SIEMENS

SIMOTION
Interpreter

Applikationshandbuch

06/2013

Beitrags-ID: 61182043

Rechtliche Hinweise

Warnhinweiskonzept

Dieses Handbuch enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungsstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

⚠GEFAHR
bedeutet, dass Tod oder schwere Körperverletzung eintreten **wird**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠WARNUNG
bedeutet, dass Tod oder schwere Körperverletzung eintreten **kann**, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

⚠VORSICHT
bedeutet, dass eine leichte Körperverletzung eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

ACHTUNG
bedeutet, dass Sachschaden eintreten kann, wenn die entsprechenden Vorsichtsmaßnahmen nicht getroffen werden.

Beim Auftreten mehrerer Gefährdungsstufen wird immer der Warnhinweis zur jeweils höchsten Stufe verwendet. Wenn in einem Warnhinweis mit dem Warndreieck vor Personenschäden gewarnt wird, dann kann im selben Warnhinweis zusätzlich eine Warnung vor Sachschäden angefügt sein.

Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung **qualifiziertem Personal** gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

Bestimmungsgemäßer Gebrauch von Siemens-Produkten

Beachten Sie Folgendes:

⚠WARNUNG
Siemens-Produkte dürfen nur für die im Katalog und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und -komponenten zum Einsatz kommen, müssen diese von Siemens empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Marken

Alle mit dem Schutzrechtsvermerk ® gekennzeichneten Bezeichnungen sind eingetragene Marken der Siemens AG. Die übrigen Bezeichnungen in dieser Schrift können Marken sein, deren Benutzung durch Dritte für deren Zwecke die Rechte der Inhaber verletzen kann.

Haftungsausschluss

Wir haben den Inhalt der Druckschrift auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Dennoch können Abweichungen nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Druckschrift werden regelmäßig überprüft, notwendige Korrekturen sind in den nachfolgenden Auflagen enthalten.

Siemens AG
Industry Sector
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

Beitrags-ID: 61182043
℗ 06/2013 Änderungen vorbehalten

Inhaltsverzeichnis

Vorwort

1

1.1 Allgemeine Hinweise

Hinweis

Die Standardapplikationen sind unverbindlich und erheben keinen Anspruch auf Vollständigkeit hinsichtlich Konfiguration und Ausstattung sowie jeglicher Eventualitäten. Die Standardapplikationen stellen keine kundenspezifischen Lösungen dar, sondern sollen lediglich Hilfestellung bieten bei typischen Aufgabenstellungen. Sie sind für den sachgemäßen Betrieb der beschriebenen Produkte selbst verantwortlich. Diese Standardapplikationen entheben Sie nicht der Verpflichtung zu sicherem Umgang bei Anwendung, Installation, Betrieb und Wartung. Durch Nutzung dieser Standardapplikationen erkennen Sie an, dass wir über die beschriebene Haftungsregelung hinaus nicht für etwaige Schäden haftbar gemacht werden können. Wir behalten uns das Recht vor, Änderungen an diesen Standardapplikationen jederzeit ohne Ankündigung durchzuführen. Bei Abweichungen zwischen den Vorschlägen in diesen Standardapplikationen und anderen Siemens Publikationen, wie z. B. Katalogen, hat der Inhalt der anderen Dokumentation Vorrang.

Gewährleistung, Haftung und Support

Im Falle kostenloser Überlassung der Applikation gilt folgendes:

Für die in diesem Dokument enthaltenen Informationen übernehmen wir keine Gewähr.

Andere und alle sonstigen Rechte und Ansprüche gegen die Siemens AG, gleich aus welchem Rechtsgrund, sind ausgeschlossen. Insbesondere Ansprüche gegen die Siemens AG auf Schadenersatz, insbesondere wegen Produktionsausfalls, Nutzungsausfalls, entgangenen Gewinns, direkter, indirekter oder Folgeschäden sind ausgeschlossen.

Dies gilt nicht, soweit zwingend gehaftet wird, z. B. nach dem Produkthaftungsgesetz, in Fällen des Vorsatzes, grober Fahrlässigkeit von Vorgesetzen und leitenden Angestellten der Siemens AG oder in Fällen des arglistigen Verschweigens von Mängeln.

Diese Haftungsbegrenzung findet auch Anwendung zu Gunsten von Subunternehmern, Zulieferern, Beauftragten, Vorgesetzten, leitenden Angestellten und Angestellten der Siemens AG.

Auf diese Vereinbarung findet für Kunden mit Firmensitz in Deutschland das deutsche Recht Anwendung für Kunden mit Firmensitz außerhalb Deutschlands findet das schweizerische Recht Anwendung. Die Anwendbarkeit des Übereinkommens der Vereinten Nationen über Verträge über den internationalen Warenkauf vom 11.04.1980 (CISG) ist ausgeschlossen.

Im Falle entgeltlicher Überlassung der Applikation gilt die für das jeweilige Geschäft zutreffende Alternative:

- Alternative 1: (Internes Geschäft)

Es gelten, sofern nicht unten etwas Abweichendes geregelt ist, die "Bedingungen für Lieferungen und Leistungen im Siemens-internen Geschäft" in der jeweils zum Zeitpunkt der Überlassung gültigen Fassung.

- Alternative 2: (Inlandsgeschäft der Siemens AG)

Es gelten, sofern nicht unten etwas Abweichendes geregelt ist, die "Allgemeine Bedingungen zur Überlassung von Software für Automatisierungs- und Antriebstechnik an Lizenznehmer mit Sitz in Deutschland" in der jeweils zum Zeitpunkt der Überlassung gültigen Fassung.

- Alternative 3: (Direktexportgeschäft der Siemens AG)

Es gelten, sofern nicht unten etwas Abweichendes geregelt ist, die "Allgemeine Bedingungen zur Überlassung von Softwareprodukten für Automation and Drives an Lizenznehmer mit Sitz außerhalb Deutschlands" in der jeweils zum Zeitpunkt der Überlassung gültigen Fassung.

Die Weitergabe oder Vervielfältigung dieser Applikationsbeispiele oder Auszüge daraus in unbearbeiteter Form sind nicht gestattet, soweit nicht ausdrücklich von Siemens Industry Sector gestattet.

Hinweis auf Exportkennzeichen

AL: N

ECCN: N

1.2 Über dieses Dokument

Ziel

Dieses Dokument soll bei der einfachen Erstellung von Motion Control Abläufen einer Maschine durch den **Interpreter** helfen. Kenntnisse beim Umgang mit dem Engineeringsystem SIMOTION SCOUT werden vorausgesetzt.

Hinweis

Diese Dokumentation erhebt nicht den Anspruch, alle Gerätedetails oder Gerätevarianten zu erfassen oder jeden denkbaren Fall des Betriebs oder der Anwendung zu berücksichtigen.

Sollten Sie weitere Informationen benötigen oder sollten spezielle Probleme auftreten, die für das Anwendungsgebiet nicht ausführlich genug behandelt werden, wenden Sie sich bitte an die örtliche Siemens-Niederlassung.

Zielgruppe

Das vorliegende Dokument wendet sich an Programmierer und Inbetriebnehmer.

Siemens Industry Online Support

Dieser Beitrag stammt aus dem Siemens Industry Online Support. Durch den folgenden Link gelangen Sie direkt zur Downloadseite dieses Dokuments:

http://support.automation.siemens.com/WW/view/de/61182043

Applikationsbeschreibung 2

2.1 Anwendungsgebiet

2.1.1 Beschreibung

Applikation Interpreter

Die Applikation **Interpreter** dient der einfachen Erstellung von Maschinenabläufen. Eine einfache Möglichkeit zur Erstellung dieser Motion Control Abläufe ist die Verwendung von Kommandos und deren Auswertung durch die Applikation **Interpreter**. Der **Interpreter** setzt die Kommandos durch die Ansteuerung von SIMOTION-Systembefehlen um. Mehrere Kommandos werden in Verfahrsatz-Tabellen bzw. Verfahrsatz-Programmen zusammengefasst, die sequenziell abgearbeitet werden.

Dieses Konzept wird z. B. beim Einfachpositionierer bei SINAMICS-Antrieben oder beim TopLoading-Paket für Handlinggeräte verwendet.

Die Applikation **Interpreter** stellt die Kommandos zur Ansteuerung von Technologieobjekten sowie Kommandos für Kontrollstrukturen wie Sprünge, Schleifen sowie Weiterschaltbedingungen zur Verfügung. Weiterhin sind einfache mathematische Funktionen realisiert.

Somit ist es möglich, ohne nähere Kenntnis der Systembefehle, den Bewegungsablauf einer Maschine zu parametrieren, statt diesen zu programmieren. Dadurch können auch ohne Erfahrung in der Programmierung Maschinenabläufe erstellt werden. Die Parametrierung der Maschinenabläufe kann sowohl zum Zeitpunkt der Projektierung als auch während der laufenden Maschine, z. B. über ein HMI erstellt werden. So kann flexibel auf unterschiedliche Vorgaben reagiert werden.

Wichtige Informationen zur Technologie und zu den einzelnen Technologieobjekten (TO) von SIMOTION SCOUT finden Sie in folgenden Dokumenten der SIMOTION-Standarddokumentation:

- *Funktionshandbuch Motion Control TO Achse elektrisch/hydraulisch, Externer Geber*
- *Funktionshandbuch SIMOTION SCOUT Basisfunktionen*

2.1.2 Einsatzbereich

Die Applikation **Interpreter** mit der Bibliothek **LMCIpr** kann universell für beliebige Anwendungen mit Motion Control Abläufen eingesetzt werden.

2.1.3 Voraussetzung

Die Applikation **Interpreter** setzt die Verwendung der Bibliothek **LMCBasic** ab V1.2.1 voraus.

2.2 Zielsetzung

2.2.1 Aufgabenstellung

Die Applikation **Interpreter** soll Grundfunktionalitäten für die Realisierung von Bewegungsabläufen zur Verfügung stellen und kann somit als Grundlage zur Umsetzung jeder beliebigen Motion Control Applikation verwendet werden.

Der **Interpreter** führt grundlegende Motion Control und andere Funktionalitäten aus. Zu diesen grundlegenden Funktionalitäten gehören:

- Kommandos für die Ansteuerung des **Interpreters**
  - Starte Programm
  - Stoppe Programm
  - Abarbeitung des nächsten Schritts
- Kommandos für Motion Control
  - Achsfreigabe setzen
  - Achsfreigabe zurücknehmen
  - Absolutes Positionieren
  - Relatives Positionieren
  - Überlagertes Positionieren
  - Drehzahlvorgabe
  - Stopp der Achse
  - Not-Halt der Achse
  - Referenzieren
  - Rücksetzen von Achsen
  - Aktivierung des Getriebegleichlaufs
  - Deaktivierung des Getriebegleichlaufs
  - Aktivierung des Kurvenscheibengleichlaufs mit Versatz der Leitachse
  - Aktivierung des Kurvenscheibengleichlaufs mit Versatz der Folgeachse
  - Deaktivierung des Kurvenscheibengleichlaufs
  - Einstellung einer Phasenverschiebung für einen aktiven Gleichlauf
  - Setzen des Override für Beschleunigung und Verzögerung
  - Aktivierung eines Nocken
  - Deaktivierung eines Nocken

- Kommandos für Weiterschaltbedingungen
  - Warten einer Zeitdauer
  - Warten auf boolesche Bedingung
  - Warten auf Achsstatus
  - Warten auf Erreichen einer Achsposition
- Kommandos für Programmverzweigungen
  - Unbedingter Sprung
  - Bedingter Sprung (Sprung bei Positionsvergleich, Sprung bei Wertevergleich, Sprung auf Grund von Zuständen)
  - Schleife
  - Start einer parallelen Sequenz
  - Stopp von parallelen Sequenzen
- Sonstige Kommandos
  - Kein Kommando (kennzeichnet Programmende)
  - Keine Operation (Programm wird nicht beendet)
  - Setzen eines booleschen Ausgangs
  - Setzen eines REAL-Werts
  - Inkrementieren eines REAL-Werts
  - Berechnen einer Variable nach einer Formel

Hinweis

Eine genaue Beschreibung der einzelnen Kommandos finden Sie im *Listenhandbuch Interpreter*.

2.2.2 Nutzen

Die Applikation **Interpreter** stellt eine einfache Schnittstelle zur Ansteuerung der grundlegenden Funktionalitäten von Technologieobjekten dar. Er verwaltet Rückmeldungen über Befehls- und Achsstatus. Der **Interpreter** erleichtert die Realisierung von Bewegungsabläufen in einem SIMOTION SCOUT Projekt.

2.3 Definitionen der Begriffe

Wichtige Definitionen und Begriffe

In diesem Dokument werden verschiedene Begriffe verwendet. Zur besseren Verständlichkeit werden die Begriffe in diesem Abschnitt definiert.

Kommando

Ein Kommando beinhaltet genau eine Anweisung für den **Interpreter**. Es besteht aus einem Namen und weiteren Parametern für die genauere Spezifizierung.

Beispiel: Kommando absolutes Positionieren

Tabelle 2- 1 CMD_POS_ABS

TO-Index	:= 0	Index im array für die TO, in diesem Beispiel ein Achs-Array
P1	:= 0	Position
P2	:= 10000	Geschwindigkeit
P3	:= 0	positive Richtung
mode	:= 0	direkte Programmierung
timeOut	:= 0	keine Überwachungszeit
nextCommand	:= WCD	Weiterschaltbedingung: weiterschalten, sobald Positionierung beendet

Verfahrsatz

Ein Verfahrsatz ist die kleinste Einheit eines Verfahrsatz-Programms. Er beinhaltet genau ein Kommando.

Verfahrsatz-Programm

Ein Verfahrsatz-Programm besteht aus einer Reihe von Verfahrsätzen (Spalte in der Verfahrsatz-Tabelle). Es ist gekennzeichnet durch eine Programm-Nummer und eine Kommando-Nummer. Es wird sequenziell abgearbeitet bis entweder das Kommando CMD_NONE als Ende-Kennzeichen gefunden wird, oder das letzte Kommando eines Programms abgearbeitet worden ist (Ende der Spalte im Array).

Verfahrsatz-Tabelle

Eine Verfahrsatz-Tabelle ist ein 2-dimensionales Array, wobei die Spalten durch die Programm-Nummer und die Zeilen durch die Kommando-Nummer gekennzeichnet sind. Jede Spalte innerhalb einer Verfahrsatz-Tabelle stellt ein Verfahrsatz-Programm dar.

Sequenz

Programme laufen in Sequenzen. Eine Sequenz ist der Ablauf von verschiedenen Kommandos, die nacheinander abgearbeitet werden. Es kann mehrere Sequenzen parallel geben. Diese werden gleichzeitig abgearbeitet. Die erste Sequenz wird durch das Kommando CMD_START_PROGRAM gestartet, alle weiteren Sequenzen können nur aus einem bereits laufenden Programm über den Befehl CMD_START_PARALLEL_SEQUENCE gestartet werden. Da Sprungbefehle auf beliebige Programme und Kommandos erlaubt sind, muss eine Sequenz nicht durchgehend immer dieselbe Programm-Nummer besitzen.

2.4 Konzept

Die Kommandos für die Maschinenabläufe werden tabellenartig in einem Verfahrsatz-Programm abgelegt. Dieses Verfahrsatz-Programm können Sie direkt in der jeweiligen Programmquelle programmieren oder über ein eventuell vorhandenes HMI parametrieren. Diese Datenstruktur wird an den **Interpreter** übergeben, welcher die einzelnen Kommandos interpretiert und anhand der Weiterschaltbedingungen abarbeitet. Die entsprechenden Rückmeldungen werden ebenfalls durch den **Interpreter** erzeugt.

Der Interpreter bietet hierfür drei verschiedene Interpreter-Modi:

- Programmbetrieb
- Einzelschritt
- Servicemode

Sie haben die Möglichkeit, eigene Kommandos zu definieren und können zusätzlich ein Programm für anwenderdefinierte Kommandos programmieren. Der **Interpreter** hat eine eigene Schnittstelle (AddOn-Schnittstelle) für anwenderdefinierte Kommandos, über die der Kommandoaustausch stattfindet.

Das Programm für die anwenderdefinierten Kommandos befindet sich nicht in der Bibliothek des **Interpreters**.

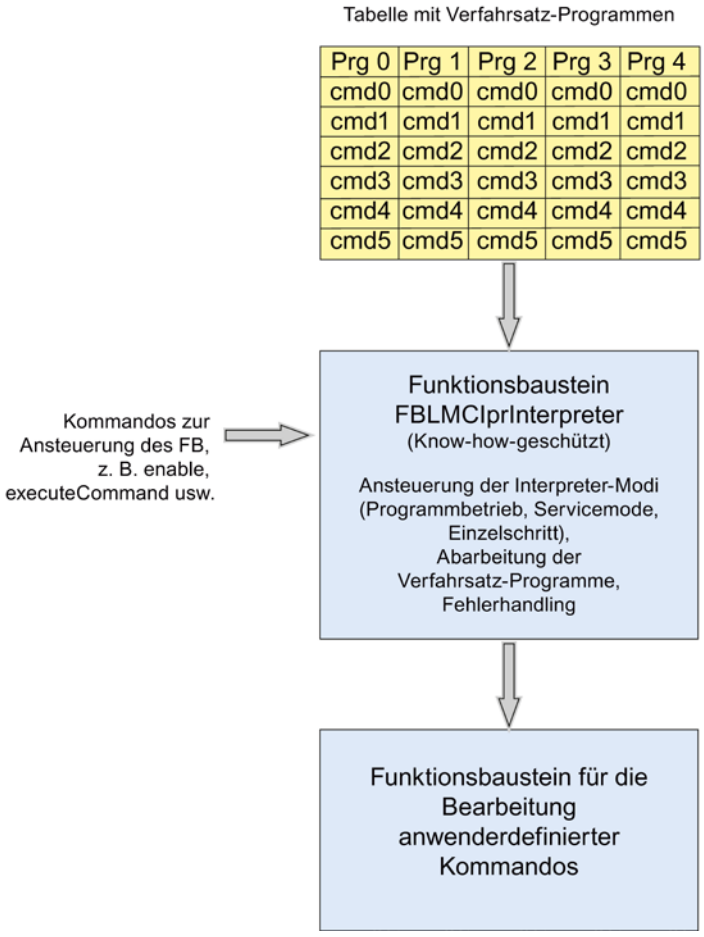


Bild 2-1 Übersicht zum Konzept des Interpreters

2.5 Automatisierungs- und Antriebsübersicht

2.5.1 Hardwarestruktur

Die Applikation **Interpreter** ist für das Motion Control Automatisierungssystem SIMOTION erstellt worden und setzt eine entsprechende Steuerung dieses Typs voraus. Sie kann für alle SIMOTION Plattformen (SIMOTION D, SIMOTION C und SIMOTION P) verwendet werden.

2.5.2 Systemvoraussetzungen

Die Applikation **Interpreter** kann ab SIMOTION SCOUT Version V4.2 SP1 verwendet werden.

2.5.3 Lieferumfang

Auf dem Auslieferungsmedium finden Sie die Bibliotheken

- **LMCBasic**
- **LMCIpr**

Die Bibliothek **LMCBasic** beinhaltet den **Achs-FB** und wird intern von der Applikation **Interpreter** genutzt. Beide Bibliotheken werden im XML-Format zur Verfügung gestellt.

Detaillierte Informationen der Bibliothek Achs-FB finden Sie Im Applikationshandbuch *SIMOTION Achs-Funktionsbaustein*.

2.6 Verfahrsatz-Tabelle

Verfahrsatz-Tabelle und deren Inhalte

In der Verfahrsatz-Tabelle werden alle Verfahrsatz-Programme abgelegt. Jedes Verfahrsatz-Programm besteht aus einzelnen Verfahrsätzen. In jedem Verfahrsatz wird genau ein Kommando aufgerufen.

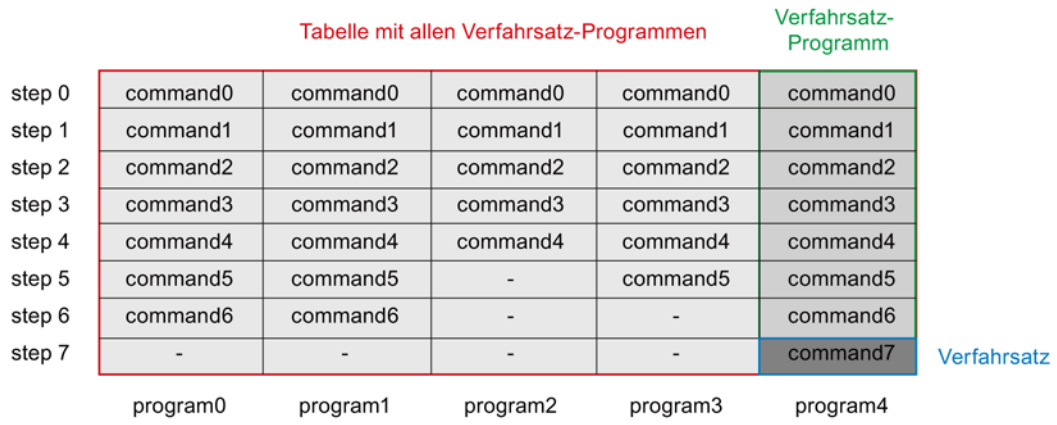


Bild 2-2 Tabelle für Verfahrsatz-Programme

Die Dimension der Verfahrsatz-Tabelle muss teilweise zum Zeitpunkt der Projektierung festgelegt werden. Es muss festgelegt werden, wie viele Verfahrsätze maximal innerhalb des Verfahrsatz-Programms enthalten sein können. Diese Vorgabe gilt dann für alle Programme. Das Verfahrsatz-Programm mit den meisten Zeilen bestimmt die Anzahl der Zeilen. Die Anzahl der Verfahrsatz-Programme wird erst beim Aufruf des **Interpreters** festgelegt. Dabei wird die SIMOTION-Funktionalität *variable Array-Länge* genutzt.

Die Zeilenanzahl der Verfahrsatz-Tabelle wird über eine globale Konstante LMCIPR_NUMBER_OF_COMMANDS festgelegt. Diese kann bei Bedarf geändert werden.

Hinweis

Wird die Konstante LMCIPR_NUMBER_OF_COMMANDS vom Anwender vergrößert, werden alle Verfahrsatz-Programme vergrößert.

Beendigung eines Verfahrsatz-Programms

Ein Verfahrsatz-Programm wird auf unterschiedliche Weise beendet:

- Durch Setzen des Kommandos CMD_STOP_PROGRAM am Eingang des FB
- Ein Verfahrsatz-Programm wird eigenständig beendet, wenn ein Verfahrsatz mit Kommando CMD_NONE oder der letzte Verfahrsatz eines Programms ohne Sprungkommando bearbeitet wurde.
- Umschalten in den Servicemode

Verwendung von Verfahrsatz-Programmen

Durch Verfahrsatz-Programme können einzelne Abläufe einer Maschine definiert werden. Diese können für den Ablauf in den verschiedenen Modi des **Interpreters** verwendet werden. Es können Sequenzen in den Verfahrsatz-Programmen definiert werden, die im Maschinenprogramm je nach Aufgabe der Maschine, miteinander kombiniert werden können. Beispiele für Aufgaben einer Maschine sind z. B: Referenzieren oder Anfahren einer Abfolge von Positionen. Innerhalb eines Verfahrsatz-Programms ist es möglich, über Sprungkommandos in andere Verfahrsatz-Programme zu wechseln.

Abarbeitung paralleler Sequenzen

Prinzipiell können mehrere Sequenzen parallel abgearbeitet werden. Die Anzahl der parallel möglichen Sequenzen wird über eine Konstante mit dem Namen LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES in der Unit **cPublic** eingestellt. Eine parallele Sequenz kann nur aus einem Programm heraus über das Kommando CMD_START_PARALLEL_SEQUENCE gestartet werden. Alle aktiven parallelen Sequenzen werden immer innerhalb eines Zyklus abgearbeitet. Dies erhöht die benötigte Rechenzeit.

Hinweis

Werden aus mehreren parallelen Sequenzen heraus im selben Zyklus für dieselbe Achse verschiedene Kommandos gleicher Priorität abgesetzt, so wird immer nur das erste Kommando ausgeführt. Die Kommandos aus späteren Sequenzen werden verworfen und es gibt einen entsprechenden Eintrag in der Diagnose-Struktur des **Interpreters**.

Verfahrsatz-Programm für den Fehlerfall

Alle Verfahrsatz-Programme sind gleichwertig, d. h., es gibt kein spezifisches Fehlerprogramm. Der Anwender entscheidet, welches Programm für welche Sequenz verwendet wird. Allerdings kann für den Fehlerfall ein Programm-Einsprung über Programm-Nummer und Kommando-Nummer definiert werden. Dieses Programm wird dann in der entsprechenden Situation abgearbeitet, falls es entsprechend projektiert wurde.

Zusätzlich kann auch projektiert werden, wie mit noch aktiven Sequenzen umgegangen werden soll, wenn ein Fehler auftritt. Dabei gibt es die folgenden Möglichkeiten:

- Keine Reaktion d. h., fehlerfreie Sequenzen laufen weiter
- Anhalten aller aktiven Sequenzen

In beiden Fällen kann zusätzlich ein Fehlerprogramm projektiert werden, um die Maschine kontrolliert anhalten zu können.

Struktur für die Reaktion eines Fehlers:

configData.sErrorProgram
configData.eErrorReaction

Folgende Enumeratoren sind möglich:

Tabelle 2- 2 Enumeratoren

Enum	Bedeutung
NONE	Keine weitere Reaktion
STOP_ALL_SEQUENCES	Alle Sequenzen werden abgebrochen.
USER_DEFINED	Es wird ein spezifisches Fehlerprogramm abgearbeitet.
USER_DEFINED_AND_STOP_ALL_SEQUENCES	Alle Sequenzen werden abgebrochen und es wird ein spezifisches Fehlerprogramm abgearbeitet.

Verfahrsatzprogramm für reset und fallende Flanke von enable

Auch hierfür kann ein spezifisches Programm definiert werden.
Zusätzlich kann die Reaktion für beide Fälle definiert werden. Hierfür gibt es zwei verschiedene Möglichkeiten:

- Anhalten und Deaktivieren aller TO
- Ausführen eines spezifischen Programms

Tabelle 2- 3 Enumeratoren

Enum	Bedeutung
STOP_ALL	Alles wird beendet, alle Technologieobjekte werden zurückgesetzt.
USER_DEFINED	Es wird ein spezifisches Stopp-Programm abgearbeitet.

Verfahrsatzprogramm für das Kommando CMD_STOP_PROGRAM

Ein spezifisches Programm kann auch mit dem Kommando CMD_STOP_PROGRAM ausgeführt werden, falls es entsprechend projektiert wurde.

Folgende Enumeratoren sind möglich:

Tabelle 2- 4 Enumeratoren

Enum	Bedeutung
NONE	Es wird kein Stopp-Programm bearbeitet.
USER_DEFINED	Es wird ein spezifisches Stopp-Programm bearbeitet.

Siehe auch

Verhalten bei verschiedenen Eingangs- und Ausgangsparameter des FB (Seite 43)

2.7 Kommandos

2.7.1 Allgemeines

Alle Kommandos des **Interpreters** werden im Dokument *Listenhandbuch Interpreter* detailliert beschrieben. Die Beschreibung ist nach den Aufgabenbereichen der Kommandos gegliedert. Die Auswahl eines Kommandos erfolgt über ein Enumerator (Enum). Allen Kommandos wird das Präfix CMD vorangestellt.

2.7.2 Aufbau eines Kommandos

Der Aufbau aller Kommandos ist einheitlich. Diese Struktur ist unten stehend abgebildet.

Bild 2-3 Aufbau eines Kommandos

Ein Kommando enthält folgende Bestandteile:

- Kommando:
  Das Kommando wird über ein Enumerator (Enum) vorgegeben. Eine Auflistung der Namen befindet sich in der Unit **dPublic**.
- TO-Index:
  Technologieobjekte (TO) für den **Interpreter** werden mit eigenen Arrays übergeben. Die TO können dann über den entsprechenden Array-Index angesprochen werden. Über den TO-Index wird das TO, z. B. die Achse bzw. der Nocken angegeben, auf welches das Kommando angewendet werden soll. Bei manchen Kommandos kann ausgewählt werden, ob es für ein spezielles oder für alle TO (Vorgabe von -1) ausgeführt werden soll. Bei einigen Befehlen ist auch ein externer Geber erlaubt, z. B. als Leitachse für einen Gleichlauf. Dieser muss dann ebenfalls im Array der Achsen eingetragen werden und wird über den zugehörigen Array-Index angesprochen.
  Da im Achs-Array unterschiedliche Achs-Typen vereinigt sind (Drehzahl-Achsen, Positionier-Achsen, Gleichlauf-Achsen, Bahnachsen, externe Geber) stellt der **Interpreter** in seiner Ausgangs-Schnittstelle ein Array mit den jeweils zugehörigen Typen zur Verfügung. Dieses Array kann z. B. von einem HMI ausgewertet werden.
- Parameter 1 bis 6:
  Für ein Kommando können bis zu sechs Parameter zur weiteren Spezifizierung angegeben werden. Bei falscher Programmierung eines Parameters wird das entsprechende Programm mit einer Fehlermeldung abgebrochen. Es wird ein Eintrag in die Diagnose-Struktur des **Interpreters** geschrieben. Bei einigen Ausnahmefällen wird eine Default-Projektierung verwendet oder der entsprechende Befehl wird ignoriert. Dies ist bei den einzelnen Befehlen explizit beschrieben.
  (siehe auch *Listenhandbuch Interpreter*)

- Mode:
  Mit dem Parameter *mode* kann ausgewählt werden, ob die Parameter 1 bis 6 absolut oder indirekt über einen Zeiger auf ein REAL-Array (Durchgangsvariable des **Interpreters**) vorgegeben werden oder ob die UserDefault-Werte von SIMOTION verwendet werden. Indirekte Vorgabe über eine Variable bedeutet, dass im entsprechenden Parameter nicht der geforderte Wert direkt steht, sondern der Index auf eine spezifische Variable im Variablenfeld *variables*, in der dann der tatsächliche Wert steht.
  Dabei bedeutet: Für jeden Parameter wird ein Bit in der Variablen *mode* verwendet: Das Bit 0 ist für den Parameter 1 zuständig. Bei Wert TRUE wird eine indirekte Vorgabe verwendet.
  Das Bit 7 wird für die Verwendung von UserDefault-Werten der Systemvariablen der TO verwendet. Dies wird jedoch derzeit nur für das Kommando CMD_ENABLE_OUTPUT_CAM verwendet. Ist Bit 7 gesetzt, dann werden die Bits 0 bis 5 ignoriert. Bit 6 hat keine Bedeutung und wird vom **Interpreter** ignoriert.
- TimeOut:
  Für Motion Control-Kommandos und Warte-Kommandos mit Ausnahme von CMD_WAIT_TIME kann eine Zeit definiert werden, nach deren Ablauf das Kommando spätestens fertig bearbeitet sein muss. Bei Überschreiten der vorgegebenen Zeit wird die betreffende Bearbeitung mit Fehler abgebrochen. Sinnvoll ist diese Angabe aber nur, wenn mit der Weiterschaltung gewartet wird, bis das Kommando erfolgreich abgearbeitet wurde. Bei der Voreinstellung von 0 ms wird diese Überwachung abgeschaltet.
- Weiterschaltbedingung:
  Bei allen Kommandos kann ausgewählt werden, ob das nächste Kommando sofort bearbeitet wird oder erst wenn das aktuelle Kommando erfolgreich ausgeführt und beendet wurde. Darüber hinaus können über diesen Parameter mehrere Kommandos geklammert werden d. h., sie werden im selben Interpreter-Zyklus ausgeführt. Ein Anwendungsbeispiel hierfür ist eine Berechnung von Stapelpositionen, die nicht von anderen Kommandos unterbrochen werden soll. Dies ist nur bis zu einem Maximalwert von Kommandos möglich, der über die Konstante LMCIPR_NUMBER_OF_COMMANDS_PER_CYCLE in der Unit **cPublic** vorgegeben wird, weil diese Funktionalität die Performancebelastung erhöht.
  Es gibt somit drei verschiedene Weiterschaltbedingungen:
  WHEN_COMMAND_DONE (WCD)
  IMMEDIATELY (IM)
  IMMEDEDIATELY_AND_NEXT_COMMAND (IM_NC)
  Bei Vorgabe eines falschen Werts wird WCD als Default-Projektierung verwendet und ein entsprechender Eintrag in der Diagnose-Struktur des **Interpreters** hinterlegt.

Hinweis

Bei manchen Kommandos sind bestimmte Werte für die Weiterschaltbedingung nicht sinnvoll. Diese können zwar programmiert werden, werden aber von der Applikation Interpreter ignoriert. Beispiel: Sprungbefehle mit der Programmierung Weiterschaltbedingung WCD. Sprünge werden prinzipiell in einem Interpretertakt abgearbeitet und wirken daher als wäre die Weiterschaltbedingung IM programmiert.

2.7.3 AddOn-Schnittstelle

Vorgehensweise für AddOn-Kommandos

AddOn-Kommandos sind Kommandos, die der **Interpreter** selbst nicht kennt und daher nicht bearbeiten kann. Ein solches Kommando wird in die AddOn-Schnittstelle eingetragen.
Randbedingung:
Werden bestimmte Parameter indirekt über das Variablenfeld vorgegeben, so kopiert der **Interpreter** die echten Werte in die entsprechenden Parameter und setzt den Mode im Kommando auf 0 (Null), siehe Abschnitt Aufbau eines Kommandos (Seite 19). D. h. die Programme, die die AddOn-Kommandos verarbeiten, können immer von einer direkten Programmierung ausgehen.

Ein zusätzliches Programm, das diese Kommandos bearbeitet, setzt als erstes das Bit *boRunning*, um zu signalisieren, dass das Kommando erkannt wurde und in Bearbeitung ist. Stellt der **Interpreter** im nächsten Bearbeitungstakt fest, dass *boRunning* nicht gesetzt, das Kommando aber noch aktiv ist, so bedeutet dies, dass kein Auswerte-Programm für dieses Kommando existiert und es wird ein Fehler der Klasse 3 abgesetzt.

Sobald das Kommando fertig bearbeitet wurde, muss es in der Schnittstelle quittiert werden, indem es auf CMD_NONE gesetzt wird. Gleichzeitig mit der Quittierung kann ein Fehler vom Anwenderprogramm abgesetzt werden, der vom **Interpreter** dann in die Diagnose-Struktur eingetragen wird. Die weitere Reaktion des **Interpreters** hängt von der eingetragenen Fehlerklasse ab:

- Bei Fehlerklasse 1 und 2 wird der Ablauf weiter bearbeitet
- Bei Fehlerklasse 3 wird die aktuelle Sequenz abgebrochen und der Ausgang *error* gesetzt
- Bei Fehlerklasse 4 wird der Ausgang *error* gesetzt. Ob das Programm abgebrochen wird, hängt von der aktuellen Einstellung der Fehlerreaktion ab.

Detaillierte Informationen zu Fehlern finden Sie im Kapitel Alarm- und Fehlermeldungen (Seite 55).

Auch die Zeitüberwachung des Kommandos wird, sofern programmiert, vom **Interpreter** vorgenommen. Ist die Überwachungszeit abgelaufen, bevor das Kommando auf CMD_NONE gesetzt worden ist, so reagiert der **Interpreter** mit einem Fehler der Klasse 3 d. h., das Verhalten ist identisch, wie bei den eigenen Kommandos des **Interpreters**. Wird ein Auswerteprogramm im gleichen Zyklus, in dem das AddOn-Kommando eingetragen wurde, mit der Bearbeitung fertig, so muss das Bit *boRunning* nicht gesetzt werden. Die Quittierung des Kommandos über CMD_NONE reicht in diesem Fall aus.

Stellt der **Interpreter** selbst einen Fehler in einem AddOn-Kommando fest, z. B. Index einer Variablen außerhalb der Array-Grenzen bei indirekter Programmierung oder Überschreitung der eingestellten Zeitüberwachung, so quittiert er die AddOn-Schnittstelle durch Eintrag von CMD_NONE.

Für jede mögliche Sequenz existiert eine eigene AddOn-Schnittstelle.

Hinweis

Der Funktionsbaustein des **Interpreters** muss in der gleichen oder einer langsameren Task wie das Programm der AddOn-Funktionalität aufgerufen werden.

Im **Servicemode** wird nur die Schnittstelle mit dem Index 0 bedient.

Wird ein Kommando übergeben, welches sich im reservierten Nummernband befindet, aber nicht bekannt ist, muss vom AddOn-Programm ein Fehler ausgegeben werden.

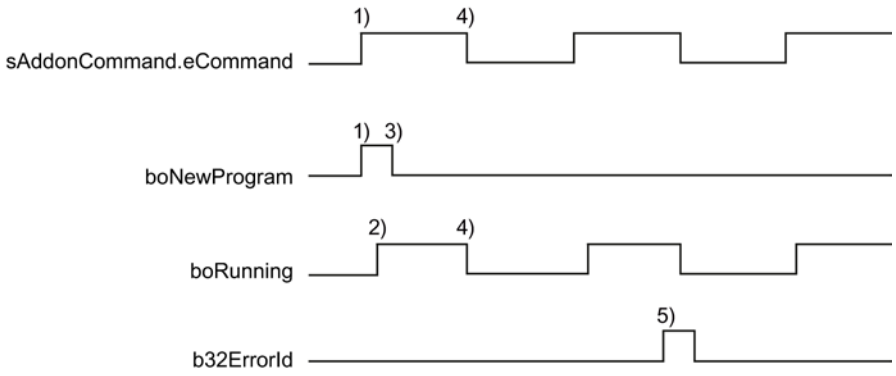
Datenstruktur der AddOn-Schnittstelle

Im Folgenden ist die Datenstruktur *sAddonCommand* dargestellt.

Tabelle 2- 5 Elemente von **sAddonCommandType**

Element	Datentyp	Bedeutung
sAddonCommand	aLMCIprCommandType	Kommando
sModuleResult	sLMCIprModuleResultType	Intern verwendetes Datum
i32ErrorId	DWORD	Fehler-Kennung
i32ErrorAddVal	DINT	Zusatz-Information zum Fehler, wird vom **Interpreter** in *i32AdditionalValue4* in die Diagnose-Struktur eingetragen.
u8ErrorClass	USINT	Fehlerklasse
boNewProgram	BOOL	Signalisiert, dass es sich um das erste Kommando eines neu gestarteten Programms handelt.
boRunning	BOOL	Zeigt an der AddOn-Schnittstelle an, dass ein Programm oder eine Sequenz in Bearbeitung ist.

Zeitdiagramm



1) Das Signal setzt die Applikation Interpreter
2) Das Signal muss der Anwender setzen
3) Für einen Takt beim ersten Kommando eines Programms
4) Das Signal muss der Anwender zurücksetzen
5) Das Signal setzt der Anwender, wenn er eine Fehlerreaktion erreichen möchte; zusätzlich müssen noch die Informationen in i32ErrorAddVal und u8ErrorClass gesetzt werden

Bild 2-4 Zeitdiagramm AddOn-Schnittstelle

2.7.4 Anwenderdefinierte Kommandos

Allgemeines

Für eine Erweiterung des **Interpreters** um eigene Kommandos können Sie diese in einem eigenen Programm erstellen. Über die AddOn-Schnittstelle des **Interpreters** werden diese zusätzlichen Kommandos an das Anwender-Programm übergeben. Um eine Kollision mit den zukünftigen Kommandos des Interpreters zu vermeiden, wird diesen Kommandos ein eigenes Nummernband zugewiesen d. h., in der AddOn-Schnittstelle werden nur Kommandos eingetragen, wenn sich die Kommando-Nummer im Nummernband größer als 10000 befindet. Die Kommandos der Applikation **Interpreter** dürfen nicht verändert werden.

Anwenderdefinierte Kommandos beginnen ab dem Wert 10000.

Hinweis

Für eine Unterscheidung zwischen ausgelieferten und anwenderdefinierten Kommandos wird zur deutlicheren Darstellung der Kommandos empfohlen, das Präfix der Konstanten auf CMD_U_ festzulegen. Das U steht für User.

Beispiel: CMD_U_TORQUE_LIMITING

Die Enumeratoren für die anwenderdefinierten Kommandos werden, wie die Interpreter-Kommandos, in der Unit **dPublic** definiert. Dabei ist darauf zu achten, dass die Bezeichnung der Enumeratoren für die bereits existierenden Kommandos des **Interpreters** nicht geändert wird.

Bibliotheksbeschreibung 3

3.1 Struktur der Bibliothek LMCIpr

Die Applikation **Interpreter** beinhaltet eine Bibliothek mit dem Namen **LMCIpr**. Alle Präfixe in dieser Bibliothek wurden entsprechend benannt.

Tabelle 3- 1 Units der Bibliothek **LMCIpr**

Unit	Bedeutung	Know-how-Schutz
aVersion	Unit für Versionsgeschichte	nein
cProtected	Unit für geschützte Konstanten	ja
cPublic	Unit für vom Anwender änderbare Konstanten	nein
dProtected	Unit für geschützte Daten	ja
dPublic	Unit für vom Anwender erweiterbare Typdeklarationen (Definition von Kommandos)	nein
fInterpreter	Unit mit Funktionsbausteinen für den **Interpreter**	ja
fTools	Unit für Hilfsprogramme	nein

Die Unit **fTools** ist eine Unit mit Hilfsfunktionen für den **Interpreter**.

3.2 Konstanten

Tabelle 3- 2 Konstanten im Interpreter

Name	Datentyp	Wert	In Unit	Anmerkung
LMCIPR_NUMBER_OF_COMMANDS	UINT	20	**cPublic**	Anzahl der Verfahrsätze innerhalb eines Verfahrsatz-Programms
LMCIPR_NUMBER_OF_COMMANDS_PER_CYCLE	USINT	3	**cPublic**	Maximale Anzahl der in einem Zyklus bearbeiteten Kommandos
LMCIPR_NUMBER_OF_LOOPS_PER_SEQUENCE	USINT	5	**cPublic**	max. Anzahl geschachtelter Schleifen
LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES	USINT	6	**cPublic**	Maximale Anzahl von parallel aktiven Sequenzen
LMCIPR_NUMBER_OF_AXES	UINT	6	**cPublic**	Maximale Anzahl verwendeter Achs-FB-Instanzen im **Interpreter**
LMCIPR_SIZE_OF_ERRORBUFFER	UINT	100	**cPublic**	Größe der Diagnose-Struktur des **Interpreters**
LMCIPR_COMMANDTRACE_ACTIVE	BOOL	FALSE	**cPublic**	Schalter für internen Kommando-Trace des **Interpreters**
LMCIPR_SIZE_OF_COMMANDTRACE	UINT	1000	**cPublic**	Größe des Kommando-Trace
LMCIPR_NO_MODE	USINT	0	**cProtected**	Wert für *kein aktiver Interpreter-Modus*
LMCIPR_PROGRAM_MODE	USINT	1	**cProtected**	Wert für **Programmbetrieb**
LMCIPR_SINGLESTEP_MODE	USINT	2	**cProtected**	Wert für **Einzelschrittbetrieb**
LMCIPR_SERVICE_MODE	USINT	3	**cProtected**	Wert für **Servicemode**

3.3 Enumeratoren

Über die folgenden Enumeratoren werden bestimmte Zustände angezeigt bzw. Angaben zur Fehlerreaktion gemacht:

Tabelle 3- 3 Enumeratoren von *eLMCIPrProgramStateType*

Enumerator	Fehlerreaktion
NOT_RUNNING	Programm wird nicht bearbeitet
RUNNING	Programm ist in Bearbeitung
SINGLESTEP_ACTIVE	Es ist ein Programm im Einzelschrittbetrieb aktiv und es sind noch nicht alle Kommandos dieses Schritts fertig bearbeitet. Sobald dies der Fall ist, wechselt der Programmstatus wieder auf RUNNING.

Enumerator	Fehlerreaktion
ABORTED	Programm wurde durch einen Fehler oder durch das Kommando CMD_STOP_SEQUENCE abgebrochen.
STOPPED	Programm wurde durch das Kommando CMD_STOP_PROGRAM oder einen Wechsel des Interpreter-Modus angehalten.
STOP_PROGRAM_RUNNING	Ein vom Anwender programmiertes Fehler-Programm oder Stopp-Programm wird abgearbeitet.

Tabelle 3- 4 Enumeratoren von *eLMCIPrErrorReactionType*

Enumerator	Fehlerreaktion
NONE	Keine weitere Reaktion
STOP_ALL_SEQUENCES	Anhalten aller Abläufe
USER_DEFINED	Abarbeiten eines spezifischen Fehlerprogramms
USER_DEFINED_AND_ STOP_ALL_SEQUENCES	Anhalten aller Abläufe und Abarbeiten eines spezifischen Fehlerprogramms

Tabelle 3- 5 Enumeratoren von *eLMCIprStopReactionType*

Enumerator	Fehlerreaktion
STOP_ALL	Anhalten aller Bewegungen/Deaktivieren aller TOs
USER_DEFINED	Abarbeiten eines spezifischen Stopp-Programms

Tabelle 3- 6 Enumeratoren von *eLMCIprStopCmdReactionType*

Enumerator	Fehlerreaktion
NONE	CMD_STOP_PROGRAM beendet nur alle aktiven Sequenzen
USER_DEFINED	Zusätzlich wird ein spezifisches Stopp-Programm ausgeführt.

Tabelle 3- 7 Enumeratoren von *eNextCommandType*

Enumerator	Beschreibung
WCD	WHEN_COMMAND_DONE
IM	IMMEDIATELY
IM_NC	IMMEDIATELY_AND_NEXT_COMMAND

3.4 Konfiguration

Notwendige Konfigurationseinstellungen

In diesem Abschnitt wird beschrieben, welche Konfigurationseinstellungen der Anwender in seinem Projekt vornehmen muss.

Konstanten

Folgende Konstanten muss der Anwender an seine Anforderungen in der Unit **cPublic** der Bibliothek **LMCIpr** anpassen.

Tabelle 3- 8 Konstanten, die über die Anweisung *define* vorgegeben werden

Name Anweisung	Voreinstellung
#define DEFINE_LMCIPR_NUMBER_OF_AXES	6
#define DEFINE_LMCIPR_NUMBER_Of_PARALLEL_SEQUENCES	6

Tabelle 3- 9 Tatsächliche Konstanten

Konstante	Voreinstellung
LMCIPR_NUMBER_OF_COMMANDS	20
LMCIPR_NUMBER_OF_COMMANDS_PER_CYCLE	3
LMCIPR_SIZE_OF_ERRORBUFFER	100

Tabelle 3- 10 Wenn Kommando-Trace verwendet werden soll

Konstante	Voreinstellung
LMCIPR_SIZE_OF_COMMANDTRACE	1000
LMCIPR_COMMAND_TRACE_ACTIVE	FALSE TRUE für die Aktivierung

Definition der Array-Längen

Im Anwenderprogramm muss der Anwender Übergabe-Arrays für den Aufruf des **Interpreters** definieren. Die Array-Länge muss auf die nachfolgenden Parameter angepasst werden.

Hinweis

Die Arrays von *axisConfigData* und *actualValues* müssen dieselben Array-Grenzen besitzen, da sich beide auf die verwendeten Achsen beziehen.

Tabelle 3- 11 Array-Längen anpassen

Parameter	Name des Array
Anzahl der für den Interpreteraufruf tatsächlich verwendeten Achsen	sLMCIprAxisConfigDataType sLMCIprActualValueType
Anzahl der für den Interpreter verwendeten Nocken	outputCamType
Anzahl der für den Interpreter verwendeten Kurvenscheiben	camType
Anzahl der für den Interpreter verwendeten Programme	sLMCIprProgramType
Anzahl der für den Interpreter verwendeten Bedingungen	BOOL
Anzahl der für den Interpreter verwendeten Variablen	REAL

Beispiel für den Aufruf des Interpreters

In diesem Beispiel werden folgende Parameter verwendet:

- 6 Achsen
- 3 Nocken
- 5 Kurvenscheiben
- 20 Programme
- 100 Bedingungen
- 50 Variablen

Tabelle 3- 12 Beispiel

```
instFBLMCIprInterpreter(
    enable         := gboEnable,
    reset          := gboReset,
    ackError       := gboAckErrors,
    executeCommand := gboExecuteCommand,
    mode           := gu8Mode,
    command        := gsCommand,     // Typ sLMCIprCommandType
    jogFunction    := gsJogFunction, // Typ  sLMCIprJogFunctionDataType

         // VAR_IN_OUT handling
    axisConfigData    := gasAxisConfigData,
         //z. B. ARRAY[0..5] OF sLMCIprAxisConfigDataType
    outputCams        := gaOutputCams,
         // z. B. ARRAY[0..2] OF outputCamType
    cams              := gaCams,
         // z. B. ARRAY[0..4] OF camType
    programTable      := gsProgramTable,
         // z. B. ARRAY[0..19] OF sLMCIprProgramType
    conditions        := gaboConditions,
         // z. B. ARRAY[0..99] OF BOOL
    variables         := gar32Variables,
         // z. B. ARRAY[0..49] OF REAL
    configData        := gsConfigData,
```

```
     // Typ sLMCIprInterpreterConfigType
actualValues      := gasActualValues,
     // z. B. ARRAY[0..5] OF sLMCIprActualValueType
addonFunctions    := gasAddOnFunctions
     //Typ ARRAY[0..LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES] OF
  sLMCIprAddonDataType
);
```

Bei den Durchgangsparametern des **Interpreters** *axisConfigData* und *actualValues* werden die Array-Grenzen beim Aufruf festgelegt. Für die intern verwendeten Daten ist dies nicht möglich. Es liegt deshalb in der Verantwortung des Anwenders, dass die Konstante für die intern verwendeten Daten (Anzahl der Achsen) nicht kleiner ist als das übergebene Array Komponenten hat. Ein kleineres Array zu übergeben ist jederzeit möglich und genau dann notwendig, wenn z. B. mehrere **Interpreter** für jeweils unterschiedliche Achs-Verbände verwendet werden sollen. In diesem Fall muss die Konstante in der Unit **cPublic** das Maximum der verwendeten Achsen beinhalten.

Beispiel

In einem Projekt werden drei Instanzen des Interpreters eingesetzt für drei verschiedene Maschineneinheiten.

- Maschineneinheit 1 verwendet 4 Achsen
- Maschineneinheit 2 verwendet 8 Achsen
- Maschineneinheit 3 verwendet 4 Achsen

Die Achs-Array für z. B. *axisConfigData* müssen dann wie folgt definiert sein:

- Maschineneinheit 1: ARRAY[0..3] OF sLMCIprAxisConfigDataType
- Maschineneinheit 2: ARRAY[0..7] OF sLMCIprAxisConfigDataType
- Maschineneinheit 3: ARRAY[0..3] OF sLMCIprAxisConfigDataType

Die Konstante in der define-Anweisung der Unit **cPublic** muss somit auf 8 eingestellt werden. Das entspricht dem Maximum aller jeweils verwendeten Achsen.

Konfigurationsdaten

Die Konfigurationsdaten des **Interpreters** *axisConfigData* und *configData* sind mit bestimmten Default-Werten vorbelegt, z. B. sind alle Achs-Dynamik-Werte mit -1.0 vorbelegt. Damit werden dann die "Default-Werte" des SIMOTION-Systems verwendet.

Vom Anwender können in die Konfigurationsdaten für die spezifische Anwendung passenden Größen eingetragen werden, z. B. per Initialisierungsprogramm in der Startup-Task.

Übergabe der Technologieobjekte (TO)

Alle Technologieobjekte, wie Achsen, externe Geber, Nocken, Kurvenscheiben, usw. werden prinzipiell mit Arrays übergeben und in den Kommandos über den jeweiligen Array-Index projektiert.

Die Arrays müssen vom Anwender im Programm mit den jeweiligen Technologieobjekten befüllt werden, z. B. über ein Initialisierungsprogramm in der Startup-Task.

Tabelle 3- 13 Die Datentypen für die einzelnen Arrays lauten wie folgt:

axisConfigData[i].toAxis :	_Axis_Ref
outputCams[i] :	outputCamType
Cams[i] :	camType

Funktionsbeschreibung

4

4.1 Allgemeines zur Funktionsbeschreibung

Die Ansteuerung des Funktionsbausteins **FBLMCBasicInterpreter** ist bezüglich des Verhaltens bei den Eingängen *execute* und *enable* der PLCopen Norm V1.1 nachempfunden.

Der Funktionsbaustein ist in der Programmiersprache Structured Text (ST) erstellt. Er ist für die Verwendung in einer zyklischen Task programmiert, typischerweise die BackgroundTask.

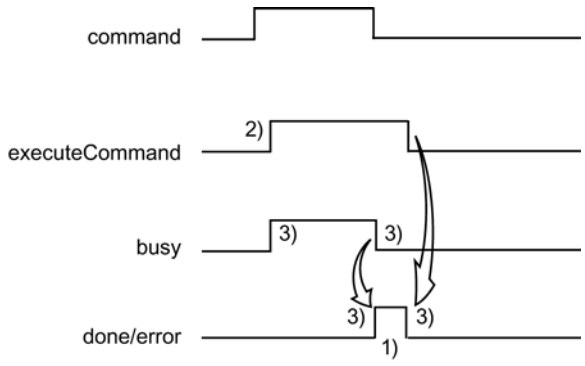
4.2 Eigenschaften des Interpreters

4.2.1 Ausführung eines Kommandos

Ein Kommando für die Ansteuerung des **Interpreters** wird am Eingang *command* übergeben. Mit einer positiven Flanke des Eingangs *executeCommand* wird dieses Kommando ausgeführt.

Solange das Kommando noch in Bearbeitung ist, wird der Ausgang *busy* gesetzt.

Nach Fertigstellung wird der Ausgang *done* gesetzt. Wechselt der Zustand des Eingangs *executeCommand* vor Fertigstellung des Kommandos auf FALSE, so wird der Ausgang *done* für nur einen Zyklus gesetzt. Wird ein Kommando bereits im ersten Zyklus beendet, wie z. B. bei CMD_START_PROGRAM, CMD_NEXT_STEP, so wird sofort *done* gesetzt ohne vorheriges *busy*.



1) Für einen Takt
2) Das Signal setzt der Anwender
3) Das Signal setzt die Applikation Interpreter
   Das Signal setzt die Applikation Interpreter zurück

Bild 4-1 Zeitdiagramm für die Ausführung eines Kommandos

4.2.2 Interpreter-Modi

Die Applikation **Interpreter** kann in den Interpreter-Modi **Servicemode**, **Programmbetrieb** und **Einzelschritt** angewendet werden. Die Interpreter-Modi werden über den Eingang *mode* angewählt.

Ermittlung des aktuellen Interpreter-Modus

Über den Eingang mode wird der aktuelle Interpreter-Modus festgelegt. Es wird geprüft, ob der aktuelle Interpreter-Modus von dem angeforderten Interpreter-Modus abweicht. Ist dies der Fall, wird untersucht, ob ein gültiger Wert für diesen Interpreter-Modus gewählt wurde. Ist dieser nicht gültig, wird ein Fehler ausgegeben. Anderenfalls wird der Interpreter-Modus umgeschaltet.

Mögliche Interpreter-Modi

Tabelle 4- 1 Interpreter-Modi in der Applikation Interpreter

Interpreter-Modus	Wert	Datentyp
LMCIPR_NO_MODE	0	USINT
LMCIPR_PROGRAM_MODE	1	USINT
LMCIPR _SINGLESTEP_MODE	2	USINT
LMCIPR_SERVICE_MODE	3	USINT

Interpreter-Modus Servicemode

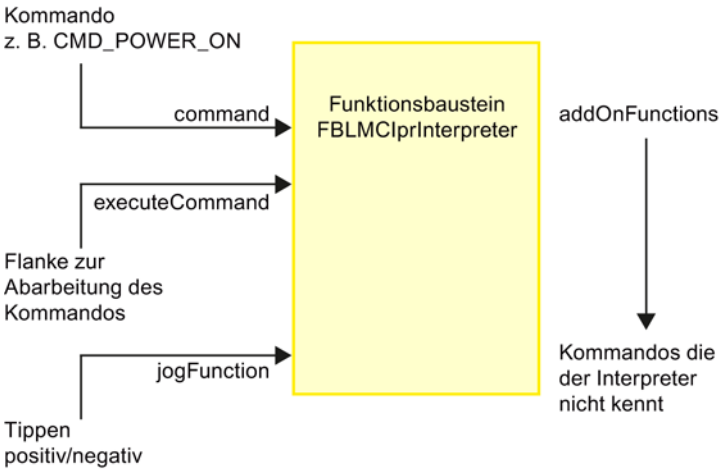


Bild 4-2 Darstellung des Interpreter-Modus Service-Mode

Erlaubte Kommandos im Service-Mode

- Alle Motion-Control Kommandos
- Alle *Sonstigen Kommandos*, lt. *Listenhandbuch Interpreter*

Im Interpreter-Modus Servicemode wird ein Kommando zur Ausführung direkt am Eingang *command* vorgegeben. Bei einer positiven Flanke von *executeCommand* wird das Kommando ausgeführt. Es wird immer nur ein Kommando pro Zyklus ausgeführt.

Im **Servicemode** sind keine Kommandos zur Programmsteuerung, Sprungkommandos und Warte-Kommandos erlaubt. Diese Kommandos werden ignoriert. Der Ausgang *error* am **Interpreter** wird gesetzt. Dieser Fehler muss nicht quittiert werden, um ein neues Kommando starten zu können.

Die Befehle CMD_NONE und CMD_NOP werden im Servicemode ohne Fehlereintrag ignoriert.

Tippen im Servicemode

Im **Servicemode** können einzelne Achsen getippt werden.

Für die Funktion **Tippen** gibt es eine eigene Schnittstelle am **Interpreter**. Damit kann eine spezifische Achse über BOOL-Variablen vorwärts oder rückwärts getippt werden.

Das Tippen ist ausschließlich im Servicemodus möglich und kann direkt über den Parameter *jogFunction* angesteuert werden. Für die Dynamiken können eigene Werte in der Konfigurations-Struktur vorgegeben werden.

Tabelle 4- 2 Elemente von **sLMCIprInterpreterJogDataType**

Element	Datentyp	Bedeutung
i32IndexAxis	DINT	Index auf Achs-Array für zu tippende Achse.
r32Increment	REAL	Zu verfahrendes Inkrement, sofern *boModeInc* auf TRUE steht
boModeInc	BOOL	FALSE: Tippen kontinuierlich (Default) TRUE: Tippen Inkrement
boJogPositive	BOOL	Tippen positiv
boJogNegative	BOOL	Tippen negativ

Wenn vorher ein Gleichlaufverband über Einzel-Kommando aktiviert wurde, können auch verkoppelte Achsverbände getippt werden.

Zusätzliche Kommandos im Servicemode

Zusätzlich können im **Servicemode** alle Motion Control-Kommandos (siehe Kapitel Kommandos (Seite 19)) für Technologieobjekte sowie Kommandos zum Setzen von Variablen und Bedingungen über den Eingang *command* an den Funktionsbaustein übergeben werden. Des Weiteren können anwenderdefinierte Kommandos bearbeitet werden. Andere Kommandos sind im **Servicemode** nicht erlaubt. Dies betrifft alle Sprung- und Programmbearbeitungsbefehle sowie die Warte-Kommandos.

Die Kommandos werden direkt ausgeführt, wobei dieselben Überwachungen wie im Programmbetrieb aktiv sind. Ebenso gelten weiterhin die Überwachungszeiten. Ein etwaiger Fehler im **Servicemode** muss nicht explizit quittiert werden, bevor ein neues Kommando ausgeführt werden kann.

Kommandos, die mit WCD programmiert wurden, werden vom Interpreter erst dann als beendet angezeigt, wenn sie wirklich beendet sind. Sie können aber bereits vorher durch ein anderes Kommando abgelöst werden.

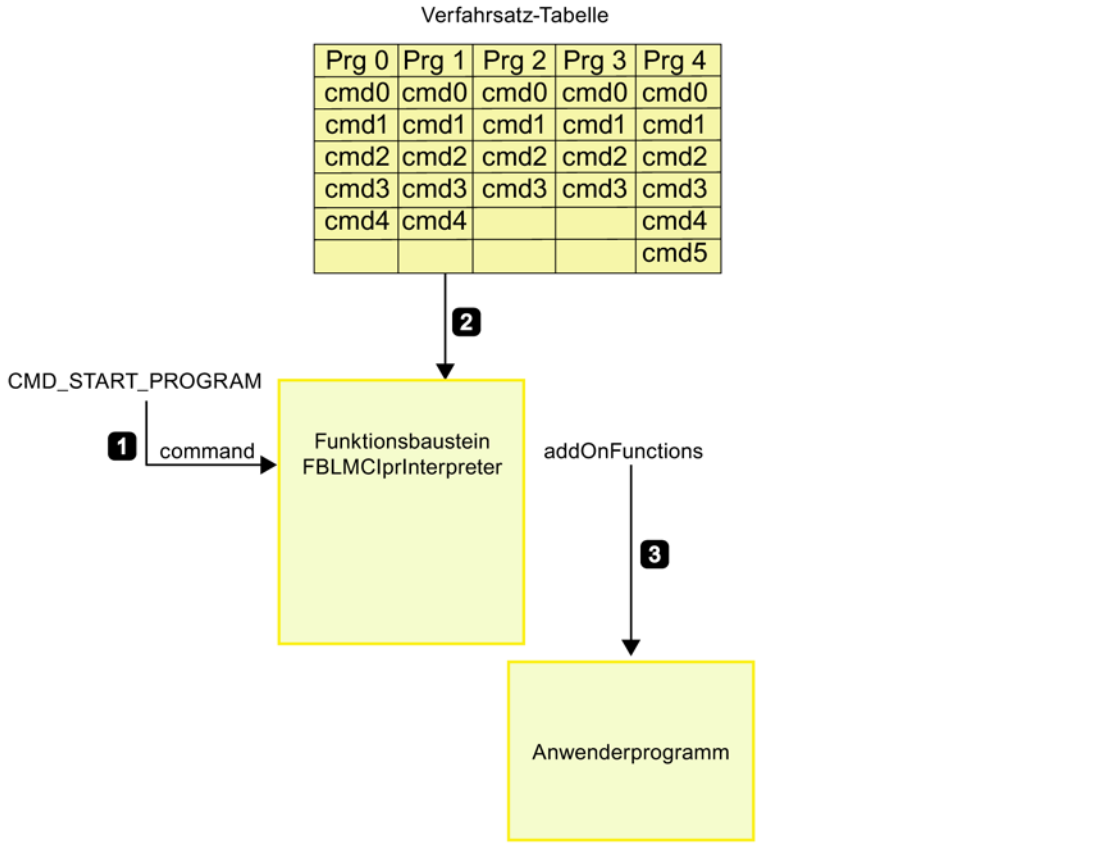
Interpreter-Modus Programmbetrieb

Im **Programmbetrieb** können über den Eingang *command* folgende Kommandos zur Programmansteuerung ausgeführt werden:

- CMD_START_PROGRAM
- CMD_STOP_PROGRAM
- CMD_NEXT_STEP ist nur für **Einzelschritt** zu verwenden.

Über diese Kommandos werden Verfahrsatz-Programme gestartet und gestoppt. Wird ein Programm auf Grund eines Fehlers abgebrochen, muss dieser Fehler zuerst quittiert werden, bevor ein neues Programm gestartet werden kann.

Im **Programmbetrieb** werden die Verfahrsatz-Programme bearbeitet, bis die Bearbeitung über ein Kommando im Programm (CMD_NONE) angehalten oder das Verfahrsatz-Programm beendet ist. Das Kommando CMD_STOP_PROGRAM beendet ebenfalls ein Programm einschließlich aller aktiven parallelen Sequenzen. Bewegungen die vorher gestartet wurden, bleiben jedoch aktiv.



1 Schritt 1: Der Anwender startet das Programm

2 Schritt 2: Die Verfahrsatz-Programme werden vom Interpreter abgearbeitet

3 Schritt 3: Wenn zusätzlich Funktionen vorhanden sind, werden diese in die addon-Schnittstelle eingetragen

Bild 4-3 Darstellung des Interpreter-Modus Programmbetrieb

Der **Programmbetrieb** wird über Kommandos angesteuert. Mit diesen Kommandos werden Verfahrsatz-Programme gestartet und angehalten. Diese Kommandos werden über den Eingang *command* übergeben. Bei einer steigenden Flanke am Eingang *executeCommand* wird geprüft, ob ein neues Kommando zur Programmsteuerung ansteht. Ist dies der Fall, wird es abgearbeitet. Anderenfalls wird es ignoriert und ein Fehler wird ausgegeben.

Für den aktuellen Verfahrsatz wird geprüft, ob dieser ein Kommando enthält. Ist dies nicht der Fall, ist das Verfahrsatz-Programm beendet. Ebenso ist das Verfahrsatz-Programm beendet, wenn der letzte Verfahrsatz eines Programms fertig bearbeitet wurde.

Bei Projektierung mit IMMEDIATELY_AND_NEXT_COMMAND (IM_NC) wird das nächste Kommando eines Verfahrsatz-Programms sofort nach Beendigung des aktuellen Kommandos bearbeitet. Diese Schleife wird bis zur maximalen Anzahl von Kommandos pro Zyklus abgearbeitet bzw. bis sie über eine andere Weiterschaltbedingung abgebrochen wird.

Einzelschritt im Programmmode

Im Interpreter-Modus **Einzelschritt** wird ein Verfahrsatz-Programm schrittweise ausgeführt. Bei Start des Verfahrsatz-Programms über CMD_START_PROGRAM wird der erste Verfahrsatz ausgeführt. Mit Übergabe des Kommandos CMD_NEXT_STEP über den Eingang *command* sowie einer positiven Flanke am Eingang *executeCommand* wird die Bearbeitung des nächsten Verfahrsatzes durchgeführt. Dies gilt für alle aktiven Sequenzen. Das Kommando CMD_NEXT_STEP wird ignoriert, wenn ein Kommando in einer Sequenz nicht beendet wurde. Dies ist am Ausgang *programState* des **Interpreters** erkennbar.

Wechsel zwischen den Interpreter-Modi

Tabelle 4- 3 Aktionen beim Wechsel zwischen den Interpreter-Modi

Wechsel der Interpreter-Modus		Aktion
von	zu	
Programmbetrieb	**Servicemode**	• Sofortiger Abbruch des aktuellen Programms, einschließlich aller parallelen Sequenzen. Aktive Kommandos bleiben jedoch aktiv. Programmstatus geht in STOPPED, sofern ein Programm aktiv war. • Wechsel in den neuen Interpreter-Modus
Einzelschritt	**Servicebetrieb**	• Sofortiger Abbruch des aktuellen Programms, einschließlich aller parallelen Sequenzen. Aktive Kommandos bleiben jedoch aktiv. Programmstatus geht in STOPPED, sofern ein Programm aktiv war. • Wechsel in den neuen Interpreter-Modus
Programmbetrieb	**Einzelschritt**	Wechsel sofort (immer möglich)
Einzelschritt	**Programmbetrieb**	Wechsel sofort (immer möglich)
Servicemode	**Programmbetrieb**	Wechsel in den neuen Interpreter-Modus
Servicebetrieb	**Einzelschritt**	Wechsel in den neuen Interpreter-Modus

Siehe auch

Aufgabenstellung (Seite 11)

4.2.3 Verhalten bei nicht erlaubten Kommandos

Wird in einem Interpreter-Modus am Eingang *command* ein Kommando erkannt, welches für den jeweiligen Interpreter-Modus nicht gültig ist, dann wird dieses Kommando ignoriert. Es wird eine entsprechende Fehlermeldung ausgegeben.

Beispiele:

- Ein Motion Control-Kommando bei **Programmbetrieb** oder **Einzelschritt**
- Das Kommando für Start eines Verfahrsatz-Programms im **Servicemode**

Wird bei der Bearbeitung der Applikation **Interpreter** ein Fehler der Klasse 3 oder 4 festgestellt, so wird dieser an den Ausgängen *error* und *errorId* gemeldet. Zusätzlich kann, je nach Einstellung der Konfigurationsdaten des **Interpreters**, ein Fehlerprogramm gestartet werden, welches vom Anwender projektiert und in den Konfigurationsdaten des **Interpreters** angegeben werden muss.

4.2.4 Abarbeitung der Kommandos

Ein Verfahrsatz-Programm kann mehrere Kommandos in einem Zyklus ausführen, sofern die Weiterschaltbedingung IMMEDIATELY_AND_NEXT_COMMAND (IM_NC) programmiert wurde. Bei Projektierung mit IM_NC wird das nächste Kommando eines Verfahrsatz-Programms sofort nach Beendigung des aktuellen Kommandos bearbeitet. Es wird eine projektierbare Anzahl an Kommandos pro Zyklus ausgeführt, siehe Konstante LMCIPR_NUMBER_OF_COMMANDS_PER_CYCLE. Über diese Konstante in der Unit **cPublic** wird eingestellt, wie viele Kommandos innerhalb eines Zyklus maximal bearbeitet werden können. Wird die Grenze innerhalb eines Zyklus erreicht, dann wird das nächste Kommando erst im nächsten Aufruf ausgeführt. Da die Bearbeitungs-Schleife zulasten der Performance geht, sollte diese Konstante nicht zu groß eingestellt werden.

Es ist immer nur ein Kommando pro Achse aktiv. Wird während der Ausführung eines Motion Control-Kommandos an einer Achse ein weiteres Motion Control-Kommando abgesetzt, so wird das erste Kommando abgebrochen und das zweite Kommando ausgeführt. Die Ausnahme sind überlagerte Verfahrbefehle, hier können zwei Motion Control-Kommandos gleichzeitig an einer Achse aktiv sein.

Hinweis

Wenn aus verschiedenen parallelen Sequenzen gleichzeitig Kommandos an dieselbe Achse gegeben werden, wird nur das erste Kommando ausgeführt, da pro Bearbeitungstakt nur ein Kommando pro Achse ausgeführt werden kann. Ein entsprechender Fehler der Klasse 1 wird in die Diagnose-Struktur des **Interpreters** eingetragen. Eine Fehlermeldung wird nicht am Ausgang *error* ausgegeben.

Der aktuelle Befehl für den **AchsFB** einer Achse wird in der Ausgangsstruktur *actualValues* des **Interpreters** ausgegeben.

4.2.5 Initialisierung bei Programmstart

Die Initialisierung wird bei einer positiven Flanke des booleschen Eingangs *enable* durchgeführt. Dieser Eingang ist Pegel-gesteuert und muss dauerhaft auf TRUE gesetzt werden, solange der **Interpreter** verwendet werden soll. Der interne Interpreter-Modus wird auf NO_MODE gesetzt, Variablen initialisiert sowie Parameter und Konstanten auf ihre Richtigkeit überprüft. Werden Fehler festgestellt, wird das Fehlerbit *error* gesetzt sowie weitere Informationen in die Diagnose Struktur des **Interpreters** abgelegt. Steht zu diesem Zeitpunkt bereits ein gültiger Interpreter-Modus an, wird er im selben Takt übernommen, d. h., der Interpreter-Modus NO_MODE erscheint nur dann, wenn kein gültiger Interpreter-Modus vorgegeben wird.

4.3 Funktionsbaustein FBLMCIprInterpreter

4.3.1 Schematische Darstellung KOP/FUP

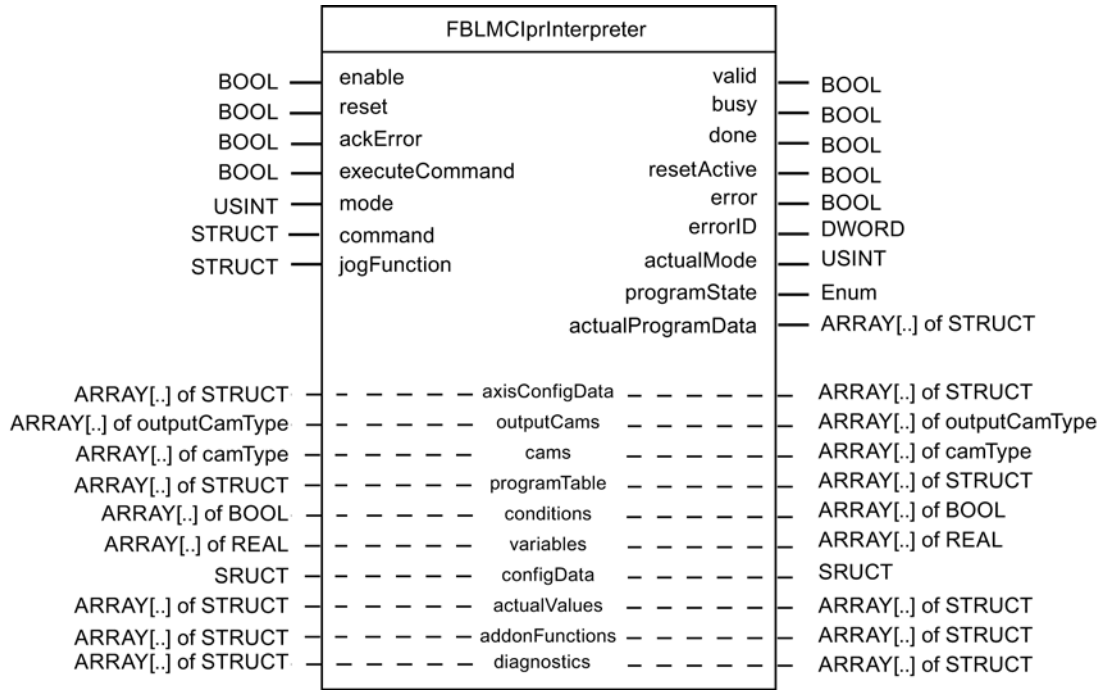


Bild 4-4 Schematische Darstellung KOP/FUP

4.3.2 Eingangs- und Ausgangsparameter

Der Funktionsbaustein **FBLMCIprInterpreter** hat folgende Eingangs- und Ausgangsparameter.

Tabelle 4- 4 Eingangs– und Ausgangsparameter

Bezeichnung	Typ [1]	Datentyp	M/O [2]	Initialwert	Beschreibung
enable	IN	BOOL	M	FALSE	TRUE: Bearbeitung des FB Alle anderen Eingänge werden nur bearbeitet, wenn *enable* auf TRUE steht. Fallende Flanke: Verhalten einstellbar Default: Abbrechen aller Programmabläufe, Anhalten der Achsen, Rücksetzen aller Fehler und TO-Alarme
reset	IN	BOOL	O	FALSE	TRUE: Rücksetzen des FB Verhalten einstellbar: Default: Abbrechen aller Programmabläufe, Anhalten der Achsen, Rücksetzen aller Fehler und TO-Alarme
ackError	IN	BOOL	O	FALSE	TRUE: Quittieren der Fehler des FB und aller anstehenden TO-Alarme
executeCommand	IN	BOOL	M	FALSE	Steigende Flanke: Befehl am Eingang *command* wird ausgeführt
mode	IN	USINT	M	NO_MODE	Vorgegebener Interpreter-Modus
command	IN	sLMCIprCommandType	M	-	Auszuführendes Kommando
jogFunction	IN	sLMCIprJogFunctionDataType	O	-	Daten für Tippen, nur im Service-Mode
valid	OUT	BOOL	-	FALSE	TRUE: Ausgänge des FB sind gültig
busy	OUT	BOOL	-	FALSE	TRUE: ein über *executeCommand* angestoßenes Kommando wird bearbeitet
done	OUT	BOOL	-	FALSE	TRUE: ein über *executeCommand* angestoßenes Kommando ist fertig bearbeitet
resetActive	OUT	BOOL	-	FALSE	TRUE: Die *reset*-Bearbeitung des FB ist aktiv
error	OUT	BOOL	-	FALSE	TRUE: Fehler der Klasse 3 oder 4 ist aufgetreten
errorId	OUT	DWORD	-	16#00000000	Fehlercode des Fehlers der Klasse 3 oder Klasse 4
actualMode	OUT	USINT	-	NO_MODE	Angewählter Interpreter-Modus

Bezeichnung	Typ [1]	Datentyp	M/O [2]	Initialwert	Beschreibung
programState	OUT	eLMCIprProgramStateType	-		RUNNING: mindestens ein Programm ist aktiv NOT_RUNNING: es ist kein Programm in Bearbeitung ABORTED: die Programmbearbeitung wurde durch einen Fehler abgebrochen. Ein Fortsetzen ist nicht möglich STOPPED: alle Sequenzen wurden durch CMD_STOP_PROGRAM oder einen Wechsel der Interpreter-Modi angehalten. SINGLESTEP_ACTIVE: es ist mindestens ein Einzelschritt-Kommando in einer Sequenz aktiv STOP_PROGRAM_RUNNING: ein vom Anwender definiertes Stopp-Programm ist aktiv
actualProgramData	OUT	ARRAY[0..LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES] OF sLMCIprProgramDataType	-	-	Informationen zu den aktuellen Programmsequenzen
axisConfigData	IN/OUT	ARRAY[..] OF sLMCIprAxisConfigDataType	M	-	Konfigurationsdaten zu den Achsen
outputCams	IN/OUT	ARRAY[..] OF outputCamType	M		Liste der verwendeten Nocken
cams	IN/OUT	ARRAY[..] OF camType	M		Liste der verwendeten Kurvenscheiben
programTable	IN/OUT	ARRAY[..] OF sLMCIprProgramType	M	-	Verfahrsatz-Tabelle
conditions	IN/OUT	ARRAY[..] OF BOOL	M	FALSE	Boolesches Array für Bedingungen
variables	IN/OUT	ARRAY[..] OF REAL	M	0.0	Array von REAL-Werten zum Lesen und Schreiben
configData	IN/OUT	sLMCIprInterpreterConfigType	M		Konfigurationsdaten zum Interpreter
actualValues	IN/OUT	ARRAY[..] OF sLMCIprActualValueType	M		Aktuelle Informationen zu den Achsen

Bezeichnung	Typ [1]	Datentyp	M/O [2]	Initialwert	Beschreibung
addOnFunctions	IN/OUT	ARRAY[0..LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES] OF sLMCIprAddonDataType	M		Schnittstelle für die anwenderdefinierten Kommandos
diagnostics	IN/OUT	sLMCIprInterpreterDiagnosticDataType	-	-	Ringspeicher mit Diagnosedaten

[1] Parametertypen: IN = Eingangsparameter, OUT = Ausgangsparameter, IN/OUT = Durchgangsparameter

[2] Parameterart: M = Pflichtparameter, O = Optionaler Parameter

4.3.3 Verhalten bei verschiedenen Eingangs- und Ausgangsparameter des FB

Verhalten der Signale *busy* und *done*

Diese Signale beziehen sich prinzipiell auf die Kommandos, die mit *executeCommand* gestartet wurden, d. h., im Servicemode wird angezeigt, ob ein mit WCD programmiertes Kommando (z. B. eine Positionierung) noch in Bearbeitung ist (*busy* = TRUE) oder abgeschlossen ist (*done* = TRUE). Das Signal *done* bleibt so lange anstehen, wie der Eingang *executeCommand* ansteht. Ist der Eingang bereits auf FALSE, wenn das Kommando fertig ist, steht *done* für genau einen Zyklus an.

Im Programmbetrieb und Einzelschritt-Modus beziehen sich die Signale auf die Programmbearbeitungsbefehle, die alle in einem Zyklus abgearbeitet werden d. h., in diesem Fall wird *busy* nicht ausgegeben.

Zur Anzeige des Programm-Status werden die Ausgänge *programState* und *actualProgramData* verwendet.

Die Daten (Programm-Nr. und Kommando-Nr.) in *actualProgramData* bleiben auch nach Programmende erhalten.

Verhalten bei *reset*

Mit der steigenden Flanke von *reset* wird geprüft, welches Verhalten eingestellt wurde.

Es gibt die folgenden Möglichkeiten:

- STOP_ALL:
  Es werden alle Fehler gelöscht, alle Programme beendet, Gleichlaufverbände aufgehoben, Nocken deaktiviert und alle Bewegungen angehalten.
- USER_DEFINED:
  Es wird das projektierte Stopp-Programm abgearbeitet. Eventuell anstehende Fehler werden quittiert. Ist kein gültiges Programm vorhanden, so wird reagiert, als wäre STOP_ALL projektiert.

Verhalten bei fallender Flanke von *enable*

Mit der fallenden Flanke von *enable* wird geprüft, welches Verhalten eingestellt wurde.

Es gibt die folgenden Möglichkeiten:

- STOP_ALL:
  Es werden alle Fehler gelöscht, alle Programme beendet, Gleichlaufverbände aufgehoben, Nocken deaktiviert und alle Bewegungen angehalten.
- USER_DEFINED:
  Es wird das projektierte Stopp-Programm abgearbeitet. Eventuell anstehende Fehler werden quittiert. Ist kein gültiges Programm vorhanden, so wird reagiert, als wäre STOP_ALL projektiert.

Wurde ein Stopp-Programm definiert, wird dieses ausgeführt. Während der Ausführung bleibt der Ausgang *valid* auf TRUE. Sobald das Stopp-Programm beendet ist, wird zusätzlich die Routine von STOP_ALL abgearbeitet und am Ende der Ausgang *valid* auf FALSE gesetzt.

4.3.4 Struktur für die Kommandoübergabe

Tabelle 4- 5 Elemente von *sLMClprCommandType*

Element	Datentyp	Bedeutung
eCommand	Enum	Kommando Default: CMD_NONE
i32IndexTO	DINT	Index auf das Technologieobjekt, für das das Kommando aktiv ist -1: Kommando gilt für alle TO dieses Typs 0..n: Kommando gilt für ein TO dieses Typs Default: -1
r32Parameter1	REAL	Parameter 1 des Kommandos Default: 0.0
r32Parameter2	REAL	Parameter 2 des Kommandos Default: 0.0
r32Parameter3	REAL	Parameter 3 des Kommandos Default: 0.0
r32Parameter4	REAL	Parameter 4 des Kommandos Default: 0.0
r32Parameter5	REAL	Parameter 5 des Kommandos Default: 0.0
r32Parameter6	REAL	Parameter 6 des Kommandos Default: 0.0
b8Mode	BYTE	Bit 7 = TRUE: Übernahme von Werten aus *UserDefault* FALSE: Übernahme von Werten aus Parametern 1 - 6 Bit 0-5 = TRUE: Übernahme von Werten aus Referenzen auf Array *variables* für die Parameter von 1 bis max. 6 Default: alle Bits = FALSE Bit 7 hat Vorrang vor den Bits 0 bis 5

Element	Datentyp	Bedeutung
u16TimeOut	UINT	Überwachungszeit für Kommando [ms] Default: 0
eNextCommand	Enum	WCD: Weiterschaltung, wenn Kommando beendet IM: Weiterschaltung im nächsten Zyklus IM_NC: direkte Weiterschaltung und Bearbeitung des nächsten Kommandos im selben Zyklus Default: WCD (sofern sinnvoll möglich)

Konfigurationsstruktur achsspezifisch

Tabelle 4- 6 Elemente von *sLMCIprAxisConfigDataType*

Element	Datentyp	Bedeutung
toAxis	AXIS_REF	Achse für Interpreter (Drehzahl-, Positionier-, Gleichlauf-, Bahnachse und Externer Geber) Der tatsächliche TO-Typ wird vom Interpreter ermittelt und im Parameter *actualValues* ausgegeben.
sPosData	sLMCIprDynamicDataType	Konfigurationsdaten für Positionier-Kommando
sPosSuperimposedData	sLMCIprDynamicDataType	Konfigurationsdaten für Kommando Pos-Superimposed-Kommando
sMoveData	sLMCIprDynamicDataType	Konfigurationsdaten für Move-Kommando
sMoveSuperimposedData	sLMCIprDynamicDataType	Konfigurationsdaten für Kommando Move-Superimposed
sJogData	sLMCIprJogAndSyncDataType	Konfigurationsdaten für Jog-Funktion
sStopData	sLMCIprStopDataType	Konfigurationsdaten für Stop-Kommando
sEStopData	sLMCIprStopDataType	Konfigurationsdaten für EStop-Kommando
sGearingData	sLMCIprJogAndSyncDataType	Konfigurationsdaten für Gearing-Kommando
sCammingData	sLMCIprJogAndSyncDataType	Konfigurationsdaten für Camming-Kommando
sPhasingData	sLMCIprDynamicDataType	Konfigurationsdaten für Phasing-Kommando
sConfigData	sLMCIprConfigDataType	Konfigurationsdaten für den Interpreter

Konfigurationsdaten für die Kommandos Pos, Pos-Superimposed, Move und Phasing

Tabelle 4- 7 Elemente für die Strukturen von *Pos*, *Pos-Superimposed*, *Move* und *Phasing*

Element	Datentyp	Bedeutung
r32Acceleration	REAL	Beschleunigung
r32Deceleration	REAL	Verzögerung
r32Jerk	REAL	Ruck

Konfigurationsdaten für die Kommandos Jog, Gearing und Camming

Tabelle 4- 8 Elemente für die Strukturen von *Jog*, *Gearing* und *Camming*

Element	Datentyp	Bedeutung
r32Velocity	REAL	Geschwindigkeit
r32Acceleration	REAL	Beschleunigung
r32Deceleration	REAL	Verzögerung
r32Jerk	REAL	Ruck

Konfigurationsdaten für die Kommandos Stop und EStop

Tabelle 4- 9 Elemente für die Strukturen von *Stop* und *EStop*

Element	Datentyp	Bedeutung
r32Deceleration	REAL	Verzögerung
r32Jerk	REAL	Ruck

Achs-Konfigurationsdaten für den Interpreter (*sConfigData*)

Hinweis

Nähere Erläuterungen zu den einzelnen Struktur-Elementen, siehe Dokument *Applikationshandbuch SIMOTION Achs-Funktionsbaustein*.

Tabelle 4- 10 Elemente von *sConfigData*

Element	Datentyp	Bedeutung
eFollowingErrorReaction	eLMCIprFollowingErrorReactionType	Einstellung der Schleppfehlerüberwachung im Achs-FB
eEncoderIdentification	eLMCBasicEncoderIdentificationType	Einstellung, ob die Geberindentifizierung im Achs-FB automatisch durchgeführt werden soll AUTOMATIC: Achs-FB ermittelt Gebertyp ABSOLUTE_ENCODER, INCREMENTAL_ENCODER: Vorgabe durch den Anwender
r32MinPositionTolerance	REAL	Minimale Toleranz für die Position: Einstellwerte für die Schleppfehlerüberwachung
r32MaxPositionTolerance	REAL	Minimale Toleranz für die Position: Einstellwerte für die Schleppfehlerüberwachung
r32MinVelocity	REAL	Minimale Toleranz für die Geschwindigkeit: Einstellwerte für die Schleppfehlerüberwachung
r32MaxVelocity	REAL	Minimale Toleranz für die Geschwindigkeit: Einstellwerte für die Schleppfehlerüberwachung
u32BrakeCloseTime	UDINT	Zeit zum Schließen der Bremse

Element	Datentyp	Bedeutung
boMotorWithoutBrake	BOOL	TRUE: Motor ohne Bremse
boReplaceStopPossible	BOOL	TRUE: ein Stopp-Kommando kann durch ein Bewegungskommando abgelöst werden

Programmtabelle

Tabelle 4- 11 Element von *asCommand*

Element	Datentyp	Bedeutung
asCommand	ARRAY[0..LMCIPR_NUMBER_OF_COMMANDS-1] OF sLMCIPRCommandType	Verfahrsatz-Programm

Konfigurationsdaten für den Interpreter

Tabelle 4- 12 Elemente von *configData*

Element	Datentyp	Bedeutung
r32ServiceVelocityOverride	REAL	Override für den Servicemode
r32ProgramVelocityOverride	REAL	Override für den Programmbetrieb
sStopProgram	sLMCIprProgramDefinitionType	Programm bzw. Kommando für *reset* und fallende Flanke *enable* Programm-Nr = -1: kein Stopp-Programm
eStopReaction	eLMCIprStopReactionType	Reaktion auf *reset* und fallende Flanke von *enable*: STOP_ALL: Anhalten aller aktiver Sequenzen, Deaktivieren aller TO (Default) USER_DEFINED: Ausführen des unter *sStopProgram* definierten Stopp-Programms
sStopCmdProgram	sLMCIprProgramDefinitionType	Programm bzw. Kommando zum Anhalten aller aktiven Sequenzen. Programm-Nr = -1: kein Kommando-Stopp-Programm
eStopCmdReaction	eLMCIprStopCmdReactionType	Reaktion bei CMD_STOP_PROGRAM NONE: Anhalten aller aktiven Sequenzen USER_DEFINED: zusätzliche Abarbeitung eines spez. Stopp-Programms

Element	Datentyp	Bedeutung
sErrorProgram	sLMCIprProgramDefinitionType	Programm bzw. Kommando als Fehlerreaktion Programm-Nr -1: kein Programm als Fehlerreaktion
eErrorReaction	eLMCIprErrorReactionType	Reaktion auf Fehler im Interpreter: NONE: keine weitere Reaktion STOP_ALL_SEQUENCES: Beenden aller aktiven parallelen Sequenzen (Default) USER_DEFINED: Ausführen des unter *sErrorProgram* definierten Fehlerprogramms USER_DEFINED_AND_STOP_ALL_SEQUENCES: Beenden aller aktiven parallelen Sequenzen und Ausführen des unter *sErrorProgram* definierten Fehlerprogramms

Behandlung der Override-Werte:

Die beiden Werte werden auf Änderung überprüft. Wurde eine Änderung festgestellt, wird diese als Override für den Achs-FB eingetragen. Für alle Achsen gilt derselbe Wert, jedoch können für **Servicemode** und **Programmbetrieb** unterschiedliche Werte vorgegeben werden.

Der Achs-FB schreibt den Wert des Override auf die Systemvariable *override.velocity*.

Struktur für Stopp- und Fehler-Programm

Tabelle 4- 13 Elemente von *sLMCIprProgramDefinitionType*

Element	Datentyp	Bedeutung
i32ProgramNo	DINT	Program-Nr. Default: -1
i32CommandNo	DINT	Schritt-Nr. Default: -1

Struktur für Istwerte

Tabelle 4- 14 Elemente von *sLMCIprActualValuesTypeElemente*

Element	Datentyp	Bedeutung
b32ActCommand	DWORD	Aktuelles Kommando für **Achs-FB** der Achsen
r32FollowingErrorPercentage	REAL	Ausgabe des **Achs-FB** zur Schleppfehlerüberwachung
u8ActualTOType	USINT	Achs-Typ (Drehzahl-(1), Positionier-(2), Gleichlauf-Achse(3), oder externer Geber(8))
boAxisEnabled	BOOL	TRUE: Achse freigegeben FALSE: Achse nicht freigegeben
boHomed	BOOL	TRUE: Achse referenziert FALSE: Achse nicht referenziert
boInSync	BOOL	TRUE: Achse synchron FALSE: Achse nicht synchron
boInPos	BOOL	TRUE: Achse im Positionierfenster FALSE: Achse nicht im Positionierfenster

Element	Datentyp	Bedeutung
boStandStill	BOOL	TRUE: Achse steht still FALSE: Achse in Bewegung
boInVelocity	BOOL	TRUE: Sollgeschwindigkeit bei Move-Befehl erreicht FALSE: außerhalb Sollgeschwindigkeit
boEStopActive	BOOL	TRUE: Notstopp an der Achse aktiv FALSE: kein Notstopp an der Achse aktiv
boError	BOOL	TRUE: Fehler am Achs-FB steht an
boTOAlarmActive	BOOL	TRUE: an der Achse steht ein schwerwiegender Alarm an

Hinweis

Die Status-Signale, die für einen bestimmten Achstyp nicht relevant sind, z. B. *boInSync* für Positionierachsen werden vom Interpreter fest auf TRUE gesetzt. Das erleichtert die Abfrage eines Gesamtstatus einer Maschine, da dann der Achstyp dort nicht berücksichtigt werden muss.

Struktur für Programm-Informationen

Tabelle 4- 15 Elemente von *sLMCIprProgramDataType*

Element	Datentyp	Bedeutung
i32ActualProgram	DINT	Aktuelle Programm-Nummer Initialisierung: -1
i32ActualCommand	DINT	Aktuelle Kommando-Nummer Initialisierung: -1
sState	sLMCIprProgramDataStateType	*siehe unten*
boSingleStepActive	BOOL	Nur für Einzelschrittbearbeitung: TRUE: Kommando ist aktiv FALSE: Kommando ist fertig bearbeitet

Tabelle 4- 16 Elemente von *sLMCIprProgramDataStateType*

Element	Datentyp	Bedeutung
boActive	BOOL	TRUE: Bearbeitung ist aktiv FALSE: keine Bearbeitung
i32Program	DINT	Programm-Nummer Initialisierung: -1
i32Command	DINT	Kommando-Nummer Initialisierung: -1

Hinweis

Die Variablen *actualProgramData [x].i32 ActualProgram* und *actualProgramData [x].i32 ActualCommand* werden auf -1 gesetzt wenn kein Programm aktiv ist.

Die Variablen unter *actualProgramData [x].state* zeigen den letzten Verfahrsatz eines Programms an.

Struktur für anwenderdefinierte Kommandos

Tabelle 4- 17 Elemente von *sLMCIprAddonDataType*

Element	Datentyp	Bedeutung
sAddonCommand	sLMCIprCommandType	Kommando
sModuleResult	sLMCIprModuleResultType	Zusatz-Schnittstelle für interne Weiterverarbeitung
b32ErrorId	DWORD	Fehler-Kennung
i32ErrorAddVal	DINT	Zusatz-Information zum Fehler, wird vom Interpreter in *i32AdditionalValue4* in die Diagnose-Struktur des Interpreters eingetragen
u8ErrorClass	USINT	Fehler-Klasse
boNewProgram	BOOL	Erstes Kommando eines neuen Programms bzw. Sequenz
boRunning	BOOL	TRUE: Kommando ist in Arbeit

Diagnosestruktur für den Ringspeicher

Tabelle 4- 18 Elemente von *sLMCIprInterpreterDiagnosticDataType*

Element	Datentyp	Bedeutung
u16ErrorClass1	UINT	Anzahl der Fehler der Fehlerklasse 1
u16ErrorClass2	UINT	Anzahl der Fehler der Fehlerklasse 2
u16ErrorClass3	UINT	Anzahl der Fehler der Fehlerklasse 3
u16ErrorClass4	UINT	Anzahl der Fehler der Fehlerklasse 4
sFirstErrorHighestPriority	sLMCIprDiagnosticDataType	Struktur für Fehler höchster Priorität
asErrorBuffer	ARRAY[0..LMCIPR_SIZE_OF_ERRORBUFFER-1] OF sLMCIprDiagnosticDataType	Ringspeicher für alle bisher aufgetretenen Fehler
u16Index	UINT	Zeiger auf den letzten Eintrag im Ringspeicher der Fehler
asCommandTrace	ARRAY[0..LMCIPR_SIZE_OF_COMMANDTRACE-1] OF sLMCIprCommandTraceType	Kommando-Trace
u16TracePtr	UINT	Zeiger auf den letzten Eintrag im Kommando-Trace

Tabelle 4- 19 Elemente von *sLMCIprDiagnosticDataType*

Element	Datentyp	Bedeutung	
u8ErrorClass	USINT	Fehlerklasse	
u8TOType	USINT	Typ des Fehler verursachenden TO: wird nicht im **Interpreter** verwendet	
i16TONumber	INT	Arrayindex des Fehler verursachenden TO: wird nur bei Fehlern der Klasse 4 verwendet Inhalt bei Fehlerklasse 1 bis 3: -1	
b32ErrorCode	DWORD	Fehlercode, derselbe Wert wie in *errorId* beschreibt die Fehlerursache (Konstante in **cPublic**)	
i32AdditionalValue1	DINT	Sequenz, in welcher der Fehler aufgetreten ist	Nur bei Fehlern der Klasse 3
i32AdditionalValue2	DINT	Verfahrsatz-Programm, in welchem der Fehler aufgetreten ist	
i32AdditionalValue3	DINT	Verfahrsatz, in welchem der Fehler aufgetreten ist	
i32AdditionalValue4	DINT	Fehlerhafter Parameter	
b32AdditionalValue5	DWORD	Nicht verwendet	
r32AdditionalValue6	REAL	Nicht verwendet	
dtTimeStamp	DATE_AND_TIME	Zeitstempel für den Eintrag in die Diagnose-Struktur des **Interpreters**	

Struktur für den Kommando-Trace

Tabelle 4- 20 Elemente von *sLMCIprCommandTraceType*

Element	Datentyp	Bedeutung
sCommand	sLMCIprCommandType	Kommando
dtTimeStamp	DT	Zeitstempel
u8Sequence	USINT	Nummer der Sequenz
i32Program	DINT	Programm-Nummer
i32Command	DINT	Kommando-Nummer

4.4 FunktionFCLMCIprSetCMD

4.4.1 Allgemeines zur Funktion FCLMCIprSetCMD

Mit der Funktion **FCLMCIprSetCMD** kann der Anwender Verfahrsatz-Programme programmieren.

4.4.2 Schematische Darstellung KOP/FUP

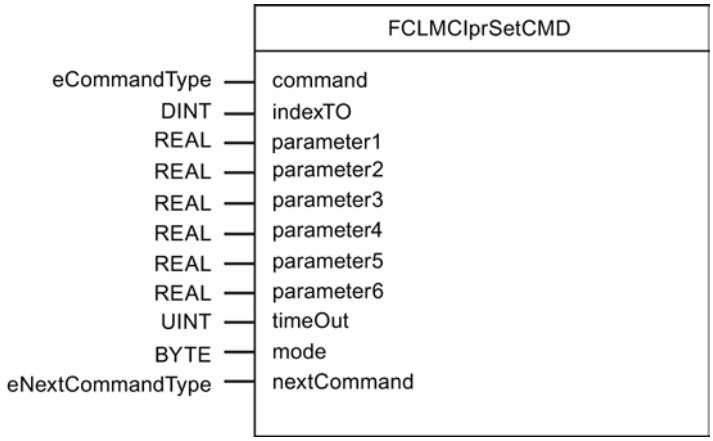


Bild 4-5 KOP-Darstellung der Funktion FCLMCIprSetCMD

4.4.3 Eingangsparameter für die Funktion

Die Funktion **FCLMCIprSetCMD** hat folgende Eingangsparameter:

Tabelle 4- 21 Beschreibung der Parameter der Funktion FCLMCIprSetCMD

Bezeichnung	Typ [1]	Datentyp	M/O [2]	Beschreibung
command	IN	eCommandType	M	Kommando
indexTO	IN	DINT	M	Index auf Array TO
parameter1	IN	REAL	M	Zusatzparameter 1
parameter2	IN	REAL	M	Zusatzparameter 2
parameter3	IN	REAL	M	Zusatzparameter 3
parameter4	IN	REAL	M	Zusatzparameter 4
parameter5	IN	REAL	M	Zusatzparameter 5
parameter6	IN	REAL	M	Zusatzparameter 6
timeOut	IN	UINT	M	Zeitüberwachung im ms
mode	IN	BYTE	M	Angabe, welche Parameter indirekt vorgegeben werden
nextCommand	IN	eNextCommandType	M	Weiterschaltbedingung

4.5 Interner Kommando-Trace

Funktionalität

Die Applikation **Interpreter** kann alle Kommandos, die der Reihe nach im Interpreter-Modus Programmbetrieb ausgeführt wurden, in einen Kommando-Trace schreiben. Diese Funktionalität wird über die Konstante LMCIPR_COMMANDOTRACE_ACTIVE in der Unit **cPublic** aktiviert. Es werden folgende Daten in den Kommando-Trace geschrieben:

- Programm-Nummer
- Kommando-Nummer
- Sequenz-Nummer
- Kommando
- Zeitstempel

Bei indirekter Adressierung werden die tatsächlich verwendeten Werte in den Kommando-Trace gespeichert. Das gleiche gilt für Default-Einstellungen, wenn falsche Vorgaben im Kommando projektiert werden. Auch in diesem Fall wird die tatsächlich verwendete Projektierung in den Kommando-Trace eingetragen.

Im Service-Mode wird der Kommando-Trace nicht geschrieben.

Der Kommando-Trace liegt in der Diagnosestruktur.

Alarm- und Fehlermeldungen

5

5.1 Fehlerhandling des Interpreters

Der **Interpreter** hat Fehler unterschiedlicher Fehlerklassen und Fehlerreaktionen. An den Ausgängen *error* und *errorId* werden nur die Fehler der Klasse 3 und 4 angezeigt, die zum Abbruch des Programms führen können. Die Fehler der Klassen 1 und 2 führen nicht zu einem Abbruch des Programms und werden daher nicht an *error* und *errorId* angezeigt.

Alle Fehler werden zusammen mit Zusatz-Informationen in einen Ringspeicher eingetragen, damit diese für weiterführende Diagnosen zur Verfügung stehen. Für den Fehler der jeweils höchsten Priorität wird noch eine zusätzliche Zeile für den ersten Eintrag mitgeführt.

Der Ringspeicher wird nicht gelöscht, sondern ständig fortgeschrieben. Sobald der Ringspeicher gefüllt ist, werden die bereits bestehenden Einträge durch neue Fehlermeldungen überschrieben.

Fehler der Klasse 3 und 4 müssen im **Programmbetrieb** über *ackError* quittiert werden, bevor wieder eine Bearbeitung gestartet werden kann.

Hinweis

Der Fehler, der anzeigt, dass an einer Achse ein TO-Alarm aktiv ist wird nicht an *error / errorId* angezeigt, sondern nur in der Diagnosestruktur eingetragen.

Diagnosestruktur des Interpreters

Tabelle 5- 1 Elemente von **sLMCIprInterpreterDiagnosticDataType**

Element	Datentyp	Bedeutung
u16ErrorClass1	UINT	
u16ErrorClass2	UINT	
u16ErrorClass3	UINT	
u16ErrorClass4	UINT	
sFirstErrorHighestPriority	sLMCIprDiagnosticDataType	Eintrag des ersten aufgetretenen Fehlers der jeweils höchsten Priorität
u16Index	UINT	Zeiger auf den letzten Eintrag im Ringspeicher
asErrorBuffer	ARRAY of sLMCIprDiagnosticDataType	Ringspeicher für die aufgetretenen Fehler
asCommandTrace	ARRAY[0..LMCIPR_SIZE_OF_COMMANDTRACE-1] OF sLMCIprCommandTraceType	Kommando-Trace

Die Informationen in *u16ErrorClass* und *sFirstErrorHighestPriority* werden mit *ackError*, *reset* und fallender Flanke *enable* gelöscht.

5.2 Verhalten im Fehlerfall

Schreiben in die Diagnose-Struktur des Interpreters

Schwerwiegende Fehler (der Klassen 3 und 4), die während der Bearbeitung auftreten, werden an den Ausgängen des **Interpreters** *error* und *errorId* gemeldet und in die Diagnose-Struktur des **Interpreters** geschrieben. Bei entsprechender Projektierung wird eine Fehlersequenz aufgerufen. Darüber hinaus kann der Anwender einstellen, ob alle aktiven, parallelen Sequenzen abgebrochen werden oder weiter bearbeitet werden sollen. Die Fehlersequenz muss vom Anwender mit den passenden Maschinenreaktionen programmiert werden.

Bei Fehlern der Klassen 3 und 4 des **Interpreters** werden zwei verschiedene Fälle unterschieden:

- Fehler bei der Vorgabe eines Kommandos (falsche Syntax)
- Fehler bei der Ausführung eines Kommandos (Fehler des intern genutzten **Achs_FB**)

Ein Fehler bei der Vorgabe eines Kommandos beendet die aktuelle Sequenz, da das fehlerhafte Kommando bekannt ist. Alle anderen Sequenzen werden weiter bearbeitet. Über ein Enumerator in der Konfigurations-Struktur des **Interpreters** kann zusätzlich angegeben werden, ob die anderen Sequenzen gestoppt werden sollen oder nicht. Ansonsten erfolgt keine weitere Reaktion d. h., gestartete Bewegungen werden nicht angehalten, Gleichlaufverbindungen bleiben bestehen und Nocken bleiben aktiviert.

Darüber hinaus kann eingestellt werden, ob ein spezifisches Programm abgearbeitet werden soll. In diesem Fall müssen alle gewünschten Aktivitäten, z. B. Stoppen von Sequenzen, Deaktivieren von Bewegungen, in diesem Programm durchgeführt werden.

Bei Fehlern, die der intern genutzte **Achs-FB** meldet, liegen keine direkten Zuordnungen zur Sequenz mehr vor. Deshalb kann über die Projektierung (Enumerator) nur definiert werden, ob alle Sequenzen weiter bearbeitet werden sollen oder ob alle Sequenzen angehalten werden sollen. Ansonsten ist das Verhalten identisch zum Fall eines fehlerhaften Kommandos.

In beiden Fällen kann der Anwender zusätzlich ein Programm projektieren, das im Fehlerfall abgearbeitet wird. Für beide Fehlerarten wird dasselbe Programm verwendet. Wird kein Programm angegeben (Programm-Nummer = -1) erfolgt keine weitere Reaktion.

Konsequenz:

Für ein Stopp-Programm bzw. Fehler-Programm muss vom Anwender ein eigenes Programm programmiert werden. Dieses Programm wird vom **Interpreter** in der betreffenden Situation automatisch gestartet.

5.3 Fehlerklassen

Auflistung der Fehlerklassen

Folgende Fehlermeldungen sind in der Applikation **Interpreter** möglich.

Tabelle 5- 2 Mögliche Fehlerklassen

Bezeichnung	Fehler-klasse	Beschreibung
LMCIPR_FBINTERPRETER_ERRORCLASS_WARNING	1	Warnung
LMCIPR_FBINTERPRETER_ERRORCLASS_ERROR1	2	Fehler in einem Kommando Das fehlerhafte Kommando wird mit Default-Werten abgearbeitet.
LMCIPR_FBINTERPRETER_ERRORCLASS_ERROR2	3	Schwerwiegender Fehler in einem Kommando Die aktuelle Sequenz wird abgebrochen.
LMCIPR_FBINTERPRETER_ERRORCLASS_AXIS_ERROR	4	Fehler an Achsen Reaktion je nach Einstellung

5.4 Fehlermeldungen im Interpreter

Auflistung der Fehlermeldungen

Folgende Fehlermeldungen sind in der Applikation **Interpreter** möglich.

Tabelle 5- 3 Mögliche Fehlermeldungen

Fehler-nummer ([hex])	Fehlername	Beschreibung
5000	LMCIPR_FBINTERPRETER_ERROR_WRONG_MODE	Der vorgegebene Modus ist ungültig
5001	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_AXIS	Die vorgegebene Achs-Nummer existiert nicht
5002	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_CAM	Die vorgegebene Nummer der Kurvenscheibe existiert nicht
5003	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_OUTPUT_CAM	Die vorgegebene Nummer des Nocken existiert nicht
5004	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_CONDITIONS	Die vorgegebene Nummer einer Bedingung existiert nicht
5005	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_VARIABLES	Die vorgegebene Nummer einer Variablen existiert nicht
5006	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_PROGRAM	Die vorgegebene Programm-Nummer existiert nicht

Fehler-nummer ([hex])	Fehlername	Beschreibung
5007	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_COMMAND	Die vorgegebene Kommando-Nummer existiert nicht
5008	LMCIPR_FBINTERPRETER_ERROR_WRONG_VALUE_AXIS_ARRAY	Das übergebene Achs-Array ist zu groß bzw. passt nicht zur Größe der Konstanten in der Unit **cPublic**
5009	LMCIPR_FBINTERPRETER_ERROR_WRONG_NO_MASTER	Es wurde ein falscher Index auf die Leitachse programmiert
500A	LMCIPR_FBINTERPRETER_ERROR_WRONG_COMMAND	Es wurde ein unzulässiges Kommando gestartet
500B	LMCIPR_FBINTERPRETER_ERROR_NO_SINGLESTEP	Der Interpreter-Modus Einzelschritt ist nicht aktiv
500C	LMCIPR_FBINTERPRETER_ERROR_NO_POSAXIS	Eine benötigte Positionierachse wurde nicht konfiguriert
500D	LMCIPR_FBINTERPRETER_ERROR_NO_TO_CONFIGURED	Ein TO wurde nicht konfiguriert, z. B. TO#NIL im Achs-Array
500E	LMCIPR_FBINTERPRETER_ERROR_WRONG_TYPE_TO	Es wurde ein falscher TO-Typ konfiguriert
500F	LMCIPR_FBINTERPRETER_ERROR_NO_CAM	Es wurde keine Kurvenscheibe angegeben
5010	LMCIPR_FBINTERPRETER_ERROR_NO_TO_FOR_JOG	Ein TO für Tippen wurde nicht konfiguriert
5011	LMCIPR_FBINTERPRETER_ERROR_WRONG_VALUE	Es wurde ein falscher Wert in einem Kommando-Parameter programmiert
5012	LMCIPR_FBINTERPRETER_ERROR_WRONG_VALUE_VELOCITY	Es wurde ein falscher Wert für die projektierte Geschwindigkeit programmiert
5013	LMCIPR_FBINTERPRETER_ERROR_WRONG_VALUE_GEARING	Es wurde ein unzulässsiger Getriebefaktor programmiert
5014	LMCIPR_FBINTERPRETER_ERROR_WRONG_VALUE_VARIABLE	Indirekte Programmierung und in einer Variablen steht ein unzulässiger Wert
5020	LMCIPR_FBINTERPRETER_ERROR_TIME_MONITORING	Die Überwachungszeit ist abgelaufen
5021	LMCIPR_FBINTERPRETER_ERROR_ACCEL_LIMITED	Der Wert für interne Beschleunigung wurde begrenzt
5022	LMCIPR_FBINTERPRETER_ERROR_NEXT_COMMAND_WRONG	Es wurde ein falscher Wert im Parameter *eNextCommand* programmiert
5024	LMCIPR_FBINTERPRETER_ERROR_NO_NEW_SEQUENCE_POSSIBLE	Es ist keine neue Sequenz mehr möglich
5025	LMCIPR_FBINTERPRETER_ERROR_NO_LOOP_POSSIBLE	Es ist keine geschachtelte Schleife mehr möglich

Fehler-nummer ([hex])	Fehlername	Beschreibung
5026	LMCIPR_FBINTERPRETER_ADDON_FUNCTION_NOT_ACTIVE	Es ist kein Programm für anwenderdefinierte Kommandos vorhanden
5027	LMCIPR_FBINTERPRETER_ERROR_DOUBLE_COMMAND_SKIPPED	Es wurden zwei Kommandos für dieselbe Achse programmiert, aber es wird nur eines ausgeführt
5028	LMCIPR_FBINTERPRETER_ERROR_JUMP_TO_SAME_COMMAND	Es wurde ein Sprung auf sich selbst programmiert
5029	LMCIPR_FBINTERPRETER_ERROR_TOALARM_ACTIVE	Der Achs-FB meldet einen TO-Alarm für mindestens eine Achse
5030	LMCIPR_FBINTERPRETER_ERROR_EMPTY_PROGRAM	Es wurde versucht, ein leeres Programm (CMD_NONE) aufzurufen
5031	LMCIPR_FBINTERPRETER_ERROR_NO_LOOP_ACTIVE	CMD_END_LOOP und kein LOOP aktiv
5032	LMCIPR_FBINTERPRETER_ERROR_SEQUENCE_NOT_EXISTS	CMD_STOP_SEQUENCE und keine Sequenz aktiv
5033	LMCIPR_FBINTERPRETER_ERROR_AXIS_NO	Die Array-Länge von *actualValues* und von den Achs-Konfigurationsdaten passt nicht zusammen
5034	LMCIPR_FBINTERPRETER_ERROR_COMMANDABORTED	Der Achs-FB meldet *command aborted*
5100	LMCIPR_FBINTERPRETER_RESET_EXECUTED	Das Signal *reset* wurde ausgeführt
5101	LMCIPR_FBINTERPRETER_ENABLE_SWITCHED_OFF	Es wurde eine fallende Flanke für das Signal *enable* programmiert

5.5 Diagnose

Im Fehlerfall wird der boolesche Ausgang *error* auf TRUE gesetzt und über *errorId* ein Fehlercode ausgegeben. Weitere Informationen über den Fehler werden in der Diagnosestruktur *diagnostics* gespeichert.

Zusätzlich zu den Fehlerinformationen wird in der Diagnosestruktur eine Fehlerklasse über das Element *u8ErrorClass* ausgegeben. Diese dient der Bewertung des Fehlers durch eine zentrale Stelle im Maschinenprogramm. Informationen über die Fehler-ID finden Sie in der Unit **cPublic** der Bibliotheken **LMCIpr** sowie **LMCBasic**.

Fehlerklassen im Interpreter

Tabelle 5- 4 Fehlerklassen beim Interpreter für Fehler bei Technologieobjekten

Fehlerklasse	Bedeutung	Fehlerursache
0	Hinweis	Nicht verwendet
1	Warnung	Syntaxfehler im Kommando, der vom **Interpreter** behoben wird, z. B. Vorhaltezeit für Nocken nicht lesbar bei indirekter Programmierung
2	Fehler beeinträchtigt Funktionsweise	Syntaxfehler im Befehl, der dazu führt, dass das Kommando ignoriert wird. Diese Fehlerklasse wird nur im **Servicemode** verwendet. Beispiel: CMD_MOVE und Parameter 3 (Richtung) = 4
3	kritischer Fehler	Fehler, der zum Abbruch des Programms der aktuellen Sequenz führt
4	Fehler des Achs-FB	Während der Bearbeitung ist ein Fehler im **Achs-FB** aufgetreten, z. B. CMD_MOVE und die Achse hat keine Reglerfreigabe
5 - 7	reserviert	

Hinweis

Die Interpreterbibliothek verwendet intern einen FB zum Eintrag von Fehlern in die Diagnosestruktur. Dieser FB kann bei Bedarf auch vom Anwender verwendet werden.

Schreiben von Kommandos

Siehe dazu Abschnitt Interner Kommando-Trace (Seite 53).

Anwendungsbeispiel

6

6.1 Beschreibung des Anwendungsbeispiels

Beschreibung zum Code-Beispiel

In dem untenstehenden Code-Beispiel finden Sie eine kompilierbare ST-Quelle mit einem Aufruf des **Interpreters**. Durch *USELIB LMCIpr* werden dem Anwender der Funktionsbaustein **Interpreter** und die zugehörigen Datentypen zur Verfügung gestellt. Im Interface Abschnitt wurden die Variablen angelegt, die dem Aufruf des FB übergeben werden müssen. Diese Variablen können vom Anwender bzw. von einem HMI verwendet werden. Das Programm **pInitialize** ist für die Verwendung in der *StartupTask* vorgesehen. Hier werden Dynamikvorgaben für die Achsen und die Sequenzen des **Interpreters** vorgegeben. In diesem Beispielablauf wurden die folgenden Kommandos projektiert:

Die Achsen werden

- freigegeben
- referenziert
- absolut positioniert

Für eine einfache Vorgabe der Kommandos gibt es die Funktion **FCLMCIprSetCMD**, mit der der Anwender laut dem beigefügten Schema die Parameter ausfüllt und danach das Kommando in die Programmstruktur einträgt. Für eine detaillierte Beschreibung der Kommandos und Parameter schlagen Sie im Dokument *Listenhandbuch Interpreter* nach. Im Programm **pAutomatic** befindet sich der Aufruf des FB Interpreter. Dieses Programm ist für Verwendung in der *BackgroundTask* vorgesehen.

6.2 Code des Anwendungsbeispiels

Tabelle 6- 1 Code in ST

```
INTERFACE
    USEPACKAGE cam;
    USELIB LMCIpr;

    VAR_GLOBAL
        gboEnable         : BOOL;
        gboReset          : BOOL;
        gboAckErrors      : BOOL;
        gboExecute        : BOOL;
        gu8SetPointMode   : USINT;
        gsActCmd          : sLMCIprCommandType;
        gsActJogData      : sLMCIprJogFunctionDataType;
        gasAxisConfigData : ARRAY[0..9] OF sLMCIprAxisConfigDataType;
        gaOutputCams      : ARRAY[0..5] OF outputCamType;
        gaCams            : ARRAY[0..3] OF camType;
        gaActProgramTable : ARRAY[0..9] OF sLMCIprProgramType;
        gaConditions      : ARRAY[0..599] OF BOOL;
        gaVariables       : ARRAY[0..599] OF REAL;
        gsConfigData      : sLMCIprInterpreterConfigType;
        gasActualValues   : ARRAY[0..0] OF sLMCIprActualValueType;
        gasAddonFunctions  : ARRAY[0..LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES] OF
sLMCIprAddonDataType;
        gaActProgramInfo  : ARRAY[0..LMCIPR_NUMBER_OF_PARALLEL_SEQUENCES] OF
                            sLMCIprProgramDataType;
        gu8ActMode        : USINT;
    END_VAR

    PROGRAM pInitialize;
    PROGRAM pAutomatic;
END_INTERFACE

IMPLEMENTATION
    VAR_GLOBAL
        FBInterpreter : FBLMCIprInterpreter
    END_VAR

    PROGRAM pInitialize
        //configuration values
        gasAxisConfigData[0].toAxis := A0;
        gasAxisConfigData[1].toAxis := A1;

        gasAxisConfigData[0].sPosData.r32Acceleration := 5000.0;
        gasAxisConfigData[1].sPosData.r32Acceleration := 5000.0;

        gasAxisConfigData[0].sPosData.r32Deceleration := 6000.0;
        gasAxisConfigData[1].sPosData.r32Deceleration := 6000.0;
```

```
        //programs
             //                TO-Index,P1 ,P2   ,P3 ,P4 ,P5 ,P6 ,mode ,tMon ,next
             //    enable all axes
        gaActProgramTable[0].asCommand[0] :=
        FCLMCIprSetCMD(CMD_POWER_ON ,-1 ,0  ,0    ,0  ,0  ,0  ,0  ,0    ,0    ,WCD);
             //    A0 direct homing
        gaActProgramTable[0].asCommand[1] :=
        FCLMCIprSetCMD(CMD_HOMING   ,0  ,0  ,2    ,0  ,0  ,0  ,0  ,0    ,0    ,WCD);
             //    A1 direct homing
        gaActProgramTable[0].asCommand[2] :=
        FCLMCIprSetCMD(CMD_HOMING   ,1  ,0  ,2    ,0  ,0  ,0  ,0  ,0    ,0    ,WCD);
             //    A0 position positive
        gaActProgramTable[0].asCommand[3] :=
        FCLMCIprSetCMD(CMD_POS_ABS  ,0 ,100 ,1000 ,0  ,0  ,0  ,0  ,0    ,0    ,WCD);
             //    A1 position positive
        gaActProgramTable[0].asCommand[4] :=
        FCLMCIprSetCMD(CMD_POS_ABS  ,1 ,10  ,1000 ,0  ,0  ,0  ,0  ,0    ,0    ,WCD);
             //end of program
        gaActProgramTable[0].asCommand[5] :=
        FCLMCIprSetCMD(CMD_NONE     ,0  ,0  ,0    ,0  ,0  ,0  ,0  ,0    ,0    ,WCD);
    END_PROGRAM

    PROGRAM pAutomatic
        //call interpreter
        FBInterpreter(
             enable := gboEnable
            ,reset := gboReset
            ,ackError := gboAckError
            ,executeCommand := gboExecute
            ,mode := gu8SetPointMode
            ,command := gsActCmd
            ,jogFunction := gsActJogData
            ,axisConfigData := gasAxisConfigData
            ,outputCams := gaOutputCams
            ,cams := gaCams
            ,programTable := gaActProgramTable
            ,conditions := gaConditions
            ,variables := gaVariables
            ,configData := gsConfigData
            ,actualValues := gasActualValues
            ,addOnFunctions := gasAddonFunctions
            ,actualProgramData => gaActProgramInfo
            ,actualMode => gu8ActMode
        );
    END_PROGRAM
END_IMPLEMENTATION
```

Kontakt

A

A.1 Ansprechpartner

Siemens AG
Industry Sector
I DT MC PMA APC
Frauenauracher Straße 80
D - 91056 Erlangen
Fax.: +49 9131 98 1297
mailto: tech.team.motioncontrol@siemens.com

A.2 Internetadressen

Weitere Informationen zu den verschiedenen Themen finden Sie auf folgenden Internetseiten.

Siehe auch

SIMOTION (www.siemens.com/simotion)

SINAMICS (www.siemens.com/sinamics)

MotionControl/Applikationszentrum (www.siemens.com/motioncontrol/apc)

Verpackung (www.siemens.com/packaging)