SIEMENS

SINUMERIK

SINUMERIK 828D
Easy XML

Programmierhandbuch

Gültig für:

CNC-Software Version 4.95

07/2021
6FC5398-3EP40-0AA0

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
Digital Industries
Postfach 48 48
90026 NÜRNBERG
DEUTSCHLAND

Dokumentbestellnummer: 6FC5398-3EP40-0AA0
Ⓟ 06/2021 Änderungen vorbehalten

Inhaltsverzeichnis

Einleitung 1

1.1 Über SINUMERIK

Von einfachen CNC-Standardmaschinen über standardisierte Maschinen, bis hin zu modularen Premium-Maschinenkonzepten – die CNC-Steuerungen SINUMERIK bieten für jedes Maschinenkonzept die passende Lösung. Ob Einzelteil- oder Massenfertigung, einfache oder komplexe Werkstücke – SINUMERIK ist die hochproduktive Automatisierungslösung durchgängig für alle Fertigungsbereiche – vom Muster- und Werkzeugbau über den Formenbau bis zur Großserienfertigung.

Für weitere Informationen besuchen Sie die Internetseite zu SINUMERIK (https://www.siemens.de/sinumerik).

1.2 Über diese Dokumentation

Zielgruppe

Die vorliegende Druckschrift wendet sich an:

- Programmierer
- Projekteure

Nutzen

Das Programmierhandbuch befähigt die Zielgruppe, kunden- und applikationsspezifische Bedienoberflächen mit XML-basierenden Skriptelementen zu entwerfen.

Standardumfang

In der vorliegenden Dokumentation ist die Funktionalität des Standardumfangs beschrieben. Dieser kann vom Umfang der Funktionalitäten des gelieferten Systems abweichen. Die Funktionalitäten des gelieferten Systems entnehmen Sie ausschließlich den Bestellunterlagen.

Im System können weitere, in dieser Dokumentation nicht erläuterte Funktionen ablauffähig sein. Es besteht jedoch kein Anspruch auf diese Funktionen bei der Neulieferung bzw. im Servicefall.

Diese Dokumentation kann aus Gründen der Übersichtlichkeit nicht sämtliche Detailinformationen zu allen Typen des Produkts enthalten. Ferner kann diese Dokumentation nicht jeden möglichen Fall der Aufstellung, des Betriebs und der Instandhaltung berücksichtigen

Durch den Maschinenhersteller vorgenommene Ergänzungen oder Änderungen am Produkt dokumentiert der Maschinenhersteller.

Webseiten Dritter

Dieses Dokument kann Hyperlinks auf Webseiten Dritter enthalten. Siemens übernimmt für die Inhalte dieser Webseiten weder eine Verantwortung noch macht Siemens sich diese Webseiten und ihre Inhalte zu eigen. Siemens kontrolliert nicht die Informationen auf diesen Webseiten und ist auch nicht für die dort bereitgehaltenen Inhalte und Informationen verantwortlich. Das Risiko für deren Nutzung trägt der Nutzer.

1.3 Dokumentation im Internet

1.3.1 Dokumentationsübersicht SINUMERIK 828D

Eine umfangreiche Dokumentation zu den Funktionen von SINUMERIK 828D ab der Version 4.8 SP4 finden Sie unter Dokumentationsübersicht 828D (https://support.industry.siemens.com/cs/ww/de/view/109766724).

Sie haben die Möglichkeit, die Dokumente anzuzeigen oder im PDF- und HTML5-Format herunterzuladen.

Die Dokumentation ist in folgende Kategorien unterteilt:

- Anwender: Bedienen
- Anwender: Programmieren
- Hersteller/Service: Projektieren
- Hersteller/Service: Inbetriebnahme
- Hersteller/Service: Funktionen
- Hersteller/Service: Safety Integrated
- SINUMERIK Integrate/MindApp
- Info & Training

1.3.2 Dokumentationsübersicht SINUMERIK-Bedienkomponenten

Eine umfangreiche Dokumentation zu SINUMERIK-Bedienkomponenten finden Sie unter Dokumentationsübersicht SINUMERIK-Bedienkomponenten (https://support.industry.siemens.com/cs/ww/de/view/109783841).

Sie haben die Möglichkeit, die Dokumente anzuzeigen oder im PDF- und HTML5-Format herunterzuladen.

Die Dokumentation ist in folgende Kategorien unterteilt:

- Bedientafeln
- Maschinensteuertafeln
- Machine Pushbutton Panel
- Handheld Unit/Mini-Handgeräte
- Weitere Bedienkomponenten

Einen Überblick über die wichtigsten Dokumente, Beiträge und Links zum Thema "SINUMERIK" finden Sie unter SINUMERIK Übersicht-Themenseite (https://support.industry.siemens.com/cs/document/109766201/sinumerik-ein-%C3%BCberblick-der-wichtigsten-dokumente-und-links?lc=de-ww).

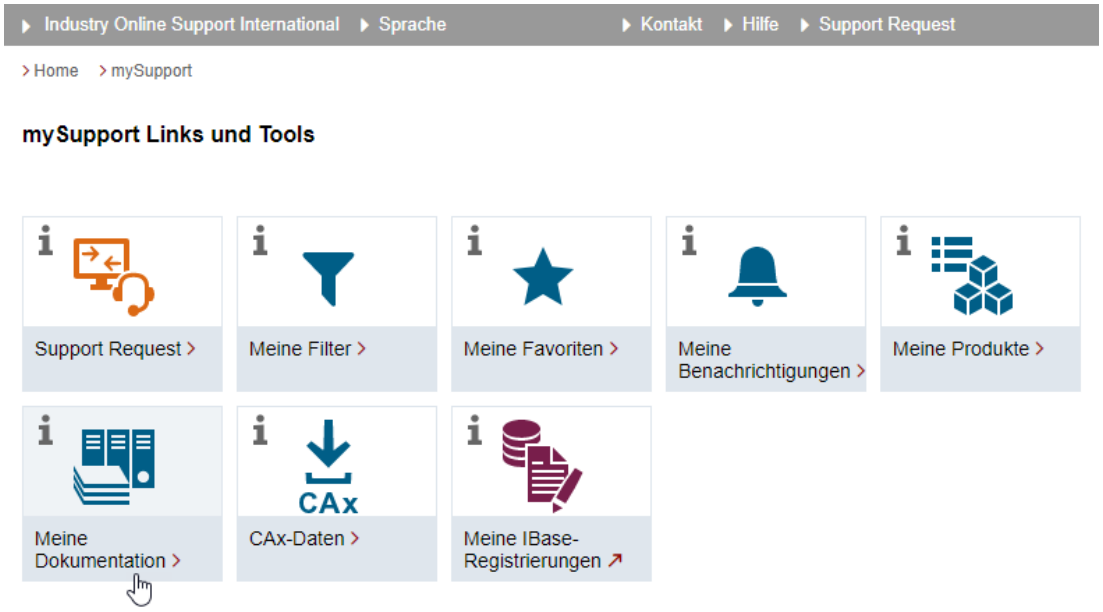
1.4 Feedback zur technischen Dokumentation

Bei Fragen, Anregungen oder Korrekturen zu der im Siemens Industry Online Support veröffentlichten technischen Dokumentation nutzen Sie den Link "Feedback senden" am Ende eines Beitrags.

1.5 mySupport-Dokumentation

Mit dem webbasierten System "mySupport-Dokumentation" können Sie Ihre Dokumentation auf Basis der Siemens-Inhalte individuell zusammenstellen und für die eigene Maschinendokumentation anpassen.

Sie starten die Anwendung über die Kachel "Meine Dokumentation" auf der mySupport-Startseite (https://support.industry.siemens.com/cs/ww/de/my):



Der Export des konfigurierten Handbuchs ist im RTF-, PDF- oder XML-Format möglich.

Hinweis

Siemens-Inhalte, die die Anwendung mySupport-Dokumentation unterstützen, erkennen Sie am Vorhandensein des Links "Konfigurieren".

1.6 Service und Support

Product Support

Weitere Informationen zum Produkt finden Sie im Internet:

Product support (https://support.industry.siemens.com/cs/ww/de/)

Unter dieser Adresse finden Sie Folgendes:

- Aktuelle Produkt-Informationen (Produktmitteilungen)
- FAQ (häufig gestellte Fragen)
- Handbücher
- Downloads
- Newsletter mit den neuesten Informationen zu Ihren Produkten
- Forum zum weltweiten Informations- und Erfahrungsaustausch für Anwender und Spezialisten
- Ansprechpartner vor Ort über unsere Ansprechpartner-Datenbank (→ "Kontakt")
- Informationen über Vor-Ort Service, Reparaturen, Ersatzteile und vieles mehr (→ "Services")

Technical Support

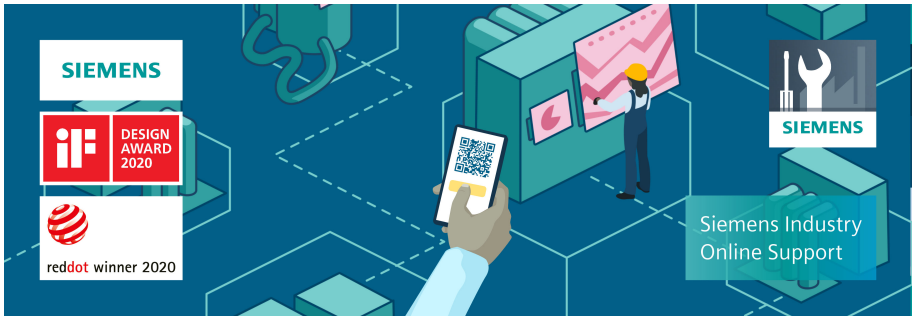
Landesspezifische Telefonnummern für technische Beratung finden Sie im Internet unter der Adresse (https://support.industry.siemens.com/cs/ww/de/sc/4868) im Bereich "Kontakt".

Um eine technische Frage zu stellen, nutzen Sie das Online-Formular im Bereich "Support Request".

Training

Unter folgender Adresse (https://www.siemens.de/sitrain) finden Sie Informationen zu SITRAIN. SITRAIN bietet Trainingsangebote für Siemens-Produkte, Systeme und Lösungen der Antriebs- und Automatisierungstechnik.

Siemens-Support für unterwegs



Mit der preisgekrönten App "Siemens Industry Online Support" haben Sie jederzeit und überall Zugang zu über 300.000 Dokumenten der Siemens Industry-Produkte. Die App unterstützt Sie unter anderem in folgenden Einsatzfeldern:

- Lösen von Problemen bei einer Projektumsetzung
- Fehlerbehebung bei Störungen
- Erweiterung oder Neuplanung einer Anlage

Außerdem haben Sie Zugang zum Technical Forum und weiteren Beiträgen, die von unseren Experten für Sie erstellt werden:

- FAQs
- Anwendungsbeispiele
- Handbücher
- Zertifikate
- Produktmitteilungen und viele andere

Die App "Siemens Industry Online Support" ist für Apple iOS und Android verfügbar.

Data-Matrix-Code auf dem Typenschild

Der Data-Matrix-Code auf dem Typenschild beinhaltet die spezifischen Daten des Geräts. Dieser Code kann mit jedem Smartphone eingelesen werden, über die Mobile App "Industry Online Support" können damit technische Informationen zum entsprechenden Gerät angezeigt werden.

1.7 Wichtige Produktinformationen

Verwendung von OpenSSL

Dieses Produkt kann folgende Software enthalten:

- Software, die durch das OpenSSL-Projekt für die Nutzung innerhalb des OpenSSL-Toolkits entwickelt wurde
- Von Eric Young erstellte kryptografische Software
- Von Eric Young entwickelte Software

Weitere Informationen finden Sie im Internet:

- OpenSSL (https://www.openssl.org)
- Cryptsoft (https://www.cryptsoft.com)

Einhaltung der Datenschutz-Grundverordnung

Siemens beachtet die Grundsätze des Datenschutzes, insbesondere die Gebote der Datenminimierung (privacy by design).

Für dieses Produkt bedeutet das:

Das Produkt verarbeitet oder speichert keine personenbezogenen Daten, lediglich technische Funktionsdaten (z. B. Zeitstempel). Verknüpft der Anwender diese Daten mit anderen Daten (z. B. Schichtplänen) oder speichert er personenbezogene Daten auf dem gleichen Medium (z. B. Festplatte) und stellt so einen Personenbezug her, hat er die Einhaltung der datenschutzrechtlichen Vorgaben selbst sicherzustellen.

Grundlegende Sicherheitshinweise 2

2.1 Allgemeine Sicherheitshinweise

⚠ WARNUNG

Elektrischer Schlag und Lebensgefahr durch weitere Energiequellen

Beim Berühren unter Spannung stehender Teile können Sie Tod oder schwere Verletzungen erleiden.

- Arbeiten Sie an elektrischen Geräten nur, wenn Sie dafür qualifiziert sind.
- Halten Sie bei allen Arbeiten die landesspezifischen Sicherheitsregeln ein.

Generell gelten die folgenden Schritte zum Herstellen von Sicherheit:

1. Bereiten Sie das Abschalten vor. Informieren Sie alle Beteiligten, die von dem Vorgang betroffen sind.
2. Schalten Sie das Antriebssystem spannungsfrei und sichern Sie gegen Wiedereinschalten.
3. Warten Sie die Entladezeit ab, die auf den Warnschildern genannt ist.
4. Prüfen Sie die Spannungsfreiheit aller Leistungsanschlüsse gegeneinander und gegen den Schutzleiteranschluss.
5. Prüfen Sie, ob vorhandene Hilfsspannungskreise spannungsfrei sind.
6. Stellen Sie sicher, dass sich Motoren nicht bewegen können.
7. Identifizieren Sie alle weiteren gefährlichen Energiequellen, z. B. Druckluft, Hydraulik oder Wasser. Bringen Sie die Energiequellen in einen sicheren Zustand.
8. Vergewissern Sie sich, dass das richtige Antriebssystem völlig verriegelt ist.

Nach Abschluss der Arbeiten stellen Sie die Betriebsbereitschaft in umgekehrter Reihenfolge wieder her.

⚠ WARNUNG

Elektrischer Schlag beim Anschluss einer ungeeigneten Stromversorgung

Durch den Anschluss einer ungeeigneten Stromversorgung können berührbare Teile unter gefährlicher Spannung stehen. Der Kontakt mit gefährlicher Spannung kann zu schweren Verletzungen oder Tod führen.

- Verwenden Sie für alle Anschlüsse und Klemmen der Elektronikbaugruppen nur Stromversorgungen, die SELV- (Safety Extra Low Voltage) oder PELV- (Protective Extra Low Voltage) Ausgangsspannungen zur Verfügung stellen.

⚠ WARNUNG

Elektrischer Schlag bei beschädigten Geräten

Unsachgemäße Behandlung kann zur Beschädigung von Geräten führen. Bei beschädigten Geräten können gefährliche Spannungen am Gehäuse oder an freiliegenden Bauteilen anliegen, die bei Berührung zu schweren Verletzungen oder Tod führen können.

- Halten Sie bei Transport, Lagerung und Betrieb die in den technischen Daten angegebenen Grenzwerte ein.
- Verwenden Sie keine beschädigten Geräte.

⚠ WARNUNG

Elektrischer Schlag bei nicht aufgelegten Leitungsschirmen

Durch kapazitive Überkopplung können lebensgefährliche Berührspannungen bei nicht aufgelegten Leitungsschirmen entstehen.

- Legen Sie Leitungsschirme und nicht benutzte Adern von Leitungen mindestens einseitig auf geerdetes Gehäusepotenzial auf.

⚠ WARNUNG

Elektrischer Schlag bei fehlender Erdung

Bei fehlendem oder fehlerhaft ausgeführtem Schutzleiteranschluss von Geräten mit Schutzklasse I können hohe Spannungen an offen liegenden Teilen anliegen, die bei Berühren zu schweren Verletzungen oder Tod führen können.

- Erden Sie das Gerät vorschriftsmäßig.

ACHTUNG

Geräteschaden durch ungeeignete Schraubwerkzeuge

Ungeeignete Schraubwerkzeuge oder ungeeignete Schraubverfahren können die Schrauben des Geräts beschädigen.

- Verwenden Sie Schraubenantriebe, die genau zum Schraubenkopf passen.
- Ziehen Sie die Schrauben mit dem in der technischen Dokumentation angegebenen Drehmoment an.
- Verwenden Sie einen Drehmomentschlüssel oder einen mechanischen Präzisions-Drehschrauber mit dynamischem Drehmomentsensor und Drehzahlbegrenzung

⚠ WARNUNG

Brandausbreitung bei Einbaugeräten

Im Falle eines Brands können die Gehäuse der Einbaugeräte nicht verhindern, dass Feuer und Rauch austreten. Schwere Personen- oder Sachschäden können die Folge sein.

- Bauen Sie Einbaugeräte in einen geeigneten Metallschaltschrank ein, sodass Personen vor Feuer und Rauch geschützt sind, oder schützen Sie Personen durch eine andere geeignete Maßnahme.
- Stellen Sie sicher, dass Rauch nur über kontrollierte Wege entweicht.

⚠ WARNUNG

Unerwartete Bewegung von Maschinen durch Funkgeräte oder Mobiltelefone

Beim Einsatz von Funkgeräten oder Mobiltelefonen in unmittelbarer Nähe der Komponenten können Funktionsstörungen der Geräte auftreten. Die Funktionsstörungen können die funktionale Sicherheit von Maschinen beeinflussen und somit Menschen gefährden oder Sachschäden verursachen.

- Wenn Sie den Komponenten näher als 20 cm kommen, schalten Sie Funkgeräte oder Mobiltelefone aus.
- Benutzen Sie die "SIEMENS Industry Online Support App" nur am ausgeschalteten Gerät.

⚠ WARNUNG

Brand wegen unzureichender Lüftungsfreiräume

Unzureichende Lüftungsfreiräume können zu Überhitzung von Komponenten und nachfolgendem Brand mit Rauchentwicklung führen. Dies kann die Ursache für schwere Körperverletzungen oder Tod sein. Weiterhin können erhöhte Ausfälle und verkürzte Lebensdauer von Geräten/Systemen auftreten.

- Halten Sie die für die jeweilige Komponente angegebenen Mindestabstände als Lüftungsfreiräume ein.

ACHTUNG

Überhitzung bei unzulässiger Einbaulage

Bei unzulässiger Einbaulage kann das Gerät überhitzen und dadurch beschädigt werden.

- Betreiben Sie das Gerät ausschließlich in zugelassenen Einbaulagen.

⚠ WARNUNG

Unerwartete Bewegung von Maschinen durch inaktive Sicherheitsfunktionen

Inaktive oder nicht angepasste Sicherheitsfunktionen können unerwartete Bewegungen an Maschinen auslösen, die zu schweren Verletzungen oder Tod führen können.

- Beachten Sie vor der Inbetriebnahme die Informationen in der zugehörigen Produktdokumentation.
- Führen Sie für sicherheitsrelevante Funktionen eine Sicherheitsbetrachtung des Gesamtsystems inklusive aller sicherheitsrelevanten Komponenten durch.
- Stellen Sie durch entsprechende Parametrierung sicher, dass die angewendeten Sicherheitsfunktionen an Ihre Antriebs- und Automatisierungsaufgabe angepasst und aktiviert sind.
- Führen Sie einen Funktionstest durch.
- Setzen Sie Ihre Anlage erst dann produktiv ein, nachdem Sie den korrekten Ablauf der sicherheitsrelevanten Funktionen sichergestellt haben.

Hinweis

Wichtige Sicherheitshinweise zu Safety Integrated Funktionen

Sofern Sie Safety Integrated Funktionen nutzen wollen, beachten Sie die Sicherheitshinweise in den Safety Integrated Handbüchern.

⚠ WARNUNG

Fehlfunktionen der Maschine infolge fehlerhafter oder veränderter Parametrierung

Durch fehlerhafte oder veränderte Parametrierung können Fehlfunktionen an Maschinen auftreten, die zu Körperverletzungen oder Tod führen können.

- Schützen Sie die Parametrierung vor unbefugtem Zugriff.
- Beherrschen Sie mögliche Fehlfunktionen durch geeignete Maßnahmen, z. B. NOT-HALT oder NOT-AUS.

2.2 Geräteschaden durch elektrische Felder oder elektrostatische Entladung

Elektrostatisch gefährdete Bauelemente (EGB) sind Einzelbauteile, integrierte Schaltungen, Baugruppen oder Geräte, die durch elektrostatische Felder oder elektrostatische Entladungen beschädigt werden können.

ACHTUNG

Geräteschaden durch elektrische Felder oder elektrostatische Entladung

Elektrische Felder oder elektrostatische Entladung können Funktionsstörungen durch geschädigte Einzelbauteile, integrierte Schaltungen, Baugruppen oder Geräte verursachen.

- Verpacken, lagern, transportieren und versenden Sie elektronische Bauteile, Baugruppen oder Geräte nur in der Original-Produktverpackung oder in anderen geeigneten Materialien, z. B. leitfähigem Schaumgummi oder Aluminiumfolie.
- Berühren Sie Bauteile, Baugruppen und Geräte nur dann, wenn Sie durch eine der folgenden Maßnahmen geerdet sind:
  - Tragen eines EGB-Armbands
  - Tragen von EGB-Schuhen oder EGB-Erdungsstreifen in EGB-Bereichen mit leitfähigem Fußboden
- Legen Sie elektronische Bauteile, Baugruppen oder Geräte nur auf leitfähigen Unterlagen ab (Tisch mit EGB-Auflage, leitfähigem EGB-Schaumstoff, EGB-Verpackungsbeutel, EGB-Transportbehälter).

2.3 Gewährleistung und Haftung für Applikationsbeispiele

Applikationsbeispiele sind unverbindlich und erheben keinen Anspruch auf Vollständigkeit hinsichtlich Konfiguration und Ausstattung sowie jeglicher Eventualitäten. Applikationsbeispiele stellen keine kundenspezifischen Lösungen dar, sondern sollen lediglich Hilfestellung bieten bei typischen Aufgabenstellungen.

Als Anwender sind Sie für den sachgemäßen Betrieb der beschriebenen Produkte selbst verantwortlich. Applikationsbeispiele entheben Sie nicht der Verpflichtung zu sicherem Umgang bei Anwendung, Installation, Betrieb und Wartung.

2.4 Security-Hinweise

Siemens bietet Produkte und Lösungen mit Industrial Security-Funktionen an, die den sicheren Betrieb von Anlagen, Systemen, Maschinen und Netzwerken unterstützen.

Um Anlagen, Systeme, Maschinen und Netzwerke gegen Cyber-Bedrohungen zu sichern, ist es erforderlich, ein ganzheitliches Industrial Security-Konzept zu implementieren (und kontinuierlich aufrechtzuerhalten), das dem aktuellen Stand der Technik entspricht. Die Produkte und Lösungen von Siemens formen einen Bestandteil eines solchen Konzepts.

Die Kunden sind dafür verantwortlich, unbefugten Zugriff auf ihre Anlagen, Systeme, Maschinen und Netzwerke zu verhindern. Diese Systeme, Maschinen und Komponenten sollten nur mit dem Unternehmensnetzwerk oder dem Internet verbunden werden, wenn und soweit dies notwendig ist und nur wenn entsprechende Schutzmaßnahmen (z.B. Firewalls und/oder Netzwerksegmentierung) ergriffen wurden.

Weiterführende Informationen zu möglichen Schutzmaßnahmen im Bereich Industrial Security finden Sie unter:
https://www.siemens.com/industrialsecurity (https://www.siemens.com/industrialsecurity)

Die Produkte und Lösungen von Siemens werden ständig weiterentwickelt, um sie noch sicherer zu machen. Siemens empfiehlt ausdrücklich, Produkt-Updates anzuwenden, sobald sie zur Verfügung stehen und immer nur die aktuellen Produktversionen zu verwenden. Die Verwendung veralteter oder nicht mehr unterstützter Versionen kann das Risiko von Cyber-Bedrohungen erhöhen.

Um stets über Produkt-Updates informiert zu sein, abonnieren Sie den Siemens Industrial Security RSS Feed unter:
https://www.siemens.com/industrialsecurity (https://new.siemens.com/global/en/products/services/cert.html#Subscriptions)

Weitere Informationen finden Sie im Internet:

Projektierungshandbuch Industrial Security (https://support.industry.siemens.com/cs/ww/de/view/108862708)

⚠ WARNUNG

Unsichere Betriebszustände durch Manipulation der Software

Manipulationen der Software, z. B. Viren, Trojaner oder Würmer, können unsichere Betriebszustände in Ihrer Anlage verursachen, die zu Tod, schwerer Körperverletzung und zu Sachschäden führen können.

- Halten Sie die Software aktuell.
- Integrieren Sie die Automatisierungs- und Antriebskomponenten in ein ganzheitliches Industrial Security-Konzept der Anlage oder Maschine nach dem aktuellen Stand der Technik.
- Berücksichtigen Sie bei Ihrem ganzheitlichen Industrial Security-Konzept alle eingesetzten Produkte.
- Schützen Sie die Dateien in Wechselspeichermedien vor Schadsoftware durch entsprechende Schutzmaßnahmen, z. B. Virenscanner.
- Prüfen Sie beim Abschluss der Inbetriebnahme alle security-relevanten Einstellungen.

2.5 Restrisiken von Antriebssystemen (Power Drive Systems)

Der Maschinenhersteller oder Anlagenerrichter muss bei der gemäß entsprechenden lokalen Vorschriften (z. B. EG-Maschinenrichtlinie) durchzuführenden Beurteilung des Risikos seiner Maschine bzw. Anlage folgende von den Komponenten für Steuerung und Antrieb eines Antriebssystems ausgehende Restrisiken berücksichtigen:

1. Unkontrollierte Bewegungen angetriebener Maschinen- oder Anlagenteile bei Inbetriebnahme, Betrieb, Instandhaltung und Reparatur z. B. durch:
   - HW- und/oder SW-Fehler in Sensorik, Steuerung, Aktorik und Verbindungstechnik
   - Reaktionszeiten der Steuerung und des Antriebs
   - Betrieb und/oder Umgebungsbedingungen außerhalb der Spezifikation
   - Betauung/leitfähige Verschmutzung
   - Fehler bei der Parametrierung, Programmierung, Verdrahtung und Montage
   - Benutzung von Funkgeräten/Mobiltelefonen in unmittelbarer Nähe der elektronischen Komponenten
   - Fremdeinwirkungen/Beschädigungen
   - Röntgen-, ionisierende und Höhenstrahlung
2. Im Fehlerfall kann es innerhalb und außerhalb der Komponenten zu außergewöhnlich hohen Temperaturen kommen, einschließlich eines offenen Feuers, sowie Emissionen von Licht, Geräuschen, Partikeln, Gasen etc., z. B. durch:
   - Bauelementeversagen
   - Softwarefehler
   - Betrieb und/oder Umgebungsbedingungen außerhalb der Spezifikation
   - Fremdeinwirkungen/Beschädigungen
3. Gefährliche Berührspannungen z. B. durch:
   - Bauelementeversagen
   - Influenz bei elektrostatischen Aufladungen
   - Induktion von Spannungen bei bewegten Motoren
   - Betrieb und/oder Umgebungsbedingungen außerhalb der Spezifikation
   - Betauung/leitfähige Verschmutzung
   - Fremdeinwirkungen/Beschädigungen
4. Betriebsmäßige elektrische, magnetische und elektromagnetische Felder, die z. B. für Träger von Herzschrittmachern, Implantaten oder metallischen Gegenständen bei unzureichendem Abstand gefährlich sein können
5. Freisetzung umweltbelastender Stoffe und Emissionen bei unsachgemäßem Betrieb und/oder bei unsachgemäßer Entsorgung von Komponenten
6. Beeinflussung von netzgebundenen Kommunikationssystemen, z. B. Rundsteuersendern oder Datenkommunikation über das Netz

Weitergehende Informationen zu den Restrisiken, die von den Komponenten eines Antriebssystems ausgehen, finden Sie in den zutreffenden Kapiteln der technischen Anwenderdokumentation.

Anwenderdialoge erstellen 3

3.1 Funktionsumfang

Übersicht

Mit der Funktion "Anwenderdialoge erstellen" wird eine Offenheit gewährleistet, die es dem Anwender ermöglicht kunden- und applikationsspezifische Oberflächen in SINUMERIK Operate zu entwerfen.

Zum Erstellen von Anwenderdialogen bietet die Steuerung eine auf XML basierende Skriptsprache an.

Diese Skriptsprache ermöglicht das Darstellen von maschinenspezifischen Menüs und Dialogmasken in SINUMERIK Operate im Bedienbereich <CUSTOM>.

Alle Dialogmasken lassen sich sprachunabhängig gestalten. Für diesen Fall liest das System die anzuzeigenden Texte aus der mitgelieferten Sprachdatenbank aus.

Verwendung

Die definierten XML-Anweisungen ermöglichen folgende Eigenschaften:

1. Dialoge einblenden und bereitstellen von:
   - Softkeys
   - Variablen
   - Text und Hilfetext
   - Grafiken und Hilfebildern
2. Dialoge aufrufen durch:
   - Betätigen der (Einstiegs-) Softkeys
3. Dialoge dynamisch umstrukturieren:
   - Softkeys ändern, löschen
   - Variablenfelder definieren und gestalten
   - Anzeigetexte (sprachabhängig oder -unabhängig) einblenden, austauschen, löschen
   - Grafiken einblenden, austauschen, löschen
   - Ausführbare Skriptanteile dynamisch erstellen und in die Skriptverarbeitung einbinden
4. Aktionen in Gang setzen bei:
   - Dialoge einblenden
   - Werte (Variablen) eingeben
   - Softkeys betätigen
   - Dialoge verlassen

5. Datenaustausch zwischen Dialogen
6. Variablen
   – Lesen (NC-, PLC-, Anwendervariablen)
   – Schreiben (NC-, PLC-, Anwendervariablen)
   – Verknüpfen mit mathematischen, vergleichenden oder logischen Operatoren
7. Funktionen ausführen:
   – Unterprogramme
   – Datei-Funktionen
   – PI-Dienste
8. Berücksichtigung von Schutzstufen nach Benutzergruppen

Die gültigen Elemente (Tags) für die Scriptsprache beschreibt das Kapitel "XML-Tags" (Seite 45).

Hinweis

Der nachfolgende Abschnitt "Grundlagen für die Projektierung" erhebt bzgl. der XML-Beschreibung (Extensible Markup Language) nicht den Anspruch der Vollständigkeit. Für weitergehende Informationen wird auf die entsprechende Fachliteratur verwiesen.

3.2 Grundlagen für die Projektierung

Projektierungsdateien

Die Beschreibung neuer Bedienoberflächen wird in Projektierungsdateien gespeichert. Diese Dateien werden automatisch interpretiert und das Ergebnis am Bildschirm dargestellt. Die Projektierungsdateien sind bei Lieferung nicht vorhanden und müssen erst angelegt bzw. nachgeladen werden.

Zur Erstellung der Projektierungsdateien kann ein XML-Editor oder ein anderer Texteditor verwendet werden.

Hinweis

Der Dateiname darf nur Kleinbuchstaben enthalten.

Prinzip des Menübaums

Mehrere miteinander verknüpfte Dialoge bilden einen Menübaum. Eine Verknüpfung besteht, wenn Sie von einem Dialog in einen anderen wechseln können. Über neu definierte horizontale oder vertikale Softkeys in diesem Dialog können Sie zum Vorgängerdialog oder in einen beliebigen anderen Dialog wechseln.

Hinter dem Einstiegsmenü kann über projektierte Einstiegssoftkeys ein weiterer Menübaum erzeugt werden:

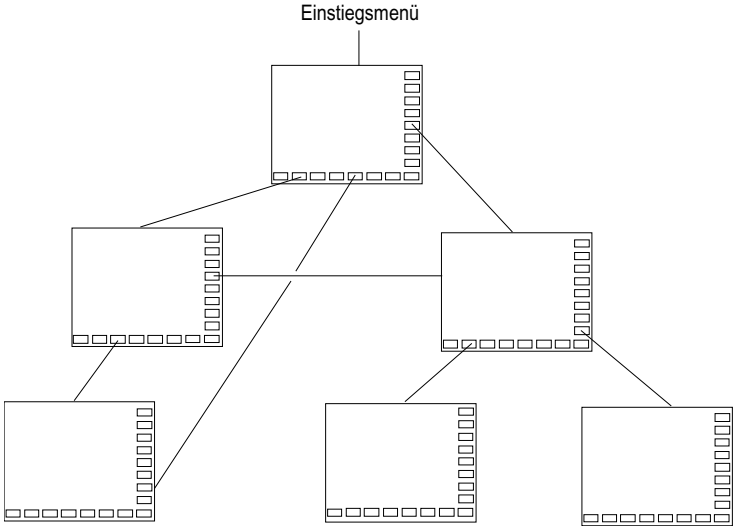


Bild 3-1 Menübaum Anwenderdialoge

Einstiegsmenü

In der Datei "xmldial.xml" wird das Einstiegsmenü mit dem Namen "main" definiert. Das Einstiegsmenü ist der Ausgangspunkt für eigene Bedienabläufe.

Mit dem Menü "main" kann das Laden eigener Dialoge oder weiterer Softkeyleisten verbunden sein, mit denen dann die weiteren Aktionen durchgeführt werden.

Das folgende Bild zeigt das Herstellerverzeichnis "System CF-Card/oem/sinumerik/hmi" auf der Steuerung.

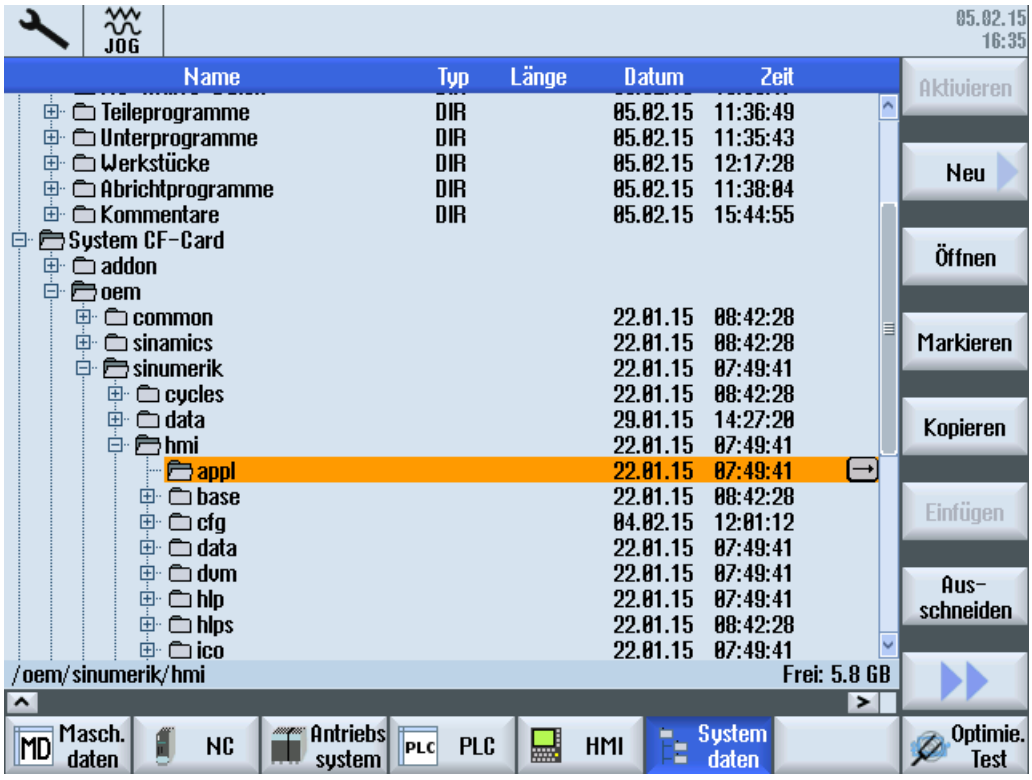


Bild 3-2 Herstellerverzeichnis

Zur Konfiguration von Anwenderdialogen werden in der Steuerung folgende Dateien im Herstellerverzeichnis "System CF-Card/oem/sinumerik/hmi" benötigt:

Tabelle 3-1 Dateien zur Konfiguration

Dateityp	Name der Datei	Bedeutung	Speicherort im Bedienbereich Inbetriebnahme > Systemdaten
Skriptdatei	"xmldial.xml"	Diese Skriptdatei steuert über XML-Tags das Abbild der projektierten Softkeymenüs und Dialogmasken in SINUMERIK Operate im Bedienbereich <CUSTOM>.	Herstellerverzeichnis > Unterverzeichnis "appl" für die Applikationen
Textdatei	"oem_xml_screens_xxx.ts"	Standard-Textdatei für den Customer-Bedienbereich Diese Textdatei enthält die Texte für die Menüs und Dialogmasken für die einzelnen Sprachen.	Herstellerverzeichnis > Unterverzeichnis "lng" für die gewünschten Sprachen

Dateityp	Name der Datei	Bedeutung	Speicherort im Bedienbereich Inbetriebnahme > Systemdaten
Bitmaps	---	Die Steuerung unterstützt die Formate BMP und PNG.	Herstellerverzeichnis > Unterverzeichnis "ico"
XML-Dateien, die in der Steuerdatei "xmldial.xml" mit dem XML-Tag "INCLUDE" eingefügt werden.	z. B. "machine_settings.xml"	Diese Dateien enthalten ebenfalls programmierte Anweisungen zur Darstellung der Dialogmasken und Parameter in SINUMERIK Operate.	Herstellerverzeichnis > Unterverzeichnis "appl" für die Applikationen

Easy XML-Anbindungspunkte

Folgende Anbindungspunkte gibt es für Easy XML-Skripte:

Hard-/Softkey	Anbindungspunkt
Customer	Standardskript-Dateiname "xmldial.xml"
Softkey Operating Menu	Standardskript-Dateiname "xmldial.xml" Wird in der Datei "slamconfig.ini" aktiviert **Beispiel:** `[CustomXML]` `TextId=SL_AM_CUSTOM` `SidescreenTextId=SL_SIDESCREEN_CUSTOM` `TextFile=slam` `TextContext=SlAmAreaMenu` `SoftkeyPosition=12` `Visible=true`
Menu User	Standardskript-Dateiname "menu_user.xml"
Menu Function	Standardskript-Dateiname "menu_function.xml"
PLC-Hardkey	Der Skriptdateiname bezieht sich auf die Key-Beschreibung. **Beispiel:** `KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog, cmdline:="-conf slagmdialog.hmi -mainModule restore.xml -entry cmc2"` `KEY51.0 = area:=CustomXML, dialog:=SlEECustomDialog, cmdline:="-conf slagmdialog.hmi -mainModule activate.xml -entry cmc1"`
MMC-Befehl	Der Skriptdateiname bezieht sich auf die NC-Befehlsbeschreibung. **Beispiel:** `MMC("POPUPDLG, PICTURE_ON, TEST.XML, cmd1","A)`

Hard-/Softkey	Anbindungspunkt
Reservierte Softkeys eines Arbeitsbereichs	Der Skriptdateiname bezieht sich auf die Menü-Beschreibung. **Beispiel:** `<MENU name="DgGlobalHu">` `<ETCLEVEL id="0">` `<SOFTKEYGROUP name="GroupEtc">` `<SOFTKEY position="7">` `<PROPERTY name="textID"` `type="QString">DG_SK</PROPERTY>` `<PROPERTY name="translationContext"` `type="QString">EASY_XML</PROPERTY>` `<FUNCTION args="-area CustomXML -` `dialog SlEECustomDialog -mainModule` `dg.xml -entry main"` `name="switchToDialog"/>` `</SOFTKEY>` `</SOFTKEYGROUP>` `</ETCLEVEL>` `</MENU>`
Sidescreen	Der Skriptdateiname bezieht sich auf die Sidescreen-Beschreibung der Datei "slsidescreen.ini".
Easy Extend-Softkey	Standardskript-Dateiname "agm.xml"

Tasten zur Schnellauswahl

Den Schnellauswahltasten <MENU USER> und <MENU FUNCTION> auf der Bedientafelfront der PPU können ebenfalls beliebige Anwenderdialoge zugewiesen werden. Es stehen dafür die Skriptdatei-Benennungen "menu_user.xml" und "menu_function.xml" zur Verfügung.

Die Skriptdateien kann der Anwender aus der Toolbox in das Herstellerverzeichnis kopieren oder eigene Dateien mit den Benennungen anlegen. Als Einstiegsmenü wird der Name "main" definiert.

3.3 Projektierungsdateien

Einleitung

Folgendes Bild stellt das Herstellerverzeichnis "System CF-Card/oem/sinumerik/hmi" auf der Steuerung dar.

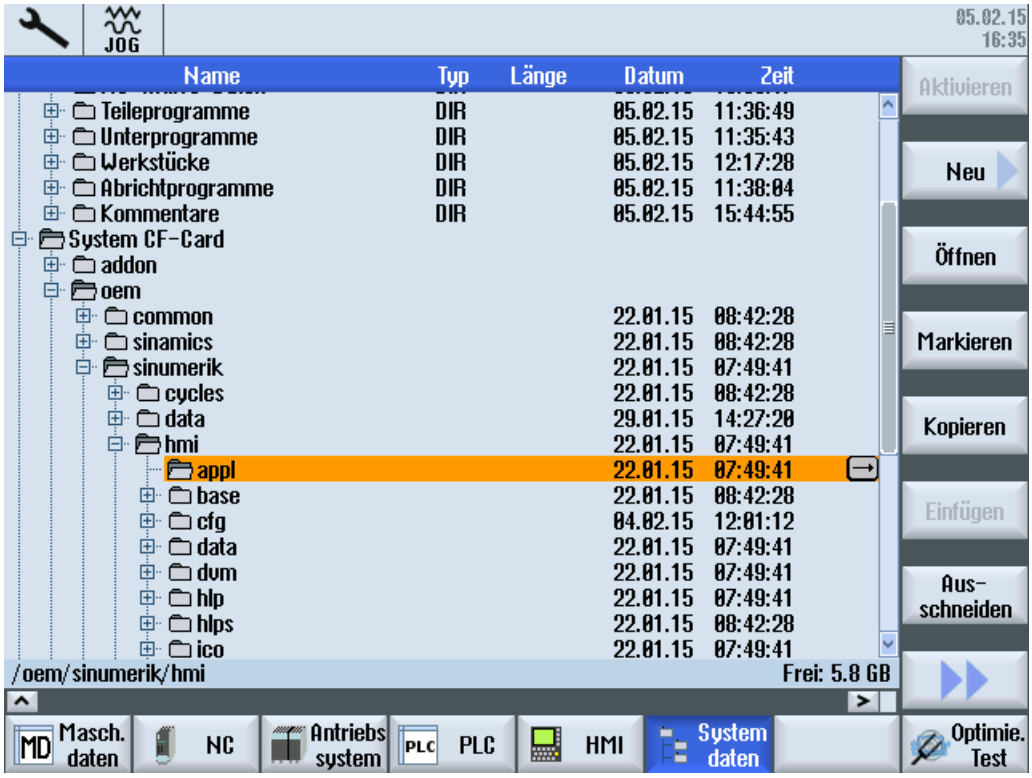


Bild 3-3 Herstellerverzeichnis

Zur Konfiguration von Anwenderdialogen werden in der Steuerung folgende Dateien im Herstellerverzeichnis "System CF-Card/oem/sinumerik/hmi" benötigt:

Tabelle 3-2 Dateien zur Konfiguration

Dateityp	Name der Datei	Bedeutung	Ablageort im Bedienbereich Inbetriebnahme > Systemdaten
Skriptdatei	"xmldial.xml"	Diese Skriptdatei steuert über XML-Tags das Abbild der projektierten Softkeymenüs und Dialogmasken in SINUMERIK Operate im Bedienbereich <CUSTOM>.	Herstellerverzeichnis > Unterverzeichnis "appl" für die Applikationen
Textdatei	"oem_xml_screens_xxx.ts"	Diese Textdatei enthält die Texte für die Menüs und Dialogmasken für die einzelnen Sprachen.	Herstellerverzeichnis > Unterverzeichnis "lng" für die Sprachen

Dateityp	Name der Datei	Bedeutung	Ablageort im Bedienbereich Inbetriebnahme > Systemdaten
Bitmaps		Die Steuerung unterstützt die Formate BMP und PNG.	Herstellerverzeichnis > Unterverzeichnis "ico" Die Bitmaps werden in die Unterverzeichnisse für die zur Steuerung gehörenden Bildschirmauflösung gespeichert. Bem.: Wird ein Pfad auf die Bitmapdatei angegeben, können die Dateien direkt in diesem Verzeichnis abgelegt werden.
XML-Dateien, die in der Steuerdatei "xmldial.xml" mit dem XML-Tag "INCLUDE" eingefügt werden.	z. B. "machine_settings.xml"	Diese Dateien enthalten ebenfalls programmierte Anweisungen zur Darstellung der Dialogmasken und Parameter in SINUMERIK Operate.	Herstellerverzeichnis > Unterverzeichnis "appl" für die Applikationen

Abhängigkeiten der Dateien für die Konfiguration der Anwenderdialoge

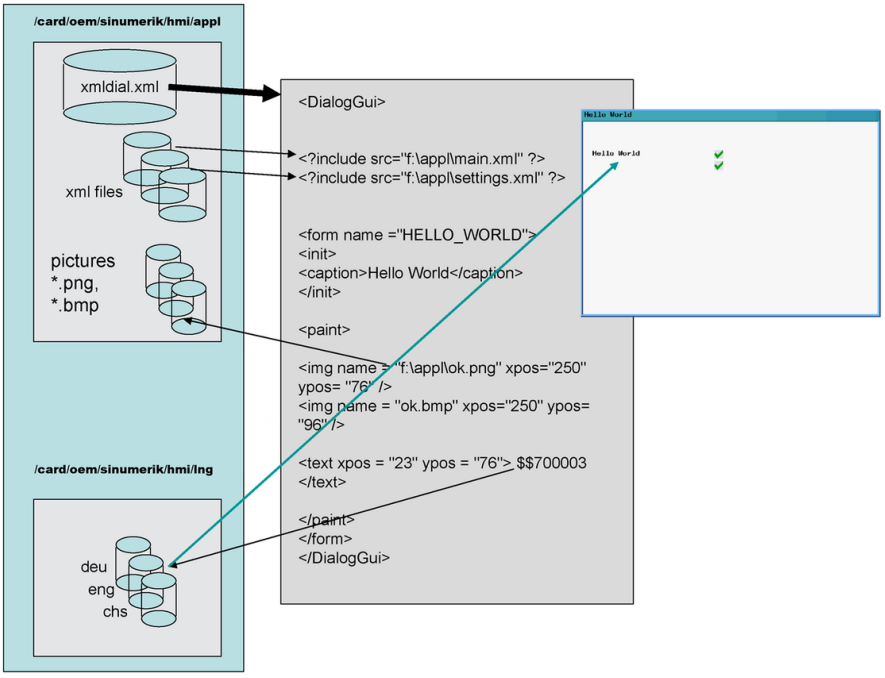


Bild 3-4 Abhängigkeiten

Konfiguration laden

Wie in der vorherigen Tabelle "Dateien zur Konfiguration" in der Spalte "Ablageort im Bedienbereich" beschrieben, sind die erstellten Dateien in die entsprechenden Unterverzeichnisse im Herstellerverzeichnis zu kopieren.

Hinweis

Sobald sich im Unterverzeichnis für Applikationen eine Skriptdatei "xmldial.xml" befindet, kann der Anwender diesen Anwenderdialog im Bedienbereich <CUSTOM> starten.

Nach erstmaligem Kopieren muss ein Reset der Steuerung über "Normalhochlauf" erfolgen.

Beispiel eines Anwenderdialogs in SINUMERIK Operate

Beim Aufrufen des Bedienbereichs <CUSTOM> werden die projektierten Softkeymenüs angezeigt. Der Anwender kann darüber die jeweiligen projektierten Dialogmasken bedienen.

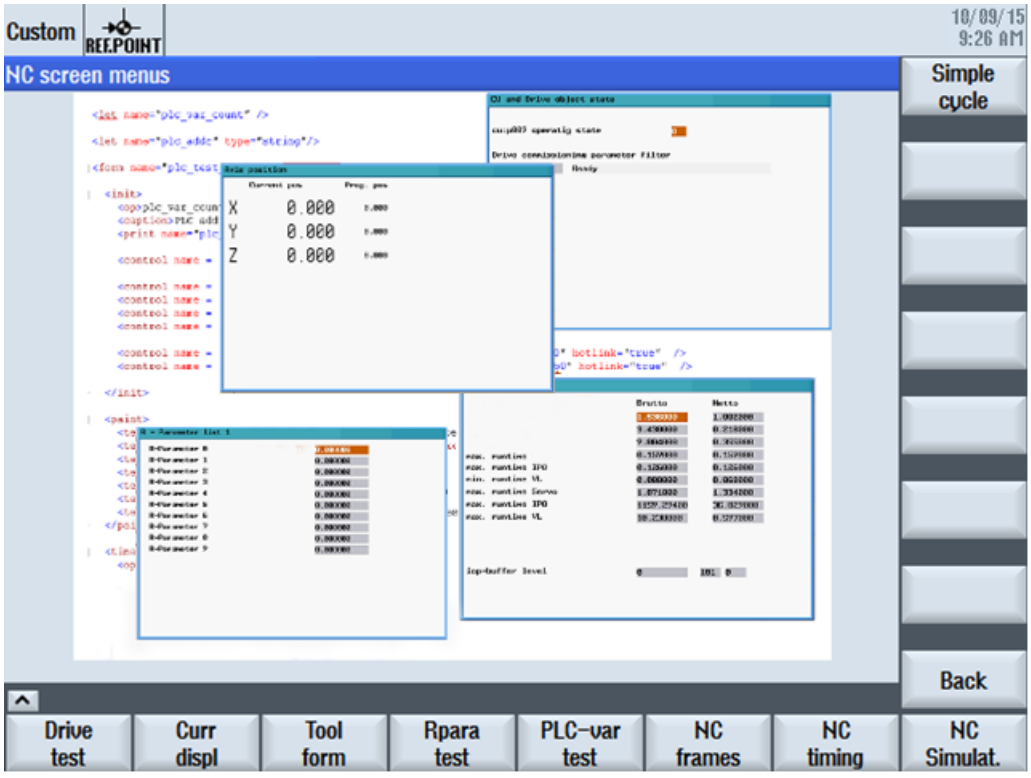


Bild 3-5 Beispiel eines Anwenderdialoges im Bedienbereich <CUSTOM>

Weitere Beispiele sind in der Toolbox zu finden.

3.4 Struktur der Projektierungsdatei

Übersicht

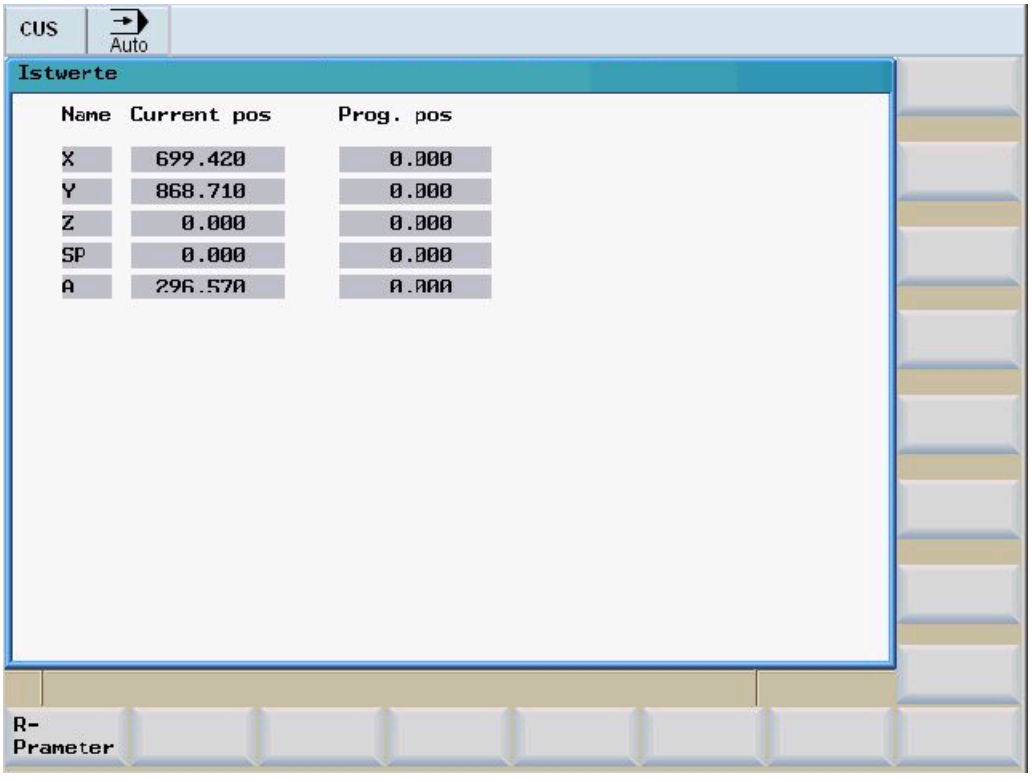
Eine Projektierungsdatei besteht aus folgenden Elementen:

- Beschreibung des Einstiegsmenüs "main" mit Einstiegssoftkeys
- Definition von Dialogen
- Definition der Variablen
- Beschreibung der Blöcke
- Definition von Softkeyleisten

Die folgenden Beispiele stellen das XML-Skript der Datei "xmldial.xml" und die entsprechenden Bildschirmabzüge dar.

Das Skript enthält die Dialoge zum Anzeigen von Istwerten und Restwegen, sowie eine R-Parameterliste.

Dialogmasken-Abschnitt Istwerte



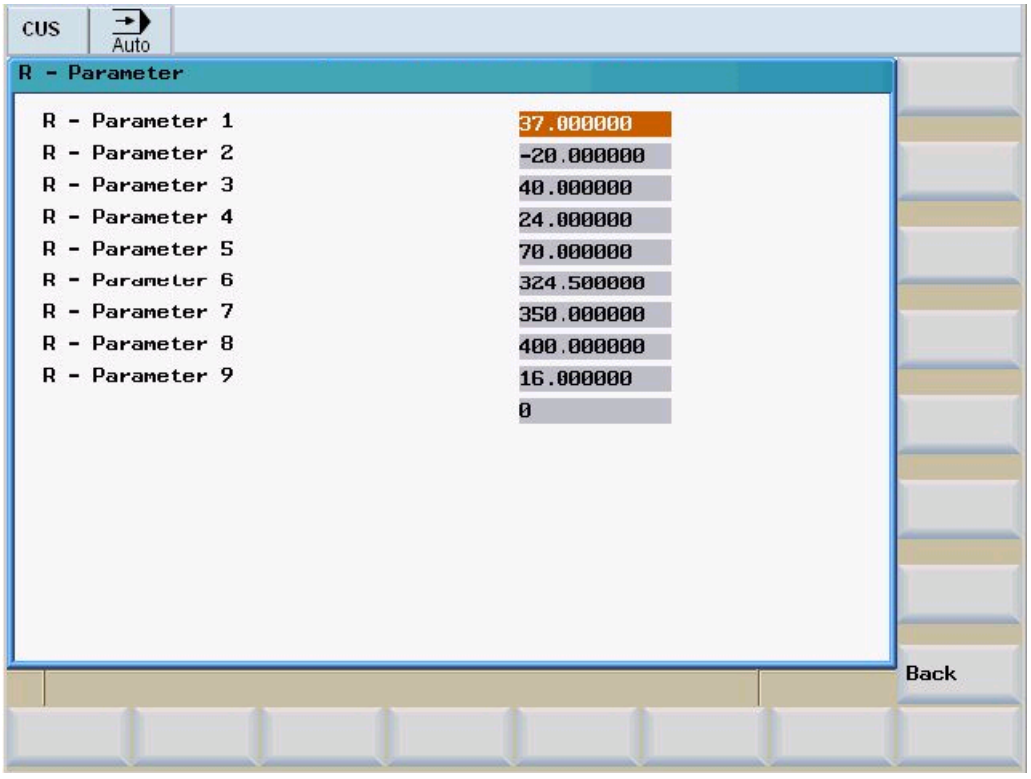
xmldial.xml

```
<DialogGui>

<!--
main menu
It is called by the system software. It starts the application.
The menu tag manages the soft key reactions. One input form can be assigned
to a menu tag.
-->
<menu name = "MAIN">
<OPEN_FORM name = "CURRENT_DISPLAY" />

<softkey POSITION="1">
<caption>R-%nPrameter</caption>
<navigation>MENU_R_PARAMETER</navigation> <!-- opens the menu R parameter --
>
</softkey>


</menu>


<form name = "CURRENT_DISPLAY">

<init>
<caption>Istwerte</caption>

<control name = "label1" xpos = "36" ypos = "56" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[0]" />
<control name = "label2" xpos = "36" ypos = "76" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[1]" />
<control name = "label3" xpos = "36" ypos = "96" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[2]" />
<control name = "label4" xpos = "36" ypos = "116" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[3]" />
<control name = "label5" xpos = "36" ypos = "136" width = "32"
fieldtype="readonly" refvar="nck/Channel/GeometricAxis/name[4]" />

<control name = "edit1" xpos = "80" ypos = "56" refvar="nck/Channel/
GeometricAxis/actProgPos[0]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit2" xpos = "80" ypos = "76" refvar="nck/Channel/
GeometricAxis/actProgPos[1]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit3" xpos = "80" ypos = "96" refvar="nck/Channel/
GeometricAxis/actProgPos[2]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit4" xpos = "80" ypos = "116" refvar="nck/Channel/
GeometricAxis/actProgPos[3]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit5" xpos = "80" ypos = "136" refvar="nck/Channel/
GeometricAxis/actProgPos[4]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
```

xmldial.xml

```
<control name = "edit11" xpos = "210" ypos = "56" refvar="nck/Channel/
GeometricAxis/progDistToGo[0]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit12" xpos = "210" ypos = "76" refvar="nck/Channel/
GeometricAxis/progDistToGo[1]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit13" xpos = "210" ypos = "96" refvar="nck/Channel/
GeometricAxis/progDistToGo[2]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit14" xpos = "210" ypos = "116" refvar="nck/Channel/
GeometricAxis/progDistToGo[3]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>
<control name = "edit15" xpos = "210" ypos = "136" refvar="nck/Channel/
GeometricAxis/progDistToGo[4]" hotlink="true" fieldtype="readonly"
format="%9.3f" time="super fast"/>

</init>

<paint>
<text xpos= "36" ypos="30">Name</text>
<text xpos= "80" ypos="30">Current pos</text>
<text xpos= "210" ypos="30">Prog. pos</text>

</paint>
</form>

.....
```

Dialogmasken-Abschnitt R-Parameter



xmldial.xml

```
.....

<!-- ************************************************************
Menu R- Parameter
-->

<menu name ="MENU_R_PARAMETER">
<OPEN_FORM name = "R-Parameter" />

<softkey POSITION="16">
<caption>Back</caption>
<navigation>MAIN</navigation>
</softkey>

</menu>

<form name = "R-Parameter">

<init>
<DATA_ACCESS type="true" />
<caption>R - Parameter</caption>

<control name = "edit1" xpos = "322" ypos = "34" refvar="nck/Channel/
Parameter/R[1]" />
<control name = "edit2" xpos = "322" ypos = "54" refvar="nck/Channel/
Parameter/R[2]" />
<control name = "edit3" xpos = "322" ypos = "74" refvar="nck/Channel/
Parameter/R[3]" />
<control name = "edit4" xpos = "322" ypos = "94" refvar="nck/Channel/
Parameter/R[4]" />
<control name = "edit5" xpos = "322" ypos = "114" refvar="nck/Channel/
Parameter/R[5]" />
<control name = "edit6" xpos = "322" ypos = "134" refvar="nck/Channel/
Parameter/R[6]" />
<control name = "edit7" xpos = "322" ypos = "154" refvar="nck/Channel/
Parameter/R[7]" />
<control name = "edit8" xpos = "322" ypos = "174" refvar="nck/Channel/
Parameter/R[8]" />
<control name = "edit9" xpos = "322" ypos = "194" refvar="nck/Channel/
Parameter/R[9]" />
<control name = "edit10" xpos = "322" ypos = "214" refvar="nck/Channel/
Parameter/R[10]" />
<control name = "edit10" xpos = "322" ypos = "214" refvar="plc/mb170" />
</init>

<paint>
<text xpos = "23" ypos = "34">R - Parameter 1</text>
<text xpos = "23" ypos = "54">R - Parameter 2</text>
<text xpos = "23" ypos = "74">R - Parameter 3</text>
<text xpos = "23" ypos = "94">R - Parameter 4</text>
<text xpos = "23" ypos = "114">R - Parameter 5</text>
<text xpos = "23" ypos = "134">R - Parameter 6</text>
<text xpos = "23" ypos = "154">R - Parameter 7</text>
<text xpos = "23" ypos = "174">R - Parameter 8</text>
<text xpos = "23" ypos = "194">R - Parameter 9</text>
```

xmldial.xml

```
</paint>
</form>

</DialogGui>
```

3.5 Sprachabhängigkeit

Sprachabhängige Texte werden verwendet für:

- Softkey-Beschriftungen
- Überschriften
- Hilfetexte
- Andere beliebige Texte

Die sprachabhängigen Texte werden in Textdateien abgelegt.

Hinweis

Bei Verwendung dieser Textdateien sind folgende Handlungen notwendig:

- In den erforderlichen Sprachen zur Verfügung stellen.
- In die entsprechenden Sprachverzeichnisse der Steuerung übertragen.

3.6 XML-Diagnose

Zur Diagnose und zum Auffinden von Skriptfehlern bietet das System die Funktion Easy XML-Diagnose an.

Die Anwendung der Diagnosefunktion überprüft die XML-Syntax und durchläuft alle zum Projekt gehörenden Skripte. Dazu werden auch Funktionen, Menüs, Formen sowie die Existenz und Gültigkeit von Variablen überprüft. Die dabei gefundenen Fehler werden aufgelistet.

Die Funktion Easy XML-Diagnose kann entweder mit einem Attribut im **DialogGui**-Tag oder per Anzeigemaschinendatum aktiviert werden.

Easy XML-Diagnose aktivieren

Beispiel:

```
<DialogGui diagnose="true" >
...
...
</DialogGui >
```

oder

Anzeigemaschinendatum:

MD9113 $MM_EASY_XML_DIAGNOSE		Diagnose- und Korrekturhilfeunterstützung für Easy XML-Skripte
= 0	keine Diagnose aktiv	
= 1	Syntaxüberprüfung aktiv	
= 0x100	Die Meldungen werden zusätzlich in der Datei error_log.txt gespeichert.	

Diagnose über Trace-Ausgaben

Trace-Ausgaben können mit dem Tag **debug** in ein Skript integriert werden. Das Speichern erfolgt nur, wenn das Attribut **debug_msg** mit dem Wert **on** im Tag **DialogGui** angegeben ist.

Stoftkey-Funktionsübersicht

Dialog	Softkey	Funktion
Hauptmenü "EasyXML Diagnose"	Start Skript	Das geladene Projekt wird direkt gestartet.
	Start Überprüfung	Die Syntaxüberprüfung wird gestartet. Alle entstandenen Meldungen können eingesehen werden. Erneutes Prüfen ist möglich.

Dialog	Softkey	Funktion
Dialoge "Fehler" und "Warnungen"	Fehler	Das Ergebnis wird sortiert nach Fehlern und Warnungen angezeigt.
	Warnungen	
	Gehe zu Fehler	Wenn Fehler oder Warnungen gefunden wurden, wird der Softkey angezeigt. Der Softkey öffnet die markierte XML-Datei aus der Fehlerliste zum Editieren. Der Cursor wird automatisch auf die fehlerhafte Zeile gestellt.
	Überprüfungsergebnis	Alle entstandenen Meldungen können eingesehen werden.
Dialog "Dokument"	Speichern	Änderungen in der XML-Datei werden gespeichert.
	Abbruch	Die XML-Datei wird unverändert geschlossen. Das Überprüfungsergebnis wird wieder angezeigt.

3.7 XML-Bezeichner

3.7.1 Allgemeiner Aufbau

Aufbau und Anweisungen der Skriptdatei für die Dialogprojektierung

Alle Dialogprojektierungen sind innerhalb des **DialogGui**-Tags abzulegen.

```
<DialogGui>
…
</DialogGui>
```

Beispiel:

```
<?xml version="1.0" encoding="utf-8"?>
<DialogGui>
…
<FORM name ="Hello_World">
<INIT>
<CAPTION>Hello World</CAPTION>
</INIT>
...
</FORM>

</DialogGui>
```

Anweisungen

Zum Abarbeiten von bedingten Anweisungen sowie zum Abarbeiten von Programmschleifen bietet die Sprache folgende Anweisungen an:

- For-Schleife
- While-Schleife
- Do-While-Schleife
- Bedingte Bearbeitung
- Switch- und Case-Anweisungen
- Bedienelemente in einer Dialogmaske
- Softkeybeschreibungen
- Variablen definieren

Eine detaillierte Beschreibung der Anweisungen finden Sie im Kapitel "Anweisung-/ Bezeichnerbeschreibungen (Seite 46)".

3.7.2 Anweisung-/Bezeichnerbeschreibungen

Zum Erstellen der Dialoge und Menüs sowie zum Abarbeiten von Programmsequenzen sind folgende **XML-Tags** definiert:

Hinweis

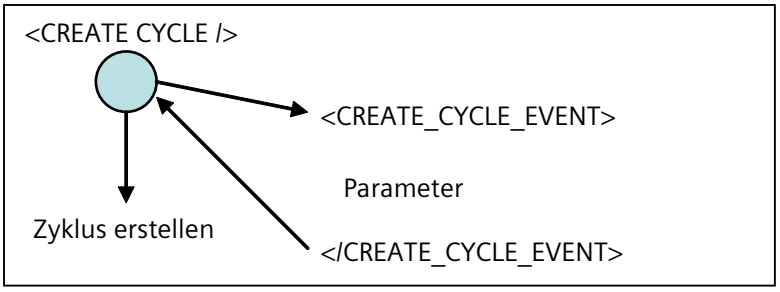
Attributwerte, die in Anführungszeichen durch "<...>" gekennzeichnet sind, sind durch die aktuell verwendeten Ausdrücke zu ersetzen.

Beispiel:
<DATA_LIST action="read/write/append" id="<list name>">
wird folgendermaßen progammiert:

<DATA_LIST action="read/write/append" id="my datalist">

Beim Kopieren von mehrzeiligen XML-Tags und Beispielen ist darauf zu achten, dass die Auszeichnungen nicht durch ungewollte Zeilenumbrüche voneinander getrennt werden.

Tag-Bezeichner	Bedeutung
BREAK	Bedingter Abbruch einer Schleife.
CONDITION	Das Tag ist einzeilig zu programmieren oder bei mehrzeiliger Schreibweise sind die Bedingungen getrennt vom Tag-Namen anzugeben. **Beispiel:** `<if>` `<condition>test <= 2</condition>` `...` oder `<if>` `<condition>` `test <= 2` `</condition>` `...`
CONTROL	Das Tag dient zum Erstellen von Steuerelementen. Die Beschreibung ist dem Kapitel "Softkeymenüs und Dialogmasken erstellen (Seite 78)" zu entnehmen.
CONTROL_RESET	Das Tag ermöglicht, eine Steuerungskomponente oder mehrere neu zu starten. **Syntax:** `<CONTROL_RESET resetnc="TRUE" />` **Attribute:** • **RESETNC = "TRUE"** Die NC-Komponente wird neu gestartet • **RESETDRIVE = "TRUE"** Die Antriebskomponenten werden neu gestartet.

Tag-Bezeichner	Bedeutung
CREATE_CYCLE_EVENT	Startet der Parser die Verarbeitung des Tags CREATE_CYCLE, wird zuerst die Nachricht <CREATE_CYCLE_EVENT> an die aktive Form geschickt. Diese Nachricht kann zum Aufbereiten der Zyklenparameter genutzt werden, bevor der Parser aus der Parameterliste und der Generiervorschrift die NC Anweisung generiert.  **Syntax:** `<CREATE_CYCLE_EVENT>` … … `</CREATE_CYCLE_EVENT>` **Beispiel:** `<SOFTKEY_OK>` … `<CREATE_CYCLE />` … … `<SOFTKEY_OK>` … … `<FORM>` `<NC_INSTRUCTION>MY_CYCLE($P1, $P2)</ NC_INSTRUCTION >` … … `<CREATE_CYCLE_EVENT>` `<type_cast name="P1" type="int"/>` `<OP> P1 = P1 * 150 </OP>` `</CREATE_CYCLE_EVENT>` … … `</FORM>`

Tag-Bezeichner	Bedeutung
DATA	Das Tag ermöglicht, das Schreiben auf NC, PLC, GUD und Antriebsdaten. Die Adressbildung ist dem Kapitel "Komponentenadressierung (Seite 143)" zu entnehmen. **Attribut:** • **name** Variablenadresse **Tag-Wert:** Als Tag-Wert wird eine Konstante, **eine Variable oder ein Term** erwartet. **Syntax:** <DATA name="<variable name>"> value </DATA> <DATA name="<variable name>"> <term> </DATA> **Beispiel:** <DATA name = "plc/mb170"> 1 </DATA> … <LET name = "tempVar"> 7 </LET> <!— der Inhalt der lokalen Variablen "tempVar" wird auf das Merker-Byte 170 geschrieben → <DATA name = "plc/mb170">$tempVar</DATA>
DATA Fortsetzung	**Beispiel:** Berechnung eines Terms. Das Ergebnis wird der angegebenen Variablen zugewiesen. <let name="a" >#f</let> <let name="b" >7</let> <data name = "b"> a & b</data> <let name="c" type="string"></let> <data name = "c"> _T" test" + _T" and test"</data>

Tag-Bezeichner	Bedeutung
DATA_LIST	Das Tag ermöglicht das Sichern bzw. Wiederherstellen der aufgeführten Antriebs- und Maschinendaten. Die Auflistung der Adressen erfolgt zeilenweise. Die Adressbildung ist dem Kapitel "Komponentenadressierung (Seite 143)" zu entnehmen. Es können bis zu 20 temporäre Datenlisten angelegt werden. **Attribute:** • **action** *read* – die Werte der aufgelisteten Variablen werden in einem temporären Speicher abgelegt *append* – die Werte der aufgelisteten Variablen werden an eine bestehende Liste angefügt *write* – die gesicherten Werte werden in die entsprechenden Maschinendaten kopiert • **id** der Bezeichner dient zur Identifikation des temporären Speichers **Syntax:** `<DATA_LIST action="<read/write/append>"  id="<list name>">` `NC/PLC Adresszusammenstellung` `</DATA_LIST>` **Beispiel:** `<DATA_LIST action ="read" id="<name>">` `nck/channel/parameter/r[2]` `nck/channel/parameter/r[3]` `nck/channel/parameter/r[4]` `$MN_USER_DATA_INT[0]` … `</ DATA_LIST>` `<DATA_LIST action ="write" id="<name>" />`
DIALOGGUI	Dieses Tag stellt das Root-Tag für die Funktion Easy XML dar. Alle eingeschlossenen Anweisungen werden durch den Easy XML-Parser bearbeitet. **Attribute:** Die aufgeführten Attribute können optional angegeben werden, um zentrale Funktionen einzuschalten: • **diagnose** - Wert **true** aktiviert die XML-Diagnose • **debug_msg** - Wert „on“ Es erfolgt das Speichern von Trace-Nachrichten **Weitere Informationen** unter Kapitel "XML-Diagnose (Seite 43)" • **textfile** - Name der zu verwendenden Textdatei • **textcontext** - zu verwendender Textkontext nur gültig im Zusammenhang mit dem Attribut textfile • **textfilelist** - ermöglicht das Laden einer Liste mit verschiedenen Textdateien **Weitere Informationen** unter Kapitel "Projektspezifische Textdateien (Seite 253)"
DIALOGGUI Fortsetzung	• **restoresession** - Wert **true** Wird das Easy XML-Projekt wiederholt angewählt, öffnet der Parser das zuletzt angewählte Menü und die zuletzt angewählte Form. Dieses Verhalten bleibt bis zu einem HMI-Neustart erhalten. **Weitere Informationen** unter Kapitel "Sitzung wiederherstellen (Seite 259)"

Tag-Bezeichner	Bedeutung
DEBUG_MSG	Das Attribut steuert die Speicherung von Trace-Ausgaben. Die Beschreibung ist dem Kapitel "Softkeymenüs und Dialogmasken erstellen (Seite 78)" unter dem Tag DEBUG zu entnehmen. Ist der Wert **on** zugewiesen, speichert der Parser alle Ausgaben in eine Datei. Dies kann zu einer verlangsamten Skriptabarbeitung führen. Standardmäßig ist der Wert **off** gesetzt.
DO_WHILE	Do-While-Schleife `DO` `  Anweisungen` `WHILE (Test)` **Syntax:** `<DO_WHILE>` `Anweisungen` `…` `<CONDITION>…</CONDITION>` `</DO_WHILE>` Die Do-While-Schleife besteht aus einem Anweisungsblock und einer Bedingung. Der Code innerhalb des Anweisungsblocks wird zuerst ausgeführt, anschließend die Bedingung ausgewertet. Falls die Bedingung wahr (true) ist, führt die Funktion den Codeanteil erneut aus. Dies wiederholt sich, bis die Bedingung falsch (false) wird. **Beispiel:** `<DO_WHILE>` `<DATA name = "PLC/qb11"> 15 </DATA>` `<CONDITIION> "plc/ib9" == 0 </CONDITION>` `</DO_WHILE>`
DYNAMIC_INCLUDE	Das Tag inkludiert eine XML-Skriptdatei. Im Gegensatz zum INCLUDE-Tag wird das Einlesen erst beim Abarbeiten der entsprechenden Anweisung ausgeführt. Bei großen Projekten führt die Verwendung des Tags zu einer Verkürzung der Ladezeit des Customer-Bereichs bzw. der Zyklenunterstützung. Weiterhin sinkt der durchschnittliche Ressourcenbedarf, da nicht immer alle Dialoge während einer Sitzung aufgerufen werden. **Syntax:** <DYNAMIC_INCLUDE src="path name"/> **Beispiel:** `<SOFTKEY POSITION="3">` `  <CAPTION>MY_MENU</CAPTION>` `  <DYNAMIC_INCLUDE src="f:\appl\sk3_script.xml"/>` `  <NAVIGATION>MY_MENU</NAVIGATION>` </SOFTKEY>
ELSE	Anweisung, wenn die Bedingung nicht erfüllt wurde (**IF, THEN, ELSE**)

Tag-Bezeichner	Bedeutung
FOR	For-Schleife `for (Initialisierung; Test; Fortsetzung) Anweisung(en)` **Syntax:** `<FOR>` `<INIT>…</INIT>` `<CONDITION>…</CONDITION>` `<INCREMENT>…</INCREMENT>` `Anweisungen` `…` `</FOR>` Die For-Schleife wird wie folgt ausgeführt: 1. Auswertung des Ausdrucks **Initialisierung** (INIT). 2. Auswertung des Ausdrucks **Test** (CONDITION) als boolescher Ausdruck. Falls der Wert falsch (false) ist, wird die For-Schleife beendet. 3. Ausführen der nachfolgenden Anweisungen. 4. Auswerten des Ausdrucks **Fortsetzung** (INCREMENT). 5. Weiter mit 2. Alle verwendeten Variablen, die im INIT-, CONDITION- und INCREMENT-Zweig genutzt werden, sind außerhalb der For-Schleife anzulegen. **Beispiel:** `<LET name = "count">0</LET>` `<FOR>` `<INIT>` `<OP> count = 0</OP>` `</INIT>` `<CONDITION> count <= 7 </CONDITION>` `<INCREMENT>` `<OP> count = count + 1 </OP>` `</INCREMENT>` `<OP> "plc/qb10" = 1+ count </OP>` `</FOR>`
FORM	Das Tag beinhaltet die Beschreibung eines Anwenderdialogs. Die Beschreibung ist dem Kapitel "Softkeymenüs und Dialogmasken erstellen (Seite 78)" zu entnehmen.
HMI_RESET	Das Tag initiiert einen HMI-Neustart. Die Interpretation wird nach dieser Anweisung abgebrochen.

Tag-Bezeichner	Bedeutung
IF	Bedingte Anweisung (**IF, THEN, ELSE**) Die Tags THEN und ELSE sind in das IF-Tag eingeschlossen. Dem IF-Tag folgt die Bedingung, die im CONDITION-Tag ausgeführt wird. Das Operationsergebnis entscheidet über die weitere Bearbeitung der Anweisungen. Ist das Funktionsergebnis wahr, wird der THEN-Zweig ausgeführt und der ELSE-Zweig übersprungen. Ist das Funktionsergebnis unwahr, arbeitet der Parser den ELSE-Zweig ab. **Syntax:** `<IF>` `<CONDITION> Bedingung != 7 </CONDITION>` `<THEN>` `Anweisung für den Fall: Bedingung erfüllt` `</THEN>` `<ELSE>` `Anweisung für den Fall: Bedingung nicht erfüllt` `</ELSE>` `</IF>` **Beispiel:** `<IF>` `<CONDITION> "plc/mb170" != 7 </CONDITION>` `<THEN>` `<OP> "plc/mb170" = 7 </OP>` … `</THEN>` `<ELSE>` `<OP> "plc/mb170" = 2 </OP>` … `</ELSE>` `</IF>`
IF Fortsetzung	Wenn in der Bedingung ausschließlich lokale Variablen verwendet werden, kann die Bedingung als Attribut im IF-Tag angegeben werden. **Beispiel:** `<let name="var1">1</let>` `<if condition="var == 1">` `<then>` `...` `...` `</then>` `</if>` **Hinweis:** Controls, die nicht mit einer Referenzvariablen gekoppelt werden, ist der Datentyp Integer zugewiesen. Ausnahmen: Den Controls vom Typ **imagebox** und **multiline** werden der Datentyp **string** zugewiesen.

Tag-Bezeichner	Bedeutung
INCLUDE	Die Anweisung inkludiert eine XML-Beschreibung. (siehe auch DYNAMIC_INCLUDE in dieser Tabelle) **Attribut:** • **src** enthält den Pfadnamen. **Syntax:** `<?INCLUDE src="<Path name>" ?>`

<table>
<tr><th>Tag-Bezeichner</th><th>Bedeutung</th></tr>
<tr><td>LET</td><td>
Die Anweisung legt eine lokale Variable unter dem angegebenen Namen an.
Felder:
Mit dem Attribut dim (Dimension) lassen sich ein- oder zweidimensionale Felder anlegen. Die Adressierung der einzelnen Feldelemente erfolgt über den Feldindex.
Bei einem zweidimensionalen Feld wird zuerst der Zeilenindex angegeben und dann der Spaltenindex.
• Eindimensionales Feld:
Index 0 bis 4
<table>
<tr><td>0</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td></tr>
<tr><td>3</td></tr>
<tr><td>4</td></tr>
</table>
• Zweidimensionales Feld:
Index Zeile 0 bis 3 und Index Spalte 0 bis 5
<table>
<tr><td>0,0</td><td>0,1</td><td>0,2</td><td>0,3</td><td>0,4</td><td>0,5</td></tr>
<tr><td>1,0</td><td>1,1</td><td>1,2</td><td>1,3</td><td>1,4</td><td>1,5</td></tr>
<tr><td>2,0</td><td>2,1</td><td>2,2</td><td>2,3</td><td>2,4</td><td>2,5</td></tr>
<tr><td>3,0</td><td>3,1</td><td>3,2</td><td>3,3</td><td>3,4</td><td>3,5</td></tr>
</table>
Attribute:
• name
Variablenname
• type
Der Variablentyp kann Integer (INT), Unsigned Integer (UINT), Double (DOUBLE), Float (FLOAT), String (STRING) oder STRUCT sein. Weiterhin ist es möglich, eine per typedef erstellte Struktur als Variablentyp zu verwenden (siehe Tag-Bezeichner TYPEDEF). Ist keine Typ-Anweisung angegeben, legt das System eine Integervariable an.
<code><LET name = "VAR1" type = "INT" /></code>
• permanent
Ist das Attribut auf true gesetzt, wird der Variablenwert dauerhaft gespeichert. Dieses Attribut wirkt nur für globale Variable.
• poweroff
Ist der Wert des Attributs true, bleiben die Werte bis zum Ausschalten der Steuerung erhalten.
Weitere Informationen unter Kapitel "Sitzung wiederherstellen (Seite 259)"
• dim
Es ist die Anzahl Feldelemente anzugeben. Bei einem zweidimensionalen Feld wird die zweite Dimension durch ein Komma getrennt nach der ersten Dimension angegeben. Der Zugriff auf ein Feldelement erfolgt über den Feldindex, der in eckigen Klammern nach dem Variablennamen anzugeben ist.
name[index] oder name[row,column]
– Eindimensionales Feld: dim="<Anzahl Elemente>"
– Zweidimensionales Feld: dim="<Anzahl Zeilen>,<Anzahl Spalten>"
Nicht initialisierte Feldelemente sind mit "0" vorbelegt.
</td></tr>
</table>

Tag-Bezeichner	Bedeutung
LET Fortsetzung	**Beispiel:** Eindimensionales Feld: `<let name="array" dim="10"></let>` Zweidimensionales Feld: `<let name="list_string" dim="10,3" type="string"></let>` **Vorbelegung:** Eine Variable kann mit einem Wert initialisiert werden. `<LET name = "VAR1" type = "INT"> 10 </LET>` Werden Werte aus NC oder PLC-Variablen in einer lokalen Variable abgelegt, passt die Zuweisungsoperation das Format automatisch an das Format der eingelesenen Variablen an. • Vorbelegung einer Stringvariablen: Einer Stringvariablen lassen sich mehrzeilige Texte zuweisen, indem man den formatierten Text als Wert übergibt. Soll eine Zeile mit einem line feed <LF> (Zeilenvorschub) abgeschlossen werden, sind die Zeichen "\\n" am Ende der Zeile anzufügen. `<LET name = "text" type = "string"> F4000 G94\\n` `G1 X20\\n` `Z50\\n` `M2\\n` `</LET>>` **`Felder (Arrays):`** `<let name="list" dim="10,3">` `  {1,2,3},` `  {1,20}` `</let>` `<let name="list_string" dim="10,3" type="string">` `  {"text 10","text 11"},` `  {"text 20","text 21"}` `</let>` **Zuweisung:** Werte aus den Maschinendaten bzw. Subroutinen können mit der Zuweisungsoperation "=" einer Variablen zugewiesen werden. Die Gültigkeit einer Variable erstreckt sich bis zum Ende des übergeordneten XML-Blocks. Variablen, die global zur Verfügung stehen sollen, sind direkt nach dem Tag **DialogGUi** anzulegen. Bei einer Dialogbox ist folgendes zu beachten: • Die Nachrichtenbearbeitung öffnet das entsprechende Tag. • Nach dem Ausführen der Nachricht wird das Tag geschlossen. • Alle Variablen innerhalb des Tags werden mit dem Schließen gelöscht.

Tag-Bezeichner	Bedeutung
LET Fortsetzung	**Variablentyp struct**: Dieser Variablentyp beinhaltet eine Zusammensetzung von Variablen, die unter Verwendung des Strukturnamens angesprochen werden können. Eine Struktur kann alle Variablentypen sowie Strukturen enthalten. Innerhalb der Struktur wird eine Variable durch das Tag „element" deklariert. Die Attribute des Tags und die Initialisierung entsprechen den Attributen und der Initialisierung der let Anweisung. `<let name="name" type="struct">` `<element name="<Name>" type="<Variablen Typ>" />` ... ... `</let>` Einem Strukturelement kann ein Default-Wert zugewiesen werden, der beim Erstellen einer Variablen als Initialwert verwendet wird. **Weitere Informationen** unter Abschnitt "Initialisieren von Strukturen" Dieser Wert wird überschrieben, wenn ein Initialwert beim Anlegen der Variablen angegeben wird.

Tag-Bezeichner	Bedeutung
LET Fortsetzung	Der Zugriff auf eine Variable der Struktur erfolgt über den Struktur- und Variablennamen. Beide Namen werden durch den Punktoperator getrennt. **Syntax:** `<op>` `Struktur_name.variablen_name = wert;` `</op>` **Beispiel:** `<let name="info" type="struct">` `  <element name="id" type="int" />` `  <element name="name" type="string" />` `  <element name="phone" type="string" />` `</let>` `<op>` `  info.id = 1;` `  info.name = _T"my name";` `  info.phone = _T"0034 45634";` `</op>`

Tag-Bezeichner	Bedeutung
LET Fortsetzung	**Initialisieren von Strukturen:** Strukturen können beim Erzeugen der Variablen initialisiert werden, indem man pro Strukturelement einen Initialwert angibt. Bei einem Feld von Strukturen ist jede Struktur durch geschweifte Klammern voneinander zu trennen. **Beispiel:** `<typedef name="StructRect" type="struct" >` `  <element name="left" type="int">22</element>` `  <element name="top" type="int">122</element>` `  <element name="right" type="int">220</element>` `  <element name="bottom" type="int">56</element>` `</typedef>` `<let name="rect" type="StructRect" >` Mit dem Anlegen der Variablen **rect** sind die Strukturelemente mit den folgenden Werten initialisiert: `rect.left= 22` `rect.top = 122` `rect.right = 220` `rect.bottom = 56` Weist man einer Strutur Initialwerte zu, überschreiben diese die Defaultwerte. `<let name="rect" type="StructRect" >` `  {11,12}` `</let>` Mit dem Anlegen der Variablen rect sind die Strukturelemente mit den folgenden Werten initialisiert: `rect.left= 11` `rect.top = 12` `rect.right = 220` `rect.bottom = 56`

Tag-Bezeichner	Bedeutung
LET Fortsetzung	**Verschachtelte Strukturen:** Strukturen sind Variablen eines vom Anwender festgelegten Variablentyps und können somit als Strukturelemente in einer Struktur verwendet werden. Die Initialisierung der Elemente erfolgt nach der beschriebenen Regel. **Beispiel:** `<typedef name="StructRectRect" type="struct" >` `  <element name="r1" type="StructRect">77, 99, 232, 23</element>` `  <element name="r2" type="StructRect">77, 88, 45, 12</element>` `</typedef>` `<let name="rect_rect_array" type="StructRectRect" >` Mit dem Anlegen der Variablen **rect_rect_array** sind die Strukturelemente mit den folgenden Werten initialisiert: `r1.left= 77` `r1.top = 99` `r1.right = 232` `r1.bottom = 23` `r2.left= 77` `r2.top = 88` `r2.right = 45` `r2.bottom = 12` `<let name="rect_rect_array1" type="StructRectRect" >` `  {{11,12,13,14},{111,188,199,114}}` `</let>` Mit dem Anlegen der Variablen **rect_rect_array** sind die Strukturelemente mit den folgenden Werten initialisiert: `r1.left= 11` `r1.top = 12` `r1.right = 13` `r1.bottom = 14` `r2.left= 111` `r2.top = 188` `r2.right = 199` `r2.bottom = 114`
LET Fortsetzung	**Initialisieren von globalen String-Variablen:** Verwendet man zum Initialisieren einer globalen String-Variablen einen Text-Identifier, wird der Identifier und der zu ersetzende Text in der Standardtextdatei **oem_xml_screens_xxx.ts** oder in der im Tag **DialogGUI** angegebenen Textdatei erwartet. Beim Laden der Applikation ersetzt der Text der aktiven Sprache den Text-Identifier. Wird ein Sprachwechsel durchgeführt, während die Applikation aktiv ist, erfolgt keine neue Initialisierung der globalen String-Variablen. Der Text kann in der Nachricht **LANGUAGE_CHANGED** ausgetauscht werden.
LOCK_OPERATING_AREA	Der Bedienbereichswechsel wird gesperrt. Mit dem Tag **UNLOCK_OPERATING_AREA** wird die Bedienbereichswechselsperre wieder aufgehoben. **Syntax:** `<LOCK_OPERATING_AREA />`

Tag-Bezeichner	Bedeutung
MSG	Die Bedienkomponente zeigt die im Tag angegebene Nachricht an. Wird eine Alarmnummer verwendet, zeigt die Dialogbox den für die Nummer hinterlegten Text an. **Beispiel:** `<MSG text ="my message" />`
MSGBOX	Die Anweisung öffnet eine Messagebox, deren Rückgabewert zur Verzweigung genutzt werden kann. **Syntax:** `<MSGBOX text="<Message>" caption="<caption>" retvalue="<variable name>" type="<button type>" />` **Attribute:** • **text** Text • **caption** Überschrift • **retvalue** Name der Variablen, in die der Rückgabewert kopiert wird: 1 – OK 0 – CANCEL • **type** Quittierungsmöglichkeiten: "BTN_OK" "BTN_CANCEL" "BTN_OKCANCEL" Wenn für das Attribut **"text"** oder **"caption"** eine Alarmnummer verwendet wird, dann zeigt die Messagebox den für die Nummer hinterlegten Text an. **Beispiel:** `<MSGBOX text="Test message" caption="Information" retvalue="result" type="BTN_OK" />`

Tag-Bezeichner	Bedeutung
OP	Das Tag führt die angegebenen Operationen aus. Es können die im Kapitel "Operatoren (Seite 76)" aufgeführten Operationen ausgeführt werden. Für den Zugriff auf die NC-/PLC- und Antriebsdaten ist der vollständige Variablenname in **Anführungszeichen** zu setzten. Die Adressbildung ist dem Kapitel "Komponentenadressierung" (Seite 143) zu entnehmen. `PLC: "PLC/MB170"` `NC: "NC/Channel/…"` **Beispiel:** `<LET name = "tmpVar" type="INT"> </LET>` `<OP> tmpVar = "plc/mb170" </OP>` `<OP> tmpVar = tmpVar *2 </OP>` `<OP> "plc/mb170" = tmpVar </OP>` Innerhalb eines Operation-Tags können mehrere Gleichungen verwendet werden. Ein Semikolon markiert das Ende der Anweisung. **Beispiel:** `<op>` `x = x+1;` `y = y+1;` `</op>` **Zeichenkettenverarbeitung:** Die Operationsanweisung ist in der Lage, Zeichenketten zu verarbeiten und die resultierende Zeichenkette der in der Gleichung angegebenen String-Variablen zu zuweisen. Zum Kennzeichnen von Textausdrücken ist dem Text die Kennung _T voranzustellen. Weiterhin ist das Formatieren von Variablenwerten möglich. Die Formatierungsvorschrift ist mit der Kennung _F einzuleiten, gefolgt von der Formatanweisung. Anschließend wird die Adresse der Variablen angegeben. **Beispiel:** `<LET name="buffer" type="string"></LET>` … … … `<OP> buffer = _T"unformatted value R0= " + "nck/Channel/Parameter/R[0]" + _T" and " + _T"$$85051" + _T" formatted value R1 " + _F %9.3f"nck/Channel/Parameter/R[1]" </OP>`

Tag-Bezeichner	Bedeutung
OPERATION	**Operation** Eine Verschiebungsoperation kann innerhalb einer Gleichung verwendet werden. **Operator nach links verschieben "<<"** Bits werden mit der Funktion << nach links verschoben. Sie können den zu verschiebenden Wert und die Anzahl von Verschiebungsinkrementen direkt oder über eine Variable festlegen. Wenn die Grenze des Datenformats erreicht ist, werden die Bits über die Grenze hinaus verschoben, ohne dass eine Fehlermeldung angezeigt wird. **Ausführungsregel** Ausführung von links nach rechts und '+' und '-' vor '<<' oder '>>' **Beispiel:** `<op> idx = idx << 2 </op>` `<op> idx = 3 + idx << 2 </op>` **Operator nach rechts verschieben ">>"** Bits werden mit der Funktion >> nach rechts verschoben. Sie können den zu verschiebenden Wert und die Anzahl von Verschiebungsinkrementen direkt oder über eine Variable festlegen. Wenn die Grenze des Datenformats erreicht ist,werden die Bits über die Grenze hinaus verschoben, ohne dass eine Fehlermeldung angezeigt wird. **Ausführungsregel** Ausführung von links nach rechts und '+' und '-' vor '<<' oder '>>' **Beispiel:** `<op> idx = idx >> 2 </op>` `<op> idx = 3+idx >> 2</op>`
PASSWORD	Für die Funktion „EasyExtend" dient dieses Tag zum Freischalten von Zusatzaggregaten. Die eingegebene Zeichenkette wird in die angegebene Referenzvariable geschrieben und ist im PLC-Anwenderprogramm zu verarbeiten. **Syntax:** `<PASSWORD refVar ="<variable name>" />` **Attribute:** • **refVar** Name der Referenzvariablen • **text** Eine Attributangabe ersetzt den Standardtext (optional) **Beispiel:** `<PASSWORD refvar="plc/mw107" />` **Beispiel optional:** `<password refVar = "plc/MD108" text="Password" />`
POWER_OFF	Eine Meldung fordert den Bediener zum Ausschalten der Maschine auf. Der Meldetext ist fest im System hinterlegt.

Tag-Bezeichner	Bedeutung
PRINT	Das Tag gibt einen Text in der Dialogzeile aus oder kopiert den Text in die angegebene Variable. Enthält der Text Formatierungskennungen, werden die Werte der Variablen an den entsprechenden Stellen eingefügt. **Syntax:** `<PRINT name="Variablenname" text="text %Formatierung"> Variable, … </PRINT>` `<PRINT text="text %Formatierung"> Variable, … </PRINT>` **Attribute:** • **name** Name der Variablen, in die der Text abgelegt werden soll (optional) • **text** Text **Formatierung:** Das "%"-Zeichen führt zur Formatierung der als Wert angegebenen Variablen. %[Flags] [Breite] [.Nachkommastellen] Typ • Flags: Optionales Zeichen zum Festlegen der Ausgabeformatierung: – rechts- oder linksbündig ("-" für linksbündig) – Vornullen hinzufügen ("0") – mit Leerzeichen auffüllen • Breite: Das Argument legt die minimale Ausgabebreite einer nicht negativen Zahl fest. Besitzt der auszugebende Wert weniger Stellen als durch das Argument festgelegt, werden die fehlenden Stellen mit Leerzeichen aufgefüllt. • Nachkommastellen: Bei einer Gleitkommazahl legt der optionale Parameter die Anzahl von Nachkommastellen fest. • Typ: Das Typzeichen legt fest, welche Datenformate der Print–Anweisung übergeben wurden. Diese Zeichen müssen angegeben werden. – d: Integer Wert – f: Gleitkommazahl – s: String

Tag-Bezeichner	Bedeutung
PRINT Fortsetzung	**Werte:** Anzahl der Variablen, deren Werte in den Text eingefügt werden sollen. Die Variablentypen müssen mit der entsprechenden Typkennzeichnung der Formatierungsvorschrift übereinstimmen und sind durch Kommas voneinander zu trennen. **Beispiel:** Ausgabe eines Textes in der Informationszeile `<PRINT text="Infotext" />` Ausgabe eines Textes mit Variablenformatierung `<LET name="trun_dir"></LET>` `<PRINT text="M%d">trun_dir</PRINT>` Ausgabe eines Textes in einer Stringvariablen mit Variablenformatierung `<LET name="trun_dir"></LET>` `<LET name="str" type="string" ></LET>` `<print name="str" text="M%d ">trun_dir</print>`
PROGRESS_BAR	Das Tag öffnet oder schließt einen Fortschrittsbalken. Der Balken wir unterhalb des Applikationsfensters eingeblendet. **Syntax:** `<PROGRESS_BAR type="<true/false>"> value </ PROGRESS_BAR>` **Attribute:** • **type =** "TRUE" - öffnet den Fortschrittsbalken • **type** = "FALSE" - schließt den Fortschrittsbalken • **min** (optional) – minimaler Wert • **max** (optional) – maximaler Wert **Wert:** • **Value** Prozentuale Position des Balkens **Beispiel:** `<PROGRESS_BAR type="true" min="0" max="101" >20< /` `PROGRESS_BAR>......<PROGRESS_BAR >50< /` `PROGRESS_BAR>......<PROGRESS_BAR type="false" >100< /PROGRESS_BAR>`

Tag-Bezeichner	Bedeutung
SEND_MESSAGE	Das Tag sendet eine Nachricht mit zwei Parametern zur aktiven Form, die im Tag Message verarbeitet wird. **Syntax:** `<SEND_MESSAGE>p1, p2</SEND_MESSAGE>` **Beispiel:** `<SOFTKEY POSITION="3">` `  <caption>Set%nParameter</caption>` `  <send_message>1, 0</send_message>` `</SOFTKEY>` `<FORM>` … … `<MESSAGE>` `<SWITCH>` `<CONDITION>$message_par1</CONDITION>` `<CASE value="1">` … … `</CASE>` `</SWITCH>` `</MESSAGE>` … … `</FORM>`
SHOW_CONTROL	Mit Hilfe des Tags lässt sich die Sichtbarkeit eines Controls steuern. **Syntax:** `<SHOW_CONTROL name="<name>" type="<type>" />` **Attribute:** • **name** Name des Controls • **type =** "TRUE" - Control wird sichtbar • **type** = "FALSE" - Control wird unsichtbar (ausgeblendet) **Beispiel:** `<SHOW_CONTROL name="myEditfield" type="false" />` … … `<SHOW_CONTROL name="myEditfield" type="true" />`

Tag-Bezeichner	Bedeutung
SLEEP	Das Tag unterbricht die Abarbeitung des Skriptes für die angegebene Zeitspanne. Die Unterbrechungszeit ergibt sich aus dem übergebenen Wert multipliziert mit der Zeitbasis von 50ms. **Syntax:** `<SLEEP value="Unterbrechungszeit" />` **Beispiel:** Wartezeit 1.5 sec. `<SLEEP value="30" />`
STOP	Die Interpretation wird an dieser Stelle abgebrochen.
SWITCH	Die **SWITCH**-Anweisung beschreibt eine Mehrfachauswahl. Ein Ausdruck wird einmal ausgewertet und mit einer Anzahl Konstanten verglichen. Stimmt der Ausdruck mit der Konstanten überein, werden die Anweisungen innerhalb der **CASE**-Anweisung abgearbeitet. Die **DEFAULT**-Anweisung wird abgearbeitet, wenn keine Konstante mit dem Ausdruck übereinstimmt. **Syntax:** `<SWITCH>` `<CONDITION> Wert </CONDITION>` `<CASE value="<Konstante 1>">` `Anweisungen` `…` `</CASE>` `<CASE value="<Konstante 2>">` `Anweisungen` `…` `</CASE>` `<DEFAULT>` `Anweisungen` `…` `</DEFAULT>` `</SWITCH>`

Tag-Bezeichner	Bedeutung
SWITCHTOAREA	Das Tag SWITCHTOAREA wechselt aus dem Customer-Bereich in den angegebenen Bedienbereich. Der Parameter wird als Attributwert angegeben. **Syntax:** `<switchToArea name="area" args="argument "/>` **Attribute:** **name** Folgende Bedienbereichsnamen sind für die Bedienbereiche vereinbart: • **AreaMachine** - Maschine • **AreaParameter** - Parameter • **AreaProgramEdit** - Editor • **AreaProgramManager** - Programm-Manager • **AreaDiagnosis** - Diagnose • **AreaStartup** - Inbetriebnahme **args** (reserviert) Dem Bedienbereich können zum Aktivieren von Dialogen entsprechende Laufzeitargumente mitgegeben werden. **Beispiel:** `<switchToArea name="AreaMachine" />`
SWITCHTODYNAMICTARGET	Definiert sich der vorherige Dialog als dynamisches Sprungziel, aktiviert das Tag SWITCHTODYNAMICTARGET diesen Dialog und beendet die Skriptverarbeitung. **Syntax:** `<switchToDynamicTarget />`
THEN	Anweisung, wenn die Bedingung erfüllt wurde (**IF, THEN, ELSE**)
TYPE_CAST	Das Tag dient zur Datentypwandlung einer lokalen Variable. **Syntax:** <type_cast name="variable name" type=" new type" /> **Attribute:** • **name** Name der Variable • **type** Der Variable wird der neue Datentyp zugewiesen. • **convert** Der Variable wird der neue Datentyp zugewiesen. Zusätzlich erfolgt eine Konvertierung des Variablenwerts auf den neuen Datentyp.

Tag-Bezeichner	Bedeutung
TYPEDEF	Mit dem Tag kann ein neuer Bezeichner für einen Datentyp festgelegt werden. Dies hat für Strukturdefinitionen den Vorteil, dass man einmal den Datentyp definiert und diesen anschließend als Datentyp in einer LET-Anweisung verwenden kann. Als Attribute werden der Bezeichner und der Typ erwartet. Der Parser unterstützt lediglich das Festlegen von Strukturdefinitionen. Innerhalb der Typdefinition wird eine Variable durch das Tag „element" deklariert. Die Attribute des Tags entsprechen den Attributen der let Anweisung. `<typedef name="<Bezeichner>" type="struct">` `  <element name="<Name>" type="<Variablen Typ>" />` … … … `</typedef>` Nach dem Definieren kann der Bezeichner als Datentyp für die LET Anweisung verwendet werden. `<let name="<VariablenName>" type="<Bezeichner>"></let>` **Beispiel**: `<typedef name="my_struct" type="struct">` `  <element name="id" type="int" />` `  <element name="name" type="string" />` `  <element name="phone" type="string" />` `</typedef>` `<let name="info" type="my_struct"></let>` … … `<op>` `  info.id = 1;` `  info.name = _T"my name";` `  info.phone= _T"0034 45634";` `</op>`

Tag-Bezeichner	Bedeutung
TYPEDEF Fortsetzung	Einige vordefinierte Funktionen erwarten als Aufrufparameter Variablen vom Strukturtyp RECT, POINT oder SIZE. Diese Strukturen sind in der Datei struct_def.xml definiert. **Weitere Informationen** unter Kapitel "Datei struct_def.xml erstellen (Seite 141)" **RECT:** `<typedef name="StructRect" type="struct" > <element name="left" type="int">0</element> <element name="top" type="int">0</element> <element name="right" type="int">0</element> <element name="bottom" type="int">0</element> </typedef>` **POINT:** `<typedef name="StructPoint" type="struct" > <element name="x" type="int">0</element> <element name="y" type="int">0</element> </typedef>` **SIZE:** `<typedef name="StructSize" type="struct" > <element name="width" type="int">0</element> <element name="height" type="int">0</element> </typedef>`
UNLOCK_OPERATING_AREA	Aufheben der Bedienbereichswechselsperre **Syntax:** `<UNLOCK_OPERATING_AREA />`
WAITING	Das Tag wartet nach einem NC- oder Drive-Reset auf den Wiederanlauf der Komponente. **Attribute:** • **WAITINGFORNC** = "TRUE" - es wird auf den Wiederanlauf der NC gewartet • **WAITINGFORDRIVE** = "TRUE" - es wird auf den Wiederanlauf der Antriebe gewartet **Syntax:** `<WAITING WAITINGFORNC = "TRUE"/>` **Beispiel:** ... `<CONTROL_RESET resetnc = "true" resetdrive = "true"/> <WAITING waitingfornc = "true" waitingfordrive = "true" /> `...`

Tag-Bezeichner	Bedeutung
WHILE	While-Schleife `WHILE (Test)` `  Anweisung` **Syntax:** `<WHILE>` `<CONDITION>…</CONDITION>` `Anweisungen` `…` `</WHILE>` Die While-Schleife dient dazu, eine Abfolge von Anweisungen mehrfach auszuführen, solange eine Bedingung erfüllt ist. Diese Bedingung wird geprüft, bevor die Anweisungsfolge abgearbeitet wird. **Beispiel:** `<WHILE>` `<CONDITION> "plc/ib9" == 0 </CONDITION>` `<DATA name = "PLC/qb11"> 15 </DATA>` `</WHILE>`
XML_PARSER	Das Tag "XML_PARSER" kann zum Parsen von XML-Dateien verwendet werden. Der Parser interpretiert eine XML-Datei und ruft definierte Rückruffunktionen auf. Jede Rückruffunktion gehört zu einem vordefinierten Ereignis. Innerhalb dieser Funktion kann der Programmierer die XML-Daten verarbeiten. **Vordefinierte Ereignisse:** • start document Der Parser öffnet das Dokument und beginnt mit dem Parsen. • end document Der Parser schließt das Dokument. • start element Der Parser hat ein Element gefunden und erstellt eine Liste mit allen Attributen und Attributwerten. Diese Listen werden an die Rückruffunktion weitergeleitet. • end element Das Ende des Elements wurde gefunden. • characters Der Parser leitet alle Zeichen eines Elements weiter. • error Der Parser hat einen Syntaxfehler erkannt. Tritt ein Ereignis auf, ruft der Parser die Rückruffunktion auf und überprüft den Rückgabewert der Funktion. Liefert die Funktion den Wert "true" zurück, setzt der Parser den Prozess fort.

Tag-Bezeichner	Bedeutung
XML_PARSER Fortsetzung	**Schnittstellen** Der Wert des Attributs **name** enthält den Pfad zur XML-Datei. Um den Rückruffunktionen Ereignisse zuzuweisen, sind folgende Eigenschaften festzulegen: **Standard** startElementHandler endElementHandler charactersHandler **Optional** errorHandler documentHandler Der Wert eines Attributs definiert den Namen der Rückruffunktion. **Beispiel:** `<XML_PARSER name="f:\appl\xml_test.xml">` `<!-- standard handler -->` `<property startElementHandler="startElementHandler" />` `<property endElementHandler="endElementHandler" />` `<property charactersHandler="charactersHandler" />` `<!-- optional handler -->` `<property errorHandler="errorHandler" />` `<property documentHandler="documentHandler" />` `</XML_PARSER>`

Tag-Bezeichner	Bedeutung
XML_PARSER Fortsetzung	Zusätzlich liefert der Parser Variablen, damit die Rückruffunktionen auf die Ereignisdaten zugreifen können. **startElementHandler:** **Funktionsparameter** tag_name - Tagname num - Anzahl gefundener Attribute **Systemvariablen** $xmlAttribute String-Array mit der Anzahl von durch num angegebenen Elemente. $xmlValue String-Array, die den Attributwertebereich 0-num enthält. **Beispiel:** `<function_body name="startElementHandler" return="true" parameter="tag_name, num">` `  <print text="attribute name %s"> $xmlAttribute[0]</print>` `  <print text="attribute value %s"> $xmlValue[0]</print>` `</function_body>` **endElementHandler:** **Funktionsparameter** tag_name - Tagname **Beispiel:** `<function_body name="endElementHandler" parameter="tag_name">` `  <print text="name %s"> tag_name </print>` `</function_body>`

Tag-Bezeichner	Bedeutung
XML_PARSER Fortsetzung	**charactersHandler:** **Systemvariablen** $xmlCharacters String mit den Daten $xmlCharactersStart immer 0 $xmlCharactersLength Anzahl der Bytes **Beispiel:** `<function_body name="charactersHandler" return="true" >` `<print text="chars %s"> $xmlCharacters </print>` `</function_body>` **documentHandler:** **Funktionsparameter** state 1 start document, 2 end document **errorHandler:** **Systemvariablen** $xmlErrorString Enthält die ungültige Zeile (Systemvariable) **Beispiel:** `<function_body name="errorHandler" >` `<print text="error %s">$xmlErrorString</print>` `</function_body>` **Rückrufergebnis:** $return wenn 1 (true), setzt der Parser das Parsen der Datei fort

3.7.3 Farbcodierung

Das Attribut color benutzt das Farbcodierungsschema der HTML-Sprache.

Eine Farbangabe setzt sich syntaktisch aus dem Zeichen "#" (Raute) und sechs Ziffern des Hexadezimalsystems zusammen, wobei jede Farbe durch zwei Ziffern repräsentiert wird.

R – rot

G – grün

B – blau

#RRGGBB

Beispiel:

color= "#ff0011"

Beispielfarben:

rot	grün	blau	gelb	weiß	schwarz
#FF0000	#00FF00	#0000FF	#ffff00	#FFFFFF	#000000

3.7.4 Spezielle XML-Syntax

Zeichen, die bei der XML-Syntax eine besondere Bedeutung haben, müssen umschrieben werden, wenn diese von einem allgemeinen XML-Editor korrekt dargestellt werden sollen.

Folgende Zeichen sind betroffen:

Zeichen	Notation in XML
<	<
>	>
&	&
"	"
'	'

3.7.5 HTML-Sonderzeichen

Zur Darstellung von Zeichen oder zur Verwendung von Steuerzeichen wird die HTML-Sonderzeichenauswertung angeboten.

Es können die Dezimal- und Hexadezimal-Codierungen der Codepage Latin 1 verwendet werden.

Beispiel

Zeichen	Beschreibung	Notation dezimal	Notation hexadezimal
"	Anführungszeichen oben	"	"
LF	Line Feed

3.7.6 Operatoren

Die Operationsanweisung verarbeitet folgende Operatoren:

Operator	Bedeutung
=	Zuweisung
==	gleich
<, <	kleiner als
>, >	größer als
<=, <=	kleiner gleich
>=, >=	größer gleich
|	bitweise OR-Verknüpfung
||	logische OR-Verknüpfung
&, &	bitweise AND-Verknüpfung
&&, &&	logische AND-Verknüpfung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
!	Nicht
!=	ungleich

Operationsanweisungen werden von links nach rechts abgearbeitet. Unter Umständen kann das Klammern von Ausdrücken sinnvoll sein, um die Abarbeitungspriorität der Teilausdrücke festzulegen.

3.7.7 Systemvariablen

Systemvariablen sind Variablen, die in jedem Skript zur Verfügung stehen und zum Datenaustausch zwischen dem Parser und der Skriptausführung dienen.

Die nachfolgende Tabelle gibt einen Überblick, für welche Tags automatisch Variablen erzeugt werden:

Variablenname	Bedeutung	Gültig in Tag
$actionresult	Signalsiert dem Parser, ob der Event durch den Parser bearbeitet werden soll	KEY_EVENT
$focus_name	Beinhaltet den Namen des Feldes, welches den Eingabefokus besitzt	FOCUS_IN INDEX_CHANGED EDIT_CHANGED
$focus_item_data	Beinhaltet den numerischen Wert item_data, der dem Feld zugewiesen wurde	
$gestureinfo	Bei Fingergesten, stellt der Parser die Gesteninformationen in der Variablen bereit und führt das Tag gesture_event aus.	GESTURE_EVENT
$return	Die Variable dient zum Transport des Rückgabewertes einer Unterfunktion	FUNCTION_BODY
$message_par1 $message_par2	Beinhaltet die Aufrufparameter der Funktion SEND_MESSAGE	MESSAGE
$xmlAttribute	Beinhaltet eine Liste der gefundenen Tag-Attribute	XML_PARSER startElementHandler
$xmlValue	Beinhaltet eine Liste der gefundenen Tag-Werte	
$xmlCharacters	Beinhaltet den Datenstrom	XML_PARSER charactersHandler
$xmlCharactersStart	Beinhaltet den Startindex	
$xmlCharactersLength	Beinhaltet die Anzahl der im Datenstrom gespeicherten Zeichen	
$mouse_event.type $mouse_event.x $mouse_event.y $mouse_event.id $mouse_event.buttons $mouse_event.button	Strutur zur Übergabe der Parameter des Maus-Events	MOUSE_EVENT

3.7.8 Softkeymenüs und Dialogmasken erstellen

Zum Einbinden von Dialogmasken muss ein Menü-Tag mit dem Namen "main" in der XML-Beschreibung vorhanden sein. Dieses Tag wird vom System nach dem Aktivieren des Bedienbereichs <CUSTOM> aufgerufen. Innerhalb des Tags können weitere Dialogmasken verzweigt sowie das Aktivieren einer Dialogbox definiert werden.

```
<menu name= "MAIN">
<OPEN_FORM name= "main dialogue">
<softkey POSITION="1">
<caption>sub menu 1</caption>
<navigation>sub menu 1</navigation>
</softkey>
<softkey POSITION="8">
<caption>sub menu 8</caption>
<navigation>sub menu 8</navigation>
</softkey>
</menu>

<menu name= "sub menu 1">
<OPEN_FORM name= "dialogue 1">
</menu>

<menu name= "sub menu 8">
<OPEN_FORM name= "dialogue 8“>
</menu>
```

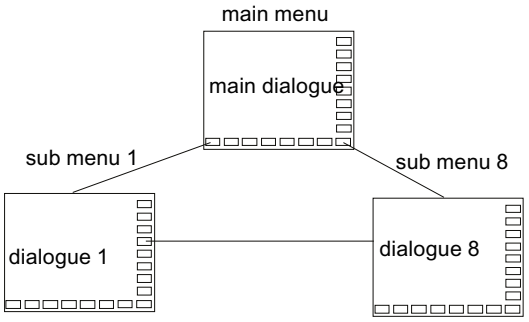
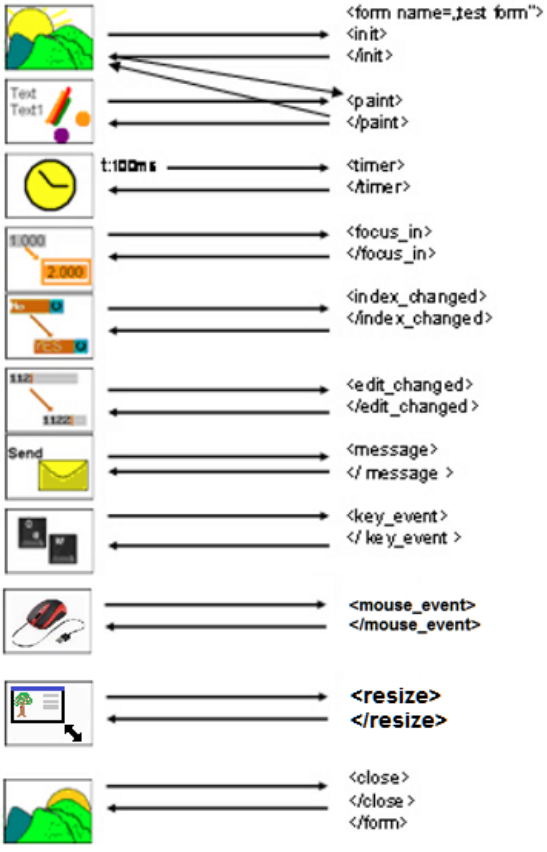


Bild 3-6 Menüstruktur

Tag-Bezeichner	Bedeutung
FORM	Das Tag beinhaltet die Beschreibung eines Anwenderdialogs. Die entsprechenden Tags sind im Kapitel Menüs und Dialogmasken erstellen beschrieben. **Syntax:** `<FORM name="<dialog name>" color="#ff0000">` **Attribute:** • **color** Hintergrundfarbe der Dialogmaske **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" – Default weiß • **name** Bezeichner der Form • **type** Zulässiger Wert ist *cycle* der eine Anwenderzyklenmaske kennzeichnet • **xpos** X-Position der linken oberen Ecke der Dialogbox (optional) • **ypos** Y-Position der linken oberen Ecke (optional) • **width** Ausdehnung in X-Richtung (in Pixel) (optional) • **height** Ausdehnung in Y-Richtung (in Pixel) (optional)
FORM Fortsetzung	Mit dem Attribut AUTOSCALE_CONTENT kann das Verhalten der Controls einer Form bei unterschiedlichen Bildschirmauflösungen festgelegt werden. Als Standard werden die Controls automatisch an die vorgefundene Bildschirmauflösung angepasst. Möchte man die Positionierung und Dimensionierung selbst steuern, ist im Form-Tag das Attribut mit dem Wert OFF anzugeben. **Attribut:** **autoscale_content** **Werte:** • **on** Die Koordinaten der Controls werden automatisch an die Bildschirmauflösung angepasst (Standard) • **off** Die Koordinaten der Controls werden unverändert verwendet **Beispiel:** `<form name="main_form" autoscale_content="off">` … … … `</form>`

Tag-Bezeichner	Bedeutung
FORM Fortsetzung	Attribute: • **textfile** Das Attribut ermöglicht das Angeben einer formbezogenen Textdatei. Als Standardtextkontext verwendet das System den Bezeichner EASY_XML. • **textcontext** Das Attribut ermöglicht das Angeben eines Bezeichners für den zu verwendenden Textkontext. Dieses Attribut ist im Zusammenhang mit dem Attribut **textfile** gültig.
FORM Fortsetzung	Zum Anwenden von Standard-Layouts bzw. Layouts, die in der Datei easyscreen.ini abgelegt sind, ist das Attribut LAYOUT zu verwenden. Als Wert ist der Name des zu verwendenden Layouts anzugeben. **Hinweis:** Eine Veränderung der Standardeinstellungen ist nur für Popup-Dialoge sinnvoll, da hier der aufrufende Dialog nicht abgeblendet wird. **Attribut:** **layout** **Syntax:** `<form name="Name der Form" layout="Name des Layouts" >` … … `</form>` **Beispiel:** `<form name="form0" layout="slstandardscreenlayout.` `SlStandardScreenLayout.LowerForm" >` … … `</form>` Neben den vordefinierten Maskenpositionen und Dimensionen ist es möglich, eigene Definitionen in der Datei easyscreen.ini abzulegen. Eintrag in easyscreen.ini: `[640x480]` `MyPanel = x:=0, y:=220, width:=340, height:=174` `[800x480]` `MyPanel = x:=0, y:=220, width:=420, height:=174` **Beispiel:** `<form name="form0" layout="MyPanel" >` … … `</form>`

Tag-Bezeichner	Bedeutung
FORM Fortsetzung	**Dialognachrichten:** • **INIT** • **PAINT** • **TIMER** • **CLOSE** • **FOCUS_IN** • **INDEX_CHANGED** • **EDIT_CHANGED** • **GESTURE_EVENT** • **KEY_EVENT** • **MESSAGE** • **MOUSE_EVENT** • **RESIZE** • **CHANNEL_CHANGED** • **LANGUAGE_CHANGED**
FORM Fortsetzung	

Tag-Bezeichner	Bedeutung
FORM Fortsetzung	**Syntax:** `<FORM name = "<dialog name>" color = "#ff0000">` **Beispiel:** `<FORM name = "R-Parameter">` ` <INIT>` ` <DATA_ACCESS type = "true" />` ` <CAPTION>R - Parameter</CAPTION>` ` <CONTROL name = "edit1" xpos = "322" ypos = "34" refvar = "nck/Channel/Parameter/R[1]" />` ` <CONTROL name = "edit2" xpos = "322" ypos = "54" refvar = "nck/Channel/Parameter/R[2]" />` ` <CONTROL name = "edit3" xpos = "322" ypos = "74"` ` </INIT>` `  <PAINT>` ` <TEXT xpos = "23" ypos = "34">R - Parameter 1</TEXT>` ` <TEXT xpos = "23" ypos = "54">R - Parameter 2</TEXT>` ` <TEXT xpos = "23" ypos = "74">R - Parameter 3</TEXT>` ` </PAINT>` `</FORM>`
INIT	**Dialogboxnachricht** Das Tag wird unmittelbar nach dem Erstellen der Dialogbox abgearbeitet. Hier sind alle Eingabeelemente sowie "Hotlinks" der Dialogmaske anzulegen.

Tag-Bezeichner	Bedeutung
KEY_EVENT	Dialognachricht Das Tag KEY_EVENT kann zum Bewerten von Tastaturereignissen in die Form eingebunden werden. Ist das Tag in einer Form vorhanden, sendet das System den MF2-Tastaturcode an die aktive Form. Wird die Variable **$actionresult** nicht auf Null gesetzt, verarbeitet anschließend das System das Tastaturereignis. Der Tastaturcode wird in der Variable **$keycode** als Integerwert bereitgestellt. **Beispiel:** Das in der Variable **exclude_key** eingegebene Zeichen soll aus dem Eingabestrom herausgefiltert werden. `<LET name="stream" type="string"/>` `<LET name="exclude_key" type="string"/>` `<FORM name = "keytest_form">` `  <INIT>` `  <CONTROL name = "p1" xpos = "120" ypos = "84" width ="200" refvar="stream" hotlink="true" />` `  <CONTROL name = "p2" xpos = "160" ypos = "104" width ="8" refvar="exclude_key" hotlink="true" />` `  </INIT>` `  <PAINT>` `  <text xpos = "8" ypos = "84">data stream</text>` `  <text xpos = "8" ypos = "104">exclude key</text>` `  </PAINT>` `  <KEY_EVENT>` `  <LET name="excl_keycode" type="string"/>` `  <OP>excl_keycode = exclude_key</OP>` `  <type_cast name="excl_keycode" type="int" />` `  <PRINT text="%d %d">$keycode, excl_keycode</PRINT>` `  <IF>` `  <CONDITION>$keycode == excl_keycode</CONDITION>` `  <THEN>` `  <op> $actionresult = 0</op>` `  </THEN>` `  </IF>` `  </KEY_EVENT>` `</FORM>`

Tag-Bezeichner	Bedeutung
MOUSE_EVENT	Das Tag kann zum Verarbeiten von Mouse-Events in das Skript eingebunden werden. Es wird ausgeführt, wenn folgende Aktivitäten mit der Maus ausgeführt wurden: • Eine Taste wurde gedrückt • Eine Taste wurde losgelassen • Die Maus wurde bewegt Der Parser stellt die Informationen in einer Struktur bereit und legt die Strukturvariable **$mouse_event** mit folgenden Elementen an: **Strukturelemente:** • **type** Kodierung der Aktivität – 2 - Eine Taste wurde gedrückt – 3 - Eine Taste wurde losgelassen – 5 - Die Maus wurde bewegt • **x** X-Position des Cursors in Pixel; bezogen auf die aktuelle Bildschirmauflösung • **y** Y-Position des Cursors in Pixel; bezogen auf die aktuelle Bildschirmauflösung • **id** Bezeichner – -1, wenn die Position keinem Control zugeordnet werden kann – != -1, befindet sich der Maus-Cursor innerhalb eines Controls, wird der Inhalt des Attributs **idemdata** geliefert • **button** Beinhaltet den Zustand der Tasten, zum Zeitpunkt des Events – 0 - keine Taste – 1 - linke Taste – 2 - rechte Taste – 4 - mittlere Taste Die Tasten können mit einer bitweisen ODER-Operation verknüpft sein. **Beispiel:** `<MOUSE_EVENT>` `  <print text="button %d type %d x %d y %d ">$mouse_event.button, $mouse_event.type, $mouse_event.x, $mouse_event.y</print>` `</MOUSE_EVENT>`

Tag-Bezeichner	Bedeutung
MESSAGE	**Dialognachricht** Wird im Skript die Anweisung Send_message ausgeführt, arbeitet der Parser das Tag Message ab. Es werden die Werte p1 und p2 in den Variablen $message_par1 und $message_par2 bereitgestellt (siehe Tag "SEND_MESSAGE"). **Syntax:** `<MESSAGE>` `</MESSAGE>` **Beispiel:** `<LET name="user_selection" />` `<SOFTKEY POSITION="3">` `<CAPTION>Set%nParameter</CAPTION>` `<SEND_MESSAGE>1, 10</SEND_MESSAGE>` `</SOFTKEY>` … … `<FORM>` … … `<MESSAGE>` `<SWITCH>` `<CONDITION>$message_par1</CONDITION>` `<CASE value="1">` `<OP> user_selection = $message_par2 </OP>` … … `</CASE>` `</SWITCH>` `</MESSAGE>` … … `</FORM>`

Tag-Bezeichner	Bedeutung
SEND_MESSAGE	Das Tag sendet eine Nachricht mit zwei Parametern zur aktiven Form, die im Tag Message verarbeitet wird (siehe auch MESSAGE). **Syntax:** `<SEND_MESSAGE>p1, p2</SEND_MESSAGE>` **Beispiel:** `<LET name="user_selection" />` `<SOFTKEY POSITION="3">` `  <CAPTION>Set%nParameter</CAPTION>` `  <SEND_MESSAGE>1, 10</SEND_MESSAGE>` `</SOFTKEY>` … … `<FORM>` … … `<MESSAGE>` `<SWITCH>` `<CONDITION>$message_par1</CONDITION>` `<CASE value="1">` `<OP> user_selection = $message_par2 </OP>` … `</CASE>` `</SWITCH>` `</MESSAGE>` … … `</FORM>`

Tag-Bezeichner	Bedeutung
RESIZE	**Dialogboxnachricht** Das Tag kann zum Verarbeiten eines RESIZE-Events in das Skript eingebunden werden. Dieses Ereignis wird durch eine dynamische Auflösungsumschaltung erzeugt. 

Tag-Bezeichner	Bedeutung
RESIZE Fortsetzung	**Beispiel:** `<let name="screen_size" type="StructSize" />` `<form name="menu_userscale_form" autoscale_content="off">` ` <init>` ` <caption>Use auto scaling</caption>` ` <function name="hmi.get_hmi_resolution">screen_size</function>` ` <print text="screen size: w %d h: %d">screen_size.width, screen_size.height</print>` ` <data_access type="true" />` ` <control name="s_c" xpos="8" ypos="140" fieldtype="readonly" width="500" refvar="string_var" hotlink="true"/>` ` <if>` ` <condition>screen_size.width == 800</condition>` ` <then>` ` <control name="s_1" xpos="508" ypos="140" fieldtype="readonly" width="100" refvar="string_var1" hotlink="true"/>` ` </then>` ` </if>` ` <control name="s_w" xpos="200" ypos="350" fieldtype="readonly" refvar="screen_size.width" hotlink="true"/>` ` <control name="s_h" xpos="200" ypos="370" fieldtype="readonly" refvar="screen_size.height" hotlink="true"/>` ` </init>` ` <paint>` ` <text xpos ="68" ypos="310">Text</text>` ` <text xpos ="8" ypos="350">screen width</text>` ` <text xpos ="8" ypos="370">screen height</text>` ` </paint>` ` <resize>` ` <function name="hmi.get_hmi_resolution">screen_size</function>` ` <print text="resize_event screen size: w %d h: %d">screen_size.width, screen_size.height</print>` ` <switch>` ` <condition>screen_size.width</condition>` ` <case value="640">` ` <function name="control.delete">_T"s_1"</function>` ` </case>` ` <case value="800">` ` <control name="s_1" xpos="508" ypos="140" fieldtype="readonly" width="100" refvar="string_var1" hotlink="true"/>` ` </case>` ` </switch>` ` </resize>` `</form>`

Tag-Bezeichner	Bedeutung
CHANNEL_CHANGED	**Dialognachricht** Das Tag ermöglicht die Bearbeitung des Kanalwechsel-Events. Es wird ausgeführt, wenn der Anwender die Kanalvorschalttaste betätigt. **Syntax:** `<channel_changed>` `</channel_changed>` **Beispiel:** Bei einem Kanalwechsel soll die Form geschlossen und wieder geöffnet werden, um die Daten des angewählten Kanals anzeigen zu können. `<channel_changed>` `  <open_form name = "form1" reopen="true"/>` `</channel_changed>` oder Bei einem Kanalwechsel wird ein anderes Menü aktiviert. `<channel_changed>` `  <navigation>menu2</navigation>` `</channel_changed>`
LANGUAGE_CHANGED	**Dialognachricht** Das Tag ermöglicht die Bearbeitung einer Sprachumschaltungs-Anforderung. Es wird ausgeführt, wenn der Anwender die Sprache bei geöffneter Dialogmaske wechselt. **Syntax:** `<language_changed>` `</language_changed>` **Beispiel:** Bei einem Sprachwechsel soll die Form geschlossen und wieder geöffnet werden, um die Controls mit den Textelementen der aktivierten Sprache versorgen zu können. `<language_changed>` `  <open_form name = "form1" reopen="true"/>` `</language_changed>`
FOCUS_IN	**Dialogboxnachricht** Das Tag wird aufgerufen, wenn das System den Focus auf ein Control setzt. Zur Identifikation des Controls kopiert das System den Namen des Controls in die Variable **$focus_name** und den Wert des Attributes **item_data** in die Variable **$focus_item_data**. Diese Nachricht kann z. B. benutzt werden, um Bilder in Abhängigkeit der Focus-Position ausgeben zu können. **Beispiel:** `<focus_in>` `<PRINT text="focus on filed:%s, %d">$focus_name,` `$focus_item_data </PRINT>` `</focus_in>`

Tag-Bezeichner	Bedeutung
PAINT	**Dialogboxnachricht** Das Tag wird mit dem Aufblenden der Dialogbox abgearbeitet. Hier sind alle Texte und Bilder anzugeben, die die Dialogbox anzeigen soll. Weiterhin wird das Tag ausgeführt, wenn das System feststellt, dass Teile der Dialogbox neu anzuzeigen sind. Dieses kann z.B. durch das Schließen von überlagernden Fenstern initiiert werden.
TIMER	**Dialogboxnachricht** Das Tag wird zyklisch abgearbeitet. Jeder Form ist ein Timer zugeordnet, der ca. aller 100 ms die Abarbeitung des Timer – Tags anstößt.
CAPTION	Das Tag beinhaltet den Titel der Dialogbox. Dieses Tag ist innerhalb des INIT-Tags zu verwenden. Die Titelzeile kann in mehrere Spalten unterteilt werden. Zur Unterteilung der Titelzeile ist vor dem Ausgeben des Textes das Tag caption mit dem Attribut define_section zu programmieren. Das Attribut define_section legt die Anzahl Spalten fest Mit dem Attribut property wird für jede Spalte die Länge, die Startposition und die Textausrichtung definiert. Die Position- und Längenangabe ist ein prozentualer Wert, der sich auf die Breite der Form bezieht. Der Wert des Attributs value indiziert die Spaltennummer (mit Null beginnend) `<caption define_sections="3">` `<property value="0" length="20" position="2" alignment="left" />` `<property value="1" length="20" position="79" alignment="right" />` `</caption>` Anschließend kann der Text der Spalte zugewiesen werden, indem man zusätzlich das Attribut index mit dem Spaltenindex angibt. `<caption index="0">Spalte1</caption>` `<caption index="1">Spalte2</caption>` **Syntax:** `<CAPTION>Titel</CAPTION>` **Beispiel:** `<CAPTION>my first dialogue</CAPTION>`
CLOSE	**Dialogboxnachricht** Vor dem Schließen der Dialogbox wird dieses Tag abgearbeitet.

Tag-Bezeichner	Bedeutung
CLOSE_FORM	Das Tag schließt den aktiven Dialog. Diese Anweisung ist nur notwendig, wenn der Dialog durch den MMC-Befehl geöffnet wurde und dem Bediener eine Softkey-Funktion zum Schließen des Dialogs angeboten wird. Generell werden Dialoge automatisch verwaltet und müssen nicht explizit geschlossen werden. **Syntax:** `<CLOSE_FORM/>` **Beispiel:** `<softkey_ok>` ` <caption>OK</caption>` ` <CLOSE_FORM />` ` <navigation>main_menu</navigation>` `</softkey_ok>`

Tag-Bezeichner	Bedeutung
CONTROL	Das Tag dient zum Erstellen von Steuerelementen. **Syntax:** `<CONTROL name = "<control name>" xpos = "<X-Position>" ypos = "<Y-Position>" refvar = "<NC-Variable>" hotlink = "true" format = "<Format>" />` **Hinweis:** Variablen, die zum Setzen der Attributwerte verwendet werden, müssen als globale Variablen deklariert werden. **Attribute:** • **name** Bezeichner des Feldes. Der Bezeichner stellt gleichzeitig eine lokale Variable dar und darf in der Form nicht mehrfach verwendet werden. • **xpos** X-Position der linken oberen Ecke • **ypos** Y-Position der linken oberen Ecke • **fieldtype** Feldtyp Ist kein Typ angegeben, wird das Feld als Edit-Feld eingerichtet. – **edit** Daten können verändert werden – **readonly** Daten können nicht verändert werden **combobox** Anstelle von Zahlenwerten zeigt das Feld entsprechende Bezeichner. Wird der Feldtyp **combobox** ausgewählt, sind zusätzlich die darzustellenden Ausdrücke dem Feld zuzuordnen. Dafür ist das Tag <ITEM> zu verwenden. Die Combo Box speichert den Index des aktuell ausgewählten Textes in der zum Control gehörenden Variablen (siehe Attribut **refvar**). Nach dem Anlegen des Controls lassen sich weitere Elemente mit den Funktionen **addItem** oder **insertItem** einfügen. – **progressbar** Es erfolgt die Darstellung eines Fortschrittsbalkens im Wertebereich von 0 bis 100. Der Wertebereich lässt sich mit den Eigenschaften Minimal- und Maximalwert an das darzustellende Datum anpassen.

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	• **fieldtype edit** und **readonly** Bei dem Feldtyp **edit** und **readonly** ist mit dem Attribut **multiline** ein mehrzeiliges Anzeigen von Texten möglich. Es erfolgt der Zeilenumbruch an einer Wortgrenze oder mit dem Line Feed-Zeichen <LF>. **Beispiel:** `<control name="mulitline_edit" xpos="280" ypos="60" width="200" height="100" type="string">` `  <property multiline="true" />` `</control>` `<control name="c2" xpos="280" ypos="260" width="200" height="100" type="string" fieldtype="readonly">` `  <property multiline="true" />` `</control>`

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	• **fieldtype** – **graphicbox** Der Feldtyp erzeugt ein 2d-Strichgrafik-Control. Mit dem Tag <ITEM> kann ein grafisches Element in das Control eingefügt werden. Die Parameter **width** und **height** geben die Breite und Höhe der Box an. Nach dem Anlegen des Controls lassen sich weitere Elemente mit den Funktionen **addItem** oder **insertItem** einzufügen. Der Parameter **itemdata** wird bei diesem Control nicht ausgewertet. Wird dem Control eine Referenzvariable zugewiesen, erfolgt das Aufzeichnen ihrer Werte im 100 ms-Raster. Maximal lassen sich bis zu 10000 Werte aufzeichnen. Mit der Eigenschaft **max** erfolgt ein endloses Aufzeichnen der Werte, wobei mit dem Erreichen der maximalen Aufzeichnungsdauer der jeweils älteste Wert gelöscht wird. Die Aufzeichnungsdauer ist in Millisekunden anzugeben. Die zeitdiskret gespeicherten Werte stellt das Control als miteinander verbundene Linien dar. Das Attribut **channel** ermöglicht das Aufzeichnen von bis zu drei weiteren Referenzvariablen. Der Index beginnt mit eins. Index null dient zum Setzen der Eigenschaften der im Control-Tag zugewiesenen Variablen. **Beispiel:** `<control name= "c_gbox1" xpos = "250" ypos="24" width="240" height="356" fieldtype="graphicbox" refvar="val" hotlink="true" COLOR_BK="#ffffff">` `  <property max="60000" /> <!-- steps of ms -->` `  <property min="0" />` `  <property channel="0" color="#00ffff" penwidth="3"/>` `  <property ORDINATE="Y sinus" />` `  <property ABSCISSA="time *100 ms" />` `  <property channel="1" refvar="val2" color="#000000" />` `</control>` `<request name="nck/Channel/Parameter/R[2]" />` `<control name= "c_gbox" xpos = "0" ypos="5"  width="260" height="200" fieldtype="graphicbox" refvar="nck/Channel/Parameter/R[1]" hotlink="true" COLOR_BK="#ffffff">` `  <property max="400" />` `  <!-- steps of  ms  -->` `  <property min="0" />` `  <property channel="0" color="#00ff00" penwidth="1"/>` `  <property ORDINATE="Power" />` `  <property ABSCISSA="Time *100 ms" />` `  <property channel="1" refvar="nck/Channel/Parameter/R[2]" color="#FF0000" penwidth="1"/>` `  <property FIX_TIME_AREA="on" />` `</control>`

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Beispiel graphicbox:** `<CONTROL name= "graphic" xpos = "8" ypos="23" width="300" height="352" fieldtype="graphicbox" />` • Hinzufügen von Elementen: Elemente werden mit der Funktion **additem** oder **loaditem** hinzugefügt. Folgende 2d-Elemente können benutzt werden: – Linie - l(inc) – Kreissektor - c(ircle) – Punkt - p(oint) **Aufbau eines Elementes:** <Elementtyp>; Koordinaten • Linie: l; xs; ys; xe, ye I - Kennzeichnung Linie Xs - X-Startposition Ys - Y-Startposition Xe - X-Endposition Ye - Y-Endposition • Circle: C, xs, ys, xe, ye, cc_x, cc_y, r C - Kennzeichnung Kreissektor Xs - X-Startposition Ys - Y-Startposition Xe - X-Endposition Ye - Y-Endposition Cc_x - X-Koordinate Kreismittelpunk CC_y - Y-Koordinate Kreismittelpunk • Radius: R • Point: P, x, y P - Kennzeichnung Punkt X - X-Position Y - Y-Position • Graphik löschen: Der Inhalt wird mit der Funktion **empty** gelöscht.

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	Folgende Feldtypen werden im Kapitel "Eigene Buttons projektieren (Seite 213)" beschrieben. • **fieldtype** – **pushbutton** Der Feldtyp wird als Taster oder Schalter eingesetzt. – **radiobutton** Der Feldtyp ermöglicht das Auswählen einer aus mehreren Optionen. – **checkbox** Der Feldtyp ermöglicht das Auswählen mehrerer Optionen. – **groupbox** Der Feldtyp umschließt optisch eine Gruppe von Controls mit einem Rahmen und einem Titel. – **switch** Der Feldtyp ist ein graphisches Element, das einen von zwei Zuständen durch eine Ikone signalisiert. – **scrollarea** Der Feldtyp wird verwendet, um Controls innerhalb eines festgelegen Bereichs anzuzeigen.

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	• **fieldtype** – **listbox** Der Feldtyp erzeugt ein leeres Listbox-Control. Mit dem Tag <ITEM> kann ein Listboxelement in die Listbox eingefügt werden. Das ITEM-Attribut **value** ermöglicht, diesem Element einen eindeutigen Wert zu zuweisen. Dieser kann z. B. zur Identifikation des Elementes dienen. Die Parameter **width** und **height** geben die Breite und Höhe der Listbox an. Nach dem Anlegen des Controls lassen sich weitere Listboxelemente mit den Funktionen **addItem**, **insertItem** oder **loadItem** einfügen. Nach dem Anlegen des Controls lassen sich weitere Listboxelemente mit der Funktion **addItem** oder **insertItem** einfügen. Der Parameter **itemdata** wird bei diesem Control nicht ausgewertet. – **itemlist** Der Feldtyp erzeugt ein Static-Control, das Anstelle von Zahlenwerten entsprechende Bezeichner anzeigt. Mit dem Tag <ITEM> kann dem Feld ein Bezeichner zugewiesen werden. • **item_data** Dem Attribut kann ein benutzerspezifischer Integerwert zugewiesen werden. Dieser Wert wird der FOCUS_IN - Nachricht zur Identifikation des Focus-Feldes mitgegeben. • **refvar** Bezeichner der Referenzvariablen, die mit dem Feld verknüpft werden kann (optional). • **hotlink** = "TRUE" " Ändert sich der Wert der Referenzvariablen, wird das Feld automatisch aktualisiert (optional). • **format** Das Attribut definiert das Anzeigeformat der angegebenen Variablen. Formatierungsangabe siehe print-Tag (optional). Das Attribut **format** ermöglicht auch das Darstellen eines Integerwertes in anderen Zahlensystemen, bzw. das Darstellen als Boole'schen Wert für die Feldtypen **edit** und **readonly**. Eine Formatierung über Zeichenanzahl und Ausrichtung wird nicht unterstützt. Folgende Darstellungen können verwendet werden: • **%o** - octal • **%x** - hexdecimal • **%n** - binar • **%b** - bool
CONTROL Fortsetzung	**Beispiel zum Anlegen von Spalten, Attriput property:** `<control name="lb1" xpos = "10" ypos = "94" width="200" height="250" fieldtype="listbox" refvar="tmp" hotlink="true">` `<property columns="4" />` `<property column="0" type="width">190</property>` `<property column="1" type="width">100</property>` `<property column="2" type="width">190</property>` `<property column="3" type="width">16</property>` `</control>`

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Beispiel Attribut format:** `<let name="val_test">255</let>` `<form name="main_form">` `<init>` `<caption>bool hex octal bin display</caption>` `<control name="test_oVal" xpos="250" ypos="60" refvar="val_test" hotlink = "true" />` `<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>` `<control name="test_hex" xpos="250" ypos="120" refvar="val_test" hotlink = "true" format="%x"/>` `<control name="test" xpos="250" ypos="150" refvar="val_test" hotlink = "true" format="%o"/>` `<control name="test_b" xpos="200" ypos="180" width="300" refvar="val_test" hotlink = "true" format="%n"/>` `</init>` `<paint>` `<text xpos="16" ypos="60">integer value</text>` `<text xpos="16" ypos="90">boolean display</text>` `<text xpos="16" ypos="120">hexadecimal display</text>` `<text xpos="16" ypos="150">octal display</text>` `<text xpos="16" ypos="180">binary display</text>` `</paint>` `</form>`
CONTROL Fortsetzung	**Attribute:** • **color_bk** Das Attribut setzt die Hintergrundfarbe des Controls. • **color_fg** Das Attribut setzt die Vordergrundfarbe des Controls. **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" • **display_format** Das Attribut definiert das Verarbeitungsformat der angegebenen Variablen. Dieses Attribut muss bei einem Zugriff auf eine PLC – Float Variable angewendet werden, da der Zugriff über das Lesen eines Doppelwortes erfolgt. Folgende Datenformate sind zulässig: – FLOAT – INT – DOUBLE – STRING Ausdrücke (z. B. anzuzeigender Text oder Grafikelement) einer Listbox, Grafikbox oder Combobox zuweisen: **Syntax:** <ITEM>Ausdruck</ITEM> <ITEM value ="<Wert>">Ausdruck</ITEM>

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Beispiel:** `<CONTROL name = "button1" xpos = "10" ypos = "10" fieldtype = " combobox ">` `<ITEM>text1</ITEM>` `<ITEM>text2</ITEM>` `<ITEM>text3</ITEM>` `<ITEM>text4</ITEM>` `</CONTROL>` Soll ein beliebiger Integer-Wert einem Ausdruck zugeordnet werden, ist das Attribut **value = "Wert"** dem Tag hinzuzufügen. Anstelle der fortlaufenden Nummerierung enthält jetzt die Control-Variable den zugewiesenen Wert des Items. **Beispiel:** `<CONTROL name = "button1" xpos = "10" ypos = "10" fieldtype = " combobox ">` `<ITEM value = "10">text1</ITEM>` `<ITEM value = "20">text2</ITEM>` `<ITEM value = "12">text3</ITEM>` `<ITEM value = "1">text4</ITEM>` </CONTROL> **Beispiel Fortschrittsbalken:** `<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/ GeometricAxis/actProgPos[1]">` `<PROPERTY min = "0" />` `<PROPERTY max = "1000" />` `</CONTROL>` **Beispiel Listbox:** `<let name="item_string" type="string"></let>` `<let name="item_data" ></let>` `<CONTROL name="listbox1" xpos = "360" ypos="150" width="200" height="200" fieldtype="listbox" />` • Hinzufügen von Elementen: Elemente werden mit der Funktion **additem** oder **loaditem** hinzugefügt. • Inhalt löschen: Der Inhalt wird mit der Funktion **empty** gelöscht. `<op> item_string = _T"text1\\n" </op>` `<function name="control.additem">_T"listbox1", item_string, item_data </function>` `<op> item_string = _T"text2\\n" </op>` `<function name="control.additem">_T"listbox1", item_string, item_data </function>`

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Beispiel itemlist:** `<CONTROL name = "itemlist1" xpos = "10" ypos = "10" fieldtype = "itemlist">` `<ITEM value = "10">text1</ITEM>` `<ITEM value = "20">text2</ITEM>` `<ITEM value = "12">text3</ITEM>` `<ITEM value = "1">text4</ITEM>` `</CONTROL>`
CONTROL Fortsetzung	Das Control Imagebox verwaltet eine Grafik im Bitmap- oder GIF-Format. Ist die Grafik größer als der darstellbare Bereich, blendet das Control Scrollbalken ein. Zum Steuern des sichtbaren Bereichs, stellt das System die Funktion CONTROL.IMAGEBOXSET bereit. **Weitere Informationen** unter Kapitel "Vordefinierte Funktionen (Seite 157)" **Syntax:** `<function name="control.imageboxget"> ... </function>`
CONTROL Fortsetzung	**Attribute:** • **disable** Das Attribut sperrt/ermöglicht die Eingabe in ein Edit-Control. • **tooltip** Ein Hinweistext wird angezeigt, wenn der Cursor auf das Control gesetzt wird. • **factor** Umrechnungsfaktor • **font** Angabe einer Schriftart • **fontpixelsize** Zeichenhöhe - Der Wert ist in Anzahl Pixel anzugeben und bezieht sich auf eine Auflösung von 640x480 Pixel. Die dargestellte Zeichenhöhe wird aus dem Verhältnis der realen Auflösung zur Referenzauflösung bestimmt. **Beispiel:** `<control name="c2" xpos="180" ypos="120" fontname="arial" fontpixelsize="26" height="100" type="string" />`

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Attribut:** • **fontname** Das Attribut dient zum Festlegen des zu verwendenden Schriftfonts. Die installierten Fonts können über die Funktion HMI.GET_FONT_FAMILIES ermittelt werden. **Wert:** Name des Fonts Folgende Siemens-Schriftfonts sind im System installiert: • Siemens AD CH Simplified • Siemens AD CH Traditional • Siemens AD JP • Siemens AD KS • Siemens AD Mono • Siemens AD Mono CH Simplified • Siemens AD Mono CH Traditional • Siemens AD Mono JP • Siemens AD Mono KS • Siemens AD Mono TH • Siemens AD Mono VN • Siemens AD Sans • Siemens AD TH • Siemens AD VN • Siemens Logo • Siemens Sans Cond Global • Siemens Sans Cond Mono • Siemens Sans Global MS-Windows basierende Systeme können weitere Schriftfonts enthalten. **Beispiel:** `<control name="c2" xpos="180" ypos="120" fontname="arial" fontpixelsize="26" height="100" type="string" />`

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Control nach der Erstellung ändern** Ein Control-Tag kann verwendet werden, um die Control-Eigenschaften eines bestehenden Controls nach der Erstellung zu ändern. Das Tag muss mit dem Namen des zu ändernden Controls und den neuen Eigenschaften angegeben werden. Es kann nur innerhalb eines Form-Tags ausgeführt werden. Folgende Eigenschaften können z. B. verändert werden: • name • xpos • ypos • width • height • color_bk • color_fg • access level • fieldtype • itemdata • min • max • default • disable • tooltip • font • factor Die Referenzvariable kann nicht verändert werden. Wenn eine Eigenschaft per Triggerung durch ein Softkey-Ereignis verändert werden soll, ist der Tag send message zu verwenden, um diese Anforderung in den Form-Kontext zu übertragen. Das Tag message wird verwendet, um die Nachricht zu erfassen.

Tag-Bezeichner	Bedeutung
CONTROL Fortsetzung	**Control-Änderung in einer Operationsanweisung** Eine weitere Möglichkeit Eigenschaften eines Controls zur Laufzeit zu ändern ist, die Änderung in einer Operationsanweisung vorzunehmen. Hierfür ist der Name des Controls und die Eigenschaft anzugeben, der ein neuer Wert zugeordnet werden soll. Die Eigenschaft wird getrennt durch einen Punkt vom Control-Namen angegeben. **Syntax:** <Name>.<Eigenschaft> **Beispiel:** `...` `...` `<let name="value" />` `<let name="w" />` `<let name="h" />` `...` `...` `<control name="c_move" xpos="$xpos" ypos="124" />` `...` `...` `<op>` `c_move.xpos = 300;` `value = c_move.xpos;` `h = c_move.height;` `w = c_move.width;` `</op>`

Tag-Bezeichner	Bedeutung
HELP_CONTEXT	Dieses Tag legt das aufzurufende Hilfethema fest. Es ist im INIT Block zu programmieren. **Systeme 808/802D sl** Der im Attribut angegebene Name wird mit Präfix XmlUserDlg_ ergänzt und an das Hilfesystem weitergeleitet. Der damit verbundene Aufbau der Hilfedatei, ist dem Thema Online-Hilfe erstellen zu entnehmen. **Ablauf beim Aktivieren des Hilfesystems:** 1. Drücken Taste "Info". 2. Dialog liefert den Ausdruck "my_dlg_help". 3. Parser wandelt den Ausdruck in "XmlUserDlg_my_dlg_help" um. 4. Aktivierung des Hilfesystems. 5. Übergabe des Suchbegriffes "XmlUserDlg_my_dlg_help". **Systeme 840D sl/828D** **Weitere Informationen** über den Aufbau und die Erzeugung eines Hilfebuchs: Inbetriebnahmehandbuch SINUMERIK Operate **Attribute:** • **name** Der im Attribut angegebene Name wird an das Hilfesystem weitergeleitet. Es ist der im Hilfebuch im Tag **entry** verwendete Dateiname anzugeben. • **anchor** Das Attribut ermöglicht die Angabe des Schlüsselwortes. **Syntax:** `<HELP_CONTEXT name="<file name>" anchor="key word"/>` **Beispiel:** `<form name = "form1">` `<init>` `<help_context name="chapter_1.html" anchor="Keyword_1" />` `...` `...` `<control name = "cpwd" xpos = "322" ypos = "64" >` `<property password="true" />` `<property help_context="chapter_1.html" anchor="Keyword_2" />` `</control>` `</form>`

Tag-Bezeichner	Bedeutung
DATA_ACCESS	Das Tag steuert das Verhalten der Dialogmasken beim Speichern von Benutzereingaben. Das Verhalten ist innerhalb des INIT-Tags festzulegen. Wird das Tag nicht verwendet, erfolgt immer das Zwischenspeichern der Eingaben. Ausnahme: Controls, bei denen das Attibut **hotlink** ist auf **true** gesetzt ist, werden immer direkt geschrieben und gelesen. **Attribut:** • **type** = "TRUE" – es erfolgt kein Zwischenspeichern der Eingabewerte. Die eingegebenen Werte kopiert die Dialogmaske direkt in die Referenzvariablen. • **type =** "FALSE" – die Werte werden erst mit dem Tag **UPDATE_CONTROLS** type = "FALSE" in die Referenzvariable kopiert **Beispiel:** `<DATA_ACCESS type = "true" />`

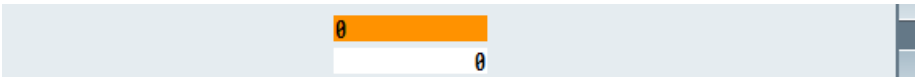
Tag-Bezeichner	Bedeutung
DEBUG	Das Tag dient zum Speichern von Trace-Ausgaben, die im Skript programmiert werden können. Zur Beschreibung des zu speichernden Textes ist das Attribut **text** zu verwenden. Dieses Attribut und dessen Werte entsprechen dem **text**-Attribut und den Werten der Print-Anweisung. Standardmäßig wird ein Debug-Tag nicht ausgeführt, da dieses zur Verlangsamung des Systems führt. Möchte man die Debug-Ausgaben aktivieren, ist im Tag **DialogGui** bzw. **AGM** das Attribut **debug_msg** mit dem Wert **on** zu programmieren. Das Speichern der Debug-Ausgaben erfolgt zusammen mit den Parser-Fehlermeldungen in folgender Datei: card/oem/sinumerik/hmi/dvm/log/dvm.log **Syntax:** `<debug text="<Text siehe Print Anweisung>">Variablen siehe Print-Anweisung</debug>` **Beispiel Easy XML:** `<DialogGui debug_msg="on">` `...` `...` `<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>` `<debug text="main_form control test_bool created value:%d">test_bool</debug>` Eintrag in der Datei dvm.log: `date: 10/22/ 2018 time: 09:38:44:351` `Debug xmldial.xml 17: main_form control test_bool created value:255` **Beispiel Easy Extend:** `<AGM debug_msg="on">` `...` `...` `<control name="test_bool" xpos="250" ypos="90" refvar="val_test" hotlink = "true" format="%b"/>` `<debug text="main_form control test_bool created value:%d">test_bool</debug>` Eintrag in der Datei dvm.log: `date: 10/22/ 2018 time: 09:38:44:351` `Debug xmldial.xml 17: main_form control test_bool created value:255`

Tag-Bezeichner	Bedeutung
EDIT_CHANGED	**Dialogboxnachricht** Das Tag wird aufgerufen, wenn sich der Inhalt eines Edit-Controls geändert hat. Zur Identifikation des Controls kopiert das System den Namen des Controls in die Variable **$focus_name** und den Wert des Attributes **item_data** in die Vatiable **$focus_item_data**. **Beispiel:** `<EDIT_CHANGED>` `<print text="index changed filed:%s, %d"> $focus_name, $focus_item_data </print>` … … `</EDIT_CHANGED>`
GESTURE_EVENT	Das Tag wird zur Ausführung von Fingergesten für die Multitouch-Bedienung verwendet. Das Tag wird im Kapitel "Multitouch-Bedienung (Seite 205)" beschrieben.
INDEX_CHANGED	**Dialogboxnachricht** Das Tag wird aufgerufen, wenn der Bediener die Auswahl einer Combobox ändert. Zur Identifikation des Controls kopiert das System den Namen des Controls in die Variable **$focus_name** und den Wert des Attributes **item_data** in die Vatiable **$focus_item_data**. **Hinweis:** Eine, dem Control zugewiesene Referenzvariable, ist zu diesem Zeitpunkt noch nicht mit der Control-Variablen abgeglichen worden und enthält den Index der vorherigen Auswahl der Combobox. **Beispiel:** `<INDEX_CHANGED>` `<print text="index changed filed:%s, %d"> $focus_name, $focus_item_data </print>` … … `</INDEX_CHANGED>`
MENU	Das Tag definiert ein Menü, das die Softkeybeschreibung und den zu öffnenden Dialog beinhaltet. **Attribut:** • **name** Menüname **Syntax:** `<MENU name = "<menu name>">` … `<open_form …>` … `<SOFTKEY …>` `</SOFTKEY>` `</MENU>`

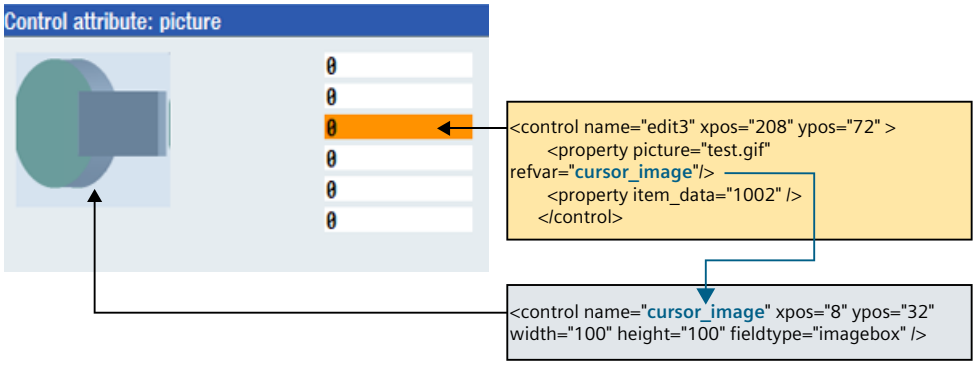
Tag-Bezeichner	Bedeutung
NAVIGATION	Das Tag legt das aufzurufende Menü fest. Es kann innerhalb eines Softkey-Blocks, eines Menü-Blocks und in einer Form eingesetzt werden. Wird dem Tag als Wert ein Variablenname zugewiesen, aktiviert der Parser das in der Variablen hinterlegte Menü. In einem Menü-Block erfolgt die Navigation an der Anweisungsposition. Nachfolgende Anweisungen werden nicht mehr ausgeführt. **Hinweis:** Wenn das Navigationsziel durch den Inhalt einer Variablen festgelegt werden soll, ist diese als globale Variable zu deklarieren. **Syntax:** `<NAVIGATION>menu name</NAVIGATION>` **Beispiel:** `<menu name = "main">` `  <softkey POSITION="1">` `  <caption>sec. form</caption>` `  <navigation>sec_menu</navigation>` `  </softkey>` `</menu>` `<menu name = "sec_menu">` `  <open_form name = "sec_form" />` `  <softkey_back>` `  <navigation>main</navigation>` `  </softkey_back>` `</menu>`

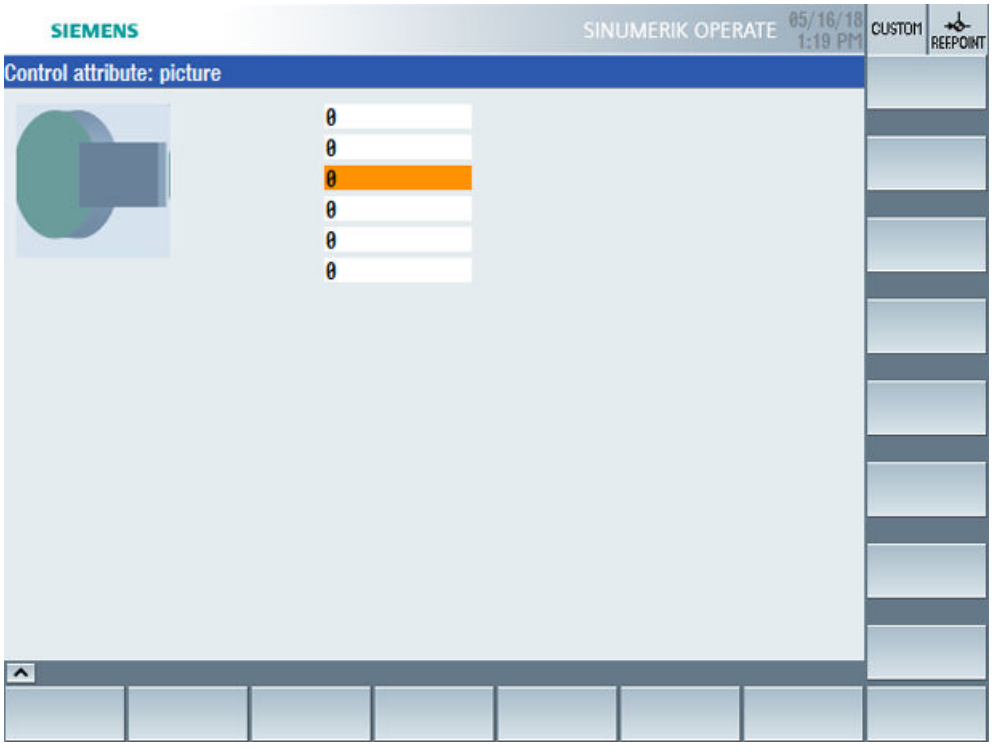
Tag-Bezeichner	Bedeutung
OPEN_FORM	Das Tag öffnet die unter dem Namen angegebene Dialogmaske. Wenn die angegebene Dialogmaske bereits aktiv ist, wird dieses Tag nicht ausgeführt. Ein wiederholtes Öffnen der Dialogmaske kann durch die Angabe des Attributs **reopen** erzwungen werden. Dies kann notwendig sein, wenn sich Systemzustände ändern, die einen Umbau der Maskeninhalte voraussetzen. **Attribut:** • **name** Name der Dialogmaske • **reopen** Wenn der Wert des Attributes auf **true** gesetzt ist, wird bei einem wiederholten Aufruf des Tags **open_form** überprüft, ob es sich um die aktive Dialogmaske handelt. Ist dies der Fall, schließt der Parser die aktive Form und öffnet sie erneut. **Syntax:** `<OPEN_FORM name = "<form name>" />` **Beispiel:** `<menu name = "main">` `  <open_form name = "main_form" />` `  <softkey POSITION="1">` `  <caption>main form</caption>` `  <navigation>main</navigation>` `  </softkey>` `</menu>` `<form name="main_form">` `  <init>` `  </init>` `  <paint>` `  </paint>` `</form>`

Tag-Bezeichner	Bedeutung
PROPERTY	Mit diesem Tag lassen sich zusätzliche Eigenschaften für ein Bedienelement festlegen. Das Tag wird im Control-Tag eingebettet. **Attribute:** • **max** = "<maximaler Wert>" • **min** = "<minimaler Wert>" • **default** = "<Vorbelegung>" • **factor** = "Umrechnungsfaktor" • **color_bk** = "<Farbcodierung Hintergrund>" • **color_fg** = "<Farbcodierung Schrift>" • **password** = "<true>" - eingegebene Zeichen werden mit "*" angezeigt • **multiline** = "<true>" - erlaubt die mehrzeilige Eingabe in ein Edit-Control • **disable** = "<true/false>" - sperrt/ermöglicht die Eingabe in ein Edit-Control • **transparent** = "Transparentfarbe einer Bitmap" **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" • **tooltip** = Hinweistext wird angezeigt, wenn der Cursor auf das Control gesetzt wird. Die Syntax lautet: <property tooltip="Hinweis text" /> • **abscissa** = "Name der ersten Koordinatenachse" (nur zulässig für Grafikbox) • **ordinate** = "Name der zweiten Koordinatenachse" (nur zulässig für Grafikbox) • **fix_time_area** = "on" - Die Attribute **min** und **max** legen den Zeitraum fest, in dem die Signale dargestellt werden sollen. Die dargestellten Signale werden von rechts nach links verschoben. **Beispiel:** `<CONTROL name = "progress1" xpos = "10" ypos = "10" width = "100" fieldtype = "progressbar" hotlink = "true" refvar = "nck/Channel/GeometricAxis/actProgPos[1]">` `<PROPERTY min = "0" />` `<PROPERTY max = "1000" />` `</CONTROL>` `<CONTROL name = "edit1" xpos = "10" ypos = "10">` `<PROPERTY min = "20" />` `<PROPERTY max = "40" />` `<PROPERTY default = "25" />` `</CONTROL>`

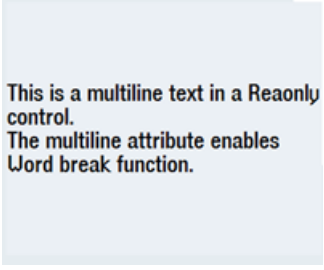
Tag-Bezeichner	Bedeutung
PROPERTY Fortsetzung	**Attribut:** **alignment** - Das Property ermöglicht eine links- oder rechtsbündige Textausrichtung. Standardmäßig wird ein Text linksbündig ausgerichtet. **Werte:** • **left** • **right** **Syntax:** `alignment="<right/left>"` **Beispiel:** `<control name="edit2" xpos="208" ypos="52" >` `<property alignment="right"/>` `</control>` 

Tag-Bezeichner	Bedeutung
PROPERTY Fortsetzung	**Attribut:** **parent** - Das Property legt die Zugehörigkeit des Controls fest. Standardmäßig sind Controls der Form zugeordnet. Möchte man Controls einer Scrollview zuweisen, ist diese als Elternfenster anzugeben. **Wert:** Name des Elternfensters **Beispiel:** `...` `<control name="area" xpos="8" ypos="72" width="554" height="320" fieldtype="scrollarea" >` `<control name="test" xpos="20" ypos="40" width="20" fieldtype="readonly" />` `</control >` `<control name="test1" xpos="20" ypos="80" width="20" fieldtype="readonly" parent="area"/>` **test** **test1** `<init>` `<control name="area" xpos="8" ypos="72" width="554" height="120" fieldtype="scrollarea">` `<control name="test" xpos="20" ypos="40" width="80" fieldtype="readonly" type="string"/>` `</control >` `<control name="test1" xpos="20" ypos="80" width="80" fieldtype="readonly" type="string" parent="area"/>` `<op>test = _T"test"</op>` `<op>test1 = _T"test1"</op>` `<update_controls type="true" />` `...`

Tag-Bezeichner	Bedeutung
PROPERTY Fortsetzung	**Attribut:** **picture** - das Attribut legt eine anzuzeigende Grafik fest Die angegebene Grafik wird angezeigt bzw. der Name der Grafik in einer Variablen abgelegt, wenn der Eingabefocus auf dieses Feld gesetzt wird. Dieses Attribut ist zusammen mit dem Attribut **refvar** zu verwenden, um die Datensenke festlegen zu können. Zum Anzeigen einer Grafik oder einer Animation ist der Name eines Controls vom Typ **imagebox** anzugeben. Dieses Control übernimmt die Verwaltung der Grafik. Wird der Name einer lokalen Variablen angegeben, muss diese Variable vom Datentyp **String** sein. Die Grafik wird solange angezeigt, bis der Referenzvariablen ein neuer Name zugewiesen wird. **Syntax:** `<property picture="<Name der Grafik>" refvar="<Datensenke>"/>`
PROPERTY Fortsetzung	**Beispiel der Feldverknüpfung:** 

Tag-Bezeichner	Bedeutung
PROPERTY Fortsetzung	**Beispiel:** `<let name="cursor_icon" type="string" />` ` <form name="main_form1">` `  <init>` `  <caption>Control attribute: picture</caption>` `  <control name="cursor_image" xpos="8" ypos="32" width="100" height="100" fieldtype="imagebox" />` `  <control name="edit1" xpos="208" ypos="32" >` `  <property picture="red_led_off.bmp" refvar="cursor_image"/>` `  </control>` `  <control name="edit2" xpos="208" ypos="52" >` `  <property picture="red_led_on.bmp" refvar="cursor_image"/>` `  </control>` `  <control name="edit3" xpos="208" ypos="72" >` `  <property picture="test.gif" refvar="cursor_image"/>` `  </control>` `  <control name="edit4" xpos="208" ypos="92" >` `  <property picture="red_led_off.bmp" refvar="cursor_image"/>` `  </control>` `  <control name="edit5" xpos="208" ypos="112" >` `  <property picture="red_led_off.bmp" refvar="cursor_icon"/>` `  </control>` `  <control name="edit6" xpos="208" ypos="132" >` `  <property picture="red_led_on.bmp" refvar="cursor_icon"/>` `  </control>` `  </init>` `  <timer><print text="%s">cursor_icon</print></timer>` ` </form>` 

Tag-Bezeichner	Bedeutung
PROPERTY Fortsetzung	**Attribut:** **cursortext** - das Attribut legt den anzuzeigenden Cursortext fest Der Text wird in der Titelzeile ausgegeben, wenn der Eingabefokus auf dieses Feld gestellt wird. Zu diesem Attribut können zwei weitere Attribute angegeben werden, die die Ausrichtung des Textes und den Cursortextbereich festlegen. Standardmäßig erfolgt eine linksbündige Ausrichtung des Textes. Der Cursortextbereich nimmt standardmäßig 50 % der Breite der Titelzeile ein. `alignment="<right/left>"` - Textausrichtung rechtsbündig/linksbündig `length="<Breite>"` - prozentualer Anteil den der Cursortext in der Titelzeile beanspruchen soll **Syntax:** `<property cursortext="<text>" />` oder `<property cursortext="<text>" alignment="<right/left>" />` oder `<property cursortext="text2" alignment="r="<text>" alignment="<right/left>" length="<Verhältniswert>"/>` **Beispiel:** `<form name="main_form2">` ` <init>` `  <caption>Control attribute: cursortext</caption>` `  <control name="edit1" xpos="208" ypos="32" >` `  <property cursortext="cursor text field edit1" />` `  </control>` `  <control name="edit2" xpos="208" ypos="52" >` `  <property cursortext="cursor text field edit2" alignment="right"/>` `  </control>` `  <control name="edit3" xpos="208" ypos="72" >` `  <property cursortext="cursor text field edit3" alignment="right" length="40"/>` `  </control>` `  <control name="edit4" xpos="208" ypos="92" >` `  <property cursortext="cursor text field edit4" />` `  </control>` ` </init>` `</form>` 

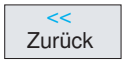
Tag-Bezeichner	Bedeutung
PROPERTY Fortsetzung	**Attribut:** **multiline** - die ermöglicht die mehrzeilige Ausgabe eines Textes in einem Edit- oder Readonly Control **Syntax:** `<property multiline="true" />` **Beispiel:** `...` `...` `<op>` `textlist[2] = _T"This is a multiline text in a Reaonly control.\` `\nThe multiline attribute enables Word break function.";` `</op>` `<control name="lstring" xpos="8" ypos="120" width="200" height="160"` `fieldtype="readonly" refvar="textlist[2]" >` `<property multiline="true" />` `</control>` 
PROPERTY Fortsetzung	**Attribute:** • **help_context** Das Attribut ermöglicht einem Control ein Hilfethema zuzuweisen.Es ist der im Hilfebuch im Tag **entry** verwendete Dateiname anzugeben. • **anchor** Das Attribut ermöglicht die Angabe des Schlüsselwortes. Dieses Attribut ist nur im Zusammenhang mit dem Attribut **help_context** gültig. **Syntax:** `<property help_context="<file name>" anchor="<key word>"/>` **Beispiel:** `<control name = "cpwd" xpos = "322" ypos = "64" >` `<property password="true" />` `<property help_context="chapter_1.html" anchor="Keyword_2" />` `</control>`

Tag-Bezeichner	Bedeutung
SOFTKEY	Das Tag definiert die Eigenschaften und Reaktionen eines Softkeys. **Attribute:** • **position** Nummer des Softkeys. 1-8 horizontale Softkeys, 9-16 vertikale Softkeys Nachfolgende Attribute wirken ab: • **type** Definiert die Eigenschaft des Softkeys. user_controled - Die Darstellung des Softkeys wird durch das Skript bestimmt toggle_softkey - Der Softkey wird wechselnd gedrückt oder nicht gedrückt dargestellt • **refvar** Nur in Verbindung mit dem Typ toggle_softkey anzuwenden. Referenzvariable, in der die aktuelle Softkeyeigenschaft kopiert wird. Es ist eine Variable vom Typ "String" anzugeben, die die Eigenschaften gedrückt, nicht gedrückt oder gesperrt enthält (siehe Tag state). • **picture** Mit Hilfe des Attributes kann eine Bitmap linksbündig auf dem Softkey ausgegeben werden. Es ist der vollständige Pfadname anzugeben. Die Anzahl darstellbarer Textzeichen reduzieren sich um die Breite der Bitmaps.

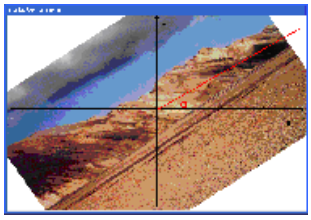
Tag-Bezeichner	Bedeutung
SOFTKEY Fortsetzung	Folgende weitere Aktionen können innerhalb des Softkey-Blocks festgelegt werden: • **picturealignment** Die Ausrichtung des Bildes wird durch dieses Attribut vorgegeben. Standardmäßig wird das Bild an der linken Seite des Softkeys ausgerichtet. Folgende Werte können zur Ausrichtung angegeben werden: – **top** obere Kante – **bottom** unter Kante – **left** linke Kante – **right** rechte Kante – **center** zentriert • **caption** Text des Softkeys • **state** Nur in Verbindung mit dem Typ user_controlled anzuwenden. Das Tag weist dem System die gewünschte Darstellung des Softkeys zu. **Syntax:** <state type="<state>" /> Es können folgende Strings angegeben werden: – **notpressed** Der Softkey wird nicht gedrückt dargestellt. – **pressed** Der Softkey wird gedrückt dargestellt. – **disabled** Der Softkey ist gesperrt und wird grau dargestellt. • **navigation** • **update_controls** • **function**

Tag-Bezeichner	Bedeutung
SOFTKEY Fortsetzung	**Syntax:** Standard-Softkey: `<state type="<softkey state>" />` `<softkey position = "<1>">` … … `</softkey>` oder Skriptgesteuerter Softkey: `<softkey position = "<1>" type="<user_defined>" >` `<state type="<softkey state>" />` … … `</softkey>` oder Toggle-Softkey: `<softkey position = "<1>" type="<toggle_softkey>" refvar="<variable name>" >` … … `</softkey>` **Beispiel:** `<let name="define_sk_type" type="string">PRESSED</let>` `<let name="sk_type">1</let>` `<softkey POSITION="1" type="user_controled" >` `<caption>Toggle%nSK</caption>` `<if>` `<condition>sk_type == 0 </condition>` `<then>` `<op> sk_type = 1 </op>` `<op> define_sk_type = _T"PRESSED" </op>` `</then>` `<else>` `<op> define_sk_type = _T"NOTPRESSED" </op>` `<op> sk_type = 0 </op>` `</else>` `</if>` `<state type="$$$define_sk_type" />` … … `</softkey>`

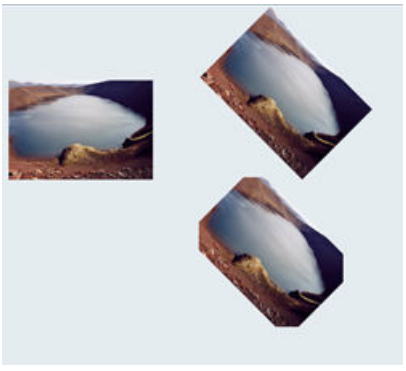
Tag-Bezeichner	Bedeutung
SOFTKEY Fortsetzung	**Beispiel:** oder <let name="curr_softkey_state" type="string">PRESSED</let> </softkey> <softkey POSITION="3" type="toggle_softkey" refvar="curr_softkey_state"> <caption>Toggle%nSK</caption> ... </softkey>  
SOFTKEY_OK	Das Tag definiert die Reaktion des Softkeys "OK".  Folgende weitere Aktionen können innerhalb des Softkey-Blocks festgelegt werden: • **navigation** • **update_controls** • **function** **Syntax:** <SOFTKEY_OK> ... ... </SOFTKEY_OK>
SOFTKEY_CANCEL	Das Tag definiert die Reaktion des Softkeys "Abbruch".  Folgende weitere Aktionen können innerhalb des Softkey-Blocks festgelegt werden: • **navigation** • **update_controls** • **function** **Syntax:** <SOFTKEY_CANCEL> ... ... </SOFTKEY_CANCEL>

Tag-Bezeichner	Bedeutung
SOFTKEY_BACK	Das Tag definiert die Reaktion des Softkeys "Zurück".  Folgende weitere Aktionen können innerhalb des Softkey-Blocks festgelegt werden: • **navigation** • **update_controls** • **function** **Syntax:** `<SOFTKEY_BACK>` ... ... `</SOFTKEY_BACK>`
SOFTKEY_ACCEPT	Das Tag definiert die Reaktion des Softkeys "Übernahme".  Folgende weitere Aktionen können innerhalb des Softkey-Blocks festgelegt werden: • **navigation** • **update_controls** • **function** **Syntax:** `<SOFTKEY_ACCEPT>` ... ... `</SOFTKEY_ACCEPT>`

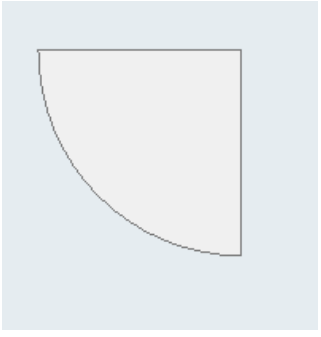
Tag-Bezeichner	Bedeutung
TEXT	Das Tag dient zum Anzeigen eines Textes an der angegebenen Position. Wird eine Alarmnummer verwendet, zeigt die Dialogbox den für die Nummer hinterlegten Text an. **Syntax:** `<TEXT xpos = "<X-Position>" ypos = "<Y-Position>"> Text </TEXT>` **Attribute:** • **xpos** X-Position der linken oberen Ecke • **ypos** Y-Position der linken oberen Ecke • **color** Textfarbe **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" **Wert:** Anzuzeigender Text
IMG	Das Tag dient zum Anzeigen eines Bildes an der angegebenen Position. Es werden die Bildformate BMP und PNG unterstützt. **Syntax:** `<IMG xpos = "<X-Position>" ypos = "<Y-Position>" name = "<name>" />` `<IMG … xrot="Winkel X-Achse" yrot=" Winkel Y-Achse " zrot=" Winkel Z-Achse " />` **Attribute:** • **xpos** X-Position der linken oberen Ecke • **ypos** Y-Position der linken oberen Ecke • **name** vollständiger Pfadname. Die Pfadangabe ist in Kleinschreibung vorzunehmen. • **transparent** Transparentfarbe der Bitmap **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)"

Tag-Bezeichner	Bedeutung
IMG Fortsetzung	**Beispiel**: Das Bild wird um 34 Grad um die Achse Z gedreht: `<img xpos = "5" ypos = "23" name = "f:/appl/pic3.bmp" height="355" width="550" zrot="34" color_bk="#ffffff" ></img>` **Hinweis:** Die Laufwerksbezeichnung variiert je nach System.  **Optional:** Soll die Darstellung des Bildes von der Originalgröße abweichen, kann die Dimension mit den Attributen **width** und **height** festgelegt werden. • **width** Breite in Pixel • **height** Höhe in Pixel **Beispiele:** `<IMG xpos = "20" ypos = "40" name = "f:/appl/test.bmp" />` `<IMG xpos = "5" ypos = "23" name = "f:/appl/test.bmp" height = "355" width = "550"/>`

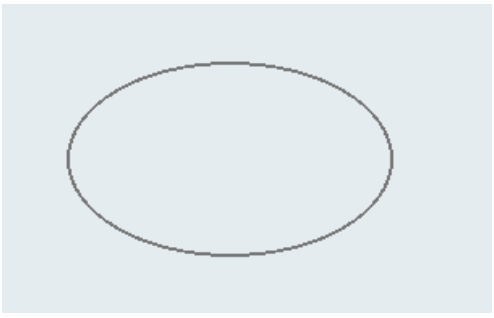
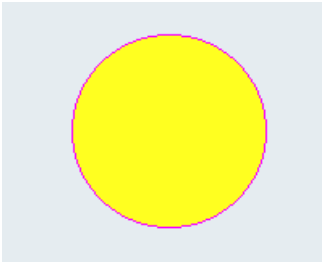
Tag-Bezeichner	Bedeutung
IMG Fortsetzung	**Attribut:** **AspectRatioMode** Das Attribut ermöglicht das Steuern des Seitenverhältnisses eines Bildes, wenn ein nicht proportionales Zoomen durchgeführt wird. **Werte:** • **Ignore** Das Seitenverhältnis der Bitmap wird an die vorgegebene Höhe und Breite angepasst. • **Keep** (Standard) Das Seitenverhältnis wird beibehalten und sichergestellt, dass das Bild auf ein Rechteck skaliert wird, das innerhalb der vorgegebenen Höhe und Breite die größte Ausdehnung annimmt. • **KeepByExpanding** Das Seitenverhältnis wird beibehalten und sichergestellt, dass das Bild auf ein Rechteck skaliert wird, das außerhalb der vorgegebenen Höhe oder Breite die kleinste Ausdehnung annimmt. **Beispiel:** `<img xpos = "8" ypos = "30" name = "test.bmp" height="155" width="150" AspectRatioMode="KeepbyExpanding">` `  <property transparent="#00ff00" />` `</img>` `<img xpos = "8" ypos = "200" name = "test.bmp" height="155" width="150" AspectRatioMode="Ignore">` `  <property transparent="#00ff00" />` `</img>` `<img xpos = "200" ypos = "200" name = "test.bmp" height="155" width="150" AspectRatioMode="Keep">` `  <property transparent="#00ff00" />` `</img>`  • Oben - Eigenschaft **KeepbyExpanding** gesetzt • Links unten - Eigenschaft **Ignore** gesetzt • Rechts unten - Eigenschaft **Keep** gesetzt

Tag-Bezeichner	Bedeutung
IMG Fortsetzung	**Attribute:** **ExpandingForRotation** Das Seitenverhältnis wird beibehalten. Das Bild wird auf ein Rechteck skaliert, das der Bounding-Box des rotierten und skalierten Bildes entspricht. **Beispiel:** `<img xpos = "8" ypos = "100" name = "pic1.bmp" height="155" width="150" >` `  <property transparent="#ffffff" />` `</img>` `<op>` `  zrot = 45.0;` `</op>` `<img xpos = "205" ypos = "23" name = "pic1.bmp" height="155" width="150" zrot="$zrot" AspectRatioMode="ExpandingForRotation">` `  <property transparent="#ffffff" />` `</img>` `<img xpos = "205" ypos = "200" name = "pic1.bmp" height="155" width="150" zrot="$zrot" >` `  <property transparent="#ffffff" />` `</img>`  • Links - Bild auf eine Größe von 150x155 Pixel skaliert • Rechts oben - Bild auf eine Größe von 150x155 Pixel skaliert und um 45 Grad mit der Eigenschaft **ExpandingForRotation** gedreht • Rechts unten - Bild auf eine Größe von 150x155 Pixel skaliert und um 45 Grad gedreht

Tag-Bezeichner	Bedeutung
ARC	Das Tag dient zum Zeichnen eines Kreissektors in einer Form. **Attribute:** Position innerhalb der Form in Pixel: • **xpos1** - Startpunkt des Kreissegments X-Koordinate • **ypos1** - Startpunkt des Kreissegments Y-Koordinate • **xpos2** - Endpunkt des Kreissegments X-Koordinate • **ypos2** - Endpunkt des Kreissegments Y-Koordinate Position innerhalb der Form in Pixel: • **ccx** - Mittelpunkt des Kreissegments X-Koordinate • **ccy** - Mittelpunkt des Kreissegments Y-Koordinate • **radius** - Kreisradius, Wert in Pixel **Hinweis:** **Mittelpunkt oder Radius** Bei der Angabe beider Parameter wird der Kreismittelpunkt verwendet.
ARC Fortsetzung	• **penwidth** Das Attribut legt die Strichstärke der Linie in Pixel fest. • **color** Linienfarbe • **color_bk** Füllfarbe des Kreissegments **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" • **style** Das Attribut legt den Linientyp fest: – **"SolidLine"** - durchgezogene Linie (Default) – **"DashLine"** - gestrichelte Linie – **"DotLine"** - gepunktete Linie – **"DashDotLine"** - Punkt-Strich-Linie – **"DashDotDotLine"** - Strich-Punkt-Punkt Linie

Tag-Bezeichner	Bedeutung
ARC Fortsetzung	• **type** – **cw** - clockwise – **ccw** - counterclockwise • **arrow** An den Segmentenden werden Pfeile dargestellt: – **"P1"** - Pfeil am Startpunkt der Linie – **"P2"** - Pfeil am Startpunkt der Linie – **"P1P2"** - Pfeile am Startpunkt und Endpunkt der Linie – **"P1_FILLED"** - Pfeil am Startpunkt der Linie gefüllt – **"P2_FILLED"** - Pfeil am Startpunkt der Linie gefüllt – **"P1P2_FILLED"** - Pfeile am Startpunkt und Endpunkt der Linie gefüllt • **arrow_size** Wird das Attribut Arrow verwendet, kann die Länge des Pfeils in Pixel angegeben werden.
ARC Fortsetzung	**Beispiele:**  `<arc xpos1="20" ypos1="200" xpos2="180" ypos2="200" ccx="100" ccy="200" type="ccw" PENWIDTH="1" color="#777777" arrow="p1p2_filled"/>` oder `<arc xpos1="20" ypos1="200" xpos2="180" ypos2="200" radius="80" type="ccw" PENWIDTH="1" color="#777777" arrow="p1p2_filled"/>`  `<arc xpos1="20" ypos1="200" xpos2="100" ypos2="280" radius="80" type="ccw" PENWIDTH="1" color="#777777" color_bk="#f0f0f0"/>`

Tag-Bezeichner	Bedeutung
BOX	Das Tag zeichnet ein gefülltes Rechteck an der angegebenen Position in der angegebenen Farbe. **Syntax:** `<BOX xpos = "<X-Position>" ypos = "<Y-Position>" width = "<X-Ausdehnung>" height = "<Y-Ausdehnung>" color = "<Farbcode>" />` **Attribute:** • **xpos** X-Position der linken oberen Ecke • **ypos** Y-Position der linken oberen Ecke • **width** Ausdehnung in X-Richtung (in Pixel) • **height** Ausdehnung in Y-Richtung (in Pixel) • **color** Farbcodierung **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)"
ELLIPSE	Das Tag dient zum Zeichnen einer Ellipse in einer Form. **Attribute:** Position innerhalb der Form in Pixel: • **xpos** - Startpunkt der Linie X-Koordinate • **ypos** - Startpunkt der Linie Y-Koordinate • **width** - Breite des umgebenden Rechtecks (bounding box) • **height** - Höhe des umgebenden Rechtecks (bounding box) • **penwidth** Das Attribut legt die Strichstärke der Linie in Pixel fest. • **color** Linienfarbe • **color_bk** Füllfarbe der Ellipse **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" • **style** Das Attribut legt den Linientyp fest: – **"SolidLine"** - durchgezogene Linie (Default) – **"DashLine"** - gestrichelte Linie – **"DotLine"** - gepunktete Linie – **"DashDotLine"** - Punkt-Strich-Linie – **"DashDotDotLine"** - Strich-Punkt-Punkt Linie

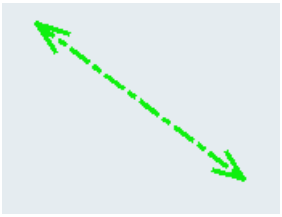
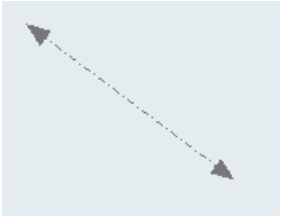
Tag-Bezeichner	Bedeutung
ELLIPSE Fortsetzung	**Beispiele:**  `<ellipse xpos="302" ypos="160" width="100" height="60" PENWIDTH="2" color="#777777" />`  `<ellipse xpos="402" ypos="360" width="60" height="60" PENWIDTH="1" color="#f800ff"  color_bk="#ffff20"/>`

Tag-Bezeichner	Bedeutung
FUNCTION	**Funktionsaufruf** Das Tag führt den unter dem Attribut "**name**" angegebenen Funktionskörper aus. **Attribute:** • **name** = "Name des Funktionskörpers" • **return** = "Variablenname zum Speichern des Funktionsergebnisses" **Werte:** Liste der Variablen, die an den Funktionskörper übergeben werden sollen. Die Variablen sind durch ein Komma voneinander zu trennen. Es können max. 10 Parameter übergeben werden. Weiterhin ist es möglich, Konstanten oder Textausdrücke als Aufrufparameter anzugeben. Zum Kennzeichnen eines Textausdrucks ist dem Text die Kennung _T voranzustellen. **Syntax:** `<FUNCTION name = "<function name>" />` Aufrufende Funktion erwartet einen Rückgebewert `<FUNCTION name = "<function name>" return = "<Variablenname>" />` Parameterübergabe `<FUNCTION name = "<function name>"> var1, var2, var3 </FUNCTION>` `<FUNCTION name = "<function name>"> _T"Text", 1.0, 1 </FUNCTION>` **Beispiele:** Siehe "FUNCTION_BODY"

Tag-Bezeichner	Bedeutung
FUNCTION_BODY	**Funktionskörper** Das Tag beinhaltet den Funktionskörper einer Unterfunktion. Der Funktionskörper ist innerhalb des DialogGui-Tags zu programmieren. **Attribute:** • **name** = "Name des Funktionskörpers" • **parameter** = "Parameterliste" (optional) Das Attribut listet die benötigten Übergabeparameter auf. Die Parameter sind durch ein Komma voneinander zu trennen. Beim Aufruf des Funktionskörpers werden die Werte der im Funktionsaufruf angegebenen Parameter in die aufgeführten Übergabeparameter kopiert. • **return** = "true" (optional) Ist das Attribut auf true gesetzt, wird die lokale Variable $return angelegt. In diese ist der Rückgabewert der Funktion zu kopieren, der mit dem Verlassen der Funktion an die aufrufende Funktion weitergeleitet wird. **Syntax:** **Funktionskörper ohne Parameter** `<FUNCTION_BODY name = "<function name>">` … … … `</ FUNCTION_BODY>` **Funktionskörper mit Parameter** `<FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>">` … `<LET name = "tmp></LET>` `<OP> tmp = p1 </OP>` … `</FUNCTION_BODY>` **Funktionskörper mit Rückgabewert** `<FUNCTION_BODY name = "<function_name>" parameter = "<p1, p2, p3>" return = "true">` … `<LET name = "tmp></LET>` `<OP> tmp = p1 </OP>` … `<OP> $return = tmp </OP>` `</FUNCTION_BODY>`

Tag-Bezeichner	Bedeutung
FUNCTION_BODY Fortsetzung	**Beispiel:** `<function_body name = "test" parameter = "c1,c2,c3" return = "true">` `<LET name = "tmp">0</LET>` `<OP> tmp = c1+c2+c3 </OP>` `<OP> $return = tmp </OP>` `</function_body>` … … … `<LET name = "my_var"> 4 </LET>` `<function name = "test" return = " my_var "> 2, 3,4</function>` `<print text = "result = %d"> my_var </print>` … …
LINE	Das Tag dient zum Zeichnen einer Linie in einer Form. **Attribute:** Position innerhalb der Form in Pixel: • **xpos1 -** Startpunkt der Linie X-Koordinate • **ypos1** - Startpunkt der Linie Y-Koordinate • **xpos2** - Endpunkt der Linie X-Koordinate • **ypos2** - Endpunkt der Linie Y-Koordinate • **penwidth** Das Attribut legt die Strichstärke der Linie in Pixel fest. • **color** Linienfarbe **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" • **style** Das Attribut legt den Linientyp fest: – **"SolidLine"** - durchgezogene Linie (Default) – **"DashLine"** - gestrichelte Linie – **"DotLine"** - gepunktete Linie – **"DashDotLine"** - Punkt-Strich-Linie – **"DashDotDotLine"** - Strich-Punkt-Punkt Linie

Tag-Bezeichner	Bedeutung
LINE Fortsetzung	• **arrow** An den Linienenden werden Pfeile dargestellt: – **"P1"** - Pfeil am Startpunkt der Linie – **"P2"** - Pfeil am Startpunkt der Linie – **"P1P2"** - Pfeile am Startpunkt und Endpunkt der Linie – **"P1_FILLED"** - Pfeil am Startpunkt der Linie gefüllt – **"P2_FILLED"** - Pfeil am Startpunkt der Linie gefüllt – **"P1P2_FILLED"** - Pfeile am Startpunkt und Endpunkt der Linie gefüllt • **arrow_size** Wird das Attribut Arrow verwendet, kann die Länge des Pfeils in Pixel angegeben werden. **Syntax:** `<line xpos1="<X-Koordinate Startpunkt>" ypos1="<Y-Koordinate Startpunkt>" xpos2="<X-Koordinate Endpunkt>" ypos2="<Y-Koordinate Startpunkt>"` `PENWIDTH="<Strichstärke>" color="<Linienfarbe>" style="<Linien-Style>" arrow="<Pfeiltyp>" arrow_size="<Pfeilgröße>"/>`

Tag-Bezeichner	Bedeutung
LINE Fortsetzung	**Beispiele:** `<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="3" color="#0ff00f" style="DashDotLine" arrow="p1p2" arrow_size="12"/>`  `<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="1" color="#777777" style="DashDotLine" arrow="p1p2_filled" arrow_size="9"/>`  `<line xpos1="20" ypos1="40" xpos2="100" ypos2="100" PENWIDTH="2" color="#777777" arrow="p2_filled" arrow_size="9"/>` 

Tag-Bezeichner	Bedeutung
REQUEST	Mit Hilfe des Tags wird eine Variable in den zyklischen Lesedienst (Hotlink) aufgenommen. Dadurch verkürzt sich die Zugriffszeit auf Variablen, die nicht an ein Control gebunden sind. Soll bei einer Wertänderung automatisch eine Funktion aufgerufen werden, ist als weiteres Attribut der Name der Funktion anzugeben. Dieses Tag wird nur innerhalb der INIT-Anweisung verarbeitet. **Attribute:** • **name** Adressbezeichner • **function** Funktionsname **Syntax:** `<REQUEST name = "<NC-Variable>" />` oder `<REQUEST name = "<NC-Variable>" function="<function name>"/>` **Beispiel:** `<request name ="plc/mb10" />` oder `<function_body name="my_function" >` `<print text="value changed" />` `</function_body>` … … … `<request name ="plc/mb10" function="my_function"/>`

Tag-Bezeichner	Bedeutung
RECALL	Dieses Tag kann in einem Menü verwendet werden, wenn eine Navigation über die Recall-Taste erfolgen soll. Ist das Tag programmiert, wird das Recall-Symbol eingeblendet und die Verarbeitung der Recall-Taste durch den Parser zugelassen. **Syntax:** `<recall>` ... ... `</recall>`
UPDATE_CONTROLS	Das Tag führt einen Abgleich zwischen den Bedienelementen und den Referenzvariablen durch. **Attribut:** • **type** Das Attribut legt die Richtung des Datenabgleichs fest. **= TRUE** – Die Daten werden aus den Referenzvariablen gelesen und in die Bedienelemente kopiert. **= FALSE** – Die Daten werden aus den Bedienelementen in die Referenzvariablen kopiert. **Syntax:** `<UPDATE_CONTROLS type = "<Richtung>"/>` **Beispiel:** `<SOFTKEY_OK>` `< UPDATE_CONTROLS type="false"/>` `</SOFTKEY_OK>`

3.8 Anwendermenüs erstellen

3.8.1 Bearbeitungszyklenmasken erstellen

Die Funktion Zyklenunterstützung ermöglicht das automatische Erstellen und Rückübersetzen eines Zyklenaufrufes durch den Form-Dialog.

Zum Handhaben dieser Funktionalität stehen die folgenden Tags zur Verfügung:

- **NC_INSTRUCTION**
- **CREATE_CYCLE**

Zum Kennzeichnen einer Zyklenmaske ist im Tag **FORM** das Attribut **type** mit dem Wert **cycle** anzugeben. Diese Kennzeichnung ermöglicht das Verarbeiten der Anweisung **NC_INSTRUCTION**.

Beispiel

```
<FORM name = "cycle100_form" type= "CYCLE">
...
...
</FORM>
```

Das Tag **NC_INSTRUCTION** beinhaltet den zu erzeugenden Zyklenaufruf. Alle Zyklenparameter sind durch Platzhalter zu reservieren.

Beispiel

```
<FORM name = "cycle100_form" type= "CYCLE">
<NC_INSTRUCTION refvar= "cyc_string" >Cycle100 ($p1, $p2, $p3)</
NC_INSTRUCTION>
...
...
...
</FORM
```

Das Tag **CREATE_CYCLE** bereitet die in den Platzhalter-Variablen gespeicherten Werte auf und generiert die NC-Anweisung.

Diese wird anschließend in die angegebene Variable kopiert.

Tag-Bezeichner	Bedeutung
NC_INSTRUCTION	Mit diesem Tag wird die zu erstellende NC-Anweisung definiert. Alle aufgeführten Zyklusparameter werden automatisch als String-Variablen der **FORM** angelegt und stehen der **FORM** zur Verfügung. Voraussetzung: Das FORM-Attribut **type** ist auf den Wert **CYCLE** gesetzt. **Attribut:** • **refvar** Ist dem Tag eine Referenzvariable zugewiesen, werden alle Parameter mit den Werten aus dem in der Referenzvariable hinterlegten NC-Satz vorbelegt. **Syntax:** `<NC_INSTRUCTION> NC – Anweisung mit Platzhaltern </ NC_INSTRUCTION>` **Beispiel:** `<let name="cyc_string" type="string"> Cycle100(0, 1000, 5)</ let>` … … … `<FORM name = "cycle100_form" type= "CYCLE">` `<NC_INSTRUCTION refvar= "cyc_string" >Cycle100($p1, $p2, $p3)</ NC_INSTRUCTION>` … … … `</FORM>`

Tag-Bezeichner	Bedeutung
CREATE_CYCLE	Das Tag erzeugt einen NC–Satz, dessen Syntax durch den Wert des **NC_INSTRUCTION** – Tags festgelegt ist. Vor dem Erzeugen der NC-Anweisung ruft der Parser das Tag CYCLE_CREATE_EVENT der FORM auf. Dieses Tag kann zur Berechnung der Zyklenparameter benutzt werden. **Syntax:** `<CREATE_CYCLE/>` **Option:** Wenn eine Referenzvariable angegeben wird, kopiert die Anweisung den erzeugten Aufruf in diese Variable. `<create_cycle refvar="name" />` **Attribut:** • **refvar** Ist dem Tag eine Referenzvariable zugewiesen, wird die NC- Anweisung in diese Variable kopiert. **Beispiel:** `<LET name="cyc_string" type="string"> Cycle100(0, 1000, 5)</LET>` `<SOFTKEY_OK>` `  <caption>OK</caption>` `  <CREATE_CYCLE />` `  <close_form />` `  <navigation>main_menu</navigation>` `</SOFTKEY_OK>` `oder` `<SOFTKEY_OK>` `  <caption>OK</caption>` `  <CREATE_CYCLE refvar= "cyc_string" />` `  <close_form />` `  <navigation>main_menu</navigation>` `</SOFTKEY_OK>`

3.8.2 Ersetzungszeichen

Das System bietet die Möglichkeit, Control-Eigenschaften (Attributwerte) zur Laufzeit festzulegen. Um diese Funktion nutzen zu können, ist die gewünschte Eigenschaft in einer lokalen Variablen bereitzustellen und der Variablenname mit einem vorangestellten **$-Zeichen** als Attributwert dem Tag zu übergeben.

Erwartet das Tag einen String als Attributwert oder Wert, sind die Zeichen $$$ dem Variablennamen voranzustellen.

Beispiel:

```
<let name="my_ypos">100</let>
<let name="field_name" type="string"></let>

<control name = "edit1" xpos = "322" ypos = "$my_ypos" refvar="nck/
Channel/Parameter/R[1]" />

<op>my_ypos = my_ypos +20 </op>

<control name = "edit2" xpos = "322" ypos = "$my_ypos" refvar="nck/
Channel/Parameter/R[2]" />

<print name ="field_name" text="edit%d">3</print>
<op>my_ypos = my_ypos +20 </op>

<control name = "$field_name" xpos = "322" ypos = "$my_ypos"
refvar="nck/Channel/Parameter/R3]" />

<caption>$$$field_name</caption>
```

3.9 Datei struct_def.xml erstellen

Vorgehen

In einigen Funktionen wird die Datei **struct_def.xml** zur Typdefinition verwendet. Wenn Ihnen diese XML-Datei fehlt, erstellen Sie mit folgender Auszeichnung eine Datei-Kopie und integrieren Sie diese im Funktionsprojekt.

Hinweis

Erstellen Sie die XML-Datei im UTF8-Format.

Skript

struct_def.xml

```
<!--
Copyright 2015 by Siemens AG and Wolfram Kuhnert
(wolfram.kuhnert@siemens.com)
Permission to use, copy, modify, distribute, and sell this softwareand its
documentation for any purpose is hereby granted without fee,provided that
the above copyright notice appear in all copies and that both that copyright
notice and this permission notice appearin supporting documentation, and
that the names of Siemens AG and Wolfram Kuhnertnot be used in advertising
or publicity pertaining todistribution of the software without specific,
written priorpermission.
Siemens AG and Wolfram Kuhnert make no representations about the
suitability of this software for any purpose. It isprovided "as is" without
express or implied warranty.
Required software version: Operate 4.7 Sp1 HF1
-->

<!--
    typedef struct tagRECT {
     LONG left;
     LONG top;
     LONG right;
     LONG bottom;
    } RECT;
-->

<typedef name="StructRect" type="struct" >
  <element name="left" type="int">0</element>
  <element name="top" type="int">0</element>
  <element name="right" type="int">0</element>
  <element name="bottom" type="int">0</element>
</typedef>

<typedef name="StructPoint" type="struct" >
  <element name="x" type="int">0</element>
  <element name="y" type="int">0</element>
</typedef>

<typedef name="StructSize" type="struct" >
  <element name="width" type="int">0</element>
  <element name="height" type="int">0</element>
</typedef>

<typedef name="StructMDLimits" type="struct" >
  <element name="lowerLimit" type="double">0</element>
  <element name="upperLimit" type="double">0</element>
  <element name="arrayDim1" >0</element>
  <element name="arrayDim2" >0</element>
</typedef>
```

3.10 Komponenten adressieren

Zum Adressieren von NC-Variablen, PLC-Bausteinen oder Antriebsdaten sind Adressbezeichner auf das gewünschte Datum zu bilden. Eine Adresse besteht aus den Teilpfaden **Komponentenname** und **Variablenadresse**. Als Trennzeichen ist ein Schrägstrich zu verwenden.

3.10.1 PLC adressieren

Das Adressieren der PLC beginnt mit dem Pfadanteil **plc**.

Tabelle 3-3 Folgende Adressen sind zulässig:

DBx.DB(f)	Datenbaustein
I(f)x	Eingang
Q(f)x	Ausgang
M(f)x	Merker
V(f)x	Variable

DBx.DBXx.b	Datenbaustein
Ix.b	Eingang
Qx.b	Ausgang
Mx.b	Merker
Vx.b	Variable

Tabelle 3-4 Datenformat **f**:

B	Byte
W	Wort
D	Doppelwort

Bei einer Bitadressierung entfällt die Datenformatkennzeichnung.

Adresse **x**:

Gültiger S7 200 Adressbezeichner

Bit-Adressierung:

b – Bitnummer

Beispiele:

```
<data name = "plc/mb170">1</data>
<data name = "i0.1"> 1 </data>
<op> "m19.2" = 1 </op>
refvar="plc/db9062.dbb0:string"
```

3.10.2 NC-Variablen adressieren

Das Adressieren der NC-Variablen beginnt mit dem Pfadanteil **nck**.

Diesem Anteil folgt die Adresse des Datums, deren Aufbau dem Listenhandbuch NC-Variable und Nahtstellensignale zu entnehmen ist.

Beispiel:

```
<LET name = "tempStatus"></LET>
<OP> tempStatus ="nck/channel/state/chanstatus" </OP>
```

3.10.3 Kanalspezifisch adressieren

Wenn im Adress-Token keine Kanalnummer angegeben ist, erfolgt der Zugriff immer auf Kanal 1 der Bedien-Software.

Wenn es notwendig ist, Daten aus einem speziellen Kanal zu lesen, wird der Adresse die Kennzeichnung **u** (Unit) mit der gewünschten Kanalnummer hinzugefügt.

Beispiel:

```
nck/Channel/MachineAxis/actFeedRate[3]
nck/Channel/MachineAxis/actFeedRate[u1, 3]
```

3.10.4 NC/PLC-Adressen zur Laufzeit erzeugen

Es besteht die Möglichkeit einen Adressbezeichner zur Laufzeit zu erstellen.

Dabei wird der Inhalt einer String-Variablen als Adresse in einer Operationsanweisung sowie in den Funktionen nc.cap.read und nc.cap.write genutzt.

Für diesen Adressierungsmode folgendes beachten:

- Den Variablennamen in Anführungszeichen schreiben.
- Dem Variablennamen drei ‚$'-Zeichen voranstellen.

Syntax:

```
"$$$variable name"
```

Beispiel:

```
<PRINT name="var_adr" text="DB9000.DBW%d"> 2000</PRINT>
<OP> "$$$var_adr" = 1 </OP>
```

3.10.5 Antriebskomponenten adressieren

Das Adressieren der Antriebskomponenten beginnt mit dem Pfadanteil **drive**.

Es folgt die Spezifizierung des Antriebgerätes:

CU – Control Unit

DC – Drive Control (Motormodul)

CULNK – Erweiterungsmodule (HUBs)

TM – Terminal-Module

LM – Line-Module

Diesem Anteil wird der zu setzende Parameter angefügt.

Beispiel:

```
<LET name="r0002_content"></LET>
<LET name="p107_content"></LET>

<!— Lesen des Werts r0002 auf der CU ->

<OP> r0002_content = "drive/cu/r0002" </OP>
<OP> r0002_content = "drive/cu/r0002[CU1]" </OP>

<!— Lesen des Werts r0002 auf der NX1 ->
<OP> r0002_content = "drive/cu/r0002[CU2]" </OP>

<!— Lesen des Werts p107[0] auf der CU ->
<OP> p107_content = "drive/cu/p107[0]" </OP>
<PRINT text="%d"> p107_content </PRINT>

<!— Lesen des Werts p107[0] auf der CU ->

<OP> p107_content = "drive/cu/p107[0, CU1]" </OP>
<PRINT text="%d"> p107_content </PRINT>

<!— Lesen des Werts p107[0] auf der NX1 ->

<OP> p107_content = "drive/cu/p107[0, CU2]" </OP>
<PRINT text="%d"> p107_content </PRINT>
```

Adressierung der Antriebsobjekte:

Zum Adressieren der einzelnen Objekte, ist nach dem Parameter das gewünschte Objekt in eckigen Klammern anzugeben.

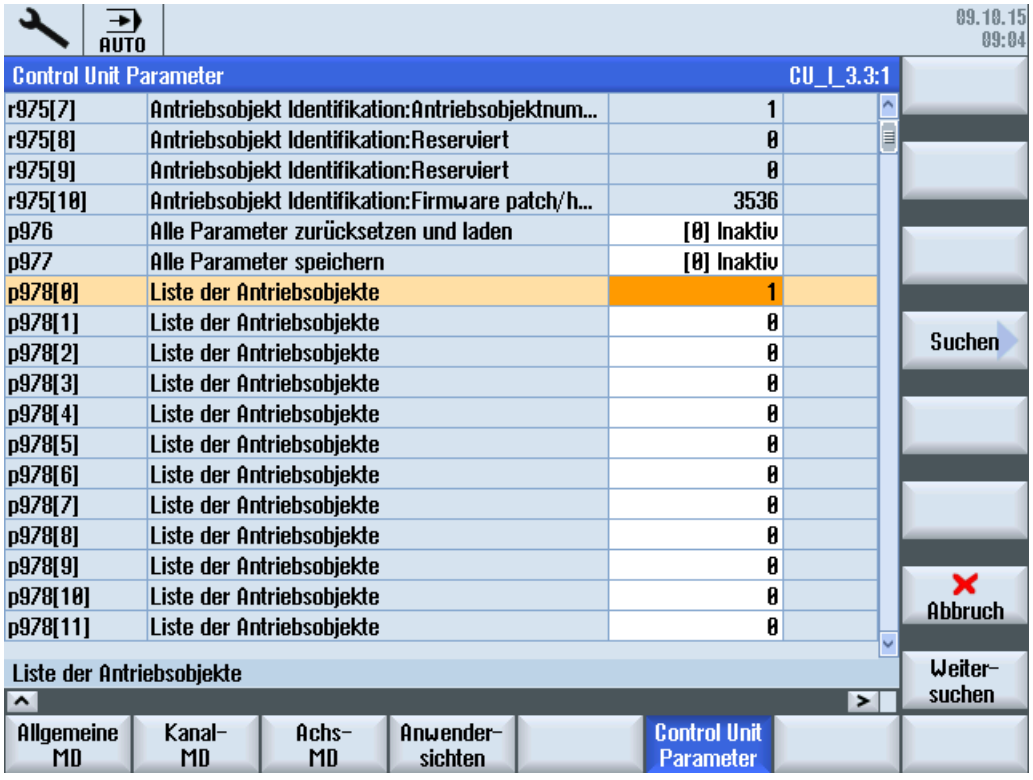


Bild 3-7 Antriebsparameter p0978 [...] an der Steuerung

```
Parameternummer[do<DO-index>]
```

Beispiel:

```
p0092[do1]
```

Alternativ kann der Drive-Index mittels "Ersetzungszeichen" **$<Variablenname>** aus einer lokalen Variablen gelesen werden.

```
z.B. DO$lokaleVariable
```

Beispiel:

```
<DATA name ="drive/cu/p0092">1</DATA>
<DATA name ="drive/dc/p0092[do1] ">1</DATA>
```

Indirekte Adressierung:

```
<LET name = "driveIndex> 0 </LET>
<OP> driveIndex = $ctrlout_module_nr[0, AX1] </OP>
<DATA name ="drive/dc[do$driveIndex]/p0092">1</DATA>
```

3.10.6 Beispiel: DO-Nummer für Motormodul ermitteln

Die DO-Nummer eines Motormoduls Typ 11 (Servo) kann wie folgt ermittelt werden:

Alle angeschlossenen Drive-Objekte werden ihrer Steckplatznummer im Feld p978 der entsprechenden CU aufgelistet. Gleichzeitig erfolgt das Auflisten der Komponentennummern im Feld p101 und das der Komponententypen im Feld p107.

Für die nachfolgenden Komponententypen ist jeweils eine eigene Indizierung zu verwenden:

CU – Control Units

DC – Drive Controls (Motormodul)

CULNK – Erweiterungsmodule (HUBs)

TM – Terminal-Module

LM – Line-Module

Der Adressierungsindex lässt sich ermitteln, indem je angeschlossener CU das Feld p0107 aufsteigend durchlaufen und mit jedem Auftreten des gesuchten Typs der Typindex um eins inkrementiert wird. Der Basiswert ist eins. Werden in diesem Feld NX-Komponenten gefunden, wird die Zählung auf einer NX-Komponente erst dann fortgesetzt, wenn das aktuelle Array vollständig durchlaufen wurde. Das Abarbeiten der NX- und CU-Komponenten erfolgt in der vorgefundenen Reihenfolge.

Indexermittlung: CUI mit NX

CUI		NX1		Adressierungsindex	
				DO	CU
p107[0]	[3]SINAMICS				1
p107[2]	[11]SERVO			1	
p107[3]	[11]SERVO			2	
p107[4]	[11]SERVO			3	
p107[5]	[254]CU-LINK				2
p107[6]	[11]SERVO			4	
		p107[1]	[11]SERVO	5	
		p107[2]	[11]SERVO	6	
		p107[3]	[11]SERVO	7	

In dieser Topologie sind sieben Motormodule enthalten. Die Indizes eins bis vier adressieren die Motormodule, die der CU zugeordnet sind. Die Indizes fünf bis sieben adressieren die Motormodule der NX. Für den Zugriff auf die CU ist der Index eins zu verwenden. Die NX wird mit dem Index zwei adressiert.

Beispielskripte: xmldial.xml und drv_sys_hlpfunct.xml

xmldial.xml

```
<DialogGui>

 <?include src="f:\appl\drv_sys_hlpfunct.xml" ?>

 <menu name = "main">
  <open_form name = "main_form" />
  <softkey POSITION="1">
   <caption></caption>
   <navigation>main</navigation>
  </softkey>
 </menu>

 <form name="main_form">
  <init>
   <caption>Component arrangement</caption>
    <let name="count" />
    <let name="str" type="string" />
    <let name="do_name" type="string" />
    <let name="cu_name" type="string" />
    <let name="cui_idx" />

    <op>
     cui_idx = 0;
    </op>

    <function name="load_component" />
    <print text="%d CUs found">num_cus</print>

    <function name="calculate_do_index" />

    <control name="list_comp_no_do_idx" xpos="8" ypos="80"
fieldtype="listbox" width="360" height="500" >
     <property item_data="100" />
    </control>

    <op>
     count = 0;
    </op>

    <while>
    <condition>count &lt; 32 && address_idx_map[$count].comp_no != 0</
condition>

    <op>
     do_name= _T"";
    </op>

    <function name="read_do_name_fast" return="do_name">count</function>
    <function name="read_cu_name_fast"
return="cu_name">address_idx_map[$count].cu_idx</function>

    <print name="str" text="%3d %3d %3d %s
%s">address_idx_map[$count].cu_idx, address_idx_map[$count].comp_no,
address_idx_map[$count].do_idx, do_name, cu_name</print>
```

xmldial.xml

```
    <function name="control.additem">_T"list_comp_no_do_idx", str, count</
function>

    <op>
     count = count +1;
    </op>
    </while>

   <paint>
    <text xpos="8" ypos="30">Motor moduls</text>
    <text xpos="8" ypos="60">CU</text>
    <text xpos="40" ypos="60">Comp.</text>
    <text xpos="92" ypos="60">DO Index</text>
    <text xpos="172" ypos="60">Name</text>
  </paint>

 </form>
</DialogGui>
```

drv_sys_hlpfunct.xml

```
<typedef name="components" type="struct">
 <element name="cu_p978" dim="25" />
 <element name="do_p0101" dim="25" />
 <element name="do_p0107" dim="25" />
</typedef>

<typedef name="comp_no_do_idx_map" type="struct">
 <!-- cu index -->
 <element name="cu_idx" />
 <!-- component number -->
 <element name="comp_no" />
 <!-- address index -->
 <element name="do_idx" />
</typedef>

<let name="componentsList" dim="5" type="components" />
<let name="address_idx_map" dim="32" type="comp_no_do_idx_map" />
<let name="num_cus" />
<let name="_drv_sys_comp_array_size">23</let>

<!-- ------------------------------------------------------------

 function: load_components_description
 This function loads the parameter arrays p978, p101 and p107 into a local
memory

 input:
 cuno: CU index 0 based (index 0 == CUI)

 output:
 componentsList structure

 ------------------------------------------------------------------- -->

<function_body name="load_components_description" parameter="cuno">
 <let name="count" />
 <let name="next_nx" >0</let>
 <let name="cuidx"></let>
 <let name="error" />

 <op>
  count = _drv_sys_comp_array_size;
  next_nx = cuno;
  cuidx = cuno+1;
 </op>

 <print text="gather data" />

 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].cu_p978, "drive/cu/p0978[0, cu
$cuidx]"</function>
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].do_p0101, "drive/cu/p0101[0, cu
$cuidx]"</function>
```

drv_sys_hlpfunct.xml

```
 <function name="ncfunc.cap.read" return="error"
rows="$count">componentsList[$cuno].do_p0107, "drive/cu/p0107[0, cu
$cuidx]"</function>

 <print text="gather data finished" />
 <sleep value="20" />

 <op>
  count = 0;
 </op>

 <while>
  <condition>count < _drv_sys_comp_array_size</condition>
  <if>
   <condition>componentsList[$cuno].cu_p978[$count] == 60</condition>
  <then>
   <op>
    next_nx= next_nx +1;
   </op>
  <print text="next nx %d">next_nx</print>
  <function name="load_components_description">next_nx</function>
  </then>
  </if>

 <op>
  count = count+1;
 </op>
 </while>
 <op>
  num_cus = next_nx+1;
 </op>
</function_body>

<!-- ------------------------------------------------------------

 function: calculate_do_index
 This function is looking for components of type 11 (SERVO) and lists these
in the componentsList array.

 input:
 -

 output:
 componentsList array filled

 ------------------------------------------------------------------- -->

<function_body name="calculate_do_index">
 <let name="cuno" />
 <let name="count" />
 <let name="do_index" >1</let>
 <let name="map_index" />
 <while>
  <condition>
   cuno < num_cus</condition>
   <op>
```

drv_sys_hlpfunct.xml

```
    count = 0;
   </op>
  <while>
   <condition>count < _drv_sys_comp_array_size</condition>
   <if>
    <condition> componentsList[$cuno].do_p0107[$count] == 11</condition>
    <then>
     <op>
      address_idx_map[$map_index].cu_idx = cuno+1;
      address_idx_map[$map_index].comp_no =
componentsList[$cuno].do_p0101[$count];
      address_idx_map[$map_index].do_idx = do_index;
      do_index = do_index+1;
      map_index = map_index +1;
     </op>
    </then>
   </if>
   <op>
    count = count +1;
   </op>
  </while>
  <op>
   cuno = cuno +1;
  </op>
 </while>
</function_body>
```

3.10.7 Maschinen- und Settingdaten adressieren

Antriebs- und Settingdaten werden mit dem $-Zeichen gekennzeichnet, gefolgt vom Namen des Datums.

Maschinendaten:

$Mx_<Name[**index, AX<Achsnummer>**]>

Settingdaten:

$Sx_<Name[index, AX<Achsnummer>]>

x:

N – allgemeine Maschinen- oder Settingdaten

C – kanalspezifische Maschinen- oder Settingdaten

A – achsspezifische Maschinen- oder Settingdaten

Index:

Bei einem Feld gibt der Parameter den Index des Datums an.

AX<Achsnummer>:

Bei achsspezifischen Daten ist die gewünschte Achse (**<Achsnummer>**) zu spezifizieren.

Alternativ kann der Achsindex mittels "Ersetzungszeichen" **$<Variablenname>** aus einer lokalen Variablen gelesen werden.

z. B. AX$lokaleVariable

Beispiel:

```
<DATA name ="$MN_AXCONF_MACHAX_NAME_TAB[0] ">X1</DATA>
```

Direkte Adressierung der Achse:

```
<DATA name ="$MA_CTRLOUT_MODULE_NR[0, AX1] ">1</DATA>
...
...
```

Indirekte Adressierung der Achse:

```
<LET name ="axisIndex"> 1 </LET>
<DATA name ="$MA_CTRLOUT_MODULE_NR[0, AX$axisIndex] ">1</DATA>
```

3.10.8 Kanalspezifische Maschinendaten

Wenn im Adress-Token keine Kanalnummer angegeben ist, erfolgt der Zugriff immer auf den aktuell eingestellten Kanal der Bedien-Software.

Wenn es notwendig ist, Daten aus einem speziellen Kanal zu lesen, wird der Adresse die Kennzeichnung **u** (Unit) mit der gewünschten Kanalnummer in eckigen Klammern hinzugefügt. Bei einer Array-Adressierung erfolgt die Kanalangabe als letztes Argument in den eckigen Klammern.

Beispiel:

```
$MC_RESET_MODE_MASK
```

oder

```
$MC_AXCONF_GEOAX_ASSIGN_TAB[0]
```

```
$MC_RESET_MODE_MASK[u1]
```

oder

```
$MC_AXCONF_GEOAX_ASSIGN_TAB[0, u1]
$MC_RESET_MODE_MASK
```

3.10.9 Anwenderdaten adressieren

Das Adressieren der Anwenderdaten beginnt mit dem Pfadanteil **gud**, gefolgt von der Kennzeichnung, ob es sich um globale oder kanalspezifische GUDs handelt. Diesem Adressteil schließt sich der GUD-Bereich und der GUD-Name an.

Bei einem Feld ist nach dem Namen, der gewünschte Feldindex in eckigen Klammern anzugeben.

Beispiel:
```
<DATA name ="gud/nck/mgud/syg_rm[0]" />
<OP>"gud/nck/mgud /syg_rm[0]" = 10 </op>
```

Adressierung der globalen Anwenderdaten

Die Adressierung beginnt mit dem Pfadanteil gud, gefolgt von der Bereichsspezifikation NCK. Diesem Adressteil folgt die Spezifizierung der GUD-Bereiche:

GUD Bereiche	Zuordnung
sgud	GUD von Siemens
mgud	GUD des Maschinenherstellers
ugud	GUD des Anwenders

Adressierung der kanalspezifischen Anwenderdaten

Die Adressierung beginnt mit dem Pfadanteil gud, gefolgt von der Bereichsspezifikation CHANNEL. Diesem Adressteil folgt die Spezifizierung der GUD-Bereiche.

Anschließend ist der GUD-Name anzugeben. Ist ein Feld zu adressieren, folgt dem Namen der Feldindex in eckigen Klammern.

Beispiel:
```
<data name ="gud/channel/mgud/syg_rm[0]">1</data>
<op>"gud/channel/mgud/syg_rm[0]" = 5*2 </op>
```

Adressierung von mehrdimensionalen Arrays

Bei der Adressierung von mehrdimensionalen Arrays wird der Zeilenindex gefolgt vom Spaltenindex erwartet. Die Indices sind durch einen Punkt voneinander getrennt anzugeben.

Für kanalspezifische GUDs gilt:

- Die Kanalnummer ist mit der Kennzeichnung **u** (Unit) gefolgt von der Nummer vor oder nach der Zeilen/Spalten-Spezifikation zu notieren.
- Die Sequenzen sind durch ein Komma voneinander zu trennen.
- Wenn keine Kanalnummer angegeben wird, erfolgt der Zugriff auf den ersten Kanal.

Beispiel:
```
"gud/Channel/sgud/_WP[u2, 2.0]"
```

oder

```
"gud/Channel/sgud/_WP[2.0, u2]"
```

3.11 Vordefinierte Funktionen

Die Skriptsprache bietet verschiedene Stringverarbeitungs- und mathematische Standardfunktionen an. Die nachfolgend aufgeführten Funktionsnamen sind reserviert und können nicht überladen werden.

Funktionsname	Bedeutung
Ncfunc cap read	Die Funktion kopiert einen Wert aus der angegebenen Adresse in eine lokale Variable. Ist das Lesen fehlerfrei erfolgt, enthält die Return-Variable den Wert Null. Im Gegensatz zur Operationsanweisung bricht diese Funktion im Fehlerfall die Abarbeitung der Skript-Anweisungen nicht ab. **Attribute:** • **return** - Ausführungsstatus – Wert = 0 - fehlerfrei – Wert = 1 - Das Lesen der Variablen konnte nicht durchgeführt werden • **rows** - Anzahl der zusätzlich zu lesenden Zeilen eines Arrays (optional) Wenn bei der Referenzvariablen ein Arrayindex angegeben wird, kopiert die Funktion die gelesenen Werte ab diesem Index in die Zielvariable. **Syntax:** `<function name="ncfunc.cap.read" return="error"> lokale variable, "address"</function>` **Beispiel:** `<let name="error"></let>` `<function name="ncfunc.cap.read" return="error"> 3, "drive/cu/p0009"</function>` `<if>` `<condition>error != 0</condition>` `<then>` `<break />` `</then>` `</if>` oder `<let name="cu_p978" dim="25" ></let>` `...` `...` `<function name="ncfunc.cap.read" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"` `</function>`

Funktionsname	Bedeutung
Ncfunc cap write	Die Funktion schreibt einen Wert in die angegebene Variable. Ist das Schreiben fehlerfrei erfolgt, enthält die Return-Variable den Wert Null. Im Gegensatz zur Operationsanweisung bricht diese Funktion im Fehlerfall die Abarbeitung der Skript-Anweisungen nicht ab. **Attribute:** • **return** - Ausführungsstatus – Wert = 0 - fehlerfrei – Wert = 1 - Das Lesen der Variablen konnte nicht durchgeführt werden • **rows** - Anzahl der zusätzlich zu schreibenden Zeilen eines Arrays (optional) Wenn bei der Referenzvariablen ein Arrayindex angegeben wird, kopiert die Funktion ab diesem Index die Werte in die Zielvariable. **Syntax:** `<function name="ncfunc.cap.read" return="error"> local variable or constant, "address"</function>` **Beispiel:** `<let name="error"></let>` `<function name="ncfunc.cap.write" return="error"> 0, "drive/cu/p0009"</function>` `<if>` `<condition>error != 0</condition>` `<then>` `<break />` `</then>` `</if>` oder `<let name="cu_p978" dim="25" ></let>` `...` `...` `<function name="ncfunc.cap.write" return="error" rows="23">cu_p978, "drive/cu/p0978[0, 1]"` `</function>`

Funktionsname	Bedeutung
Ncfunc PI-Service	Mit dem Programminstanz (PI)-Dienst lassen sich Aufträge an den NCK übermitteln. Ist der Dienst fehlerfrei ausgeführt worden, liefert die Funktion den Wert 1 in der Return-Variablen zurück. **Manipulation der Werkzeugliste:** _N_CREATO - Werkzeug anlegen _N_DELETO - Werkzeug löschen _N_CREACE - Werkzeugschneide anlegen _N_DELECE - Werkzeugschneide löschen **Aktivieren von Nullpunktverschiebungen**: _N_SETUFR - Setzt den aktuellen User-Frame aktiv _N_SETUDT - Setzt die aktuellen User-Daten aktiv **Satzsuchlauf**: _N_FINDBL - Suchlauf aktivieren _N_FINDAB - Satzsuchlauf abbrechen **Umkonfiguration von Maschinendaten:** _N_CONFIG - Maschinendaten mit der Wirksamkeitsstufe NEW_CONF, die nacheinander vom Bediener eingegeben wurden, werden gleichzeitig aktiviert **Syntax:** `<function name="ncfunc.pi_service" return="return var"> pi name, var1, var2, var3, var4, var5` `</function>` **Attribute:** name - Funktionsname return - Name der Variablen, in der das Ausführungsergebnis abgelegt wird: • Wert == 1 - Auftrag erfolgreich ausgeführt • Wert == 0 - Auftrag fehlerhaft **Tagwerte:** pi name - Name des PI – Dienstes (String) var1 bis var5 - PI spezifische Argumente

Funktionsname	Bedeutung
Ncfunc PI-Service Fortsetzung	**Argumente:** • _N_CREATO var1 - Werkzeugnummer • _N_DELETO var1 - Werkzeugnummer • _N_CREACE var1 - Werkzeugnummer var2 - Schneidennummer • _N_DELECE var1 - Werkzeugnummer var2 - Schneidennummer • _N_SETUFR Keine Argumente • _N_SETUDT var1 - zu aktivierender Bereich der Anwenderdaten – 1 - Werkzeugkorrekturdaten – 2 - aktiver Basis-Frame – 3 - aktiver einstellbarer Frame • _N_FINDBL var1 - Suchlaufmode – 2 - Suchlauf mit Konturberechnung – 4 - Suchlauf auf den Satzendpunkt – 1 - Suchlauf ohne Berechnung • _N_FINDAB Keine Argumente • _N_CONFIG Keine Argumente **Beispiel:** Anlagen eines Werkzeuges - Werkzeugnummer 3 `<function name="ncfunc.pi_service">_T"_N_CREATO", 3</function>` Schneide 1 des Werkzeuges 5 löschen `<function name="ncfunc.pi_service">_T"_N_DELECE", 5, 1` `</function>`

Funktionsname	Bedeutung
ncfunc chan PI-Service	Die Funktion führt einen PI- Dienst kanalbezogen aus. Die Kanalnummer wird nach dem PI-Dienstnamen übergeben. Danach folgen alle anderen Aufrufparameter. **Parameter:** **channel** - Kanalnummer **Syntax:** `<function name="ncfunc.chan_pi_service" return="error">` `_T"_N_SETUFR", channel, ...` `</function>` **Beispiel:** `<let name="chan" >1</let>` `<function name="ncfunc.chan_pi_service" return="error">` `_T"_N_SETUFR", chan</function>` `<function name="ncfunc.chan_pi_service" return="error">` `_T"_N_SETUDT", chan, _T"016", _T"00000", _T"00000"</function>`
Ncfunc display resolution	Die Funktion liefert die in der Steuerung festgelegte Konvertierungsvorschrift für Gleitkommazahlen. Als Variable ist eine Stringvariable bereitzustellen. Siehe auch Anzeigemaschinendaten MD203 DISPLAY_RESOLUTION und MD204 DISPLAY_RESOLUTION_INCH **Syntax:** `<function name="ncfunc.displayresolution" return="dislay_res" />` **Beispiel:** `<let name="dislay_res" type="string"></let>` ... `<function name="ncfunc.displayresolution" return="dislay_res" />` `  <control name = "cdistToGo" xpos = "210" ypos = "156" refvar="nck/Channel/GeometricAxis/progDistToGo[2]" hotlink="true" height="34" fieldtype="readonly" format="$$$dislay_res" time="superfast" color_bk="#ffffff"/>`

Funktionsname	Bedeutung
Ncfunc Get drive by axis name	Die Funktion liefert den Adressierungsindex eines Motormoduls durch die Angabe des zugehörigen Achsnames. Die Funktion liefert den Wert -1 zurück, wenn die Zuordnung nicht ermittelt werden kann. Dies kann z. B. vorkommen, wenn die Achs-/Antriebszuordnung nicht erfolgt ist. **Parameter:** **str** - Achsname **Rückgabewert:** Index des Motormoduls - Name auf eine Variable, in die der ermittelte Index geschrieben wird. **Syntax:** `<function name="NCFUNC.GETDRVBYAX_NAME" return="<index>">` `<axis name></function>` **Beispiel:** `<let name="drv_idx" />` `<function name="NCFUNC.GETDRVBYAX_NAME" return="drv_idx">_T"X1"` `</function>` `<control name = "c_do2" xpos = "140" ypos = "114" width = "320"` `fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]"` `hotlink="true" color_bk="#f1f1f1">` ` <item value="0">Ready</item>` ` <item value="1">Quick commissioning</item>` ` <item value="2">Power unit commissioning</item>` ` <item value="3">Motor commissioning</item>` ` <item value="4">Encoder commissioning</item>` ` <item value="5">Technological application/units` `</item>` ` <item value="15">Data sets</item>` ` <item value="17">Basic positioning commissioning` `</item>` ` <item value="25">Commissioning the position control</item>` ` <item value="29">Download</item>` ` <item value="30">Parameter reset</item>` ` <item value="95">Safety Integrated commissioning` `</item>` `</control>` `</init>`

Funktionsname	Bedeutung
Ncfunc Get drive by axis index	Die Funktion liefert den Adressierungsindex eines Motormoduls durch die Angabe des zugehörigen Achsindexes. Die Funktion liefert den Wert -1 zurück, wenn die Zuordnung nicht ermittelt werden kann. Dies kann z. B. vorkommen, wenn die Achs-/Antriebszuordnung nicht erfolgt ist. **Parameter:** **index** - Achsindex **Rückgabewert:** Index des Motormoduls - Name auf eine Variable, in die der ermittelte Index geschrieben wird. **Syntax:** `<function name="NCFUNC.GETDRVBYAX_IDX" return="<index>"> <axis index></function>` **Beispiel:** `<let name="drv_idx" /> <function name="NCFUNC.GETDRVBYAX_IDX" return="drv_idx">1</function> <control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1"> <item value="0">Ready</item> <item value="1">Quick commissioning</item> <item value="2">Power unit commissioning</item> <item value="3">Motor commissioning</item> <item value="4">Encoder commissioning</item> <item value="5">Technological application/units </item> <item value="15">Data sets</item> <item value="17">Basic positioning commissioning </item> <item value="25">Commissioning the position control</item> <item value="29">Download</item> <item value="30">Parameter reset</item> <item value="95">Safety Integrated commissioning </item> </control> </init>`

Funktionsname	Bedeutung
Ncfunc Get drive by drive name	Die Funktion liefert den Adressierungsindex eines Motormoduls durch die Angabe des Motormodulnamens. Die Funktion liefert den Wert -1 zurück, wenn die Zuordnung nicht ermittelt werden kann. Dies kann z. B. vorkommen, wenn die Achs-/Antriebszuordnung nicht erfolgt ist. **Parameter:** **string** - Name des Motormoduls **Rückgabewert:** Index des Motormoduls - Name auf eine Variable, in die der ermittelte Index geschrieben wird. **Syntax:** `<function name="NCFUNC.GETDRVBYDRV_NAME" return="<index>"> <drive name></function>` **Beispiel:** `<let name="drv_idx" /> <function name="NCFUNC.GETDRVBYDRV_NAME" return="drv_idx">_T"SERVO_3.13:4"</function> <control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1"> <item value="0">Ready</item> <item value="1">Quick commissioning</item> <item value="2">Power unit commissioning</item> <item value="3">Motor commissioning</item> <item value="4">Encoder commissioning</item> <item value="5">Technological application/units </item> <item value="15">Data sets</item> <item value="17">Basic positioning commissioning </item> <item value="25">Commissioning the position control</item> <item value="29">Download</item> <item value="30">Parameter reset</item> <item value="95">Safety Integrated commissioning </item> </control> </init>`

Funktionsname	Bedeutung
Ncfunc Get drive by bud address	Die Funktion liefert den Adressierungsindex eines Motormoduls durch die Angabe Busadresse des Morotmoduls. Die Funktion liefert den Wert -1 zurück, wenn die Zuordnung nicht ermittelt werden kann. Dies kann z. B. vorkommen, wenn die Achs-/Antriebszuordnung nicht erfolgt ist. **Parameter:** **bus** - Bus-Nummer **slave** - Slave-Nummer **component** - Komponentennummer **Rückgabewert:** Index des Motormoduls - Name auf eine Variable, in die der ermittelte Index geschrieben wird. **Syntax:** `<function name="NCFUNC.GETDRVBYBUS_ADDR" return="<index>"><bus>,<slave>,<component></function>` **Beispiel:** `<let name="drv_idx" />` `<function name="NCFUNC.GETDRVBYBUS_ADDR" return="drv_idx"> 3,13,4 </function>` `<control name = "c_do2" xpos = "140" ypos = "114" width = "320" fieldtype="itemlist" refvar="drive/dc/p10[`**`do$ drv_idx`**`]" hotlink="true" color_bk="#f1f1f1">` `<item value="0">Ready</item>` `<item value="1">Quick commissioning</item>` `<item value="2">Power unit commissioning</item>` `<item value="3">Motor commissioning</item>` `<item value="4">Encoder commissioning</item>` `<item value="5">Technological application/units </item>` `<item value="15">Data sets</item>` `<item value="17">Basic positioning commissioning </item>` `<item value="25">Commissioning the position control</item>` `<item value="29">Download</item>` `<item value="30">Parameter reset</item>` `<item value="95">Safety Integrated commissioning </item>` `</control>` `</init>`

Funktionsname	Bedeutung
Ncfunc Get MD limits	Die Funktion kopiert die Limits eines Maschinendatums in die angegebene Variable vom Datentyp **StructMDLimits**. Die Strukturdefinition ist in der Datei struct_def.xml enthalten. **Weitere Informationen** unter Kapitel "Datei struct_def.xml erstellen (Seite 141)" **Parameter:** **range** - Es ist ein String zu übergeben, der den Maschinendatenbereich spezifiziert: • **displayMD** - Anzeigemaschinendaten • **generalMD** - NC globale Maschinendaten • **channelMD** - NC kanalspezifische Maschinendaten • **axisMD** - NC achsspezifische Maschinendaten • **generalSettingMD** - NC globale Setting-Daten • **channelSettingMD** - NC kanalspezifische Setting-Daten • **axisSettingMD** - NC achsspezifische Setting-Daten • **channelGUD** - NC kanalspezifische Anwenderdaten • **globalGUD** - NC globale Anwenderdaten **MD-number** - Maschinendatennummer des Bereichs **variabel name** - Nach dem Funktionsaufruf beinhaltet diese Variable die Werte der Limits **range/unit** - optional (z. B. Kanalnummer) **Syntax:** `<function name="NCFUNC.GETMD_LIMITS"><range>, <MD-number, <variabel name>, <range/unit>></function>` **Beispiel:** `<let name="limits" type="StructMDLimits" /> <function name="NCFUNC.GETMD_LIMITS">_T"generalMD", 19220, _T"limits"</function> <op> upper_BAGS = limits.upperLimit; </op>`
Ncfunc bico to int	Die Funktion konvertiert einen im BICO-Format angegebenen String in einen Integerwert. (siehe SINAMICS). **Syntax:** `<function name="ncfunc.bicotoint" return="integer variable"> bico-string</function>` **Beispiel:** `<let name="s_np0480_0" type="string"></let> <let name="i_p0480_0">0</let>` `<function name="ncfunc.bicotoint" return="i_p0480_0">s_np0480_0 </function>`

Funktionsname	Bedeutung
Ncfunc int to bico	Die Funktion konvertiert einen Integerwert in einen BICO-Format -String. (siehe SINAMICS). **Syntax:** `<function name="ncfunc.inttobico"` `return="string variable">integer variable</function>` **Beispiel:** `<function name="ncfunc.inttobico"` `return="s_p0480_0">"drive/dc/p0480[0, DO2]"</function>`
Ncfunc is bico str valid	Die Funktion liefert den Wert Null, wenn es sich um einen im BICO-Format angegebenen String handelt. (siehe SINAMICS) **Syntax:** `<function name="ncfunc.isbicostrvalid"` `return="integer variable">string varaible</function>` **Beispiel:** `…` `<let name="s_np0480_0" type="string"></let>` `…` `…` `<control name = "cp0480_0" xpos = "402"` `ypos = "76" hotlink="true" refvar="s_np0480_0" >` `  <property item_data="4001" />` `</control>` `…` `…` `<function name="ncfunc.isbicostrvalid"` `return="valid">cp0480_0</function>`
Ncfunc password	Die Funktion setzt oder löscht eine Passwortstufe der NC. • Passwort setzen: Als Parameter ist das Passwort für die gewünschte Passwortstufe anzugeben. • Passwort löschen: Ein Leerstring löscht die Passwortstufe. **Syntax:** `<function name="ncfunc.password">password` `</function>` **Beispiel:** `<let name="password" type="string"></let>` `<function name="ncfunc.password" > password` `</function>` `<function name="ncfunc.password" > _T"CUSTOMER"` `</function>` Passwort löschen: `<function name="ncfunc.password" > _T"" </function>`

Funktionsname	Bedeutung
Control form color	Die Funktion liefert die Text- oder Hintergrundfarbe der Dialogbox als String. **Weitere Informationen** unter Kapitel "Farbcodierung (Seite 74)" **Bereich:** • BACKGROUND – Farbwert des Hintergrunds anfordern • TEXT – Farbwert des Textes (Vordergrund) anfordern **Syntax:** `<function name="control.formcolor" return="variable">_T"Bereich"</function>` **Beispiel:** `<let name="bk_color" type="string"></let>` `<function name="control.formcolor" return="bk_color">_T"BACKGROUND"</function>`
Control local time	Die Funktion kopiert die lokale Zeit in ein Feld mit 7 Feldelementen. Als Aufrufparameter wird der Name der Variablen erwartet. Folgendes wird im Feldelement abgelegt: • Index 0 - Jahr • Index 1 - Monat • Index 2 - Wochentag • Index 3 - Tag • Index 4 - Stunde • Index 5 - Minute • Index 6 - Sekunde **Syntax:** `<function name ="control.localtime">_T"time_array" </function>` **Beispiel:** `<!-- index` `0 = Year` `1 = Month` `2 = Day of week` `3 = Day` `4 = Hour` `5 = Minute` `6 = Second` `-->` `<let name="time_array" dim="7" />` `<function name ="control.localtime">_T"time_array" </function>`

Funktionsname	Bedeutung
Control exist	Die Funktion liefert den Wert 1 zurück, wenn das angegebene Control existiert. **Syntax:** `<function name="CONTROL.EXIST"` `return="<return variable>"><control name></function>` **Beispiel:** `<let name="ret" /> <function name="CONTROL.EXIST"` `return="ret">_T"edit"</function>`
Control is modified	Die Funktion liefert den Wert 1 zurück, wenn der Inhalt des angegebenen Edit-Controls geändert wurde. **Syntax:** `<function name="CONTROL.ISMODIFIED"` `return="<return variable>"><control name></function>` **Beispiel:** `<let name="ret" /> <function name="CONTROL.ISMODIFIED"` `return="ret">_T"edit"</function>`
Control is visible	Die Funktion liefert den Wert 1 zurück, wenn das angegebene Control sichtbar ist. **Syntax:** `<function name="CONTROL.ISVISIBLE"` `return="<return variable>"><control name></function>` **Beispiel:** `<let name="ret" /><function name="CONTROL.ISVISIBLE"` `return="ret">_T"edit"</function>`
Control set modified	Die Funktion ändert das Modified-Flag des angegebenen Edit-Controls. **Parameter:** **control name** • Wert = 1 - Feld als geändert kennzeichnen • Wert = 0 - Feld als ungeändert kennzeichnen **Syntax:** `<function name="CONTROL.SETMODIFIED"` `return="<return variable>"><control name>,` `<Flag></function>` **Beispiel:** `<let name="b" >1</let>` `<let name="ret" />` `<control name="edit" xpos="10" ypos="80" width="30" refvar="b"` `hotlink = "true" />` `<function name="CONTROL.SETMODIFIED"` `return="ret">_T"edit", 1</function>`

Funktionsname	Bedeutung
Control set working area	Ändert man den Inhalt einer **Scrollview**, z. B. durch Löschen oder Hinzufügen von Controls, ist der sichtbare Bereich der Scrollview neu zu berechnen. Durch den Aufruf dieser Funktion wird die Berechnung durchgeführt. **Wert:** Name der **Scrollview** **Beispiel:** `...` `<control name="area" xpos="8" ypos="72" width="554" height="320" fieldtype="scrollarea" >` `  <control name="test" xpos="20" ypos="40"  width="20" fieldtype="readonly" />` `</control >` `<control name="test1" xpos="20" ypos="80"  width="20" fieldtype="readonly" parent="area"/>` `<control name="test2" xpos="20" ypos="100"  width="20" fieldtype="readonly" parent="area"/>` `<function name="CONTROL.SETWORKINGAREA">_T"area"</function>` `...`
String to compare	Es werden zwei Strings lexikographisch miteinander verglichen. Die Funktion liefert den Rückgabewert Null, wenn die Strings gleich sind, kleiner Null, wenn der erste String kleiner als der zweite String ist oder größer Null, wenn der zweite String kleiner als der erste String ist. **Parameter:** **str1** - String **str2** - Vergleichsstring **Syntax:** `<function name="string.cmp" return ="<int var>" > str1, str2 </function>` **Beispiel:** `<let name="rval">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string">A brown bear hunts a brown dog.</let>` `<function name="string.cmp" return="rval"> str1, str2 </function>` **Ergebnis:** rval= 0

Funktionsname	Bedeutung
String to compare ohne Unterscheidung der Groß-/Kleinschreibung	Es werden zwei Strings lexikographisch ohne Unterscheidung der Groß-/Kleinschreibung miteinander verglichen. Die Funktion liefert den Rückgabewert Null, wenn die Strings gleich sind, kleiner Null, wenn der erste String kleiner als der zweite String ist oder größer Null, wenn der zweite String kleiner als der erste String ist. **Parameter:** **str1** - String **str2** - Vergleichsstring **Syntax:** `<function name="string.icmp" return ="<int var>" > str1, str2 </function>` **Beispiel:** `<let name="rval">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string">A brown Bear hunts a brown Dog.</let>` `<function name="string.icmp" return="rval"> str1, str2 </function>` **Ergebnis:** rval= 0
String left	Die Funktion extrahiert die ersten nCount Zeichen von String 1 und kopiert diese in die Returnvariable. **Parameter:** **str1** - String **nCount** - Anzahl Zeichen **Syntax:** `<function name="string.left" return="<result string>"> str1, nCount </function>` **Beispiel:** `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string"></let>` `<function name="string.left" return="str2"> str1, 12 </function>` **Ergebnis:** str2="A brown bear"

Funktionsname	Bedeutung
String right	Die Funktion extrahiert die letzten nCount Zeichen von String 1 und kopiert diese in die Returnvariable. **Parameter:** **str1** - String **nCount** - Anzahl Zeichen **Syntax:** `<function name="string.right" return="<result string>"> str1, nCount </function>` **Beispiel:** `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string"></let>` `<function name="string.right" return="str2"> str1, 10 </function>` **Ergebnis:** str2="brown dog."
String middle	Die Funktion extrahiert ab dem Index iFirst die angegebene Anzahl Zeichen von String 1 und kopiert diese in die Returnvariable. **Parameter:** **str1** - String **iFirst** - Startindex **nCount** - Anzahl Zeichen **Syntax:** `<function name="string.middle"` `return="<result string>"> str1, iFirst, nCount </function>` **Beispiel:** `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<let name="str2" type="string"></let>` `<function name="string.middle" return="str2"> str1, 2, 5 </function>` **Ergebnis:** str2="brown"

Funktionsname	Bedeutung
String length	Die Funktion liefert die Anzahl Zeichen eines Strings. **Parameter:** **str1** - String **Syntax:** `<function name="string.length" return="<int var>"> str1 </function>` **Beispiel:** `<let name="length">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog.</let>` `<function name="string.length" return="length"> str1 </function>` **Ergebnis:** length = 31
Strings to replace	Die Funktion ersetzt alle gefundenen Teilstrings mit dem neuen String. **Parameter:** **string** - Stringvariable **find string** - zu ersetzender String **new string** - neuer String **Syntax:** `<function name="string.replace"> string, find string, new string </function>` **Beispiel:** `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.replace" > str1, _T"a brown dog" , _T"a big salmon"</function>` **Ergebnis:** str1 = "A brown bear hunts a big salmon!"

Funktionsname	Bedeutung
Strings to remove	Die Funktion löscht alle gefundenen Teilstrings. **Parameter:** **string** - Stringvariable **remove string** - zu löschender Teilstring **Syntax:** `<function name="string.remove"> string, remove string </function>` **Beispiel:** `<let name="index">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.remove" > str1, _T"a brown dog" </function>` **Ergebnis:** str1 = "A brown bear hunts"
Strings to insert	Die Funktion fügt einen String am angegebenen Index ein. **Parameter:** **string** - Stringvariable **index** - Index (Null basierend) **insert string** - einzufügender String **Syntax:** `<function name="string.insert"> string, index, insert string` `</function>` **Beispiel:** `<let name="str1" type="string">A brown bear hunts. </let>` `<let name="str2" type="string">a brown dog</let>` `<function name="string.insert" > str1, 19, str2` `</function>` **Ergebnis:** str1 = "A brown bear hunts a brown dog"

Funktionsname	Bedeutung
String delete	Die Funktion löscht ab der angegebenen Startposition die festgelegte Anzahl Zeichen. **Parameter:** **string** - Stringvariable **start index** - Startindex (Null basierend) **nCount** - Anzahl zu löschender Zeichen **Syntax:** `<function name="string.delete"> string, start index , nCount` `</function>` **Beispiel:** `<let name="str1" type="string">A brown bear hunts. </let>` `<function name="string.delete" > str1, 2, 5` `</function>` **Ergebnis:** str1 = "A bear hunts"
String find	Die Funktion sucht im übergebenen String nach der ersten Übereinstimmung mit dem Teilstring. Wurde der Teilstring gefunden, liefert die Funktion den Index auf das erste Zeichen (mit Null beginnend), ansonsten -1. **Parameter:** **string** - String-Variable **findstring** - zu suchender String **startindex** – Startindex (optional) **Syntax:** `<function name="string.find" return="<int val>"> str1, find string` `</function>` **Beispiel:** `<let name="index">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.find" return="index"> str1, _T"brown"` `</function>` **Ergebnis:** Index = 2 **oder** `<function name="string.find" return="index"> str1, _T"brown", 1` `</function>`

Funktionsname	Bedeutung
String reverse find	Die Funktion sucht im übergebenen String nach der letzten Übereinstimmung mit dem Teilstring. Wurde der Teilstring gefunden, liefert die Funktion den Index auf das erste Zeichen (mit Null beginnend), ansonsten -1. **Parameter:** **string** - String-Variable **find string** - zu suchender String **startindex** - Startindex (optional) **Syntax:** `<function name="string.reversefind"` `return="<int val>"> str1, find string </function>` **Beispiel:** `<let name="index">0</let>` `<let name="str1" type="string">A brown bear hunts a brown dog. </let>` `<function name="string.reversefind" return="index"> str1, _T"brown"` `</function>` **Ergebnis:** Index = 21 **oder** `<function name="string.reversefind"` `return="index"> str1, _T"brown" 10 </function>` **Ergebnis:** Index = 2
String GetAt	Die Funktion liest ein Zeichen von der spezifizierten Position. **Parameter:** **str** - String **index** - Null basierender Index auf das zu lesende Zeichen **Rückgabewert:** Es ist der Name einer String-Variablen anzugeben, in der das Zeichen abgelegt werden soll. **Syntax:** `<function name="string.getat" return="<result string>"><string>,` `<index></function>` **Beispiel:** `<let name="resStr" type="string" />` `<function name="string.getat"` `return="resStr">_T"brown", 2</function>`

Funktionsname	Bedeutung
String pixel height	Berechnung der Höhe einer Zeichenkette in Pixel. Die Berechnung wird unter Verwendung des aktuellen Schriftfonts des angegebenen Controls oder der Form durchgeführt. Der Name des Controls ist als String zu übergeben. Wird ein Leerstring angegeben, bezieht sich die Berechnung auf die Form. **Parameter:** **control name** **Zeichenkette** Die Zeichenkette kann direkt als Text angegeben werden oder es ist eine Stringvariable anzugeben. **Return:** Es wird der Name einer Integer-Variablen erwartet, in die die Höhe geschrieben wird. **Syntax:** `<function name="STRING.PIXELHEIGHT"` `return="<return variable>"><control name>,` `<varaible or text></function>` **Beispiel:** Berechnung der Breite bezogen auf den Font einer Form `<let name="pixelsh" />` `...` `<function name="STRING.PIXELHEIGHT"` `return="pixelsh">_T"", cedit1</function>` Berechnung der Breite bezogen auf den Font eines Controls `<function name="STRING. PIXELHEIGHT"` `return="pixelsh"> cedit1, cedit1</function>`

Funktionsname	Bedeutung
String pixel width	Berechnung der Breite einer Zeichenkette in Pixel. Die Berechnung wird unter Verwendung des aktuellen Schriftfonts des angegebenen Controls oder der Form durchgeführt. Der Name des Controls ist als String zu übergeben. Wird ein Leerstring angegeben, bezieht sich die Berechnung auf die Form. **Parameter:** **control name** **Zeichenkette** Die Zeichenkette kann direkt als Text angegeben werden oder es ist eine Stringvariable anzugeben. **Return:** Es wird der Name einer Integer-Variablen erwartet, in die die Textbreite geschrieben wird. **Syntax:** `<function name="STRING.PIXELWIDTH"` `return="<return variable>"><control name>,` `<varaible or text></function>` **Beispiel:** Berechnung der Breite bezogen auf den Font einer Form `<let name="pixels" />` `...` `<function name="STRING. PIXELWIDTH"` `return="pixels">_T"", cedit1</function>` Berechnung der Breite bezogen auf den Font eines Controls `<function name="STRING.PIXELWIDTH"` `return="pixels"> cedit1, cedit1</function>`
String SetAt	Die Funktion schreibt ein Zeichen an die spezifizierte Position. Der Index muss kleiner als die maximale Textlänge sein. **Parameter:** **str** - String **char** - Zeichen, das an die spezifizierte Position geschrieben werden soll **index** - Null basierender Index auf die Zeichenkette der Zielvariable **Syntax:** `<function name="string.setat" ><destination string>,` `<character string>, <index></function>` **Beispiel:** `<let name="str" type="string" >br_wn</string>` `<function name="string.setat">str, ">_T"o", 2` `</function>`

Funktionsname	Bedeutung
String Split	Die Funktion zerlegt einen String in eine Anzahl von Teil-Strings und kopiert diese in das angegebene String-Array. Die Trennung erfolgt an dem angegebenen Trennzeichen. Das Trennzeichen wird nicht im Teilstring gespeichert. Die Funktion expandiert automatisch das Array, wenn die gefundene Anzahl von Trennzeichen größer ist, als die festgelegte Array-Größe. **Parameter:** **str** - String **char** - Name auf eine Variable, die das Trennzeichen enthält **number** - Name auf eine Variable, die nach der Funktionsausführung die Anzahl erzeugter Teil-Strings enthält. **Rückgabewert:** Es ist der Name eines String-Arrays anzugeben, das nach der Funktionsausführung die Teilstrings enthält. **Syntax:** `<function name=" string.split"` `return="<result string array>"><string>, <char>,` `<number></function>` **Beispiel:** `<let name="strlist" type="string" dim="2"/>` `<let name="str_num" />` `<function name="string.split" return="strlist">` `_T"brown;green;blue;red", _T";", str_num </function>`
String trim left	Die Funktion löscht führende Leerzeichen aus einem String. **Parameter:** **str1** - Stringvariable **Syntax:** `<function name="string.trimleft" > str1 </function>` **Beispiel:** `<let name="str1" type="string">test trim left</let>` `<function name="string.trimleft" > str1 </function>` **Ergebnis:** str1 = "test trim left"

Funktionsname	Bedeutung
String trim right	Die Funktion löscht nachfolgende Leerzeichen aus einem String. **Parameter:** **str1** - Stringvariable **Syntax:** `<function name="string.trimright" > str1 </function>` **Beispiel:** `<let name="str1" type="string"> test trim right </let>` `<function name="string.trimright" > str1` `</function>` **Ergebnis:** str1 = "test trim right"
Sinus	Die Funktion berechnet den Sinus des übergebenen Wertes in Grad. **Parameter:** **double** - Winkel **Syntax:** `<function name="sin" return="<double val>"> double </function>` **Beispiel:** `<let name= "sin_val" type="double"></let>` `<function name="sin" return="sin_val"> 20.0` `</function>`
Cosinus	Die Funktion berechnet den Kosinus des übergebenen Wertes in Grad. **Parameter:** **double** - Winkel **Syntax:** `<function name="cos" return="<double val>"> double </function>` **Beispiel:** `<let name= "cos_val" type="double"></let>` `<function name="cos" return="cos_val"> 20.0` `</function>`

Funktionsname	Bedeutung
Tangens	Die Funktion berechnet den Tangens des übergebenen Wertes in Grad. **Parameter:** **double** - Winkel **Syntax:** `<function name="tan" return="<double val>"> double </function>` **Beispiel:** `<let name= "tan_val" type="double"></let>` `<function name="tan" return="tan_val"> 20.0` `</function>`
ARCSIN	Die Funktion berechnet den Arcussinus des im Gradmaß übergebenen Wertes. **Parameter:** **double** - x im Bereich von -PI/2 bis +PI/2 **Syntax:** `<function name="arcsin" return="<double val>"> double </function>` **Beispiel:** `<let name= "arcsin_val" type="double"></let>` `<function name="arcsin" return="arcsin_val"> 20.0` `</function>`
ARCOS	Die Funktion berechnet den Arcuskosinus des im Gradmaß übergebenen Wertes. **Parameter:** **double -** x im Bereich von -PI/2 bis +PI/2 **Syntax:** `<function name="arcos" return="<double val>"> double </function>` **Beispiel:** `<let name= "arccos_val" type="double"></let>` `<function name="arccos" return="arccos_val"> 20.0` `</function>`
ARCTAN	Die Funktion berechnet den Arcustangens des im Gradmaß übergebenen Wertes. **Parameter:** **double** - Arcustangens von y/x **Syntax:** `<function name="arctan" return="<double val>"> double </function>` **Beispiel:** `<let name= "arctan_val" type="double"></let>` `<function name="arctan" return="arctan_val"> 20.0` `</function>`

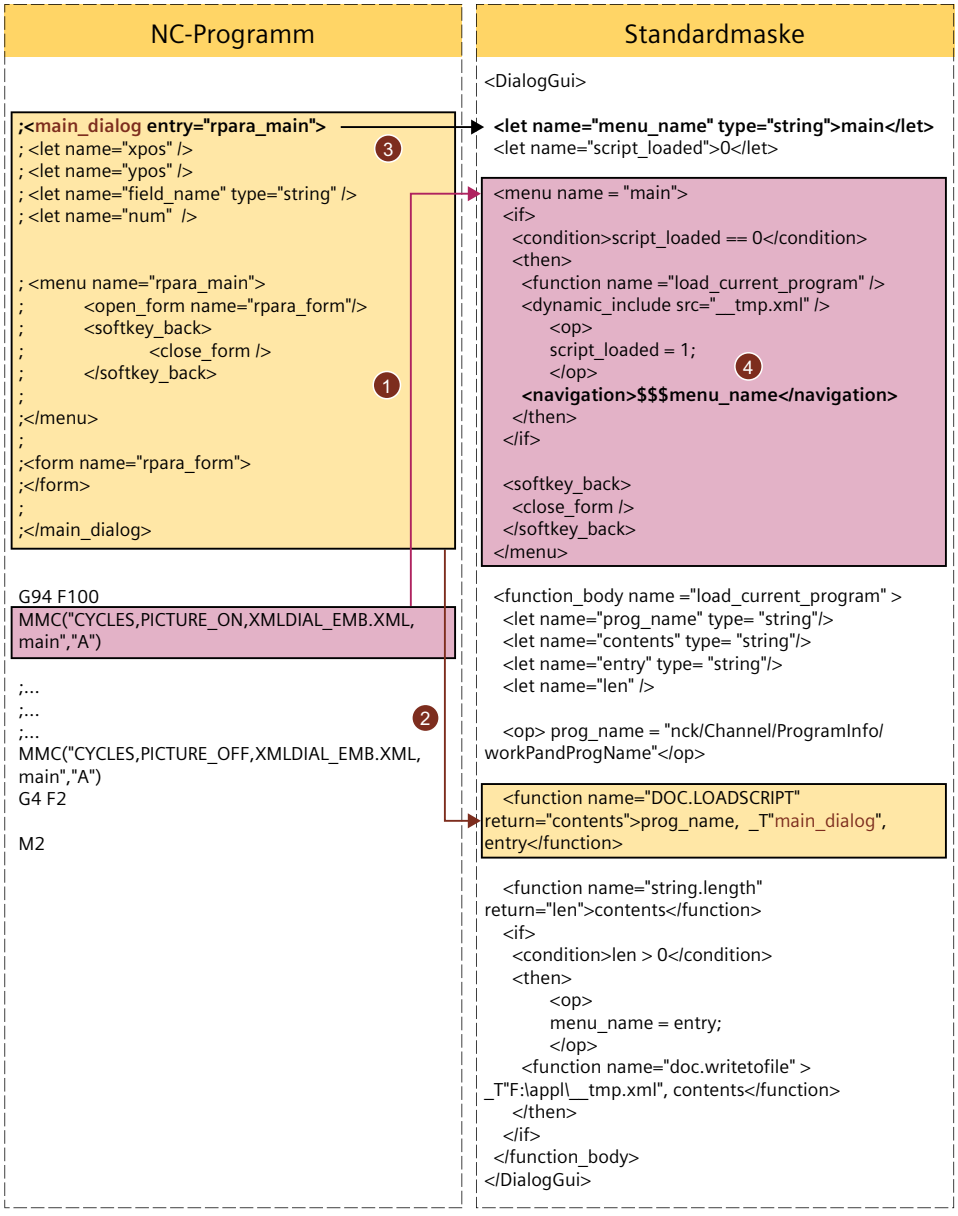
Funktionsname	Bedeutung
Dateiverarbeitung	
Lesen einer Datei	Die Funktion liest den Inhalt der angegebenen Datei in eine Stringvariable. Optional kann die Anzahl zu lesender Zeichen als zweiter Parameter angegeben werden. **Attribut:** **name -** Die Angabe des Dateinamens ist in Kleinschreibung vorzunehmen. Der Zugriff auf Dateien in anderen Verzeichnissen erfolgt über eine relative Pfadangabe, die als Ausgangspunkt das Verzeichnis appl oder dvm verwendet. **return** - Name der lokalen Variablen **Parameter:** **progname** - Dateiname **number of characters** - Anzahl zu lesender Zeichen in Byte (optional) **Syntax:** `<function name="doc.readfromfile"` `return="<string var>"> progname, number of characters` `</function>` **Beispiel:** `<let name = "my_var" type="string" ></let>` NC-Dateisystem `<function name="doc.readfromfile"` `return="my_var"> _T"n:\mpf\test.mpf"` `</function>` CF-Karte `<function name="doc.readfromfile"` `return="my_var"> _T"f:\appl\test.mpf"` `</function>` oder `<function name="doc.readfromfile"` `return="my_var"> _T".\test.mpf"` `</function>`

Funktionsname	Bedeutung
Schreiben einer Datei	Die Funktion schreibt den Inhalt einer Stringvariable in die angegebene Datei. Die Angabe des Dateinamen ist in Kleinschreibung vorzunehmen. Der Zugriff auf Dateien in anderen Verzeichnissen erfolgt über eine relative Pfadangabe, die als Ausgangspunkt das Verzeichnis appl oder dvm verwendet. **Parameter:** **progname** - Dateiname **str1** - String **Syntax:** `<function name="doc.writetofile" > progname, str1 </function>` **Beispiel:** `<let name = "my_var" type="string" > file content </let>` NC-Dateisystem `<function name="doc.writetofile">_T"n:\mpf\test.mpf", my_var` `</function>` CF-Karte `<function name="doc.writetofile">_T"f:\appl\test.mpf", my_var` `</function>` oder `<function name="doc.writetofile">_T".\test.mpf", my_var </function>`

Funktionsname	Bedeutung
Löschen einer Datei	Die Funktion löscht die angegebene Datei aus dem Verzeichnis. Die Angabe des Dateinamen ist in Kleinschreibung vorzunehmen. Der Zugriff auf Dateien in anderen Verzeichnissen erfolgt über eine relative Pfadangabe, die als Ausgangspunkt das Verzeichnis appl oder dvm verwendet. **Parameter:** **progname** - Dateiname **Syntax:** `<function name="doc.remove" > progname </function>` **Beispiel:** NC-Dateisystem `<function name="doc.remove">_T"n:\mpf\test.mpf"` `</function>` CF-Karte `<function name="doc.remove">_T"f:\appl\test.mpf"` `</function>` oder `<function name="doc.remove">_T".\test.mpf"` `</function>`

Funktionsname	Bedeutung
Extrahieren von Skriptanteilen	Die Funktion kopiert eine in einem Teileprogramm eingebettete Dialogbeschreibung in die angegebene lokale Variable. Als Aufrufparameter sind der Programmname, der Dialogname und eine Variable zum Ablegen des Hauptmenünamens anzugeben. Wurde der Name der Dialogbeschreibung im Teileprogramm gefunden, enthält die Return-Variable diese Beschreibung. Wenn der Inhalt der Variable in eine Datei gespeichert wird, lässt sich das Skript durch einen indirekten Aufruf ausführen. Das System stellt ein Skript bereit, das die Dialogbeschreibung aus dem aktiven Teileprogramm extrahiert und den Dialog aktiviert. Dieses Skript kann in einem MMC-Befehl aufgerufen werden, um die zum Teileprogramm gehörende Maske zu aktivieren. **Syntax:** `<function name="doc.loadscript"` `return="<name of script variable>">progname, _T"dialog part name",` `main menu </function>` **Attribut**: **return** - Variable, in die das extrahierte Skript abgelegt wird **Parameter:** **progname** - vollständiger Pfad, auf das Programm (Der Pfadname kann in DOS-Notation der Funktion übergeben werden.) **main menu** - der gefundene Menüname wird in diese Variable kopiert **dialog part name** - Tag-Name, in dem die Dialogbeschreibung eingebettet ist **Beispiel:** `<function name="doc.loadscript" return="contents">prog_name,` `_T"main_dialog", entry</function>`

Funktionsname	Bedeutung
Extrahieren von Skriptan-teilen Fortsetzung	**Beispieldiagramm:**

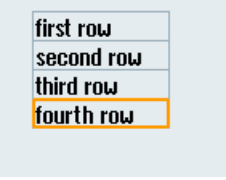


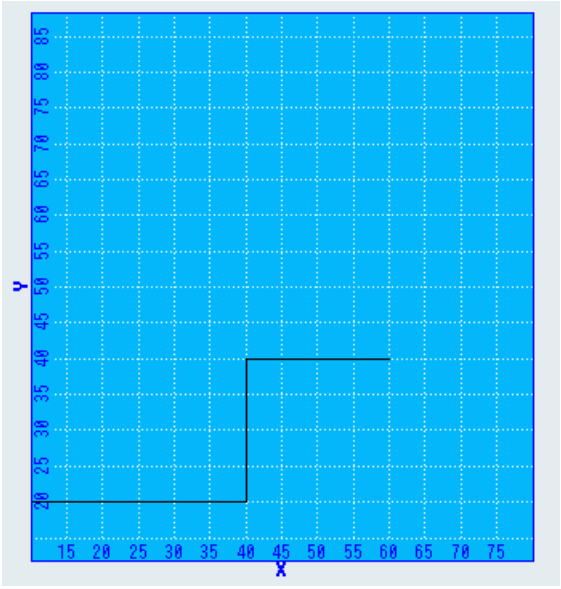
Funktionsname	Bedeutung
Exist	Existiert die Datei, liefert die Funktion den Wert 1. Die Angabe des Dateinamen ist in Kleinschreibung vorzunehmen. Der Zugriff auf Dateien in anderen Verzeichnissen erfolgt über eine relative Pfadangabe, die als Ausgangspunkt das Verzeichnis appl oder dvm verwendet. **Parameter:** **progname** - Dateiname **Syntax:** `<function name="doc.exist" return="<int_var>" > progname </function>` **Beispiel:** `<let name ="exist">0</let>` NC-Dateisystem `<function name="doc.exist" return="exist">_T"n:\mpf\test.mpf" </function>` CF-Karte `<function name="doc.exist" return="exist">_T"f:\appl\test.mpf" </function>` oder `<function name="doc.exist" return="exist">_T".\test.mpf" </function>`
NC Programmanwahl	Die Funktion wählt das angegebene Programm zur Abarbeitung an. Das Programm muss im NC Dateisystem abgelegt sein. **Parameter:** **progname** - Dateiname **Syntax:** `<function name="ncfunc.select"> progname </function>` **Beispiel:** NC-Dateisystem `<function name="ncfunc.select"> _T"n:\mpf\test.mpf" </function>`

Funktionsname	Bedeutung
Setzen eines einzelnen Bits	Die Funktion dient zur Manipulation einzelner Bits der angegebenen Variablen. Die Bits können gesetzt oder rückgesetzt werden. **Syntax:** `<function name="ncfunc.bitset" refvar="address" value="set/reset" > bit0, bit1, … bit9 </function>` **Attribute:** **refvar** - gibt den Namen der Variablen an, in welche die Bitkombination geschrieben werden soll **value** – Bitwert Wertebereich 0 und 1 **Werte:** Als Funktionswerte sind die Bitnummern mit 0 beginnend zu übergeben. Maximal können 10 Bits pro Aufruf modifiziert werden. **Beispiel:** `<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="1" > 0, 2, 3, 7 </function>` `<function name="ncfunc.bitset" refvar="nck/Channel/Parameter/R[1]" value="0" > 1, 4 </function>`
Control löschen	Die Funktion löscht das angegebene Control. **Syntax:** `<function name="control.delete"> control name </function>` **Attribut:** **name** – Funktionsname **Wert:** **control name** – Name des Controls **Beispiel:** `<function name="control.delete"> _T"my_editfield" </function>`

Funktionsname	Bedeutung
Add Item	Die Funktion fügt ein neues Element am Ende der Liste ein. Hinweis: Die Funktion steht nur für die Control-Typen "listbox" und "graphicbox" zur Verfügung. **Syntax:** `<function name="control.additem"> control name, item </function>` **Attribut:** **name** – Funktionsname **Werte:** **control name** – Control name **item** - einzufügender Ausdruck **itemdata** - ganzzahliger Wert; durch den Anwender festgelegt **Beispiel:** `<let name ="itemdata">1</let>` … … … `<op> item_string = _T"text1" </op>` `<function name="control.additem">_T"listbox1", item_string, itemdata </function>`

Funktionsname	Bedeutung
Insert Item	Die Funktion fügt ein neues Element an der angegebenen Position ein. Hinweis: Die Funktion steht nur für den Control-Typ "listbox" zur Verfügung. **Syntax:** `<function name="control.insertitem"> control name, index, item, itemdata </function>` **Attribut:** **name** – Funktionsname **Werte:** **control name** - Control name **index** - Position mit Null beginnend **item** - einzufügender Ausdruck **itemdata** - ganzzahliger Wert; durch den Anwender festgelegt **Beispiel:** `<let name ="itemdata">1</let>` … … … `<op> item_string = _T"text2" </op>` `<function name="control.insertitem">_T"listbox1", 1, item_string, itemdata </function>`
Delete Item	Die Funktion löscht ein Element an der angegebenen Position. Hinweis: Die Funktion steht nur für den Control-Typ "listbox" zur Verfügung. **Syntax:** `<function name="control.deleteitem"> control name, index </function>` **Attribut:** **name** - Funktionsname **Werte:** **control name** - Control name **index**- Index mit 0 beginnend **Beispiel:** `<function name="control.deleteitem">_T"listbox1", 1</function>`

Funktionsname	Bedeutung
Load Item	Die Funktion fügt eine Liste von Ausdrücken in das Control ein. Die Funktion steht nur für die Control-Typen "listbox" und "graphicbox" zur Verfügung. **Syntax:** `<function name="control.loaditem"> control name, list </function>` **Attribut:** **name** – Funktionsname **Werte:** **control name** – Control name **list**- Stringvariable Die in der Variablen enthaltene Zeichenkette muss dem unten beschriebenen Aufbau entsprechen. **Aufbau der Liste:** Die Liste beinhaltet eine Anzahl von Ausdrücken, die jeweils mit einem \\n voneinander zu trennen sind. **Beispiel:** In dem Beispiel wird der Listbox der Inhalt über die Funktion control.loaditem zugewiesen. Das Steuerzeichen \\n trennt die zur Zeile gehörenden Ausdrücke voneinander. `<CONTROL name = "list" xpos = "160" ypos = "32"` `width ="100" height="200" fieldtype="listbox"/>` `<let name="lb_list" type="string" />` `<op>` `lb_list = _T"first row\\n";` `lb_list = lb_list + _T"second row\\n";` `lb_list = lb_list + _T"third row\\n";` `lb_list = lb_list +_T"fourth row\\n";` `</op>` `<function name="control.loaditem">_T"list",` `lb_list</function>` 

Funktionsname	Bedeutung
Load Item Fortsetzung	**Beispiel:** In dem Beispiel wird der Graphicbox der Inhalt über die Funktion control.loaditem zugewiesen. Das Steuerzeichen \\n trennt die grafischen Elemente voneinander. `<control name= "c_gbox" xpos = "6" ypos="32" width="328" height="356" fieldtype="graphicbox" /> <let name="plot_list" type="string" /> <op> plot_list = _T"l;10;20;40;20\\n"; plot_list = plot_list + _T"l;40;20;40;40\\n"; plot_list = plot_list + _T"l;40;40;60;40\\n"; plot_list = plot_list +_T"l;80;0;80;100\\n"; </op> <function name="control.loaditem">_T"c_gbox", plot_list</function>` 
Empty	Die Funktion löscht den Inhalt des angegebenen Listbox- oder Graphicbox-Controls. **Syntax:** `<function name="control.empty"> control name, </function>` **Attribut:** **name** – Funktionsname **Werte:** **control name** – Control name **Beispiel:** `<function name="control.empty">_T"listbox1" </function>`

Funktionsname	Bedeutung
Get focus	Die Funktion liefert den Namen des Controls, das den Eingabefocus besitzt. **Syntax:** `<function name="control.getfocus" return="focus_name" />` **Attribute:** **name** – Funktionsname **return** – Es ist eine String-Variable anzugeben, in die der Name des Controls kopiert wird. **Beispiel:** `<let name>="focus_field" type="string"></let>` `<function name="control.getfocus" return="focus_field"/>`
Set focus	Die Funktion setzt den Eingabefocus auf das angegebene Control. Der Controlname ist als Textausdruck der Funktion zu übergeben. **Syntax:** `<function name="control.setfocus"> control name` `</function>` **Attribut:** **name** - Funktionsname **Wert:** **control name** - Name des Controls **Beispiel:** `<function name="control.setfocus" ">_T"listbox1"` `</function>`
Get cursor selection	Die Funktion liefert bei einer Listbox den Cursorindex. Der Controlname ist als Textausdruck der Funktion zu übergeben. **Syntax:** `<function name="control.getcurssel" return="var">control name` `</function>` **Beispiel:** `<let name>="index"></let>` `<function name="control.getcurssel" return="index">_T"listbox1"` `</function>`

Funktionsname	Bedeutung
Set cursor selection	Die Funktion setzt bei einer Listbox den Cursor auf die entsprechende Zeile. Der Controlname ist als Textausdruck der Funktion zu übergeben. **Syntax:** `<function name="control.setcurssel" > control name, index </function>` **Beispiel:** `<let name>="index">2</let>` `<function name="control.setcurssel" ">_T"listbox1",index </function>`
Get Item	Die Funktion kopiert bei einer Listbox den Inhalt der ausgewählten Zeile in die angegebene Variable. Als Referenzvariable ist eine Stringvariable anzugeben. Der Controlname ist als Textausdruck der Funktion zu übergeben. **Syntax:** `<function name="control.getitem" return="var"> control name, index </function>` **Beispiel:** `<let name>="index">2</let>` `<let name>="item" type="string"></let>` `<function name="control.getitem" return="item" ">_T"listbox1",index </function>`
Get Item Data	Die Funktion kopiert bei einer Listbox den anwenderspezifisch vergebenen Wert eines Elementes in die angegebene Variable. Bei einem Edit-Control kopiert die Funktion den anwenderspezifisch vergebenen Wert (item_data) in die angegebene Variable. Als Referenzvariable ist eine Integer-Variable anzugeben. Der Controlname ist als Textausdruck der Funktion zu übergeben. **Syntax:** `<function name="control.getitemdata" return="var"> control name, index </function>` **Beispiel:** `<let name>="index">2</let>` `<let name>="itemdata"></let>` `<function name="control.getitemdata" return="itemdata" ">_T"listbox1",index</function>`

Funktionsname	Bedeutung
Image Box Set	Diese Funktion dient zum Steuern des sichtbaren Bereichs der Controls **Imagebox** und **Graphicbox**. Als Aufrufparameter sind der Control-Name, das Steuerkommando und die zugehörigen Werte anzugeben. **Syntax:** `<function name="control.imageboxset">control name, command, command parameter</function>` **Aufrufparameter:** • **x_offs** Die Funktion verschiebt den Bildausschnitt auf die angegebene X-Position. Als weiterer Parameter ist die Koordinate der linken Ecke anzugeben. • **y_offs** Die Funktion verschiebt den Bildausschnitt auf die angegebene Y-Position. Als weiterer Parameter ist die Koordinate der oberen Ecke anzugeben. • **xy_offs** Die Funktion verschiebt den Bildausschnitt auf die angegebenen X/Y-Positionen. Als weiterer Parameter sind die Koordinaten linke und obere Ecke anzugeben. • **followCursor** Der Bildausschnitt folgt der gesetzten Cursorposition. • **zoomplus** Das Bild wird um einen Faktor von 0,12 vergrößert. • **zoomminus** Das Bild wird um einen Faktor von 0.12 verkleinert. • **autozoom** Das Bild wird automatisch auf den Anzeigebereich skaliert. • **Update** Das Control wird erneut gezeichnet. • **SetBkColor** Das Kommando setzt die spezifizierte Hintergrundfarbe. Als weiterer Parameter ist die Farbkodierung anzugeben. • **SetCursorRect** Der als Rechteck angegebene Ausschnitt wird in den sichtbaren Bereich verschoben. • **SetAnimationState** (gilt nur für das Control Imagebox) – **start** - Das Kommando startet eine Animation. – **stop** - Das Kommando stoppt eine Animation.

Funktionsname	Bedeutung
Image Box Set Fortsetzung	**Beispiel:** `...` `...` `<softkey POSITION="1">` `<caption>zoom%nin</caption>` `<function name="control.imageboxset">_T"c_gbox",` `_T"zoomplus"</function>` `</softkey>` `...` `...` `<form name = "main_form">` `<init>` `<caption> graphicbox</caption>` `<control name= "c_gbox" xpos = "6" ypos="32"` `width="328" height="356" fieldtype="graphicbox" />` `<let name="plot_list" type="string" />` `<op>` `plot_list = _T"1;10;20;40;20\\n";` `plot_list = plot_list + _T"1;40;20;40;40\\n";` `plot_list = plot_list + _T"1;40;40;60;40\\n";` `plot_list = plot_list +_T"1;80;0;80;100\\n";` `</op>` `<function name="control.loaditem">_T"c_gbox",` `plot_list</function>` `</init>` `</form>`
Image Box Get	Diese Funktion dient zum Abfragen der Control-Eigenschaften Imagebox. Als Aufrufparameter sind das Steuerkommando und die zugehörigen Werte anzugeben. **Syntax:** `<function name="control.imageboxget">control name, command,` `command parameter</function>` **Aufrufparameter:** **GetViewSize** Liefert die Größe des Anzeigebereiches Als weiterer Parameter ist eine Variable vom Strukturtyp SIZE anzugeben. **Beispiel:** `<let name="view_size" type="size" />` `<function name="control.imageboxget">_T"topoview", _T"`**`GetViewSize`**`",` `view_size</function>`

Funktionsname	Bedeutung
Bitmap Dim	Die Funktion kopiert die Dimension einer Bitmap in eine Variable vom Strukturtyp SIZE zurück. Zur Typdefinition ist die Datei struct_def.xml in das Projekt zu integrieren. **Weitere Informationen** unter Kapitel "Datei struct_def.xml erstellen (Seite 141)" **Syntax:** `<function name="bitmap.dim" >name, variable type` `</function>` **Parameter:** **name** - Dateipfad **variable type** - Variablenname einer Variable vom Typ SIZE **Beispiel:** `<let name="bmp_size" type="size" />` `<function name="bitmap.dim" >_T"test.bmp", bmp_size` `</function>`
Screen Resolution	Die Funktion kopiert die absolute Bildschirmauflösung des Systems in eine Variable vom Strukturtyp SIZE zurück. Zur Typdefinition ist die Datei struct_def.xml in das Projekt zu integrieren. **Weitere Informationen** unter Kapitel "Datei struct_def.xml erstellen (Seite 141)" **Syntax:** `<function name="hmi.screen_resolution" >varialble type </function>`
Get HMI Resolution	Die Funktion kopiert die von SINUMERIK Operate verwendete Bildschirmauflösung in eine Variable vom Strukturtyp SIZE zurück. Zur Typdefinition ist die Datei struct_def.xml in das Projekt zu integrieren. **Weitere Informationen** unter Kapitel "Datei struct_def.xml erstellen (Seite 141)" **Syntax:** `<function name="hmi.get_hmi_resolution" >variable type </function>`
Get Caption Height	Die Funktion liefert die Titelzeilenhöhe in Pixel. **Syntax:** `<function name="hmi.get_caption_height" return="<return var>" />` **Attribute:** **return** - Integer Variable

Funktionsname	Bedeutung
Get Font Families	Die Funktion liefert eine Liste der installierten Fonts. Die Font-Namen sind durch ein LF-Zeichen voneinander getrennt aufgelistet. Als Return-Wert ist der Name einer String-Variablen zu übergeben. **Syntax:** `<function name="HMI.GET_FONT_FAMILIES"` `return="< String-Variable>" />` **Beispiel:** `<init>` `...` `<let name="families" type="string" />` `<function name="HMI.GET_FONT_FAMILIES" return="families" />` `<control name="lb" xpos="8" ypos="40" width="260" height="140"` `fieldtype="listbox">` `</control>` `<function name="control.loaditem">_T"lb", families</function>` `...` `<update_controls type="true" />` `</init>`
Abs	Die Funktion liefert den Absolutwert der angegebenen Zahl. **Syntax:** `<function name="abs" return="var"> value` `</function>`
SDEG	Die Funktion rechnet den angegebenen Wert in Grad um. **Syntax:** `<function name="sdeg" return="var"> value` `</function>`
SRAD	Die Funktion rechnet den angegebenen Wert in RADian um. **Syntax:** `<function name="srad" return="var"> value` `</function>`
SQRT	Die Funktion berechnet die Wurzel des angegebenen Wertes. **Syntax:** `<function name="sqrt" return="var"> value` `</function>`
ROUND	Die Funktion rundet die übergebene Zahl auf die spezifizierte Anzahl von Nachkommastellen. Wird keine Anzahl von Nachkommastellen vorgegeben, dann rundet die Funktion unter Einbeziehung der ersten Nachkommastelle. **Syntax:** `<function name="round" return="var"> value, nDecimalPlaces` `</function>`

Funktionsname	Bedeutung
FLOOR	Die Funktion liefert den größtmöglichen ganzzahligen Wert, der kleiner oder gleich dem übergebenen Wert ist. **Syntax:** `<function name="floor" return="var"> value </function>`
CEIL	Die Funktion liefert den kleinstmöglichen ganzzahligen Wert, der größer oder gleich dem übergebenen Wert ist. **Syntax:** `<function name="ceil" return="var"> value </function>`
LOG	Die Funktion berechnet den Logarithmus des angegebenen Wertes. **Syntax:** `<function name="log" return="var"> value </function>`
LOG10	Die Funktion berechnet den dekadischen Logarithmus des angegebenen Wertes. **Syntax:** `<function name="log10" return="var"> value </function>`
POW	Die Funktion berechnet den Wert "a^b". **Syntax:** `<function name="pow" return="var"> a, b </function>`
MIN	Die Funktion vergleicht die übergebenen Werte und gibt den kleineren Wert zurück. **Syntax:** `<function name="min" return="var"> value1, value2 </function>`
MAX	Die Funktion vergleicht die übergebenen Werte und gibt den größeren Wert zurück. **Syntax:** `<function name="max" return="var"> value1, value2 </function>`
RANDOM	Die Funktion liefert eine Pseudo-Zufallszahl. **Syntax:** `<function name="random" return="var" </function>`

Funktionsname	Bedeutung
MMC	**Funktion**: Über den Befehl MMC können aus dem Teileprogramm in SINUMERIK Operate anwenderdefinierte Dialogmasken angezeigt werden. Das Aussehen der Dialogmasken wird durch rein textuelle Projektierung festgelegt (XML-Datei im Herstellerverzeichnis), die System-Software bleibt dabei unverändert. Durch Änderungen im Operate-Base-System sind die Parameter wie folgt: XML → CYCLES oder POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF **Syntax:** MMC ("Bedienbereich, Befehl, XML-Skriptdatei, Menüname, reserviert, reserviert, Anzeigezeit oder Quittungsvariable, reserviert", "Quittungsmodus") **Bedeutung:** • MMC Aus dem Teileprogramm interaktiv Dialogmaske in SINUMERIK Operate aufrufen. • CYCLES oder POPUPDLG Kennzeichnung für Anwenderdialoge, die aus einem Teileprogramm gestartet werden sollen. • PICTURE_ON bzw. PICTURE_OFF Befehl: Dialoganwahl bzw. Dialogabwahl • XML-Datei Name der XML-Skriptdatei • Menü Name des Menü-Tags, das den anzuzeigenden Dialog verwaltet • reserviert reserviert für SINUMERIK Operate • reserviert reserviert für SINUMERIK Operate • Zeit Anzeigezeit des Dialogsbei Quittungsmodus "N" • reserviert reserviert für SINUMERIK Operate • Quittungsmodus "S" synchron, Quittung über den Softkey "OK" "N" asynchron, Dialog wird nach der vereinbarten Zeit geschlossen

Funktionsname	Bedeutung
MMC Fortsetzung	**Beispiel synchroner Aufruf:** Durch Änderungen im Operate-Base-System sind die Parameter wie folgt: XML → CYCLES oder POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF NC-Anweisung `MMC("POPUPDLG or CYCLES",PICTURE_ON,mmc_cmd.xml,cmd1,,,,,","S")` `Datei: mmc_cmd.xml` `<menu name = "cmd1">` `<open_form name = "cmd1_form" />` `<softkey_ok>` `<close_form />` `</softkey_ok>` `</menu>` `<form name = "cmd1_form" xpos ="12" ypos="100" width="500" height="240">` `<init>` `</init>` `<paint>` `</paint>` `</form>` **Beispiel asynchroner Aufruf (keine Quittung erwartet):** Durch Änderungen im Operate-Base-System sind die Parameter wie folgt: XML → CYCLES oder POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF NC-Anweisung `MMC("POPUPDLG or CYCLES",PICTURE_ON,mmc_cmd.xml,cmd1,,,10,,","N")` `Datei: mmc_cmd.xml <menu name = "cmd1">` `<open_form name = "cmd1_form" />  <softkey_ok>  <close_form />` `</softkey_ok>` `</menu>` `<form name = "cmd1_form" xpos ="12" ypos="100" width="500" height="240">` `<init>` `</init>` `<paint>` `</paint>` `</form>`

Funktionsname	Bedeutung
MMC Fortsetzung	**Beispiel extrahieren von Skriptanteilen aus einem Teileprogramm:** Durch Änderungen im Operate-Base-System sind die Parameter wie folgt: XML → CYCLES oder POPUPDLG XML_ON → PICTURE_ON XML_OFF → PICTURE_OFF NC-Anweisung `MMC("POPUPDLG, PICTURE_ON, XMLDIAL_EMB.XML,main","S")` `Datei: xmldial_emb.xml` `<DialogGui>` ` <let name="menu_name" type="string">main</let>` ` <let name="script_loaded">0</let>` ` <menu name = "main">` ` <if>` ` <condition>script_loaded == 0</condition>` `  <then>` ` <function name ="load_current_program" />` ` <dynamic_include src="__tmp.xml" />` `  <op>` `  script_loaded = 1;` `  </op>` ` <navigation>$$$menu_name</navigation>` ` </then>` ` </if>` ` <softkey_back>` ` <close_form />` ` </softkey_back>` ` </menu>` ` <function_body name ="load_current_program" >` ` <let name="prog_name" type= "string"/>` ` <let name="contents" type= "string"/>` ` <let name="entry" type= "string"/>` ` <let name="len" />` ` <op> prog_name = "nck/Channel/ProgramInfo/workPandProgName"</op>` ` <function name="DOC.LOADSCRIPT"` `return="contents">prog_name, _T"main_dialog", entry` `</function>` ` <function name="string.length" return="len">contents</function>` ` <if>` ` <condition>len > 0</condition>` ` <then>` `  <op>` `  menu_name = entry;` `  </op>` ` <function name="doc.writetofile" >_T"__tmp.xml", contents` `</function>` ` </then>` ` </if>` ` </function_body>` `</DialogGui>`

Funktionsname	Bedeutung
MMC Fortsetzung	Programmbeispiel

```
;<main_dialog entry="rpara_main">
; <let name="xpos" />
; <let name="ypos" />
; <let name="field_name" type="string" />
; <let name="num" />
; <menu name="rpara_main">
; <open_form name="rpara_form"/>
; <softkey_back>
; <close_form />
; </softkey_back>
;</menu>
;
;<form name="rpara_form">
; <init>
; <caption>test mask</caption>
; <let name="count" >0</let>
; <op>
; xpos = 120;
; ypos = 34;
;
; "nck/Channel/Parameter/R[10]" = 10;
; </op>
; <!-- load the number of controls -->
; <op>
; num = "nck/Channel/Parameter/R[10]";
; </op>
;
; <while>
; <condition>count < num</condition>
; <print name="field_name" text="edit%d">count</print>
;
; <control name = "$field_name" xpos = "$xpos" ypos = "$ypos"
refvar="nck/Channel/Parameter/R[$count]" hotlink="true" />
; <op>
; ypos = ypos +24;
; count = count +1;
; </op>
; </while>
;
; </init>
;
; <paint>
; <op>
; xpos = 8;
; ypos = 36;
; count = 0;
; </op>
; <while>
; <condition>count < num</condition>
; <print name="field_name" text="R-Parameter%d">count </print>
;
; <text xpos = "$xpos" ypos = "$ypos" >$$$field_name</text>
; <op>
```

Funktionsname	Bedeutung
	; ypos = ypos +24; ; count = count +1; ; </op> ; </while> ; ; </paint> ; ;</form> ; ;</main_dialog> … G94 F100 MMC("POPUPDLG, PICTURE_ON, XMLDIAL_EMB.XML,main","A") … … … MMC("POPUPDLG, PICTURE_OFF, XMLDIAL_EMB.XML,main","A") G4 F2 M2
ISNUMERIC	Die Funktion überprüft den Inhalt eines Strings, ob dieser als Zahl ausgewertet werden kann. Leerzeichen und Tabulatoren werden ignoriert. Als weiterhin gültige Zeichen werden das Vorzeichen und der Dezimalpunkt betrachtet. Die Funktion liefert den Wert 1 zurück, wenn die Bedingungen erfüllt sind. **Parameter:** **string** - Zeichenkette **Syntax:** <function name="isnumeric" return="result"><string> </function>
ROOT	Die Funktion zieht die n-te Wurzel aus einer Zahl. **Parameter:** **value** - Zahl **n** - Wurzelexponent **Syntax:** <function name="ROOT" return="result"><value>, <n></function>

3.12 Multitouch-Bedienung

3.12.1 Multitouch-Funktion

Übersicht

Mit dem Einführen von Multitouch-Displays ergibt sich die Notwendigkeit, die Bedienung an die erweiterte Funktionalität der Displays anzupassen. Die starre Positionierung von z. B. Softkeys wird aufgehoben und durch eine freie Positionierung von Buttons ersetzt bzw. um diese erweitert. Durch die Vielfalt der Gestaltungsmöglichkeiten von Buttons ist es nicht mehr sinnvoll, für die Programmierung nur ein Control anzuwenden, dessen Aussehen sich ausschließlich an den Design-Vorgaben der Bedien-Software ausrichtet. Toggle-Controls, die nur zwei Zustände kennen, können z. B. durch Schalter ersetzt werden, die den jeweiligen Zustand durch eine Ikone darstellen und auf die Tap- oder Flick-Geste reagieren.

Angezeigte Grafiken können in die Lage versetzt werden, auf die Pan-Geste zu reagieren. Hierfür wird das Control **imagebox** entsprechend erweitert.

Hinweis

Grafiken, die im Paint-Tag ausgegeben werden, reagieren nicht auf Gesten.

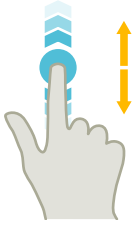
Außerhalb der vom Parser bereitgestellten Controls können Fingergesten im Skript verarbeitet werden, um unter anderem die Möglichkeit zu bieten, neue Strategien zum Navigieren in den Dialogen anzubieten. Die Fingergesten könnten zum Vergrößern/Verkleinern der Dialoginhalte verwendet werden.

Der Easy XML- Parser unterstützt die nachfolgenden Fingergesten:

Fingergesten

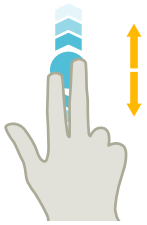
Antippen (Tap)

- Fenster auswählen
- Objekt auswählen (z. B. NC-Satz)
- Eingabefeld aktivieren
  - Wert eingeben bzw. überschreiben
  - Erneut tippen zum Ändern des Werts

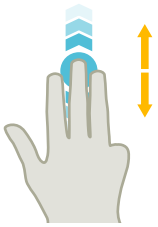


Vertikal Wischen mit 1 Finger (Flick)

- Scrollen in Listen (z. B. Programme, Werkzeuge, Nullpunkte)
- Scrollen in Dateien (z. B. NC-Programm)



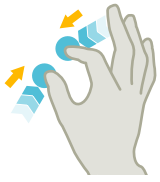
Vertikal Wischen mit 2 Fingern (Flick)
- Seitenweise Scrollen in Listen (z. B. NPV)
- Seitenweise Scrollen in Dateien (z. B. NC-Programme)



Vertikal Wischen mit 3 Fingern (Flick)
- An Anfang oder Ende von Listen scrollen
- An Anfang oder Ende von Dateien scrollen

Horizontales Wischen mit 1 Finger (Flick)
- Scrollen in Listen mit vielen Spalten

Vergrößern (Spread)
- Vergrößern von Grafikinhalten (z. B. Simulation, Formenbauansicht)



Verkleinern (Pinch)
- Verkleinern von Grafikinhalten (z. B. Simulation, Formenbauansicht)

Verschieben mit 1 Finger (Pan)
- Verschieben von Grafikinhalten (z. B. Simulation, Formenbauansicht)
- Verschieben von Listeninhalten

Zum Handhaben der Fingergesten werden das Tag **gesture_event** und die Variable **$gestureinfo** verwendet. Sie ermöglichen Programmaktionen für dekodierte Fingergesten.

3.12.2 Fingergesten programmieren

Tag gesture_event

Hinweis

Dieses Tag wird nur ausgeführt, wenn die Bedientafel Multitouch-Bedienung unterstützt.

Erkennt die Bedien-Software eine Fingergeste, stellt der Parser die Gesteninformationen in der Variablen **$gestureinfo** bereit und führt das Tag **gesture_event** aus. Die Variable besitzt den Datentyp **GestureInfoStruct** und beinhaltet folgende Attribute:

Attribut	Wert	Bedeutung
type	3	Geste Verschieben
	4	GesteVerkleinern
	5	Geste Antippen
flag	1	Skalierungsfaktor geändert
	2	Drehwinkel geändert
state	1	Gestartet
	2	Aktualisiert
	3	Beendet
item_data		Identifikation des aktiven Controls
	-1	Die Geste ist keinem Control zugeordnet
	!= -1	Die Geste wurde in einem Control ausgeführt, dem dieser Wert beim Erstellen zugeordnet wurde
point		Position der Geste
	point.x	X-Position der Geste
	point.y	Y- Position der Geste
delta		Differenz zwischen dem Beginn der Geste und dem aktuellen Event
	<Skalierungsdifferenz>	Bei geändertem Skalierungsfaktor
	<Winkeldifferenz>	Bei geändertem Drehwinkel

Die Attribute sind in der Datei **struct_def.xml** vordefiniert.

Weitere Informationen unter Kapitel "Datei struct_def.xml erstellen (Seite 141)"

3.12.3 Gestensteuerung für Grafiken

Control-Variable imagebox

Für die Gestensteuerung von Grafiken gibt es folgende drei Erweiterungen bei der Control-Variable **Imagebox:**

Attribut	Bedeutung/Verhalten
rotationangle	Der Attributwert gibt den Winkel an, in dem die Grafik dargestellt werden soll.
setrotationmode	Der Attributwert **true** erlaubt die Verarbeitung der Pinch-Geste
setzoommode	Der Attributwert **true** erlaubt das Vergrößern/Verkleinern sowie das Verschieben einer Grafik im Anzeigebereich. Dieses Attribut ist standardmäßig auf **false** gesetzt.

Syntax

```
<property rotationangle="Winkel" />
<property setrotationmode="true/false" />
<property setzoommode="true/false" />
```

Beispiel Bitmap drehen

Bitmap um 30,5 Grad im Uhrzeigersinn drehen:

```
<let name="image_box_pict_name" type="string" >pic1.bmp</let>
…
…
…
 <control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property rotationangle="30.5" />
 <property setrotationmode ="true" />
 </control>
```

Beispiel Bitmap vergrößern

Vergrößern einer Bitmap zulassen:

```
<control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property setzoommode="true" />
 </control>
```

Control-Funktion imageboxset

Weiter lassen sich die Eigenschaften über die Funktion **control.imageboxset** setzen.

Folgende Kommandos stehen zur Verfügung:

Kommando	**Bedeutung/Verhalten**
SetRotationAngle	Die Grafik wird um den im **lparam1** angegebene Winkel gedreht.
SetRotationMode	Der Wert von **lparam1** legt die Auswertung der Pinch-Geste bezüglich der Rotation fest. Wert gleich 1 - die Geste wird vom Control verarbeitet
SetZoomMode	Ist der Wert vom **lparam1** gleich 1, erfolgt die Auswertung der Pinch-Geste zum Vergrößern/Verkleinern sowie Verschieben der Grafik.

3.12.4 Gestenverarbeitung

Tag message

Ist der Skalierungsfaktor größer 1.0 wird das Bild im Darstellungsbereich verschoben. Bei einem Faktor von 1.0 kann diese Fingergeste z. B. zum Blättern in einer Liste von Bildern verwendet werden. In diesem Fall sendet der Parser eine Nachricht an die Form, die im Tag **message** ausgewertet werden kann.

Der Message Parameter **$message_par1** beinhaltet den Item_Data – Wert des Controls. Der Message Parameter **$message_par2** signalisiert die Bewegungsrichtung der Geste.

Folgende Werte kann der **$message_par2** annehmen:

- -1 nach links rollen
- 1 nach rechts rollen
- -2 nach oben rollen
- 2 nach unten rollen

Beispiel

```
…
…
<let name="image_box_pict_name" type="string" ></let>
<let name="pictindex" />
<let name="image_box_list" type="string" dim="4">
 "pic1.bmp",
 "pic2.bmp",
 "pic3.bmp",
 "pic4.bmp",
</let>

…
…
<control name="image_view" xpos="100" ypos="123" width="300"
height="200" fieldtype="imagebox" refvar="image_box_pict_name"
hotlink="true" >
 <property item_data="1000" />
 <property setzoommode="false" />
</control>
…
…
…
<!--
 -1 nach links rollen
1 nach rechts rollen
-2 nach oben rollen
2 nach unten rollen
 -->
<message>
 <if condition="$message_par1 == 1000">
```

```
  <then>

   <switch>
    <condition>$message_par2</condition>
    <case value="1">
     <op>
      pictindex = pictindex+1;
     </op>
     <if condition="pictindex > 3">
      <then>
       <op>
        pictindex = 0;
       </op>
      </then>
     </if>
    </case>
    <case value="-1">
     <op>
      pictindex = pictindex-1;
     </op>
     <if condition="pictindex < 0">
      <then>
       <op>
        pictindex = 3;
       </op>
      </then>
     </if>
    </case>
   </switch>
    <op>
     image_box_pict_name = image_box_list[$pictindex];
    </op>
   </then>
  </if>
 </message>
…
…
```

3.13 Eigene Buttons projektieren

Buttons lassen sich als Aktionsknöpfe oder Auswahlknöpfe in eine Form einbinden.

3.13.1 Push-Button

Tag property

Der Push-Button ist ein Steuerelement, das als Taster oder Schalter (Taster mit Einrastfunktion) eingesetzt werden kann. Der Button lässt sich per Attributdefinitionen im Stil "Softkey Look & Feel" sowie im anwenderspezifischen Stil gestalten. Die diesbezüglichen Attribute sind mit dem Tag **property** zu definieren.

Das Ändern der Vorder- und Hintergrundfarbe ist mit den Attributen **color_fg**, **color_bk**, **color_fg_pressed** und **color_bk_pressed** möglich.

Die Zustände **gedrückt/eingerastet** oder **losgelassen** werden durch die Werte Eins oder Null repräsentiert. Zum Auswerten der Zustandsänderung eines Tasters ist die Angabe einer Handler-Funktion notwendig. Die Angabe der Handler-Funktion erfolgt mit dem Attribut **function** im Control-Tag.

Der Zustand eines Schalters kann durch das Auslesen der Control-Variablen oder einer zugewiesenen Referenzvariablen ermittelt werden.

Bei einer Touch-Bedienung erfolgt das Zustellen der Fingergesten "gedrückt" und "losgelassen" erst mit dem Beenden der Fingergeste "losgelassen".

Syntax

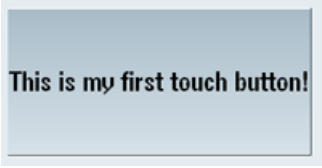
```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption></caption>
</control>
```

Oder mit Handler-Funktion

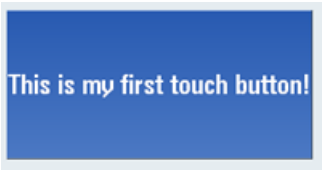
```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" function="button_handler" hotlink="true" >
 <caption></caption>
</control>
…
…
…
 <function_body name="button_handler">
…
…
…
 </function_body>
```

Oder

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" refvar="button_state" hotlink="true" >
 <caption></caption>

 <property checkable="true" />
</control>
```



ausgeschaltet



eingeschaltet, eingerastet

Beispiel

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" refvar="button_state" hotlink="true"
color_fg="#ff00ff" color_bk="#000eee">
 <property checkable="true" />
  <caption></caption>
 <property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
</control>
```

3.13.2 Funktionen des Push-Button

3.13.2.1 Sub-Tags für den Push-Button

Tag caption

Mit dem Tag **caption** legt der Programmierer den anzuzeigenden Button-Text für den nicht gedrückten Zustand fest. Standardmäßig erfolgt die Darstellung des Texts zentriert. Die Ausrichtung des Texts lässt sich mit dem Attribut **alignment** ändern. Folgende Attributwerte können angegeben werden:

Attributwert	Ausrichtung
left	linksbündig
right	rechtsbündig
top	oben
bottom	unten
center	zentriert

Soll für den gedrückten Zustand ein anderer Text angezeigt werden, ist eine zweite Caption-Anweisung mit dem Attribut **pressed** und dem Wert **true** zu programmieren.

Syntax

Zentrierte Darstellung

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption>Button text</caption>
</control>
```

Linksbündige Darstellung

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption alignment= "left">Button text</caption>
</control>
```

Gedrückte Darstellung

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption pressed="true">Button text pressed</caption>

</control>
```

3.13.2.2 Eigenschaften für den Push-Button

Zusätzliche Eigenschaften werden mit dem Tag **property** dem Control zugewiesen.

Tastereigenschaft festlegen

Das Attribut **checkable** ermöglicht den Button als Schalter zu verwenden. Dazu ist der Wert des Attributs auf **true** zu setzen.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <property checkable="true/false" />
</control>
```

Taster sperren

Das Attribut **disabled** steuert die Bedienbarkeit des Buttons. Ist der Wert **true**, erfolgt keine Verarbeitung von Bedienhandlungen.

Zustandsänderungen, die durch eine zugewiesene Referenzvariable hervorgerufen werden, führen zur Aktualisierung des Buttons.

Syntax

```
<property disabled="true/false" />
```

Beispiel

```
<control name="name" xpos="x position" ypos="y position"
fieldtype="pushbutton" >
 <caption alignment="left">Button text</caption>
 <property disabled="true " />
</control>
```

Ikonen zuweisen

Das vordefinierte Design lässt sich durch die Angabe eigener Ikonen bzw. Button-Farben überprägen. Für die Zustände "nicht gedrückt", "gedrückt" und "gesperrt" kann jeweils eine Ikone dem Button zugewiesen werden. Erfolgt keine Zuweisung für die Zustände "gedrückt" oder "gesperrt", zeigt das Control die Ikone für den Zustand "nicht gedrückt" an.

Weitere Attribute steuern die Ausrichtung, Skalierung und Transparenz der jeweiligen Ikone.

Attribut	Bedeutung/Verhalten
Picture	Ikone im Vordergrund Name der Ikone für den "nicht gedrückten" Zustand
PicturePressed	Ikone im Vordergrund Name der Ikone für den "gedrückten" Zustand
PictureDisabled	Ikone im Vordergrund Name der Ikone für den Zustand "gesperrt"

Attribut	Bedeutung/Verhalten
BackgroundPicture	Ikone im Hintergrund Name der Ikone für den "nicht gedrückten" Zustand
BackgroundPicturePressed	Ikone im Hintergrund Name der Ikone für den "gedrückten" Zustand
BackgroundPictureDisabled	Ikone im Hintergrund Name der Ikone für den Zustand "gesperrt"

Attribut alignment

Im Tag können weiterte Attribute angegeben werden, deren Werte sich auf das **alignment** der aufgeführten Ikonen-Zuweisungen beziehen:

Attributwert	Ausrichtung
left	linksbündig
right	rechtsbündig
top	oben
bottom	unten
center	zentriert
stretch	Hintergrund-Ikone: Wird der Wert **stretch** verwendet, skaliert der Parser die Ikone auf den Rechteckbereich des Buttons. Ein beliebiger Umriss des Buttons lässt sich erzeugen, indem die Transparentfarbe mit dem Attribut **transparent** bekanntgegeben wird.

3.13.2.3 Control-Variablen für den Push-Button

Nachträglich lassen sich die Eigenschaften des Buttons verändern, indem man den unten aufgeführten Attributvariablen neue Werte zuweist. Die Zuweisung erfolgt in einer Operationsanweisung durch die Angabe des Control-Namens, gefolgt vom Control-Variablennamen. Beide Namen sind durch einen Punkt voneinander zu trennen.

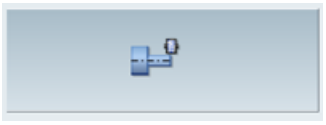
Syntax

<Control-Name>.<Control-Variable>

Control-Variable	Bedeutung/Verhalten
backgroundpicture	Variablentyp: String Name der Hintergrund-Ikone
backgroundpicturedisabled	Variablentyp: String Name der Hintergrund-Ikone für den gesperrten Zustand
backgroundpicturepressed	Variablentyp: String Name der Hintergrund-Ikone für den gedrückten Zustand
picture	Variablentyp: String Name der Vordergrund-Ikone
picturedisabled	Variablentyp: String Name der Vordergrund-Ikone für den gesperrten Zustand
picturepressed	Variablentyp: String Name der Vordergrund-Ikone für den gedrückten Zustand
picture_textalignedtopicture	Variablentyp: Bool Wert: 1 = true 0 = false Button-Text wird an der Ikone ausgerichtet
picturedisabled_textalignedtopicture	Variablentyp: Bool Wert: 1 = true 0 = false Button-Text wird an der Ikone "gesperrter Zustand" ausgerichtet
picturepressed_textalignedtopicture	Variablentyp: Bool Wert: 1 = true 0 = false Button-Text wird an der Ikone "gedrückter Zustand" ausgerichtet

Control-Variable	Bedeutung/Verhalten
backgroundpicture_keepaspectratio	Variablentyp: Bool Wert: 1 = true 0 = false Das Seitenverhältnis der Hintergrund-Ikone wird beibehalten oder ignoriert
backgroundpicturedisabled_keepaspectratio	Variablentyp: Bool Wert: 1 = true 0 = false Das Seitenverhältnis der Hintergrund-Ikone "gesperrter Zustand" wird beibehalten oder ignoriert
backgroundpicturepressed_keepaspectratio	Variablentyp: Bool Wert: 1 = true 0 = false Das Seitenverhältnis der Hintergrund-Ikone "gedrückter Zustand" wird beibehalten oder ignoriert
picture_keepaspectratio	Variablentyp: Bool Wert: 1 = true 0 = false Das Seitenverhältnis der Ikone wird beibehalten oder ignoriert
picturedisabled_keepaspectratio	Variablentyp: Bool Wert: 1 = true 0 = false Das Seitenverhältnis der Ikone "gesperrter Zustand" wird beibehalten oder ignoriert
picturepressed_keepaspectratio	Variablentyp: Bool Wert: 1 = true 0 = false Das Seitenverhältnis der Ikone "gedrückter Zustand" wird beibehalten oder ignoriert
backgroundpicture_scaled	Variablentyp: Bool Wert: 1 = true 0 = false Die Hintergrund-Ikone wird an die Dimension des Buttons angepasst

Control-Variable	Bedeutung/Verhalten
backgroundpicturedisabled_scaled	Variablentyp: Bool Wert: 1 = true 0 = false Die Hintergrund-Ikone "gesperrter Zustand" wird an die Dimension des Buttons angepasst
backgroundpicturepressed_scaled	Variablentyp: Bool Wert: 1 = true 0 = false Die Hintergrund-Ikone "gedrückter Zustand" wird an die Dimension des Buttons angepasst
picture_scaled	Variablentyp: Bool Wert: 1 = true 0 = false Die Ikone wird an die Dimension des Buttons angepasst
picturedisabled_scaled	Variablentyp: Bool Wert: 1 = true 0 = false Die Ikone "gesperrter Zustand" wird an die Dimension des Buttons angepasst
picturepressed_scaled	Variablentyp: Bool Die Ikone "gedrückter Zustand" wird an die Dimension des Buttons angepasst
backpicture_alignment	Variablentyp: String Siehe Attribut alignment
backgroundpicturedisabled_alignment	
backgroundpicturepressed_alignment	
picture_alignment	
picturedisabled_alignment	
picturepressed_alignment	
caption	Variablentyp: String Button-Text "nicht gedrückter Zustand"
captionpressed	Variablentyp: String Button Text "gedrückter Zustand"
caption_alignment	Variablentyp: String Siehe Attribut alignment
captionpressed_alignment	
captiondisabled_alignment	
disabled	Variablentyp: Bool Wert: 1 = true - Push-Button gesperrt 0 = false - Push-Buttom bedienbar



Picture

PicturePressed

PictureDisabled

BackgroundPicture + Picture

Attribut TextAlignedToPicture

Ist der Wert des Attributes **true**, wird der Text auf die Ikonen bezogen ausgerichtet.

```
<property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" />
```

Attribut scaled

Ist der Wert des Attributes **true**, erfolgt die Anpassung der Dimension des Bildes an die Dimension des Buttons in der Form, dass maximal 50 % der Höhe oder der Breite des Buttons der Grafik zur Verfügung gestellt wird.

```
<property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
```

Attribut stretch (nur für Hintergrundikonen wirksam)

Ist der Wert des Attributes **true**, erfolgt die Anpassung der Dimension des Bildes an die Dimension des Buttons.

```
<property backgroundpicture="f:\appl\pbutton.png"
alignment="stretch" />
```

```
<property backgroundpicture="f:\appl\pbutton.png"
alignment="stretch" transparent="#ffffff"/>
```

```
<caption alignment="left" >This is my first touch button!</caption>
 <property picture="sk_circ_grind_cg.png" alignment="left"
TextAlignedToPicture="true" scaled="true"/>
 <property backgroundpicture="pbutton.png" alignment="stretch"
transparent="#ffffff"/>
```

3.13.3 Switch on/off

Ein Switch-Control ist ein graphisches Element, das einen von zwei Zuständen durch eine Ikone signalisiert. Dieses Control ist per Touch oder Maus bedienbar.

Der eingenommene Zustand kann in einer Referenzvariablen gespeichert oder der Zustandswechsel in einer dem Control zugewiesenen Funktion ermittelt werden. Die Zuweisung erfolgt mit dem Attribut **function** im Control-Tag.

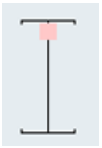
Das Attribut **alignment** ermöglicht die vertikale oder horizontale Ausrichtung des Buttons.

Dieses Control besitzt kein Standarddesign. Sind keine Ikonen dem Control zugeordnet, werden lediglich die Bounding-Box und die Schalterstellung (durch einen Punkt angezeigt) dargestellt.

Syntax

```
<control name="name" xpos = "x position" ypos="y position"
width="width" height="height" fieldtype="switch" alignment="hr/vr">
 <property left_position="value" />
 <property right_position="value" />
</control>
```

Attribut alignment



Schalter vertikal: Wert **vr**

Schalter horizontal: Wert **hr**

3.13.4 Funktionen des Switch

3.13.4.1 Eigenschaften für den Switch

Tag property

Folgende Schalterpositionen lassen sich mit dem Tag **property** dem Control zuordnen:

Attribut	Bedeutung/Verhalten
left_position	Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach links bewegt wurde.
right_position	Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach rechts bewegt wurde.
upper_position	Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach oben bewegt wurde.
lower_position	Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach unten bewegt wurde.
disabled	Das Attribut steuert die Bedienbarkeit des Schalters. Ist der Wert **true**, erfolgt keine Auswertung einer Bedienhandlung. Zustandsänderungen, die durch eine zugewiesene Referenzvariable hervorgerufen werden, führen zur Aktualisierung des Schaltzustandes.

Das endgültige Schalterdesign ist durch die Angabe weiterer Attribute festzulegen. Diese Attribute sind gemeinsam mit dem Attributen **left_position**, **right_position**, **upper_position** oder **lower_position** zu definieren.

Ein beliebiger Umriss des Schalters lässt sich erzeugen, indem eine Farbe als Transparentfarbe mit dem Attribut **transparent** bekanntgegeben wird.

Attribut	Bedeutung/Verhalten
picture	Ikone im Vordergrund Name der Ikone für den "nicht gedrückten" Zustand
pictureDisabled	Ikone im Vordergrund Name der Ikone für den Zustand "gesperrt"

Attribut	Bedeutung/Verhalten
transparent	Der Attributwert beinhaltet den Farbwert der transparenten Farbe. Dieser Farbwert wird für alle Ikonen die dem Switch zugeordnet sind verwendet.
caption	Der Attributwert beinhaltet den zur Schalterposition gehörenden Text. Dieser Text wird auf dem Element dargestellt und durch die Dimension des Schalters begrenzt.

3.13.4.2 Control-Variablen für den Switch

Nachträglich können die Eigenschaften des Schalters verändert werden, indem man den unten aufgeführten Control-Variablen neue Werte zuweist. Die Zuweisung erfolgt in einer Operationsanweisung durch die Angabe des Control-Namens, gefolgt von dem Control-Variablennamen. Beide Namen sind durch einen Punkt voneinander zu trennen.

Syntax

<Control-Name>.<Control-Variable>

Control-Variable	Bedeutung/Verhalten
Left_position	Variablentyp: Int Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach links bewegt wurde.
Right_position	Variablentyp: Int Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach rechts bewegt wurde.
Lower_position	Variablentyp: Int Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach unten bewegt wurde.
Upper_position	Variablentyp: Int Der Control-Variablen wird der Wert des Attributs zugewiesen, wenn der Schaltknopf nach oben bewegt wurde.
Picture_left_position	Variablentyp: String Name der Ikone für die linke Position
Picture_right_position	Variablentyp: String Name der Ikone für die rechte Position
Picture_upper_position	Variablentyp: String Name der Ikone für die obere Position
Picture_lower_position	Variablentyp: String Name der Ikone für die untere Position
Picturedisabled_left_position	Variablentyp: String Name der Ikone für die linke Position "gesperrter Zustand"
Picturedisabled_right_position	Variablentyp: StringName der Ikone für die rechte Position Position „gesperrter Zustand“
Picturedisabled_upper_position	Variablentyp: String Name der Ikone für die obere Position "gesperrter Zustand"
Picturedisabled_lower_position	Variablentyp: String Name der Ikone für die untere Position "gesperrter Zustand"

Control-Variable	Bedeutung/Verhalten
Caption_left_position	Variablentyp: String Text an der linken Position
Caption_right_position	Variablentyp: String Text an der rechten Position
Caption_upper_position	Variablentyp: String Text an der oberen Position
Caption_lower_position	Variablentyp: String Text an der unteren Position
disabled	Variablentyp: Bool Wert: 1 = true - Schalter gesperrt 0 = false - Schalter bedienbar

Horizontale Darstellung

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="switch" alignment="hr">
 <property left_position="value" picture="name" />
 <property right_position="value" picture="name" />
</control>
```

Vertikale Darstellung

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="switch" alignment ="vr">
 <property upper_position="value" picture="name" />
 <property lower_position="value" picture="name" />
</control>
```

Oder Handler-Funktion zuweisen

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="switch"
function="switch_handler" hotlink="true" alignment ="vr">
</control>
```

Beispiel

icon_left.bmp

icon_right.bmp

```
<let name="switch_state" />
```

```
<control name="main_switch" xpos="100" ypos="53" width="40"
height="30" fieldtype="switch" refvar="switch_state" hotlink="true"
alignment="hr">
 <property left_position="10" picture="icon_left.bmp" />
 <property right_position="11" picture="icon_right.bmp" />
</control>
```

Eigenschaften über Control-Variablen zuweisen

```
<control name="main_switch" xpos = "450" ypos="340" width="20"
height="46" fieldtype="switch" refvar="switch_statebf"
hotlink="true" alignment="vr">
 <property upper_position="1" />
 <property lower_position="0" />
</control>
…
…
…
<op>
  main_switch.transparent = #ffffff;
  main_switch.picture_upper_position = _T"switch_up.png";
  main_switch.picture_lower_position = _T"switch_bottom.png";
  main_switch.disable= 1;
</op>
```

3.13.5 Radio-Button

Radio-Buttons ermöglichen das Auswählen einer aus mehreren Optionen. Für jede Option ist ein Button zu erstellen. Alle zur einer Form, Groupbox oder Scrollarea gehörenden Radio-Buttons interagieren miteinander, so dass nur ein Button als ausgewählt markiert ist. Die Button-Zustände werden auf die Control-Variablen übertragen.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="radiobutton"  >
 <caption>button text</caption>
</control>
```

3.13.6 Checkbox

Checkboxen ermöglichen das Auswählen mehrerer Optionen. Für jede Option ist ein Button zu erstellen. Der Zustand der Checkbox wird auf die Control-Variable übertragen.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="checkbox"  >
 <caption>button text</caption>
</control>
```

3.13.7 Groupbox

Eine Groupbox umschließt optisch eine Gruppe von Controls mit einem Rahmen und einem Titel. Sie dient zur Gruppierung von logisch zusammenhängenden Controls, die nur innerhalb der Gruppe interagieren sollen. Die Koordinaten der eingebetteten Controls beziehen sich auf die linke obere Ecke unterhalb der Titelzeile.

Alle zur Gruppe gehörenden Controls werden als Sub-Controls in das Control-Tag eingebettet.

Bei der Vergabe der eingebetteten Control-Namen und dem Wert **Item_Data** ist zu beachten, dass diese eindeutig in Bezug auf alle zur Form gehörenden Controls sein müssen.

Der Titel der Groupbox wird mit dem Tag **caption** festgelegt.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="groupbox">
 <caption>caption text</caption>
  <control>…</control>
…
…
…
</control>
```

3.13.8 Scroll-Area

Eine Scroll-Area wird verwendet, um Controls innerhalb eines festgelegen Bereichs anzuzeigen. Die Koordinaten der eingebetteten Controls beziehen sich auf die linke obere Ecke der Scroll-Area. Werden Controls außerhalb des sichtbaren Bereichs angelegt, wird das Verschieben des sichtbaren Bereichs ermöglicht. Alle zur Area gehörenden Controls sind als Sub-Controls in das Control-Tag einzubetten.

Bei der Vergabe der eingebetteten Control-Namen und dem Wert **Item_Data** ist zu beachten, dass diese eindeutig in Bezug auf alle zur Form gehörenden Controls sein müssen.

Syntax

```
<control name="name" xpos="x position" ypos="y position"
width="width" height="height" fieldtype="scrollarea">
  <control>…</control>
…
…
…
</control>
```

Touch-Bedienung

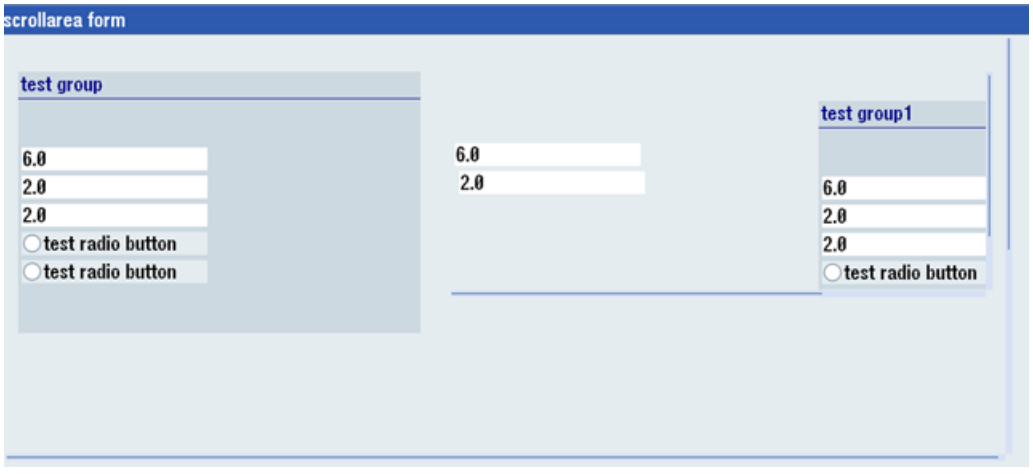
Wird der Fokus auf ein Control gesetzt, das momentan nicht vollständig sichtbar ist, erfolgt die Verschiebung des sichtbaren Bereichs, bis das Control vollständig angezeigt werden kann.

Tab-Geste

Diese Geste wird an das Skript weitergeleitet, wenn die Geste keinem eingebetteten Control zugeordnet werden kann.

Beispiel

Verschachtelte Scrollareas



```
<let name="CB1" >1</let>
<let name="RB1" />

<form name="scrollarea_form">
 <init>
  <caption>scrollarea form</caption>
   <data_access type="true" />
    <control name= "c_scroll" xpos = "2" ypos="24" width="520"
height="300" fieldtype="scrollarea" >
    <control name= "c_group" xpos = "6" ypos="24" width="208"
height="186" fieldtype="groupbox" >
    <caption>test group</caption>
    <control name = "c18" xpos = "2" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
    <control name = "c19" xpos = "2" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
    <control name = "c20" xpos = "2" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
    <control name="radiobg" xpos = "2" ypos="114"
fieldtype="radiobutton" >
    <caption>test radio button</caption>
    <property backgroundpicture="f:\appl\cycle_my1.png" />
   </control>
   <control name="radiobg1" xpos = "2" ypos="134"
fieldtype="radiobutton" >
    <caption>test radio button</caption>
    <property backgroundpicture="f:\appl\cycle_my1.png" />
   </control>
  </control>
 <control name= "c_scroll_emb" xpos = "230" ypos="24" width="280"
height="160" fieldtype="scrollarea" >

 <control name = "c18emb" xpos = "2" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
 <control name = "c19emb" xpos = "4" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
 <control name = "c20emb" xpos = "3" ypos = "194" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
 <control name= "c_group1" xpos = "190" ypos="24" width="208"
height="186" fieldtype="groupbox" >
 <caption>test group1</caption>
 <control name = "c18gemb" xpos = "2" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
 <control name = "c19gemb" xpos = "2" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
 <control name = "c20gemb" xpos = "2" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
 <control name="radiobggemb" xpos = "2" ypos="114"
fieldtype="radiobutton" >
  <caption>test radio button</caption>
  <property backgroundpicture="f:\appl\cycle_my1.png" />
</control>
```

```
<control name="radiobg1emb" xpos = "2" ypos="134"
fieldtype="radiobutton" >
 <caption>test radio button</caption>
  <property backgroundpicture="f:\appl\cycle_my1.png" />
 </control>
 </control>
 </control>
<control name = "c22" xpos = "2" ypos = "400" refvar="nck/Channel/
Parameter/R[12]" hotlink="true" />
 <control name="ChB1" xpos="208" ypos="430" width="50" height="30"
fieldtype="checkbox" refvar="PLC/M100.7" hotlink = "true"
color_bk="#00ffff">
  <caption>Check1</caption>
 </control>
  <control name="ChB2" xpos="208" ypos="460" width="58" height="30"
fieldtype="checkbox" refvar="PLC/M100.6" hotlink = "true"
color_bk="#00ffff">
  <caption>Check2</caption>
 </control>
  <control name="Radio1" xpos="208" ypos="490" width="40"
height="30" fieldtype="radiobutton" refvar="PLC/M100.0" hotlink =
"true" color_bk="#00ffff">
   <caption>Radio1</caption>
  </control>
   <control name="Radio2" xpos="208" ypos="520" width="45"
height="30" fieldtype="radiobutton" refvar="PLC/M100.1" hotlink =
"true" color_bk="#00ffff">
    <caption>Radio2</caption>
  </control>
   <control name="Radio3" xpos="208" ypos="550" width="50"
height="30" fieldtype="radiobutton" refvar="PLC/M100.2" hotlink =
"true" >
    <caption>Radio3</caption>
   </control>
    <control name="VChB1" xpos="8" ypos="430" width="50" height="30"
refvar="PLC/MB100" hotlink = "true" color_bk="#00ffff"/>
    <control name="VChB2" xpos="8" ypos="465" width="53" height="30"
refvar="PLC/MB100" hotlink = "true" color_bk="#00ffff"/>
   </control>
   <control name="ChB3" xpos="208" ypos="340" width="60" height="30"
fieldtype="checkbox" refvar="CB1" >
    <caption>Check3</caption>
   </control>
   </init>
  <timer>
  </timer>
</form>
```

3.14 Sidescreen-Anwendung

3.14.1 Easy XML im Sidescreen

Die Skriptsprache Easy XML kann ebenfalls zum Erstellen von Dialogen im Sidescreen-Bereich angewendet werden. Zum Anzeigen der Dialoge stehen die Sidescreen-Komponenten **Page** und **Widget** zur Verfügung. Die Anbindung erfolgt über die Datei **slsidescreen.ini**. Sidescreen-Komponenten werden beim Hochlauf der Bedien-Software automatisch gestartet und erst mit dem Herunterfahren beendet. D. h., der Parser lädt die zugeordneten Skripte einmalig beim ersten Aktivieren der Sidescreen-Komponente.

Die Dimension des Sidescreen-Bereichs ist durch die Layout-Parameter aus der Datei **slsidescreen_<Bildschirmauflösung>.ini** festgelegt. Um eine davon unabhängige Projektierung zu ermöglichen, beträgt die nutzbare Breite 276 Pixel für Easy XML-Skripte. Die nutzbare Höhe eines Easy XML-Dialogs hängt von der verwendeten Sidescreen-Komponente ab. Bei einer Page stehen 768 Pixel zur Verfügung. Bei einem Widget wird die Höhe durch die Sidescreen-Verwaltung festgelegt und kann mit der Funktion **GETHMIResolution** erfragt werden.

Der Skriptparser erwartet ein Menü zum Aktivieren einer Form bzw. zum Verzweigen zu anderen Formen. Das Hauptmenü muss den Namen **main** erhalten. Im Sidescreen werden keine Softkey-Tags unterstützt, sodass das Navigieren zu anderen Formen über ein Bedienelement der Form erfolgen muss.

Die Anzahl von Sidescreen-Pages oder Sidescreen-Widgets wird durch den Parser nicht begrenzt.

3.14.2 Sidescreen-Dialoge einbinden

Das Einbinden von Pages oder Widgets erfolgt über die Datei **slsidescreen.ini**.

Pages sind in der Sektion **[Sidescreen]** anzulegen. Jede Page ist mit dem Schlüsselwort **PAGE** gefolgt von der fortlaufenden Page-Nummer und den notwendigen Attributen anzugeben. Das Attribut **name** legt den Objektnamen der Page fest. Das Attribut **implementation** gibt die Implementierungsbibliothek und den Klassennamen an. Beide Attribute sind durch ein Komma getrennt anzugeben. Pages vom Typ **SlSideScreenPage** können Sidescreen-Elemente einbinden. Dies geschieht über so genannte **ELEMENT**-Einträge. Die Regel zum Anlegen einer Elementzeile **ELEMENTxxx** entspricht der Regel zum Anlegen einer Page. Soll die Page nur Dialoge enthalten, die mit Easy XML projektiert wurden, ist **slagmforms.SlEESideScreenPage** als Wert für das Attribut **implementation** anzugeben.

Widgets können einem Sidescreen-Element in der Sektion **[Element_<name>]** hinzugefügt werden.

Die Regel zum Anlegen einer Widget-Zeile **WIDGETxxx** entspricht der Regel zum Anlegen einer Page.

Soll ein Easy XML-Sidescreen-Widget aktiviert werden, ist **slagmforms.SlEESideScreenWidget** als Wert für das Attribut **implementation** anzugeben.

Syntax

```
ELEMENT002= name:=<name>, implementation:=SlSideScreenElement

...
...

[Element_<name>]
WIDGET001= name:=<Widget Name>, implementation:=
slagmforms.SlEESideScreenWidget
```

Der Widget-Bezeichner wird standardmäßig zur Bildung des Main-Modulnamens verwendet.

Beispiel

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

Main-Modulname: sidescreen_proginfluence.xml

Weitere Eigenschaften

Weitere Eigenschaften können den Pages oder Widgets zugewiesen werden, indem eine Sektion mit dem Namen **PAGE_<pagename>**, **ELEMENT_<elmentname>** oder **WIDGET_<widgetname>** in die Datei **slsidescreen.ini** eingetragen wird.

Folgende Eigenschaften lassen sich einer Page, einem Element oder einem Widget zuweisen:

Attribut	Bedeutung/Verhalten
TextId	Sprachunabhängige Bezeichner des Textes
TextFile	Textdateiname

Attribut	Bedeutung/Verhalten
TextContext	Textkontext EASY_XML
Icon	Name der Ikone

Property	Bedeutung/Verhalten
maskPath	Die Property legt den zu verwendenden Main-Modulnamen fest
maskName	Die Property legt den zu verwendenden Main-Menünamen fest
focusable	Die Property legt fest, ob der Eingabefokus auf das Element gesetzt werden kann

Beispiel

```
[Page_sidescreen_proginfluence]
PROPERTY001= name:=focusable, type:=bool, value:="true"
Icon=sidescreen_proginfluence.png

[PAGE_sidescreen_proginfluence]
Icon= sidescreen_proginfluence.png
PROPERTY001= name:=focusable, type:=bool, value:="true"
PROPERTY002= name:=maskPath, type:=QString, value:="
sidescreen_proginfluence.xml"
PROPERTY003= name:=maskName, type:=QString, value:="main"
```

3.14.3 Sprach- und Textverwaltung

Zum Verwalten der sprachunabhängigen Texte kann jeder Komponente eine Textdatei zugeordnet werden. Der Textdateiname bildet sich aus dem Namen der Komponente, der Sprachversion und dem Extension ".ts".

Als Textkontext ist **EASY_XML** zu verwenden.

Syntax

<name>_<language>.ts

Beispiel

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage
```

Text-Dateiname: sidescreen_proginfluence**_eng.ts**

Wenn keine Textdatei angegeben wird, sucht der Parser in der Standardtextdatei **oem_xml_screens_xxx.ts** nach dem Textbezeichner.

3.14.4 Sidescreen-Komponenten

3.14.4.1 Sidescreen-Element

Ist eine Page mit der Standardimplementierung verknüpft, können dieser Page Elemente zugeordnet werden. Dafür ist eine Sektion **[Page_<page_name>]** anzulegen und alle zur Page gehörenden Elemente aufzulisten.

Beispiel

```
[Sidescreen]
PAGE001= name:=<page_name>, implementation:=SlSideScreenPage

[Page_<page_name>]

ELEMENT<nummer>= name:=<Element name>,
implementation:=SlSideScreenElement
```

Jedes Element benötigt eine Sektion **[Element_<element name>]** in der die Eigenschaften und zugehörigen Widgets aufgelistet sind.

```
[Element_<element_name>]
WIDGET001= name:=<widget_name>, implementation:= <Implementierung>
```

3.14.4.2 Sidescreen-Widget

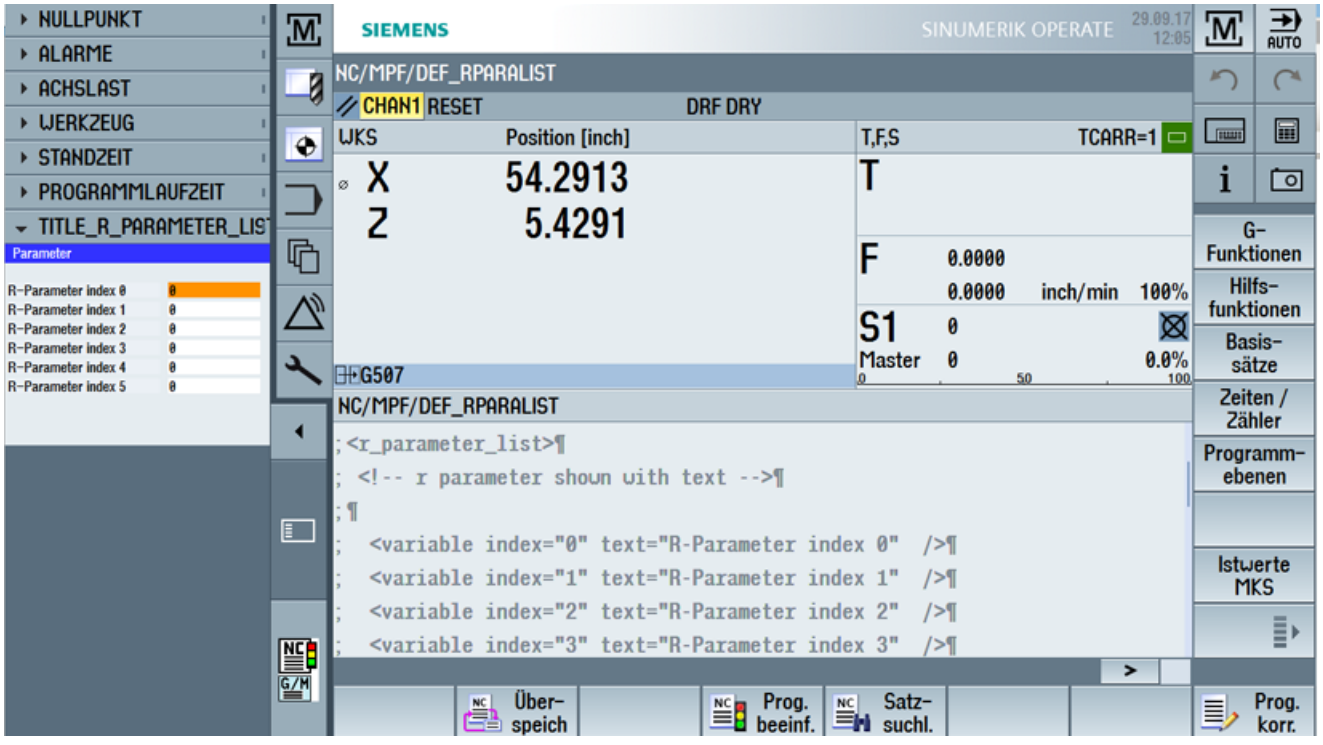
Einem Widget wird durch die Sidescreen-Verwaltung ein Bereich zugewiesen, in dem die projektierten Formen dargestellt werden können. Standardmäßig erhält ein Widget keinen Eingabefokus, sodass ein Editieren von Feldern nicht möglich ist. Dieses Verhalten ist durch das Angeben des Attributs **focusable** änderbar. Mit dem Öffnen des Widgets öffnet der Parser die zum Main-Menü gehörende Form. Innerhalb der Form kann mit einer Navigationsanweisung zu weiteren Menüs verzweigt werden.

Beispiel

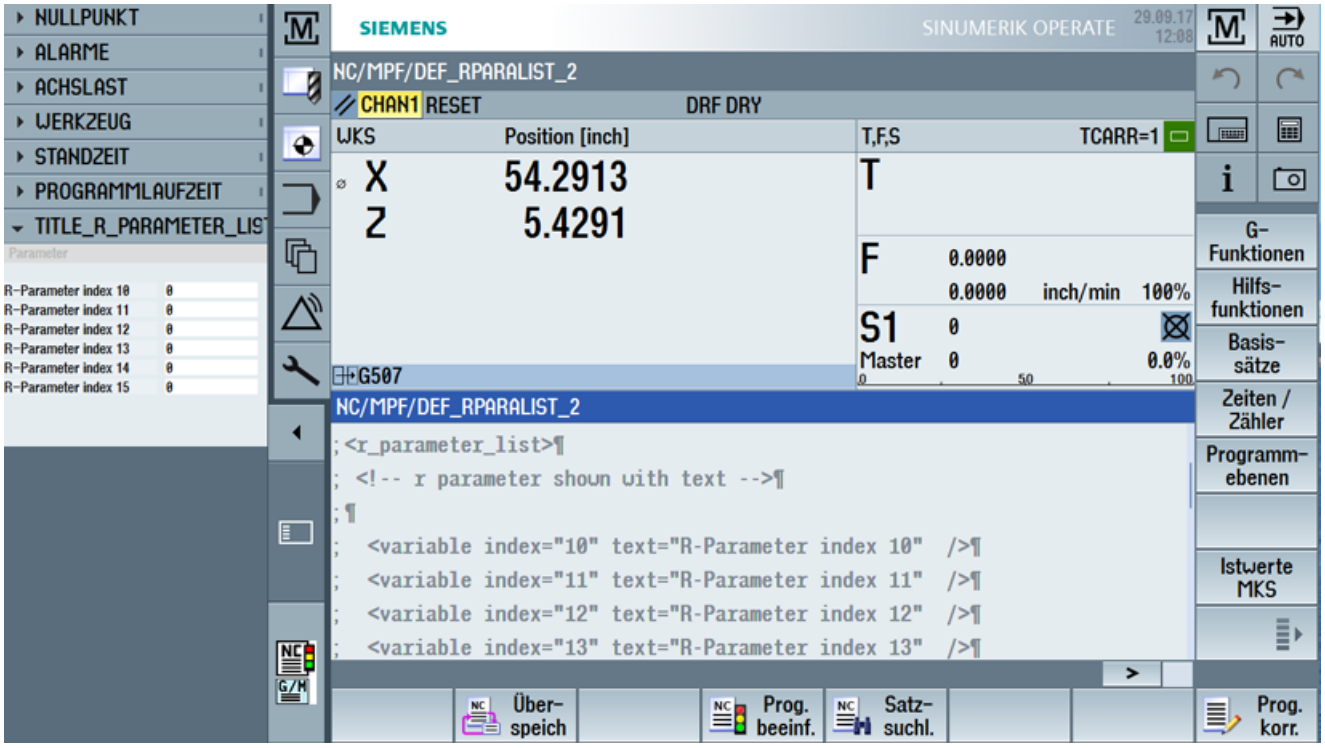
Das Easy XML-Skript sucht im aktiven Teileprogramm nach der Beschreibung für den Dateninhalt des Widgets. Im konkreten Beispiel wird die Beschreibung einer Liste mit R-Parametern erwartet, die zur Anzeige gebracht werden sollen. Jedem Parameter kann ein Beschreibungstext zugeordnet werden.

Toolbox Beispiel

Custom Screen Sample\
side_screen_examples\sidescreenwidget\displayRparameter\rparameter_widget.xml



Wird ein anderes Teileprogramm angewählt, erfolgt das automatische Überarbeiten des Widgets.


NULLPUNKT
ALARME
ACHSLAST
WERKZEUG
STANDZEIT
PROGRAMMLAUFZEIT
TITLE_R_PARAMETER_LIST
Parameter
R-Parameter index 10 0
R-Parameter index 11 0
R-Parameter index 12 0
R-Parameter index 13 0
R-Parameter index 14 0
R-Parameter index 15 0
SIEMENS
SINUMERIK OPERATE
29.09.17
12:08
NC/MPF/DEF_RPARALIST_2
CHAN1 RESET
DRF DRY
WKS
Position [inch]
X 54.2913
Z 5.4291
T,F,S
TCARR=1
T
F 0.0000
0.0000 inch/min 100%
S1 0
Master 0 0.0%
G507
NC/MPF/DEF_RPARALIST_2
;<r_parameter_list>¶
; <!-- r parameter shoun uith text -->¶
;¶
; <variable index="10" text="R-Parameter index 10" />¶
; <variable index="11" text="R-Parameter index 11" />¶
; <variable index="12" text="R-Parameter index 12" />¶
; <variable index="13" text="R-Parameter index 13" />¶
AUTO
G-Funktionen
Hilfs-funktionen
Basis-sätze
Zeiten / Zähler
Programm-ebenen
Istwerte MKS
Über-speich
Prog. beeinf.
Satz-suchl.
Prog. korr.

3.14.4.3 Sidescreen-Page

Eine Page, die über die Implementierung **slagmforms.SIEESideScreenPage** gestartet wird, stellt den gesamten Sidescreen-Bereich einer Form zur Verfügung. Sie besitzt keine Titelzeile, sodass eine Überschrift nicht mit der Caption-Anweisung projektiert werden kann. Standardmäßig erhält eine Page keinen Eingabefokus, sodass ein Editieren von Feldern nicht möglich ist. Dieses Verhalten ist durch das Angeben des Attributs **focusable** änderbar. Mit dem Öffnen der Page öffnet der Parser die zum Main-Menü gehörende Form. Innerhalb der Form kann mit einer Navigationsanweisung zu weiteren Menüs verzweigt werden.

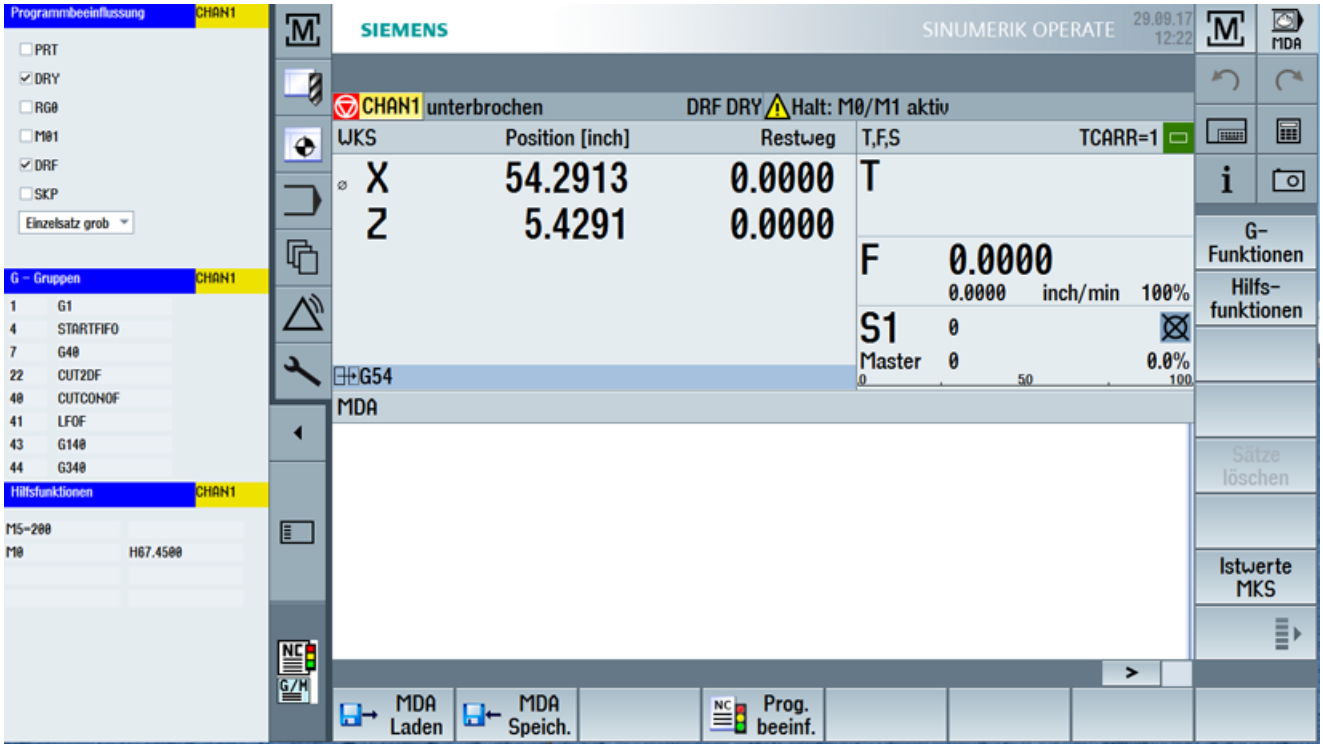
Beispiel

Die Sidescreen-Page wird als Page angezeigt, die ausschließlich eine Easy XML-Form anzeigt.

In diesem Beispiel erfolgt das Anzeigen von Zusatzinformationen in Sidescreen-Bereich.

Toolbox Beispiel

Custom Screen Sample\side_screen_examples\sidecreenpage\sidescreen_proginfluence.xml




```
[Sidescreen]
PROPERTY001= name:=minimizable, type:=bool, value:="true"
PROPERTY002= name:=buttonBarVisible, type:=bool, value:="true"
PAGE001= name:=WidgetsPage, implementation:=SlSideScreenPage
```

```
PAGE004= name:=sidescreen_proginfluence,
implementation:=slagmforms.SlEESideScreenPage

[Page_sidescreen_proginfluence]
PROPERTY001= name:=focusable, type:=bool, value:="true"
Icon=sidescreen_proginfluence.png
```

Textdatei: sidescreen_proginfluence_deu.ts

3.15 Dialoganwahl über PLC-Hardkeys

Einsatzbereich

Von der PLC können in der Bedien-Software folgende Funktionen initiiert werden:

- Anwahl von Bedienbereich
- Anwahl bestimmter Szenarien innerhalb von Bedienbereichen
- Ausführung auf Softkeys projektierte Funktionen

Hardkeys

Alle Tasten - auch die PLC-Keys - werden nachfolgend als Hardkeys bezeichnet. Es können maximal 254 Hardkeys definiert werden. Dabei gilt folgende Aufteilung:

Tastennummer	Verwendung
Key 1 – Key 9	Tasten an der Bedientafelfront
Key 10 – Key 49	reserviert
Key 50 – Key 254 Key 50 – Key 81	PLC-Keys: für OEM reserviert
Key 255	Durch Steuerinformation vorbelegt

Die Hardkeys 1 - 9 sind folgendermaßen vorbelegt:

Tastenbezeichnung		Aktion / Auswirkung
HK1	MACHINE	Anwahl Bedienbereich "Maschine", letzter Dialog
HK2	PROGRAM	Anwahl Bedienbereich "Programm", letzter Dialog oder letztes Programm
HK3	OFFSET	Anwahl Bedienbereich "Parameter", letzter Dialog
HK4	PROGRAM MANAGER	Anwahl Bedienbereich "Programm", Grundbild "Programm-Manager" keine Funktion
HK5	ALARM	Anwahl Bedienbereich "Diagnose", Dialog "Alarmliste"
HK6	CUSTOM	Anwahl Bedienbereich "Custom"
HK7	MENU SELECT	Anwahl "Grundmenü"
HK8	MENU FUNCTION	Anwahl Bedienbereich "Function"
HK9	MENU USER	Anwahl Bedienbereich "User"

Projektierung

Die Projektierung erfolgt in der Konfigurationsdatei systemconfiguration.ini im Abschnitt [keyconfiguration]. Jede Zeile definiert ein so genanntes Hardkey-Ereignis. Unter einem Hardkey-Ereignis wird die n-te Betätigung eines bestimmten Hardkeys verstanden. Zum Beispiel können die zweite und dritte Betätigung eines bestimmten Hardkeys unterschiedliche Reaktionen zur Folge haben.

Die Einträge in der Konfigurationsdatei systemconfiguration.ini können mit anwenderspezifischen Einstellungen überladen werden. Dazu stehen die Verzeichnisse [System user-Verzeichnis]/cfg und [System oem-Verzeichnis]/cfg zur Verfügung.

Die Zeilen zur Projektierung der Hardkey-Ereignisse haben folgenden Aufbau:

```
KEYx.n = area:=area, dialog:=dialog, screen:=screen, forms:=form,
menus:=menu, action:=menu.action, cmdline:=cmdline
KEYx.n = area:=area, dialog:=dialog, cmdline:=cmdline,
action:= action
```

x: Nummer des Hardkeys, Wertebereich: 1 – 254

n: Ereignisnummer – entspricht der n-ten Betätigung des Hardkeys, Wertbereich: 0 – 9

Voraussetzung

Das PLC-Anwenderprogramm muss folgende Voraussetzung erfüllen:

Es wird immer nur ein Hardkey abgearbeitet. Deshalb darf eine neue Anforderung nur dann gesetzt werden, wenn die Bedien-Software die vorhergehende Anforderung quittiert hat. Wenn das PLC-Anwenderprogramm den Hardkey aus einer MCP-Taste ableitet, dann muss es für eine ausreichende Zwischenpufferung der Taste(n) sorgen, damit auch bei schneller Bedienung kein Tastendruck verloren geht.

PLC-Nahtstelle

In der PLC-Nahtstelle wird ein Bereich für die Anwahl eines Hardkeys vorgesehen. Der Bereich befindet sich im DB19.DBB10. Hier kann die PLC direkt einen Tastenwert zwischen 50 und 254 vorgeben.

Die Quittung durch die Bedien-Software erfolgt in zwei Schritten. Diese Vorgehensweise ist erforderlich, damit der gleiche Tasten-Code zweimal hintereinander von der Bedien-Software korrekt als zwei separate Ereignisse erkannt werden kann. Im ersten Schritt wird die Steuerinformation 255 in das Byte DB19.DBB10 geschrieben. Durch diesen definierten virtuellen Tastendruck kann jede Tastensequenz der PLC eindeutig erkannt werden. Die Steuerinformation hat für das PLC-Anwenderprogramm keine Bedeutung und darf nicht verändert werden. Im zweiten Schritt erfolgt dann die eigentliche Quittung gegenüber der PLC, indem DB19.DBB10 gelöscht wird. Ab diesem Zeitpunkt kann das PLC-Anwenderprogramm einen neuen Hardkey vorgeben. Parallel dazu wird die Anforderung des aktuellen Hardkeys in der Bedien-Software bearbeitet.

Beispiel

Konfigurationsdatei:

```
; configuration of OP hardkeys (KEY1-KEY9) and
; PLC hardkeys (KEY50-KEY254)
[keyconfiguration]
; MACHINE key (hardkey block)
KEY1.0 = area:=AreaMachine, dialog:=SlMachine
; PROGRAM key (hardkey block)
KEY2.0 = area:=AreaProgramEdit
; OFFSET key (hardkey block)
```

```
KEY3.0 = area:=AreaParameter
; PROGRAM MANAGER key (hardkey block)
KEY4.0 = area:=AreaProgramManager
; ALARM key (hardkey block)
KEY5.0 = area:=AreaDiagnosis, dialog:=SlDgDialog,
cmdline:="-slGfwHmiScreen SlDgAeAlarmsScreen"
; CUSTOM (hardkey block)
KEY6.0 = area:=Custom
; MACHINE key (optional, Shift + F10)
KEY7.0 = area:=AreaMachine, dialog:=SlMachine,
cmdline:="-MKey"
; MENU USER (hardkey block, Ctrl + F10)
KEY10.0 = area:=MenuUser
; MENU FUNCTION (hardkey block, Ctrl + Shift + F10)
KEY11.0 = area:=MenuFunction
KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog
```

Die Area- und Dialogbezeichner sind der systemconfiguration.ini aus [System siemens-Verzeichnis]/cfg zu entnehmen.

Wird nur die Dialogangabe **SlEECustomDialog** verwendet, erwartet der Parser die Datei **xmldial.xml** im Applikationsverzeichnis sowie ein Menü-Tag mit dem Namen **main**. Andere Dateinamen bzw. Main-Menünamen können durch das Hinzufügen des Schlüssels **cmdline** bekannt gegeben werden. Der Parameter **-mainModule** legt die zu ladende XML-Beschreibung fest. In diesem Skript wird das Main-Menü vom Parser erwartet. Der Parameter **-entry** legt den Namen des Main-Menüs fest. Fehlt dieser Parameter, verwendet der Parser den Namen **main** zum Starten der Funktion. Der Parameter **-conf** mit dem Wert **slagmdialog.hmi** muss immer enthalten sein.

Ein Beispiel:

```
KEY50.0 = area:=CustomXML, dialog:=SlEECustomDialog,
cmdline:="-conf slagmdialog.hmi -mainModule restore.xml -entry cmc2"
KEY51.0 = area:=CustomXML, dialog:=SlEECustomDialog,
cmdline:="-conf slagmdialog.hmi -mainModule activate.xml
-entry cmc1"
```

Bedienoberfläche

Die folgenden Einträge in der Datei systemconfiguration.ini ermöglichen das Verwenden von Easy XML für die oben beschriebene Funktion.

```
AREA0111= name:=CustomXML, dialog:=SlEECustomDialog,
panel:=SlHdStdHeaderPanel
DLG056= name:=SlEECustomDialog,
implementation:=slagmdialog.SlEECustomDialog, process:=SlHmiHost1,
preload:=false, terminate:=true, cmdline:="-conf slagmdialog.hmi"
```

3.16 Anwenderdialoge reservierten Softkeys zuordnen

In allen Standardbedienbereichen ist ein Softkey für die Erweiterung der bestehenden Funktionalität reserviert. Zum Aktivieren und Verknüpfen eines OEM-Softkeys mit einem Anwenderprojekt sind im System in folgendem Verzeichnis entsprechende Dateien abgelegt:

/siemens/sinumerik/hmi/template/cfg

Diese Dateien beinhalten bedienbereichsspezifische XML-Beschreibungen, die zum Startzeitpunkt der Bedien-Software interpretiert und in den Menübaum des Bedienbereichs integriert werden.

Zur Integration einer eigenen Applikation ist die für den entsprechenden Bedienbereich relevante Datei aus dem Template-Verzeichnis in das folgende Verzeichnis zu kopieren und zu modifizieren:

/oem/sinumerik/hmi/template/cfg

3.16.1 Softkeytext sprachunabhängig festlegen

Das Tag TEXTFILE hält den Dateinamen der Textdatei, die dem OEM-Bereich zugeordnet werden soll. Der Dateiname ist ohne Sprach- und Dateierweiterung anzugeben.

Syntax

```
<TEXTFILE>oemtextfile</TEXTFILE>
```

Beispiel

oem_xml_screens_deu.ts

```
<TEXTFILE>oem_xml_screens</TEXTFILE>
```

Der anzuzeigende Softkeytext wird im Tag **property** im Tag **softkey** verwaltet. Der Wert OEM ist durch die gewünschte Text-ID zu ersetzen. Die Eigenschaft **translationContext** verweist auf einen Bereich in der Textdatei, dem der Text zugeordnet ist. Im Fall easyXML ist der Kontext EASY_XML anzugeben.

Syntax

```
<PROPERTY name="textID" type="QString">OEM</PROPERTY>
<PROPERTY name="translationContext" type="QString">OEMContext</
PROPERTY>
```

Beispiel

```
<PROPERTY name="textID" type="QString">MY_TEXT1</PROPERTY>
<PROPERTY name="translationContext" type="QString">EASY_XML</
PROPERTY>
```

3.16.2 Easy XML-Area zuweisen

Im Tag **softkey** ist weiterhin das Navigationsziel festzulegen. Dazu ist im Tag **area** das Navigationsziel **CustomXML** im Attribut **name** einzutragen. Die Attribute **dialog** und **screen** sind zu löschen, da diese Parameter automatisch festgelegt werden.

Syntax

```
<NAVIGATION target="area">
<AREA name="OEMArea" dialog="OEMDialog" screen="OEMScreen"/>
</NAVIGATION>
```

Beispiel

```
<NAVIGATION target="area">
<AREA name="CustomXML" />
</NAVIGATION>
```

Benutzt man das Navigationsziel **area**, erwartet der Parser die Datei xmldial.xml als Startmodul und das Menü **main** als Eintrittspunkt für die Menüverwaltung. Um mehrere Applikationen parallel anbieten zu können, ist die Funktion **switchToDialog** zu verwenden. Dafür ist das Tag NAVIGATION durch das Tag FUNCTION zu ersetzen. Dieser Funktion sind als Aufrufargumente der Bereich **CustomXML**, der Dialog **SlEECustomDialog** sowie der Name des Startmoduls und des Hauptmenüs zu übergeben.

Aus dem easyXML-Bereich kehrt man mit dem Tag **switchToArea** in den Standardbedienbereich zurück.

Syntax

```
<switchToArea name="<Area>"/>
...
<FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule
<Filename> -entry <Menuname>" name="switchToDialog"/>
```

Beispielprojekt

sldgarea_oem.xml

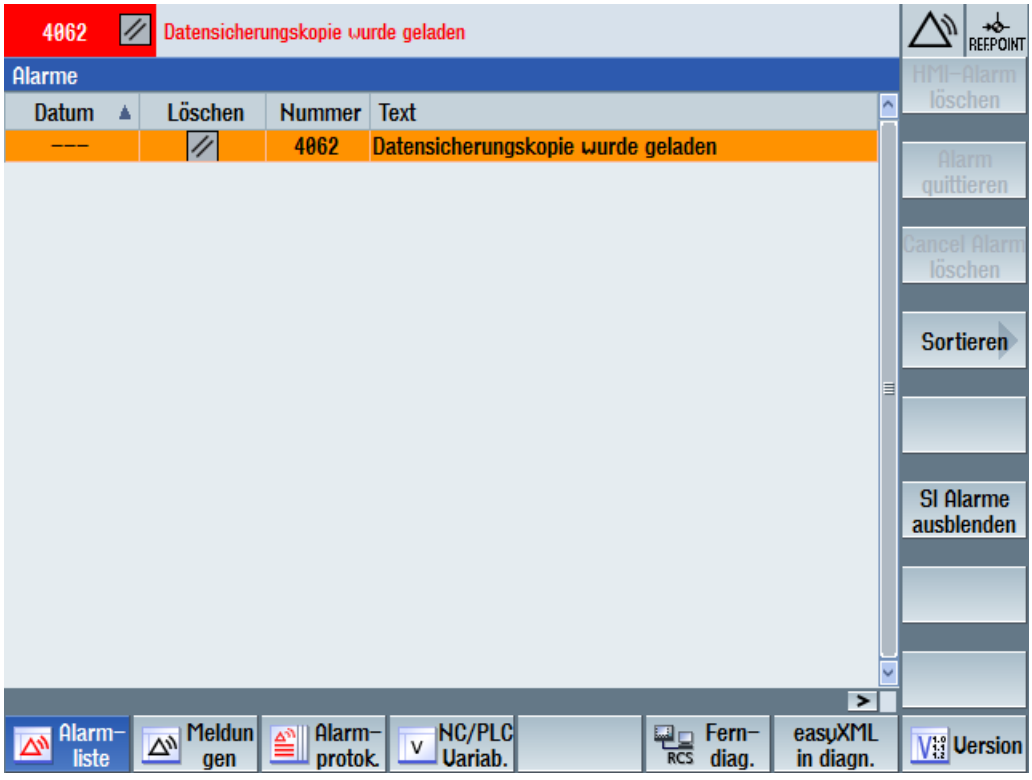
```
<!DOCTYPE DIALOG>
<DIALOG>
<TEXTFILE>oem_xml_screens</TEXTFILE>
<!-- ========================================================================
-->
<!-- OEM softkey number 7 on first horizontal main menu in area diagnosis --
>
<!-- ========================================================================
-->
<MENU name="DgGlobalHu">
 <ETCLEVEL id="0">
 <SOFTKEYGROUP name="GroupEtc">
 <SOFTKEY position="7">
  <PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
  <PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
  <NAVIGATION target="area">
   <AREA name="CustomXML" />
  </NAVIGATION>
 </SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>
</DIALOG>
```

oder

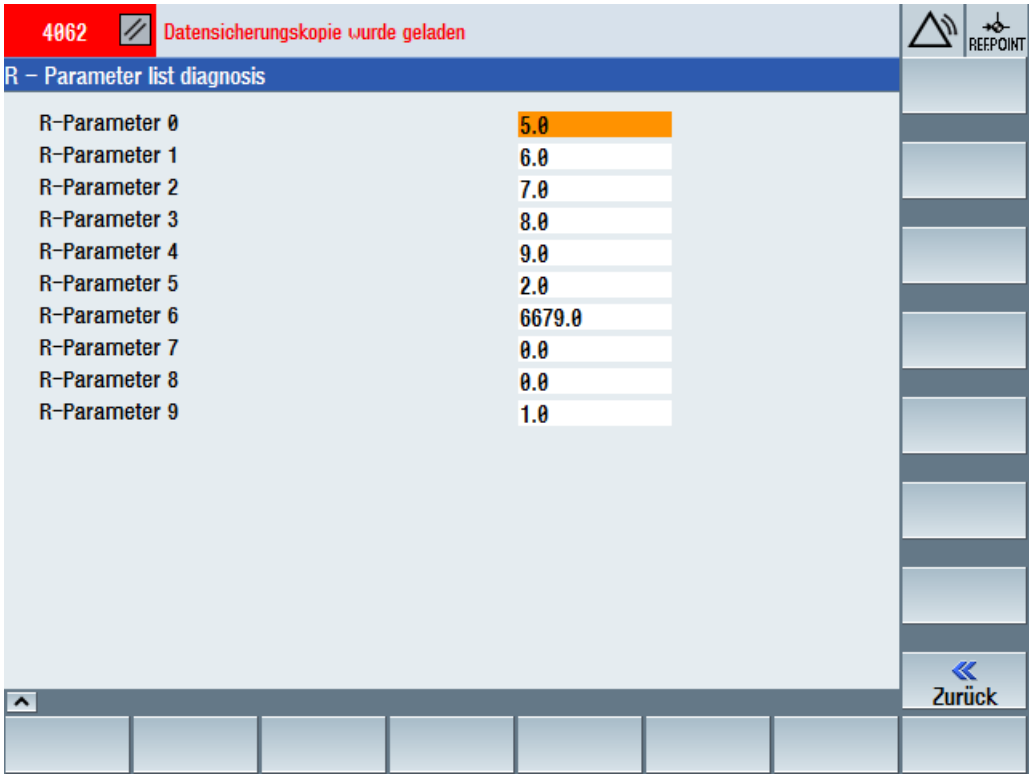
```
<!DOCTYPE DIALOG>
<DIALOG
<!-- ========================================================================
-->
<!-- OEM softkey number 7 on first horizontal main menu in area diagnosis --
>
<!-- ========================================================================
-->
<TEXTFILE>oem_xml_screens</TEXTFILE>
<MENU name="DgGlobalHu">
 <ETCLEVEL id="0">
 <SOFTKEYGROUP name="GroupEtc">
 <SOFTKEY position="7">
  <PROPERTY name="textID" type="QString">DG_SK</PROPERTY>
  <PROPERTY name="translationContext" type="QString">EASY_XML</PROPERTY>
  <FUNCTION args="-area CustomXML -dialog SlEECustomDialog -mainModule
dg.xml -entry main" name="switchToDialog"/>
 </SOFTKEY>
</SOFTKEYGROUP>
</ETCLEVEL>
</MENU>
</DIALOG>
```

dg.xml

```
<DialogGui>
<menu name = "main">
 <open_form name = "R_PARAMETER_LIST" />
 <softkey_back>
  <switchToArea name="AreaDiagnosis" dialog="SlDgDialog"/>
 </softkey_back>
</menu>
<!-- This form shows a R - parameter list -->
<form name = "R_PARAMETER_LIST">
 <init>
 <caption>R - Parameter list diagnosis</caption>
 <data_access type="true" />
  <control name = "c1" xpos = "322" ypos = "34" refvar="nck/Channel/
Parameter/R[0]" hotlink="true" />
  <control name = "c2" xpos = "322" ypos = "54" refvar="nck/Channel/
Parameter/R[1]" hotlink="true" />
  <control name = "c3" xpos = "322" ypos = "74" refvar="nck/Channel/
Parameter/R[2]" hotlink="true" />
  <control name = "c4" xpos = "322" ypos = "94" refvar="nck/Channel/
Parameter/R[3]" hotlink="true" />
  <control name = "c5" xpos = "322" ypos = "114" refvar="nck/Channel/
Parameter/R[4]" hotlink="true" />
  <control name = "c6" xpos = "322" ypos = "134" refvar="nck/Channel/
Parameter/R[5]" hotlink="true" />
  <control name = "c7" xpos = "322" ypos = "154" refvar="nck/Channel/
Parameter/R[6]" hotlink="true" />
  <control name = "c8" xpos = "322" ypos = "174" refvar="nck/Channel/
Parameter/R[7]" hotlink="true" />
  <control name = "c9" xpos = "322" ypos = "194" refvar="nck/Channel/
Parameter/R[8]" hotlink="true" />
  <control name = "c10" xpos = "322" ypos = "214" refvar="nck/Channel/
Parameter/R[9]" hotlink="true" />
 </init>
 <paint>
  <text xpos = "23" ypos = "34">R-Parameter 0</text>
  <text xpos = "23" ypos = "54">R-Parameter 1</text>
  <text xpos = "23" ypos = "74">R-Parameter 2</text>
  <text xpos = "23" ypos = "94">R-Parameter 3</text>
  <text xpos = "23" ypos = "114">R-Parameter 4</text>
  <text xpos = "23" ypos = "134">R-Parameter 5</text>
  <text xpos = "23" ypos = "154">R-Parameter 6</text>
  <text xpos = "23" ypos = "174">R-Parameter 7</text>
  <text xpos = "23" ypos = "194">R-Parameter 8</text>
  <text xpos = "23" ypos = "214">R-Parameter 9</text>
 </paint>
</form>
</DialogGui>
```


4062
Datensicherungskopie wurde geladen
REFPOINT
Alarme
Datum
Löschen
Nummer
Text

4062
Datensicherungskopie wurde geladen
HMI-Alarm löschen
Alarm quittieren
Cancel Alarm löschen
Sortieren
SI Alarme ausblenden
Alarm-liste
Meldungen
Alarm-protok.
NC/PLC Variab.
Fern-diag.
easyXML in diagn.
Version


4062
Datensicherungskopie wurde geladen
REFPOINT
R – Parameter list diagnosis
R-Parameter 0 5.0
R-Parameter 1 6.0
R-Parameter 2 7.0
R-Parameter 3 8.0
R-Parameter 4 9.0
R-Parameter 5 2.0
R-Parameter 6 6679.0
R-Parameter 7 0.0
R-Parameter 8 0.0
R-Parameter 9 1.0
Zurück

3.16.3 Tag-Beschreibungen

Tag softkey

Außerhalb des Tags **softkey** kann der Softkeyzustand mit dem Attribut **state** unter Verwendung des Attributes POSITION gesetzt werden. Als Attributwert ist die Softkeynummer anzugeben, für den der Zustand gesetzt werden soll.

Beispiel

```
<menu name = "menu_sktest">

 <state type="$$$define_sk_type" POSITION="2"/>

<softkey POSITION="2" type="user_controlled" picture="$$$bmp_name">
 <caption>Toggle%nSK</caption>
 <if>
 <condition>sk_type == 0 </condition>
 <then>
  <op> sk_type = 1 </op>
  <op> define_sk_type = _T"PRESSED" </op>
  <op>bmp_name = _T"f:\appl\red_led_on.bmp"</op>
  <state type="$$$define_sk_type" />
 </then>
 <else>
  <op> define_sk_type = _T"NOTPRESSED" </op>
  <op>bmp_name = _T"f:\appl\red_led_off.bmp"</op>
  <op> sk_type = 0 </op>
  <state type="$$$define_sk_type" />
 </else>
 </if>
  <print text="%s">sk_type_name</print>
  <navigation>menu_sktest</navigation>
</softkey>
...
...
```

Tag typedef

Hinweis

Die Vorbelegung der Elemente in den Beispielen muss entfernt werden.

Innerhalb der Typdefinition wird eine Variable durch das Tag **element** deklariert. Die Attribute des Tags entsprechen den Attributen der **let**-Anweisung. Die Vorbelegung der Elemente kann beim Anlegen der Variablen erfolgen.

Beispiel

RECT:

```
<typedef name="StructRect" type="struct" >
<element name="left" type="int" />
<element name="top" type="int" />
<element name="right" type="int" />
<element name="bottom" type="int" />
</typedef>
```

POINT:

```
<typedef name="StructPoint" type="struct" >
<element name="x" type="int" />
<element name="y" type="int" />
</typedef>
```

SIZE:

```
<typedef name="StructSize" type="struct" >
<element name="width" type="int" />
<element name="height" type="int" />
</typedef>
```

Tag let

Mit dem Attribut **dim** ist die Anzahl Feldelemente anzugeben. Bei einem zweidimensionalen Feld wird die zweite Dimension durch ein Komma getrennt nach der ersten Dimension angegeben. Der Zugriff auf ein Feldelement erfolgt über den Feldindex, der in eckigen Klammern nach dem Variablennamen anzugeben ist. Die Dimensionen der Arrays können direkt oder durch den Inhalt einer Variablen definieren werden.

Beispiel

```
 <let name="d1" >3</let>

<let name="list_string" dim="3" type="string" />
```

oder

```
<let name="list_string" dim="$d1" type="string" />
...
...
<op>
  list_string[2] = _T"test";
</op>
```

oder

```
<let name="d1" >3</let>
<let name="d2" >6</let>

<let name="list_string3" dim="3, $d2" type="string" />

<op>
  list_string3[2, 5] = _T"test";
</op>
```

3.17 Sprachunabhängige Texte erstellen

Alle in den Dialogmasken verwendeten Texte können sprachunabhängig verwaltet werden, indem man in den Dialogen Textbezeichner verwendet, die zur Laufzeit durch einen Text in der aktuell ausgewählten Sprache ersetzt werden. Dieser Text ist zusammen mit dem Textbezeichner für jede angebotene Sprache in einer Textdatei im UTF8-Format zu speichern.

Standardmäßig ist die Datei oem_xml_screens_<Sprachkürzel>.ts der Funktion Easy XML als Textdatei zugeordnet. Als Textkontext ist der Bezeichner EASY_XML anzugeben.

Zum Kennzeichnen eines Textbezeichners sind im Skript dem Textbezeichner zwei Dollarzeichen voranzustellen.

Aufbau

oem_xml_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>MY_TEXT1</source>
    <translation>text1 from textfile</translation>
    </message>
    <message>
    <source>MY_TEXT2</source>
    <translation>text2 from textfile</translation>
    </message>
</context>
</TS>
```

Beispiel

```
<text xpos ="120" ypos="158">$$MY_TEXT1</text>
```

3.18 Projektspezifische Textdateien

3.18.1 Projektspezifische Textdateien erstellen

Zum Verwalten von getrennten Projekten können für ein Projekt, jede Form und jedem Menü separate Textdateien verwendet werden. Die Bekanntgabe erfolgt mit dem Attribut TEXTFILE in den entsprechenden Tags.

Als Attributwert ist der Name der Textdatei ohne Sprachbezeichner und Dateierweiterung zu übergeben. Auf die Texte kann solange zugegriffen werden, bis die Form oder das Menü geschlossen werden. Wird eine Textdatei mit dem Tag **DialogGui** geladen, kann auf den Inhalt bis zum Verlassen des Projektes zugegriffen werden. Verwendet man einen Textkontext mehrfach, erfolgt das Suchen der Textbezeichner in der zuletzt geladenen Datei. Pro Textdatei kann ein Textkontext verwendet werden.

Beispiel

main_form_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>MAIN_FORM_TITLE</source>
    <translation>user text file title</translation>
    </message>
</context>
</TS>
```

Aktivierung

```
<DialogGui textfile="main_form_screens">
...
...
</DialogGui>
```

oder

```
<form name="main_form" textfile="main_form_screens">
...
...
</form>
```

oder

```
<menu name="main" textfile="main_form_screens">
...
...
</menu>
```

Beispieldialog

main_form_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>MAIN_FORM_TITLE</source>
    <translation>user text file title</translation>
    </message>
    <message>
    <source>MAIN_FORM_TEXT</source>
    <translation>text from text file</translation>
    </message>
</context>
</TS>
```

main2_sktext_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>SK1</source>
    <translation>Softkey1</translation>
    </message>
</context>
</TS>
```

form2_screens_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_XML</name>
    <message>
    <source>FORM2_TITLE</source>
    <translation>Form2 Title</translation>
    </message>
    <message>
    <source>FORM2_TEXT</source>
    <translation>Form2 text from text file</translation>
    </message>
</context>
</TS>
```

Dialogskript

xmldial.xml

```
<DialogGui textfile="main_form_screens">
<menu name = "main">
    <open_form name = "main_form" />
    <softkey POSITION="1">
    <caption>Menu 2</caption>
    <navigation>menu_2</navigation>
    </softkey>
</menu>
<menu name = "menu_2" textfile="main2_sktext_screens">
    <open_form name = "form2" />
    <softkey POSITION="1">
    <caption>$$SK1</caption>
    </softkey>
    <softkey_back>
    <navigation>main</navigation>
    </softkey_back>
</menu>
<form name="main_form" >
 <init>
    <data_access type = "true" />
    <caption>$$MAIN_FORM_TITLE</caption>
 </init>
 <paint>
    <text xpos="5" ypos="30">$$MAIN_FORM_TEXT</text>
 </paint>
</form>
<form name="form2" textfile="form2_screens">
 <init>
    <data_access type = "true" />
    <caption>$$FORM2_TITLE</caption>
 </init>
 <paint>
    <text xpos="5" ypos="30">$$FORM2_TEXT</text>
 </paint>
</form>
</DialogGui>
```

3.18.2 Textkontext angeben

Innerhalb einer Textdatei kann eine Gruppierung der Texte durch selbst festgelegte Bereichsnamen vorgenommen werden. Innerhalb des Tags **context** wird mit dem Tag **name** der Bereichsbezeichner festgelegt.

Syntax

```
<context>
 <name>name</name>
</context>
```

Beispiel

```
<context>
 <name>my context 1</name>
...
...
</context>
<context>
 <name>my context 2</name>
...
...
</context>
```

Wird mit einem eigenen Kontext gearbeitet, ist der Kontextname zusammen mit dem Textdateinamen für ein Projekt, eine Form oder ein Menü über das Attribut TEXTCONTEXT anzugeben. Auf die Texte, die innerhalb des Kontextes abgelegt sind, kann so lange zugegriffen werden, bis ein neuer Kontext gesetzt wird. D. h. ein am Projekt gesetzter Kontext wird durch den an einer Form oder einem Menü gesetzten Kontext ersetzt.

Projektbeispiel

form2_screens_deu.ts

```
?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
 <name>EASY_XML</name>
  <message>
   <source>FORM2_TITLE</source>
   <translation>Form2 Title</translation>
  </message>
  <message>
   <source>FORM2_TEXT</source>
   <translation>Form2 text from text file</translation>
  </message>
</context>
<context>
 <name>MY_FORM_CONTEXT</name>
  <message>
   <source>FORM3_TITLE</source>
   <translation>Title from the text context MY_FORM_CONTEXT</translation>
  </message>
  <message>
   <source>FORM3_TEXT</source>
   <translation>Form3 text from text file</translation>
  </message>
</context>
<context>
 <name>MY_SOFTKEY_CONTEXT</name>
  <message>
   <source>MENU3_SK1</source>
   <translation>SK%n1</translation>
  </message>
</context>
</TS>
```

using_textcontext.xml

```
...
<menu name = "menu3" textfile="form2_screens"
textcontext="MY_SOFTKEY_CONTEXT">
 <open_form name = "menu3_form" />
 <softkey POSITION="1">
  <caption>$$SK1</caption>
 </softkey>
 <softkey_back>
  <navigation>main</navigation>
 </softkey_back>
</menu>
<form name="menu3_form" textfile="form2_screens"
textcontext="MY_FORM_CONTEXT">
 <init>
  <data_access type = "true" />
  <caption>$$FORM3_TITLE</caption>
 </init>

 <paint>
  <text xpos="5" ypos="30">$$FORM3_TEXT</text>
 </paint>
</form>
...
...
```

3.18.3 Textfilelist angeben

Möchte man mehrere Textdateien für ein Projekt verwenden, kann man die Namen der benötigten Textdateien in einer Liste über das Attribut **textfilelist** dem Parser übergeben.

Die Notation der Textdateinamen erfolgt zeilenweise und ist mit einem Zeilenumbruch abzuschließen. Als Textdateiname wird der Name der Datei ohne Spracherweiterung und Dateiextension erwartet.

Beispiel

Die Datei `form2_screens_deu.ts` ist in der Liste mit `form2_screens` aufzuführen.

textfilelist.ini

```
form2_screens
...
...
```

Aktivierung

```
<DialogGui textfilelist="txtfilelist.ini">
...
...
</DialogGui>
```

3.19 Sitzung wiederherstellen

Mit dem Schließen eines Easy XML-Projektes werden alle lokalen Variablen freigegeben und Skriptanweisungen entladen. Möchte man Daten von Variablen über das Ausschalten der Steuerung erhalten, sind diese Variablen mit dem Attribut **permanent** zu versehen. Daten, die bis zum Ausschalten der Steuerung erhalten bleiben sollen, sind mit dem Attribut **poweroff** zu kennzeichnen.

Wird die Easy XML-Applikation wieder angewählt, kann man im Main-Menü steuern, welches Menü oder welche Form nach der erneuten Anwahl der Applikation aktiviert werden soll. Es obliegt dem Programmierer sicherzustellen, dass alle dafür benötigten Daten zur Verfügung stehen.

Wird das Attribut **restoresession** im Tag **DialogGUI** gesetzt, organisiert der Parser die Wiederanwahl des zuletzt geöffneten Menüs und der zuletzt geöffneten Form. Für die Bereitstellung der notwendigen Daten ist der Programmierer verantwortlich.

Inbetriebnahmedialoge erstellen

4

4.1 Funktionsübersicht

Verwendungszweck

Zusatzgeräte lassen sich auf einfache Art und Weise mit der Funktion "Easy Extend" in Betrieb nehmen, aktivieren, deaktivieren oder testen. Die verfügbaren Geräte und Gerätezustände zeigt die Steuerung in einer Liste an. Das System kann maximal 64 Geräte verwalten.

Das Aktivieren oder Deaktivieren eines Gerätes erfolgt per Softkey-Bedienung.

Die Funktion "Easy Extend" steht im Bedienbereich "Parameter" zur Verfügung.

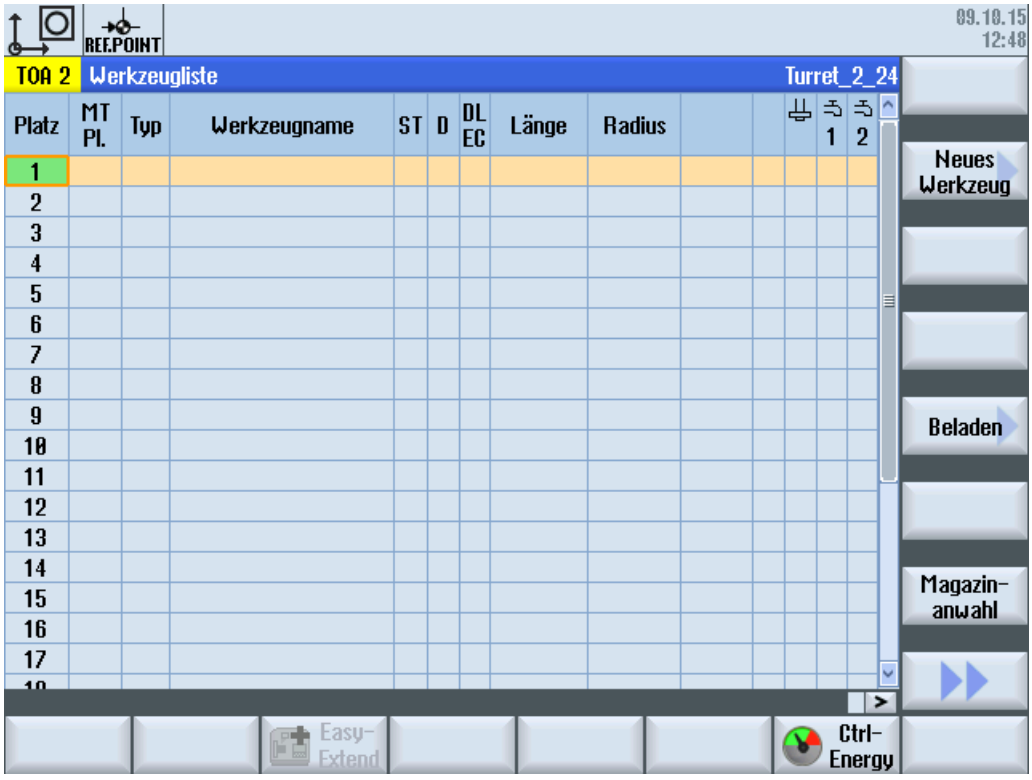


Bild 4-1 Grundbild "Parameter"

Projektierung

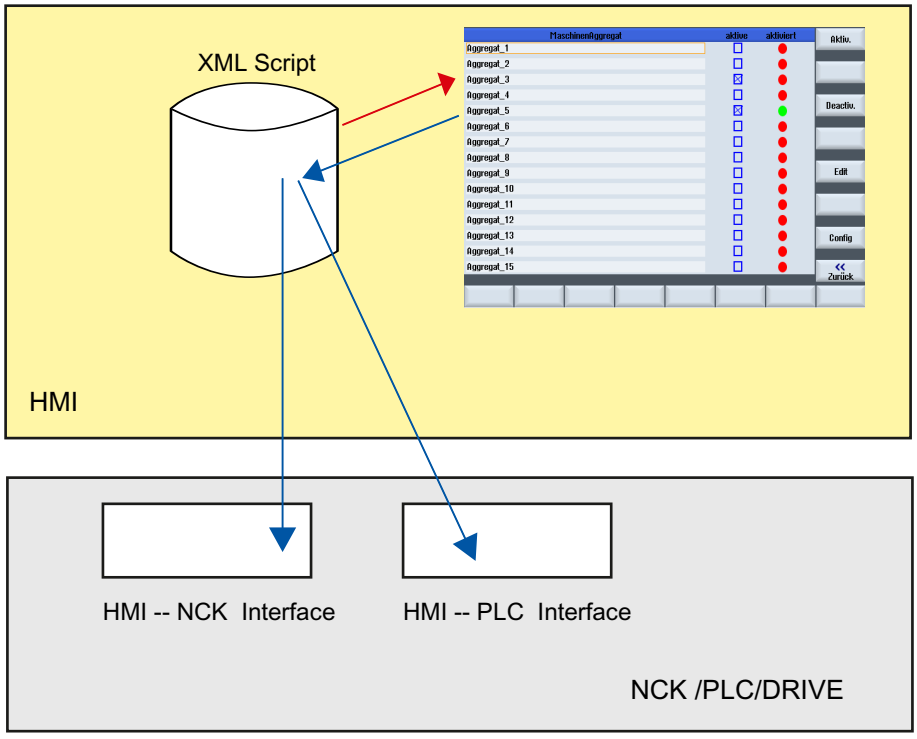


Bild 4-2 Funktionsweise von "Easy Extend"

Um die Funktion "Easy Extend" nutzen zu können, sind vom Maschinenhersteller folgende Funktionen zu projektieren:

- **Schnittstelle PLC ↔ HMI**
  Die Verwaltung der optionalen Geräte wird über die Schnittstelle zwischen der Bedienoberfläche und der PLC abgewickelt.
- **Skript-Verarbeitung**
  Der Maschinenhersteller hinterlegt in einem Anweisungsskript die Abläufe, die zum Installieren, Aktivieren, Deaktivieren und Testen eines Gerätes auszuführen sind.
- **Parameter-Dialog (optional)**
  Der Parameter-Dialog dient zum Anzeigen von Geräte-Informationen, die in der Skript-Datei hinterlegt worden sind.

Ablage der Dateien

Die Ablage der zu "Easy Extend" gehörenden Dateien erfolgt auf der System CompactFlash Card im Verzeichnis "dvm" (Maschinenhersteller).

Datei	Name	Zielverzeichnis
Textdatei	oem_aggregate_xxx.ts	\oem\sinumerik\hmi\lng
Skriptdatei	agm.xml	\oem\sinumerik\hmi\dvm
Archivdatei	beliebig	\oem\sinumerik\hmi\dvm\archives
PLC-Anwenderprogramm	beliebig	PLC

4.2 Projektierung im PLC-Anwenderprogramm

Konfigurationen laden

Die erstellten Konfigurationen werden zusammen mit der Skript- und Textdatei in das Herstellerverzeichnis der Steuerung übertragen. Zusätzlich ist das entsprechende PLC-Anwenderprogramm zu laden.

Programmierung der Geräte

Die Kommunikation zwischen der Bedienkomponente und der PLC erfolgt im PLC-Anwenderprogramm über den Datenbaustein DB9905, bei dem 128 Wörter für die Verwaltung der Geräte reserviert sind. Die Beschreibung für den Datenbaustein-Bezeichner ist dem Kapitel "PLC_INTERFACE (Seite 274)" zu entnehmen.

Beginnend mit dem Gerät 1 erfolgt die Vergabe der PLC-Wörter:

Datenbaustein	Gerätebezeichnung
DB9905.DBB0	Gerät 1
DB9905.DBB4	Gerät 2
...	...
DB9905.DBB192	Gerät 49
DB9905.DBB196	Gerät 50

Für jedes Gerät werden vier Bytes mit folgender Bedeutung verwendet:

Byte	Bit	Beschreibung	
0	0	== 1	Gerät ist in Betrieb genommen (HMI Rückmeldung)
	1	== 1	Gerät ist zu aktivieren (HMI Anforderung)
	2	== 1	Gerät ist zu deaktivieren (HMI Anforderung)
	3-7	reserviert	
1	0-7	reserviert	
2	0	== 1	Gerät ist aktiv (PLC Rückmeldung)
	1	== 1	Gerät ist fehlerhaft
	2-7	reserviert	
3	0-7	eindeutige Kennzeichnung des Gerätes	

Ablage der Dateien

Die Ablage der Dateien erfolgt auf der System CompactFlash Card im Verzeichnis "oem" (MANUFACTURER) und "oem_i" (INDIVIDUAL).

Hinweis

Der Dateiname darf nur Kleinbuchstaben enthalten.

Datei	Name	Zielverzeichnis
Textdatei	oem_aggregate_xxx.ts	/oem/sinumerik/hmi/lng/ /oem_i/sinumerik/hmi/lng/
Skriptdatei	agm.xml	/oem/sinumerik/hmi/dvm /oem_i/sinumerik/hmi/dvm
Archivdatei	beliebig	/oem/sinumerik/hmi/dvm/archives /oem_i/sinumerik/hmi/dvm/archives
PLC-Anwenderprogramm	beliebig	PLC

Genereller Ablauf

Zum Bereitstellen der notwendigen Daten sind vom Maschinenhersteller folgende Schritte auszuführen:

1. Erstellen eines PLC-Anwenderprogramms, das die Geräte in ihrer PLC-seitigen Ansteuerung berücksichtigt.
2. Inbetriebnahme der "Standardmaschine" mit anschließender Sicherung der Daten in ein Serien-IBN Archiv.
3. Anbau der Geräte, Inbetriebnahme und anschließendes Auslesen der Daten als Differenz-Serien-IBN Archiv.

Hinweis

Änderung der Maschinenkonfiguration

Ist das Ändern von Antriebsmaschinendaten notwendig, sollten diese zuerst in der Steuerung angepasst werden. Dieser Vorgang ist für alle Geräte und Konstellationen zu wiederholen.

Achsen hinzufügen

Wird die Maschine um Maschinenachsen erweitert, ist eine feste Reihenfolge beim Anbau der Drive-Objekte (DO) einzuhalten, da das Serien-Inbetriebnahme Archiv die Konstellation der Referenzmaschine des Maschinenherstellers beinhaltet und nicht bei einer geänderten Reihenfolge angewendet werden kann.

Es wird empfohlen für die "Steuerungskomponenten" folgende Einstellungen zu wählen:

- NC-Daten
- PLC-Daten
- Antriebs-Daten
  - ACX-Format (binär)

4.3 Darstellung auf der Bedienoberfläche

Dialoge auf der Bedienoberfläche

Folgende Dialoge sind für die Funktion "Easy Extend" verfügbar:

- Die Steuerung bietet einen **projektierbaren Dialog** an, in dem die verfügbaren Geräte angezeigt werden.
- Wurde noch keine Erstinbetriebnahme durchgeführt, öffnet die Steuerung den **Dialog zur Inbetriebnahme**.

Ist für das Gerät eine Inbetriebnahmeprozedur (XML-Anweisung: "START_UP") programmiert und das Gerät noch nicht in Betrieb genommen, startet die Steuerung die Inbetriebnahmeprozedur.

Dazu erfolgt zuerst eine vollständige Datensicherung, bevor die in der Skriptdatei hinterlegten Serien-IBN-Archive eingelesen werden.

Im Fehlerfall kann der Inbetriebsetzer entscheiden, ob das Zurückrollen der Inbetriebnahme erfolgen soll oder er manuell eventuelle Fehler in den Maschinenkonfigurationen beheben möchte.

- Mit der Funktion "Abbruch" ist ein vorzeitiger Abbruch der Inbetriebnahme möglich. Danach kopiert die Steuerung die zuvor gesicherten Inbetriebnahmedateien zurück.

Ist nach erfolgreichem Abschluss der Inbetriebnahme das Ausschalten der Maschine erforderlich, kann über die XML-Anweisung "POWER_OFF" programmiert werden, dass eine entsprechende Meldung an der Steuerung ausgegeben wird.

4.4 Sprachabhängige Texte erstellen

Aufbau der Textdatei

Die xml-Dateien mit den sprachabhängigen Texten sind im UTF8-Format zu erstellen:

Beispiele

oem_aggregate_eng.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_EXTEND</name>
    <message>
    <source>DEVICE_ONE</source>
    <translation>Device one</translation>
    </message>
    <message>
    <source>DEVICE_TWO</source>
    <translation>Device two</translation>
    </message>
</context>
</TS>
```

oem_aggregate_deu.ts

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE TS>
<TS>
<context>
    <name>EASY_EXTEND</name>
    <message>
    <source>DEVICE_ONE</source>
    <translation>Erstes Gerät</translation>
    </message>
    <source>DEVICE_TWO</source>
    <translation>Zweites Gerät</translation>
    </message>
</context>
</TS>
```

4.5 Anwenderbeispiel für ein Leistungsteil

Drive-Objekt aktivieren

Das zu aktivierende Drive-Objekt wurde bereits in Betrieb genommen und vom Maschinenhersteller wieder deaktiviert, um die Achse(n) optional zu vermarkten.

Folgende Schritte sind zum Aktivieren der Achse auszuführen:

- Antriebsobjekt über p0105 aktivieren.
- 2. Achse in den Kanal-Maschinendaten frei schalten.
- Antriebsmaschinendaten über p0971 sichern.
- Warten, bis die Daten geschrieben wurden.
- Neustart von NCK und Antrieben anstoßen.

Programmierung:

```
<DEVICE>
  <list_id>1</list_id>
  <name> "Antrieb aktiv schalten" </name>

  <SET_ACTIVE>
    <data name = "drive/dc/p105[DO5]">1</data>
    <data name = "$MC_AXCONF_MACHAX_USED[4]">5</data>
    <data name = "drive/dc/p971[DO5]">1</data>
    <while>
      <condition> "drive/dc/p971[DO5]" !=0 </condition>
    </while>
    <control_reset resetnc ="true" resetdrive = "true"/>
  </SET_ACTIVE>

  <SET_INACTIVE>
    <data name = "drive/dc/p105[DO5]">0</data>
    <data name = "$MC_AXCONF_MACHAX_USED[4]">0</data>
    <data name = "drive/dc/p971[DO5]">1</data>
    <while>
      <condition> "drive/dc/p971[DO5]" !=0 </condition>
    </while>
    <control_reset resetnc ="true" resetdrive = "true"/>
  </SET_INACTIVE>

</DEVICE>
```

PLC-gesteuertes Gerät aktivieren

Das Gerät wird über das Ausgangsbyte 10 angesprochen und meldet die Betriebsbereitschaft über das Eingangsbyte 9 an die PLC zurück.

Zum Aktivieren wird das Ausgangsbyte auf die festgelegte Kodierung gesetzt. Anschließend wartet die While-Schleife auf die Betriebsbereitschaft des Geräts.

Programmierung:

```
<SET_ACTIVE>
  <DATA name = "plc/qb10"> 8 </DATA>
  <while>
    <condition> "plc/ib9" !=1 </condition>
  </while>
</SET_ACTIVE>
```

4.6 Skript-Sprache

Hinweis

Alle in der Funktion Anwenderdialoge erstellen (Seite 27) beschriebenen Skript-Elemente bilden die Grundlage für die Funktion "Easy Extend". Zum Verwalten von Zusatzaggregaten wurden weitere Skipt-Elemente definiert.

Programmteile des Skripts

Das Skript ist in folgende Bereiche aufgeteilt:

- "Easy Extend" - Bezeichner
- Rahmen zum Festlegen der Aktionen, die für ein Gerät ausgeführt werden können
- Bezeichner für das Gerät
- Bezeichner zur Inbetriebnahme des Geräts
- Bezeichner zum Aktivieren des Geräts
- Bezeichner zum Deaktivieren des Geräts
- Bezeichner zum Testen des Geräts
- Bezeichner für den Parameter-Dialog

Die einzelnen Tags sind in den nachfolgenden Kapiteln beschrieben.

Beschreibung

Bezeichner <tag>	Bedeutung
AGM	Bezeichner für die Funktion "IBN Assistent"
DEVICE	Bezeichner zur Beschreibung des Geräts. **Attribute:** • **option_bit** Für die Optionsverwaltung wird dem Gerät eine feste Bitnummer zugewiesen.
NAME	Der Bezeichner legt den im Dialog anzuzeigenden Namen des Geräts fest. Wird eine Textreferenz verwendet, zeigt der Dialog den für den Bezeichner hinterlegten Text an.
START_UP	Der Bezeichner beinhaltet die Beschreibung der Abläufe, die zur Inbetriebnahme des Gerätes notwendig sind.

Bezeichner <tag>	Bedeutung
SET_ACTIVE	Der Bezeichner beinhaltet die Beschreibung der Abläufe, die zum Aktivieren des Gerätes notwendig sind. **Attribute:** • **timeout** Das Attribut ermöglicht die Angabe einer Timeoutzeit in Sekunden. Ist das Skript nach dieser Zeit noch nicht beendet, bricht das System die Bearbeitung ab.
SET_INACTIVE	Der Bezeichner beinhaltet die Beschreibung der Abläufe, die zum Abschalten des Gerätes notwendig sind. **Attribute:** • **timeout** Das Attribut ermöglicht die Angabe einer Timeoutzeit in Sekunden. Ist das Skript nach dieser Zeit noch nicht beendet, bricht das System die Bearbeitung ab.
TEST	Der Bezeichner beinhaltet die Anweisungen, mit denen ein Gerät auf dessen Funktionsfähigkeit getestet werden kann. **Attribute:** • **timeout** Das Attribut ermöglicht die Angabe einer Timeoutzeit in Sekunden. Ist das Skript nach dieser Zeit noch nicht beendet, bricht das System die Bearbeitung ab.
UID	Eindeutiger numerischer Bezeichner zur Identifikation des Geräts in der Schnittstelle PLC ↔ HMI.
VERSION	Bezeichner für eine Version

Funktionsausführung negativ quittieren

Das System bietet mit der automatisch bereitgestellten Variablen "$actionresult" die Möglichkeit, dem XML-Parser ein negatives Ausführungsergebnis mitzuteilen. Wird der Wert auf Null gesetzt, bricht der Parser die Funktionsbearbeitung ab.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE AGM>
<AGM>                              Bezeichner für "IBN Assistent"
<DEVICE>
  <NAME> Device 1 </NAME>          Bezeichner für das Gerät
  <START_UP>                       Bezeichner zur Inbetriebnahme des Geräts
  …
  </START_UP>
  <SET_ACTIVE>                     Bezeichner zum Aktivieren des Geräts
  …
  </SET_ACTIVE>
  <SET_INACTIVE>                   Bezeichner zum Deaktivieren des Geräts
  …
  </SET_INACTIVE>
```

```
    <TEST>                          Bezeichner zum Testen des Geräts
    …
    </TEST>
  </DEVICE>
  …
  </AGM>
```

4.6.1 CONTROL_RESET

Beschreibung

Dieser Bezeichner erlaubt es, eine oder mehrere Steuerungskomponenten neu zu starten. Die Abarbeitung des Skripts wird erst fortgesetzt, wenn die Steuerung den zyklischen Betrieb wieder aufgenommen hat.

Programmierung

Bezeichner:	**CONTROL_RESET**	
Syntax:	<CONTROL_RESET resetnc="TRUE" />	
Attribute:	resetnc="true"	Die NC-Komponente wird neu gestartet.
	resetdrive="true"	Die Antriebskomponenten werden neu gestartet.

4.6.2 FILE

Beschreibung

Der Bezeichner ermöglicht das Einlesen oder Erstellen von Standardarchiven.

- Archiv einlesen:
  Zum Einlesen eines Archivs ist der Dateiname des Archivs anzugeben.
- Archiv erstellen:
  Ist das Attribut create= "true" angegeben, erstellt die Funktion ein Standardarchiv (*.arc) unter dem angegebenen Namen und legt die Datei im Verzeichnis .../dvm/archives ab.

Programmierung

Bezeichner:	**FILE**	
Syntax:	<file name ="<Archivname>" />	
	<file name ="<Archivname>" create="true" class="<Datenklassen>" group="<Bereich>" />	
Attribute:	name	Bezeichner für den Dateinamen

create	Es wird ein Inbetriebnahmearchiv unter dem angegebenen Namen im Verzeichnis …/dvm/archives/ angelegt.
group	Spezifiziert die Datengruppen, die im Archiv enthalten sein sollen. Sollen mehrere Datengruppen gesichert werden, sind die Gruppen durch ein Leerzeichen getrennt anzugeben. Folgende Datengruppen können im Archiv enthalten sein: • NC • PLC • HMI • DRIVES

Beispiel

```
<!-- Datenklassenarchiv erstellen -->
<file name="user.arc" create="true"
group="nc plc hmi" />

<!—Archiv in die Steuerung einlesen->
<file name="user.arc" />
```

4.6.3 OPTION_MD

Beschreibung

Der Bezeichner erlaubt das Umdefinieren eines Options-Maschinendatums. Im Auslieferungszustand verwendet das System die MD14510 $MN_USER_DATA_INT[0] bis $MN_USER_DATA_INT[3].

Verwaltet das PLC-Anwenderprogramm die Optionen, sind die entsprechenden Datenwörter in einem Datenbaustein oder GUD bereitzustellen.

Das Datum ist bitweise organisiert. Beginnend mit Bit 0 erfolgt eine feste Zuordnung der Bits zur Auflistungsreihenfolge der Geräte, d.h. Bit 0 ist dem Gerät 1 zugeordnet, Bit 1 dem Gerät 2 usw. Sollen mehr als 16 Geräte verwaltet werden, erfolgt die Zuweisung der Adressbezeichner der Gerätegruppen 1-3 über den Bereichsindex.

Hinweis

Wertebereich umrechnen

Der Wertebereich des MD14510 $MN_USER_DATA_INT[i] ist von -32768 bis +32767. Um die Geräte bitweise über den Maschinendaten-Dialog freischalten zu können, ist eine Umrechnung der Bitkombination in die Dezimaldarstellung notwendig.

Programmierung

Bezeichner:	**OPTION_MD**	
Syntax:	Bereich 0:	
	<option_md name = "Adressbezeichner des Datums" />	
	ODER:	
	<option_md name = "Adressbezeichner des Datums" index= "0"/>	
	Bereich 1 bis 3:	
	<option_md name = "Adressbezeichner des Datums" index= "Bereichsindex"/>	
Attribute:	name	Bezeichner für die Adresse, z. B. $MN_USER_DATA_INT[0]
	index	Bezeichner für den Bereichsindex: 0 (Voreinstellung): Gerät 1 bis 16 1: Gerät 17 bis 32 2: Gerät 33 bis 48 3: Gerät 49 bis 64

4.6.4 PLC_INTERFACE

Beschreibung

Der Bezeichner erlaubt das Umdefinieren der PLC ↔ HMI-Schnittstelle. Vom System werden 128 adressierbare Wörter erwartet.

Voreinstellung: DB9905

Programmierung

Bezeichner:	**PLC_INTERFACE**	
Syntax:	<plc_interface name = "Adressbezeichner des Datums" />	
Attribute:	name	Bezeichner für die Adresse, z. B. "plc/mb170"

Beispiel: `plc/mb170`

4.6.5 POWER_OFF

Beschreibung

Bezeichner für eine Meldung, die den Bediener zum Ausschalten der Maschine auffordert. Der Meldetext ist fest im System hinterlegt.

Programmierung

Bezeichner:	**POWER_OFF**
Syntax:	<power_off />
Attribute:	--

4.6.6 WAITING

Beschreibung

Nach einem Reset der NC oder des Antriebs wird auf den Wiederanlauf der jeweiligen Komponente gewartet.

Programmierung

Bezeichner:	**WAITING**	
Syntax:	<WAITING WAITINGFORNC ="TRUE" />	
Attribute:	waitingfornc="true"	Es wird auf den Wiederanlauf der NC gewartet.
	waitingfordrive="true"	Es wird auf den Wiederanlauf des Antriebs gewartet.

4.6.7 XML Bezeichner für den Dialog

Dialog zur Parametrierung

Um zusätzliche Parameter zur Laufzeit zu setzen oder auszugeben, kann für jedes Gerät ein Dialog projektiert werden. Dieser wird mit dem Drücken des Softkeys "Zusätzliche Parameter" angezeigt.

Für die Dialogerstellung stehen alle im Kapitel Anwenderdialoge erstellen (Seite 27) beschriebenen Skriptelemente zur Verfügung.

Beispiel

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE AGM>
<AGM>
<DEVICE>
  <NAME> Device 1 </NAME>
  <START_UP>
  …
  </START_UP>
  <SET_ACTIVE>
  …
  </SET_ACTIVE>
…
  <FORM name="<dialog name>">          Bezeichner für einen Anwender-Dialog
    <INIT>
      <CONTROL name = "edit1" .../>    Bezeichner für ein Eingabefeld
    </INIT>
    <PAINT>                            Bezeichner für Text- oder
      <TEXT>hello world !</TEXT>       Bildanzeige
    </PAINT>
  </FORM>

</DEVICE>
…
</AGM>
```

4.6.8 SOFTKEY_OK, SOFTKEY_CANCEL

Beschreibung

Der Bezeichner SOFTKEY_OK überschreibt das Standardverhalten beim Schließen eines Dialogs mittels Softkey "OK". Der Bezeichner SOFTKEY_CANCEL überschreibt das Standardverhalten beim Schließen eines Dialogs mittels Softkey "CANCEL".

Innerhalb dieser Bezeichner können folgende Funktionen ausgeführt werden:

- Datenmanipulationen
- Bedingte Bearbeitung
- Schleifenbearbeitung

Programmierung

Bezeichner:	**SOFTKEY_OK**
Syntax:	<SOFTKEY_OK>
	...
	</SOFTKEY_OK>
Bezeichner:	**SOFTKEY_CANCEL**
Syntax:	<SOFTKEY_CANCEL>
	...
	</SOFTKEY_CANCEL>

Index