSetAndGetSafeValue
```

入力パラメータ

variable

データタイプ: ANY

読み込むシステム変数、コンフィグレーションデータ項目、またはI/O変数。

accesssMode

データタイプ: EnumAccessMode

読み取りアクセス時にエラーが発生したときの対応。

```
TYPE EnumAccessMode : (  
/構成時の対応:  
    CONFIGURED      // システム変数用および  
                    // コンフィグレーションデータ: 運転状態STOP  
                    //   への移行  
                    // I/O変数用:  
                    // I/O変数の作成時に指定する戦略  
// 設定されたものとは異なる応答:
```

```

NO_CHANGE          // システム変数および
                   // コンフィグレーションデータ: 変数
                   // 読み込みません。
                   // I/O変数用: 使用可能な
                   // (有効な) 最新値を
                   // 受理します。

DEFAULT_VALUE      // I/O値のみ使用可能。
                   // 置換値を適用します。

STOP_DEVICE        // デバイスはSTOP状態に
                   移行します。
END_TYPE

```

getvalue

データタイプ: ANY

システム変数(またはコンフィグレーションデータ項目)またはI/O変数の現在値を書き込む変数の名前。

このデータタイプには、以下が適用されます。

- これは、読み込まれる変数のデータタイプと同じである必要があります。または、
- 読み込まれる変数のデータタイプは、暗黙的なタイプ変換によってこのデータタイプに変換可能である必要があります(「障害およびイベントの評価」を参照)。

戻り値

データタイプ: EnumSetAndGetSafeValue

この戻り値は、アクセスが成功したかどうかを示します。

```

TYPE EnumSetAndGetSafeValue : (
  // アクセス成功:
  OK          // アクセスが成功しました。
  // アクセスエラー:
  , NO_CHANGE      // システム変数用: 変数は
                  // 読み込まれず、値は不定です
                  // I/O変数用: 使用可能な
                  // (有効な) 最新値が
                  // 受理されました。
  , DEFAULT_VALUE  // I/O変数のみ使用可能。
                  // 置換値は受理されました。
)

```



```
, INVALID_VALUE // コンフィグレーションデータ専用:  
                //   コンフィグレーションデータが  
                //   読み込まれず、値は不定です  
                //   許可されないパラメータ  
                //   (accessMode = DEFAULT_VALUE)。  
, NO_ACCESS )   // コンフィグレーションデータ専用:  
                //   コンフィグレーションデータが  
                //   読み込まれず、値は不定です  
END_TYPE
```

下記も参照

障害およびイベントの評価 (ページ 93)

システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー (ページ 95)

システムデータへのアクセス時のエラー (ページ 124)

7.13.3 [_setSafeValue]ファクション

このファクションは、指定された値を、システム変数、コンフィグレーションデータ要素、またはI/O変数に書き込み、オプションで現在書き込んだ値を別の変数で値を返します。故障の場合の応答は、設定された応答とは別に設定することができます(「**システム変数およびコンフィグレーションデータへのアクセス時のアクセスエラー**」を参照)。

STOPモードへの移行が発生すると、ExecutionFaultTaskは起動されません。

(変数割り当ての代わりに)このファクションを使用すると、システム変数、コンフィグレーションデータ、またはI/O変数へのアクセス中にエラーが発生した場合(テクノロジーオブジェクトの再起動や入/出力の障害時など)、STOPモードへの移行を回避できます。

エラー応答の指定

[accessmode]パラメータを使って、エラー応答を制御することができます。

- CONFIGURED (デフォルト):
指定されたエラー応答については、「**システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー**」を参照してください。

- NO_CHANGE:
 - システム変数およびコンフィグレーションデータ
値は変更されません。つまり最後に使用した値です。
 - I/O変数:
読み出しアクセス(入力部または出力部への)では
最後に有効だった値が適用されます。

書き込みアクセス(出力部への)では 値が変数に書き込まれます。
ただし、出力が再度使用できるようになるまで、出力部で有効化されません。
 - DEFAULT_VALUE: 置換値または制限値を取ります。
 - STOP_DEVICE: SIMOTIONデバイスが、STOPモードに切り替わります。
- この戻り値を使って、アクセスが成功したか確認することができます。

通知

```
_setSafeValue (
    variable      : ANY,           //システム変数,
                                // コンフィグレーションデータまたは
                                // I/O変数
    value         : ANY,
    accessMode    : EnumAccessMode,
    { setValue    : ANY
    }
) : EnumSetAndGetSafeValue
```

入力パラメータ

variable

データタイプ: ANY
書き込むシステム変数、コンフィグレーションデータ項目、またはI/O変数。

値

データタイプ: ANY
システム変数(またはコンフィグレーションデータ項目)またはI/O変数に書き込む値。
値のデータタイプは、書き込む変数のデータタイプと同じであるか、または暗黙的な
データ変換によってこのデータタイプに変換可能である必要があります。(「障害およ
びイベントの評価」を参照)。

accessMode

データタイプ: EnumAccessMode
書き込みアクセス時にエラーが発生したときの対応:

```
TYPE EnumAccessMode : (  
  // 構成時の対応:  
    CONFIGURED      // システム変数用および  
                      // コンフィグレーションデータ: 運転状態STOP  
                      // 動作モードSTOP  
                      // I/O変数の場合:  
                      // I/O変数の作成時に  
                      // 指定する戦略  
  // 設定されたものとは異なる応答:  
    , NO_CHANGE      // システム変数および  
                      // コンフィグレーションデータ:  
                      // 変数の値は変更されません  
                      // I/O変数の場合  
                      // パラメータに渡す値が  
                      // 変数に書き込まれ  
                      // ます。 出力部では有効ではありません  
                      // 出力が再び  
  // 使用可能になるまで  
  , DEFAULT_VALUE    // システム変数およびI/O変数  
  // のみ使用可能  
  // システム変数の場合:  
  // 変数は  
  // 制限値で記述します。  
  // I/O変数用:  
  // 変数は  
  // 変数の作成時に指定された置換値で  
  // 記述します。  
  STOP_DEVICE        // デバイスはSTOPモードに  
  // 移行します  
END_TYPE
```

setvalue

データタイプ: ANY

システム変数、コンフィグレーションデータ項目、またはI/O変数の現在値を書き込む変数の名前。

このデータタイプには、以下が適用されます。

- これは、書き込まれる変数のデータタイプと同じである必要があります。または、
- 書き込まれる変数のデータタイプは、暗黙的なタイプ変換によってこのデータタイプに変換可能である必要があります(「障害およびイベントの評価」を参照)。

たとえば、100を書き込むべきところを90しか入力がない場合、*setValue*には90が入ります。

戻り値

データタイプ: EnumSetAndGetSafeValue

この戻り値は、アクセスが成功したかどうかを示します。

```
TYPE EnumSetAndGetSafeValue : (
  // アクセス成功:
  OK                // アクセスが成功しました。
  // アクセスエラー:
  , NO_CHANGE       // システム変数用:
                      //   Variablesは変更されませんでした
                      //   (不正な値または
                      //   使用不可の変数)。
                      //   コンフィグレーションデータの場合: コンフィグレーションデータ
                      //   の値は変更されませんでした
                      //   (不正な値)。
                      //   I/O変数用: パラメータで
                      //   伝送された値が
                      //   書き込まれました。 出力部では有効ではありません
                      //   出力が再び
                      //   使用可能になるまで
  , DEFAULT_VALUE   // システム変数の場合: 制限
                      //   が有効なので、制限値が
                      //   書き込まれました。
                      //   I/O変数用: 置換値が
                      //   書き込まれました。 出力部では有効ではありません
                      //   出力が再び
                      //   使用可能になるまで
  , INVALID_VALUE    // コンフィグレーションデータ専用: コンフィグレーションデータ項目の
```

```
// 値が  
// 変更されませんでした。不正な  
// パラメータ  
//      (accessMode = DEFAULT_VALUE)。  
, NO_ACCESS )      // コンフィグレーションデータ専用:規格値  
// 値が  
// 変更されませんでした。  
// コンフィグレーションデータ項目は  
//      使用できません)。  
END_TYPE
```

下記も参照

システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー (ページ 95)

障害およびイベントの評価 (ページ 93)

システムデータへのアクセス時のエラー (ページ 124)

7.13.4 [_getInOutByte]ファクション

このファクションにより、入/出力のアドレスを指定して個々のI/Oバイトに直接読み取りアクセスできます。

アクセスエラー発生時には、PeripheralFaultTaskが呼び出されず、対応する値が戻り値のfunctionResultコンポーネントで返されます。

指定されたI/OアドレスにI/O変数(周期的タスクの直接アクセスまたはプロセスイメージ(『STプログラミングマニュアル』の「**周期的タスクの直接アクセスおよびプロセスイメージ**」を参照)に)が定義され、置換値が指定されている場合、アクセスエラー発生時にはこの置換値が返されます。

この戻り値の評価により、入力または出力がハードウェアのアドレスに割り当てられているかどうかを確認できます。

通知
このファクションの実行時間は非常に長くなる場合があります。そのため、このファクションは、高速な周期的タスクの用途には不向きです。

通知

```
_getInOutByte (
    mode      : EnumInOutDirection,
    logAddress : DINT
): StructRetGetInOutByte
```

入力パラメータ

mode

データタイプ: EnumInOutDirection

logAddressを入力または出力へのアクセスのどちらに使用するかを指定します。

```
TYPE EnumInOutDirection : (
    IN          // 入力へのアクセス
    , OUT )     // 出力へのアクセス
END_TYPE
```

logAddress

データタイプ: DINT

入力または出力の論理アドレス

戻り値

データタイプ: StructRetGetInOutByte

関クションの呼び出し結果および読み取りバイト値

```
TYPE StructRetGetInOutByte : STRUCT
    functionResult : DINT; // ファンクション呼び出しの
                          // 結果
    value          : BYTE; // 読み込まれたバイトの値
END_STRUCT;
END_TYPE
```

functionResult (ファンクション呼び出しの結果)のとり得る値:

0 エラーなし。アクセス成功。

16#FFFF_FFFA	使用可能メモリが不十分
16#FFFF_FFFE	アクセスエラー
16#FFFF_FFFF	入/出力使用不可

7.14 ユーザープログラムからのデータのバックアップ

7.14.1 ユーザープログラムからのデータのバックアップの概要

下で説明するファンクションの用途は次のとおりです。

- 次の値の保存、ロード、初期化
 - ユニットのインターフェースの非保持型または保持型ユニット変数または実装セクション(例: STソースファイル、MCCユニット)
 - 非保持型または保持型グローバルデバイス変数。

データバックアップは、関連するファンクションを選択して実行します。

- 2進法:
データセグメントのバージョンIDを変更すると、バックアップデータセットは読み込めなくなります。

5-268ページの情報にも注意してください。

- ASCII形式:
データセグメントのバージョンIDを変更しても、バックアップデータセットは読み込めます。

- 値をバックアップするデータセットの管理

値はデータセットにバックアップされます。これは、保存場所、ユニット名、データセット番号を指定すると、一意に識別されます。

これらのファンクションの応用(特にステップ有効化条件)については、「ユーザープログラムからのデータバックアップおよび初期化」に詳細な説明があります。

下記も参照

ユーザープログラムからのデータバックアップとデータ初期化 - 機能と手順 (ページ 385)

7.14.2 [_saveUnitDataSet]ファンクション

次の変数の値は、バイナリデータレコードで保存されます。

- SIMOTION KernelのバージョンV3.2以降:
 - ユニットのインターフェースの非保持型または保持型ユニット変数または実装セクション(例: STソースファイル、MCCユニット)
 - 非保持型または保持型グローバルデバイス変数。

- SIMOTION KernelのバージョンV3.1以前
ユニットのインターフェースセクションの非保持型ユニット変数(例: STソースファイル、MCCユニット)

データが格納される場所を選択することができます。

- 一時データ保管(RAMディスク)、電源異常があると削除
- 恒久的データ保管(メモリカード)、電源異常があっても保持

Ethernetを介したUDPプロトコルによる通信」の情報も参照してください。

バックアップするデータの整合性に注意してください(「**変数の読み取りおよび書き込みの整合性(セマフォ)**」を参照)。

このファクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_saveUnitDataSet (
    unitName      : STRING,
    id            : UDINT,
    storageType   : EnumDeviceStorageType,
    { path        : STRING,
      overwrite   : BOOL,
      nextCommand : EnumNextCommandMode,
      dataScope   : EnumDeviceDataScope,
      kindOfData  : EnumDeviceKindOfData,
    }
): StructRetUnitDataSetCommand
```

入カパラメータ

unitName

データタイプ: STRING

ユニットの名前(たとえば、STソースファイル、MCCユニットなど)。このユニット変数がバックアップされます。

名前は、小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

'_device'を指定すると、グローバルデバイス変数が保存されます(SIMOTION Kernelのバージョン3.2以降で可能)。

id

データタイプ: UDINT

変数値が保存されるデータセット数(ユニットあたり最大1_000_000データセット)。

storageType

データタイプ: EnumDeviceStorageType

データが保存される場所。

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE      // 一時保管
                           // (RAMディスク),
                           // は、電源が故障した場合には削除されます
    , PERMANENT_STORAGE    // 永続的保管
                           // (メモリカード),
                           // 電源故障でも保持
```

```

, USER_DEFINED )           // パス定義付き
                           // (準備)
END_TYPE

```

path (オプション)
 データタイプ: STRING
 デフォルト: '' (空の文字列)
 Destination path, if storageType = USER_DEFINED
 準備として、初期値を指定することができます。

overwrite (オプション)
 データタイプ: BOOL
 デフォルト: FALSE
 TRUEの場合、既存のデータセットが上書きされます。

nextCommand (オプション)
 データタイプ: EnumNextCommandMode
 デフォルト: IMMEDIATELY
 次のコマンドへ進む

```

TYPE EnumNextCommandMode : (
  IMMEDIATELY           // 即座
  , WHEN_COMMAND_DONE ) // コマンドの完了または中止
                       // 後
END_TYPE

```

dataScope (オプション)
 データタイプ: EnumDeviceDataScope
 デフォルト: _INTERFACE
 このパラメータはSIMOTION KernelのバージョンV3.2以降で使用可能です。
 ユニット変数を保存するセクションの指定。

```

Type EnumDeviceDataScope : (
  _INTERFACE           // インターフェースセクション
  , _IMPLEMENTATION    // 実装セクション
  , _INTERFACE_AND_IMPLEMENTATION ) // インターフェースおよび
                                   // 実装セクション
END_TYPE

```

unitName =
'_device'が指定されている場合、dataScopeには値_INTERFACEまたは_INTERFACE_AND_IMPLEMENTATIONのみが使用できます。
SIMOTION
KernelのバージョンV3.1以前では、ユニットのインターフェースセクションの保持されない変数のみをバックアップできます。

KindOfData (オプション)
デフォルト NO_RETAIN_GLOBAL
このパラメータはSIMOTION KernelのバージョンV3.2以降で使用可能です。
保持されないグローバル変数または保持されるグローバル変数の指定が保存されます。
。

```
TYPE EnumDeviceKindOfData : (  
    NO_RETAIN_GLOBAL           // 保持されない変数  
    , _RETAIN                  // 保持される変数  
    , ALL_GLOBAL )             // 保持される変数および保持されない  
                                // 変数  
END_TYPE
```

SIMOTION
KernelのバージョンV3.1以前では、ユニットのインターフェースセクションの保持されない変数のみを保存できます。

戻り値

データタイプ: StructRetUnitDataSetCommand
戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。
これは、以下で構成されます。

- functionResultコンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- handle コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]関数で使用すると、データバックアップ関数の現在の状態をチェックできます(特にステップ有効化条件IMMEDIATELYに有用)([_getStateOfUnitDataSetCommand]関数を参照)。

```

TYPE
EnumDeviceUnitDataSetCommand: (
DONE // 起動の実行
// 成功
, ACTIVE // 実行中
, INTERNEL_ERROR // 内部エラー
// (ホットラインにご連絡
// ください)
, COMMAND_FAILED// コマンドを実行できません
, NO_COMMAND_BUFFER_AVAILABLE// コマンドバッファがいっぱいです
, COMMAND_NOT_FOUND // コマンド(ハンドル)が
// 見つかりません
, DATASET_ID_NOT_VALID// データセット番号が不正です
, READ_ERROR // データ読み取りエラー
// (記憶媒体に欠陥があります)
, NO_STORAGE_AVAILABLE// 使用可能なストレージがありません
, ACCESS_DENIED// アクセスが拒否されました
// (書き込み / 読み取り権限が
// ありません)
, DATASET_ALREADY_EXISTS// データセットは既に存在します
, DATASET_NOT_FOUND// データセットが見つかりません
, VERSION_MISMATCH// インポートされる
// データレンジのバージョンIDが
// 不正です(
// ユニットのセクションなど
, UNIT_NOT_FOUND// ユニット(STソース、
// MCCソースなど)が見つかりません
, DATA_INCOMPLETE// データは不完全に
// インポートされました
, DATA_MISMATCH// インポートするデータレンジは
// データセットに
// 含まれません
, SYMBOL_INFORMATION_NOT_AVAILABLE ) //シンボル情報が
// 使用できません
// (プログラムソースでOPC-XMLを
// 有効にします
StructRetUnitDataSetCommand
: STRUCT
functionResult : EnumDeviceUnitDataSetCommand;
handle : UDINT;
END_STRUCT;
END_TYPE

```

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

ユーザープログラムからのデータのバックアップの概要 (ページ 344)

7.14.3 [_loadUnitDataSet]ファクション

次の変数の値は、[_saveUnitDataSet]によりバイナリデータセットとしてロードされます。

- SIMOTION KernelのバージョンV3.2以降:
 - ユニットのインターフェースの非保持型または保持型ユニット変数または実装セクション(例: STソースファイル、MCCユニット)
 - 非保持型または保持型グローバルデバイス変数。
- SIMOTION KernelのバージョンV3.1以前
ユニットのインターフェースセクションの非保持型ユニット変数(例: STソースファイル、MCCユニット)

データセットが保存されてから、ロードするすべてのデータセグメント(ユニットのインターフェースセクションの保持されない変数または保持される変数など)のバージョンIDに変更がない場合のみ、データセットをロードできます。
データセグメントとそのバージョンIDについては、『STプログラミングマニュアル』の「**変数初期化の時間**」を参照してください。

Ethernetを介したUDPプロトコルによる通信」の情報も参照してください。

データセットでバックアップされたデータセグメントのサブセットをロードすることができます(一部のデータセグメントのバージョンIDが変更された場合など)。

このファクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_loadUnitDataSet (
    unitName      : STRING,
    id            : UDINT,
    storageType    : EnumDeviceStorageType,
    { path        : STRING,
      nextCommand : EnumNextCommandMode,
      dataScope   : EnumDeviceDataScope,
      kindOfData  : EnumDeviceKindOfData,
    }
): StructRetUnitDataSetCommand
```

入力パラメータ

unitName	
データタイプ:	STRING

ユニットの名前(たとえば、STソースファイル、MCCユニットなど)。このユニット変数がロードされます。

名前は、小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

'_device'を指定すると、グローバルデバイス変数が保存されます(SIMOTION Kernelのバージョン3.2以降で可能)。

id

データタイプ: UDINT

変数値が保存されるデータセット数(ユニットあたり最大1_000_000データセット)。

storageType

データタイプ: EnumDeviceStorageType

データが保存される場所。

```

TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE          // 一時保管
                                // (RAMディスク),
                                // は、電源が故障した場合には削除されます
    , PERMANENT_STORAGE        // 永続的保管
                                // (メモリカード),
                                // 電源故障でも保持
    , USER_DEFINED )          // パス定義付き
                                // (準備)
END_TYPE

```

パス (オプション)

データタイプ: STRING

デフォルト: '' (空の文字列)

Destination path, if storageType = USER_DEFINED

準備として、初期値を指定することができます。

nextCommand (オプション)

データタイプ: EnumNextCommandMode

デフォルト: IMMEDIATELY

次のコマンドへ進む

```

TYPE EnumNextCommandMode : (
    IMMEDIATELY                // 即座
    , WHEN_COMMAND_DONE )      // コマンドの完了または中止
                                // 後
END_TYPE

```

dataScope (オプション)

データタイプ: EnumDeviceDataScope

デフォルト _INTERFACE

このパラメータはSIMOTION KernelのバージョンV3.2以降で使用可能です。

ユニット変数を保存するセクションの指定。

```
Type EnumDeviceDataScope : (  
    _INTERFACE                // インターフェースセクション  
    , _IMPLEMENTATION         // 実装セクション  
    , _INTERFACE_AND_IMPLEMENTATION ) // インターフェースおよび  
                                    // 実装セクション  
END_TYPE
```

unitName =

'_device'が指定されている場合、*dataScope*には値_INTERFACEまたは_INTERFACE_AND_IMPLEMENTATIONのみが使用できます。

SIMOTION

KernelのバージョンV3.1以前では、ユニットのインターフェースセクションの保持されない変数のみを保存およびロードできます。

KindOfData (オプション)

デフォルト NO_RETAIN_GLOBAL

このパラメータはSIMOTION KernelのバージョンV3.2以降で使用可能です。

保持されないグローバル変数または保持されるグローバル変数の指定が保存されます。

```
TYPE EnumDeviceKindOfData : (  
    NO_RETAIN_GLOBAL          // 保持されない変数  
    , _RETAIN                 // 保持される変数  
    , ALL_GLOBAL )           // 保持される変数および保持されない  
                              // 変数  
END_TYPE
```

SIMOTION

KernelのバージョンV3.1以前では、ユニットのインターフェースセクションの保持されない変数のみを保存およびロードできます。

戻り値

データタイプ: StructRetUnitDataSetCommand

戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。

これは、以下で構成されます。

- *functionResult* コンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]ファンクションで使用すると、データバックアップファンクションの現在の状態をチェックできます(特にステップ有効化条件IMMEDIATELYに有用)([_getStateOfUnitDataSetCommand]ファンクションを参照)。

データタイプStructRetUnitDataSetCommandおよびEnumDeviceUnitDataSetCommandについては、[_saveUnitDataSet]ファンクションの戻り値のセクションを参照してください([_saveUnitDataSet]ファンクションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

[_getStateOfUnitDataSetCommand]ファンクション (ページ 362)

[_saveUnitDataSet]ファンクション (ページ 344)

ユーザープログラムからのデータのバックアップの概要 (ページ 344)

7.14.4 [_exportUnitDataSet]ファンクション(カーネルV3.2以降)

このファンクションは、SIMOTION Kernel V3.2以降で使用できます。

次の変数の値は、ASCII形式のデータセットとしてエクスポートされます。

- ユニットのインターフェースセクションの非保持型ユニット変数(例: STソースファイル、MCCユニット)
- ユニットのインターフェースセクションの保持型ユニット変数

データが格納される場所を選択することができます。

- 一時データ保管(RAMディスク)、電源異常があると削除
- 恒久的データ保管(メモリカード)、電源異常があっても保持

SIMOTIONデバイスのインターフェースセクションにあるユニット変数のシンボル情報が、関連するユニットで使用できることを確認します。

そのため、コンパイラのローカル設定のプログラムソースで、[Enable OPC-XML]チェックボックスを有効にします(プログラミングマニュアルの「コンパイラの操作」または「STコンパイラの設定」を参照してください)。

エクスポートするデータの整合性に注意してください(「**変数の読み取りおよび書き込みの整合性(セマフォ)**」を参照)。

エクスポートされたデータセットは、データ領域(ユニットのインターフェースセクションの保持される変数など)のバージョンIDが変更された場合(データ構造の変更など)、[_import UnitDataSet]ファクションを使用してインポートすることもできます(サブセクション6.17.4を参照)。

このファクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_exportUnitDataSet (
    unitName      : STRING,
    id            : UDINT,
    storageType    : EnumDeviceStorageType,
    { path        : STRING,
      overwrite    : BOOL
    nextCommand   : EnumNextCommandMode,
    dataScope     : EnumDeviceDataScope,
    kindOfData    : EnumDeviceKindOfData,
    }
): StructRetUnitDataSetCommand
```

入力パラメータ

unitName

データタイプ: STRING

ユニットの名前(たとえば、STソースファイル、MCCユニットなど)。このユニット変数がエクスポートされます。

名前は、小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

ここでは'_device'

(グローバルデバイス変数のエクスポート用)を指定することはできません。

id

データタイプ: UDINT

変数値が保存されるデータセット数(ユニットあたり最大1_000_000データセット)。

storageType

データタイプ: EnumDeviceStorageType

データが保存される場所。

```
TYPE EnumDeviceStorageType : (
    TEMPORARY_STORAGE      // 一時保管
                           // (RAMディスク),
                           // は、電源が故障した場合には削除されます
    , PERMANENT_STORAGE    // 永続的保管
```

```

// (メモリカード),
// 電源故障でも保持
, USER_DEFINED ) // パス定義付き
// (準備)
END_TYPE

```

パス (オプション)
 データタイプ: STRING
 デフォルト: '' (空の文字列)
 Destination path, if storageType = USER_DEFINED
 準備として、初期値を指定することができます。

overwrite (オプション)
 データタイプ: BOOL
 デフォルト: FALSE
 TRUEの場合、既存のデータセットが上書きされます。

nextCommand (オプション)
 データタイプ: EnumNextCommandMode
 デフォルト: IMMEDIATELY
 次のコマンドへ進む

```

TYPE EnumNextCommandMode : (
  IMMEDIATELY // 即座
, WHEN_COMMAND_DONE ) // コマンドの完了または中止
// 後
END_TYPE

```

dataScope (オプション)
 データタイプ: EnumDeviceDataScope
 デフォルト: _INTERFACE
 (今後導入予定)値_INTERFACE または_INTERFACE_AND_IMPLEMENTATION
 が許可されます。
 ユニット変数をエクスポートするセクションの指定。

```

Type EnumDeviceDataScope : (
  _INTERFACE // インターフェースセクション
, _IMPLEMENTATION // 実装セクション

```

```
, _INTERFACE_AND_IMPLEMENTATION )    // インターフェースおよび  
                                        // 実装セクション  
END_TYPE
```

(保持されないまたは保持される)1つのユニットのインターフェースセクションの変数のみ
エクスポートまたはインポートできます。

KindOfData (オプション)
デフォルト NO_RETAIN_GLOBAL
保持されないグローバル変数または保持されるグローバル変数の指定がエクスポート
されます。

```
TYPE EnumDeviceKindOfData : (  
    NO_RETAIN_GLOBAL          // 保持されない変数  
    , _RETAIN                  // 保持される変数  
    , ALL_GLOBAL )            // 保持されるおよび保持されない  
                                // 変数  
END_TYPE
```

戻り値

データタイプ: StructRetUnitDataSetCommand

戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。
これは、以下で構成されます。

- *functionResult*コンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]ファクションで使用すると、データバック
アップファクションの現在の状態をチェックできます(特にステップ有効化条件IMME
DIATELYに有用)([_getStateOfUnitDataSetCommand]ファクションを参照)。

データタイプStructRetUnitDataSetCommandおよびEnumDeviceUnitDataSetCommandに
ついては、[_saveUnitDataSet]ファクションの戻り値のセクションを参照してください([
_saveUnitDataSet]ファクションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデ
ータの初期化の使用 (ページ 385)」を参照してください

下記も参照

- 一貫したデータアクセス (ページ 383)
- [_getStateOfUnitDataSetCommand]ファンクション (ページ 362)
- ユーザープログラムからのデータのバックアップの概要 (ページ 344)

7.14.5 [_importUnitDataSet]ファンクション(カーネルV3.2以降)

このファンクションは、SIMOTION Kernel V3.2以降で 사용할 수 있습니다.

次の変数の値は、[_exportUnitDataSet]でエクスポートされたデータセットからインポートされます。

- ユニットのインターフェースセクションの非保持型ユニット変数(例: STソースファイル、MCCユニット)
- ユニットのインターフェースセクションの保持型ユニット変数

SIMOTIONデバイスのインターフェースセクションにあるユニット変数のシンボル情報が、関連するユニットで使用できることを確認します。

そのため、コンパイラのローカル設定のプログラムソースで、[Enable OPC-XML]チェックボックスを有効にします(プログラミングマニュアルの「コンパイラの操作」または「STコンパイラの設定」を参照してください)。

データセットが保存されてから、ロードするデータセグメントのバージョンIDが変更されていても(ユニットのインターフェースセクションの保持されない変数など)、データセットをインポートできます。

- すでに存在しない変数は無視されます。
- 追加された変数の値は、変更されないままになります。
- 変更されたデータタイプを持つ変数の場合、新規のデータタイプに変換することができる場合は、値が伝送されます。変更できない場合は、変数の値は保持されます。
- このファンクションの戻り値はDATA_INCOMPLETEです。

変数に不要な値を入れないように、データセットを[_resetUnitData]ファンクションでインポートする前に関連するデータセグメントを初期化できます(SIMOTIONデバイスのシステムファンクションのパラメータマニュアルを参照してください)。

データセグメントとそのバージョンIDについては、『STプログラミングマニュアル』の「**変数初期化の時間**」を参照してください。

データセットでエクスポートされたデータセグメントのサブセットをインポートできます。

このファンクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_exportUnitDataSet (
    unitName      : STRING,
    id            : UDINT,
    storageType   : EnumDeviceStorageType,
    { path        : STRING,
      nextCommand : EnumNextCommandMode,
```

```
dataScope      : EnumDeviceDataScope,  
kindOfData     : EnumDeviceKindOfData,  
}  
) : StructRetUnitDataSetCommand
```

入力パラメータ

unitName

データタイプ: STRING

ユニットの名前(たとえば、STソースファイル、MCCユニットなど)。このユニット変数がインポートされます。

名前は、小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

'_device'を指定すると、グローバルデバイス変数がインポートされます。

id

データタイプ: UDINT

変数値が保存されるデータセット数(ユニットあたり最大1_000_000データセット)。

storageType

データタイプ: EnumDeviceStorageType

データが保存される場所。

```
TYPE EnumDeviceStorageType : (  
  TEMPORARY_STORAGE // 一時保管  
  // (RAMディスク),  
  // は、電源が故障した場合には削除されます  
  , PERMANENT_STORAGE // 永続的保管  
  // (メモリカード),  
  // 電源故障でも保持  
  , USER_DEFINED ) // パス定義付き  
  // (準備)  
END_TYPE
```

パス (オプション)

データタイプ: STRING

デフォルト: '' (空の文字列)

Destination path, if storageType = USER_DEFINED

準備として、初期値を指定することができます。

overwrite (オプション)

データタイプ: BOOL

デフォルト: FALSE

TRUEの場合、既存のデータセットが上書きされます。

nextCommand (オプション)
 データタイプ: EnumNextCommandMode
 デフォルト: IMMEDIATELY
 次のコマンドへ進む

```
TYPE EnumNextCommandMode : (
    IMMEDIATELY          // 即座
    , WHEN_COMMAND_DONE ) // コマンドの完了または中止
                          // 後
END_TYPE
```

dataScope (オプション)
 データタイプ: EnumDeviceDataScope
 デフォルト _INTERFACE
 (今後導入予定)値_INTERFACE または_INTERFACE_AND_IMPLEMENTATION
 が許可されます。
 ユニット変数をインポートするセクションの指定。

```
Type EnumDeviceDataScope : (
    _INTERFACE          // インターフェースセクション
    , _IMPLEMENTATION   // 実装セクション
    , _INTERFACE_AND_IMPLEMENTATION ) // インターフェースおよび
                                      // 実装セクション
END_TYPE
```

(保持されないまたは保持される)1つのユニットのインターフェースセクションの変数のみ
 エクスポートまたはインポートできます。

KindOfData (オプション)
 デフォルト NO_RETAIN_GLOBAL
 保持されないグローバル変数または保持されるグローバル変数の指定もインポートさ
 れます。

```
TYPE EnumDeviceKindOfData : (  
    NO_RETAIN_GLOBAL          // 保持されない変数  
    , _RETAIN                  // 保持される変数  
    , ALL_GLOBAL )            // 保持されるおよび保持されない  
                                // 変数  
END_TYPE
```

保持される変数または保持されない変数がエクスポートされたデータセットに格納されている場合、保持されている変数または保持されていない変数を個別にインポートできます。

戻り値

データタイプ: StructRetUnitDataSetCommand

戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。

これは、以下で構成されます。

- *functionResult*コンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]ファクションで使用すると、データバックアップファクションの現在の状態をチェックできます(特にステップ有効化条件IMMEDIATELYに有用)([_getStateOfUnitDataSetCommand]ファクションを参照)。

データタイプStructRetUnitDataSetCommandおよびEnumDeviceUnitDataSetCommandについては、[_saveUnitDataSet]ファクションの戻り値のセクションを参照してください([_saveUnitDataSet]ファクションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

[_getStateOfUnitDataSetCommand]ファクション (ページ 362)

ユーザープログラムからのデータのバックアップの概要 (ページ 344)

[_loadUnitDataSet]ファクション (ページ 349)

7.14.6 [_deleteUnitDataSet]ファンクション

次の変数の保存された値を含む単一のデータレコードを削除します。

- ユニットのインターフェースまたは実装セクション(例: STソースファイル、MCCユニット)のバックアップされた非保持型または保持型ユニット変数。
- バックアップされた非保持型または保持型のグローバルデバイス変数。
- エクスポートされたユニットのインターフェースセクション(例: STソースファイル、MCCユニット)の非保持型または保持型ユニット変数。

このファンクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_deleteUnitDataSet (
    unitName      : STRING,
    id             : UDINT,
    storageType    : EnumDeviceStorageType,
    { path         : STRING,
      nextCommand : EnumNextCommandMode,
    }
): StructRetUnitDataSetCommand
```

入力パラメータ

unitName

データタイプ: STRING

ユニット名(例: STソースファイル、MCCユニット):

名前は小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

'_device'を指定すると、グローバルデバイス変数のデータセットが削除されます(SIMOTION Kernelのバージョン3.2以降可能)。

id

データタイプ: UDINT

削除するデータセット数(ユニットあたり最大1_000_000データセット)。

storageType

データタイプ: EnumDeviceStorageType

データセットを削除する場所。


```
TYPE EnumDeviceStorageType : (  
    TEMPORARY_STORAGE          // 一時保管  
                                // (RAMディスク),  
                                // は、電源が故障した場合には削除されます  
    , PERMANENT_STORAGE        // 永続的保管  
                                // (メモリカード),  
                                // 電源故障でも保持  
    , USER_DEFINED )          // パス定義付き  
                                // (準備)  
END_TYPE
```

パス (オプション)
データタイプ: STRING
デフォルト: '' (空の文字列)
Destination path, if storageType = USER_DEFINED
準備として、初期値を指定することができます。

nextCommand (オプション)
データタイプ: EnumNextCommandMode
デフォルト: IMMEDIATELY
次のコマンドへ進む

```
TYPE EnumNextCommandMode : (  
    IMMEDIATELY                // 即座  
    , WHEN_COMMAND_DONE )      // コマンドの完了または中止  
                                // 後  
END_TYPE
```

戻り値

データタイプ: StructRetUnitDataSetCommand

戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。

これは、以下で構成されます。

- *functionResult*コンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]関クションで使用すると、データバックアップ関クションの現在の状態をチェックできます(特にステップ有効化条件IMMEDIATELYに有用)(「[_getStateOfUnitDataSetCommand]関クションを参照)。

データタイプStructRetUnitDataSetCommandおよびEnumDeviceUnitDataSetCommandについては、[_saveUnitDataSet]関クションの戻り値のセクションを参照してください([_saveUnitDataSet]関クションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

[_deleteAllUnitDataSets]関クション (ページ 365)

ユーザープログラムからのデータのバックアップの概要 (ページ 344)

[_exportUnitDataSet]関クション(カーネルV3.2以降) (ページ 352)

7.14.7 [_getStateOfUnitDataSetCommand]関クション

これは、データバックアップの関クションの状態を返します。

通知

```
_getStateOfUnitDataSetCommand (
    handle      : UDINT
): EnumDeviceUnitDataSetCommand
```

入力パラメータ

handle

データタイプ: DINT

状態をチェックするデータバックアップ関クションのハンドル

このハンドルは、データバックアップ関クションの戻り値のコンポーネントとして得られます。

戻り値

データタイプ: EnumDeviceUnitDataSetCommand

これにより、エラーおよび現在の状態に関する情報が提供されます。

データタイプEnumDeviceUnitDataSetCommandについては、[_saveUnitDataSet]ファクションの戻り値のセクションを参照してください([_saveUnitDataSet]ファクションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

[_exportUnitDataSet]ファクション(カーネルV3.2以降) (ページ 352)

ユーザープログラムからのデータのバックアップの概要 (ページ 344)

7.14.8 [_checkExistingUnitDataSet]ファクション

次の変数の保存値を含む指定されたデータレコードが記憶媒体に存在するかどうかを確認するチェックが行われます。

- ユニットのインターフェースまたは実装セクション(例: STソースファイル、MCCユニット)のバックアップされた非保持型または保持型ユニット変数。
- バックアップされた非保持型または保持型のグローバルデバイス変数。
- エクスポートされたユニットのインターフェースセクション(例: STソースファイル、MCCユニット)の非保持型または保持型ユニット変数。

このファクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_checkExistingUnitDataSet (  
    unitName      : STRING,  
    id            : UDINT,  
    storageType   : EnumDeviceStorageType,  
    { path        : STRING,  
      nextCommand : EnumNextCommandMode,  
    }  
): StructRetUnitDataSetCommand
```

入力パラメータ

unitName

データタイプ: STRING

ユニット名(例: STソースファイル、MCCユニット):

名前は小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

'_device'を指定すると、グローバルデバイス変数のデータセットがチェックされます(SIMOTION Kernelのバージョン3.2以降可能)。

id

データタイプ: UDINT

データセット数(ユニットあたり最大1_000_000データセット)。

storageType

データタイプ: EnumDeviceStorageType

データが保存される場所。

```
TYPE EnumDeviceStorageType : (  
    TEMPORARY_STORAGE          // 一時保管  
                                // (RAMディスク),  
                                // は、電源が故障した場合には削除されます  
    , PERMANENT_STORAGE        // 永続的保管  
                                // (メモリカード),  
                                // 電源故障でも保持  
    , USER_DEFINED )          // パス定義付き  
                                // (準備)  
END_TYPE
```

パス (オプション)

データタイプ: STRING

デフォルト: '' (空の文字列)

Destination path, if storageType = USER_DEFINED

準備として、初期値を指定することができます。

nextCommand (オプション)

データタイプ: EnumNextCommandMode

デフォルト: IMMEDIATELY

次のコマンドへ進む

```
TYPE EnumNextCommandMode : (  
    IMMEDIATELY          // 即座  
    , WHEN_COMMAND_DONE )    // コマンドの完了または中止  
                                // 後  
END_TYPE
```

戻り値

データタイプ: StructRetUnitDataSetCommand

戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。
これは、以下で構成されます。

- *functionResult*コンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]ファクションで使用すると、データバックアップファクションの現在の状態をチェックできます(特にステップ有効化条件IMMEDIATELYに有用)([_getStateOfUnitDataSetCommand]ファクションを参照)。

データタイプStructRetUnitDataSetCommandおよびEnumDeviceUnitDataSetCommandについては、[_saveUnitDataSet]ファクションの戻り値のセクションを参照してください([_saveUnitDataSet]ファクションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

[_exportUnitDataSet]ファクション(カーネルV3.2以降) (ページ 352)

7.14.9 [_deleteAllUnitDataSets]ファクション

次の変数の保存値を含むすべてのデータレコードは削除されます。

- ユニットのインターフェースまたは実装セクション(例: STソースファイル、MCCユニット)のバックアップされた非保持型または保持型ユニット変数。
- バックアップされた非保持型または保持型のグローバルデバイス変数。
- エクスポートされたユニットのインターフェースセクション(例: STソースファイル、MCCユニット)の非保持型または保持型ユニット変数。

このファクションを短縮形で呼び出す場合、すべてのパラメータ(すべてのオプションパラメータも含む)を指定する必要があります。

通知

```
_deleteAllUnitDataSet (  
    unitName      : STRING,  
    id            : UDINT,  
    storageType   : EnumDeviceStorageType,  
    { path        : STRING,  
      nextCommand : EnumNextCommandMode,  
    }  
): StructRetUnitDataSetCommand
```

入力パラメータ

unitName

データタイプ: STRING

ユニット名(例: STソースファイル、MCCユニット):

名前は小文字で入力し、単一引用符で括る必要があります(例: 'st_unit1')。

'_device'を指定すると、グローバルデバイス変数のすべてのデータセットが削除されます(SIMOTION Kernelのバージョン3.2以降可能)。

storageType

データタイプ: EnumDeviceStorageType

データセットを削除する場所。

```
TYPE EnumDeviceStorageType : (  
    TEMPORARY_STORAGE      // 一時保管  
                            // (RAMディスク),  
                            // は、電源が故障した場合には削除されます  
    , PERMANENT_STORAGE    // 永続的保管  
                            // (メモリカード),  
                            // 電源故障でも保持  
    , USER_DEFINED )      // パス定義付き  
                            // (準備)  
END_TYPE
```

パス (オプション)

データタイプ: STRING

デフォルト: '' (空の文字列)

Destination path, if storageType = USER_DEFINED

準備として、初期値を指定することができます。

nextCommand (オプション)

データタイプ: EnumNextCommandMode

デフォルト: IMMEDIATELY

次のコマンドへ進む

```
TYPE EnumNextCommandMode : (  
    IMMEDIATELY          // 即座  
    , WHEN_COMMAND_DONE ) // コマンドの完了または中止  
                                // 後  
END_TYPE
```

戻り値

データタイプ: StructRetUnitDataSetCommand

戻り値はデータタイプStructRetUnitDataSetCommandの構造体です。
これは、以下で構成されます。

- *functionResult*コンポーネント: EnumDeviceUnitDataSetCommand
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを[_getStateOfUnitDataSetCommand]ファクションで使用すると、データバックアップファクションの現在の状態をチェックできます(特にステップ有効化条件IMMEDIATELYに有用)([_getStateOfUnitDataSetCommand]ファクションを参照)。

データタイプStructRetUnitDataSetCommandおよびEnumDeviceUnitDataSetCommandについては、[_saveUnitDataSet]ファクションの戻り値のセクションを参照してください([_saveUnitDataSet]ファクションを参照)。

データバックアップについては、「ユーザープログラムからのデータバックアップおよびデータの初期化の使用 (ページ 385)」を参照してください

下記も参照

[_exportUnitDataSet]ファクション(カーネルV3.2以降) (ページ 352)

7.15 commandId用ファクション

7.15.1 [_getCommandId]ファクション

このファクションを使用して、全プロジェクトで一意のcommandIdを取得できます。これはコマンドの明示的なIDとして使用できます。

commandIdは必ず作成され、エラーフィードバックはありません。

通知

```
_getCommandId ( ) : CommandIdType
```

入力パラメータ

なし

戻り値

データタイプ: CommandIdType

コマンドの状態を追跡できる全プロジェクトで一意のCommandId

```
TYPE
  CommandIdType : STRUCT
    SystemId_low      : UDINT;    // 下位階層部
    SystemId_high     : UDINT;    // 上位階層部
  END_STRUCT
END_TYPE
```

下記も参照

正しいcommandIdパラメータの使用 (ページ 437)

7.15.2 [_getSyncCommandId]ファクション

このファクションを使用して、全プロジェクトで一意のsyncCommandIdを取得できます。
このIDはシステムファクション[BEGIN_SYNC]および[_startSyncCommand]に転送して、モーションシーケンスを同期的に起動できます(SIMOTIONデバイスのパラメータマニュアルを参照)。
syncCommandIdは必ず作成され、エラーフィードバックはありません。

通知

```
_getSyncCommandId ( ) : CommandIdType
```

入力パラメータ

なし

戻り値

データタイプ: CommandIdType
コマンドの状態を追跡できる全プロジェクトで一意のsyncCommandId

```
TYPE  
CommandIdType : STRUCT  
    SystemId_low      : UDINT;    // 下位階層部  
    SystemId_high     : UDINT;    // 上位階層部  
END_STRUCT  
END_TYPE
```

下記も参照

正しいcommandIdパラメータの使用 (ページ 437)

7.16 待機時間の定義

7.16.1 [_waitTime]ファクション

このファクションは、この呼び出しで指定された時間が経過するまで、このファクションをトリガするタスクを中断させます。

通知

このファクションは、MotionTasksのみで使用してください。これを周期的タスクで使用すると、時間監視エラーにつながります!

- SynchronousTasksでは SIMOTION Kernel V3.2以降では、時間監視を中断するかを設定することができます。デフォルトでは、時間監視は有効です。
IPOsynchronousTaskでは、これ以外に以下を考慮します。
UserInterruptTasksは、トリガ事象では開始されなくなっています!
- 他の周期的タスク(BackgroundTask、TimerInterruptTasks)では時間監視は常に有効です。

周期的タスクでは、タイマーシステムファクションブロックを使用して(「タイマー」を参照)待機時間を実装します。

¹ SIMOTION

KernelのバージョンV3.1まででは、SynchronousTasksでは時間監視は中断されます。

[_waitTime]ファクションは常に実行可能で、戻り値= 0です。

通知

```
_waitTime (
    timeValue      : TIME          // 待機時間
) : DINT           // 常に0
```

入力パラメータ

timeValue

データタイプ: TIME

処理中のタスクを中断する時間を示します。

戻り値

データタイプ: DINT
常に0です。

下記も参照

タイマー (ページ 422)
ラウンドロビン実行レベルでの時間割り当て (ページ 187)
タスクを指定した時間だけ待機させる (ページ 242)
周期的タスクでの待機時間 (ページ 436)

7.17 デバイス固有のファンクション

7.17.1 [_getDeviceId]ファンクション

このファンクションにより、ハードウェア情報ブロックからSIMOTIONデバイスのハードウェアIDを読み取ります。
このファンクションを呼び出すときに入力パラメータとして読み込むIDのタイプを指定します。

通知

```
_getDeviceId (
    idType          : EnumDeviceIdType
)                  : StructRetGetDeviceId0
```

入力パラメータ

idType

データタイプ: EnumDeviceIdType
読み出すIDの指定

```
TYPE EnumDeviceIdType : (
    SERIAL_NUMBER )    //シリアル No.
    , HW_TYPE           // ハードウェアタイプ
    , SPECIFIC_NUMBER ) //
END_TYPE
```

戻り値

データタイプ: StructRetGetDeviceId

戻り値は、データタイプStructRetGetDeviceIdの構造体です。
これは、以下で構成されます。

- *functionResult*コンポーネント: DINT.
これは、エラーに関する情報を提供します。
- *id* コンポーネント: STRING[254]
これには、メモリカードの読み出しハードウェアIDが含まれます。

```
TYPE StructRetGetDeviceId : STRUCT
    functionResult : DINT;          // エラーステータス
                                     // 0: エラーなし
                                     // <> 0: エラー
    id : STRING[254];              // ハードウェアIDの読み取り
END_STRUCT;
END_TYPE
```

7.17.2 [_getMemoryCardId]ファクション

このファクションにより、ハードウェア情報ブロックからメモリカードのハードウェアIDを読み取ります。
このファクションを呼び出すときに入力パラメータとして読み込むIDのタイプを指定します(現時点ではシリアル No.のみ可能)。

通知

```
_getMemoryCardId (
    idType          : EnumMemoryCardIdType
)                  : StructRetGetMemoryCardId0
```

入力パラメータ

idType
データタイプ: EnumMemoryCardIdType
読み出すIDの指定

```
TYPE EnumMemoryCardIdType : (
    SERIAL_NUMBER )      // シリアル No.
END_TYPE
```

戻り値

データタイプ: StructRetGetMemoryCardId

戻り値は、データタイプStructRetGetMemoryCardIdの構造体です。
これは、以下で構成されます。

- *functionResult*コンポーネント: DINT.
これは、エラーに関する情報を提供します。
- *id*コンポーネント: STRING[254]
これには、メモ리카ードの読み出しハードウェアIDが含まれます。

```
TYPE StructRetGetMemoryCardId : STRUCT
    functionResult : DINT;           // エラーステータス
                                     // 0: エラーなし
                                     // <> 0: エラー
    id : STRING[254];               // ハードウェアIDの読み取り
END_STRUCT;
END_TYPE
```

7.17.3 [_setDeviceErrorLED]ファクション

このファクションを使用して、SIMOTIONデバイスのテクノロジー/オプションオブジェクトのライセンス未取得エラーを設定します。
SIMOTIONデバイス上の対応するLEDは赤色に点滅します(SIMOTIONデバイスのマニュアルを参照)。

通知

```
_setDeviceErrorLED ( ) :DINT
```

入力パラメータ

なし

戻り値

データタイプ:	DINT
0	エラーなし
<> 0	不良

7.18 データタイプまたは変数のメモリサイズの設定

7.18.1 [_sizeof]ファクション

このファクションは、変数またはデータタイプに必要なメモリサイズを、バイト単位で定数値として返します。
また、これはデータタイプおよび変数の宣言にも使用できます(配列の大きさなど)。

通知

```
_sizeof (  
    in  : ANY    // データタイプまたは変数の  
                // 識別子  
    )    : DINT
```

入力パラメータ

in
データタイプ: ANY
サイズを特定する変数またはデータタイプの識別子。

戻り値

データタイプ: DINT
必要なメモリサイズ(バイト単位)。
メモリサイズは適切なレイアウトを考慮して指定します。つまり、メモリのデータタイプの現実的な割り当てに従います。
したがって、ARRAYでは使用するデータタイプに必要な、効率的なサイズを指定します。
実際に必要なサイズは小さくなる場合があります。
例:

```
TYPE
  a_type      : STRUCT
    a          : LREAL;    // 8バイト
    b          : BOOL;     // 1バイト
  END_STRUCT;
END_TYPE
//..
x := _sizeof (a_type);      // 値16が取得されます
```


7.19 使用可能な追加のファンクション

SIMOTIONでは、SIMOTIONデバイスおよびテクノロジーオブジェクトなどにより導入される、追加のシステムファンクションを使用できます。
次の表に、これらのファンクションの説明のある箇所を示します。

表 7-16 追加のSIMOTION
STのシステムファンクションおよびシステムファンクションブロックの概要

システムファンクション	説明
テクノロジーオブジェクトのシステムファンクション	『SIMOTION Camテクノロジーパッケージ、システムファンクションパラメータマニュアル』(リファレンスリスト) 『SIMOTION TControlテクノロジーパッケージパラメータマニュアル』(リファレンスリスト) また、テクノロジーオブジェクトの機能マニュアルも参照してください。
SIMOTIONデバイスのシステムファンクション	SIMOTIONデバイスのパラメータマニュアル(リファレンスリスト)
PLCopen規格に準拠した軸制御用システムファンクション	『SIMOTION Camテクノロジーパッケージ、システムファンクションパラメータマニュアル』(リファレンスリスト)
I/Oモジュールおよびドライブコンポーネントの制御用標準ファンクション	対応するI/Oモジュールおよびドライブコンポーネントのパラメータマニュアル(リファレンスリスト)

7.20 特定のシステムファンクションの適用

7.20.1 メッセージのプログラム

7.20.1.1 一般

次のファンクションを使用して、故障メッセージなどのメッセージをプログラムしたり状態をチェックすることができます。

- [`_alarmSId`] (確認応答なしのメッセージの生成)
- [`_alarmSId`] (確認応答付きのメッセージの生成)
- [`_alarmScId`] (メッセージ状態に関するクエリー)

ただし、前提条件としてアプリケーション固有の設定されたメッセージ名が必要です。

注記

[`_writeAndSendMessage`] ファンクションを使用して、診断バッファにユーザー定義のエントリを作成できます。

このファンクションの詳細については、SIMOTIONデバイスのパラメータマニュアルを参照してください。

7.20.1.2 ファンクションの概要

ここで説明するファンクションはSIMOTION KernelのバージョンV3.2以降で使用可能です。これらはライブラリで使用できます。

それぞれの呼び出し時に[`_alarmSId`]は**確認応答が不要なメッセージ**を生成します。メッセージは信号によってトリガされ、外部値を付加することができます。メッセージは、表示用に登録されたすべての表示デバイスに伝送されます。

それぞれの呼び出し時に[`_alarmScId`]は**確認応答が必要なメッセージ**を生成します。メッセージは信号によってトリガされ、外部値を付加することができます。メッセージは、表示用に登録されたすべての表示デバイスに伝送され、そのデバイスで確認応答できます。

ファンクションの入力信号として次を使用します。

1. メッセージをトリガする信号

これは次のように解釈されます。

- 信号が立ち上がりエッジ(このメッセージ名による最後の呼び出しに対して)である場合、受信メッセージが生成されます。
このメッセージ名での最初の呼び出しの信号状態がTRUEの場合、受信メッセージも生成されます。
- 信号が立ち下がりエッジ(このメッセージ名による最後の呼び出しに対して)である場合、送信メッセージが生成されます。

2. コンパイルするメッセージ

全プロジェクトで一意的アラームIDを使用して指定します。

アラームIDについての詳細は、下の関連するセクションを参照して下さい。

3. オプションで補助値(メッセージコンフィグレーションで補助値が指定された場合)。

[_alarmScId]ファンクションを使用して、メッセージの状態およびその確認応答状態をクエリーします。ここでは2つのシナリオがあります。

メッセージは一意的アラームIDによって指定します。

ファンクションの正式な構造については、「[_alarmSId]」および「[_alarmScId]」を参照してください。

設定されたメッセージ名を使用してメッセージを指定する同様のファンクションは、SIMOTION

KernelのバージョンV3.0以前で使用可能です。これらのファンクションはライブラリで使わないでください。

アプリケーション固有の設定されたメッセージ名の詳細は、オンラインヘルプを参照してください。

注記

送信メッセージは受信メッセージの後に生成する必要があります。そうしないと故障メッセージが出力されます。

設定されたメッセージ名を伝送するコマンドは、ショートフォームでのみ呼び出すことができます(つまり、すべてのパラメータ値の完全なリストで指定し、仮パラメータは指定しません)。

補助値(オプション)は、基本データタイプの変数である必要があります。ファンクションの呼び出しに定数値を使用しないでください。

7.20.1.3 アラームID

生成するメッセージは、[_alarmSId]、[_alarmScId]、および[_alarmScId]ファンクションのアラームIDで指定します。

設定したメッセージ名に対するアラームIDを、以下の方法で取得することができます。

- 変数_alarm.nameとして。ただし、nameはSIMOTION SCOUTで設定されたメッセージ識別子。
- _getAlarmId(name)ファンクションを使用して。

下記も参照

ファンクション_alarmSId (ページ 277)

ファンクション_getAlarmId (ページ 285)

7.20.1.4 AlarmSのバッファ管理

説明

AlarmSメッセージでは、40のバッファ領域を持つメッセージリストを使用できます。AlarmSメッセージは、そのIDとともにメッセージリストに入力されます。それぞれの送信AlarmSに対して、同じIDを持つ受信AlarmSがメッセージリストに存在している必要があります。総計40のリストエントリのそれぞれに、送信バッファもあります。この送信バッファは、登録されているクライアント(HMIまたはSIMOTION SCOUT)への通知を編成するために使用されます。

メッセージリストおよび送信バッファ

メッセージリストおよび送信バッファは、次のように使用されます。

戻り値		重要度
受信AlarmSの戻り値(立ち上がりエッジによるファンクション呼び出し)		
	16#8002	メッセージリストにすでにメッセージがあります。AlarmSはメッセージリストに入力されませんでした。
	16#8003	このIDの送信alarmSがまだメッセージリストに残っており、送信バッファが占有されています。AlarmSはメッセージリストに入力されませんでした。
	16#8004	このIDの受信AlarmSはすでにメッセージリストにあります。AlarmSはメッセージリストに入力されませんでした。
	16#0000	AlarmSのIDがメッセージリストに入ります。関連付けられた送信バッファが割り当てられます。システムファンクションは戻り値0000を返します。登録されているクライアントに通知されます。クライアントに正常に通知されると、送信バッファは再度解放されます。AlarmSのIDはAlarmSを受信するメッセージリストに残ります。
送信AlarmSの戻り値(立ち下がりエッジによるファンクション呼び出し)		
	16#8003	このIDの送信alarmSがまだメッセージリストに残っており、送信バッファが割り当てられています。AlarmSはメッセージリストに入力されませんでした。

戻り値		重要度
	16#8004	このIDの送信AlarmSはすでにメッセージリストにあります。AlarmSはメッセージリストに入力されませんでした。
	16#8007	このIDの受信エントリが見つかりませんでした。AlarmSはメッセージリストに入力されませんでした。
	16#0000	そのIDの受信AlarmSがメッセージリストにあり、送信バッファが割り当てられていない場合、このエントリは送信AlarmSで上書きされます。 関連付けられた送信バッファ割り当てられます。 システムは戻り値0000を返します。 登録されているクライアントに通知されます。 クライアントに正常に通知されると、送信バッファは再度解放されます。 メッセージリストのエントリは削除されます。

7.20.1.5 メッセージ生成の例

表の例では温度をチェックし、温度が高すぎる場合、確認応答が不要の受信メッセージを生成します(*Temperature too high, incoming*など)。

温度が指定された最大値まで低下すると、送信メッセージが生成されます(受信メッセージは消滅します)。

*SCOUT_alarm_name*という名前のメッセージがSIMOTION SCOUTに設定され、メッセージは「*Temperature too high: @1!%2d@degrees.*」などとなります。

状態変数を使用して、同じメッセージを繰り返さないようにします。

*handleAlarm*プログラムはBackgroundTaskに割り当てられます。

表 7-17 メッセージ生成の例

```

INTERFACE
  PROGRAM handleAlarm;
END_INTERFACE

IMPLEMENTATION
  PROGRAM handleAlarm
    VAR
      retVal          : DWORD;      // 戻り値
      temperature      : INT;        // チェックする状態
      maxTemperature : INT := 60;    // 状態の比較値

```

```

    mySignal : BOOL := FALSE;    // 信号の状態:yes/no
END_VAR
//...
IF temperature > maxTemperature THEN
    IF mySignal = FALSE THEN
        // 受信メッセージ、確認応答不要。
        retVal := _alarmSid (
            Sig := TRUE,
            Ev_id := _alarm.SCOUT_alarm_name,
            Sd := temperature);
        mySignal := TRUE;
    END_IF;
ELSE
    IF mySignal = TRUE THEN
        // 送信メッセージ、確認応答不要。
        retVal := _alarmSid (
            Sig := FALSE,
            Ev_id := _alarm.SCOUT_alarm_name,
            Sd := temperature);
        mySignal := FALSE;
    END_IF;
END_IF;
//...
END_PROGRAM

```

7.20.1.6 故障番号とメッセージの状態のチェック(戻り値のフィルタ)

`[_alarmSid]`および`[_alarmScId]`ファンクションの戻り値には故障番号が含まれおり、これにより実行時にエラーが発生したかどうか示されます。
多くのシステムファンクションと同様に、戻り値=0は実行時エラーがなかったことを示します。

ただし、`[_alarmScId]`ファンクションの戻り値はエラー番号とメッセージの状態の両方が示されます。

そのため、このファンクションで状態をチェックする場合、最初にALARMS_ERROR定数(= 16#8000)で戻り値をフィルタする必要があります。

そうすることで、ファンクションの実行中にエラーが発生したかどうかを確認できます。

フィルタと故障番号は、AND演算で組み合わせたときに真となるように選びます。

エラーが発生しなかった場合、メッセージの状態を評価できます。

故障番号とメッセージ状態の全リストは、「メッセージプログラミング用ファンクション」にあります。

これにより、`[_alarmScId]`コマンドが発行されたときに、次のようにエラーをチェックできます(データタイプDWORDの`retVal`変数には、ファンクションの戻り値が含まれます)。

表 7-18 エラーチェックの例

```

retVal := _alarmScId (Ev_id := _alarm.SCOUT_alarm_name);
// ここでエラーをチェックします
IF (retVal AND ALARMS_ERROR) <> 0 THEN
    // 条件が合致。エラーが発生。
    IF retVal = (ALARMS_ERROR OR

```

```
        DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID) THEN
        ; // メッセージ番号不正
    END_IF;
ELSE
    // 条件が合致しない。エラーが発生していない。
    // メッセージおよび確認応答状態のチェック
    IF retVal = 16#0000 THEN
        ; // メッセージが消失。確認応答なし
        ELSIF retVal = ALARMS_STATE THEN
        ; // メッセージが到着。確認応答なし
        ELSIF retVal = 16#0010 THEN
        ; // メッセージが存在しない
        ELSIF retVal = (ALARMS_QSTATE OR ALARMS_STATE) THEN
        ; // メッセージが到着。確認応答あり
    END_IF;
//...
END_IF;
```

注記

エラーのチェックには、定数値も記号定数も同様に使用できます。「メッセージプログラミング用ファンクション」を参照してください。

7.20.2 変数の一貫した読み取りおよび書き込み(セマフォ)

7.20.2.1 一貫したデータアクセス

基本データタイプ(「基本データタイプ」のセクションを参照)の変数へのすべてのアクセスは、システムにより一貫して管理されます。

システムでは、これらの変数の処理中に変更されないように保証されています。

派生データタイプのグローバル変数にアクセスする場合(「ユーザー定義データタイプ(UDT)」を参照)、複数のタスクで同じ変数(記号I/O変数、SIMOTIONデバイスのシステム変数、テクノロジーオブジェクトのシステム変数、グローバルデバイス変数、およびユニット変数。「変数モデル」のセクションを参照)にアクセスする際に、ユーザーはデータの一貫性を確保する必要があります。

派生データタイプのローカル変数へのアクセスは、常に一貫性があります。これは、それらが定義されているプログラム(またはファンクションまたはファンクションブロック)の内部でのみ使用可能なためです。

注記

タスク内で常に一貫したデータへのアクセスが確実に行われるようになります。

7.20.2.2 セマフォ

グローバル変数の読み書きに一貫性を確保するには、セマフォを使用します。

データタイプDINTのグローバル変数(*semaA*など)がセマフォとして使用されます。変数が配列の要素の場合は、コンパイル時にインデックスを指定する必要があります(例: *a[2]*)。

以下のファンクションを使って、セマフォのステータスを変更およびテストします。

- `_testAndSetSemaphore (sema : DINT) : BOOL`

このファンクションを使用して、セマフォが設定されているかチェックします。

- 戻り値TRUE: セマフォは有効です。
- 戻り値FALSE: セマフォは設定されています。

ファンクション終了時には、セマフォは必ず設定されます。 [`_releaseSemaphore (semaA)`]ファンクションが呼び出されるまで、このファンクションの追加の呼出しは(他のプログラムからのものも)値FALSEを返します。

- `_releaseSemaphore (sema: DINT) : VOID`

ファンクションの正式な構造については、「*変数の処理*」のセクションを参照してください。

セマフォは解放されます。

グローバル変数への一貫したデータアクセスは、以下の条件で確実に行うことができます。

1. すべてのタスクが、グローバル変数へのアクセスをセマフォを設定することで示すこと。
2. すべてのタスクが、セマフォが有効化された場合のみグローバル変数にアクセスすること。

7.20.2.3 例: セマフォによる一貫したデータアクセス

次の表に、データを読み取るプログラムとデータを書き込むプログラムでのセマフォの使用例を示します。

表 7-19 セマフォを使用してグローバル変数に一貫したアクセスを確保する例

```
IMPLEMENTATION
VAR_GLOBAL
    myArray : ARRAY [0..1] OF DINT;
    semaA : DINT;
END_VAR

PROGRAM Writer
    // 一貫性のある変数の書き込み
    IF _testAndSetSemaphore(sema := semaA) THEN
        myArray[0] := 18;
        myArray[1] := 19;
        _releaseSemaphore(sema := semaA);
```



```
// TRUEでセマフォを必ず解放すること
// クエリの回路：これは次のことを確実にを行う
// リセットされた場合のみ
// 解放する。
ELSE
    ; // エラー処理
END_IF;
// _releaseSemaphore(sema := semaA);
// この時点での解放は不正となる。
// セマフォは常に解放される。
END_PROGRAM
PROGRAM Reader
    VAR
        var0          : DINT;
        var1          : DINT;
    END_VAR

    // 一貫性のある変数の読み取り
    IF _testAndSetSemaphore(sema := semaA) THEN
        var0 := myArray[0];
        var1 := myArray[1];
        _releaseSemaphore(sema := semaA);
        // TRUEでセマフォを必ず解放すること
        // クエリの回路：これは次のことを確実にを行う
        // リセットされた場合のみ
        // 解放する。
    ELSE
        ; // エラー処理
    END_IF;
    // _releaseSemaphore(sema := semaA);
    // この時点での解放は不正となる。
    // セマフォは常に解放される。
END_PROGRAM
END_IMPLEMENTATION
```

7.20.3 ユーザープログラムからのデータのバックアップおよび初期化

7.20.3.1 ユーザープログラムからのデータバックアップとデータ初期化 - 機能と手順

ユーザープログラムから、次のデータセットの変数の値を保存、ロード、初期化できます。

- ユニットのインターフェースの非保持型または保持型ユニット変数または実装セクション(例: STソースファイル、MCCユニット)

- 非保持型または保持型グローバルデバイス変数。

これには、次のファンクションを使用できます。.....

- `[_saveUnitDataSet]`: 値はバイナリデータセットとして保存されます。

- `[_loadUnitDataSet]`:

`[_saveUnitDataSet]`で保存されたバイナリデータセットから値をロードします。

データセットが保存されてから、ロードするすべてのデータセグメントのバージョンIDが変更されていない場合のみ、データセットをロードできます。

次の事項にも注意してください。

- `[_exportUnitDataSet]` (SIMOTION KernelのバージョンV3.2以降):
値はXMLフォーマットでZIPアーカイブとしてエクスポートされます(ファイル名は*.dat)。

- `[_importUnitDataSet]` (SIMOTION KernelのバージョンV3.2以降):

`[_exportUnitDataSet]`によりXMLフォーマットでZIPアーカイブとしてエクスポートされたデータセット(ファイル名は*.dat)から、値をインポートします。

データセットが保存されてから、ロードするデータセグメントのバージョンIDが変更されていても、データセットをインポートできます。

- すでに存在しない変数は無視されます。
- 追加された変数の値は、変更されないままになります。
- 変更されたデータタイプを持つ変数の場合、新規のデータタイプに変換することができる場合は、値が伝送されます。変更できない場合は、変数の値は保持されます。

- `[_deleteUnitDataSet]`: データレコードを1行削除します。

- `[_checkExistingUnitDataSet]`:

指定されたデータセットが記憶媒体に存在するかどうかをチェックします。

- `[_deleteAllUnitDataSets]`: すべてのデータセットを削除します。

- `[_resetUnitData]` (SIMOTION KernelのバージョンV3.2以降): 変数の値を初期化します。

たとえば、データセットをインポートする前に`[_importUnitDataSet]`により関連するデータセグメントを初期化して、変数に不要な値を入れないようにできます。

データセグメントおよびバージョンIDについては、「*変数初期化の時間*」のセクションの表「*グローバル変数のバージョンIDおよびダウンロード時の初期化*」を参照してください。

SIMOTIONデバイスのシステムファンクションのパラメータマニュアル(リファレンスリスト)に`[_resetUnitData]`の説明があります。

ここで、パラメータを詳細に説明します。

通知

データセグメントのデータ構造をユニットで変更する場合、またはグローバルデバイス変数で変更する場合、SIMOTIONデバイスへのプロジェクトのロードに次が適用されます。

- 一時的にバックアップされたすべてのデータセットは削除されます。
- (`[_saveUnitDataSet]`により)恒久的に保存されたすべてのデータセットは、このデータセグメントに対して読み取りができません。

デバイス上に最大16のバックアップを同時に実行できます。

保存できるデータセットの個数(最大1_000_000)は、使用可能なメモリ空間により異なります。

バックアップまたはエクスポートするデータの整合性に注意してください(「*変数の読み取りおよび書き込みの整合性(セマフォ)*」のセクションを参照)。

注記

SIMOTIONデバイスから `[_saveUnitDataSet]` または `[_exportUnitDataSet]` で保存したデータセットをアップロードできます。SIMOTION

SCOUTで**変数の保存**ファンクションを使用します。

`[_saveUnitDataSet]` で保存されたデータセットは、自動的にXMLフォーマットに変換されます。

変数の回復ファンクションを使用して、SIMOTIONデバイスにこれらのデータレコードおよび変数をダウンロードすることができます。

そのためには、プロジェクトナビゲータで関連するSIMOTIONデバイスを選択して、コンテキストメニューからファンクションを選択します。詳細については、『SIMOTION SCOUT設定マニュアル』を参照してください。

これにより、たとえば、変数がプロジェクトのダウンロードにより初期化されたり、(SIMOTION SCOUTのバージョンの変更などのため)使用不能になったりした場合でも、それらを取得できます。

7.20.3.2 入力パラメータ

次に、最も重要なパラメータについて簡潔に説明します。

個々のファンクションおよびそのパラメータの詳細は、機能マニュアル『*SIMOTION基本機能*』を参照してください。

- *unitName*: ユニットの名前(STソースファイル、MCCユニットなど)

'*device*'が指定されると、データバックアップファンクションはグローバルデバイス変数に適用されます(SIMOTION KernelのバージョンV3.2以降可能)。

- *id*: データレコード番号

インデックスとしてデータセット番号を指定すると、ユニット変数の複数のバージョンを保存して、選択的にダウンロードできます。

$id < 1\_000\_000$

- *storageType*: メモリの場所を指定して、次を選択できます。

- TEMPORARY_STORAGE:

データはRAMディスクに保存されます(電源異常があると削除されます)

- PERMANENT_STORAGE:

データはメモリカードに保存されます(電源異常があっても保持されます)

- *nextCommand*: ステップの有効化条件

ステップの有効化条件を指定する場合、次のオプションを選択できます。

- STソースファイルの次のコマンドを即座に実行する(IMMEDIATELY)
- コマンドが終了するまで待機する(WHEN_COMMAND_DONE)

ステップ有効化条件の詳細は、本章の関連する説明を参照してください。

- *dataScope*

このパラメータはSIMOTION KernelのバージョンV3.2以降で使用可能です。

これにより、データバックアップファンクションを適用するユニットのセクションを選択できます。

- *_INTERFACE*:
ファンクションはユニットのインターフェースセクションに適用されます
- *_IMPLEMENTATION*: ファンクションはユニットの実装セクションに適用されます
- *_INTERFACE_AND_IMPLEMENTATION*:
ファンクションはユニットのインターフェースセクションおよび実装セクションに適用されます

unitName =

'_device'を指定した場合、*dataScope*には値*_INTERFACE*または*_INTERFACE_AND_IMPLEMENTATION*のみが使用できます。

SIMOTION

KernelのバージョンV3.1以前では、データバックアップファンクションはユニットのインターフェースセクションの保持されない変数にしか適用できませんでした。

- *KindOfData*:

このパラメータはSIMOTION KernelのバージョンV3.2以降で使用可能です。

これにより、データバックアップファンクションを、保持されるグローバル変数または保持されないグローバル変数に適用するか選択できます。

- *NO_RETAIN_GLOBAL*:
ファンクションは保持されないグローバル変数に適用されます
- *_RETAIN*: ファンクションは保持されるグローバル変数に適用されます
- *ALL_GLOBAL*:
ファンクションは保持されるグローバル変数および保持されないグローバル変数に適用されます

保持される変数または保持されない変数がデータセットに格納されている場合、保持されている変数または保持されていない変数を選択的にロードまたはインポートできます。

SIMOTION

Kernelのバージョン3.1以前では、データバックアップファンクションはユニットのインターフェースセクションの保持されない変数にしか適用できませんでした。

7.20.3.3 戻り値

戻り値はデータタイプ *StructRetUnitDataSetCommand* の構造体です。
これは、以下で構成されます。

- *functionResult* コンポーネント: *EnumDeviceUnitDataSetCommand*
これにより、エラーおよび現在のステータスに関する情報が提供されます。
- *handle* コンポーネント: UDINT
これを *_getStateOfUnitDataSetCommand* ファクションで使用すると、データバックアップファクションの現在の状態をチェックできます(これはステップ有効化条件 IMMEDIATELY に必要です)。

7.20.3.4 保存場所およびメモリ要件

保存場所は入力パラメータ *storageType* で定義します。

- TEMPORARY_STORAGE: データセットはRAMディスクに保存されます。
- PERMANENT_STORAGE: データセットはメモリカードに保存されます。

保存できるデータセットの個数は、個々のメモリの場所の使用可能なメモリ空間により異なります。使用可能なメモリ空間に関する情報は、**[System utilization]** タブのデバイス診断によって取得できます(オンラインヘルプを参照)。

通知
使用可能なすべてのメモリ空間をデータセットでいっぱいにしないでください。いっぱいになると、プロジェクトをターゲットシステムにダウンロードしたり、プロジェクトデータが増加したときにRAMからROMにコピーしたりすることができなくなります。

次の情報により、(*_saveUnitDataSet* で保存される)バイナリデータセットに必要なメモリ空間を見積もることができます

- 基本データタイプは、メモリ上で決められたデータ幅を占有します(「基本データタイプ」セクションの表「基本データタイプのビット幅および数値の範囲」を参照。データタイプ BOOL は1バイトを占有)。
- さらに次の用途でメモリ空間が必要です。
 - ワードまたはダブルワードの境界へのメモリアドレスの適合
 - 適合性情報(データセットごとに約100バイト)
 - ファイルシステムの一般的な補足データ(セクタヘッダー、ディレクトリ、セクタ全体のみの占有など)

(*_exportUnitDataSet* により)XMLフォーマットでエクスポートされたデータセットの場合、メモリ要件は非常に高くなり、この方法では確定できません。

7.20.3.5 ステップの有効化条件

ステップ有効化条件は、入力パラメータ *nextCommand* で示されます。
これが WHEN_COMMAND_DONE に設定されると、ST ソースファイルの次のコマンドはファンクションが終了してから実行されます(同期実行)。

戻り値の *functionResult* コンポーネントには実行されたファンクションの結果が含まれます(例を参照)。

表 7-20

ステップ有効化条件 WHEN_COMMAND_DONE でのデータバックアップファンクションの呼び出し

```
VAR_GLOBAL
  ds_ret : StructRetUnitDataSetCommand;
  error : BOOL := FALSE;
END_VAR

PROGRAM save_data_seq
  // プログラムは逐次的タスクに割り当てられます。
  // ファンクションは同期的に実行されます。
  ds_ret := _loadUnitDataSet (
    unitName := 'ds3',
    id := 1,
    storageType:= TEMPORARY_STORAGE,
    nextCommand := WHEN_COMMAND_DONE);
  // ファンクションの終了、結果の評価
  IF (ds_ret.functionResult <> DONE) THEN
    error := TRUE;    // エラー
  END_IF;
END_PROGRAM
```

この手続きは主に逐次的タスクで使用されます。

このファンクションを実行すると、非常に時間がかかる場合があります。
したがって、時間監視機能は周期的タスク(BackgroundTask など)について応答する場合があります。

そのため、*nextCommand* パラメータを IMMEDIATELY に設定して、ファンクションを非同期的に実行することもできます。

この場合、ファンクションが開始されてすぐに、ソースの次のコマンドが処理されます。

戻り値から次のことがわかります。

- 起動が成功したかどうか(コンポーネントの *functionResult* = DONE)
- 詳細状態クエリー用ハンドル(ハンドルコンポーネント)

コマンドの起動が成功した場合、結果が ACTIVE 以外のものになるまで *_getStateOfUnitDataSetCommand* ファンクションとハンドルを使用してデータバックアップファンクションの現在の状態をチェックする必要があります(例を参照)。

表 7-21

ステップ有効化条件IMMEDIATELYによるデータバックアップファンクションのファンクション呼び出し

```
VAR_GLOBAL
  error : BOOL := FALSE;
  ds_rslt : EnumDeviceUnitDataSetCommand;
  ds_ret : StructRetUnitDataSetCommand;
  cmd_busy : BOOL := FALSE;
  cmd_done : BOOL := FALSE;
END_VAR

PROGRAM save_data_cycl
  // プログラムは周期的タスクに割り当てられます。
  IF NOT cmd_busy THEN
    cmd_busy := TRUE;
    // ファンクションは非同期的に実行されます。
    ds_ret := _saveUnitDataSet (
      unitName := 'ds1',
      id := 1,
      storageType:= TEMPORARY_STORAGE,
      overwrite := TRUE,
      nextCommand:= IMMEDIATELY);
    IF (ds_ret.functionResult <> DONE) THEN
      cmd_busy := FALSE;
      error := TRUE; // ファンクションの起動が失敗
      // (サービスが多すぎるなど)
    END_IF;
  ELSE
    // ファンクションの実行中、結果の待機
    ds_rslt := _getStateOfUnitDataSetCommand (
      ds_ret.handle);
    IF (ds_rslt <> ACTIVE) THEN
      cmd_busy := FALSE;
      IF (ds_rslt = DONE) THEN
        cmd_done := TRUE; // ファンクションは正常に
        // 終了
      ELSE
        error := TRUE; // ファンクションは失敗
      END_IF;
    END_IF;
  END_IF;
END_PROGRAM
```

7.20.4 任意のデータタイプとバイト配列との間の変換(マーシャリング)

任意のデータタイプの変数とバイト配列との間での変換は、異なるデバイス間でデータ交換用に定義する伝送フォーマットを作成するために頻繁に使用されます(「通信ファンクション」を参照)。

次のファンクションを使用して、任意のデータタイプ(基本データタイプ、テクノロジーパッケージおよびデバイスの標準データタイプ、およびユーザー定義データタイプ)の変数をバイト配列に変換したり、その逆に変換することができます。

- AnyType_to_BigByteArray
- AnyType_to_LittleByteArray
- BigByteArray_to_AnyType
- LittleByteArray_to_AnyType

(「[BigByteArray_to_AnyType]ファンクション」、「[LittleByteArray_to_AnyType]ファンクション」、および「[BigByteArray_to_AnyType]ファンクション」、「[LittleByteArray_to_AnyType]ファンクション」のセクションを参照)

すべてのファンクションにおいて、オプションでオフセットを指定して、バイト配列の最初の要素を割り当てたり評価することができます。

区切りは次のように行います。

- 変換の方向(バイト配列への変換またはバイト配列からの変換)
- 配列内のバイト順序(表を参照)
 - ビッグエンディアン: 最上位バイトをメモリの下位アドレスに配置 (Motorola, SUN Sparc, SIMATIC S7)
 - リトルエンディアン: 最下位バイトをメモリの下位アドレスに配置 (Intel, DEC Alpha)

表 7-22 バイト順序の例(ビッグエンディアンおよびリトルエンディアン)

アドレス	数値34677374 = 16#2_11_22_7E (データタイプUDINT)		キャラクタ文字列"Byte" (データタイプDWORD)	
	ビッグエンディアン	リトルエンディアン	ビッグエンディアン	リトルエンディアン
2#...11	16#7E = 126	16#02 = 2	16#65 = "e"	16#42 = "B"
2#...10	16#22 = 34	16#11 = 17	16#74 = "t"	16#79 = "y"
2#...01	16#11 = 17	16#22 = 34	16#79 = "y"	16#74 = "t"
2#...00	16#02 = 2	16#7E = 126	16#42 = "B"	16#65 = "e"

通知

TOデータタイプは変換できません(TOデータタイプの変換については、「テクノロジーオブジェクトデータタイプの変換」のセクションを参照してください)。

構造体および配列を変換する場合、基本変数レベルでのみ一貫性が保証されます。上位レベルの一貫性はユーザーが確保する必要があります(「変数の読み取りおよび書き込みの整合性(セマフォ)」および「変数の処理」を参照)。

注記

次のデータタイプの変換は可搬性がありません。つまり、システム間で伝送フォーマットが定義されていません。SIMOTIONデバイス間の変換は、何の制限もなく可能です。

- 時刻タイプ: 変換フォーマットについては、表を参照してください。
- 列挙子(列挙データタイプ)

これらのデータタイプを変換する場合、コンパイラで警告が出力されます(16013)。

表 7-23 マーシャリングファンクションの時刻データタイプの変換フォーマット

データタイプ	変換フォーマット
TIME	時間の仕様、ms単位(UDINT)
TIME_OF_DAY (TOD)	TOD#0:0:0で開始する時刻、ms単位(UDINT)
DATE	DATE#1992_01_01からの日数(UDINT)。 DATE#1992_01_01は1となります。
DATE_AND_TIME (DT)	指定された順序でのTODとDATEの組み合わせ

通知

プログラムの実行中にマーシャリングファンクションの結果でエラーが発生すると、タスクコンフィグレーションで指定されたエラー応答がトリガされます。

バイト配列を、一般的なANY_REALデータタイプまたはこのデータタイプを含む構造に変換する場合、注意が必要です。

バイト配列のビット列は、ANY_REAL値としてのチェックが行われません。

バイト配列のビット列は、必ず、IEEEに準拠した正規化浮動小数点数のビットパターンと一致するようにしなければなりません。そのために、ファンクション[*finite*]

(「[*finite*]ファンクション」のセクションを参照)および[*isNaN*]

(「[*isNaN*]ファンクション」のセクションを参照)を使用できます。

IEEEビットパターンと一致しない場合は、(たとえば、プログラム内の、または、シンボルブラウザでの監視時の)算術演算でANY_REAL値が最初に使用されたときにすぐに、エラーがトリガされます。

表 7-24 マーシャリングファンクションの使用例

```
TYPE
  Struct_1 : STRUCT
    m_word      : WORD;
    m_byte      : BYTE;
  END_STRUCT;
  Struct_2 : STRUCT
    m_struct     : ARRAY [0..2] OF Struct_1;
    m_lreal      : LREAL;
  END_STRUCT;
END_TYPE
```

```

VAR
    gsbVar          : Struct_2;
    big_b_Array     : ARRAY [0..16] OF BYTE;
    lit_b_Array     : ARRAY [0..16] OF BYTE;
END_VAR

// 構造体への変数の割り当て
gsbVar.m_struct[0].m_word      := WORD#16#7FF1;
gsbVar.m_struct[0].m_byte     := BYTE#16#F9;
gsbVar.m_struct[1].m_word      := WORD#16#9FF7;
gsbVar.m_struct[1].m_byte     := BYTE#16#80;
gsbVar.m_struct[2].m_word      := WORD#16#A881;
gsbVar.m_struct[2].m_byte     := BYTE#16#BC;
gsbVar.m_lreal              := LREAL#-12345.6789e123;

// ビッグエンディアンへの変換
big_b_Array := AnyType_to_BigByteArray (
    anyData      := gsbVar,
    offset       := 0);

// big_b_array(ビッグエンディアン)の要素の内容:
// 下の表の2列目を参照

// リトルエンディアンへの変換
lit_b_Array := AnyType_to_LittleByteArray (
    anyData      := gsbVar,
    offset       := 0);

// lit_b_array(リトルエンディアン)の要素の内容:
// 下の表の3列目を参照

// ビッグエンディアンからの変換
gsbVar := BigByteArray_to_AnyType (
    bytearray := big_b_Array,
    offset := 0);

// リトルエンディアンからの変換
gsbVar := BigByteArray_to_AnyType (
    bytearray := lit_b_Array,
    offset := 0);
    
```

表 7-25 例のbig_b_arrayおよびlit_b_arrayの配列要素の内容

バイト配列big_b_arrayまたはlit_b_array			変数gsbVarのコンポーネント	
配列インデックス	big_b_array (ビッグエンディアン)	lit_b_array (リトルエンディアン)	名前	規格値
16	BYTE#16#07	BYTE#16#DA	m_lreal	LREAL#- 12345.6789e123
15	BYTE#16#F0	BYTE#16#52		
14	BYTE#16#43	BYTE#16#3C		
13	BYTE#16#68	BYTE#16#EC		

バイト配列big_b_arrayまたはlit_b_array			変数gsbVarのコンポーネント	
配列インデックス	big_b_array (ビッグエンディアン)	lit_b_array (リトルエンディアン)	名前	規格値
12	BYTE#16#EC	BYTE#16#68		
11	BYTE#16#3C	BYTE#16#43		
10	BYTE#16#52	BYTE#16#F0		
9	BYTE#16#DA	BYTE#16#07		
8	BYTE#16#BC	BYTE#16#BC	m_struct[2].m_byte	BYTE#16#BC
7	BYTE#16#81	BYTE#16#A8	m_struct[2].m_word	WORD#16#A88
6	BYTE#16#A8	BYTE#16#81		
5	BYTE#16#80	BYTE#16#80	m_struct[1].m_byte	BYTE#16#80
4	BYTE#16#F7	BYTE#16#9F	m_struct[1].m_word	WORD#16#9FF7
3	BYTE#16#9F	BYTE#16#F7		
2	BYTE#16#F9	BYTE#16#F9	m_struct[0].m_byte	BYTE#16#F9
1	BYTE#16#F1	BYTE#16#7F	m_struct[0].m_word	WORD#16#7FF1
0	BYTE#16#7F	BYTE#16#F1		

7.20.5 通信ファンクション

7.20.5.1 使用可能なファンクション

STでは次のような未設定リンク上の通信用のファンクションを提供します。

- _Xsend
- _GetStateOfXCommand
- _Xreceive

これらの通信ファンクションはデータの送受信に使用されます。

- SIMOTIONデバイス間
- SIMOTIONとSIMATIC S7のデバイス間
(S7-300、S7-400、M7-300、M7-400など)。

2つのファンクション[_Xsend]および[_Xreceive]を使用して、クライアントのユーザープログラムで順序立てて送信されたデータパケットの伝送を透過的にして、サーバーのユーザープログラムで順序立てて受信することができます。

追加情報

詳細は、『システム/コンフィグレーションマニュアル - 通信』を参照してください。

受信装置のプログラムは、データパケットに追加されたユーザー定義可能な整数に基づいてデータパケットを受信するかどうかを認識します。

データ交換は、ユーザープログラムがデータパケットを受信し、拒否しなかった場合にのみ成功します。

送信データはバイトシーケンスの配列の形式です。つまり、データには論理的構造がありません。

SIMOTIONデバイスは一度に最大200バイトまで送受信できます。実際のユーザーデータ長は通信ピアによって異なります。

XSendまたはXReceiveリクエストの状態は[`GetStateOfXCommand`]コマンドを使用してチェックできます。

注記

そのほかに次のような通信ファンクションがあります。

- [`udpSend`]
- [`udpReceive`]

これらのファンクションを使用して、Ethernetを介したUDPプロトコルによる通信ができます。(SIMOTIONデバイスのパラメータマニュアルを参照)。

7.20.5.2 [`Xsend`]のパラメータリスト

[`Xsend`]ファンクションでは、データパケットを透過的に通信ピアに送信します。パラメータの詳細な構文については、SIMOTIONデバイスのパラメータマニュアルを参照してください。

概要は次のとおりです。

- 通信モード(*communicationModel*パラメータ)により、データ伝送終了後に接続を保持するかどうかを選択できます。
- *address*パラメータで通信ピアの伝送先アドレスを指定します。パラメータはデータタイプ *StructXSendDestAddr* です。以下の表に、個々の要素の意味を示します。

表 7-26 伝送先アドレスの構造

パラメータ/データタイプ	意味/値	値
deviceId (USINT)	接続のポイント	C230-2、C240: X8は1 X9は2
		P350: X101は1 X102は2
		D4x5: X126は1 X136は2
remoteSubnetIdLength (USINT)	サブネットダイアログボックスの長さ	0 (MPI、PROFIBUS用)

パラメータ/データタイプ	意味/値	値
remoteStaddrLength (USINT)	伝送先システムのステーション アドレス(ステーション番号)の長 さ。	1 (MPI、PROFIBUS用)
nextStaddrLength (USINT)	ルータアドレスの長さ	0 (MPI、PROFIBUS用)
remoteSubnetId (ARRAY [0..5] OF USINT)	サブネットマスク	(MPI、PROFIBUSでは該当しません)
remoteStaddr (ARRAY [0..5] OF USINT)	ターゲットシステムのステーシ ョンアドレス(実際の伝送先アド レス)	MPI、PROFIBUSのステーション番号 : 例: remoteStaddr[0] = 25
nextStaddr (ARRAY [0..5] OF USINT)	ルータアドレス	(MPI、PROFIBUSでは該当しません)

[Xsend]ファンクションの追加パラメータは次のとおりです。

- 受信者は、データパケットに追加した**ジョブ識別子**(*messageId*パラメータ)からデータパケットを識別します。
- 2つの**データ伝送のモード**(*nextCommand*パラメータ)から選択できます:
同期または非同期
 - 同期的なデータ伝送時は、プログラムは受信側がデータパケットの確認応答を行うまで待機してから実行を再開します。
これはデータパケットの受信時に自動的に実行されます。
 - 非同期データ伝送中は、コマンドの発行後すぐにプログラムが再開されます。
コマンドのステータスは、[GetStateOfXCommand]を使用してチェックできます。
- 必須の**commandId**パラメータは、STの内部のコマンドの検出に使用されます。パラメータ値は[getCommandId]ファンクションによりローカル変数(*CommandIdType*)に保存する必要があります。この変数は、パラメータ値として使用することができます。
- **データパケット**(*data*パラメータ)には、それぞれが1バイトずつの200のエントリが入るリストが含まれます。
送信データの量が最大値より小さい場合は、リストがこの長さになる必要はありません。
- **データ長**(*dataLength*パラメータ)を使用して、伝送するデータパケットの実際の長さを指定します。
- **戻り値**に基づいて、コマンドの実行が成功したか(戻り値=0)を確認できます。
戻り値に0以外の値が返された場合、エラーが発生していることになります(SIMOTIONデバイスのパラメータマニュアルのコマンドの構文を参照)。

7.20.5.3 [Xreceive]のパラメータリスト

[Xreceive]ファンクションを使用して、通信ピアが[Xsend]により送信した伝送データを受信します。

次にファンクションのパラメータの概要を示します(SIMOTIONデバイスのパラメータマニュアルのシステムファンクションのセクションを参照)。

- 送信側で追加したジョブ識別子(*messageId*パラメータ)から想定されたデータパケットを識別します。
- 送信時と同様に、2つのデータ受信のモード(*nextCommand*パラメータ)から選択できます: 同期または非同期
 - 同期的なデータの受信時は、プログラムはデータパケットが到着し終わるまで待機してから実行を再開します。
データパッケージが受信されたことは、送信側に対して自動的に通知されます。
 - 非同期データ伝送中は、コマンドの発行後すぐにプログラムが再開されます。
コマンドのステータスは、`[_GetStateOfXCommand]`を使用してチェックできます。
- 必須の*commandId*パラメータは、STの内部のコマンドの検出に使用されます。パラメータ値は、`[_getCommandId]`ファンクションによりローカル変数(*CommandIdType*)に保存する必要があります(機能マニュアル『SIMOTION基本機能』のセクションを参照)。この変数は、パラメータ値として使用することができます。
- 戻り値は、構造体です。
 - *functionResult*要素に基づいて、コマンドの実行が成功したか(*functionResult*=0)を確認できます。
0以外の値が返された場合、エラーが発生していることになります(SIMOTIONデバイスのパラメータマニュアルのコマンドの構文を参照)。
 - *dataLength*要素は、受信したデータパッケージのデータ長を示します。
 - *data*要素は、受信したデータパッケージを示します(それぞれが1バイトずつの最大200エントリの配列)。

7.20.5.4 `[_GetStateOfXCommand]`のパラメータリスト

`[_Xsend]`または`[_Xreceive]`コマンドの状態は、`[_GetStateOfXCommand]`ファンクションによりチェックできます。

次にファンクションパラメータの概要を示します(テクノロジーファンクションの取扱説明書も参照)。

- 状態クエリーにより、それぞれの送受信ファンクションに一意に割り当てられた必須の*commandId*パラメータにどのコマンドが関連するかがわかります(上の`[_Xsend]`および`[_Xreceive]`のパラメータに関する情報を参照)。
- 戻り値は故障番号(コマンド実行が成功した場合は0、それ以外は0より大きな値)とチェックするコマンドの状態(SIMOTIONデバイスのシステムファンクションのコマンドの構文を参照)で構成されます。

7.20.5.5 SIMOTIONとSIMATIC S7デバイス間の通信

SIMOTIONとSIMATIC

S7デバイス間の通信において、次の事項を考慮する必要があります。

- 通信は、`[_Xsend]`および`[_Xreceive]`でのみ可能です。
- 1パケットで伝送可能なデータの最大量は76バイトに制限されています。
より大きなデータパケットを伝送すると、故障メッセージが出力されます。
- SIMOTIONインターフェースは、SIMATIC S7デバイスのMPIインターフェースに接続する必要があります。
SIMOTIONインターフェースの転送速度は、SIMATIC

S7デバイスの転送速度に合わせて設定する必要があります。たとえば、SIMATIC S7-300の転送速度は187.5 Kbits/sに設定する必要があります(関連するSIMATIC S7デバイスの取扱説明書を参照)。

通知

`[_Xsend]`および`[_Xreceive]`コマンドのパラメータは、SIMOTIONシステムでは別の名前を持ち、場合によっては、以前のSIMATIC S7システムで使用されていたものと別の意味を持つものもあります。次の表に、それぞれの比較を示します。
`[_GetStateOfXCommand]`コマンドは、SIMATIC S7システムには存在しないので、比較表にありません。

表 7-27 SIMATIC S7とSIMOTIONデバイスの`[_Xsend]`のパラメータの比較

SIMATIC S7デバイス (SFC 65 X_SENDのパラメータ)	SIMOTIONデバイス (<code>[_Xsend]</code> のパラメータ)
REQ (有効化のリクエスト、データタイプBOOL)	— (使用不可)
DEST_ID (通信ピアのMPIステーション番号)	アドレス (データタイプStructXSendDestAddrの構造体としての通信ピアの伝送先アドレス)
CONT (接続の保持、ブール値TRUEまたはFALSE)	communicationMode (接続の保持、列挙値M_TRUEまたはM_FALSE)
— (使用不可)	nextCommand (データ伝送モード、同期および非同期用の列挙型値)
REQ_ID (ジョブ識別子、データタイプDWORDの値)	messageld (ジョブ識別子、データタイプUDINTの値)
SD (送信データの変数アドレス、データタイプANYポインタの値、最大76バイト長)	データ (送信するデータパケット、データタイプUSINTの最大200個の値を持つ1次元配列)
— (使用不可)	dataLength (送信するデータパケットのデータ長、バイト単位、データタイプUDINTの値)
— (使用不可)	commandId (内部のコマンド識別子、上記のパラメータの説明を参照)
BUSY (有効化状態、データタイプBOOLの値)	— (使用不可)
<戻り値> (= 0、エラーなしの場合。<> 0、エラー発生した場合。SIMATIC S7取扱説明書を参照)	<戻り値、 (= 0、エラーなしの場合。<> 0、エラー発生した場合。SIMOTIONデバイスの 『パラメータマニュアル』を参照)

表 7-28 SIMATIC S7とSIMOTIONデバイスの[_Xreceive]のパラメータの比較

SIMATIC S7デバイス (SFC 66 X_RCVのパラメータ)	SIMOTIONデバイス ([_Xreceive]のパラメータ)
EN_DT (有効なデータ送受信、データタイプBOOL)	- (使用不可)
REQ_ID (ジョブ識別子、データタイプDWORDの値)	messageld (ジョブ識別子、データタイプUDINTの値)
- (使用不可)	nextCommand (データ伝送モード、同期および非同期用の列挙型値)
- (使用不可)	CommandId (内部のコマンド識別子、上記のパラメータの説明を参照)
RD (受信データの変数アドレス、データタイプANY ポインタの値、 最大長76バイト)	<戻り値、データ構造体要素> (受信データパケット、それぞれが1バイトずつの 最大200エントリの配列)
- (使用不可)	<戻り値、 dataLength構造体要素> (受信データパケットのデータ長)
<戻り値> (= 0、エラーなしの場合。<> 0、エラー発生した場合。SIMATIC S7取扱説明書を参照)	<戻り値、 functionResult構造体要素> (= 0、エラーなしの場合。<> 0、エラー発生した場合。SIMOTIONデバイスの 『パラメータマニュアル』を参照)

7.20.5.6 送受信プログラムの例

次に送受信プログラムのソーステキストを示します。

表 7-29 送信プログラムの例

```

INTERFACE
    PROGRAM xsend_control;
END_INTERFACE

IMPLEMENTATION
// 以下のプログラムは、MotionTaskに割り当てる
// 必要がある。
// タスクの設定で、[Activation
// after Startup Task]オプションを選択すること。
PROGRAM xsend_control
VAR
    retVal          : DINT;
    myAddress       : StructXSendDestAddr;
    myStaddr       : ARRAY[0..4] OF BYTE;
    myData         : ARRAY[0..199] OF BYTE;

```



```
END_VAR
// 伝送先アドレスおよびPROFIBUSインターフェースの指定 //////////
myAddress.deviceID      := 1;
                        // PROFIBUSインターフェースX8
myAddress.remoteStaddrLength := 1;
                        // 常に値1を割り当てる必要があります
myAddress.remoteStaddr[0] := 4;
                        // 受信ステーションのPROFIBUSアドレス

// 送信データ
myData[0] := 170;

// 送信ファンクションの呼び出し
retVal := _Xsend( communicationMode := ABORT_CONNECTION
, address      := myAddress
, messageId    := 1
, nextCommand  := WHEN_COMMAND_DONE
, commandId    := _getCommandId()
, data         := myData
, dataLength   := 1
);
END_PROGRAM
END_IMPLEMENTATION
```

表 7-30 受信プログラムの例

```
INTERFACE
    PROGRAM xreceive_control;
END_INTERFACE

IMPLEMENTATION
VAR_GLOBAL
    retVal : StructRetXReceive;
END_VAR
// 以下のプログラムは、MotionTaskに割り当てる
// 必要がある。
// タスクの設定で、[Activation
// after Startup Task] オプションを選択すること。
PROGRAM xreceive_control

// 受信ファンクションの呼び出し
retVal := _Xreceive( messageId      := 1
, nextCommand    := WHEN_COMMAND_DONE
, commandId      := _getCommandId()
);
END_PROGRAM
END_IMPLEMENTATION
```

7.20.5.7 Ethernetを介したTCP/IPプロトコルによる通信

Ethernetインターフェースを備えたSIMOTIONデバイスでは、TCP/IPプロトコルによる通信も可能です。

次の表に、送信者(クライアント)と受信者(サーバ)との間の通信を確立する個々の手順を示します。

表 7-31 TCP/IP接続および対応するシステムファンクションの通信手順

通信手順		システムファンクション
接続の確立		
1	受信者(サーバ)は、通信要求を待機します。	_tcpOpenServer
2	送信者(クライアント)は、受信者に接続の確立を要求します。	_tcpOpenClient
3	受信者(サーバ)は、通信要求が確立されます。	
通信		
4	送信者は受信者にデータを送信します。	_tcpSend
5	受信者は送信者からデータを受信します。	_tcpReceive
接続の終了		
6	送信者はそれ以上データを送信せず、接続を閉じます。	_tcpCloseConnection
7	必要な接続はそれ以上ありません。	_tcpCloseServer

システムファンクションについては、SIMOTIONデバイスの『パラメータマニュアル』のシステムファンクションに関する章で詳細に説明しています。

注記

クライアントおよびサーバのいずれも、接続を確立するときには送信者にも受信者にもなりません。

TCP/IP接続の確立には、少なくとも1つのクライアントと1つのサーバが必要です。

クライアント-サーバの関係は、接続が確立されるまでは有効です。

接続が確立されると、両方の通信ピアは同等になり、両者のそれぞれが送信も受信もできて、任意の時点で通信を閉じることができます。

詳細な情報については、「*SIMOTION Utilities & Applications*」CD-ROMの「よくある質問」(FAQ)を参照してください。

7.20.5.8 Ethernetを介したUDPプロトコルによる通信

Ethernetインターフェースを備えたSIMOTIONデバイスでは、UDPプロトコルによる通信も可能です。

- `_udpSend` ファンクションを使用して、IPアドレスとポート番号で指定された通信ピアにデータパケットを送信します。

送信するデータはARRAY [0..1399] OF BYTEとしてファンクションに伝送されます。

- `[_udpReceive]` ファンクションを使用して、通信ピアが `[_udpSend]` により送信したデータパケットを受信します。

受信したデータはデータタイプ `ARRAY [0..1399] OF BYTE` の変数に格納されます。この識別子はパラメータとしてファンクションに伝送されます。

システムファンクションについては、SIMOTIONデバイスの『パラメータマニュアル』のシステムファンクションに関する章で詳細に説明しています。

詳細な情報については、「*SIMOTION Utilities & Applications*」CD-ROMの「よくある質問」(FAQ)を参照してください。

7.20.6 同期起動

同期起動では、複数のコマンドがIPOサイクルクロックまたはIPO_2サイクルクロック内で起動されます。

次の手順に従います。

1. まず、システムから一意のSyncCommandIdを取得します。
このSyncCommandIdは、同期するコマンドの一意の名称として必要になります。
そのために、`_getSyncCommandId()` システムファンクションを使用します。
2. 同期的に実行するコマンドを `BEGIN_SYNC(SyncCommandId)` と `END_SYNC()` ファンクションの間に置きます。これにより、そのコマンドが同期起動に定義されます。
`BEGIN_SYNC/END_SYNC` 構造の中ですべてのモーションコマンドが使用可能です。
`nextCommand` パラメータとして、値 `IMMEDIATELY` を伝送する必要があります。
3. 同期起動自体は `_startSyncCommand(SyncCommandId)` ファンクションで実行します。
同期起動に定義されたコマンドは、並列に処理されます。

サンプルプログラムを参照してください。

通知

モーションコマンドの同期起動は、次の条件が満たされた場合のみ保証されます。

1. 同期起動に含まれるコマンドは、さまざまなテクノロジーオブジェクトに作用する必要があります。
2. 関連するテクノロジーオブジェクトは、同じ実行レベルである必要があります(IPOまたはIPO_2)。

`[BEGIN_SYNC]`、`[END_SYNC]`、および `[_startSyncCommand]` ファンクションは、SIMOTIONデバイスのシステムファンクションです。詳細は、関連するSIMOTIONデバイスのパラメータマニュアルを参照してください。

注記

同期起動は、しばしば `WAITFORCONDITION` 構造と組み合わせて使用されます。
この場合、システムは条件が満たされるまで待機してから同期起動が行われます。
`[_startSyncCommand]` ファンクションは、`WAITFORCONDITION` 構造の中で実行する必要があります。

表 7-32 WAITFORCONDITIONを使用した同期起動のサンプルプログラム

```
INTERFACE
  USEPACKAGE cam;
  PROGRAM sync_motion;
END_INTERFACE

IMPLEMENTATION

  EXPRESSION wait_sync_expression
    wait_sync_expression := TRUE;
  END_EXPRESSION

  PROGRAM sync_motion
    VAR
      ret_val : DINT;
      sync_id : CommandIdType;
    END_VAR

    sync_id := _getSyncCommandId();
    BEGIN_SYNC(sync_id);
    (* Position axis ('Pos') *)
    ret_val := _pos (axis := Axis_1,
      positioningMode := ABSOLUTE,
      position := 100,
      mergeMode := IMMEDIATELY,
      nextCommand := IMMEDIATELY,
      commandId := _getCommandId() );
    (* Position axis ('Pos') *)
    ret_val := _pos (axis := Axis_2,
      positioningMode := ABSOLUTE,
      position := 50,
      mergeMode := IMMEDIATELY,
      nextCommand := IMMEDIATELY,
      commandId := _getCommandId() );
  END_SYNC();

  WAITFORCONDITION wait_sync_expression DO
    ;
  END_WAITFORCONDITION;

  ret_val := _startSyncCommands(sync_id);

END_PROGRAM
END_IMPLEMENTATION
```

7.21 HMI (マンマシンインターフェース)接続

7.21.1 HMIとSCOUTまたはSIMOTIONとの間のインターフェース

SIMOTIONでは、操作パネルなどのHMIデバイス(エンドユーザー用操作デバイス)との通信ができます。

1つ以上のSIMOTIONデバイスがPROFIBUSまたはPROFINET上に接続されている場合、HMIデバイスで、SIMOTIONデバイスの変数、メッセージ、およびアラームを表示できます。各SIMOTIONデバイスのソースを作成できるので、ソースの変数をHMIデバイスに表示できます。

同様に、HMIデバイスを使用してプログラムしたファンクションやSIMOTIONデバイスのプログラムを初期化できます。

そのようなHMIデバイスのタスクの実装に、WinCC flexibleおよびProToolソフトウェアパッケージを使用できます。WinCC flexibleまたはProToolを使用して、インターフェースセクションで宣言されたシステム変数およびユニット変数を読み取りおよび書き込みできます。1つのユニットでのHMI変数(ページ 444)を参照してください。

SIMOTIONにもOPCサーバが備わります。

このサーバを使用して、独自仕様でないHMIソフトウェアによりプロセスデータにアクセスできます。

注記

HMIで使用するための変数は、必ずソースファイルのインターフェースセクションでユニット変数として作成する必要があります。

7.21.2 ProToolアプリケーションの例

ProToolアプリケーションの詳細なサンプルは、『*ProTool/Pro - SIMOTION HMIシステム起動用オペレータインターフェースのサンプル*』オペレータズガイドを参照してください。

7.21.3 HMIデバイスへの一貫性のあるデータアクセス(例)

HMIデバイスは、SIMOTIONデバイスのユーザーデータへの一貫性のあるアクセスを行うこともできます。

次の例には、SIMOTIONデバイス上のSTプログラム(例を参照)、およびHMIデバイス上のアプリケーション(Visual Basicプログラム)(例を参照)が含まれます。

ユーザーはHMIアプリケーション内からユーザー変数(配列を含む)への一貫性のある読み取りアクセスを要求します。ここでは、HMIアプリケーションはVisual Basicですが、Visual C++なども可能です。奇数値がconsistencyFlag変数に書き込まれます。

SIMOTIONユーザープログラムはデータをコピーします。HMIアプリケーションは、SIMOTIONユーザープログラムが、偶数値をconsistencyFlag変数に書き込むことによって

データのコピーが完了したことを確認するまで待機します。
その期間に変更がなかったために、HMIはこの変数を矛盾なく読み取ります。

STプログラムを周期的タスクに割り当てることができます。たとえば、軸位置と速度を同時に(スナップショットとして)読み取る場合など、IPOSynchronousTaskを使用する必要がある場合があります。

例:

表 7-33 HMIデバイスからの一貫性のあるデータアクセスを行うSTプログラム

```
INTERFACE
  VAR_GLOBAL
    consistencyFlag : DINT;
    myDint          : DINT;
    myArray : ARRAY[0..10] OF LREAL;
  END_VAR
  PROGRAM OPC_Prog;
END_INTERFACE

IMPLEMENTATION
  PROGRAM OPC_Prog
    IF (consistencyFlag MOD 2) = 1 THEN
      myDint := 99;
      myArray[0] := 0.0;
      myArray[1] := 1.0;
      myArray[10] := 10.0;
      consistencyFlag := consistencyFlag + 1;
    END_IF;
  END_PROGRAM
END_IMPLEMENTATION
```

表 7-34 一貫性のあるデータアクセスを行うHMIアプリケーション(Visual Basic)

```
Option Explicit
Option Base 1

Dim g_Server As OPCServer
Dim g_GroupObj As OPCGroup

Dim g_myItem1 As OPCItem
Dim g_myItem2 As OPCItem

Dim g_myItem3 As OPCItem
Const OPC_DS_DEVICE = 2

Dim consistencyFlag As Long

Private Sub Form_Load()
  Set g_Server = New OPCServer
  g_Server.Connect ("OPC.SimaticNet")
  Set g_GroupObj = g_Server.OPCGroups.Add("Test1")
  g_GroupObj.IsActive = False
  Set g_myItem1 = g_GroupObj.OPCItems.AddItem("C240.consistencyFlag", 2)
  Set g_myItem2 = g_GroupObj.OPCItems.AddItem("C240.myDint", 2)
  Set g_myItem3 = g_GroupObj.OPCItems.AddItem("C250.myArray", 3)
```

```
        consistencyFlag = 1
    End Sub
    Private Sub Form_Unload(Cancel As Integer)
        Set g_myItem1 = Nothing
        Set g_myItem2 = Nothing
        Set g_myItem3 = Nothing
        Set g_GroupObj = Nothing
        Set g_Server = Nothing
        終了
    End Sub

    Private Sub cmdRead_Click()
        Static reentrancyFlag As Boolean
        Dim var1 As Variant
        Dim var2 As Variant
        Dim var3 As Variant

        If reentrancyFlag = False Then
            reentrancyFlag = True
            consistencyFlag = consistencyFlag + 2
            g_myItem1.Write consistencyFlag
            DO
                g_myItem1.Read OPC_DS_DEVICE, var1
            Loop Until var1 = consistencyFlag + 1
            g_myItem2.Read OPC_DS_DEVICE, var2
            g_myItem3.Read OPC_DS_DEVICE, var3
            reentrancyFlag = False
        End if
    End Sub
```


一般的なシステムファンクションブロックのプログラミング

8

8.1 ファンクションブロックの概要

STには、あらかじめ宣言することなくSTソースファイルで利用できる、一連のシステムファンクションブロックが含まれます。
これは、インスタンスの作成と必要なパラメータの割り当てのみが必要です。

システムファンクションブロックの概要

次に、システムに実装されるすべてのシステムファンクションブロックの概要を示します。
セクション7.1以降で、定義と詳細な使用について説明します。

表 8-1 システムファンクションブロック

総称	名前	入力パラメータ		出力パラメータ	
		名前	: タイプ	名前	: タイプ
双安定ファンクションブロック	SR¹ セット優先	S1 R	: BOOL; : BOOL;	Q1	: BOOL;
	RS¹ リセット優先	SR1	: BOOL; : BOOL;	Q1	: BOOL;
エッジ検出	R_TRIG¹ 信号立ち上がり	CLK	: BOOL;	Q	: BOOL;
	F_TRIG¹ 立下りエッジ	CLK	: BOOL;	Q	: BOOL;
COUNTER	CTU¹ アップカウンタ データタイプ:INT	CU	: BOOL;	Q	: BOOL;
		R	: BOOL;	CV	: INT;
		PV	: INT;		
	CTU_DINT¹ アップカウンタ データタイプ:DINT	CU	: BOOL;	Q	: BOOL;
		R	: BOOL;	CV	: DINT;
		PV	: DINT;		
	CTU_UDINT¹ アップカウンタ データタイプ: UDINT	CU	: BOOL;	Q	: BOOL;
		R	: BOOL;	CV	: UDINT;
		PV	: UDINT;		
	CTD¹ ダウンカウンタ データタイプ:INT	CD	: BOOL;	Q	: BOOL;
		LD	: BOOL;	CV	: INT;
		PV	: INT;		

8.1 ファンクションブロックの概要

総称	名前	入力パラメータ		出力パラメータ	
	CTD_DINT¹ ダウンカウンタ データタイプ:DINT	CD	: BOOL;	Q	: BOOL;
		LD	: BOOL;	CV	: DINT;
		PV	: DINT;		
	CTD_UDINT¹ ダウンカウンタ データタイプ: UDINT	CD	: BOOL;	Q	: BOOL;
		LD	: BOOL;	CV	: UDINT;
		PV	: UDINT;		
	CTUD¹ アップ/ダウンカウンタ データタイプ:INT	CU	: BOOL;	QU	: BOOL;
		CD	: BOOL;	QD	: BOOL;
		R	: BOOL;	CV	: INT;
		LD	: BOOL;		
		PV	: INT;		
	CTUD_DINT¹ アップ/ダウンカウンタ データタイプ:DINT	CU	: BOOL;	QU	: BOOL;
		CD	: BOOL;	QD	: BOOL;
		R	: BOOL;	CV	: UINT;
		LD	: BOOL;		
		PV	: DINT;		
	CTUD_UDINT¹ アップ/ダウンカウンタ データタイプ: UDINT	CU	: BOOL;	QU	: BOOL;
		CD	: BOOL;	QD	: BOOL;
		R	: BOOL;	CV	: UDINT;
		LD	: BOOL;		
		PV	: UDINT;		
タイマー	TP¹ Puls	IN	: BOOL;	Q	: BOOL;
		PT	: TIME;	ET	: TIME;
	TON¹ ON遅延	IN	: BOOL;	Q	: BOOL;
		PT	: TIME;	ET	: TIME;
	TOF¹ ON遅延	IN	: BOOL;	Q	: BOOL;
		PT	: TIME;	ET	: TIME;
	RTC¹ リアルタイムクロック	SET	: BOOL;	CDT	: DT;
		READ	: BOOL;		
ビット列データタイプの 分割	_BYTE_TO_8BOOL	n	: BYTE	bit0、bit1、 bit2、bit3、 bit4、bit5、 bit6、bit7	: BOOL;
	_WORD_TO_2BYTE	in	: WORD	byte0、byte 1	: BYTE;
	_DWORD_TO_2WORD	in	: DWORD	word0、wor d1	: WORD;
	_DWORD_TO_4BYTE	in	: DWORD	byte0、byte 1、byte2、 byte3	: DWORD;

総称	名前	入力パラメータ		出力パラメータ	
SIMATIC S7コマンドのエミュレーション	_S7_COUNTER ¹	CU	: BOOL;	Q	: BOOL;
		CD	: BOOL;	CV	: INT;
		PV	: INT;		
		S	: BOOL;		
		R	: BOOL;		
	_S7_TIMER ¹	T_Type	: WORD;	Q	: BOOL;
		PR	: BOOL;	BI	: TIME;
		S	: BOOL;		
		R	: BOOL;		
		TV	: TIME;		
		TV_BCD	: WORD;		
¹ これらのファンクションブロックは、(周期的タスクなどで)呼び出しの繰り返しが必要です。					

SIMOTIONとSIMATICのシステムファンクションブロックの比較

Utilities & Applications CDの2_FAQディレクトリに、SIMATIC S7とSIMOTIONのシステムファンクションブロックの比較があります。

8.2 双安定要素(フリップフロップのセット)

STでは、システムファンクションブロックSRを使用してフリップフロップをセット/リセットし(セット優先)、システムファンクションブロックRSを使用してフリップフロップをリセット/セットします(リセット優先)。

双安定ファンクションブロックSR (セット優先)

格納されている値は、出力Q1で取得することができます。
S1が1のとき、出力Q1は1です。S1とRが0のとき、出力Q1は変化しません。

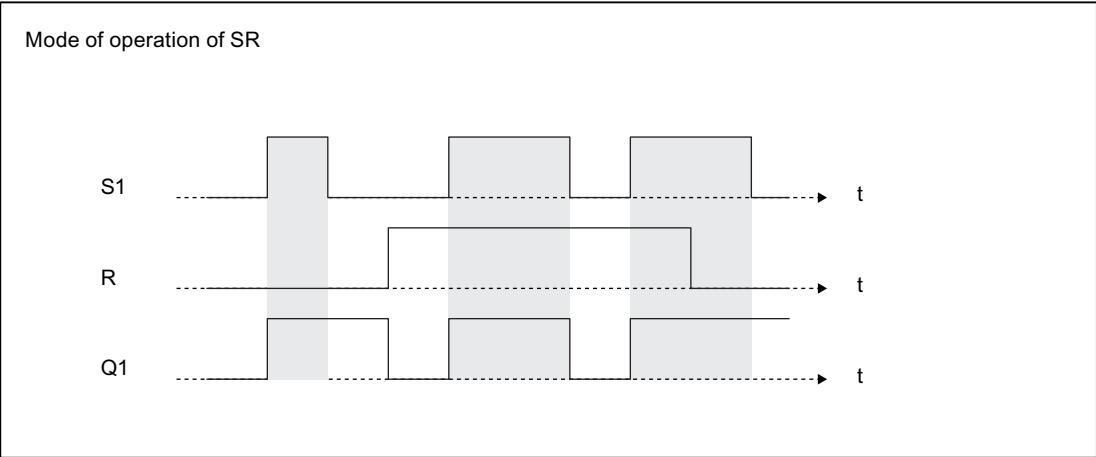


図 8-1 SR双安定ファンクションブロックの動作モード

表 8-2 SR双安定ファンクションブロックのプログラムコード

```
FUNCTION_BLOCK SR

VAR_INPUT
    S1,R          : BOOL;
END_VAR

VAR_OUTPUT
    Q1: BOOL;
END_VAR

Q1 := S1 OR (NOT R AND Q1);

END_FUNCTION_BLOCK

END_IMPLEMENTATION
```

表 8-3 SRの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
S1	Input	BOOL	設定
R	Input	BOOL	リセット
Q1	出力	BOOL	信号ステータス

双安定ファンクションブロックRS (リセット優先)

格納されている値は、出力Q1で取得することができます。
R1が1のとき、出力Q1は0です。R1とSが0のとき、出力Q1は変化しません。

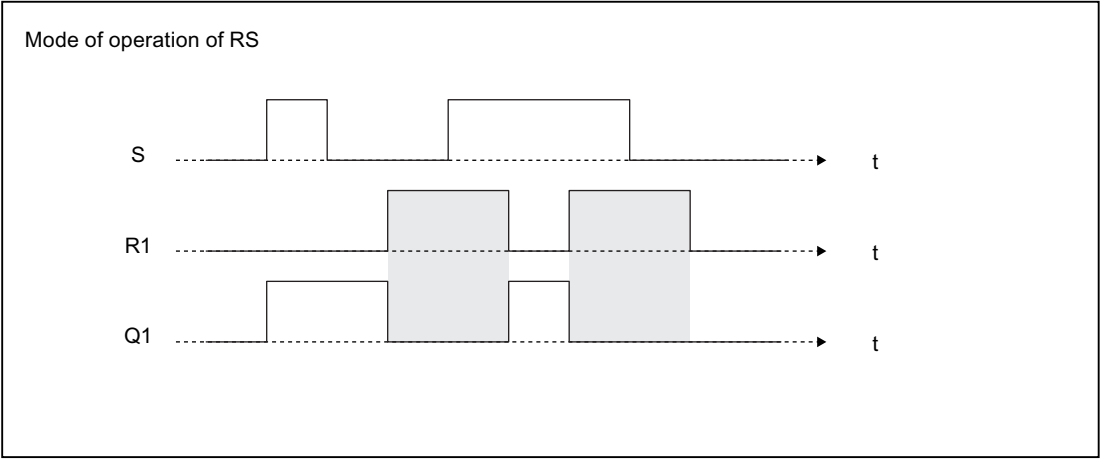


図 8-2 RS双安定ファンクションブロックの動作モード

表 8-4 RS双安定ファンクションブロックのプログラムコード

```
FUNCTION_BLOCK RS

VAR_INPUT
    R1,S          :    BOOL;
END_VAR

VAR_OUTPUT
    Q1:    BOOL;
END_VAR

    Q1 := NOT R1 AND (S OR Q1);

END_FUNCTION_BLOCK
```

表 8-5 RSの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
S	Input	BOOL	設定
R1	Input	BOOL	リセット
Q1	出力	BOOL	信号ステータス

8.3 エッジ検出

システムファンクションブロックR_TRIGを使用して、立ち上がりエッジを検出できます。
F_TRIGは立ち下がりエッジを検出できます。
たとえば、このファンクションを使用して、独自のファンクションブロックのシーケンスを設定できます。

立ち上がりエッジR_TRIGの検出

入力で立ち上がりエッジ(R_TRIG、立ち上がりトリガ)、つまり0から1への状態変化があった場合、1サイクルタイムの期間の出力に1が適用されます。

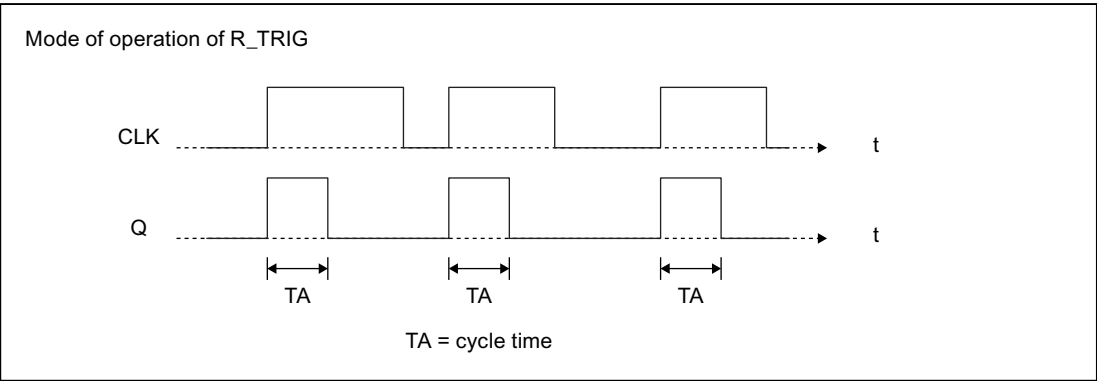


図 8-3 R_TRIG (立ち上がりエッジ)ファンクションブロックの動作モード

表 8-6 ファンクションブロックR_TRIG (立ち上がりエッジ)のプログラムコード

```
FUNCTION_BLOCK R_TRIG

  VAR_INPUT
    CLK      : BOOL;
  END_VAR

  VAR_OUTPUT
    Q: BOOL;
  END_VAR

  VAR
    M: BOOL := 0;
  END_VAR

  Q := CLK AND NOT M;
  M := CLK;

END_FUNCTION_BLOCK
```

表 8-7 R_TRIGの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
CLK	Input	BOOL	エッジ検出の入力
Q	出力	BOOL	エッジのステータス

立ち下がりエッジF_TRIGの検出

入力で立ち下がりエッジ(F_TRIG、立ち下がりトリガ)、つまり1から0への状態変化が発生した場合、1サイクルタイムの期間の出力に1が適用されます。

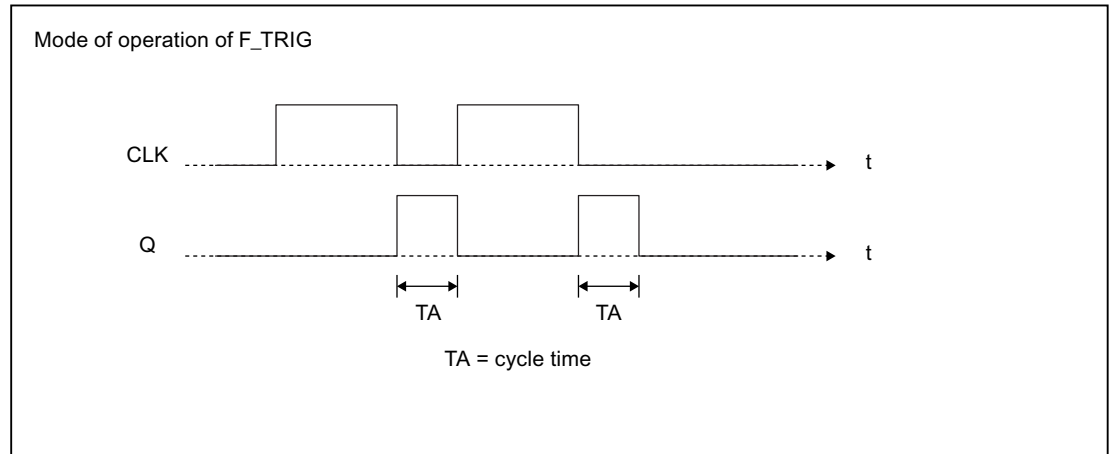


図 8-4 F_TRIG(立ち下がりエッジ)ファンクションブロックの動作モード

表 8-8 ファンクションブロックF_TRIG (立ち下がりエッジ)のプログラムコード

```
FUNCTION_BLOCK F_TRIG

    VAR_INPUT
        CLK      : BOOL;
    END_VAR

    VAR_OUTPUT
        Q: BOOL;
    END_VAR

    VAR
        M: BOOL      := 1;
    END_VAR

    Q := NOT CLK AND NOT M;
    M := NOT CLK;

END_FUNCTION_BLOCK
```

表 8-9 F_TRIGの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
CLK	Input	BOOL	エッジ検出の入力
Q	出力	BOOL	エッジのステータス

8.4 COUNTER

8.4.1 カウンタの概要

STには、STプログラムで使用できる一連のカウンタ用のシステムファンクションブロックが備わります。

8.4.2 CTUアップカウンタ

CTUカウンタにより、カウントアップ操作を実行できます。

- 入力R = TRUEの場合、FBが呼び出され、次にCV出力が0にリセットされます。
- FBが呼び出されているときにCU入力がFALSEからTRUE (0から1)に変化する場合(立ち上がりエッジ)、CV出力は1つ増加します。
- 出力Qは、CVが比較値PV以上であるかどうかを示します。

CVとPVパラメータは両方ともINTデータタイプです。これは最大カウンタ値が32_767 (= 16#7FFF)であることを意味します。

ユーザーインターフェースのプロトタイプ

サンプルコード

```
FUNCTION_BLOCK CTU
VAR_INPUT
    CU: BOOL; // カウント
    R : BOOL; // リセット
    PV : INT; // 比較値
END_VAR
VAR_OUTPUT
    Q: BOOL; // 状態
    CV : INT; // カウンタ読み込み
END_VAR
// ... (コード)
END_FUNCTION_BLOCK
```


入カパラメータ

CU

データタイプ: BOOL

FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントアップ

R

データタイプ: BOOL

TRUE: カウンタを0にリセット

PV

データタイプ: INT

比較値

出カパラメータ

Q

データタイプ: BOOL

カウンタのステータス(CV >= PV)

CV

データタイプ: INT

カウンタの値

8.4.3 CTU_DINTアップカウンタ

操作方法はCTUアップカウンタと同じですが(サブセクション7.3.1を参照)、次の点が異なります。

CVとPVパラメータは両方ともDINTデータタイプです。これは最大カウンタ値が2_147_483_647(= 16#7FFF_FFFF)であることを意味します。

表 8-10 CTU_DINTのパラメータ

識別子	パラメータ	データタイプ	説明
CU	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントアップ
R	Input	BOOL	カウンタを0にリセット
PV	Input	DINT	比較値
Q	出力	BOOL	カウンタのステータス(CV >= PV)
CV	出力	DINT	カウンタの値

8.4.4 CTU_UDINTアップカウンタ

操作方法はCTUアップカウンタと同じですが、次の点が異なります。

CVとPVパラメータは両方ともUDINTデータタイプです。これは最大カウンタ値が4_294_967_295 (= 16#FFFF_FFFF)であることを意味します。

表 8-11 CTU_UDINTのパラメータ

識別子	パラメータ	データタイプ	説明
CU	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントアップ
R	Input	BOOL	カウンタを0にリセット
PV	Input	UDINT	比較値
Q	出力	BOOL	カウンタのステータス(CV >= PV)
CV	出力	UDINT	カウンタの値

下記も参照

CTUアップカウンタ (ページ 416)

8.4.5 CTDダウンカウンタ

CTDカウンタにより、カウントダウン操作を実行できます。

- LD入力= TRUEの場合、FBが呼び出され、次にCV出力が開始値PVにリセットされます。
- FBが呼び出されているときにCD入力がFALSEからTRUE (0から1)に変化する場合(立ち上がりエッジ)、CV出力は1つ減少します。
- 出力QはCVが0以上であるかどうかを示します。

CVとPVパラメータは両方ともINTデータタイプです。これは最小カウンタ値が-32_768 (= 16#8000)であることを意味します。

表 8-12 CTDの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
CD	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントダウン
LD	Input	BOOL	カウンタを開始値にリセット
PV	Input	INT	カウンタの開始値
Q	出力	BOOL	カウンタのステータス(CV <= 0)
CV	出力	INT	カウンタの値

8.4.6 CTD_DINTダウンカウンタ

操作方法是CTDアップカウンタと同じですが、次の点が異なります。

CVとPVパラメータは両方ともDINTデータタイプです。これは最小カウンタ値が-2_147_483_648 (= 16#8000_0000)であることを意味します。

表 8-13 CTD_DINTの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
CD	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントダウン
LD	Input	BOOL	カウンタを開始値にリセット
PV	Input	DINT	カウンタの開始値
Q	出力	BOOL	カウンタのステータス(CV ≤ 0)
CV	出力	DINT	カウンタの値

8.4.7 CTD_UDINTダウンカウンタ

操作方法是CTDアップカウンタと同じですが、次の点が異なります。

CVとPVパラメータは両方ともUDINTデータタイプです。これは最小カウンタ値が0であることを意味します。

表 8-14 CTD_UDINTの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
CD	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントダウン
LD	Input	BOOL	カウンタを開始値にリセット
PV	Input	UDINT	カウンタの開始値
Q	出力	BOOL	カウンタのステータス(CV = 0)
CV	出力	UDINT	カウンタの値

8.4.8 CTUDアップ/ダウンカウンタ

CTUDカウンタにより、カウントアップおよびカウントダウン操作の両方を実行できます。

- CVカウンタ変数のリセット
 - － 入力R = TRUEの場合、FBが呼び出され、次にCV出力が0にリセットされます。
 - － LD入力= TRUEの場合、FBが呼び出され、次にCV出力が開始値PVにリセットされます。

- カウント
 - FBが呼び出されているときにCU入力がFALSEからTRUE (0から1)に変化する場合(立ち上がりエッジ)、CV出力は1つ増加します。
 - FBが呼び出されているときにCD入力がFALSEからTRUE (0から1)に変化する場合(立ち上がりエッジ)、CV出力は1つ減少します。
- カウンタ状態QUまたはQD
 - 出力Qは、CVが比較値PV以上であるかどうかを示します。
 - 出力QDはCVが0以下であるかどうかを示します。

表 8-15 CTUDのパラメータ

識別子	パラメータ	データタイプ	説明
CU	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントアップ
CD	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントダウン
R	Input	BOOL	カウンタを0にリセット(アップカウンタ)
LD	Input	BOOL	カウンタをPV開始値にリセット(ダウンカウンタ)
PV	Input	INT	比較値(アップカウンタ用) 開始値(ダウンカウンタ用)
QU	出力	BOOL	アップカウンタとしてのステータス($CV \geq PV$)
QD	出力	BOOL	ダウンカウンタとしてのステータス($CV \leq 0$)
CV	出力	INT	カウンタの値

8.4.9 CTUD_DINTアップ/ダウンカウンタ

操作方法はCTUDアップカウンタと同じですが、次の点が異なります。

CVおよびPVパラメータは、両方ともDINTデータタイプです。

表 8-16 CTU_DINTのパラメータ

識別子	パラメータ	データタイプ	説明
CU	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントアップ
CD	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントダウン
R	Input	BOOL	カウンタを0にリセット(ダウンカウンタ)
LD	Input	BOOL	カウンタをPV開始値にリセット(ダウンカウンタ)
PV	Input	DINT	比較値(アップカウンタ用) 開始値(ダウンカウンタ用)
QU	出力	BOOL	アップカウンタとしてのステータス($CV \geq PV$)
QD	出力	BOOL	ダウンカウンタとしてのステータス($CV \leq 0$)

識別子	パラメータ	データタイプ	説明
CV	出力	DINT	カウンタの値

8.4.10 CTUD_UDINTアップ/ダウンカウンタ

操作方法是CTUDアップカウンタと同じですが、次の点が異なります。

CVおよびPVパラメータは、両方ともUDINTデータタイプです。

表 8-17 CTU_UDINTのパラメータ

識別子	パラメータ	データタイプ	説明
CU	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントアップ
CD	Input	BOOL	FALSEからTRUEに値が変化した場合(立ち上がりエッジ)カウントダウン
R	Input	BOOL	カウンタを0にリセット(アップカウンタ)
LD	Input	BOOL	カウンタをPV開始値にリセット(ダウンカウンタ)
PV	Input	UDINT	比較値(アップカウンタ用) 開始値(ダウンカウンタ用)
QU	出力	BOOL	アップカウンタとしてのステータス(CV >= PV)
QD	出力	BOOL	ダウンカウンタ時の状態(CV = 0)
CV	出力	UDINT	カウンタの値

8.5 タイマー

タイマーは時間駆動プロセスの実行および監視に使用するプログラムの要素です。STには、STにアクセス可能な一連のシステムファンクションブロックが備わります。プログラムでタイマーを使用して、次のような処理を実行できます。

- 待機時間の設定
- タイマー値の有効化
- パルスの生成
- 時間の測定

TPパルス

IN入力で信号ステータスが0から1に変化すると、時間ETが開始されます。出力Qは、経過時間ETがプログラム時間値PTと等しくなるまで1のままです。ETが動作している間は、IN入力は影響がありません。

次の図は、TPパルスタイマーの動作モードを示します。

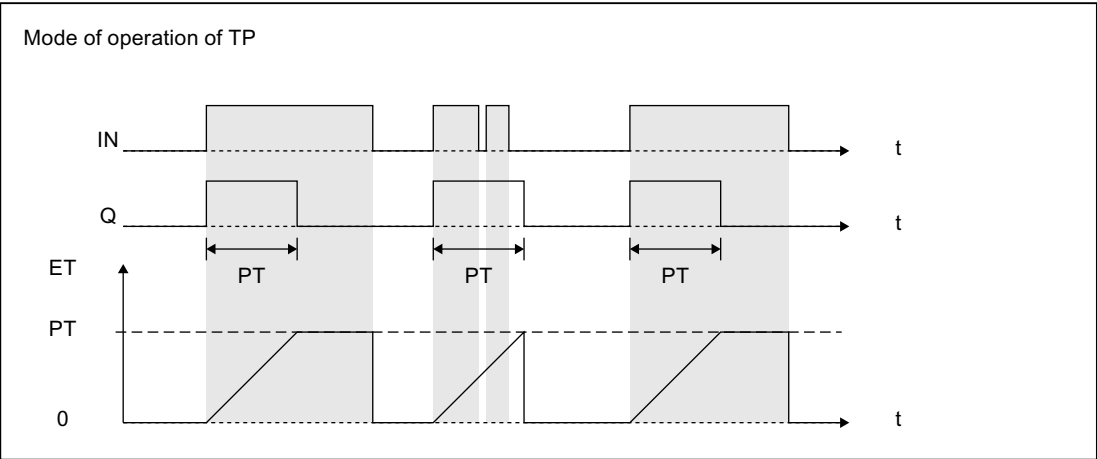


図 8-5 TPパルスの動作モード

以下の表に呼び出しパラメータを示します。

表 8-18 TPの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
IN	Input	BOOL	開始入力
PT	Input	TIME	パルスの時間
Q	出力	BOOL	時間のステータス
ET	出力	TIME	経過時間

TON ON遅延

IN入力で信号ステータスが0から1に変化すると、時間ETが開始されます。時間ET = PTが経過して、入力信号INが値1のままの場合にのみ、出力信号Qは0から1に変化します。つまり、出力Qは遅延付きでスイッチオンになります。入力信号の時間がプログラム時間PTより短い場合、出力には現れません。

次の図は、TON ON遅延タイマーの動作モードを示します。

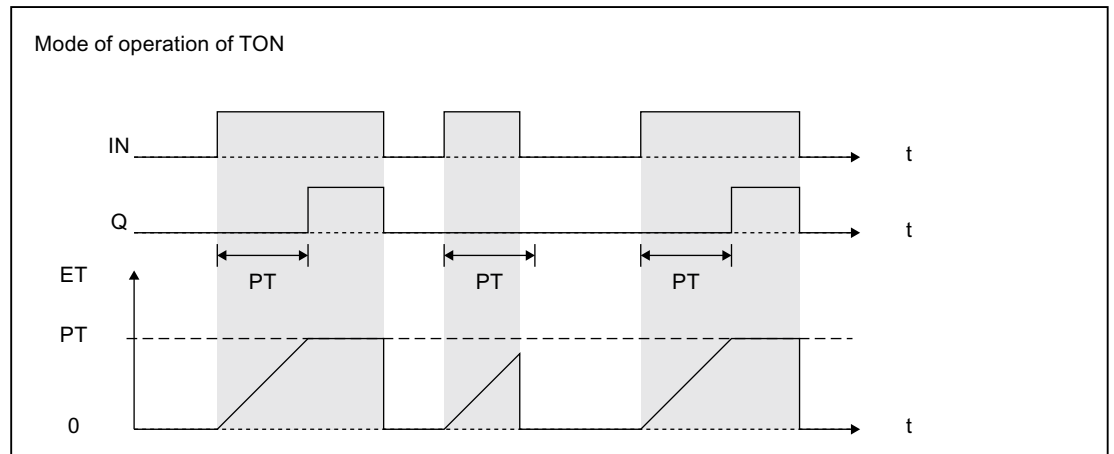


図 8-6 TON ON遅延の動作モード

以下の表に呼び出しパラメータを示します。

表 8-19 TONの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
IN	Input	BOOL	開始入力
PT	Input	TIME	入力INでの立ち上がりエッジが遅延される時間
Q	出力	BOOL	時間のステータス
ET	出力	TIME	経過時間

```

VAR_TEMP
  Mytimeout : TON;
End_VAR
Mytimeout(pt := T#2s, IN : TRUE);
IF (mytimeout.q) THEN
  //タイムアウト
ENDIF

```

TOF OFF遅延

開始入力INで信号ステータスが0から1に変化すると、出力Qにステータス1が現れます。開始入力のステータスが1から0に変化すると、時間ETが開始されます。時間ETが経過する前に入力INで0から1への変化が発生すると、タイマーの動作はリセットされます。

入力INでのステータスが1から0に変化した場合、開始は再度初期化されます。時間ET = PTが経過した後にのみ、出力Qは信号ステータス0を採用します。これは、出力が遅延付きでスイッチオフになることを意味します。

次の図は、TOF OFF遅延タイマーの動作モードを示します。

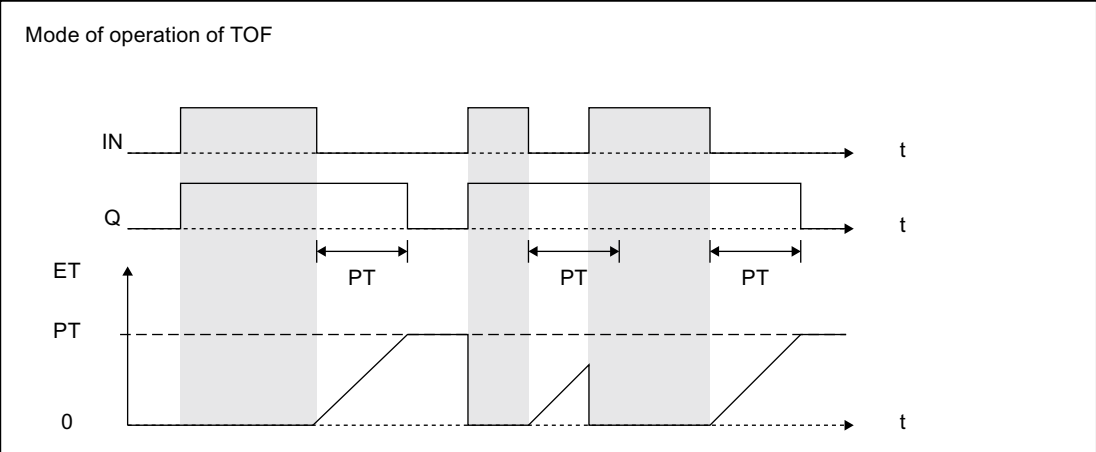


図 8-7 TOF OFF遅延時間の動作モード

以下の表に呼び出しパラメータを示します。

表 8-20 TOFの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
IN	Input	BOOL	開始入力
PT	Input	TIME	入力INでの立ち下がりエッジが遅延される時間
Q	出力	BOOL	時間のステータス
ET	出力	TIME	経過時間

リアルタイムクロックRTC

SET上での立ち上がりエッジにより、リアルタイムクロックがPDTの値に設定されます。PDTはCDTにコピーされます。

READがTRUEに設定されると、現在のシステム時刻が読み取られ、出力CDTで使用可能になります。

表 8-21 RTCの呼び出しパラメータ

識別子	パラメータ	データタイプ	説明
SET	Input	BOOL	時刻の設定、デフォルトはFALSE
READ	Input	BOOL	時刻の読み取り、デフォルトはFALSE
PDT	Input	DT	リアルタイムクロックに設定される値。デフォルトはDT#0001-01-01-0:0:0。 値がSIMOTIONデバイスのリアルタイムクロックのデフォルト値より小さい場合、リアルタイムクロックはそのデフォルト値に設定されます(たとえば、C230-2では DT#1994-01-01-00:00:00)。

識別子	パラメータ	データタイプ	説明
CDT	出力	DT	現在のシステム時刻

リアルタイムクロックの設定の精度は1 msです。この正確度は位置制御サイクルクロックに依存します。

8.6 ビット列データタイプの分割

8.6.1 ビット文字列データタイプの分割の概要

次のファンクションを使用して、ビット文字列データタイプの変数を、複数の低レベルデータタイプの変数に分割できます。

逆ファンクションもファンクションとして実装されています(「ビット文字列データタイプの結合」を参照)。

下記も参照

ビット列データタイプの結合に関する全般的な情報 (ページ 318)

8.6.2 _BYTE_TO_8BOOLファンクションブロック

このファンクションは、データタイプBYTEの変数をデータタイプBOOLの8つの変数に分割します。

ユーザーインターフェースのプロトタイプ

```
FUNCTION_BLOCK _BYTE_TO_8BOOL
VAR_INPUT
    bytein : BYTE;
END_VAR
VAR_OUTPUT
    bit0,           // 最下位ビット
    bit1, bit2, bit3, bit4, bit5, bit6,
    bit7: BOOL;     // 最上位ビット
END_VAR
// ... (コード)
END_FUNCTION_BLOCK
```

入力パラメータ

bytein

データタイプ: BYTE

データタイプBOOLの8つの変数に分割するデータタイプBYTEの変数。

出力パラメータ

bit0

...

bit7

データタイプ: BOOL

入力パラメータの個々のビットが入った8つの変数

bit0: 最下位ビット

...

ビット7: 最上位ビット

8.6.3 _WORD_TO_2BYTEファンクションブロック

このファンクションは、データタイプWORDの変数をデータタイプBYTEの2つの変数に分割します。

ユーザーインターフェースのプロトタイプ

```
FUNCTION_BLOCK _WORD_TO_2BYTE
VAR_INPUT
    wordin : WORD;
END_VAR
VAR_OUTPUT
    byte0,           // 下位バイト
    byte1: BYTE;     // 上位バイト
END_VAR
// ... (コード)
END_FUNCTION_BLOCK
```

入力パラメータ

wordin

データタイプ: WORD

データタイプBYTEの2つの変数に分割するデータタイプWORDの変数。

出力パラメータ

byte0 (オプション)
byte1 (オプション)
データタイプ: BYTE
入力パラメータの個々のバイトが入った2つの変数
byte0: 下位バイト
バイト1: 上位バイト

8.6.4 _DWORD_TO_2WORDファンクションブロック

このファンクションは、データタイプDWORDの変数をデータタイプWORDの2つの変数に分割します。

ユーザーインターフェースのプロトタイプ

```
FUNCTION_BLOCK _DWORD_TO_2WORD
VAR_INPUT
    dwordin : DWORD;
END_VAR
VAR_OUTPUT
    word0,          // 下位ワード
    word1: WORD;    // 上位ワード
END_VAR
// ... (コード)
END_FUNCTION_BLOCK
```

入力パラメータ

dwordin
データタイプ: DWORD
データタイプWORDの2つの変数に分割するデータタイプDWORDの変数。

出力パラメータ

word0
word1
データタイプ: WORD
入力パラメータの個々のバイトが入った2つの変数
word0: 下位ワード
ワード1: 上位ワード

8.6.5 _DWORD_TO_4BYTEファンクションブロック

このファンクションは、データタイプDWORDの変数をデータタイプBYTEの4つの変数に分割します。

ユーザーインターフェースのプロトタイプ

```
FUNCTION_BLOCK _DWORD_TO_4BYTE
VAR_INPUT
    dwordin : DWORD;
END_VAR
VAR_OUTPUT
    byte0,           // 最下位バイト
    byte1, byte2,
    byte3: BYTE;     // 最上位バイト
END_VAR
// ... (コード)
END_FUNCTION_BLOCK
```

入力パラメータ

in	
データタイプ:	DWORD
データタイプBYTEの4つの変数に分割するデータタイプDWORDの変数。	

出力パラメータ

byte0

byte1

byte2

byte3

データタイプ: BYTE

工場出荷時の設定(デ BYTE#0
フォルト)

入力パラメータの個々のバイトが入った2つの変数

byte0: 最下位バイト

...

バイト3: 最上位バイト

8.7 SIMATIC S7コマンドのエミュレーション

8.7.1 一般

次のファンクションブロックは、SIMATIC S7のカウンタおよびタイマー用コマンドとインターフェースの互換性があります。リファレンスマニュアル『SIMATIC 命令文リスト(STL) for S7-300/400』を参照してください。

注記

通常は、IEC 61131-3に基づいたファンクションブロック(カウンタおよびタイマー)を使用する必要があります。

下記も参照

カウンタの概要 (ページ 416)

タイマー (ページ 422)

8.7.2 _S7_COUNTERファンクションブロック

ユーザーインターフェースのプロトタイプ

```
FUNCTION_BLOCK _S7_COUNTER
  VAR_INPUT
    CU: BOOL;
    CD: BOOL;
    PV : INT;
    S:  BOOL;
    R :  BOOL;
  END_VAR
  VAR_OUTPUT
    Q:  BOOL;
    CV : INT;
  END_VAR
  // ... (コード)
END_FUNCTION_BLOCK
```

入カパラメータ

CU

データタイプ: BOOL
カウントアップ(エッジを伴う)

CD

データタイプ: BOOL
カウントダウン(エッジを伴う)

PV

データタイプ: INT
プリセット値

S

データタイプ: BOOL
プリセット値のセット(エッジを伴う)

R

データタイプ: BOOL
カウンタのリセット(静的)

出カパラメータ

Q

データタイプ: BOOL
カウンタ状態

CV

データタイプ: INT
カウンタの値

8.7.3 _S7_TIMERファンクションブロック

このファンクションブロックは、SIMATIC S7の時間ファンクションのエミュレートに使用します。

ユーザーインターフェースのプロトタイプ

```
FUNCTION_BLOCK _S7_TIMER
VAR_INPUT
    T_Type : WORD;
    PR : BOOL;
    S : BOOL;
    R : BOOL;
    TV : TIME;
    TV_BCD : WORD;
END_VAR
VAR_OUTPUT
    Q : BOOL;
    BI : TIME;
END_VAR
// ... (コード)
END_FUNCTION_BLOCK
```

入力パラメータ

T_Type

データタイプ: WORD
実行するS7タイマーファンクション
TYPE_S7T_SI_ = 16#0001 SI SP
TYPE_S7T_SV_ = 16#0002 SV SEE
TYPE_S7T_SE_ = 16#0004 SE SD
TYPE_S7T_SS_ = 16#0008 SS SS
TYPE_S7T_SA_ = 16#0010 SA SF
TYPE_S7T_BCD_IN = 16#0020 SA SF
TYPE_S7T_SI_ = 16#0040 R FR
TYPE_S7T_SI_ = 16#0080 L LC (U; UN; X ...)

FR

データタイプ: BOOL
放す

S

データタイプ: BOOL
プリセット値のセット

R

データタイプ: BOOL
タイマーのリセット

TV

データタイプ: TIME
プリセット値(データタイプTIME)

TV_BCD

データタイプ: WORD
S7コードでのプリセット値

出力パラメータ

Q1

データタイプ: BOOL
タイマー状態

BI

データタイプ: TIME
現在の時刻

エラーの原因および効率的なプログラミング

9.1 プログラミング中のエラーソース

9.1.1 プログラミング中のエラーソース

プログラミングに関する最も重要なエラーの原因は、次のとおりです。
このセクションでは、これらのエラーの原因を取り除くための解決の可能性についても説明します。

9.1.2 算術式を割り当てるときのデータタイプのチェック

算術式では、結果は必ずその式の中の最大数のフォーマットで計算されます。

式の値は、次の場合にのみ変数に割り当てることができます。

- 計算式と割り当てる変数が同じデータタイプ。
- 計算式のデータタイプが、割り当てる変数のデータタイプに暗黙的に変換できる。

そのため、式を変数に割り当てる場合、変数が十分大きなデータタイプであることを確認します。または、大きすぎる式の一部に対して明示的なデータタイプ変換を行います(『*SIMOTIONの基本機能*』機能マニュアルを参照)。

例は、参照してください。。

適宜、数値のデータタイプを明示的に指定します。たとえば、数値127がUSINTでなくデータタイプUINTとなる場合、UINT#127と指定します。

下記も参照

数値データタイプおよびビットデータタイプの変換用ファンクション (ページ 304)

9.1.3 周期的タスクのファンクションの起動の毎回のチェック

位置決め軸用ファンクションなどのTOファンクションは、それが動作していない場合にのみ周期的動作で起動する必要があります。

そうしないと、サイクルが繰り返されるたびにプログラムのコマンドが実行されます。

周期的タスクでTOファンクションを起動するための条件をプログラムできます。たとえば、コマンドの実行時に設定される補助変数の内容により起動できます。

表 9-1 周期的タスクで正しくTOファンクションの起動を行う例

```
//...
IF myStart = 0 THEN          // 補助変数が設定されていない場合
    myStart := 1;             // 補助変数を設定 (ファンクションの起動)
    myCommandID := _getCommandId ();
    myFC := _pos (axis := myAxis, // ファンクションの実行
                  position      := position_1,
                  nextCommand    := IMMEDIATELY,
                  commandID      := myCommandID);
END_IF;
//...
IF myAxis.positioningState.actualPosition = position_1 THEN
    myStart := 0;             // ファンクションの実行が必要な場合
                                // 補助変数をリセット
END_IF;
//...
```

9.1.4 周期的タスクでの待機時間

コマンドの移行を、`_pos`コマンドなどの周期的タスクでシステムファンクションを使用するときの条件にリンクしている場合、タイムアウトが発生して、それによりCPUが停止する場合があります。

これは、`nextCommand`パラメータにWHEN_MOTION_DONEなどのIMMEDIATELY以外の値が想定された任意のシステムファンクションで発生する可能性があります。

これは、ステップ有効化条件または`[waitTime]`などを使用したプログラムされた待機時間の結果として、SIMOTION SCOUTで設定されたサイクルタイムを超えた場合に、タイムアウトが発生します。

注記

周期的タスクで `[waitTime]`などの待機時間に関するコマンドを使用しないで下さい。

周期的タスクのシステムファンクションでのみ入力パラメータ`nextCommand := IMMEDIATELY`を使用します。

9.1.5 正しいcommandIdパラメータの使用

すべてのTOコマンドには、コマンドIDのパラメータが含まれる必要があります。「テクノロジーファクションの入力パラメータ」を参照してください。

適切なコマンドを呼び出す前に、コマンド[_getCommandId]を使用して全プロジェクトで一意的なコマンドIDを取得できます。

コマンドIDをローカル変数に保存し、それをTOコマンドのパラメータとして使用します。

または、`commandId:=_getCommandId()`でパラメータとして直接ファンクション呼び出しを使用します。

この一意的なコマンドIDを使用して、モーションコマンドの状態をチェックする必要があります。たとえば、[_getStateOfAxisCommand]を使用して、位置設定モーションの状態をチェックします。

システムは、コマンドIDからのみモーションコマンドを一意的に識別できます。

表 9-2 コマンドIDを使用したTOファンクションの使用例

```
//...
VAR
    myCommandID                : commandIdType;
    myState                    : StructRetCommandState;
END_VAR
//...
myCommandID := _getCommandId ();
// 一意的IDの保存
myFC := _pos (axis := myAxis,
// IDを使用したファンクションの実行
    position      := position_1,
    nextCommand   := IMMEDIATELY,
    commandID     := myCommandID);
myState := _getStateOfAxisCommand (axis:=myAxis,
    commandID     := myCommandID);

// 状態チェック
IF myState.commandIdState = WAITING_FOR_SYNC_START THEN
    ;
//...
END_IF;
//...
```

下記も参照

テクノロジーファクションの入力パラメータ (ページ 68)

[_getCommandId]ファンクション (ページ 368)

[_getSyncCommandId]ファンクション (ページ 369)

9.1.6 エラーの追跡(STプログラム)

コンパイル時に次のようなエラーメッセージが出力される場合があります。

Error 7000, 7010, 7011, or 7014 in Line ...

A syntax error has occurred. 考えられる原因:

- *Incorrectly terminated control structures (e.g. END_IF missing)*
- *Statements not terminated with ;*
- *Missing parentheses*

最初に、指定された行をチェックして、エラーがそこで実際に発生しているかどうかを調べます。つまり、制御構造が終結していないか、行の最後にセミコロンを忘れていないか、括弧がぬけていないかなどです。

指定されたエラーが見つからない場合は、エラーを追跡する必要があります。

1. エラーメッセージのある行の前に、プログラムのブロックごとにコメントアウトします。つまり、当該のセクションを文字ペア(*と*)で囲みます。
エラーメッセージのある行はコメントアウトしないで下さい。
2. プログラムを再コンパイルします。
3. ステップ1と2を行った後もエラーが発生するようなら、コメントアウトのセクションが小さすぎます。エラーメッセージが消えるまで、それを広げます。
4. エラーメッセージが出力されなくなったら、エラーはコメントアウトされたセクションに含まれています。
ここで、エラーメッセージが再度出力されるようになるまで、コメントアウトのセクションの大きさを1行ずつ小さくして再コンパイルします。
最後にエラーがなかった行がエラーメッセージがある行です。

注記

『STプログラミングマニュアル』の付録にあるコンパイラのエラーメッセージの全リストでそれを調べることができます。

9.1.7 ダウンロード時のエラー

プログラムのダウンロード時にエラーメッセージが発生した場合、SIMOTION SCOUT詳細ビューに表示されるログは停止します。ログでエラーの原因を探します。ハードウェアコンフィグレーションやプログラムをチェックして、存在しないアドレスが使用されているかなどを調べます。

ハードウェアコンフィグレーションおよびアドレスの詳細については、『SIMOTION SCOUT設定マニュアル』を参照してください。

9.1.8 CPUがRUNに切り替わらない

プログラムの開始直後にCPUがSTOPモードに戻った場合、SCOUTのデバイス診断およびアラームウィンドウをチェックします。STOPの原因は、診断バッファに入っています。

CPU

STOPの原因となっている可能性のあるテクノロジーオブジェクトのアラームは、アラームウィンドウに表示されます。

9.1.9 CPUがSTOP状態に移行

説明

CPUがSTOP動作状態に変わった場合、SCOUTのデバイス診断およびアラームウィンドウをチェックします。STOPの原因は、診断バッファに入っています。CPU STOPの原因となっている可能性のあるテクノロジーオブジェクトのアラームは、アラームウィンドウに表示されます。

CPUがSTOPの後に動作した理由として、次のような原因が考えられます。

- 不正な直接I/Oアクセス
- RESTART時にコンフィグレーションデータまたはシステム変数によるTOへのアクセス(ただし、V4.1以降はシステム変数の置換値が使用可能)
- PeripheralFaultTaskがない(プロセスイメージへの不正なアクセスがあった場合)
- TechnologicalFaultTaskがない(テクノロジーオブジェクトにエラーが合った場合)
- プログラムでのエラーの処理(またはExecutionFaultTaskがない)
- IPO/サーボタスクのタイマー値オーバーフロー
- BackgroundTaskのタイマー値オーバーフロー
- TimerInterruptTaskのタイマー値オーバーフロー
- Simotion - ドライブのサインオブライフモニタ
- STOP応答によるTOアラーム(20001など)。
- システム変数を介したモードの設定

システム変数`modeOfOperation`を使用して現在の動作モードを表示または変更します。

構文

```
modeOfOperation :EnumDeviceModeOfOperation
//読み込み可能、書き込み可能、即座に有効
EnumDeviceModeOfOperation:
[
    _STOP I
    _RUN
]
```

たとえば、これを使用してデバイスシステム変数を書き込むことにより、ローカルHMIからSTOPに移行したSIMOTION CPUを再度RUNに切り替えることができます。

下記も参照

プログラムでの実行エラー (ページ 94)

システム変数、コンフィグレーションデータ、およびI/O変数に直接アクセス時のアクセスエラー (ページ 95)

SystemInterruptTasks (ページ 157)

テクノロジーオブジェクトで発生しうるエラー (ページ 109)

9.1.10 システムクロックのチェックおよび設定

しばしば、不正確に設定されたシステムサイクルクロック(位置制御サイクルクロック、補間サイクルクロック、PWMサイクルクロック)が原因で、SIMOTIONデバイスがSTOPモードになります。

SynchronousTask

(TControlテクノロジーパッケージのタスク、IPOsynchronousTask、IPOsynchronousTask_2)の実行時間をチェックします。

ユーザープログラムまたはシステムプログラムの個々のタスクは、SIMOTION SCOUTのシステムサイクルクロックで設定された時間より多く必要な場合があります。

また、SynchronousTaskの実行時間が最小になるよう試行します。
プログラムを極力MotionTaskに移行して、必要に応じてプログラムを分割します。

個々のタスク間は整数比になるようにします。
そうしないと、優先度の低いSynchronousTaskは定期的に起動しなくなります。

不要なタスクを無効化します。

注記

TControlテクノロジーパッケージのシステムタスクは、システムサイクルクロック設定ウィンドウで無効化できます。

デバイス診断のチェック、アラームウィンドウの解釈、およびシステムクロックのチェック/変更の方法の詳細は、オンラインヘルプを参照してください。

9.1.11 REALまたはLREAL変数の比較

REAL変数またはLREAL変数(および軸位置などの対応するシステム変数)を相互に比較する場合、"="を使用しないで下さい。

内部の数値表現が異なるため、比較する数値は同一になりません。

かわりにモーションの方向などを評価したり、">"または"<"や"到達した位置ウィンドウ"のシステム変数を使用する必要があります。

9.1.12 範囲違反のチェック

範囲違反、つまりデータタイプの範囲制限の違反は、コンパイラでは警告されません。そのため、算術演算で変数を使用する場合、必ず起こりうる範囲違反をチェックする必要があります。

表 9-3 範囲違反のチェックの例

```
PROGRAM myRange
  VAR
    a,b : SINT := 100;
    c: SINT;
  END_VAR

  c := a + b;
  // cが範囲外の場合、プログラム終了
  IF (a > 0) AND (c < a) AND (c < b) OR
    (a < 0) AND (c > a) AND (c > b) THEN
    // 範囲違反
    RETURN;
  ELSE
    ; // OK
  END_IF;
END_PROGRAM
```

9.1.13 ローカルデータスタックのサイズの設定

実行システムを構成する場合、各タスクに確保されたローカルデータスタックを設定します。

プログラム構造ファンクションを使用して、スタックに呼び出されるすべてのPOU(FCおよびFB)に関するプログラムのメモリ要件についての情報を取得します(『STプログラミングマニュアル』の「プログラムの構造」を参照)。

コンフィグレーション時に、十分な空間があることを確認します。

たとえば、RUNの際のダウンロード時に、ローカルデータスタックに一時的な追加記憶域が必要な場合があります。

予約されたローカルデータスタックがRAM内の使用可能なメモリ空間を超えた場合、エラーメッセージが出力されてダウンロード操作は中止されます。

確保されたローカルデータスタックがプログラムの実行に不十分な場合、プログラムは実行時に中止されます。

エラーが発生する場合、次をチェックします。

- RAMの使用率(デバイス診断)
- 大きなデータ構造(配列、構造体)を使用しているソース(プログラムなど)。たとえば、相互参照表を使用する場合など。
- タスクのローカルデータスタックのサイズ

下記も参照

コンフィグレーション時の設定 (ページ 229)

9.2 効率的なプログラミング

9.2.1 効率的なプログラミング - 概要

多くの制御システムやプログラミング環境では、次のような矛盾が常に発生しています。

- 更新の容易性と拡張性を重視し、高度に構造化され簡潔化されたユーザープログラムは、実行時間に関して最適に実行されない。
- 一方、実行時間に最適化されたプログラムは、拡張や変更が困難である。

以下は、実行時間に最適なプログラミングおよび変更に最適なプログラミングについて説明します。

タスクに応じて、またはローカルの最適化によって、これらのオプションのいずれかに注目してください。

9.2.2 実行時間に最適化されたプログラミング

9.2.2.1 実行時間に最適化されたプログラミング

以下は、実行時間に最適化されたプログラミングについて説明します。

実行時間に最適化されたプログラミングの方法は、構造化プログラミングのルールに沿わないこともあるので注意してください。

9.2.2.2 入力および出力へのアクセスの最適化

周期的タスクのプロセスイメージへのアクセスは、入出力への直接アクセスよりはるかに高速です(参照)。

そのため、I/O変数は、その変数を使用するタスクのプロセスイメージに割り当てます。

高速アクセスのほかにも、I/O変数はタスクの実行時間中は一貫性が保たれるという利点があります。

9.2.2.3 システム変数へのアクセスの最適化

システム変数へのアクセス(「システム変数

(ページ 83)」を参照)は、ダイナミックメモリに格納された変数(ローカル変数、保持されないユニット変数)へのアクセスに比べて非常に時間がかかります。

したがって、高速の周期的タスクでは(SynchronousTasksなど)、システム変数への直接アクセスは少数のみ可能です。

そのため、システム変数に多数アクセスしている場合は、サイクル(周期的タスクのプログラム)の開始時に全体のシステム変数の構造体を、対応するシステムデータタイプのローカル変数にコピーすることを推奨します。

プログラムのサイクル時は、この変数にアクセスします。

ローカル変数を宣言する際のデータタイプは、SIMOTIONテクノロジーオブジェクト(TO)のパラメータマニュアルに記載されています。

9.2.2.4 最適な変数宣言

宣言ブロック(VAR/END_VARなど)内の変数は、サイズの小さい順に並べます。
これにより、メモリ空間を最適に使用できます。

0以外の値での変数の初期化は、必要な場合のみ行います。
タスクまたはPOUの開始時の変数の初期化は時間がかかります。
これは、逐次的タスクに割り当てられたプログラムの変数および一時変数の場合顕著です。

9.2.2.5 ファンクションブロックのパラメータへのアクセスの最適化

値を処理するファンクションブロック(FB)の作成時に、2つの方法のいずれかを選択できます。
FBの入力変数をVAR_INPUTに割り当て、そこで変数を処理し、出力パラメータへの結果をVAR_OUTPUTに割り当てることができます。

大量のデータをFBに伝送する場合、input/output
パラメータ(VAR_INPUTおよびVAR_OUTPUT)を使用するより、in/outパラメータ(VAR_IN_
OUT)を使用するほうが高速になります。
コピーがなくなるためパラメータは高速に伝送できますが、一定の条件下でFBからの変数の
アクセスに時間がかかる場合があります。

通知
in/outパラメータでユニット変数またはグローバルデバイス変数にアクセスする場合、他の タスクもこれらの変数に同時にアクセスできるので注意してください。

9.2.2.6 ファンクションブロック呼び出しの最適化

SIMOTION Kernel V3.0以下のみ対象

ファンクションブロック(FB)のインスタンスを呼び出す場合、インスタンスデータはスタックにコピーされます(参照)。インスタンスを終了すると、それらはスタックからインスタンスに戻されます。
実行時間が長くなるので、FBに大量のインスタンスデータを含めないで下さい。

9.2.2.7 プログラム構造の最適化

STソースファイルとそれに含まれるPOUは、明確かつ簡潔に構造化されるようにします。
ただし、ユニット内のファンクションおよび変数より、インポートされたユニットのファンクションおよび変数へのアクセスには時間がかかるので、ソースファイルをモジュール化しすぎないで下さい。

9.2.2.8 実行システムの最適化

IPOSynchronousTaskは1つのプログラムのみ割り当てます。
これにより、実行時のタイムアウトのリスクが低減します。

同期的に実行されるファンクションはMotionTask内でのみ使用します。
(同期処理時は、保留コマンドの処理が条件を満たすまで(たとえば完了するまで)次のコマンドは実行されません。)

あまり多くのMotionTaskを同時に有効にしないでください。

周期的タスクのプログラミングをより効率的にするには(主にBackgroundTask)、シーケンス制御プログラミングに移行します。

つまり、状況判断を介してプログラムフローを意図的に制御し、現在の状態で関連するコードの部分のみを実行します(CASE文などを使用)。

さらに、レベルでトリガされたプログラミングよりエッジでトリガされたプログラムを推奨します。

(条件のレベルがTRUEの場合)関連するコードを周期的に実行するのではなく、1回のみ実行します。

これは立ち上がりエッジを満たす条件をクエリーすることによって実行します(条件の新しい状態はTRUE、古い状態はFALSE)。

9.2.3 変更に最適化されたプログラミング

9.2.3.1 変更に最適化されたプログラミング

以下は、変更に最適化されたプログラミングについて説明します。

RUN時のダウンロードについては、参照してください。RUN時のダウンロード(ページ 244)

9.2.3.2 1つのユニットでの保持される変数の宣言

すべての保持される変数は、1つのユニットのインターフェースセクションで宣言します。これには、次の利点があります。

- これにより、限られたメモリ空間を最適に使用できます。
- このユニットのインターフェースセクションが変更された場合のみ、変数は初期化されます。
- V4.1以降は、複数の宣言ブロックを使用することもできます。参照してください。複数のVAR_GLOBAL、VAR_GLOBAL RETAINブロックの使用(ページ 232)。

9.2.3.3 1つのユニットでのHMI変数

HMIデバイスにエクスポートする変数(「HMI(マンマシンインターフェース)」および『SIMOTIONモーションコントロール基本機能』機能マニュアルを参照)を別のユニットのインターフェースセクションに宣言します。

- HMIプロジェクト(WinCC flexible、ProToolなど)は、このユニットのインターフェースセクションが変更された場合にのみ再度ダウンロードする必要があります。

- オプションで、試運転時に適合性チェックを無効にできます。

(有効なハードウェアなどに)実行される不適合性のチェックはないので注意してください。

- また、HMI関連のデータをマーキングすることもできます。参照HMI関連データのマーキング(ページ 234)。

9.2.3.4 デバイスグローバル変数およびユニットグローバル変数の使用

説明

ソースでは、(プロジェクトナビゲータにより)デバイスグローバル変数を使用するよりグローバルユニット変数を使用した方が適しています。

利点:

- 変数の構造体を使用できます。
- (StartupTaskのプログラムを介して)STOP-RUN移行の変数の初期化(初期値)が可能です。
- (新しい宣言ブロックに)新しく作成されたグローバルユニット変数は、RUN時にダウンロードすることもできます

手順

1. ソースファイルのインターフェースセクションにユニットグローバル変数を定義します。
保持される変数およびHMI変数は、それぞれ同様に個別のソースファイルまたは宣言ブロックで宣言する必要があります(上を参照)。
2. 変数の初期化が含まれるプログラムをStartupTaskに割り当てます。
変数は、起動時に定義された初期値に設定されます。

Parameter	I/O symbols	Structures	Enumerations	Connections	
	Structure name	Element name	Data type	Array length	Initial value
1	MyStruct	IntValue	INT		0
2		RealValue	REAL		0.0
3		BitValue	BOOL		FALSE
4					

図 9-1 MCCソースファイルの構造体の宣言の例

Parameter	I/O symbols	Structures	Enumerations	Connections	
	Name	Variable type	Data type	Array length	Initial value
1	m	VAR_GLOBAL CONSTANT	INT		16
2	BitArray	VAR_GLOBAL	BOOL	m	16(False)
3	IntArray	VAR_GLOBAL	INT	m	16(0)
4	ReadArray	VAR_GLOBAL	REAL	m	16(0.0)
5	StructVar	VAR_GLOBAL	MyStruct		
6					

図 9-2 MCCソースファイルのパラメータの宣言の例

表 9-4 STソースファイル(ユニット)の宣言およびプログラムの例

```

INTERFACE
  //グローバルタイプ
  TYPE
    MyStruct : STRUCT
      Intvalue : INT;
      Realvalue: REAL;
      Bitvalue : BOOL;
    END_STRUCT
  END_TYPE
  //グローバル定数
  VAR_GLOBAL CONSTANT
    n: INT := 0;
    M: INT := 15;
  END_VAR
  //グローバル変数
  VAR_GLOBAL
    Bitarray : ARRAY [n..m] OF BOOL := [16 (FALSE)];
    Intarray : ARRAY [0..15] OF INT := [16 (0) ];
    Realarray: ARRAY [0..15] OF REAL:= [16 (0.0) ];
    StructVar: MyStruct;
  END_VAR
  //プログラム
  PROGRAM Init;
  //インターフェースの終了
END_INTERFACE

IMPLEMENTATION
PROGRAM Init
  //////////////////////////////////////
  //設定時の変数の初期化//
  //////////////////////////////////////
  StructVar.Intvalue :=0;
  StructVar.Realvalue:=0;
  StructVar.Bitvalue :=FALSE;
END_PROGRAM
END_IMPLEMENTATION

```

プログラムはStartupTaskに割り当てられます。

ユニットグローバル変数を異なるソースファイルで使用するときは、必ずソースファイルを、その宣言(USES)が含まれるソースファイルと組み合わせる必要があります。

『MCCプログラミングマニュアル』の「別のプログラムソースまたはライブラリへの接続」または『STプログラミングマニュアル』の「インポートするユニットのインターフェースまたは実装セクションでのUSES文の使用」も参照してください。

9.2.3.5 MotionTaskの集中起動およびリセット

プログラミングの明確さを高めるために、MotionTaskの起動とリセットを1箇所に集中させてプログラムします。

付録A

A.1 シンボリック定数

次の表は、個々の変数名に使用できない予約済の定数名を示します。

表 A-1 Taskstartinfoの記号定数

シンボリック定数	データタイプ	規格値
_SC_ALARM_CONFIGURATION	UDINT	404
_SC_ARRAY_BOUND_ERROR_READ	UDINT	502
_SC_ARRAY_BOUND_ERROR_WRITE	UDINT	503
_SC_BACKGROUND_TIMER_OVERFLOW	UDINT	300
_SC_BATTERY_WARNING_LEVEL1	UDINT	211
_SC_BATTERY_WARNING_LEVEL2	UDINT	212
_SC_CYCLE_TIMER_OVERFLOW	UDINT	301
_SC_DEVICE_COMMAND	UDINT	401
_SC_DIAGNOSTIC_INTERRUPT	UDINT	201
_SC_DIVISION_BY_ZERO	UDINT	500
_SC_DP_CLOCK_DETECTED	UDINT	207
_SC_DP_SLAVE_NOT_SYNCHRONIZED	UDINT	210
_SC_DP_SLAVE_SYNCHRONIZED	UDINT	209
_SC_DP_SYNCHRONIZATION_LOST	UDINT	208
_SC_EXCEPTION	UDINT	403
_SC_EXTERNAL_COMMAND	UDINT	402
_SC_IMAGE_UPDATE_FAILED	UDINT	204
_SC_IMAGE_UPDATE_OK	UDINT	206
_SC_INVALID_ADDRESS	DINT	-1
_SC_INVALID_FLOATING_POINT_OPERATION	UDINT	501
_SC_IO_MODULE_NOT_SYNCHRONIZED	UDINT	215
_SC_IO_MODULE_SYNCHRONIZED	UDINT	214
_SC_MODE_SELECTOR	UDINT	400
_SC_PC_INTERNAL_FAILURE	UDINT	205
_SC_PULL_PLUG_INTERRUPT	UDINT	213
_SC_PROCESS_INTERRUPT	UDINT	200
_SC_STATION_DISCONNECTED	UDINT	202
_SC_STATION_RECONNECTED	UDINT	203
_SC_TO_INSTANCE_NOT_EXISTENT	UDINT	506

A.1 シンボリック定数

シンボリック定数	データタイプ	規格値
_SC_VARIABLE_ACCESS_ERROR_READ	UDINT	504
_SC_VARIABLE_ACCESS_ERROR_WRITE	UDINT	505

表 A-2 タスクステータスの記号定数

シンボリック定数	データタイプ	16進表記
TASK_STATE_INVALID	DWORD	16#0000
TASK_STATE_LOCKED	DWORD	16#0100
TASK_STATE_RUNNING	DWORD	16#0004
TASK_STATE_STOP_PENDING	DWORD	16#0001
TASK_STATE_STOPPED	DWORD	16#0002
TASK_STATE_SUSPENDED	DWORD	16#0020
TASK_STATE_WAIT_NEXT_CYCLE	DWORD	16#0040
TASK_STATE_WAIT_NEXT_INTERRUPT	DWORD	16#0080
TASK_STATE_WAITING	DWORD	16#0010

表 A-3 アラームメッセージの記号定数

シンボリック定数	データタイプ	16進表記
ALARMS_ERROR	DWORD	16#8000
ALARMS_QSTATE	DWORD	16#0100
ALARMS_STATE	DWORD	16#0001
DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE	DWORD	16#0006
DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED	DWORD	16#0005
DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID	DWORD	16#0001
DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR	DWORD	16#0009
DSC_SVS_DEVICE_ALARMS_IV_CALL	DWORD	16#0004
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL	DWORD	16#0007
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP	DWORD	16#0008
DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED	DWORD	16#0002
DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED	DWORD	16#0003
DSC_SVS_DEVICE_ALARMS_NO_ENTRY	DWORD	16#0010

表 A-4 基本データタイプの数値範囲の制限用記号定数

シンボリック定数	データタイプ	規格値	16進表記
SINT#MIN	SINT	-128	16#80
SINT#MAX	SINT	127	16#7F
INT#MIN	INT	-32768	16#8000
INT#MAX	INT	32767	16#7FFF

シンボリック定数	データタイプ	規格値	16進表記
DINT#MIN	DINT	-2147483648	16#8000_0000
DINT#MAX	DINT	2147483647	16#7FFF_FFFF
USINT#MIN	USINT	0	16#00
USINT#MAX	USINT	255	16#FF
UINT#MIN	UINT	0	16#0000
UINT#MAX	UINT	65535	16#FFFF
UDINT#MIN	UDINT	0	16#0000_0000
UDINT#MAX	UDINT	4294967295	16#FFFF_FFFF
T#MIN TIME#MIN	T#0ms	TIME	16#0000_0000
T#MAX TIME#MAX	T#49d_17h_2m_47s_295ms	TIME	16#FFFF_FFFF
TOD#MIN TIME_OF_DAY#MIN	TOD#00:00:00.000	TOD	16#0000_0000
TOD#MAX TIME_OF_DAY#MAX	TOD#23:59:59.999	TOD	16#0526_5BFF

表 A-5 当該のデータタイプの不正な値の記号定数

シンボリック定数	データタイプ	意味
STRUCTALARMID#NIL	StructAlarmId	不正なアラームID
STRUCTTASKID#NIL	StructTaskId	不正なTaskId
TO#NIL	ANYOBJECT	不正なテクノロジーオブジェクト

A.2 予約済みの識別子

本章ではSTプログラミング言語およびSIMOTIONデバイスのシステムファンクションおよびシステム変数で使用するすべての識別子をアルファベット順に示します。
これらの識別子は、ユーザープログラムで使用しないで下さい。

保護されたCamテクノロジーパッケージの識別子は、テクノロジーパッケージのパラメータマニュアルに記載されています。

アイコン

_abortAllReadWriteDriveParameterJobs
 _abortReadWriteRecordJobs
 _activateConfiguration
 _activateDpSlave
 _activateDpSlaveAddress
 _activateNameOfStation

_activateTo
_alarmS
_alarmSc
_alarmScId
_alarmSId
_alarmSq
_alarmSqlId
_allocateTokenToVariableName
_AND
_BOOL
_BYTE_FROM_8BOOL
_BYTE_TO_8BOOL
_changeOperationMode
_checkEqualTask
_checkExistingUnitDataSet
_configurationManagement
_cpuData
_cpuDataRW
_DATE
_DATE_AND_TIME
_deactivateDpSlave
_deactivateTo
_deallocateTokenToVariableName
_deleteAllUnitDataSets
_deleteUnitDataSet
_DINT
_disableScheduler
_DWORD_FROM_2WORD
_DWORD_FROM_4BYTE
_DWORD_TO_2WORD
_DWORD_TO_4BYTE
_enableDpInterfaceSynchronizationMode
_enableScheduler
_exportUnitDataSet
_exportUnitDataSet2
_finite

_getActiveDpSlaveAddress
_getActiveNameOfStation
_getAlarmId
_getAverageTaskIdRunTime
_getAverageTaskRunTime
_getBit
_getCommandId
_getConfigurationData
_getCurrentTaskIdRunTime
_getCurrentTaskRunTime
_getDataByToken
_getDeviceId
_getDoIndexNumberFromLogAddress
_getDpStationAddressFromLogDiagnosticAddress
_getGeoAddressFromLogAddress
_getInOutByte
_getInternalTaskIdIdx
_getInternalTaskIdx
_getInternalTimeStamp
_getIO_Part_4
_getIPConfig
_getLogDiagnosticAddressFromDpStationAddress
_getMaximalTaskIdRunTime
_getMaximalTaskRunTime
_getMemoryCardId
_getMinimalTaskIdRunTime
_getMinimalTaskRunTime
_getNextLogAddress
_getPendingAlarms
_getPnInterfacePortNeighbour
_getSafeValue
_getSegmentIdentification
_getStateOfAllDpSlaves
_getStateOfAllDpStations
_getStateOfDiagnosticDataCommand
_getStateOfDpSlave

_getStateOfProcessInterruptCommand
 _getStateOfRecordCommand
 _getStateOfSingleDpSlave
 _getStateOfTask
 _getStateOfTaskId
 _getStateOfTo
 _getStateOfUnitDataSetCommand
 _GetStateOfXCommand
 _getStationType
 _getSyncCommandId
 _getSystemTime
 _getTaskId
 _getTimeDifferenceOfInternalTimeStamps
 _imData
 _IMPLEMENTATION
 _importUnitDataSet
 _importUnitDataSet2
 _INT
 _INTERFACE
 _INTERFACE_AND_IMPLEMENTATION
 _internalTest
 _isNaN
 _LINT
 _loadUnitDataSet
 _loadUnitDataSet2
 _LREAL
 _MRES
 _NOT
 _OR
 _readDiagnosticData
 _readDriveFaults
 _readDriveMultiParameter
 _readDriveMultiParameterDescription
 _readDriveParameter
 _readDriveParameterDescription
 _readRecord

_readSystemClock
_REAL
_releaseSemaphore
_resetAlarmId
_resetAllAlarmId
_resetDriveObjectFault
_resetTask
_resetTaskId
_resetTechnologicalErrors
_resetUnitData
_restartTask
_restartTaskId
_resumeTask
_resumeTaskId
_RETAIN
_retriggerTaskControlTime
_retriggerTaskIdControlTime
_RUN
_S7_COUNTER
_S7_TIMER
_saveConfigData
_SavePersistentMemoryData;_SavePersistentMemoryData
_saveUnitDataSet
_saveUnitDataSet2
_SC_ALARM_CONFIGURATION
_SC_ARRAY_BOUND_ERROR_READ
_SC_ARRAY_BOUND_ERROR_WRITE
_SC_BACKGROUND_TIMER_OVERFLOW
_SC_BATTERY_WARNING_LEVEL1
_SC_BATTERY_WARNING_LEVEL2
_SC_CYCLE_TIMER_OVERFLOW
_SC_DEVICE_COMMAND
_SC_DIAGNOSTIC_INTERRUPT
_SC_DIVISION_BY_ZERO
_SC_DP_CLOCK_DETECTED
_SC_DP_SLAVE_NOT_SYNCHRONIZED

_SC_DP_SLAVE_SYNCHRONIZED
_SC_DP_SYNCHRONIZATION_LOST
_SC_EXCEPTION
_SC_EXTERNAL_COMMAND
_SC_IMAGE_UPDATE_FAILED
_SC_IMAGE_UPDATE_OK
_SC_INVALID_ADDRESS
_SC_INVALID_FLOATING_POINT_OPERATION
_SC_IO_MODULE_NOT_SYNCHRONIZED
_SC_IO_MODULE_SYNCHRONIZED
_SC_MODE_SELECTOR
_SC_PC_INTERNAL_FAILURE
_SC_PROCESS_INTERRUPT
_SC_PULL_PLUG_INTERRUPT
_SC_STATION_DISCONNECTED
_SC_STATION_RECONNECTED
_SC_TO_INSTANCE_NOT_EXISTENT
_SC_VARIABLE_ACCESS_ERROR_READ
_SC_VARIABLE_ACCESS_ERROR_WRITE
_sendProcessInterrupt
_SERVICE
_setBit
_setDataByToken
_setDeviceErrorLED
_setDpSlaveAddress
_setIO_Part_4
_setIPConfig
_setModeSelfAdaptingConfiguration
_setNameOfStation
_setSafeValue
_setSystemTime
_SHUTDOWN
_SINT
_sizeof
_startSyncCommands
_startTask

_startTaskId
_STARTUP
_startupData
_STOP
_STOPU
_suspendTask
_suspendTaskDebug
_suspendTaskId
_suspendTaskIdDebug
_synchronizeDpInterfaces
_tcpCloseConnection
_tcpCloseServer
_tcpOpenClient
_tcpOpenServer
_tcpReceive
_tcpSend
_testAndSetSemaphore
_testSFBSysDataInit
_TIME
_TIME_OF_DAY
_toggleBit
_triggerTestMode
_UDINT
_udpAddMulticastGroupMembership
_udpDropMulticastGroupMembership
_udpReceive
_udpSend
_UINT
_ULINT
_upsData
_USINT
_waitTime
_WORD_FROM_2BYTE
_WORD_TO_2BYTE
_writeAndSendMessage
_writeDriveMultiParameter

_writeDriveParameter
_writeRecord
_XOR
_Xreceive
_Xsend

A

ABORT_CONNECTION
ABORT_CURRENT_COMMAND
ABS
ACCESS_DENIED
ACOS
アクション
ACTIVATE_FALL_BACK_CONFIGURATION
ACTIVATED
ACTIVATED_NO_ALARM
ACTIVE
ACTUAL_ACTIVATED
ADD
ADD_DT_TIME
ADD_TIME
ADD_TOD_TIME
ALARM_S
ALARM_SQ
ALARMS_ERROR
ALARMS_QSTATE
ALARMS_STATE
ALL_GLOBAL
AND
ANYOBJECT
ANYOBJECT_TO_OBJECT
ANYTYPE_TO_BIGBYTEARRAY
ANYTYPE_TO_LITTLEBYTEARRAY
ARRAY
AS
ASIN

AT
ATAN
AUTOMATIC_INTERFACE_SYNCHRONIZATION

B

BCD_TO_BYTE
BCD_TO_DINT
BCD_TO_DWORD
BCD_TO_INT
BCD_TO_LWORD
BCD_TO_SINT
BCD_TO_WORD
BEGIN_SYNC
BigByteArray_to_AnyType
BIGBYTEARRAY_TOANYTYPE
BOOL
BOOL_TO_BYTE
BOOL_TO_DWORD
BOOL_TO_WORD
BOOL_VALUE_TO_DINT
BOOL_VALUE_TO_INT
BOOL_VALUE_TO_LREAL
BOOL_VALUE_TO_REAL
BOOL_VALUE_TO_SINT
BOOL_VALUE_TO_UDINT
BOOL_VALUE_TO_UINT
BOOL_VALUE_TO_USINT
BUFFERED
BY
BYTE
BYTE_TO_BCD
BYTE_TO_BOOL
BYTE_TO_DINT
BYTE_TO_DWORD
BYTE_TO_INT
BYTE_TO_SINT

BYTE_TO_UDINT
 BYTE_TO_UINT
 BYTE_TO_USINT
 BYTE_TO_WORD
 BYTE_VALUE_TO_LREAL
 BYTE_VALUE_TO_REAL

C

CASE
 cDINT_TO_UDINT
 CLOSE_ON_EXIT
 COMMAND_FAILED
 COMMAND_NOT_FOUND
 CommandIdType
 CONCAT
 CONCAT_DATE_TOD
 CONFIGURATION
 CONFIGURATION_DELETED
 CONFIGURATION_ERROR
 CONFIGURATION_ID_NOT_FOUND
 CONFIGURATION_NOT_FOUND
 CONFIGURATION_VERSION
 CONFIGURED
 CONSTANT
 COS
 CRITICAL
 CTD
 CTD_DINT
 CTD_LINT
 CTD_UDINT
 CTD_ULINT
 CTU
 CTU_DINT
 CTU_LINT
 CTU_UDINT
 CTU_ULINT

CTUD
CTUD_DINT
CTUD_LINT
CTUD_UDINT
CTUD_ULINT

D

DATA_EXCHANGE_INACTIVE
DATA_INCOMPATIBLE
DATA_INCOMPLETE
DATA_MISMATCH
DATASET_ALREADY_EXISTS
DATASET_ID_NOT_VALID
DATASET_NOT_FOUND
DATE
DATE_AND_TIME
DATE_AND_TIME_TO_DATE
DATE_AND_TIME_TO_TIME_OF_DAY
dccAux_2Clock
dccAuxClock
DEACTIVATED
DEACTIVATED_NO_ALARM
DEFAULT_VALUE
DELETE
DINT
DINT_TO_BCD
DINT_TO_BYTE
DINT_TO_DWORD
DINT_TO_INT
DINT_TO_LREAL
DINT_TO_REAL
DINT_TO_SINT
DINT_TO_STRING
DINT_TO_UDINT
DINT_TO_UINT
DINT_TO_USINT

DINT_TO_WORD
DINT_VALUE_TO_BOOL
DISABLE
DIV
DIVTIME
DO
DO_NOT_CLOSE_ON_EXIT
DO_NOT_SET_DP_ALARM
DONE
DP_1
DP_2
DP_3
DP_CLOCK_DETECTED
DP_INTERFACES_SYNCHRONIZED
DP_SLAVE_NOT_SYNCHRONIZED
DP_SLAVE_SYNCHRONIZED
DRIVE
DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE
DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED
DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID
DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR
DSC_SVS_DEVICE_ALARMS_IV_CALL
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP
DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED
DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED
DSC_SVS_DEVICE_ALARMS_NO_ENTRY
DT
DT_TO_DATE
DT_TO_TOD
DWORD
DWORD_TO_BCD
DWORD_TO_BOOL
DWORD_TO_BYTE
DWORD_TO_DINT
DWORD_TO_INT

DWORD_TO_REAL
DWORD_TO_SINT
DWORD_TO_UDINT
DWORD_TO_UINT
DWORD_TO_USINT
DWORD_TO_WORD
DWORD_VALUE_TO_LREAL
DWORD_VALUE_TO_REAL

E

effectiveTaskRuntime
ELSE
ELSIF
EN
Enable
END_ACTION
END_CASE
END_CONFIGURATION
END_EXPRESSION
END_FOR
END_FUNCTION
END_FUNCTION_BLOCK
END_IF
END_IMPLEMENTATION
END_INTERFACE
END_LABEL
END_PROGRAM
END_REPEAT
END_RESOURCE
END_STEP
END_STRUCT
END_SYNC
END_TRANSITION
END_TYPE
END_VAR
END_WAITFORCONDITION

END_WHILE
 ENO
 ENUM_TO_DINT
 EnumAccessMode
 EnumAlarmIdState
 EnumAlarmIdType
 EnumConfigurationInfold
 EnumDataType
 EnumDeviceAbortReadWriteJobs
 EnumDeviceCommandIdState
 EnumDeviceConfigurationActivationState
 EnumDeviceDataActivationState
 EnumDeviceDataScope
 EnumDeviceDpAlarmMode
 EnumDeviceIdType
 EnumDeviceKindOfData
 EnumDeviceModeOfOperation
 EnumDeviceStartupOperationMode
 EnumDeviceStateOfDpSlave
 EnumDeviceStorageType
 EnumDeviceUnitDataSetCommand
 EnumDpInterfaceSyncMode
 EnumDpInterfaceSyncState
 EnumDpSegmentId
 EnumDpSlaveSyncState
 EnumDriveFaultsType
 EnumEnableDisable
 EnumEventClass
 EnumFctGenStartStop
 EnumFctGenState
 EnumIncomingOutgoing
 EnumInOutDirection
 EnumInterfaceId
 EnumIoIdType
 EnumMemoryCardIdType
 EnumModeSelfAdaptingConfiguration

EnumNextCommandMode
EnumOperationMode
EnumPersistentDataState
EnumReqActDeactGetStateMode
EnumReqSysFunctMode
EnumSaveState
EnumSetAndGetSafeValue
EnumStartStop
EnumStateOfDpSlave
EnumStateOfDpStation
EnumStateOfTo
EnumTcpNextCommandMode
EnumToSetStateOfTo
EnumTraceState
EnumUdpCommunicationMode
EnumUpsBatteryState
EnumUpsState
EnumXCommandIdState
EnumXCommunicationMode
EnumXNextCommandEnable
EnumYesNo
EQ
EXIT
EXP
EXPD
EXPRESSION
EXPT

F

F_EDGE FROM
F_TRIG
FAILED
FALSE
FCTGEN_DISABLED
FCTGEN_ENABLED
FCTGEN_LIMIT_EXCEEDED

FCTGEN_PARAM_VALID
 FCTGEN_RUNNING
 FCTGEN_START
 FCTGEN_START_FG1
 FCTGEN_START_FG2
 FCTGEN_STARTING
 FCTGEN_STOP
 FCTGEN_STOPPING
 FCTGEN_WAIT_FG1
 FCTGEN_WAIT_FG2
 FIND
 FOR
 FROM_BACKUP
 FROM_FILE
 FROM_RAM
 FUNCTION
 FUNCTION_BLOCK
 FUNCTION_IS_ACTIVATED
 FUNCTION_IS_ACTIVE
 FUNCTION_IS_WAITING_FOR_ACTIVATION

G

GE GT
 GOTO

H

HOLD_CONNECTION
 HW_TYPE

I

IE_01
 IE_02
 IE_1
 IE_2
 IF
 IMMEDIATELY

IMPLEMENTATION
IN
IN_ACTIVATION
IN_DEACTIVATION
IN_OPERATION
INACTIVE
INCOMING
INCOMING_ALARM
INITIAL_STEP INT_TO_BCD
INPUT
INSERT
INT
INT_TO_BYTE
INT_TO_DINT
INT_TO_DWORD
INT_TO_LREAL
INT_TO_REAL
INT_TO_SINT
INT_TO_TIME
INT_TO_UDINT
INT_TO_UINT
INT_TO_USINT
INT_TO_WORD
INT_VALUE_TO_BOOL
INTERFACE
INTERNAL_ERROR
INTERNAL_ERROR
INVALID
INVALID_VALUE
ipoClock
ipoClock_2

L

LABEL
LAST_OPERATION_MODE
LE

左
 LEN
 LIMIT
 LINT
 LittleByteArray_to_AnyType
 LITTLEBYTEARRAY_TOANYTYPE
 LN
 LOG
 LREAL
 LREAL_TO_DINT
 LREAL_TO_INT
 LREAL_TO_REAL
 LREAL_TO_SINT
 LREAL_TO_STRING
 LREAL_TO_UDINT
 LREAL_TO_UINT
 LREAL_TO_USINT
 LREAL_VALUE_TO_BOOL
 LREAL_VALUE_TO_BYTE
 LREAL_VALUE_TO_DWORD
 LREAL_VALUE_TO_WORD
 PM
 LWORD
 LWORD_TO_BCD

G

MASTER_SLAVE_ALARMMESSAGES_1
 MAX
 MEMORY_CARD_FULL
 MEMORY_CARD_SERIAL_NUMBER
 MID
 MIN
 MOD
 modeOfDplInterfaceSynchronization
 modeOfOperation
 MUL MULTIME

MUX

N

MS

なし

NO_ACCESS

NO_ALARMMESSAGES

NO_CHANGE

NO_COMMAND_BUFFER_AVAILABLE

NO_DP_SEGMENT

NO_REQUEST_TO_ACTIVATE

NO_RESOURCES_AVAILABLE

NO_RETAIN_GLOBAL

NO_STORAGE_AVAILABLE

NO_VALID_CONFIGURATION_ID

NO_VALID_STATE

NOT

NOT_ACTIVATED

NOT_ENOUGH_RAM

NOT_EXISTENT

NOT_PRESENT

NOTSUPP

numberOfSummarizedTaskOverflow

O

O_K_

OF

OF OR

消灯

[OK]

ON

OR

ORDER_ID

OUT

OUTGOING

OUTGOING_ALARM

OUTPUT

P

PERMANENT-STORAGE
persistentDataPowerMonitoring
PN_1
PREDEFINED_APPLICATION_EVENTS
PREDEFINED_MODULE_INFORMATION
PREVIOUS_ACTIVATED
PROGRAM

R

R_EDGE RESOURCE
R_TRIG
RAM_DISK_FULL
READ_AND_WRITE_JOBS
READ_ERROR
READ_JOBS
REAL
REAL_TO_DINT
REAL_TO_DWORD
REAL_TO_INT
REAL_TO_LREAL
REAL_TO_SINT
REAL_TO_STRING
REAL_TO_TIME
REAL_TO_UDINT
REAL_TO_UINT
REAL_TO_USINT
REAL_VALUE_TO_BOOL
REAL_VALUE_TO_BYTE
REAL_VALUE_TO_DWORD
REAL_VALUE_TO_WORD
REPEAT
REPLACE
REQUEST_ABORT

REQUEST_FALSE
REQUEST_TRUE
RETAIN
RETURN
RIGHT
ROL
ROR
RS
RTC
RUN

S

SAVE_ABORTED
SAVE_FINISHED
SEL
SEMA
SERIAL_NUMBER
servoControlClock
servoTaskCycle
SET_DP_ALARM
SHL
SHR
SIN
SINT
SINT_TO_BCD
SINT_TO_BYTE
SINT_TO_DINT
SINT_TO_DWORD
SINT_TO_INT
SINT_TO_LREAL
SINT_TO_REAL
SINT_TO_UDINT
SINT_TO_UINT
SINT_TO_USINT
SINT_TO_WORD
SINT_VALUE_TO_BOOL

SLAVE_ALARMMESSAGES_1
 SPECIFIC_NUMBER
 SQRT
 SR
 START
 stateOfDpInterfaceSynchronization
 stateOfDpSlaveSynchronization
 STEP
 STOP
 STOP_DEVICE
 STOPU
 STRING
 STRING_TO_DINT
 STRING_TO_LREAL
 STRING_TO_REAL
 STRING_TO_UDINT
 STRUCT
 StructAlarmId
 StructAlarmId_TO_DINT
 StructDeviceConfigurationData
 StructDeviceConfigurationManagement
 StructDeviceCpuData
 StructDeviceCpuDataRW
 StructDeviceDataActivationState
 StructDeviceIm0
 StructDeviceIm0Softwareversion
 StructDeviceIm0Version
 StructDeviceImData
 StructDeviceStartUp
 StructDeviceUpsData
 StructDpStationAddressType
 StructEffectiveTaskRuntimeType
 StructPendingAlarmState
 structPersistentDataPowerMonitoringtype
 StructRetAllocateToken
 StructRetConfiguration

StructRetDeviceCommandState
StructRetDeviceGetStateOfAllDpStations
StructRetDeviceGetStateOfDpSlave
StructRetDeviceNameOfStation
StructRetDiagnosticData
StructRetDpSlaveAddress
StructRetGetConfigurationData
StructRetGetDataByToken
StructRetGetDeviceId
StructRetGetDoIndexNumberFromLogAddress
StructRetGetDpStationAddressFromLogDiagnosticAddress
StructRetGetGeoAddressFromLogAddress
StructRetGetInOutByte
StructRetGetLogDiagnosticAddressFromStationAddress
StructRetGetMemoryCardId
StructRetGetNextLogAddress
StructRetGetPendingAlarmState
StructRetGetSegmentIdentification
StructRetGetStateOfAllDpSlaves
StructRetGetStateOfSingleDpSlave
StructRetGetStateOfTo
StructRetGetStationType
StructRetIPConfig
StructRetReadDriveFaults
StructRetReadDriveMultiParameter
StructRetReadDriveMultiParameterDescription
StructRetReadDriveParameter
StructRetReadDriveParameterDescription
StructRetReadRecord
StructRetTcpOpenClient
StructRetTcpOpenServer
StructRetTcpReceive
StructRetUdpReceive
StructRetUnitDataSetCommand
StructRetWriteDriveMultiParameter
StructRetWriteDriveParameter

StructRetXCommandState
StructRetXreceive
StructStateOfDpStations
StructSystemLoad
StructTaskId
StructTaskOverflowType
StructTaskRuntimeType
StructTaskRuntimeValues
StructTCOFctGenOverride
StructTraceControl
StructTraceState
StructUserData
StructXsendDestAddr
SUB
SUB_DATE_DATE
SUB_DT_DT
SUB_DT_TIME
SUB_TIME
SUB_TOD_TIME
SUB_TOD_TOD
SYMBOL_INFORMATION_NOT_AVAILABLE
SYMBOL_INFORMATION_NOT_FOUND
systemClock
systemLoad

T

TAN
TASK_STATE_INVALID
TASK_STATE_LOCKED
TASK_STATE_RUNNING
TASK_STATE_STOP_PENDING
TASK_STATE_STOPPED
TASK_STATE_SUSPENDED
TASK_STATE_WAIT_NEXT_CYCLE
TASK_STATE_WAIT_NEXT_INTERRUPT
TASK_STATE_WAITING

taskRuntime
taskRuntimeMonitoring
TCOFctGenOverride
TEMPORARY_STORAGE
THEN
TIME
TIME_OF_DAY
TIME_OUT
TIME_TO_INT
TIME_TO_REAL
TIME_TO_UDINT
TO
TOD
TOF
TON
TP
TRACE_ABORTED
TRACE_FINISHED
TRACE_INACTIVE
TRACE_MISMATCH
TRACE_NO_TIME
TRACE_PARAM_VALID
TRACE_RUNNING
TRACE_WAIT_FOR_TRIGGER
traceControl
traceState
TRANSITION
TRUE
TRUNC
TYPE

U

UDINT
UDINT_TO_BYTE
UDINT_TO_DINT
UDINT_TO_DWORD

UDINT_TO_INT
 UDINT_TO_LREAL
 UDINT_TO_REAL
 UDINT_TO_SINT
 UDINT_TO_STRING
 UDINT_TO_TIME
 UDINT_TO_UINT
 UDINT_TO_USINT
 UDINT_TO_WORD
 UDINT_VALUE_TO_BOOL
 UINT
 UINT_TO_BYTE
 UINT_TO_DINT
 UINT_TO_DWORD
 UINT_TO_INT
 UINT_TO_LREAL
 UINT_TO_REAL
 UINT_TO_SINT
 UINT_TO_UDINT
 UINT_TO_USINT
 UINT_TO_WORD
 UINT_VALUE_TO_BOOL
 ULINT
 ユニット
 UNIT_NOT_FOUND
 UNTIL
 USELIB
 USEPACKAGE
 USER
 USER_DEFINED
 USER_EVENTS_1
 USER_EVENTS_2
 USER_STORAGE
 userData
 USES
 USINT

USINT_TO_BYTE
USINT_TO_DINT
USINT_TO_DWORD
USINT_TO_INT
USINT_TO_LREAL
USINT_TO_REAL
USINT_TO_SINT
USINT_TO_UDINT
USINT_TO_UINT
USINT_TO_WORD
USINT_VALUE_TO_BOOL

V

VAR
VAR_ACCESS
VAR_ALIAS
VAR_EXTERNAL
VAR_GLOBAL
VAR_IN_OUT
VAR_INPUT
VAR_OBJECT
VAR_OUTPUT
VAR_TEMP
VERSION_MISMATCH
VOID

W

WAIT_FOR_DP_CLOCK
WAITFORCONDITION
WAITING
WAITING_FOR_RESTART
WARNING
WHEN_COMMAND_DONE
WHILE
WITH
WORD

WORD_TO_BCD
 WORD_TO_BOOL
 WORD_TO_BYTE
 WORD_TO_DINT
 WORD_TO_DWORD
 WORD_TO_INT
 WORD_TO_SINT
 WORD_TO_UDINT
 WORD_TO_UINT
 WORD_TO_USINT
 WORD_VALUE_TO_LREAL
 WORD_VALUE_TO_REAL
 WRITE_JOBS

X

XOR

Y

可能

索引

—

`_additionObjectType`, 72
`_alarm`, 276, 379
`_alarmS`
 用途, 378
`_alarmSc`
 用途, 378, 379
`_alarmScId`
 用途, 378, 379
 説明, 284
`_alarmSId`
 用途, 378
 用途, 378
 説明, 277
`_alarmSq`
 用途, 378
 用途, 378
`_alarmSqlId`
 用途, 378
 用途, 378
 説明, 280
`_AND`, 299
`_BYTE_TO_8BOOL`, 426
`_camTrackType`, 72
`_checkEqualTask`
 例, 108
 説明, 266
`_checkExistingDataSet`
 用途, 386
 説明, 363
`_controllerObjectType`, 72
`_deleteAllUnitDataSets`
 用途, 386
 説明, 365
`_deleteUnitDataSet`
 用途, 386
 説明, 360
`_device`, 345, 350, 357, 360, 364, 366, 387
`_disableScheduler`
 簡単な説明, 253
`_DWORD_FROM_2WORD`, 319
`_DWORD_FROM_4BYTE`, 320
`_DWORD_TO_2WORD`, 428

`_DWORD_TO_4BYTE`, 429
`_exportUnitDataSet`
 用途, 386
 説明, 352
`_finite`
 説明, 323
`_fixedGearType`, 72
`_formulaObjectType`, 72
`_getAlarmId`
 用途, 379
 説明, 285
`_getAverageTaskIdRunTime`, 272
`_getBit`, 294
`_getCommandId`, 368
 エラーのソース, 437
`_getCurrentTaskIdRunTime`, 271
`_getDeviceId`, 372
`_getInOutByte`
 説明, 341
`_getMaximalTaskIdRunTime`, 268
`_getMemoryCardId`, 373
`_getMinimalTaskIdRunTime`, 269
`_getPendingAlarms`, 287
`_getStateOfTaskId`
 簡単な説明, 252
 説明, 254
`_getStateOfUnitDataSetCommand`
 例, 391
 説明, 362
`_GetStateOfXCommand`, 398
`_getSyncCommandId`, 369
 用途, 403
`_getTaskId`
 用途, 253
 用途, 277
 説明, 265
`_importUnitDataSet`
 用途, 386
 説明, 356
`_isNaN`
 説明, 324
`_loadUnitDataSet`
 例, 390
 用途, 386
 説明, 349

- _NOT11, 299
- _OR, 299
- _project, 80
 - 「名前空間」を参照, 80
- _releaseSemaphore
 - 用途, 384
 - 説明, 332
- _resetAlarmId, 287
- _resetAllAlarmId, 287
- _resetTaskId
 - 簡単な説明, 252
 - 説明, 256
- _resetUnitData
 - 用途, 386
- _restartTaskId
 - 簡単な説明, 252
 - 説明, 258
- _resumeTaskId
 - 簡単な説明, 252
 - 説明, 259
- _retriggerTaskControlTimeId
 - 説明, 260
- _retriggerTaskIdControlTime
 - 簡単な説明, 252
- _S7_COUNTER, 431
- _S7_TIMER, 432
- _saveUnitDataSet
 - 例, 391
 - 用途, 386
 - 説明, 344
- _SC_ALARM_CONFIGURATION, 105
- _SC_ARRAY_BOUND_ERROR_READ, 101
- _SC_ARRAY_BOUND_ERROR_WRITE, 101
- _SC_BACKGROUND_TIMER_OVERFLOW, 99
- _SC_BATTERY_WARNING_LEVEL1, 103
- _SC_BATTERY_WARNING_LEVEL2, 103
- _SC_CYCLE_TIMER_OVERFLOW, 98
- _SC_DEVICE_COMMAND, 105
- _SC_DIAGNOSTIC_INTERRUPT, 102
- _SC_DIVISION_BY_ZERO, 101
- _SC_DP_CLOCK_DETECTED, 103
- _SC_DP_SLAVE_NOT_SYNCHRONIZED, 103
- _SC_DP_SLAVE_SYNCHRONIZED, 103
- _SC_DP_SYNCHRONIZATION_LOST, 103
- _SC_EXCEPTION, 105
- _SC_EXTERNAL_COMMAND, 105
- _SC_IMAGE_UPDATE_FAILED, 102
- _SC_IMAGE_UPDATE_OK, 103
- _SC_INVALID_ADDRESS, 104
- _SC_INVALID_FLOATING_POINT_OPERATION, 101
- _SC_IO_MODULE_NOT_SYNCHRONIZED, 103
- _SC_IO_MODULE_SYNCHRONIZED, 103
- _SC_MODE_SELECTOR, 105
- _SC_PC_INTERNAL_FAILURE, 102
- _SC_PULL_PLUG_INTERRUPT, 103
- _SC_STATION_DISCONNECTED, 102
- _SC_STATION_RECONNECTED, 102
- _SC_TO_INSTANCE_NOT_EXISTENT, 101
- _SC_VARIABLE_ACCESS_ERROR_READ, 101
- _SC_VARIABLE_ACCESS_ERROR_WRITE, 101
- _sensorType, 72
- _setBit, 295
- _setDeviceErrorLED, 374
- _setSafeValue
 - 説明, 337
- _sizeOf, 375
- _startSyncCommand
 - 用途, 403
- _startTaskId
 - 簡単な説明, 252
 - 説明, 262
- _suspendTaskId
 - 簡単な説明, 252
 - 説明, 263
- _task, 253
- _testAndSetSemaphore
 - 用途, 384
 - 説明, 331
- _to, 80
 - 「名前空間」を参照, 80
- _toggleBit, 297
- _waitTime, 436
 - 用途, 242
 - 説明, 370
- _WORD_FROM_2BYTE, 319
- _WORD_TO_2BYTE, 427
- _writeAndSendMessage
 - デバイスファンクション, 378
- _XOR, 299
- _Xreceive
 - パラメータ記述, 397
 - 用途, 395
- _Xsend
 - パラメータ記述, 396
 - 伝送先アドレスの構造, 396
 - 用途, 395

1

1回だけのプログラムデータインスタンス化, 233

A

ABS, 290
ACOS, 291
AlarmS
 バッファマネージメント, 380
ALARMS_ERROR, 278, 282, 284
ALARMS_ERROR OR
DSC_SVS_DEVICE_ALARMS_IV_CALL, 279, 282
ALARMS_ERROR OR
DSC_SVS_DEVICE_ALARMS_IV_FIRST_CALL, 279, 282
ALARMS_ERROR OR
DSC_SVS_DEVICE_ALARMS_IV_SFC_TYP, 279, 283
ALARMS_ERROR OR
DSC_SVS_DEVICE_ALARMS_NO_ENTRY, 280, 283
ALARMS_STATE, 284
ALARMS_STATE OR ALARMS_QSTATE, 285
ANYOBJECT, 72, 322
AnyObject_to_Object
 説明, 322
AnyType_to_BigByteArray
 説明, 313
AnyType_to_LittleByteArray
 説明, 313
ASIN, 291
ATAN, 291

B

BackgroundTask, 130, 144
 Taskstartinfo, 98
BEGIN_SYNC
 用途, 403
BigByteArray_to_AnyType
 説明, 315, 318
BOOL_TO_BYTE, 304
BOOL_TO_DWORD, 304
BOOL_TO_WORD, 305
BOOL_VALUE_TO_DINT, 305
BOOL_VALUE_TO_INT, 305
BOOL_VALUE_TO_REAL, 305
BOOL_VALUE_TO_SINT, 305
BOOL_VALUE_TO_UDINT, 305
BOOL_VALUE_TO_UINT, 305
BOOL_VALUE_TO_USINT, 305
BYTE_TO_BOOL, 305
BYTE_TO_DINT, 305
BYTE_TO_DWORD, 305
BYTE_TO_INT, 305
BYTE_TO_SINT, 305
BYTE_TO_UDINT, 305

BYTE_TO_UINT, 305
BYTE_TO_USINT, 305
BYTE_TO_WORD, 305
BYTE_VALUE_TO_LREAL, 305
BYTE_VALUE_TO_REAL, 305

C

Cam, 72
camType, 72
CommandId, 33
 エラーのソース, 437
CONCAT_DATE_TOD, 309
ControlPanelTask, 130
COS, 291
CPU STOP, 93
CTD, 418
CTD_DINT, 419
CTD_UDINT, 419
CTU, 416
CTU_DINT, 417
CTU_UDINT, 418
CTUD, 419
CTUD_DINT, 420
CTUD_UDINT, 421

D

DATE_AND_TIME_TO_DATE, 309
DATE_AND_TIME_TO_TIME_OF_DAY, 309
DCC, 26
 DCCAux_2タスク, 213
 DCCAuxタスク, 213
 T1(DCC)タスク, 210
 T2(DCC)タスク, 211
 T3(DCC)タスク, 212
 ブロック, 208
DINT#MAX, 449
DINT#MIN, 449
DINT_TO_BYTE, 305
DINT_TO_DWORD, 305
DINT_TO_INT, 305
DINT_TO_LREAL, 305
DINT_TO_REAL, 305
DINT_TO_SINT, 305
DINT_TO_STRING, 311
DINT_TO_UDINT, 305
DINT_TO_UINT, 305
DINT_TO_USINT, 305
DINT_TO_WORD, 305
DINT_VALUE_TO_BOOL, 305

DPサイクルクロック, 175

driveAxis, 72

DSC_SVS_DEVICE_ALARMS_EVENT_ID_IN_USE, 279, 282

DSC_SVS_DEVICE_ALARMS_EVENT_ID_NOT_USED, 279, 282

DSC_SVS_DEVICE_ALARMS_ILLEGAL_EVENT_ID, 278, 282, 284

DSC_SVS_DEVICE_ALARMS_INTERNAL_ERROR, 279, 283

DSC_SVS_DEVICE_ALARMS_LAST_ENTRY_USED, 279, 282

DSC_SVS_DEVICE_ALARMS_LAST_SIGNAL_USED, 279, 282

DT_TO_DATE, 309

DT_TO_TOD, 309

DWORD_TO_BOOL, 305

DWORD_TO_BYTE, 305

DWORD_TO_DINT, 305

DWORD_TO_INT, 305

DWORD_TO_REAL, 306

DWORD_TO_SINT, 306

DWORD_TO_UDINT, 306

DWORD_TO_UINT, 306

DWORD_TO_USINT, 306

DWORD_TO_WORD, 306

DWORD_VALUE_TO_LREAL, 306

DWORD_VALUE_TO_REAL, 306

E

effectiveTaskruntime, 184

END_SYNC

用途, 403

ENUM_TO_DINT, 309

ExecutionFaultTask, 95, 159, 228

Taskstartinfo, 100

EXP, 291

EXPD, 291

EXPRESSION

説明(コンテキストに応じた), 237

EXPT, 291

externalEncoderType, 72

F

F_TRIG, 415

FBD, 26

followingAxis, 72

followingObjectType, 72

H

HMI (マンマシンインターフェース), 405

HW Config, 22

I

I/O処理

等時性, 197

InputSynchronousTask_1:, 151, 228

Taskstartinfo, 98

InputSynchronousTask_2, 151, 228

Taskstartinfo, 98

INT#MAX, 448

INT#MIN, 448

INT_TO_BYTE, 306

INT_TO_DINT, 306

INT_TO_DWORD, 306

INT_TO_LREAL, 306

INT_TO_REAL, 306

INT_TO_SINT, 306

INT_TO_TIME, 309

INT_TO_UDINT, 306

INT_TO_UINT, 306

INT_TO_USINT, 306

INT_TO_WORD, 306

INT_VALUE_TO_BOOL, 306

IPO_2サイクルクロック, 175

IPOSynchronousTask, 228

Taskstartinfo, 98

IPOSynchronousTask, 153

IPOSynchronousTask_2, 228

IPOSynchronousTask_2, 153

Taskstartinfo, 98

IPOTask, 153

IPOTask_2, 153

IPOサイクルクロック, 175

L

LIMIT, 329

LittleByteArray_to_AnyType

説明, 315

LN, 291

LOG, 291

LREAL_TO_DINT, 306

LREAL_TO_INT, 306

LREAL_TO_REAL, 306

LREAL_TO_SINT, 306

LREAL_TO_UDINT, 306

LREAL_TO_UINT, 306

LREAL_TO_USINT, 306

LREAL_VALUE_TO_BOOL, 306
LREAL_VALUE_TO_BYTE, 306
LREAL_VALUE_TO_DWORD, 306
LREAL_VALUE_TO_WORD, 306

M

MAX, 328
measuringInputType, 72
MIN, 328
Motion Control, 127
MotionTask(モーションタスク), 140
MotionTasks, 130
 Taskstartinfo, 98
MUX, 327

O

OFF遅延(システムFB), 424
ON遅延(システムFB), 423
OperationLevel, 132
outputCamType, 72

P

PeripheralFaultTask, 159, 228
 Taskstartinfo, 102
PLCopen, 26
PLC機能, 15
PNサイクルクロック, 175
posAxis, 72
PostControlTask_1, 151, 228
 Taskstartinfo, 98
PostControlTask_2, 151, 228
 Taskstartinfo, 98
ProTool, 405
PWMSynchronousTask, 151, 228
 Taskstartinfo, 98

R

R_TRIG, 414
REAL_TO_DINT, 306
REAL_TO_DWORD, 306
REAL_TO_INT, 306
REAL_TO_LREAL, 306
REAL_TO_SINT, 306
REAL_TO_STRING, 311
REAL_TO_TIME, 309
REAL_TO_UDINT, 307

REAL_TO_UINT, 307
REAL_TO_USINT, 307
REAL_VALUE_TO_BOOL, 307
REAL_VALUE_TO_BYTE, 307
REAL_VALUE_TO_DWORD, 307
REAL_VALUE_TO_WORD, 307
ROL, 292
ROR, 292
RTC, 424
RUN
 モードにならない, 439
 変数初期化に対する影響, 230
RUNがSTOP
 エラー原因, 439
 変数初期化に対する影響, 230
RUN時のダウンロード, 244

S

SEL, 326
ServoSynchronousTask, 152, 228
 Taskstartinfo, 98
SHL, 292
SHR, 292
ShutdownTask, 130, 167
 Taskstartinfo, 104
SIMATIC S7デバイス
 SIMOTIONデバイスとの通信, 398
 伝送先アドレス、構造, 396
SIMOTION
 システムアーキテクチャ, 21
 テクノロジーパッケージ, 30
 プロジェクト, 23
 モーションコントロール, 15
 ランタイムシステム, 129
 ランタイムモデル, 136
 実行システム, 127
 用途, 16
SIMOTION C2xx, 19
SIMOTION D4xx, 20
SIMOTION P350, 19
SIMOTION SCOUTエンジニアリングシステム, 22
SIMOTIONデバイスのSIMATIC
S7デバイスとの通信, 398
SIN, 291
SINT#MAX, 448
SINT#MIN, 448
SINT_TO_BYTE, 307
SINT_TO_DINT, 307
SINT_TO_DWORD, 307
SINT_TO_INT, 307
SQRT, 290

SR, 412, 413
StartupTask, 130, 138
 Taskstartinfo, 98
STOPモード, 93
STRING_TO_DINT, 311
STRING_TO_REAL, 312
STRING_TO_UDINT, 311
STRUCTALARMID#NIL, 449
STRUCTTASKID#NIL, 449
SynchronousTasks, 130, 151
 Taskstartinfo, 98
SystemInterruptTasks, 130, 157

T

T#MAX, 449
T#MIN, 449
TAN, 291
TASK_STATE_INVALID, 235, 255
TASK_STATE_LOCKED, 236, 255
TASK_STATE_RUNNING, 236, 255
TASK_STATE_STOP_PENDING, 235, 255
TASK_STATE_STOPPED, 235, 255
TASK_STATE_SUSPENDED, 236, 255
TASK_STATE_WAIT_NEXT_CYCLE, 236, 255
TASK_STATE_WAIT_NEXT_INTERRUPT, 236, 255
TASK_STATE_WAITING, 236, 255
Taskruntime, 184
TaskStartInfo(TSI), 185
TControl
 テクノロジーパッケージ, 30
Tdp, 200
Tdx, 200
TechnologicalFaultTask, 159, 228
 Taskstartinfo, 99
temperatureControllerType, 72
Ti, 200
Time allocation
 ラウンドロビン実行レベルでの, 187
 設定, 189
TIME#MAX, 449
TIME#MIN, 449
TIME_OF_DAY#MAX, 449
TIME_OF_DAY#MIN, 449
TIME_TO_INT, 309
TIME_TO_REAL, 309
TIME_TO_UDINT, 309
TimeFaultBackgroundTask, 159, 228
 Taskstartinfo, 99
TimeFaultTask, 159, 228
 Taskstartinfo, 98
TimerInterruptTask(タイマー割り込みタスク), 147

TimerInterruptTasks, 130
 Taskstartinfo, 98
To, 200
TO, 28
TO#NIL, 72, 84, 449
TOD#MAX, 449
TOD#MIN, 449
TOF, 424
TON, 423
TP Cam
 テクノロジーパッケージ, 30
TP Cam_ext
 テクノロジーパッケージ, 30
TP Path
 パスオブジェクト, 30
TRUNC, 290
TSI, 185
TSI#alarmNumber, 99
TSI#alarmP1_DINT, 99
TSI#alarmP1_LREAL, 99
TSI#alarmP1_UDINT, 99
TSI#alarmP2_DINT, 99
TSI#alarmP2_LREAL, 99
TSI#alarmP2_UDINT, 99
TSI#alarmP3_DINT, 99
TSI#alarmP3_LREAL, 99
TSI#alarmP3_UDINT, 99
TSI#alarmP4_DINT, 99
TSI#alarmP4_LREAL, 99
TSI#alarmP4_UDINT, 99
TSI#alarmP5_DINT, 100
TSI#alarmP5_LREAL, 100
TSI#alarmP5_UDINT, 100
TSI#commandId.high, 99
TSI#commandId.low, 99
TSI#currentTaskId, 98, 99, 100, 102, 104, 105
TSI#cycleTime, 98, 99, 100, 102, 104, 105
TSI#details, 104, 105
TSI#eventClass, 104
TSI#executionFaultType, 101
TSI#eventClass, 106
TSI#faultId, 104
TSI#interruptId, 98, 99, 102
TSI#InterruptId, 105, 106
TSI#logBaseAdrIn, 104
TSI#logBaseAdrOut, 104
TSI#logDiagAdr, 104
TSI#shutDownInitiator, 105
TSI#startTime, 98, 99, 100, 102, 104, 105
TSI#taskId, 98, 101
TSI#tolnst, 99

U

UDINT#MAX, 449
 UDINT#MIN, 449
 UDINT_TO_STRING, 311
 UDINT_TO_TIME, 309
 UINT#MAX, 449
 UINT#MIN, 449
 USEPACKAGE, 64
 UserInterruptTasks, 130, 162
 Taskstartinfo, 104
 USINT#MAX, 449
 USINT#MIN, 449

W

WAITFORCONDITION, 135
 説明(コンテキストに応じた), 237

ア

アイソクロナスI/O処理, 197
 アイソクロナスデータ処理, 199
 アップ/ダウンカウンタ(システムFB), 419
 アップカウンタ(システムFB), 416
 アラーム, 93
 コンフィグレーション, 93
 Taskstartinfoのクエリー;タスク: Startinfo;
 Taskstartinfo; TSI, 108
 アラームID, 276

イ

イベント駆動タスク, 130
 インスタンス化, 29, 31
 インターフェースの取り決め, 395

ウ

ウォッチドッグ, 185

エ

エキスパートリスト, 43
 エキスパートリストの使用, 46
 エッジ検出
 システムFB, 414
 メッセージプログラミングにおいて, 378
 エラー
 CommandIdの消失, 437

CPUがRUNにならない, 439
 REAL変数の比較, 440
 コマンドの戻り値, 65
 サイクル内のTOファンクション, 436
 システムデータへのアクセス時, 124
 データタイプの変換, 435
 ランタイム, 93
 レンジ違反, 441
 周期的タスク, 436
 追跡, 438
 エラーの有効化, 111
 エンコーダ
 外部, 72

オ

オフラインモード, 25
 オンラインモード, 25

カ

カム, 39
 カムトラック(camTrackType), 39
 関連資料, 4

グ

グローバル制御テレグラム(GC), 197
 グローバル応答
 プロセスアラーム, 111

ク

クロック同期化
 端子 - 端子, 205

コ

コマンド
 CommandId, 33
 ステップ有効化条件, 75
 同期/非同期, 33
 戻り値, 33, 65, 122
 コマンド参照 - commandId, 78
 コマンド実行の診断, 79
 コントローラオブジェクト, 40
 コンフィグレーション, 32
 実行システム, 170
 コンフィグレーションデータ, 32

サ

サーボサイクルクロック, 175
サイクルクロックソース
 選択, 174
サイクルタイム
 モニタリング, 185

シ

シーケンシャルプログラムの実行
 タスクのステータス, 235
 影響, 252
 確認, 229
シーケンス
 ラウンドロビン実行レベルでの, 188
システムアーキテクチャ, 21
システムクロック, 127
 定義, 175
システムサイクルクロック
 エラー原因, 439
システムタスク, 127
システムデータ
 アクセス時のエラー, 124
システムファンクション
 コマンド / 実行ステータスのクエリー(システムファンクション) 戻り値, 79
 定義, 63
システムファンクションブロック
 定義, 412
 概要, 409
システム変数, 32
 アクセスの最適化, 442
 エラー原因, 440
 概要; 変数: システム, 83
 定義, 63

ス

スタックのサイズ, 174
ステータス
 タスク(ステータス値), 235

セ

セマフォコマンド
 用途), 383
 説明, 331
センサ, 40

タ

タイプ変換ファンクション, 304
タイムアウト, 183, 436
 モニタリング, 184

ダ

ダウンカウンタ(システムFB), 418

タ

タスク, 127, 128
 BackgroundTask, 144, 228
 ExecutionFaultTask, 159
 IPOSynchronousTask, 153
 MotionTask(モーションタスク), 140, 228
 PeripheralFaultTask, 159
 ServoSynchronousTask, 152
 ShutdownTask, 167, 229
 StartupTask, 138, 228
 SynchronousTask, 228
 SynchronousTasks, 151
 SystemInterruptTask, 228
 SystemInterruptTasks, 157
 TaskStartInfo(TSI), 185
 TechnologicalFaultTask, 159
 TimeFaultBackgroundTask, 159
 TimeFaultTask, 159
 TimerInterruptTask(タイマー割り込みタスク), 147, 228
 UserInterruptTask, 228
 UserInterruptTasks, 162
 コンセプト, 227
 プログラミング, 252
 プログラムの割り当て, 170, 229
 ランタイム測定(ファンクション), 267
 ローカル変数の初期化, 230
 優先度, 133
 初期値の割り当て, 231
 制御コマンド(簡単な説明), 252
 状態, 235
 起動シーケンス, 135
タスクの実行時間, 181
タスク優先度, 133
タスク制御, 174

ツ

ツールヒント, 32

デ

データタイプ
 エラーのソース, 435
 テクノロジーオブジェクト, 71
 列挙子, 69
 変換, 304
 データの送信, 395
 データ処理
 等時性, 199

テ

テクノロジーオブジェクト, 28, 31
 インスタンス, 65
 インスタンス化, 29, 31
 クエリーの妥当性, 71
 コンフィグレーション, 32
 コンフィグレーションデータ, 87
 システム変数, 32
 データタイプ, 71
 パッケージ(定義), 63
 ファンクション(コード), 65
 ファンクション(入力パラメータ), 68
 ファンクション(定義), 63
 リセット, 90
 初期化, 71
 名前, 65
 定義, 63
 相互接続, 52
 テクノロジーオブジェクトタイプ, 29, 31
 テクノロジカルアラーム, 35, 110
 ユーザープログラムでの評価, 120
 設定, 113
 テクノロジパッケージ, 30
 ライブラリ内, 92
 定義, 63
 テクノロジ相互接続画面フォーム, 37

ド

ドライブ軸, 72

ト

トルク制限B+ / B-, 60

ハ

ハードウェアプラットフォーム, 19

パ

パスオブジェクト, 39

バ

バスサイクルクロック, 175
 バッファマネージメント
 AlarmSメッセージ, 380

パ

パラメータ
 ファンクションブロックの効率的なアクセス, 443
 パルス(システムFB), 422
 パルス幅変調サイクルクロック, 175

ビ

ビット列標準ファンクション, 292

フ

ファンクション
 セマフォ(説明), 331
 セマフォ(適用), 383
 呼出し中のエラーソース, 436
 標準ファンクション, 289
 通信, 395
 ファンクションチャート, 26
 ファンクションブロック(Function block)
 システムファンクションブロック(System function block), 409
 効率的なパラメータアクセス, 443
 フリーランニングタスク, 130
 フリップフロップのセット, 412

プ

プログラミング, 25
 エラーのソース, 435
 コマンド実行, 73
 効率の向上, 442
 プログラム, 127
 タスクの割り当て, 170, 229
 効率の向上, 442
 特定エラー, 438
 プログラムデータインスタンス化
 1回だけの, 244
 プラグマ, 245

プロジェクト, 23
プロセスアラーム
 グローバル応答, 111
 ローカル応答, 110

マ

マーシャリング, 313
マスクされた故障番号, 382

メ

メカトロニクス, 15
メッセージ
 アラーム, 93
 プログラミング, 276, 378

モ

モーションコントロール, 15, 127
モーションコントロールチャート, 25
モジュール方式の機械設計, 15, 26
モニタリング
 サイクルタイム, 185
 タイムアウトとレベルオーバーフロー, 184

ユ

ユーザープログラム, 25
ユーザープログラムタスク, 128
ユーザー割り込み
 プログラム可能, 237

ラ

ライブラリ
 テクノロジーパッケージ, 92
ラウンドロビン, 130, 187
 時間割り当ての設定, 189
ラダーダイアグラム, 26
ランタイムグループ, 208
ランタイムシステム, 129
ランタイムモデル, 136

レ

レベルオーバーフロー, 183
 モニタリング, 184
レンジ違反, 441

ロ

ローカル応答
 プロセスアラーム, 110

一

一般標準ファンクション, 290

三

三角関数標準ファンクション, 291

不

不良
 サイクルの待機時間, 436
 周期的タスク, 436

予

予約済みの識別子, 449

代

代替値戦略, 124

位

位置決め制御サイクルクロック, 175
位置決め軸, 72

例

例、完全な
 TOのデータタイプの使用方法, 72
 WaitForCondition, 240
 式, 240

信

信号立ち上がり
 システムFB, 414

値

値パラメータ、参照パラメータ、パラメータ:
 値パラメータ、パラメータ:参照パラメータ, 70

優

優先度, 134

入

入力パラメータ
システムファンクション, 68

出

出力カム, 39, 72

列

列挙子, 69
列挙子データタイプ, 69

初

初期化
テクノロジーオブジェクト, 71

割

割り込み
プログラム可能, 237

効

効率的なプログラミング, 442

双

双安定ファンクションブロック, 412

同

同期コマンド実行, 33
データ伝送用, 395
同期タスク, 130
同期起動, 403

周

周期的タスク, 130
周期的プログラム実行
タイムアウト, 436

タスクのステータス, 235
影響, 252
確認, 229

呼

呼び出しのブロック, 436

固

固定ギア, 40
固定バスサイクルタイム, 174

基

基本サイクルクロック, 175

変

変数
TOのインスタンス宣言, 65
セマフォ(適用), 383
レンジ違反, 441
効率的なアクセス, 442

外

外部エンコーダ, 39, 72

実

実トルク, 60
実行エラー, 94
実行システム, 127
設定, 170
実行レベル, 127, 128, 227
ラウンドロビン, 228
割り込み, 228
時間を限定して作成, 228

対

対数標準ファンクション, 290

従

従動オブジェクト, 39, 72
従動軸, 72

戻

戻り値, 33, 122

拡

拡張言語要素, 232

指

指数, 291

故

故障番号のフィルタ, 382

数

数値標準ファンクション, 290

数式オブジェクト, 40

時

時刻タイプ

変換, 309

時間トリガタスク, 130

暗

暗黙的な相互接続, 36

条

条件の待機, 135

構

構造化テキスト, 25

標

標準ファンクション, 289, 290

汎

汎用相互接続画面フォーム, 37

浮

浮動小数点数

エラー原因, 440

温

温度コントローラ, 72

温度チャンネル, 40

温度制御, 40, 128

測

測定入力, 39, 72

用

用途, 16

相

相互接続

テクノロジーオブジェクトの, 52

テクノロジー相互接続画面フォームによる, 37

暗黙的, 36

汎用相互接続画面フォームによる, 37

端

端子 - 端子アイソクロナスモード, 205

端子 - 端子応答時間, 151

端子 - 軸 - 応答時間, 151

統

統合オートメーション, 18

補

補間サイクルクロック, 175

補間サイクルクロック2, 175

識

識別子

基本システムで予約済, 449

起

起動シーケンス
タスク, 135

軸

軸, 39
ドライブ軸, 72
位置決め軸, 72
従動軸, 72

追

追加オブジェクト, 40
追加トルク, 60

逐

逐次的タスク, 130

通

通信コマンド, 395

運

運動
インターフェースタイプ, 58
運動成分, 59

適

適合性コマンド
用途, 383
説明, 331

障

障害, 93

非

非同期コマンド実行, 33
データ伝送用, 395