



Handbuch

OMS-FormDirector 3.2

Rechtliche Hinweise

Der Inhalt dieses Handbuches ist das geistige Eigentum der ProfiForms Projekt GmbH. Bei der Erstellung der Texte und Abbildungen dieses Handbuches wurde mit größter Sorgfalt vorgegangen. Trotzdem können Fehler nicht vollständig ausgeschlossen werden. Die ProfiForms Projekt GmbH übernimmt keinerlei Gewähr für die Aktualität, Korrektheit und Vollständigkeit der bereitgestellten Informationen.

Die ProfiForms Projekt GmbH behält sich das Recht vor, den Inhalt dieses Handbuches ohne vorherige Ankündigung zu verändern oder ergänzen und übernimmt keine Haftung für Fehler in diesem Handbuch oder daraus resultierende mögliche Schäden. Für Verbesserungsvorschläge und Hinweise auf Fehler sind Herausgeber und Autoren dankbar.

SAP, SAP R/3, SAPScript sind eingetragene Warenzeichen der SAP AG, Walldorf.

Java ist eine eingetragene Marke der Sun Microsystems, Inc..

Adobe, Adobe Present, Adobe Central, Adobe Designer, XFT, XDP, PostScript, PDF und weitere Warenzeichen sind eingetragene Warenzeichen der Adobe Systems Incorporated.

Unix ist ein Warenzeichen der Open Group.

Windows ist ein eingetragenes Warenzeichen der Microsoft Corporation.

Alle anderen Firmennamen und Produktbezeichnungen sind Warenzeichen oder eingetragene Warenzeichen der jeweiligen Firmen und unterliegen im Allgemeinen warenzeichen-, marken- und/oder patentrechtlichem Schutz.

Inhalt

DIE AUFGABE DES FORMDIRECTORS.....	4
PLATZIERUNG.....	4
ZIEL	5
AUFRUFKONVENTIONEN	6
DIE FUNKTIONSWEISE DES FORMDIRECTORS	7
DIE CVB-OBJEKTE, MIT DENEN DER FORMDIRECTOR ARBEITET	8
DIE ELEMENTE, AUS DENEN SICH DIE CVB-OBJEKTE ZUSAMMENSETZEN	15
SYNTAX DER CVB-OBJEKTE.....	19
SYSTEM	20
JOB	25
FORM	29
PAGE.....	34
RECOGNITION.....	39
CONVERSION	44
DYNBLOCK.....	68
BLOCK	71
DESIGN.....	74
SUBSTITUTE	77
VAREXEC.....	82
COMMANDORVALUE	96
KONVERTIERUNGSTABELLE.....	101
INDEX	104

Die Aufgabe des FormDirectors

Platzierung

Der FormDirector ist ein ergänzendes Werkzeug für den OMS-ReportWriter oder das Adobe Present (JetForm). Er nimmt plain-ASCII-Druckdaten oder mit Steuerzeichen durchsetzte Daten entgegen, analysiert die Daten, wandelt sie nach Anweisungen aus einer Steuerdatei (ConversionBase), reichert sie mit Steuerbefehlen an und sendet die Daten zum Druck an den OMS-ReportWriter oder das Adobe Present/Central. Der FormDirector wird dazu als eine Task/Programm vom OMS-Spooler/Adobe Present/Central aufgerufen und verhält sich wie die übrigen Module.

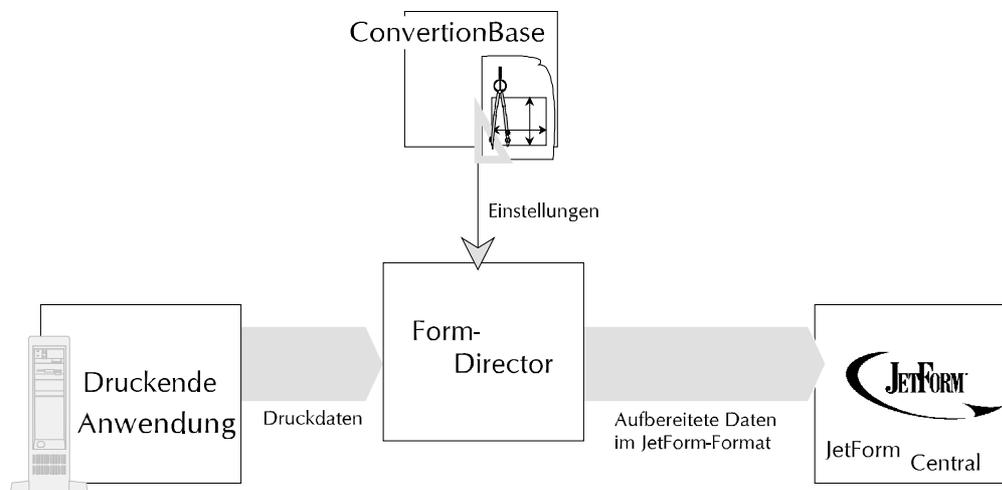


Bild: der FormDirector in seinem Arbeitsumfeld

Ziel

Mit dem FormDirector werden weiterhin folgende Ziele verfolgt:

- Druckdaten analysieren und differenziert behandeln
- den Anwender von der Programmierung der Druckausgabe in der Anwendung befreien
- die weitreichende Funktionalität von OMS-ReportWriter bzw. Adobe Present erschließen, ohne programmiertechnische Erfahrung in Adobe Present vorauszusetzen
- eine Definitionsumgebung bieten, die mit Begriffen und Kategorien arbeitet, die dem Anwender vertraut sind
- eine Logik zur Verarbeitung der Druckdaten implementieren, die sich im druckenden Anwendungsprogramm nur mit großem Aufwand nachpflegen ließe (z.B. Umsortieren der Druckdaten).

Mit anderen Worten: der FormDirector erweitert Ihre Möglichkeiten, das Aussehen Ihrer Drucke nach Ihren Wünschen zu gestalten.

Aufrufkonventionen

Der FormDir kann zwei Parameter und eine Liste mit Optionen als Übergabeparameter verarbeiten:

Syntax: formdir InputFile [CVB-File] [OptionenListe]

InputFile InputFile bezeichnet den Namen der zu verarbeitenden Datei.

CVB-File Der CVB-File ist ein optionaler Parameter und gibt den Namen der zu verwendenden CVB-Datei an. Fehlt dieser Parameter, so wird standardmäßig die Datei formdir.cvb geladen.

OptionenListe Die OptionenListe enthält einzelne Optionen, die mit dem Zeichen '-' beginnen.

-aapPATH Pfad zum Archiv-System.

-aipPATH Pfad zu den Ini-Dateien.

-acpPATH Pfad zu Config-Dateien wie CVB-Files, Variablen-Files und Substituten-Files.

-aopPATH Output Pfad

-s Unterdrücken aller Ausschriften auf cout

-v Bei der Angabe von -v erfolgt nur die Ausgabe der Versionsnummer auf den Bildschirm. Danach beendet der ReportWriter seine Arbeit mit dem Returncode 0.

-allLogFile Der FileName wird zum Logging aller Aktivitäten verwendet.

-orw Bei der Angabe von -orw (OutputFormat ReportWriter) kann der FormDirector angewiesen werden. Seine Daten für den ReportWriter zu generieren. Dabei werden alle Seiten zu ^positions und alle Blöcke zu ^subpositions.

-snoSERNO Angabe der Seriennummer. Bei Angabe der Seriennummer in der Kommandozeile wird die Seriennummer in der cvb ignoriert.

-advN=C Definiert zusätzliche Variablen über die Kommandozeile. Dabei ist N der Name und C der Content. Der Kommandozeilenparameter kann mehrfach für unterschiedliche Variablen benutzt werden. Vor jedem Lauf werden die Variablen an die globale Variablen-Tabelle angehängen.

Die Funktionsweise des FormDirectors

Der FormDirector verarbeitet zur Laufzeit der Task die Druckdaten nach den Vorgaben, die der Nutzer vorab getroffen hat. Diese Vorgaben werden in einer Datei, der **ConversionBase** (*.cvb) abgelegt. Die Vorgaben können mit einem einfachen Editor erfasst werden.

Sämtliche Vorgaben erfolgen vermittelt von CVB-Objekten, die sich ihrerseits aus einigen Elementen zusammensetzen. Die CVB-Objekte und Elemente sind weitestgehend der Denkweise des Nutzers in Bezug auf die Druckausgabe entlehnt. D.h., der Datenstrom und seine Verarbeitung werden mit den CVB-Objekten und Elementen abgebildet.

Der FormDirector verarbeitet die hereinkommenden Druckdaten in zwei Schritten:

1. Identifizieren der Daten,
2. festgelegte Verarbeitung der erkannten Daten

Beim Identifizieren der Daten zerlegt er den Datenstrom baumartig in immer kleinere Teile, die logische Einheiten bilden. Dabei werden die Daten CVB-Objekten zugewiesen. Z.B. wird eine Seite hereinkommender Daten als Rechnung erkannt und wird so zu einem *FORM-OBJEKT* mit Namen Rechnung.

Beim Zerlegen der Daten in immer kleinere Teile werden diese Teile immer weiter spezifiziert.

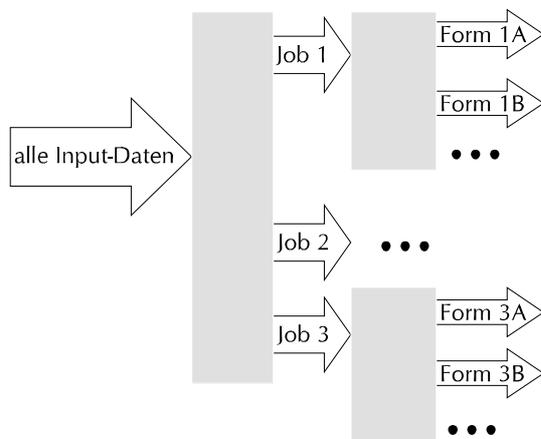


Bild: Zerlegen der Input-Daten

Die erkannten (zerlegten) Daten werden entsprechend den hinterlegten Anweisungen verarbeitet und für die Ausgabe zusammengestellt.

Folgende Festlegung in der ConversionBase wäre denkbar: sowohl für „Job1“ als auch für „Job3“ kann ein-und-dasselbe *FORM-OBJEKT* mit Namen „Rechnung“ definiert werden. „Rechnung“ wird in beiden Fällen unterschiedliche Daten repräsentieren, weil für die Erkennung der Daten sowohl die Merkmale des *JOB-OBJEKTES* als auch von dem diesem *JOB-OBJEKT* untergeordneten *FORM-OBJEKT* maßgeblich sind. Die anschließende Verarbeitung der Daten ist aber in beiden Fällen gleich - es wird die gleiche Verarbeitungsvorschrift für Daten verwendet, die einige identische Merkmale haben (in „Rechnung“ definiert) und einige unterschiedliche Merkmale aufweisen (in „Job1“ und „Job3“ definiert). Bitte durchdenken Sie den eben geschilderten Sachverhalt. Er ist wichtig für das Verständnis der Arbeitsweise des FormDirectors und zeigt viel von der Effektivität, die sich mit dem FormDirector erreichen läßt.

Das verwendete Baustein-Konzept bietet u.a. die Möglichkeit, CVB-Objekte so zu definieren, dass sie an ganz verschiedenen Stellen wiederverwendet werden können. Die Zusammenstellung einer Fußzeile aus dynamischen Druckdaten, die für alle Formulare gleich ist, muss beispielsweise nur einmal definiert werden und kann in allen Formularen verwendet werden. Das beschleunigt nicht nur den Entwurf, sondern minimiert auch den Änderungsaufwand.

Die CVB-Objekte, mit denen der FormDirector arbeitet

Im folgenden sollen die CVB-Objekte und Elemente des FormDirector einzeln vorgestellt werden. Es geht um die grundlegenden Aufgaben, die jeder der Bausteine erfüllt und in welcher Beziehung sie zueinander stehen. Diese Kapitel will Ihnen ein „Gefühl“ dafür vermitteln, wie der FormDirector arbeitet.

Genauer gesagt handelt es sich bei den Bausteinen nicht um CVB-Objekte und Elemente, sondern abstrahiert um Objekttypen und Elementtypen, welche durch ihre unterschiedlichen Ausprägungen in der ConversionBase zu CVB-Objekten und Elementen modifiziert werden. Dazu werden u.a. in den CVB-Objekten, Elemente eines bestimmten Elementtyps verwendet. Nicht in jedem Objekttyp dürfen alle Elemente vorkommen. Es ist daher wichtig, sich einen Überblick zu verschaffen, was die Aufgabe jedes Objekttyps ist; das läßt Schlüsse auf die zu verwendenden Elemente zu. In aller Regel werden die CVB-Objekte und Elemente durch Namen gekennzeichnet. Ist die Position und Verwendung eines CVB-Objektes/Elementes eindeutig, wird auf den Namen verzichtet. Beispiele hierfür sind das *SYSTEM-OBJEKT* und die jeweiligen Elemente vom Typ *RECOGNITION*.

Die CVB-Objekte in der ConversionBase haben zweierlei Aufgaben:

1. die Eigenschaften von Daten beschreiben und die Daten zerlegen und
2. Angaben zu Veränderung der Daten und Ausgabeanweisungen festlegen.

In den einzelnen Objekttypen wird vorrangig meist die eine oder andere Aufgabe gelöst, es lassen sich aber immer Anteile von beiden Aufgaben finden.

CVB-Objekte, die eher die erste Aufgabe wahrnehmen, sind selber keine Druckdaten. Sie beschreiben nur die Druckdaten, besonders, woran man sie erkennt. Aufgrund dieser Beschreibung wird der FormDirector zur Laufzeit die Daten individuell verarbeiten.

Die Daten werden auf zweierlei Weise verarbeitet: Es gibt zum einen die Manipulation der Daten an sich. Das geschieht vermittelt der → *COMMANDS* und ausschließlich innerhalb des → *CONVERSION-OBJEKTES*. Zum anderen fließen in die Ausgabe Informationen mit ein, die beim konventionellen Formulareindruck sozusagen „im Formular stecken“. Das sind Informationen, wie die Anzahl der Durchschläge, in welchen Drucker das Papier eingespannt ist usw. Diese Informationen werden (im Regelfall) nicht aus dem Datenstrom gewonnen, sondern per → *VALUE* für die Formulare festgelegt.

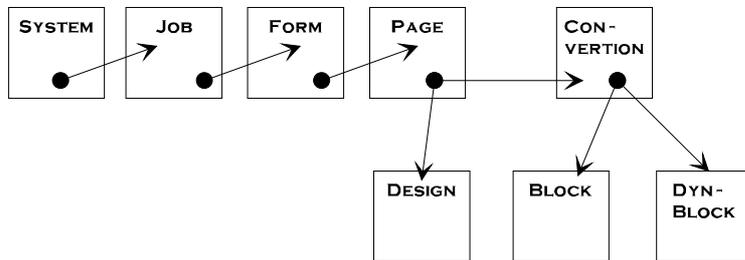
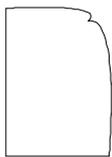


Bild: Hierarchie der Objekttypen

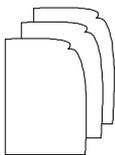
Die Objekttypen der ConversionBase stehen in einer hierarchischen Beziehung zueinander. In ein übergeordnetes CVB-Objekt können per → *OBJEKTLIST*, → *CHOICELIST* oder → *LINK* beliebig viele oder ein untergeordnetes CVB-Objekt eingebunden werden. Die Hierarchie der CVB-Objekte entspricht einer immer detaillierteren Erkennung im

ersten Teil und einer immer detaillierteren Erkennung und Verarbeitung im zweiten Teil der Datenbehandlung.



Ein zentraler Teil unserer Betrachtungen ist das Formular. Vom Formular aus starten die wichtigen Wandlungen der Daten. Aus der Sicht des druckenden Systems hat ein Formular genau eine Seite Daten. Die werden in einen Vordruck (möglicherweise einen Durchschreibesatz) eingefüllt. Ebenso versteht der FormDirector unter einem Formular eine Seite Input-Daten.

Ein Formular wird in dem *FORM-OBJEKT* abgebildet. Hier wird festgelegt, an welchen Merkmalen im Datenstrom das jeweilige Formular erkannt wird. Gleichzeitig werden den Daten Merkmale zugewiesen, die das Ausgabejobhandling und das physische Formular, nicht aber die Daten selber betreffen.



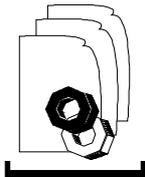
Das *JOB-OBJEKT* lehnt sich an das klassische Job-Verständnis an: nämlich, dass die Daten eines Jobs gemeinsam zum Druck aufgearbeitet werden und dass diese Daten häufig mehrere gemeinsame Eigenschaften haben. Ein *JOB-OBJEKT* repräsentiert zusammenhängende hereinkommende Druckdaten, die mindestens eine gemeinsame Eigenschaft haben. Jedes *JOB-OBJEKT* enthält auch Angaben, woran dieser Job von anderen Jobs im →

SYSTEM-OBJEKTE unterschieden werden kann. Das ganze geschieht mit dem Ziel, Informationen, die nur am Jobanfang mitgeliefert werden und für die spätere Formularerkennung nötig oder nützlich sind, auszuwerten. Für die verschiedenen Jobs kann festgelegt werden, welche *FORM-OBJEKTE* innerhalb jedes Jobs vorkommen (also identifiziert werden) können. Damit entsteht für die Formulare eine kaskadierte Erkennung.

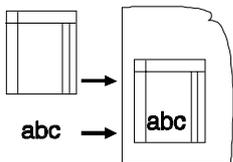
Bei guter Strukturierung können die Jobs, die Ihr druckendes System erzeugt und die Jobs, die Sie innerhalb des FormDirectors definieren, identisch sein. Anderenfalls wird der Job, den Sie zum Druck geschickt haben vom FormDirector als eine Anzahl dicht hintereinander eintreffender Einzeljobs aufgefasst. Wenn wir hier von Job sprechen und nicht explizit etwas anderes angeben, meinen wir immer den Job, den der FormDirector Hereinkommenderweise als Job identifiziert hat.

Das dem FormDirector nachgeschaltete Adobe Present/ Central erwartet eingangsseitig natürlich auch einen Job. Da dieser bei der Datenübergabe in Dateiform immer nur aus einem File bestehen darf, wollen wir bei diesen Jobs von SpoolFiles sprechen. Ein SpoolFile ist eine Menge zusammenhängender Daten, die der FormDirector ausgibt.

Ein Job ist eine Anzahl von Formularen, die gemeinsam zum Druck angestoßen werden. Im *JOB-OBJEKT* wird festgelegt, welche Formulare innerhalb eines Jobs auftreten können und woran der Job von anderen Jobs unterschieden werden kann. Der FormDirector nutzt die Jobunterscheidung, um Informationen, die nur am Jobanfang mitgeliefert werden und für die Formularerkennung nötig oder nützlich sind, auszuwerten. Damit entsteht für die Formulare eine kaskadierte Erkennung.



So, wie der Job eine Menge von Formularen zusammenfasst, fasst das *SYSTEM-OBJEKT* alle Jobs zusammen, die über den FormDirector gedruckt werden. D.h., das *SYSTEM-OBJEKT* ist die gemeinsame Wurzel aller Daten, die je den FormDirector durchlaufen. Hier wird das grundlegende Verhalten des FormDirectors festgelegt.



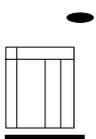
Die größte Menge Daten, für die Anweisungen festgelegt werden können, ist das *FORMULAR-OBJEKT*. Die zum *FORM-OBJEKT* gehörenden Definitionsobjekte heißen *PAGE-OBJEKT*. Ein *PAGE-OBJEKT* repräsentiert jeweils die Anweisungen für einen Durchschlag. Es werden die unmittelbar zur Kopie gehörenden Eigenschaften wie Duplexdruck, In tray, Out tray usw. festgelegt. Außerdem wird für jeden Durchschlag getrennt festgelegt, welches →

CONVERSION-OBJEKT auf die Daten angewandt werden soll. D.h., die Eingangsdaten können für jeden Durchschlag auf eine andere Weise bearbeitet werden.

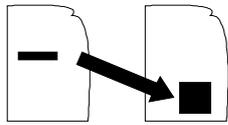
Damit können beispielsweise ein Lieferschein und eine Rechnung aus einem Datenstrom erzeugt werden, weil für die Kopie „Lieferschein“ die Preise ausgeblendet werden können. Ein anderer Anwendungsfall ist das automatische Erzeugen eines Adressaufklebers aus dem Adresskopf eines Formulars.

Weiter wird im *PAGE-OBJEKT* festgelegt, welches → *DESIGN-OBJEKT* eingebunden werden soll. D.h., jeder Durchschlag kann ein ganz individuelles Layout erhalten.

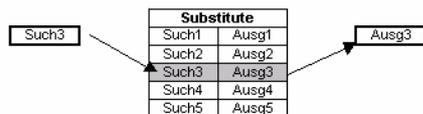
Da das endgültige Aussehen der Ausgabeseite vom verwendeten Formulardesign und der Position und Größe der verwendeten Felder abhängt, wird für die Ausgabeseite keine Ausdehnung in Zeilen und Spalten festgelegt.



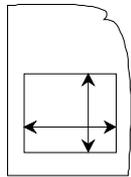
Das *DESIGN-OBJEKT* enthält eine Anzahl von Layout-Bausteinen, aus denen sich das Seitenlayout zusammensetzt.



Die *CONVERSION-OBJEKT* ist dasjenige Definitionsobjekt, in dem die Einwirkungen festgelegt werden, welche die Daten unmittelbar betreffen. Das *FORM-OBJEKT* repräsentiert eine Seite Daten (virtuelle Input-Page), die *PAGE-OBJEKTE* das Aussehen der einzelnen Ausgabeseiten. Das *CONVERSION-OBJEKT* vermittelt zwischen beiden. Es nimmt die Daten vom Eingangsdatenstrom, wandelt sie ggf. und bringt sie an die richtige Stelle im Ausgabestrom. Damit ist das *CONVERSION-OBJEKT* das entscheidende Glied bei der Überführung des Eingangsstroms in den Ausgabestrom. Daten, die nicht in einem *CONVERSION-OBJEKT* definitiv vom Eingangsstrom in den Ausgabestrom gestellt werden, werden am Ende der Aufarbeitung verworfen. Das *CONVERSION-OBJEKT* kann mittels einzelner → *COMMANDS* die Daten der gesamten Seite behandeln. Sie kann aber auch kleinere Datenmengen spezifizieren: → *DYNBLOCK-OBJEKT* und → *BLOCK-OBJEKT*. Für diese kleineren Datenmengen können wiederum *CONVERSION-OBJEKTE* vereinbart werden, die nur auf diese Daten angewandt werden. Damit lassen sich beliebig tiefe Verschachtelungen aufbauen (Siehe Bild „Verschachtelungen in dem *CONVERSION-OBJEKT*“). Das *CONVERSION-OBJEKT*, das in das *PAGE-OBJEKT* eingebunden wird, bleibt aber die Wurzel aller Druckdatenmanipulationen. Da mit den Verschachtelungen die Daten immer genauer spezifiziert werden sollen, sind iterative Schleifen nicht sinnvoll. Sie werden mit einer Fehlermeldung abgefangen.



Das *SUBSTITUTE-OBJEKT* ist ein Definitionsobjekt, in dem über eine Tabelle Fallunterscheidungen angeboten werden, die nicht gleich im *JOB-* oder *PAGE-OBJEKT* getroffen werden müssen, sondern z.B. erst im *PAGE-OBJEKT* wo die Auswahl endgültig wirksam wird. Damit ist es möglich mit einem Programmabschnitt, der mehrfach mit unterschiedlichen Parametern abgearbeitet wird, eine effizienten Programmierung zu garantieren. Das Herzstück des FormDirectors ist das Zusammenspiel von *FORM-*, *PAGE-* und *CONVERSION-OBJEKT*. Deshalb soll an dieser Stelle nochmals auf diese drei eingegangen werden. Alle Schritte, die vor *FORM*, *PAGE* und *CONVERSION* liegen, dienen der gezielten Aufspaltung der Daten, alles was danach kommt, ist in dem ersten *CONVERSION-OBJEKT* eingeklinkt und dient der verfeinerten Verarbeitung. Das endgültige, gedruckte Formular entsteht aus drei Teilen: den Nettodaten, den Designteilen und den Anweisungen zur Manipulation der Daten für den Druck. Dem *FORM-OBJEKT*, das die Nettodaten repräsentiert, wird eine Anzahl von *PAGE-OBJEKTEN* zugeordnet, die ihrerseits die einzelnen Seiten des auszugebenden Formulars darstellen. Die *PAGE-OBJEKTE* beinhaltet sowohl das *DESIGN-OBJEKT*, welches das Layout repräsentiert, als auch das *CONVERSION-OBJEKT*, die für alle Datenmanipulationen zuständig ist.



Ein *FORM-OBJEKT* repräsentiert eine Seite Nettodaten. Ein *BLOCK-OBJEKT* steht für einen Ausschnitt aus diesen Daten. Die Größe des Ausschnittes (Blocks) wird durch seine Zeilen- und Spaltenzahl festgelegt. Die Position des *BLOCK-OBJEKTES* auf der Seite liegt jedoch nicht fest.. Für die Daten, die das Objekt repräsentiert, können Verarbeitungsanweisungen festgelegt werden. Innerhalb der Blocks gilt eine eigene relative Positionierung, deren Bezugspunkt in der

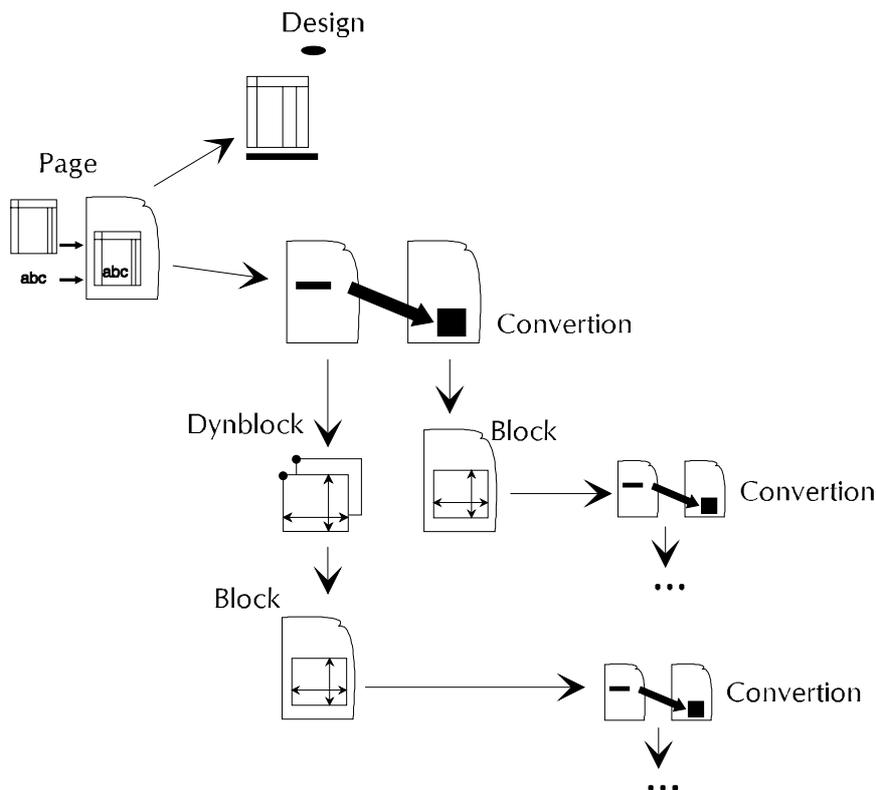
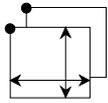


Bild: Verschachtelungen in dem *CONVERSION-OBJEKT*

linken oberen Ecke liegt. Mit einem *BLOCK-OBJEKT* werden Daten beschrieben, die immer auf dieselbe Weise bearbeitet werden sollen, die aber nicht immer an derselben Stelle auf der Seite stehen. Aufgerufen wird ein *BLOCK-OBJEKT* entweder per *LINK* oder dadurch, dass auf der Seite nach einer Erkennung für die Daten gesucht wird. Im ersten Fall muss beim Aufruf eine Position mit übergeben werden. Im zweiten Fall wird das *BLOCK-OBJEKT* an der Stelle auf der Seite positioniert, wo die Daten identifiziert wurden. Dieser Fall ist für die Verwendung von *BLOCK-OBJEKTEN* im Zusammenhang mit *DYNBLOCK-OBJEKTEN* gedacht. Wird ein *BLOCK-OBJEKT* mit einer ‚eingebauten‘ Erkennung an anderer Stelle außerhalb des *DYNBLOCK*-Kontextes aufgerufen, bleibt die Erkennung unbeachtet.



Ein *DYNBLOCK-OBJEKT* ist ein Objekt, das eine Reihe von *BLOCK-OBJEKTEN* zusammenfasst. Er findet überall dort Anwendung, wo Daten innerhalb eines bestimmten Fensters auf der Seite identifiziert und anschließend verarbeitet werden sollen. Dazu wird ein Suchfenster über einem Bereich der Seite

aufgespannt. Innerhalb des Fensters wird zeilenweise geprüft, ob es sich bei den Daten um zum *DYNBLOCK-OBJEKT* gehörende *BLOCK-OBJEKTE* handelt. Jeder der *BLOCK-OBJEKTE* trägt dazu eine eigene → *RECOGNITION*. War die Suche erfolgreich, werden die Daten wie im *BLOCK-OBJEKT* angegeben verarbeitet.

Man könnte ein *DYNBLOCK-OBJEKT* beispielsweise dafür verwenden, um zwischen die einzelnen Positionen einer Rechnung Leerzeilen zur besseren Übersichtlichkeit einzufügen.

Für die Arbeit mit sich dynamisch aufbauenden Tabellen können im *DYNBLOCK-OBJEKT* zwei *DESIGN-OBJEKTE* (als Kopf und Fuß) vereinbart werden. Diese *DESIGN-OBJEKTE* werden unabhängig von den erkannten *BLOCK-OBJEKTEN* immer eingebunden.

Das Bild „Zusammenspiel der CVB-Objekte, 1“ zeigt die Hierarchie der CVB-Objekte im Überblick. Weiterhin ist zu sehen, dass Mithilfe der ersten drei Objektdefinitionen der Datenstrom bis auf Seitengröße zerlegt wird (Zerlegen der Eingangsdaten). In den nächsten Schritten wird aus den Daten, den zugehörigen Formulardesigns und den allgemeinen Anweisungen für die Ausgabe der Ausgabedatenstrom zusammengestellt. Das Zusammenstellen der Ausgabedaten bringt je nach Erfordernissen einen oder mehrere Jobs hervor.

Für einfache Anwendungen reicht das in das *FORM-OBJEKT* eingebundene *CONVERSION-OBJEKT* aus. Für komplexe Anwendungen können in dem *CONVERSION-OBJEKT* neben den *COMMANDS* auch *BLOCK-* und *DYNBLOCK-OBJEKTE* enthalten sein.

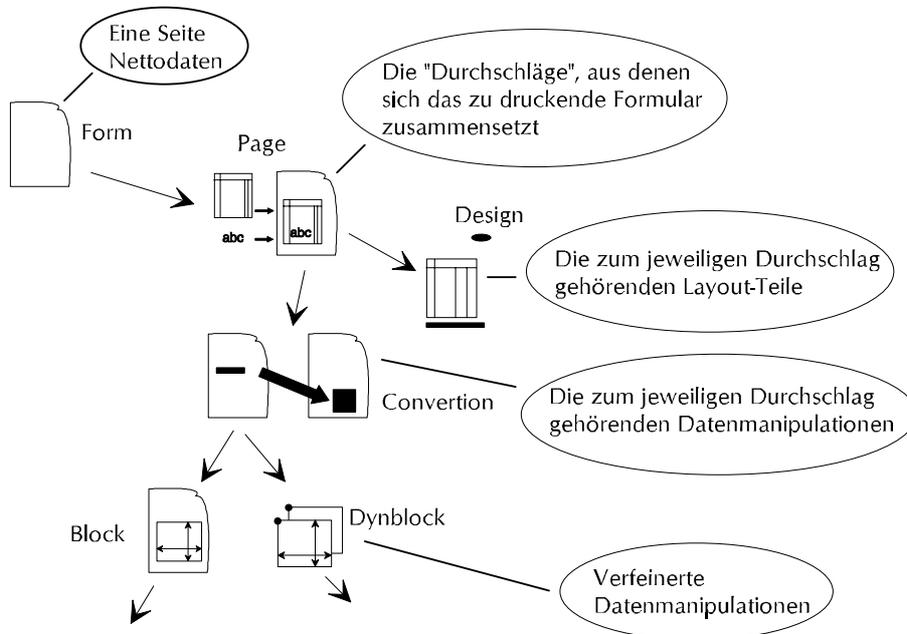
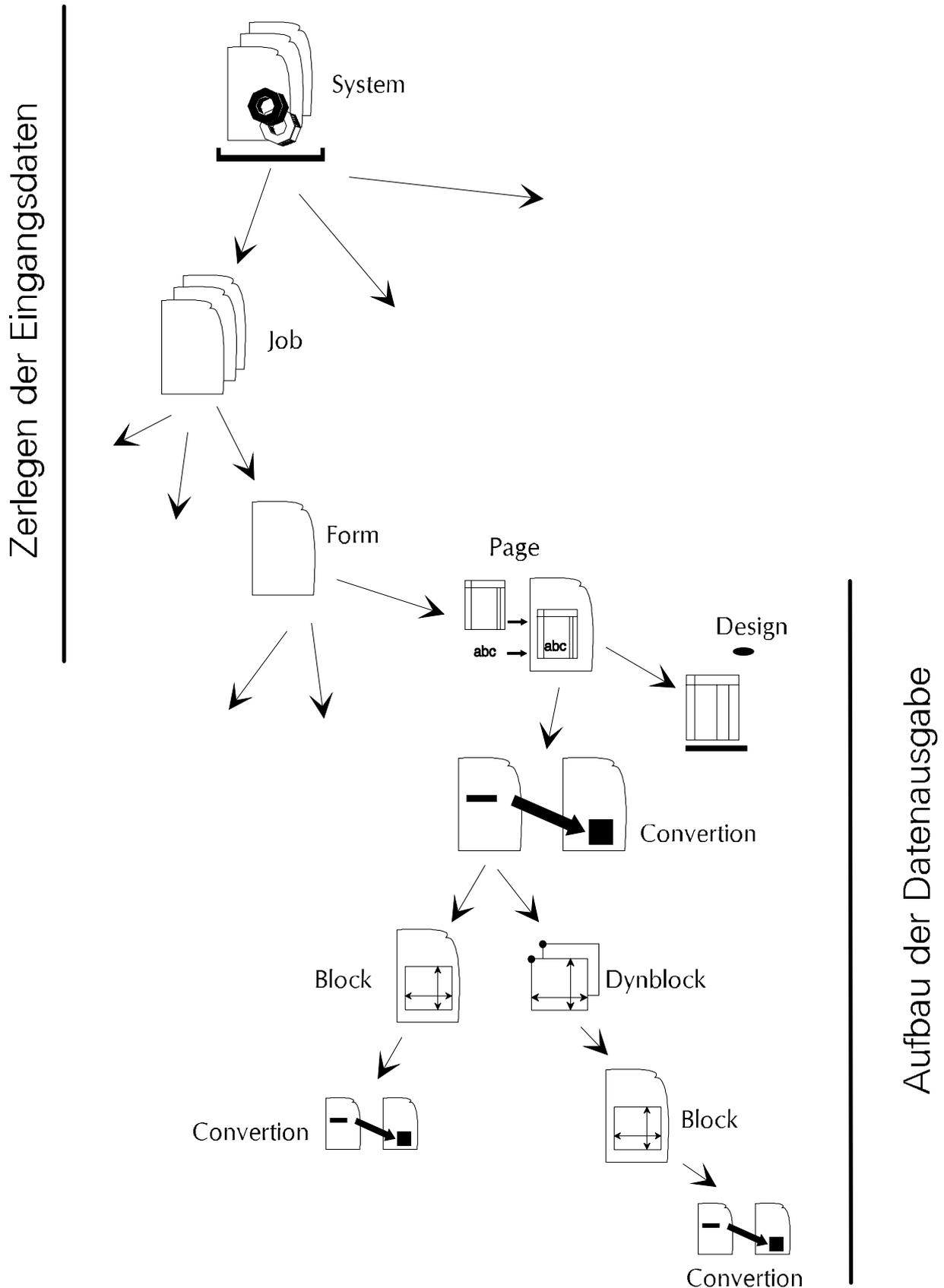


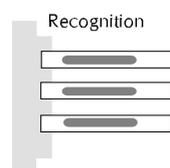
Bild: Zusammenwirken von *FORM-*, *PAGE-* und *CONVERSION-OBJEKT*



BLOCK-OBJEKTE können ihrerseits wieder *CONVERSION-OBJEKTE* enthalten. Durch die damit möglichen Verschachtelungen sind anspruchsvolle Anforderungen zu bewältigen. Der gesamte Sachverhalt ist im Bild „Zusammenspiel der CVB-Objekte, 2“ dargestellt.

Die Elemente, aus denen sich die CVB-Objekte zusammensetzen

Die CVB-Objekte der ConversionBase bauen sich aus Elementen auf. Es werden nicht alle Elemente für alle CVB-Objekte herangezogen. Ein Elementtyp steht als Synonym für ganz konkrete Elemente, z.B. *COMMAND* für alle die verschiedenen Befehle. Die Elemente haben aber gemeinsame Eigenschaften und die werden mit den Elementtypen beschrieben. Auf die Unterschiede der einzelnen Elemente wird im **Kapitel Syntax** eingegangen.

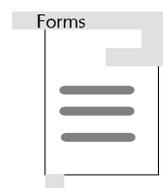


Die *RECOGNITION* dient zur Erkennung einzelner Datenobjekte. Sie ist Bestandteil des CVB-Objektes, das identifiziert werden soll. Der FormDirector untersucht in der *RECOGNITION* nur den Inhalt des Datenstroms. Seine Länge und Breite zieht er nicht heran. Lediglich im *FORM-OBJEKT* wird beim Überschreiten einer Höchstzeilenzahl ein Seitenumbruch ausgelöst. Das zieht einen Erkennungsschleife für die neu entstandene Seite nach sich.

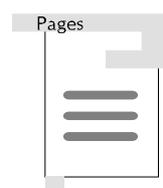
Die *RECOGNITION* enthält eine Reihe von *COMMANDS*, die auf das Vorkommen von Zahlen- und/oder Zeichenketten an einer bestimmten Position oder in einem bestimmten Bereich prüfen. Die Befehle können durch NAND-, NOR-, AND- und OR-Operatoren verknüpft werden. Für Ziffern können Wildcards angegeben werden, für Zeichen nicht. Die Wildcard-Funktionalität lässt sich jedoch immer herstellen, indem man die Operatoren nutzt.

(Wildcards drücken aus, dass an ihrer Stelle ein beliebiges Zeichen steht. Ein Beispiel ist die Verwendung von „*“ in DOS-Befehlen.)

Die *RECOGNITION* kann in den CVB-Objekten *JOB*, *FORM* und *BLOCK* vorkommen.



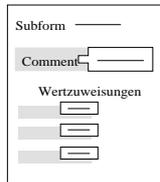
Eine *CHOICELIST* enthält eine Reihe von Verweisen auf CVB-Objekte, die an dieser Stelle eingebunden werden sollen. Die *CHOICELIST* steht für folgende Elementtypen: Jobs, Forms, und Blocks. Wie der Name jeder Liste andeutet, werden entweder *JOB*-, *FORM*-, oder *BLOCK-OBJEKTE* aufgerufen. Aus der jeweiligen Liste wird jeweils nur ein Objekt ausgewählt und eingebunden. Maßgebend ist die *RECOGNITION* der einzelnen CVB-Objekte. Die *CHOICELIST* wird dabei von oben nach unten abgearbeitet. Ein Objekt mit der Zeichenkette „XYZ“ als *RECOGNITION* muss also hinter dem Objekt stehen, das als *RECOGNITION* die Verknüpfung „XYZ“ AND „ABC“ hat, sonst wird immer das Objekt mit „XYZ“ gewählt.



Die *OBJEKTLIST* erfüllt die gleichen Aufgaben wie die ChoiceList, mit dem Unterschied, dass alle aufgeführten CVB-Objekte eingebunden werden. Daher haben die in einer *OBJEKTLIST* aufgeführten CVB-Objekte auch keine eigene *RECOGNITION*.

Der Typ Pages ist derzeit die einzige Art von Objektlisten. Die CVB-Objekte in Pages werden alle eingebunden und zwar in der aufgeführten Reihenfolge. Man kann sich das vorstellen wie die Durchschläge eines Formularsatzes.

SubForm



Es ist sinnvoll, das Aussehen eines Ausgabeblattes nicht in einem Stück zu gestalten, sondern aus „Bausteinen“ zusammensetzen. Diese „Bausteine“ nennen wir *SUBFORM*. Diese SubForms sind identisch mit den *SUBFORMS*, die sich im grafischen Designtool Adobe Present/Design erzeugen lassen. *SUBFORMS* haben keine feste Position auf der Seite, sondern bekommen ihren Platz beim Aufruf zugewiesen. Damit lassen sich beispielsweise dynamisch wachsende Tabellen mit nur einem Zeilenlayout aufbauen, das immer wieder aufgerufen wird. Oder es kann ein Designteil auf ganz unterschiedlichen Positionen

verwendet werden.

Wird ein *SUBFORM* ohne Angabe einer Position aufgerufen, wird es in die linke obere Ecke der Seite platziert.

Selbstverständlich kann man auch auf das Bausteinprinzip verzichten und eine Seite Layout aus einem Stück designen und einbinden.



Aus Gründen der Modularität werden CVB-Objekte in der ConversionBase an einer anderen Stelle definiert, als sie nachher eingebunden werden. Das erhöht auf der einen Seite die Übersichtlichkeit und erlaubt zum anderen, CVB-Objekte mehrfach zu verwenden. Das Element, mit dem das Objekt an der richtigen Stelle platziert wird, heißt *LINK*. Er besteht aus dem Typ des einzubindenden CVB-Objektes und nachfolgend dem Namen des CVB-Objektes. Per *LINK* eingebunden werden die CVB-Objekte via Element StandardJob, StandardForm, StandardBlock. *BLOCK-* und *DYNBLOCK-LINKS* enthalten zusätzlich zum Namen noch die Position bzw. Position und Größe.

Die „Standard“-*LINKS* finden sinnvollerweise im Zusammenhang mit den *CHOICELISTEN* Anwendung. Sie beschreiben, welches Objekt eingebunden werden soll, wenn kein Objekt aus der *CHOICELIST* identifiziert werden konnte oder, wenn die *CHOICELIST* nicht definiert wurde.



Je ein *COMMAND* nimmt unmittelbar auf die Daten im Input-Datenstrom Einfluss. *COMMANDS* haben entweder die Aufgabe, Daten zu analysieren oder zu manipulieren. In der Regel kommen die Daten aus dem Input-Strom und werden innerhalb desselben verändert, in ein Adobe Present/-Feld überwiesen oder in eine Variable abgespeichert. Nur Daten, die definitiv vom Input-Strom heruntergelesen und für die Ausgabe bereitgestellt wurden, erscheinen beim Ausdruck.



Um Druckdaten auf Drucksystemen korrekt auszugeben sind u.a. Papiereinzüge, Ablage-Schächte etc. anzugeben. Die dafür notwendigen Informationen kommen nicht zwangsweise aus dem Datenstrom selbst, sondern gehören sozusagen zum Wesen des gewählten Formulars. Daher müssen diese Informationen für jedes Formular festgelegt werden können. Das geschieht über Elemente vom Typ *VALUE*. Jedes dieser Elemente hat einen Namen, aus dem seine Bedeutung hervorgeht und kann einen Wert aus einer Reihe von Werten zugewiesen bekommen. Falls diese Werte im Datenstrom mitgeschickt werden sollen, können sie auch von dort gelesen und den *VALUE*-Elementen zugewiesen werden.

Ab der Version 3.0 des FormDirector besteht auch die Möglichkeit den Inhalt von Variablen oder den Rückkehrwert von Funktionen an *VALUE*-Elemente zuzuweisen.

Comment  Ein *COMMENT* ist eine Kommentarzeile.

Die nachfolgende Tabelle vermittelt einen Überblick, welche Elementtypen in welchen CVB-Objekten verwendet werden können.

VarExec  Eine *VAREXE* ist ein Element, in welchem *COMMANDS* ausgeführt werden, welche dem Setzen und Löschen von Variablen und Ausführen von Funktionen dienen.

Element- typ	 SYSTEM	 JOB	 FORM	 BLOCK	 CON- VER- TION	 PAGE	 DYN BLOCK	 DESIGN	 SUB- STI- TUTE
CHOICELIST 	Shaded	Shaded					Shaded		
OBJEKTLIST 			Shaded						
RECOGNITION 		Shaded	Shaded	Shaded					
VALUE 			Shaded	Shaded		Shaded			Shaded
LINK 	Shaded	Shaded		Shaded	Shaded		Shaded		
SUBFORM 				Shaded			Shaded	Shaded	
COMMAND 					Shaded				Shaded
COMMENT 	Shaded	Shaded	Shaded	Shaded	Shaded	Shaded	Shaded	Shaded	
VAREXEC VarExec		Shaded	Shaded			Shaded			

SUBSTITUTE	
Sub11	Auss1
Sub12	Auss2
Sub13	Auss3
Sub14	Auss4
Sub15	Auss5

Syntax der CVB-Objekte



Anmerkung: Das *SYSTEM-OBJEKT* ist das einzige, das in der ConvesonBase definiert sein muss. Alle (fast alle) anderen Angaben sind wahlfrei und hängen von Ihren Erfordernissen ab. Pflichtangaben sind als solche gekennzeichnet.. Es ist sinnvoll, die untergeordneten CVB-Objekte (im u.g. Beispiel die *FORM-OBJEKTE*) so zu benennen, dass sie dem übergeordneten Objekt (im u.g. Beispiel Job) zugeordnet werden können. Von den Elementen, aus denen sich die CVB-Objekte jeweils zusammensetzen, muss keines zwangsweise enthalten sein, d.h., Sie könnten auch ein völlig leeres Objekt definieren. Die Reihenfolge der Elemente in den CVB-Objekten ist beliebig, es empfiehlt sich aber, immer eine Reihenfolge zu verwenden, die Sie nach Ihren Bedürfnissen festlegt haben. Alle Tabulatoren werden durch Leerzeichen ersetzt.



System



Syntax:

```

System {
  Comment „Kommentar“
  Jobs {
    Name1
    Name2
    ...
  }
  StandardJob StdName
  Inline Wert
  ConvertUnderlines Wert
  ReadVarsFromFileOnStart InFileName
  WriteVarsToFileOnStop OutFileName
  ArchivePath Verzeichnis
  TaskArchive Wert
  TaskPrint Wert
  OutPath Verzeichnis
  Serno Wert
}
    
```

	<u>Parameter</u>	<u>Bedeutung</u>
System	Comment	Pflichtangabe Kommentartext Wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden.
	Jobs	Die Einträge in Jobs werden von oben nach unten abgearbeitet. Die Aufarbeitungsanweisungen des ersten Jobs, dessen <i>RECOGNITION</i> mit dem Datenstrom übereinstimmt, werden verwendet. Die aufgeführten Jobs müssen an anderer Stelle in der ConversionBase definiert sein.
	Name1, Name2	Namen der Jobs, von denen anhand der dem Job eigenen Erkennung einer ausgewählt und verwendet werden soll.

StandardJob	StdName	Name des Jobs, der aktiviert wird, wenn keiner der Jobs aus der JobList erkannt wurde oder JobList nicht definiert ist. Der als StandardJob angegebene Job ist ein beliebiger Job und kann eine eigene <i>RECOGNITION</i> besitzen. Diese bleibt jedoch für die Verwendung des Jobs als StandardJob unberücksichtigt.
Inline	Wert = 0	Der FirmDirector ignoriert die Adobe Present-Inline-Sequenzen. (Default-Wert)
	Wert = 1	Der FormDirector versteht die folgenden Adobe Present-Inline-Sequenzen: <ul style="list-style-type: none"> \b. Setzt alle nachfolgenden Buchstaben auf Attribut „fett“ \b0. Setzt alle nachfolgenden Buchstaben auf Attribut „fett aus“ \ul. Setzt alle nachfolgenden Buchstaben auf Attribut „unterstrichen“ \ul0. Setzt alle nachfolgenden Buchstaben auf Attribut „unterstrichen aus“ \i. Setzt alle nachfolgenden Buchstaben auf Attribut „kursiv“ \i0. Setzt alle nachfolgenden Buchstaben auf Attribut „kursiv aus“
ConvertUnderlines	Wert = 0	Der FormDirector wandelt die „_“-Zeichen nicht (Default-Wert)
	Wert = 1	Der FormDirector wandelt alle „_“-Zeichen in das Attribut „unterstrichen“
ReadVarsFromFileOnStart	InFileName	Der FormDirector ist in der Lage beim Start des Systems eine bestimmte Datei von der Platte zu lesen.
WriteVarsToFileOnStop	OutFileName	Der FormDirector ist in der Lage beim Herunterfahren des Systems eine Variablenliste in die Datei auszugeben.
TaskArchive TaskPrint ArchivePath	Wert	Wert 0 für nicht aktiv, Standardwert 1 aktiv
	Wert	Wert 0 für nicht aktiv, Standardwert 1 aktiv
	Verzeichnis	Wert ist veraltet. Die Definition erfolgt über den Kommandozeilenparameter –aap. Die Ausgabe der Archiv-Schnittstelle wird in das dort angegebene Verzeichnis generiert.
OutPath	Verzeichnis	Wert ist veraltet. Die Definition erfolgt über den Kommandozeilenparameter –aop. Alle Drucke werden in das dort generierte Verzeichnis generiert.
SerNo		Seriennummer. Die Seriennummer kann aber alternativ auch als Kommandozeilenparameter mitgegeben werden



Anmerkung:

Die Attribute, die ein Zeichen tragen kann, sind: fett (bold), unterstrichen (underline) und kursiv (italic). Attributierte Buchstaben behalten die tatsächliche Position auf der virtuellen Seite des FormDirectors. Etwaige

Steuersequenzen, die vor dem Buchstaben gestanden haben, um ihn auf ein bestimmtes Attribut zu setzen, verschieben den Buchstaben nicht auf der virtuellen Seite. Diese Steuersequenzen werden eliminiert und allen folgenden Buchstaben als Attribut mitgegeben. Diese Attribute hängen den Zeichen bis zur Ausgabe an Adobe Present/Central an, unabhängig ob Sie diese Zeichen vorher auf der Seite verschoben, kopiert oder andersartig manipuliert haben. Der FormDirector kennt zwei Arten des Lesens attributtierter Zeichen:

1. Inline Controls:

Im gelesenen Datenstrom befinden sich Steuerzeichen (Inline Controls), die das Ein- und Ausschalten von Attributen steuern.

2. Aus Kompatibilität zur Matrixdruckeransteuerung werden doppelt gedruckte Zeichen Fett gesetzt. Weiterhin kann mit Aktivieren des Schlüsselwortes „ConvertUnderlines,“ erreicht werden, dass alle Zeichen „_“ in des Attribut „unterstrichen“ gewandelt werden.

Die Attributierung gilt zunächst nur für die Analyse der Daten. Wenn Daten attributiert ausgegeben werden sollen, müssen in den Anweisungen im *CONVERSION-OBJEKT* entsprechende Angaben gemacht werden. (Siehe dort.)



Beispiel 1: **System** {
 Comment „es gibt drei Jobs,“
 Jobs {
 Rechnungen
 Bestellungen
 Listen
 }
 StandardJob weisses_Papier
 }



Beispiel 2: **System** {
 ...
 ReadVarsFromFileOnStart InFileName
 WriteVarsToFileOnStop OutFileName
 ...
 }

Die Variablen-Datei ist in einem Adobe Present ähnlichen Format. Es werden allerdings nur Field- und Global-Kommandos ausgewertet. Der Schreibprozess erzeugt nur Field-Kommandos. Beide Keyworte sind optional.



Beispiel 3: **System** {

```
...  
    TaskArchive 1  
    TaskPrint 0  
    ...  
}
```

Im Beispiel werden alle Daten archiviert aber nicht gedruckt.



Beispiel 4: **System** {

```
...  
    ArchivPath d:\archive\in  
    OutPath d:\jfsrvr6  
    ...  
}
```

Alle Drucke werden in das dort angegebene Verzeichnis generiert. Die Archivdaten werden in das dort angegebene Verzeichnis geschrieben.



Job



Verwendung: Beschreibt einen *JOB-OBJEKT*, besonders, welche Druckdaten er repräsentiert und welche *FORM-OBJEKTE* mit diesen Daten weiterverarbeitet werden sollen.



Syntax:

```

Job JobName {
  Comment „Kommentar“
  Recognition {
    Operator {
      Command
      Command
      ...
    }
  }
  Forms {
    FormName1
    FormName2
    ...
  }
  VarExec {
    Command
    Command
    ....
  }
  StandardForm StdFormName
}
    
```

	<u>Parameter</u>	<u>Bedeutung</u>
Job	JobName	Name des Jobs, der an dieser Stelle definiert wird; Dient dazu, sich an anderer Stelle in der ConversionBase auf diesen Job zu beziehen.
Comment	Kommentar	Kommentartext Wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden.

Recognition		Wird wegen ihrer Komplexität in einem getrennten Abschnitt behandelt. Für die <i>RECOGNITION</i> können Daten auf der gesamten Seite herangezogen werden. Das erste Zeichen auf einer Seite steht an Position 1, 1 (= 1. Zeile, 1. Spalte).
Forms	FormName1, FormName2	Namen der <i>FORM-OBJEKTE</i> , von denen anhand der dem <i>FORM-OBJEKTE</i> eigenen Erkennung eines ausgewählt und verwendet werden soll. Die Einträge in Forms werden von oben nach unten abgearbeitet, das erste Form, dessen <i>RECOGNITION</i> mit dem Datenstrom übereinstimmt, wird verwendet. Die aufgeführten <i>FORM-OBJEKTE</i> müssen an anderer Stelle in der ConversionBase definiert sein.
VarExec	Command Command	Die Tabelle <i>VAREXEC</i> kann in den CVB-Objekten <i>JOB</i> , <i>FORM</i> und <i>CONVERSION</i> genutzt werden. Wegen ihrer Komplexität wird sie in einem extra Abschnitt behandelt.
StandardForm	StdFormName	Name des <i>FORM-OBJEKTES</i> , das aktiviert wird, wenn keiner der <i>FORM-OBJEKTE</i> aus der FormList erkannt wurde oder FormList nicht definiert ist. Das als StandardForm angegebene <i>FORM-OBJEKT</i> ist ein beliebiges <i>FORM-OBJEKT</i> und kann eine eigene <i>RECOGNITION</i> besitzen. Diese bleibt jedoch für die Verwendung des <i>FORM-OBJEKTES</i> als StandardForm unberücksichtigt.

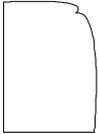


Anmerkung: Die Angabe aller Elemente ist optional.



Beispiel:

```
Job Rechnungen {  
  Comment „interne und externe Rechnung“  
  Recognition {  
    OR {  
      Is_Text_At 12 3 „Rechnung“  
      Is_Text_At 12 3 „interne Rechnung“  
    }  
  }  
  Forms {  
    Rechnung  
    Rechnung_int  
  }  
  StandardForm weisses_Papier  
}
```



Form



Verwendung: Beschreibt ein *FORM-OBJEKT*, woran es zu erkennen ist, welche „Durchschläge“ es beinhaltet, welche Ausgabeparameter für das *FORM-OBJEKT* als Ganzes gelten sollen.



Syntax:

```

Form FormName {
  Comment „Kommentar“
  Recognition {
    Command
  }
  Pages {
    PageName1
    PageName2
    PageName3
    ...
  }
  VarExec {
    Command
    Command
    ...
  }
  Sort Wert
  NewSpoolFile Wert
  Lines Wert
  EndOfJob Wert
}
    
```

	<u>Parameter</u>	<u>Bedeutung</u>
Form	FormName	Name des <i>FORM-OBJEKTES</i> , der an dieser Stelle definiert wird; dient dazu, sich an anderer Stelle in der ConversionBase auf dieses <i>FORM-OBJEKT</i> zu beziehen.
Comment	Kommentar	Kommentartext wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden.
Recognition		wird wegen ihrer Komplexität in einem getrennten Abschnitt behandelt.

Pages	PageName1, PageName2, PageName3	Namen der Durchschläge, die in der angegebenen Reihenfolge verwendet werden.
VarExec	Command Command	Das Element <i>VAREXEC</i> kann in den CVB-Objekten <i>JOB</i> , <i>FORM</i> und <i>CONVERSION</i> genutzt werden. Wegen ihrer Komplexität wird sie in einem extra Abschnitt behandelt.

Sort	Wert = 0	Formularsatzweise sortiert, Default-Wert
	Wert = 1	nach Kopien sortiert (alle Originale aufeinander, danach alle 1. Durchschläge usw.)
	Wert = 2	Sortierung geschieht nur nach Druckern Pro Drucker wird ein File angelegt
	 Anmerkung:	Soll der Parameter über mehrere Formularerkennungen hin wirksam werden (z.B. 1. Seite und Folgeseiten von Rechnung), darf in der Definition des <i>PAGE-OBJEKTES</i> des Folgeformulars weder der Befehl <i>JobName</i> noch <i>FormName</i> verwendet werden. Statt dessen ist das MDF-File über den <i>LINK</i> zu einem <i>DESIGN-OBJEKT</i> oder über <i>SUBFORM</i> im <i>CONVERSION-OBJEKT</i> einzubinden. Ist Sort=1 in zwei oder mehreren Formularen gesetzt, die nacheinander im Datenstrom erkannt werden, und ist weiter o.g. Bedingung erfüllt, wird das Sort auf alle Originale, 1. Kopien usw. dieser Formulare ausgedehnt. Sie erhalten also beispielsweise erst alle Originale von Form1 dann von Form 2, anschließend alle 1. Kopien von Form1, anschließend von Form 2 usw. Wenn der Sort-Effekt unerwünscht ist, können die Befehle <i>NewSpoolFile</i> oder <i>EndOfJob</i> zum unterbrechen des Sorts verwendet werden.
NewSpoolFile	Wert = -1	vor Abarbeitung dieses <i>FORM- OBJEKTES</i>
	Wert = 0	trifft keine Aussage, Default-Wert
	Wert = 1	nach Abarbeitung diesem <i>FORM- OBJEKTES</i>
	Wert = 2	vor und nach der Abarbeitung dieses <i>FORM-OBJEKTES</i> , der Ausgabedatenstrom dieses <i>FORM- OBJEKTES</i> wird in ein separates Spoolfile ausgegeben
Lines	Wert = 0	beliebig viele Zeilen, Default-Wert
	Werte > 0	Anzahl der Zeilen



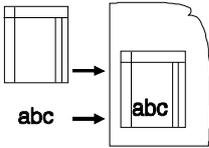
Anmerkung: Lines gibt die maximale zu interpretierende Zeilenanzahl für das aktuelle *FORM-OBJEKT* an. Bei Überschreiten der Zeilenzahl erfolgt ein Seitenumbruch. Nachfolgend werden die *RECOGNITIONS* aus der *FormList* aktiviert. D.h., es wird so verfahren, als sei eine neue Seite Input-Daten hereingekommen. Die Verwendung von Lines ist dann sinnvoll, wenn die Zeilenzahl im gedruckten *FORM-OBJEKT* reduziert werden soll gegenüber der Zeilenzahl pro Seite, die das druckende System erzeugt oder, wenn nicht sichergestellt ist, dass die Seite hereinkommende Daten mit einem *FormFeed* endet.

EndOfJob

Wert = -1	vor Abarbeitung dieses <i>FORM-OBJEKTES</i>
Wert = 0	trifft keine Aussage, Default-Wert
Wert = 1	nach Abarbeitung dieses <i>FORM-OBJEKTES</i>
Wert = 2	vor und nach Abarbeitung dieses <i>FORM-OBJEKTES</i>



Anmerkung: Es gibt weitere Ausgabeparameter, die betreffen nur einen einzelnen Durchschlag und werden im *PAGE-OBJEKT* definiert.



Page



Verwendung: Beschreibt eine Page (Durchschlag), welches *CONVERSION-OBJEKT* auf die Daten angewandt werden soll, welche Designteile zu dieser Seite gehören und welche Ausgabeparameter für diese einzelnen Seiten gelten sollen. Alle Key-Wörter im *PAGE-OBJEKT* können mit einem dynamischen Befehl definiert werden.



Syntax:

```

Page PageName {
  Conversion ConvName
  Design DesignName
  FormName JF_FormName
  JobName JF_JobName
  Intray Wert
  Outtray Wert
  Duplex Wert
  Printer PrinterName
  Recognition {
    Operator {
      Command
      Command
      ...
    }
  }
  Comment „Kommentar“
  Copy Kopiezahl
  CopyDiv Wert
  Archive Wert
  ArcRefFields „Feld1,Feld2, ...“
  Get_VAR VarName
}
    
```

	<u>Parameter</u>	<u>Bedeutung</u>
Page	PageName	Name des <i>PAGE-OBJEKTES</i> , die an dieser Stelle definiert wird; dient dazu, sich an anderer Stelle in der ConversionBase auf dieses <i>PAGE-OBJEKT</i> zu beziehen.
Comment	Kommentar	Kommentartext wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden.

Conversion	ConvName	Name des zu verwendenden <i>CONVERSION-OBJEKTES</i> , muss an anderer Stelle in der ConversionBase definiert sein.
Design	DesignName	Name des einzubindenden <i>DESIGN-OBJEKTES</i> .
Copy <i>CommandOrValue</i>	Kopiezahl	Gibt an, wie viel Kopien (inkl. Original) von dieser Seite erzeugt und ausgegeben werden sollen.
CopyDiv <i>CommandOrValue</i>	Wert	Gibt an, wie viel gleiche Elemente sich auf der zu druckenden Seite befinden ausgegeben werden sollen.
 Anmerkung:	Es ist die einfachste Vorgehensweise, mit einer oder mehreren ganzen Seiten (Durchschlägen) aus einem MDF-File zu arbeiten. Es auch möglich, diesen Eintrag wegzulassen und das Aussehen des Formulars aus einzelnen Teilen (<i>SUBFORMS</i>) aufzubauen. Dann sind die CB-Objekt-Elemente StartSubForm und StopSubForm im Objekt <i>DYNBLOCK</i> und/oder das Objekt-Element <i>SUBFORM</i> in den CVB-Objekten <i>DESIGN</i> und <i>BLOCK</i> zu verwenden.	
 Anmerkung:	Diese Option wird im Zusammenhang mit dem Druck von Etikettenbogen genutzt, wenn in den Daten nur die Anzahl der Etiketten und nicht die Anzahl der zu druckenden Etikettenbogen vorhanden ist. Die Anzahl der Etikettenbogen berechnet sich wie folgt: $Copy = \text{ganzzahliger Wert}(\text{Copy}/\text{CopyDiv})$ Dabei wird immer auf den nächsten ganzzahligen Wert aufgerundet, wenn $Copy/ CopyDiv$ nicht ganzzahlig ist. Beispiel: Copy 30 CopyDiv 8 Die Anzahl der Kopien ist demnach 4 ($30/8=3,75$).	
FormName <i>CommandOrValue</i>	JF_FormName	Name des zu verwendenden Adobe Present/Designs (*.mdf-File), ist gleichzeitig Formularaufruf für Adobe Present/Central (^ Form-Kommando) Pflichtangabe!!
JobName <i>CommandOrValue</i>	JF_JobName	Name für den Job, den der FormDirector erzeugt und der Adobe Present/Central übergeben wird (^ Job-Kommando)

Intray <i>CommandOrValue</i>	Wert = -1	beliebig (Default-Wert)
	Wert > 0	Nummer des Papierschachtes, aus dem die Seite gezogen werden soll (welcher Schacht mit welcher Nummer angesprochen wird, ist von Druckermodell zu Druckermodell verschieden.)
Outtray <i>CommandOrValue</i>	Wert = -1	beliebig (Default-Wert)
	Wert > 0	Nummer der Papierablage, in der die Seite abgelegt werden soll (welche Ablage mit welcher Nummer angesprochen wird, ist von Druckermodell zu Druckermodell verschieden.)
Duplex <i>CommandOrValue</i>	Wert = -1	alle Duplexoptionen ausschalten
	Wert = 0	keine Angabe (Default-Wert)
	Wert = 1	an der langen Seite gebunden
	Wert = 2	an der kurzen Seite gebunden
Printer <i>CommandOrValue</i>	PrinterName	Logischer Name des Druckers, auf dem die Seite gedruckt werden soll (hierbei kann es sich auch um Fax- oder Archivsysteme handeln); der logische Drucker muss im Adobe Present/Control vereinbart sein.
Archive <i>CommandOrValue</i>	Wert = 0	Bedeutet, dass die Seite nicht zu archivieren ist.
	Wert = 1	Bedeutet, dass die Seite als erste Seite eines Archiv-Dokumentes zu archivieren ist.
	Wert = 2	Bedeutet, dass die Seite als Folgeseite des Archiv-Dokumentes zu archivieren ist. Der Wert kann auch als CommandOrValue berechnet werden. z.B. Archive 2 Archive „@Command“ Archive „@GetVar BussinesArchiv“
ArcRefFields	„Feld1,Feld2,...“	Gibt die Felder an, die als Belegindex mit dem Dokument zu archivieren sind. Der Gesamtbegriff steht in Anführungszeichen. Die Feldnamen werden durch Komma getrennt. Leerzeichen sind nicht erlaubt. Folgende Felder sind zulässig: <ul style="list-style-type: none"> • Alle Variablen, die in der Variablenliste definiert wurden. Die Angabe von Variablen erfolgt stets mit einem vorangestellten @, • Alle Felder, die im entsprechenden <i>CONVERSION-OBJEKT</i> mit den Befehlen <i>READ_FIELD</i>, <i>READ_GLOBAL</i> oder deren Derivate erzeugt wurden.

Recognition

Das *RECOGNITION*-Element ist optional und hat folgende Bedeutung:

Aussage falsch → dieses *PAGE-OBJEKT* wird nicht abgearbeitet

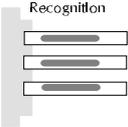
Aussage wahr → dieses *PAGE-OBJEKT* wird abgearbeitet



Beispiel :

ArcRefFields Anschrift,@Mandant,Datum

Anschrift und Datum sind Felder aus dem *CONVERSION-OBJEKT*.
Mandant ist eine Variable aus der Variablenliste.



Recognition



Verwendung: Beschreibt, anhand welcher Kriterien im Datenstrom ein Objekt erkannt werden soll



Syntax:

```

Recognition {
  OR {
    Is_Text_At xPos yPos Zeichenkette           ①
    Is_Var_Eql VarName „Match-Text“
    Is_Var_Not_Eql VarName „Match-Text“
    Is_Var_Known VarName
    Is_Var_Unknown VarName
    Is_Not_Text_At xPos yPos Zeichenkette       ②
    Is_Text_At_Eql_Var xPos yPos VarName
    AND {                                       ③
      Contains_Text_In xPos yPos xDim yDim Zeichenkette ④
      Contains_Not_Text_In xPos yPos xDim yDim Zeichenkette ⑤
    }
  }
}
    
```

Recognition	Parameter	Bedeutung
Recognition		Die <i>RECOGNITION</i> setzt sich aus einer Reihe von Prüfkommandos zusammen, die durch logische Operatoren miteinander verknüpft werden können. Prüfkommandos, die ohne Operator nacheinander aufgeführt werden, sind von sich aus UND-verknüpft.
Is_Text_At		Prüft, ob die angegebene Zeichenkette an der angegebenen Stelle im Datenstrom vorkommt.
Is_Var_Eql	VarName „Matchtext“	Entspricht der Fix in der CVB stehend Match-Text den Wert er Variable VarName JA → erkannt NEIN → nicht erkannt
Is_Var_Not_Eql	VarName „Matchtext“	Entspricht der Fix in der CVB stehend Match-Text den Wert er Variable VarName JA → nicht erkannt NEIN → erkannt
Is_Var_Known	VarName	Ist die Variable nicht definiert oder der Inhalt ist leer oder besteht nur aus Leerzeichen. JA → nicht erkannt NEIN → erkannt
Is_Var_Unknown	VarName	Ist die Variable nicht definiert oder der Inhalt

		ist leer oder besteht nur aus Leerzeichen. JA → erkannt NEIN → nicht erkannt In den CVB-Objekte von <i>JOB</i> , <i>FORM</i> und <i>BLOCK</i>
Is_Not_Text_At		Prüft, ob die angegebene Zeichenkette an der angegebenen Stelle im Datenstrom nicht vorkommt.
	Gemeinsame Parameter:	
	yPos	Zeile, in der die gesuchte Zeichenkette stehen soll.
	xPos	Spalte, in der die gesuchte Zeichenkette beginnen soll.
	Zeichenkette	Kombination aus Buchstaben, Zahlen, Leer- und Sonderzeichen, an welcher der Datenstrom erkannt werden soll.
Contains_Text_In		Prüft, ob die angegebene Zeichenkette im angegebenen Bereich im Datenstrom vorkommt.
Contains_Not_Text_In		Prüft, ob die angegebene Zeichenkette im angegebenen Bereich im Datenstrom nicht vorkommt.
	Gemeinsame Parameter:	
	yPos	Zeile, in welcher der zu durchsuchende Bereich beginnen soll
	xPos	Spalte, in welcher der zu durchsuchende Bereich beginnen soll (linker Rand).
	yDim	Ausdehnung des zu durchsuchenden Bereichs in Zeilen (oberer Rand).
	xDim	Ausdehnung des zu durchsuchenden Bereichs in Spalten
	Zeichenkette	Kombination aus Buchstaben, Zahlen, Leer- und Sonderzeichen, an welcher der Datenstrom erkannt werden soll
 Anmerkung:		Für die Parameter xPos, yPos, xDim und yDim kann anstelle einer genauen Angabe auch eine „0“ oder ein „*“ angegeben werden. In diesem Fall werden alle Zeilen und/oder Spalten herangezogen bzw. wird die Ausdehnung (xDim, yDim) bis auf den rechten und unteren Seitenrand ausgedehnt.
Is_Text_At_Eql_Var		Prüft, ob der Text auf einer bestimmten Position dem Inhalt einer Variable entspricht
	yPos	Zeile, in welcher der zu durchsuchende Bereich beginnen soll
	xPos	Spalte, in welcher der zu durchsuchende Bereich beginnen soll (linker Rand).
	yDim	Ausdehnung des zu durchsuchenden Bereichs in Zeilen (oberer Rand).

	xDim	Ausdehnung des zu durchsuchenden Bereichs in Spalten
	VarName	Name der Variable mit der verglichen wird
AND		verknüpft <i>COMMANDS</i> oder weitere Operatoren, alle in den <i>COMMANDS</i> aufgeführten Kriterien müssen wahr sein, wenn die Daten als erkannt gelten sollen
OR		verknüpft <i>COMMANDS</i> oder weitere Operatoren, es reicht aus, wenn eines der in den <i>COMMANDS</i> aufgeführten Kriterien wahr ist, damit die Daten als erkannt gelten
NAND		verknüpft <i>COMMANDS</i> oder weitere Operatoren, die Daten gelten als erkannt, wenn eines der aufgeführten Kriterien nicht wahr ist oder die Daten gelten als nicht erkannt, wenn alle aufgeführten Kriterien wahr sind.
NOR		verknüpft <i>COMMANDS</i> oder weitere Operatoren, die Daten gelten als erkannt, wenn alle aufgeführten Kriterien nicht erfüllt sind oder die Daten gelten als nicht erkannt, wenn eines der Kriterien erfüllt ist



Anmerkung: In der oben aufgeführten Konstellation sind ①, ② und ③ OR-verknüpft. OR-Kriterium ③ setzt sich aus zwei AND- verknüpften Kriterien zusammen: ④ und ⑤ .



Beispiel:

```
Recognition {
  Is_Text_At 3 121 „Seite“
  Is_Text_At 2 3 „** V8218R **“
}
```

Die beiden Kriterien sind von sich aus AND-verknüpft. Die Daten sind dann identifiziert, wenn in Zeile 3, in Spalte 121 das Wort Seite beginnt und wenn in Zeile 2, in Spalte 3 die Zeichenkette **** V8218R **** beginnt.

```
Recognition {
  OR {
    Is_Text_At 2 3 „** V8217R **“
    Is_Text_At 2 3 „** V8218R **“
  }
}
```

Eine einfache OR-Verknüpfung: Die Daten gelten als erkannt, wenn in Zeile 2, beginnend mit Spalte 3 die Zeichenkette **** V8217R **** oder die Zeichenkette **** 8218R **** steht

Syntax:

```
Recognition {  
  Is_Text_At 3 121 „Seite“  
  OR {  
    Is_Text_At 2 3 „** V8217R **“  
    Is_Text_At 2 3 „** V8218R **“  
  }  
}
```

Kommentar:

Kriterium 1

Kriterium 2

Kriterium 3

Das erste *COMMAND* und der OR-Operator sind AND-verknüpft. Die Daten gelten als erkannt, wenn Kriterium 2 oder Kriterium 3 und in jedem Fall zusätzlich Kriterium 1 erfüllt sind.

aufwendige *RECOGNITION*:

Syntax:

```
Recognition {  
  Is_Text_At 3 121 „Seite“  
  OR {  
    Is_Text_At 2 3 „** V8217R **“  
  }  
}
```

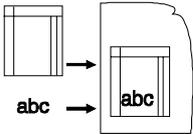
Beispiel für eine zu

Kommentar:

Kriterium 1

Kriterium 2

Das erste *COMMAND* und der OR-Operator sind AND-verknüpft. Der Operator beinhaltet keine Verknüpfung, sondern hängt nur von einem *COMMAND* ab. D.h., das Ergebnis von Kriterium 2 wirkt sich 1:1 auf die AND-Verknüpfung aus. Damit hätte der OR-Operator weggelassen werden können (siehe 1. Beispiel).



Conversion

Die Conversion ist eine Liste von Befehlen, die von oben nach unten abgearbeitet wird. Die Befehle können dem Ausgabe-Stream Ausgaben hinzufügen oder Variablen lesen und schreiben.



Syntax:

```

Conversion {
    ...
    Kommando
    ...
}

```

Folgende Kommandos sind unterstützt:

Comment	„Kommentar“
Block	yPos xPos BlockName
DynBlock	yPos xPos yDim xDim BlockName
Read	yPos xPos yDim xDim
Read_Minimized	yPos xPos yDim xDim
Read_Global	yPos xPos yDim xDim FeldName [Attr]
Read_Field	yPos xPos yDim xDim FeldName [Attr]
Read_Field_Minimized	yPos xPos yDim xDim FeldName [Attr]
Read_Global_Minimized	yPos xPos yDim xDim FeldName [Attr]
Take	yPos xPos yDim xDim
Take_Minimized	yPos xPos yDim xDim
Take_Field	yPos xPos yDim xDim FeldName [Attr]
Take_Global	yPos xPos yDim xDim FeldName [Attr]
Take_Field_Minimized	yPos xPos yDim xDim FeldName [Attr]
Take_Global_Minimized	yPos xPos yDim xDim FeldName [Attr]
Move	yPos xPos yDim xDim yTo xTo
Number	yPos xPos xDim
Copy	yPos xPos yDim xDim yTo xTo
Kill	yPos xPos yDim xDim
Insert	yPos xPos „Fixer Text“
JF_Command	„JF-Befehl“ „Wert1“ „Wert2“
Addi	VarName VarPlusName
Subi	VarName VarPlusName
Set_Var	VarName yPos xPos xDim
Set_Var_If_Unknown	VarName yPos xPos xDim
Set_Var_Fix	VarName „Fixer Text“
Set_Var_Fix_If_Unknown	VarName „Fixer Text“
Del_Var	VarName
Del_All_Vars	
Write_Vars_To_File	FileName
Read_Vars_From_File	FileName
Get_Substitute	VarName SubstTableName
Substitute	InVar SubstTableName OutVar
Concat	OutVar InVar1 InVar2 [...]
Token	InVar Delimiter OutVar1 [...]

Strip	InVar [Orientation [Char]]
Insert_Var	yPos xPos VarName
Fill	VarName Ori Size FillChar
Exit	Message
Convert_Vars_In_File	OutVar FileName InVar0 InVar1 InVar2...
Convert_Vars_In_File_If_Known	OutVar FileName InVar0 InVar1 InVar2...
Convert_Vars_In_File_If_Unknown	OutVar FileName InVar0 InVar1 InVar2...

Comment



Verwendung: Kommentar zur Strukturierung der Conversion-Befehle.



Syntax:

Comment „Kommentar“

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
Kommentar	Kein Kommentar

Block



Verwendung: Markiert einen Fensterbereich der aktuellen Seite und übergibt diesen Bereich zur Conversion an ein in der ConversionBase definiertes Block-Objekt zur Abarbeitung. Eine genaue Beschreibung wie ein Block arbeitet, finden Sie unter Block in diesem Handbuch.



Syntax:

Block yPos xPos BlockName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
yPos	Zeile, in der der DynBlock beginnt
xPos	Spalte, in der der DynBlock beginnt
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
BlockName	Names des Blocks

DynBlock



Verwendung: Markiert einen Fensterbereich der aktuellen Seite und übergibt diesen Bereich zur Conversion an ein in der ConversionBase definiertes DynBlock-Objekt zur Abarbeitung. Eine genaue Beschreibung wie ein DynBlock arbeitet, finden Sie unter DynBlock in diesem Handbuch.



Syntax:

DynBlock yPos xPos yDim xDim BlockName

Parameter

yPos
xPos
yDim
xDim
BlockName

Bedeutung/Bemerkung

Zeile, in der der DynBlock beginnt
Spalte, in der der DynBlock beginnt
Vertikale Ausdehnung in Zeilen
Horizontale Ausdehnung in Zeilen
Names des Blocks

Read



Verwendung: Ließt einen Block von der aktuellen Seite und stellt das Ergebnis ohne ein führendes ^field oder ^global in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite bleibt erhalten. Der Befehl kann dazu verwendet werden, JetForm-Kommandos von der Seite zu lesen oder weitere Daten dem vorangegangenen Field oder Global anzuhängen.



Syntax:

Read yPos xPos yDim xDim

Parameter

yPos
xPos
yDim
xDim

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll
Spalte, in der das Lesen begonnen werden soll
Vertikale Ausdehnung in Zeilen
Horizontale Ausdehnung in Zeilen

Read_Minimized



Verwendung: Ließt einen Block von der aktuellen Seite, minimiert den Block so, das führende und abschließende Leerzeilen und Leerzeichen entfernt werden und stellt das Ergebnis ohne ein führendes ^field oder ^global in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite bleibt erhalten. Der Befehl kann dazu verwendet werden, JetForm-Kommandos von der Seite zu lesen oder weitere Daten dem vorangegangenen Field oder Global anzuhängen.



Syntax:

Read_Minimized yPos xPos yDim xDim

Parameter

yPos

xPos

yDim

xDim

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll

Spalte, in der das Lesen begonnen werden soll

Vertikale Ausdehnung in Zeilen

Horizontale Ausdehnung in Zeilen

Read_Field



Verwendung: Ließt einen Block von der aktuellen Seite und stellt das Ergebnis mit einem führenden ^field FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite bleibt erhalten.



Syntax:

Read_Field yPos xPos yDim xDim FeldName [Attr]

Parameter

yPos

xPos

yDim

xDim

FeldName

Attr

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll

Spalte, in der das Lesen begonnen werden soll

Vertikale Ausdehnung in Zeilen

Horizontale Ausdehnung in Zeilen

Name des GLOBAL-Feldes

Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Read_Global



Verwendung: Ließt einen Block von der aktuellen Seite und stellt das Ergebnis mit einem führenden ^global FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite bleibt erhalten.



Syntax:

Read_Global yPos xPos yDim xDim FeldName [Attr]

Parameter

yPos

xPos

yDim

xDim

FeldName

Attr

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll

Spalte, in der das Lesen begonnen werden soll

Vertikale Ausdehnung in Zeilen

Horizontale Ausdehnung in Zeilen

Name des GLOBAL-Feldes

Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Read_Field_Minimized



Verwendung: Liebt einen Block von der aktuellen Seite, minimiert den Block so, das führende und abschließende Leerzeilen und Leerzeichen entfernt werden und stellt das Ergebnis mit einem führenden ^field FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite bleibt erhalten.



Syntax:

Read_Field_Minimized yPos xPos yDim xDim FeldName [Attr]

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
FeldName	Name des GLOBAL-Feldes
Attr	Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Read_Global_Minimized



Verwendung: Liebt einen Block von der aktuellen Seite, minimiert den Block so, das führende und abschließende Leerzeilen und Leerzeichen entfernt werden und stellt das Ergebnis mit einem führenden ^global FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite bleibt erhalten.



Syntax:

Read_Global_Minimized yPos xPos yDim xDim FeldName [Attr]

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
FeldName	Name des GLOBAL-Feldes
Attr	Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Take



Verwendung: Liebt einen Block von der aktuellen Seite und stellt das Ergebnis ohne ein führendes `^field` oder `^global` in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite geht verloren und wird mit Leerzeichen gefüllt. Der Befehl kann dazu verwendet werden, JetForm-Kommandos von der Seite zu lesen oder weitere Daten dem vorangegangenen Field oder Global anzuhängen.



Syntax:

Take yPos xPos yDim xDim

Parameter

yPos

xPos

yDim

xDim

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll

Spalte, in der das Lesen begonnen werden soll

Vertikale Ausdehnung in Zeilen

Horizontale Ausdehnung in Zeilen

Take_Minimized



Verwendung: Liebt einen Block von der aktuellen Seite, minimiert den Block so, das führende und abschließende Leerzeilen und Leerzeichen entfernt werden und stellt das Ergebnis ohne ein führendes `^field` oder `^global` in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite geht verloren und wird mit Leerzeichen gefüllt. Der Befehl kann dazu verwendet werden, JetForm-Kommandos von der Seite zu lesen oder weitere Daten dem vorangegangenen Field oder Global anzuhängen.



Syntax:

Take_Minimized yPos xPos yDim xDim

Parameter

yPos

xPos

yDim

xDim

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll

Spalte, in der das Lesen begonnen werden soll

Vertikale Ausdehnung in Zeilen

Horizontale Ausdehnung in Zeilen

Take_Field



Verwendung: Ließt einen Block von der aktuellen Seite und stellt das Ergebnis mit einem führenden ^field FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite geht verloren und wird mit Leerzeichen gefüllt.



Syntax:

Take_Field yPos xPos yDim xDim FeldName [Attr]

Parameter

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
FeldName	Name des GLOBAL-Feldes
Attr	Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Take_Global



Verwendung: Ließt einen Block von der aktuellen Seite und stellt das Ergebnis mit einem führenden ^global FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite geht verloren und wird mit Leerzeichen gefüllt.



Syntax:

Take_Global yPos xPos yDim xDim FeldName [Attr]

Parameter

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
FeldName	Name des GLOBAL-Feldes
Attr	Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Take_Field_Minimized



Verwendung: Liebt einen Block von der aktuellen Seite, minimiert den Block so, das führende und abschließende Leerzeilen und Leerzeichen entfernt werden und stellt das Ergebnis mit einem führenden ^field FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite geht verloren und wird mit Leerzeichen gefüllt.



Syntax:

Take_Field_Minimized yPos xPos yDim xDim FeldName [Attr]

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
FeldName	Name des GLOBAL-Feldes
Attr	Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Take_Global_Minimized



Verwendung: Liebt einen Block von der aktuellen Seite, minimiert den Block so, das führende und abschließende Leerzeilen und Leerzeichen entfernt werden und stellt das Ergebnis mit einem führenden ^global FeldName in den Ausgabe-Stream. Der Originale-Bereich auf der aktuellen Seite geht verloren und wird mit Leerzeichen gefüllt.



Syntax:

Take_Global_Minimized yPos xPos yDim xDim FeldName [Attr]

Parameter

Parameter	Bedeutung/Bemerkung
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
FeldName	Name des GLOBAL-Feldes
Attr	Schalter mit den Werten 0 und 1. Dabei ist 0 ist Standardwert. Ist Attr 1, dann werden Attribute wie Bold und Italic als Inline-Kommandos mitgeneriert

Move



Verwendung: Entfernt Daten im angegebenen Bereich der aktuellen Seite und fügt sie an andere Stelle der Seite wieder ein.



Syntax:

Move yPos xPos yDim xDim yTo xTo

Parameter

Parameter	Bedeutung/Bemerkung
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
yDim	Vertikale Ausdehnung in Zeilen
xDim	Horizontale Ausdehnung in Zeilen
yTo, xTo	Ziel Koordinaten in Zeilen und Spalten

Copy



Verwendung: Kopiert Daten von einem bestimmten Bereich der aktuellen Seite und fügt sie an anderer Stelle der Seite wieder ein. Die Originaldaten bleiben erhalten.



Syntax:

Copy yPos xPos yDim xDim yTo xTo

Parameter

yPos
xPos
yDim
xDim
yTo, xTo

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll
Spalte, in der das Lesen begonnen werden soll
Vertikale Ausdehnung in Zeilen
Horizontale Ausdehnung in Zeilen
Ziel Koordinaten in Zeilen und Spalten

Kill



Verwendung: Kill löscht von der aktuellen Seite einen Block der angegebenen Position und Größe indem der Bereich mit Blanks überschrieben wird.



Syntax:

Kill yPos xPos yDim xDim

Parameter

yPos
xPos
yDim
xDim

Bedeutung/Bemerkung

Zeile, in der das Löschen begonnen werden soll
Spalte, in der das Löschen begonnen werden soll
Vertikale Ausdehnung in Zeilen
Horizontale Ausdehnung in Zeilen

Insert



Verwendung: Insert fügt an der angegebenen Position einen fixen Text ein.



Syntax:

Insert yPos xPos „Fixer Text“

Parameter
yPos
xPos
Fixer Text

Bedeutung/Bemerkung
Zeile, in der das Schreiben begonnen werden soll
Spalte, in der das Schreiben begonnen werden soll
Einzufügender Text

JF_Command



Verwendung: JF_Command für einen JetForm-Befehl in den Ausgabe-Strem ein.



Syntax:

JF_Command „JF-Befehl“ „Wert1“ „Wert2“

Parameter
JF-Befehl
Wert1 und Wert2

Bedeutung/Bemerkung
JetForm-Befehl
Parameter 1 und 2 des JetForm-Befehls.

Set_Var



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der auf der Seite an der angegebenen Position steht.



Syntax:

Set_Var VarName yPos xPos xDim

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu überschreibenden oder anzulegenden Variable
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
xDim	Anzahl der Spalten, die von xPos beginnend gelesen werden sollen

Set_Var_If_Unknown



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der auf der Seite an der angegebenen Position steht. Der Befehl wird nur ausgeführt, wenn in der variable VarName vor dem Setzen kein Inhalt vorhanden war.



Syntax:

Set_Var_If_Unknown VarName yPos xPos xDim

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu überschreibenden oder anzulegenden Variable
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
xDim	Anzahl der Spalten, die von xPos beginnend gelesen werden sollen

Set_Var_Fix



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der Fix als Text angegeben ist.



Syntax:

Set_Var_Fix VarName „Fixer Text“

Parameter
VarName
FixText

Bedeutung/Bemerkung
Name der zu überschreibenden oder anzulegenden Variable
Fester Text

Set_Var_Fix_If_Unknown



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der Fix als Text angegeben ist. Der Befehl wird nur ausgeführt, wenn in der Variable VarName vor dem Setzen kein Inhalt vorhanden war.



Syntax:

Set_Var_Fix_If_Unknown VarName „Fixer Text“

Parameter
VarName
FixText

Bedeutung/Bemerkung
Name der zu überschreibenden oder anzulegenden Variable
Fester Text

Del_All_Vars



Verwendung: Del_All_Vars löscht alle Variablen der Variablen-Tabelle.



Syntax:

Del_All_Vars

Del_Var



Verwendung: Del_Var löscht die angegebene Variable aus der Variablen-Tabelle.



Syntax:

Del_Var VarName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu löschenden Variable

Read_Vars_From_File



Verwendung: Liest die Variablen, die im angegebenen File definiert sind zu den bereits existierenden Variablen hinzu oder überschreibt diese.



Syntax:

Read_Vars_From_File FileName

Write_Vars_To_File



Verwendung: Schreibt alle Variablen in den angegebenen File. Ein eventuell existierender File gleichen Namens wird dabei überschrieben.



Syntax:

Write_Vars_To_File FileName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
FileName	Name des Variablen-Files

Get_Substitute



Verwendung: Get_Substitute konvertiert eine Variable in einer Substitute-Tabelle.



Syntax:

Get_Substitute VarName SubstName

Substitute



Verwendung: Substitute konvertiert den Wert einer Variablen in einer Substitution-Tabelle und schreibt das Ergebnis in eine Variable.



Syntax:

Substitute InVar Tabelle OutVar

Parameter

InVar

Bedeutung/Bemerkung

Name der Variable mit der in einer Substitution-Tabelle ein Austausch durchgeführt werden soll

Tabelle

Name der Substitution-Tabelle

OutVar

Zielvariable, in die das Ergebnis geschrieben werden soll

Concat



Verwendung: Concat verkettet den Inhalt der unter InVarN angegebenen Variablen und schreibt das Ergebnis in die OutVar Variable.



Syntax:

Concat OutVar InVar1 [InVar2 [...]]

Parameter

OutVar

Bedeutung/Bemerkung

Name der Zielvariable

InVarN

Namen der Variablen, die zu seinem Gesamtstring zusammengefasst werden sollen

Token



Verwendung: Token unterteilt den Inhalt einer Variablen mit Hilfe des Delimiter-Trennzeichens in Teilstrings und gibt diese in die angegebenen OutVars aus.



Syntax:

Token InVar Delimiter OutVar1 OutVar2 [...]

Parameter
InVar
Delimiter
OutVarN

Bedeutung/Bemerkung
Name der Variable die aufzugliedern ist
Trennzeichen
Zielvariablen für die Teilstrings

Strip



Verwendung: Strip beseitigt nicht gewollte Zeichen aus einer Variable.



Syntax:

Strip VarName [Orientation [Char]]

Parameter
VarName
Orientation
Char

Bedeutung/Bemerkung
Name der In- und OutVar
Richtung des Strips (BOTH, LEFT und RIGHT)
Zeichen das mit Strip beseitigt werden soll (Standard Blank)

Insert_Var



Verwendung: Insert_Var fügt den Inhalt der Variable auf der aktuellen Seite an der angegebenen Position ein.



Syntax:

Insert_Var yPos xPos VarName

Parameter

yPos

xPos

VarName

Bedeutung/Bemerkung

Zeile, in der das Schreiben begonnen werden soll

Spalte, in der das Schreiben begonnen werden soll

Name Variable

Fill



Verwendung: Fill füllt den Inhalt der unter VarName angegebenen Variable mit FillChar-Zeichen auf. Ist FillChar nicht definiert, so wird Blank als FillChar-Zeichen verwendet. Das Ergebnis wird in die selbe Variable zurückgeschrieben.



Syntax:

Fill VarName Ori Size [FillChar]

Parameter

VarName

Ori

Size

FillChar

Bedeutung/Bemerkung

Name der In- und Out-Variable

Orientation (LEFT oder RIGHT)

Neue Größe des Strings

Füllbuchstabe (Standard ist Blank)

Exit



Verwendung: Exit verlässt das Programm mit der Fehlernummer -900 und gibt im Log- und Response-File die angegebene Fehlermeldung mit.



Syntax:

Exit Message

Parameter
Message

Bedeutung/Bemerkung
Abbruchnachricht

Convert_Vars_In_File



Verwendung: Convert_Vars_In_File konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle.



Syntax:

Convert_Vars_In_File OutVar FileName InVar0 InVar1 InVar2...

Parameter
OutVar
FileName
InVarN

Bedeutung/Bemerkung
Name der Zielvariable
Name des Conversion-Files
Variablen, die zu Konvertierung herangezogen werden

Convert_Vars_In_File_If_Known



Verwendung: Convert_Vars_In_File_If_Known konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle. Der Befehl wird nur ausgeführt, wenn in der Variable OutVar vor der Konvertierung bereits ein Inhalt vorhanden war.



Syntax:

```
Convert_Vars_In_File_If_Known OutVar FileName InVar1 [InVarN]
```

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
OutVar	Name der Zielvariable
FileName	Name des Conversion-Files
InVarN	Variablen, die zu Konvertierung herangezogen werden

Convert_Vars_In_File_If_Unknown



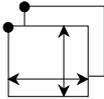
Verwendung: Convert_Vars_In_File_If_Unknown konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle. Der Befehl wird nur ausgeführt, wenn in der Variable OutVar vor der Konvertierung kein Inhalt vorhanden war.



Syntax:

```
Convert_Vars_In_File_If_Unknown OutVar FileName InVar1 [InVarN]
```

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
OutVar	Name der Zielvariable
FileName	Name des Conversion-Files
InVarN	Variablen, die zu Konvertierung herangezogen werden



DynBlock



Verwendung: Spannt auf der Seite ein Suchfenster auf, innerhalb dessen nach den *RECOGNITIONS* für die *BLOCK-OBJEKTE* gesucht wird, die im *DYNBLOCK-OBJEKT* aufgeführt werden.



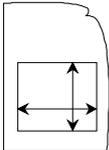
Syntax:

```

DynBlock BlockName {
  Comment „Kommentar“
  Blocks {
    Blockname1
    Blockname2
    ...
  }
  StandardBlock StdBlockName
  StartSubForm {
    Comment Kommentar
    MDF_FileName FileName
    JfPageNo Seitennummer
    PositionField FeldName
  }
  StopSubForm {
    Comment Kommentar
    MDF_FileName FileName
    JfPageNo Seitennummer
    PositionField FeldName
  }
}
    
```

	<i>Parameter</i>	<i>Bedeutung</i>
DynBlock	Blockname	Name des <i>DYNBLOCK-OBJEKTES</i> , das an dieser Stelle definiert wird; dient dazu, sich an anderer Stelle in der ConversionBase auf dieses <i>DYNBLOCK-OBJEKT</i> zu beziehen.
Comment	Kommentar	Kommentartext wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden.
Blocks	BlockName1 BlockName2	Bindet einen der aufgeführten <i>BLOCK-OBJEKTE</i> entsprechend seiner <i>RECOGNITION</i> ein. Die Liste wird von oben nach unten abgearbeitet. D.h., zuerst wird geschaut, ob die Erkennung des ersten aufgeführten <i>BLOCK-OBJEKTES</i> mit der untersuchten Zeichenkette übereinstimmt. Dann wird er mit der <i>RECOGNITION</i> des zweiten <i>BLOCK-OBJEKTES</i> verglichen usw. Sie sollten zuoberst also die <i>BLOCK-OBJEKTE</i> mit den ganz speziellen <i>RECOGNITIONS</i> aufführen, weiter unten die mit den allgemeineren.

StandardBlock	StdBlockName3	Name des <i>BLOCK-OBJEKTES</i> , der aktiviert wird, wenn keiner der <i>BLOCK-OBJEKTE</i> aus der BlockList erkannt wurde oder BlockList nicht definiert ist. Der StandardBlock kann eine eigene <i>RECOGNITION</i> enthalten, die aber unberücksichtigt bleibt.
StartSubForm		Legt fest, welches Anfangs-Designteil eingebunden werden soll, dient dem Einbinden von Tabellenköpfen o.ä. in sich dynamisch aufbauenden Formularteilen.
StoptSubForm		Legt fest, welches Ende-Designteil eingebunden werden soll, dient dem Einbinden von Tabellenfüßen o.ä. in sich dynamisch aufbauenden Formularteilen.
 Anmerkung:		StartSubForm und StopSubForm werden immer aufs Papier gebracht, sobald der <i>DYNBLOCK</i> gelinkt ist, nicht erst, wenn einer der <i>BLOCK-OBJEKTE</i> erkannt ist.
MDF_FileName <i>CommandOrValue</i>	Dateiname	Gibt den Namen der MDF-Datei an, in der das Designteil enthalten ist. Normalerweise handelt es sich dabei um das im <i>PAGE-OBJEKT</i> eingebundenen MDF-File. In diesem Fall kann (sollte) die Angabe dieses Parameters entfallen.
JfPageNo <i>CommandOrValue</i>	Seitennummer	Gibt an, auf welcher Seite in der MDF-Datei sich das Designteil befindet.
PositionField <i>CommandOrValue</i>	Feldname	Gibt an, an welchem Feld des im vorhergehenden Bearbeitungsschritt eingebundenen MDF_Files die linke obere Ecke dieses <i>SUBFORMS</i> platziert werden soll. Das <i>SUBFORM</i> wird automatisch an dem Feld ausgerichtet, in dem zur Laufzeit der Cursor steht, wenn dieser <i>VALUE</i> nicht angegeben wird.



Block



Verwendung: Definiert ein *BLOCK-OBJEKT* und dessen Details



Syntax:

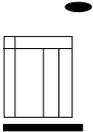
```

Block BlockName {
  Comment „Kommentar“
  Recognition {
    ...
  }
  Lines Zeilenzahl
  Columns Spaltenzahl
  Conversion ConvName
  SubForm {
    Comment „Kommentar“
    MDF_FileName Dateiname
    JfPageNo Seitennummer
    PositionField Feldname
  }
}

```

	<i>Parameter</i>	<i>Bedeutung</i>
Block	Blockname	Name des <i>BLOCK-OBJEKTES</i> , der an dieser Stelle definiert wird; dient dazu, sich an anderer Stelle in der ConversionBase auf dieses <i>BLOCK-OBJEKT</i> zu beziehen.
Comment	Kommentar	Kommentartext wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden
Recognition		wird wegen ihrer Komplexität in einem getrennten Abschnitt behandelt
Lines	Zeilenzahl	Gibt die Länge des Blocks in Zeilen an.
Columns	Spaltenzahl	Gibt die Blocks des Blocks in Spalten an.

Conversion	ConvName	Name des zu verwendenden <i>CONVERSION-OBJEKTES</i> , muss an anderer Stelle in der ConversionBase definiert sein.
SubForm		Legt fest, welches Designteil gerade eingebunden werden soll.
MDF_FileName	Dateiname	Gibt den Namen der MDF-Datei an, in der das Designteil enthalten ist. Normalerweise handelt es sich dabei um das im <i>PAGE-OBJEKT</i> eingebundenen MDF-File. In diesem Fall kann die Angabe dieses Parameters entfallen.
JfPageNo	Seitennummer	Gibt an, auf welcher Seite in der MDF-Datei sich das Designteil befindet
PositionField	Feldname	Gibt an, an welchem Feld des im vorhergehenden Bearbeitungsschritt eingebundenen MDF_Files die linke obere Ecke dieses <i>SUBFORMS</i> platziert werden soll. Das <i>SUBFORM</i> wird automatisch an dem Feld ausgerichtet, in dem zur Laufzeit der Cursor steht, wenn dieser <i>VALUE</i> nicht angegeben wird.



Design



Verwendung: Definiert ein *DESIGN-OBJEKT* und dessen Details. Alle Key-Wörter **IN DESIGN-OBJEKT** können auch mit einem dynamischen Befehl definiert werden.



Syntax:

```

Design DesignName {
  Comment „Kommentar“
  SubForm {
    Comment „Kommentar“
    MDF_FileName Dateiname
    JfPageNo Seitennummer
    PositionField Feldname
  }
  SubForm {
    Comment „Kommentar“
    MDF_FileName Dateiname
    JfPageNo Seitennummer
    PositionField Feldname
  }
  ...
}
    
```

	<i>Parameter</i>	<i>Bedeutung</i>
Design	DesignName	Name des <i>DESIGN-OBJEKTES</i> , das an dieser Stelle definiert wird; dient dazu, sich an anderer Stelle in der ConversionBase auf dieses <i>DESIGN-OBJEKT</i> zu beziehen.
Comment	Kommentar	Kommentartext wenn Leer- und Sonderzeichen enthalten sind, muss der Kommentartext in „“ eingefasst werden.
SubForm		Legt fest, welches Designteil gerade eingebunden werden soll.
MDF_FileName	Dateiname	Gibt den Namen der MDF-Datei an, in der das Designteil enthalten ist. Normalerweise handelt es sich dabei um das im <i>PAGE-OBJEKT</i> eingebundenen MDF-File. In diesem Fall kann (sollte) die Angabe dieses Parameters entfallen.
JfPageNo	Seitennummer	Gibt an, auf welcher Seite in der MDF-Datei sich das Designteil befindet

PositionField	Feldname	<p>Gibt an, an welchem Feld des im vorhergehenden Bearbeitungsschritt eingebundenen MDF_Files die linke obere Ecke dieses <i>SUBFORMS</i> platziert werden soll.</p> <p>Das <i>SUBFORM</i> wird automatisch an dem Feld ausgerichtet, in dem zur Laufzeit der Cursor steht, wenn dieser <i>VALUE</i> nicht angegeben wird.</p> <p>Alle Key-Wörter MDF_FileName, JfPageNo und PositionField können auch mit einem dynamischen Befehl definiert werden.</p>
----------------------	----------	--

SUBSTITUTE	
Such1	Ausg1
Such2	Ausg2
Such3	Ausg3
Such4	Ausg4
Such5	Ausg5

Substitute



Syntax:

```

SUBSTITUTE SubstName {
    SuchString1  AustauschString1
    SuchString2  AustauschString2
    SuchString3  AustauschString3
    ...
    DEFAULT     AustauschStringX
}

```

Das *SUBSTITUTE-OBJEKT* ist ein Objekt, ebenso wie *SYSTEM*, *JOB* oder *FORM*. Das *SUBSTITUTE-OBJEKT* kann mehrfach unter verschiedenen Namen in der ConversionBase vorkommen. Der Inhalt des *SUBSTITUTE-OBJEKTES* ist eine Tabelle, die der Reihenfolge nach, von oben nach unten Austauschpaare definiert. Eine Zeile enthält den SuchString und den AustauschString getrennt durch mindestens ein Leerzeichen. Daraus folgt, dass Leerzeichen im SuchString und im AustauschString nicht erlaubt sind.

Der SuchString DEFAULT ist reserviert. Der AustauschString von DEFAULT wird verwendet, wenn kein passender SuchString in den vorhergehenden Zeilen gefunden wurde. (Beispiel 1) Falls kein DEFAULT beim SuchString definiert wurde und die Tabelle ist durchlaufen ohne einen gültigen AustauschString zu finden, wird ein Leerzeichen übergeben.

Anstelle fixer Bezeichnungen und Texte kann auch ein Verweis auf eine Variable stehen. Dies erkennt man am ersten Zeichen des Parameters. Steht an dieser Stelle ein \$ so wird der folgende Text als der Variablenname interpretiert, indem der Ersatztext steht. (Beispiel 2)

Der Aufruf eines *SUBSTITUTE-OBJEKTES* erfolgt als *COMMAND* SUBSTITUTE oder als Funktionsaufruf GET_SUBSTITUTE. Wird das SUBSTITUTE-Objekt als Funktion aufgerufen, wird hier dem Aufruf ein @ vorangestellt. (Beispiel 3 und 4)

Aufruf: **SUBSTITUTE** VarName SubstName ZielVarName

SUBSTITUTE ist ein *COMMAND*-Element, welches im der *VAREXEC*-Element und in dem *CONVERSION-OBJEKT* arbeitet aber auch überall wo Variablen eingesetzt werden können. QuellVarName ist der Name einer bereits definierten Variablen, mit der die Konvertierungs-routine in die angegebene SubstTabelle geht die Konvertierung ausführt, das Ergebnis wird in ZielVarName geschrieben.

GET_SUBSTITUTE VarName SubstName

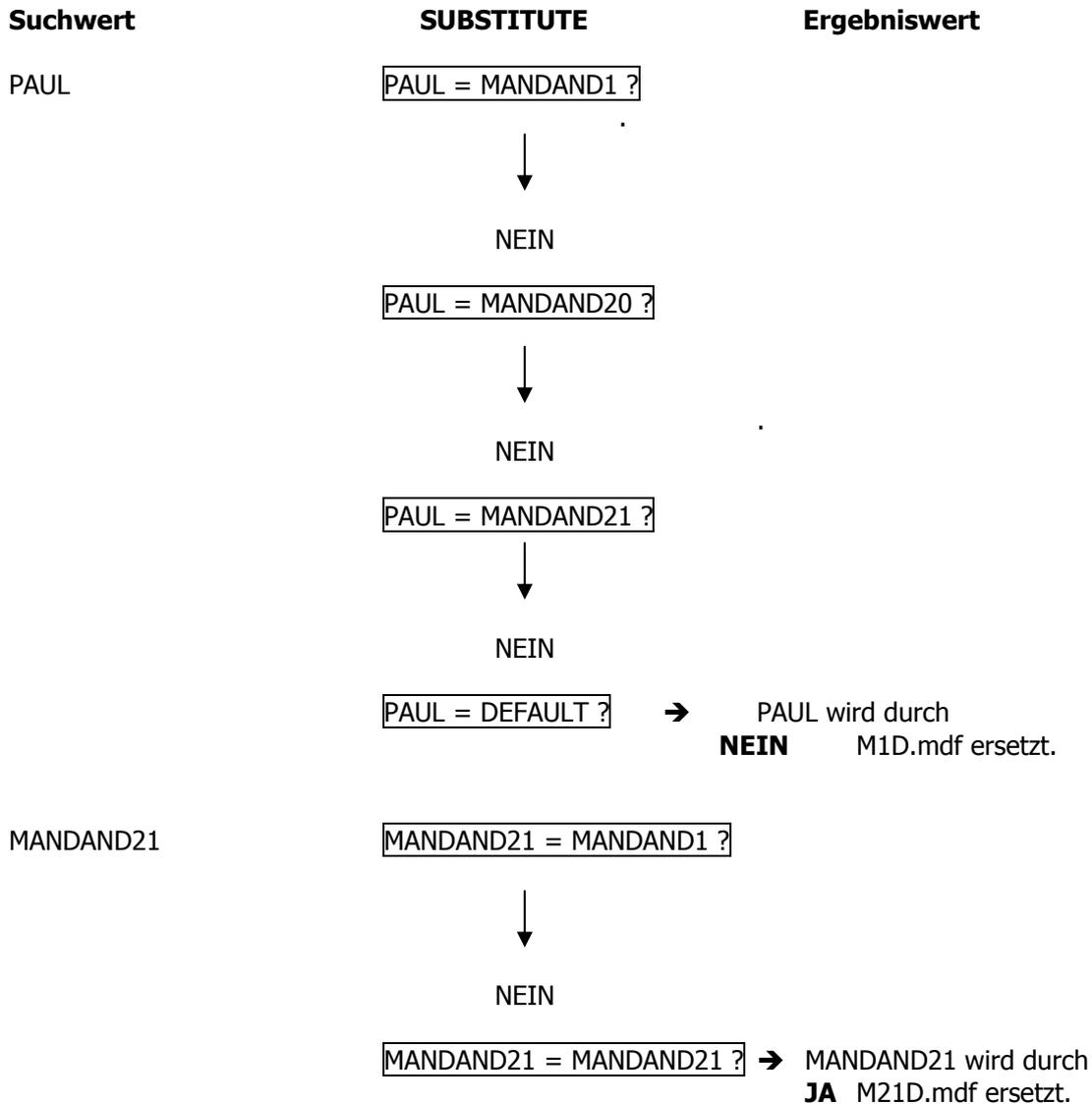
GET_SUBSTITUTE ist ein Get-Befehl und arbeitet im *CONVERSION-OBJEKT*.



Beispiel 1

```
SUBSTITUTE FormsConvert {  
    MANDAND1      M1D.mdf  
    MANDAND20     M20D.mdf  
    MANDAND21     M21D.mdf  
    DEFAULT       M1D.mdf  
}
```

Wie wird die Substitute Tabelle abgearbeitet?



Sollte kein SuchString DEFAULT definiert sein wird ein Leerzeichen übergeben.



Beispiel 2

GET_SUBSTITUTE VarNameIn \$TableVar

Ist die Variable TableVar mit dem Wert „FormTableD“ belegt, so erfolgt der Austausch von VarNameIn über die Substitute-Tabelle mit dem Namen FormTableD.



Beispiel 3

```
Substitute RE_FIRMENDESIGN {  
  0001    @SUBSTITUTE (RE_ZUSATZDESIGN, VDPR_WERK)  
  DEFAULT RE_DATEN,10;RE_FIRMEN_DESIGN,36  
}
```

```
Substitute RE_ZUSATZDESIGN {  
  1000    RE_DATEN,10;RE_FIRMEN_DESIGN,16;RE_ZUS,9  
  DEFAULT RE_DATEN,10;RE_FIRMEN_DESIGN,16;RE_ZUS,8  
}
```



Beispiel 4

```
FormName „@GET_SUBSTITUTE SPRACHE FORM_SELECT“
```

```
SUBSTITUTE FORM_SELECT {  
  D        RE_D.MDF  
  E        RE_E.MDF  
  DEFAULT  RECHNUNG:MDF  
}
```

VarExec

VarExec ist eine Liste von Befehlen, die von oben nach unten abgearbeitet wird. Jeder einzelne Befehl kann Variablen lesen und schreiben und verändert damit die Variablen-Tabelle. VarExec-Kommandos liefern niemals einen Wert zurück. Das Ergebnis eines VarExec-Kommandos kann nur eine veränderte oder neu angelegte Variable sein.



Syntax:

```

VarExec {
    ...
    Kommando
    ...
}

```

Folgende Kommandos sind unterstützt:

Addi	VarNameS1 VarNameS2
Subi	VarNameS1 VarNameS2
Concat	OutVar InVar1 [InVar2 [...]]
Convert_Vars_In_File	OutVar FileName InVar0 InVar1 InVar2...
Convert_Vars_In_File_If_Known	OutVar FileName InVar0 InVar1 InVar2...
Convert_Vars_In_File_If_Unknown	OutVar FileName InVar0 InVar1 InVar2...
Del_All_Vars	
Del_Var	VarName
Exit	Message
Fill	VarName Ori Size [FillChar]
Read_Vars_From_File	FileName
Write_Vars_To_File	FileName
Set_Var	VarName yPos xPos xDim
Set_Var_If_Unknown	VarName yPos xPos xDim
Set_Var_Fix	VarName „Fixer Text“
Set_Var_Fix_If_Unknown	VarName „Fixer Text“
Strip	InVar [Orientation [Char]]
Substitute	InVarName Tabelle OutVarName
Token	InVar Delimiter OutVar1 OutVar2 [...]

Concat



Verwendung: Concat verkettet den Inhalt der unter InVarN angegebenen Variablen und schreibt das Ergebnis in die OutVar Variable.



Syntax:

```
Concat          OutVar InVar1 [InVar2 [...]]
```

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
OutVar	Name der Zielvariable
InVarN	Namen der Variablen, die zu seinem Gesamtstring zusammengefasst werden sollen



Beispiel:

```
VarExec {  
    ...  
    SET_VAR_FIX S1 „Seite: „  
    SET_VAR S2 11 12 3  
    CONCAT PAGE S1 S2  
    ...  
}
```

Im Beispiel könnte das Ergebnis sein (PAGE): „Seite: 3“

Convert_Vars_In_File



Verwendung: Convert_Vars_In_File konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle.



Syntax:

Convert_Vars_In_File OutVar FileName InVar0 InVar1 InVar2...

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
OutVar	Name der Zielvariable
FileName	Name des Conversion-Files
InVarN	Variablen, die zu Konvertierung herangezogen werden

Convert_Vars_In_File_If_Known



Verwendung: Convert_Vars_In_File_If_Known konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle. Der Befehl wird nur ausgeführt, wenn in der Variable OutVar vor der Konvertierung bereits ein Inhalt vorhanden war.



Syntax:

```
Convert_Vars_In_File_If_Known OutVar FileName InVar1 [InVarN]
```

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
OutVar	Name der Zielvariable
FileName	Name des Conversion-Files
InVarN	Variablen, die zu Konvertierung herangezogen werden

Convert_Vars_In_File_If_Unknown



Verwendung: Convert_Vars_In_File_If_Unknown konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle. Der Befehl wird nur ausgeführt, wenn in der Variable OutVar vor der Konvertierung kein Inhalt vorhanden war.



Syntax:

```
Convert_Vars_In_File_If_Unknown OutVar FileName InVar1 [InVarN]
```

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
OutVar	Name der Zielvariable
FileName	Name des Conversion-Files
InVarN	Variablen, die zu Konvertierung herangezogen werden

Del_All_Vars



Verwendung: Del_All_Vars löscht alle Variablen der Variablen-Tabelle.



Syntax:

Del_All_Vars

Del_Var



Verwendung: Del_Var löscht die angegebene Variable aus der Variablen-Tabelle.



Syntax:

Del_Var VarName

Parameter
VarName

Bedeutung/Bemerkung
Name der zu löschenden Variable

Exit



Verwendung: Exit verlässt das Programm mit der Fehlernummer -900 und gibt im Log- und Response-File die angegebene Fehlermeldung mit.



Syntax:

Exit Message

Parameter
Message

Bedeutung/Bemerkung
Abbruchnachricht

Fill



Verwendung: Fill füllt den Inhalt der unter VarName angegebenen Variable mit FillChar-Zeichen auf. Ist FillChar nicht definiert, so wird Blank als FillChar-Zeichen verwendet. Das Ergebnis wird in die selbe Variable zurückgeschrieben.



Syntax:

Fill VarName Ori Size [FillChar]

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der In- und Out-Variable
Ori	Orientation (LEFT oder RIGHT)
Size	Neue Größe des Strings
FillChar	Füllbuchstabe (Standard ist Blank)

Read_Vars_From_File



Verwendung: Liest die Variablen, die im angegebenen File definiert sind zu den bereits existierenden Variablen hinzu oder überschreibt diese.



Syntax:

Read_Vars_From_File FileName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
FileName	Name des Variablen-Files

Write_Vars_To_File



Verwendung: Schreibt alle Variablen in den angegebenen File. Ein eventuell existierender File gleichen Namens wird dabei überschrieben.



Syntax:

Write_Vars_To_File FileName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
FileName	Name des Variablen-Files

Set_Var



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der auf der Seite an der angegebenen Position steht.



Syntax:

Set_Var VarName yPos xPos xDim

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu überschreibenden oder anzulegenden Variable
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
xDim	Anzahl der Spalten, die von xPos beginnend gelesen werden sollen

Set_Var_If_Unknown



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der auf der Seite an der angegebenen Position steht. Der Befehl wird nur ausgeführt, wenn in der variable VarName vor dem Setzen kein Inhalt vorhanden war.



Syntax:

Set_Var_If_Unknown VarName yPos xPos xDim

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu überschreibenden oder anzulegenden Variable
yPos	Zeile, in der das Lesen begonnen werden soll
xPos	Spalte, in der das Lesen begonnen werden soll
xDim	Anzahl der Spalten, die von xPos beginnend gelesen werden sollen

Set_Var_Fix



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der Fix als Text angegeben ist.



Syntax:

Set_Var_Fix VarName „Fester Text“

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu überschreibenden oder anzulegenden Variable
FixText	Fester Text

Set_Var_Fix_If_Unknown



Verwendung: Setzt oder überschreibt die angegebene Variable mit dem Wert, der Fix als Text angegeben ist. Der Befehl wird nur ausgeführt, wenn in der Variable VarName vor dem Setzen kein Inhalt vorhanden war.



Syntax:

Set_Var_Fix_If_Unknown VarName „Fixer Text“

Parameter
VarName
FixText

Bedeutung/Bemerkung
Name der zu überschreibenden oder anzulegenden Variable
Fester Text

Strip



Verwendung: Strip beseitigt nicht gewollte Zeichen aus einer Variable.



Syntax:

Strip VarName [Orientation [Char]]

Parameter
VarName
Orientation
Char

Bedeutung/Bemerkung
Name der In- und OutVar
Richtung des Strips (BOTH, LEFT und RIGHT)
Zeichen das mit Strip beseitigt werden soll (Standard Blank)



Beispiel 1: **VarExec** {

```

...
SET_VAR_FIX S2 „ Hallo World „
STRIP S2 BOTH
...
}
```

Das Ergebnis der einzelnen Operationen wäre z.B. „Hallo World“.



Beispiel 2: **VarExec** {

```

...
SET_VAR_FIX S2 „00005645“
STRIP S2 LEFT 0
...
}
```

Das Ergebnis der einzelnen Operationen wäre z.B. „5645“.

Substitute



Verwendung: Substitute konvertiert den Wert einer Variablen in einer Substitution-Tabelle und schreibt das Ergebnis in eine Variable.



Syntax:

Substitute InVar Tabelle OutVar

Parameter

InVar

Tabelle

OutVar

Bedeutung/Bemerkung

Name der Variable mit der in einer Substitution-Tabelle ein Austausch durchgeführt werden soll

Name der Substitution-Tabelle

Zielvariable, in die das Ergebnis geschrieben werden soll

Token



Verwendung: Token unterteilt den Inhalt einer Variablen mit Hilfe des Delimiter-Trennzeichens in Teilstrings und gibt diese in die angegebenen OutVars aus.



Syntax:

```
Token          InVar Delimiter OutVar1 OutVar2 [...]
```

Parameter
InVar
Delimiter
OutVarN

Bedeutung/Bemerkung
Name der Variable die aufzugliedern ist
Trennzeichen
Zielvariablen für die Teilstrings



Beispiel 1: **VarExec** {

```
...
SET_VAR_FIX S1 „Hallo World“
TOKEN S1 „ „ R1 R2 R3
...
}
```

Das Ergebnis der einzelnen Operationen wäre:

```
R1 = „Hallo“
R2 = „World“
R3 = „“
```



Beispiel 2: **VarExec** {

```
...
SET_VAR_FIX S1 „ABT12#Herr H. Wolf#PRT12#A1“
TOKEN S1 „ „ ORG_ID NAME PRINTER
...
}
```

Das Ergebnis der einzelnen Operationen wäre:

```
ORG_ID = „ABT12“
NAME = „Herr H. Wolf“
PRINTER = „PRT12“
```

Der Wert „A1“ am Ende der Zeichenkette würde ignoriert.



CommandOrValue

CommandOrValue-Angaben ist eine Möglichkeit zur Definition von Schlüsselwerten in den Objekten der ConversionBase. Schlüsselwörter die vom Type CommandOrValue sind, wurden in diesem Handbuch mit dem Zusatz „CommandOrValue“ gekennzeichnet. Unabhängig davon, welche Werte der Schlüssel annehmen kann, definiert CommandOrValue woher die Werte gelesen werden. Entweder der Wert für den Schlüssel wird fest in der ConversionBase angegeben (eine Value-Definition) oder es wird ein Kommando definiert, dass zur Laufzeit den Wert des Schlüssels ermittelt (eine Command-Definition). Zur Unterscheidung von Value- und Command-Definition beginnen alle Kommandos mit dem Zeichen „@“.



Beispiel Value-Definition

```
Page Seite {
    ...
    Printer Drucker1
    ...
}
```



Beispiel Command-Definition

```
Page Seite {
    ...
    Printer @GET_VAR PrinterVar
    ...
}
```

Auf die Value-Definition müssen wir hier nicht weiter eingehen. Schreiben Sie einfach hinter das Schlüsselwort den Wert. Interessanter ist die Command-Definition und vor allem die Auflistung und Beschreibung möglicher Kommandos zur Ermittlung des Wertes. Folgende Kommandos sind erlaubt:

@Read	yPos xPos yDim xDim
@Read_Minimized	yPos xPos yDim xDim
@Take	yPos xPos yDim xDim
@Take_Minimized	yPos xPos yDim xDim
@Get_Var	VarName
@Get_Convert_Vars_In_File	FileName InVar0 [InVar1 [...]]
@Get_Substitute	VarName SubstName

@Read @Read_Minimized



Verwendung: @Read liest von der aktuellen Seite einen Textblock.
@Read_Minimized liest einen Textblock von der aktuellen Seite und führt ein Trim auf den Block aus



Syntax:

@Read yPos xPos yDim xDim
@Read_Minimized yPos xPos yDim xDim

Parameter

yPos
xPos
yDim
xDim

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll
Spalte, in der das Lesen begonnen werden soll
Anzahl der Zeilen, die von yPos beginnend gelesen werden sollen
Anzahl der Spalten, die von xPos beginnend gelesen werden sollen

@Take @Take_Minimized



Verwendung: @Take liest von der aktuellen Seite einen Textblock und löscht ihn dort.
@Read_Minimized liest einen Textblock von der aktuellen Seite, löscht ihn dort und führt ein Trim auf den Block aus



Syntax:

@Take yPos xPos yDim xDim
@Take_Minimized yPos xPos yDim xDim

Parameter

yPos
xPos
yDim
xDim

Bedeutung/Bemerkung

Zeile, in der das Lesen begonnen werden soll
Spalte, in der das Lesen begonnen werden soll
Anzahl der Zeilen, die von yPos beginnend gelesen werden sollen
Anzahl der Spalten, die von xPos beginnend gelesen werden sollen

@Get_Var



Verwendung: @Get_Var ermittelt den Wert einer Variable.



Syntax:

@Get_Var VarName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der Variable

@Get_Convert_Vars_In_File



Verwendung: @Get_Convert_Vars_In_File konvertiert die Liste der angegebenen Variablen in einem Conversion-File. Eine genaue Beschreibung finden Sie unter im Kapitel Konvertierungstabelle.



Syntax:

@Get_Convert_Vars_In_File FileName InVar0 [InVar1 [...]]

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
FileName	Name des Conversion-Files
InVarN	Variablen, die zu Konvertierung herangezogen werden

@Get_Substitute



Verwendung: @Get_Substitute konvertiert eine Variable in einer Substitute-Tabelle.



Syntax:

@Get_Substitute VarName SubstName

<i>Parameter</i>	<i>Bedeutung/Bemerkung</i>
VarName	Name der zu konvertierenden Variable
SubstName	Name der Substitute-Tabelle

Konvertierungstabelle

Es wurde eine Konvertierungstabelle entwickelt, in der mehrere Inputwerte zu einem Outputwert konvertiert werden. Die Tabelle der Konvertierungsdatei besteht aus einer Liste von Vektoren mit dem folgenden Aufbau:

OutputWert;InputWert0[;InputWert1[;InputWert2[;...]]]

Wobei der OutputWert entweder ein fester Wert ist oder ein *LINK* auf eine Variable. Der OutputWert, der mit dem Zeichen @ beginnt und dann von einer Zahl gefolgt wird, verkörpert einen *LINK* auf den InputWert mit der entsprechenden Nummer. Dabei ist der erste InputWert der InputWert0 und wird demnach als @0 referenziert. Alle folgenden InputWerte haben die entsprechend größere Nummer beginnend bei InputWert0.

Da in der Tabelle mit Wildcards gearbeitet werden kann, sind Mehrdeutigkeiten nicht zu vermeiden. Der Algorithmus wichtet alle Treffervektoren so, dass der Vektor mit der genauesten Angabe bzw. mit der größten Wichte gewinnt. Der Algorithmus bestimmt zunächst ob ein Eintrag in der Konvertierungstabelle ein Treffer ist und welche Wichte die Aussage besitzt. Zur Ermittlung der Wichte werden alle InputWerte mit einer Wichtezahl versehen, die der 2er Potenz der InputWerte entsprechen.

InputWert0 = $2^0 = 1$; InputWert1 = $2^1 = 2$; InputWert2 = $2^2 = 4$; InputWert3 = $2^3 = 8$; usw.
Stimmt der InputWert exakt mit dem Eintrag der Konvertierungstabelle überein, wird die Wichte des Inputwertes zur Wichte des Vektors addiert. Ist der InputWert über ein Wildcard gültig, wird die Wichte nicht zur Wichte des Vektors addiert.

Beispiel:

<u>Suchvektor</u>		
InputWert0	InputWert1	InputWert2
(Mandant)	(PC)	(Person)
110	WST03	Müller

<u>Vektor der Konvertierungsdatei</u>				
InputWert0	InputWert1	InputWert2	Treffer	Wichte
(Mandant)	(PC)	(Person)		
120	WST01	Müller	0	
110	*	*	1	1
110	WST03	*	1	3
110	*	Müller	1	5
110	WST03	Müller	1	7
*	*	*	1	0
*	WST03	*	1	2
*	*	Müller	1	4
*	WST03	Müller	1	6

Die Tabelle der Konvertierungsdatei besteht aus 8 Treffern und einen Nicht-Treffer.

Der Algorithmus würde den Vektor

110 WST03 Müller auswählen, da er die größte Wichte hat.

Angesprochen wird die Liste über die Befehle

CONVERT_VARS_IN_FILE	OutVar FileName InVar0 InVar1 InVar2 ...
CONVERT_VARS_IN_FILE_IF_KNOWN	OutVar FileName InVar0 InVar1 InVar2 ...
CONVERT_VARS_IN_FILE_IF_UNKNOWN	OutVar FileName InVar0 InVar1 InVar2 ...
GET_CONVERT_VARS_IN_FILE	FileName InVar0 InVar1 InVar2 ...

All diese Befehle können in der *VAREXEC* von *JOB* und *FORM* auftreten.

CONVERT_VARS_IN_FILE_IF_UNKNOWN prüft lediglich vor seiner Ausführung ab, ob die Ausgabevariable leer ist und macht dies zur Voraussetzung ob die Konvertierung durchgeführt wird oder nicht.

CONVERT_VARS_IN_FILE_IF_KNOWN prüft lediglich vor seiner Ausführung ab, ob die Ausgabevariable mit einem Wert belegt ist und macht dies zur Voraussetzung ob die Konvertierung durchgeführt wird oder nicht.

GET_CONVERT_VARS_IN_FILE schreibt den OutWert nicht in eine Variable sondern liefert diese als Rückgabewert und kann damit nur an den Stellen stehen, wo auch **GET_VAR** erlaubt ist.

Index

@Get_Convert_Vars_In_File.....	98, 100
@Get_Substitute.....	98, 101
@Get_Var.....	98, 100
@Read.....	98, 99
@Read_Minimized.....	98, 99
@Take.....	98, 99
@Take_Minimized.....	98, 99
Addi.....	46, 59, 84, 85
AND	
Recognition.....	42
siehe auch Recognition.....	15, 40, 42, 43
Aufrufkonventionen.....	6
Ausgabeseite.....	10, 11
Ausgabestrom.....	11
b	
Inline.....	22
b0	
Inline.....	22
Block.....	12, 13, 15, 16, 18, 36, 72, 73
Columns.....	73
Comment.....	73
Conversion.....	73
JfPageNo.....	73
Lines.....	73
MDF_FileName.....	73
PositionField.....	73
Recognition.....	73
siehe auch Link.....	16
SubForm.....	73
Block.....	46
Block.....	48
BlockName	
Block.....	70, 73
Blocks.....	11, 12, 13, 15, 70, 71, 73
DynBlock.....	70
siehe auch ChoiceList.....	15
ChoiceList.....	9, 15, 16, 18
Columns	
Block.....	73
Command	
Job.....	15, 16, 18, 26, 30
CommandOrValue.....	97
Commands.....	8, 11, 13, 15, 16, 42
Comment.....	46, 48
Block.....	73
Conversion.....	21
Design.....	76
DynBlock.....	70
Form.....	30
Page.....	35
StartSubForm.....	70
StopSubForm.....	70
SubForm.....	73
System ...	17, 18, 21, 23, 26, 28, 30, 35, 70, 73, 76
Concat.....	64, 84, 87
Concat.....	46
Contains_Not_Text_In	
Recognition.....	40, 41
Contains_Text_In	
Recognition.....	40, 41
Conversion.....	10
Conversion ..	8, 10, 11, 12, 13, 16, 18, 23, 32, 35, 36, 45, 73, 74
Block.....	73
Page.....	35
siehe auch Link.....	16
ConversionBase...4, 7, 8, 9, 15, 16, 19, 21, 26, 27, 30, 35, 36, 70, 73, 74, 76	
*.cvb.....	7
Convert_Vars_In_File.....	47, 67, 84, 88
Convert_Vars_In_File_If_Known.....	47, 68, 84, 89
Convert_Vars_In_File_If_Unknown.....	47, 68, 84, 89
ConvertUnderlines	
System.....	21, 23
Copy.....	46, 57
Page.....	35
Del_All_Vars.....	84
Del_ALL_Vars.....	46
Del_Var.....	46, 62, 84, 90
Design....	10, 11, 13, 16, 18, 32, 35, 36, 75, 76
Comment.....	76
JfPageNo.....	76
MDF_FileName.....	76
Page.....	35
PositionField.....	76
siehe auch Link.....	16
SubForm.....	76
Druckdaten	
plain-ASCII.....	4, 5, 7, 8, 9, 26
Steuerzeichen.....	4
Duplex	
Page.....	35, 37
Duplexdruck.....	10, 16
Durchschlag.....	10, 33, 35
DynBlock.....	46, 49
Blocks.....	70

Comment.....	70	Insert.....	46, 58
Datenfenster	13	Insert_Var	47, 66
JfPageNo	70	Intray.....	6, 10, 35, 37
MDF_FileName	70	Page.....	35
PositionField.....	70	Is_Not_Text_At	
siehe auch Link.....	12, 13, 16, 18, 36, 69, 70, 71, 73	Recognition	40, 41
StandardBlock	70	Is_Text_At	
StartSubForm	70	Recognition	28, 40, 43, 44
StoptSubForm	70	Is_Text_At_Eql_Var	
DynBlocks.....	11, 12, 13, 70	Recognition	40, 41
Eingangsdaten		Is_Var_Eql	
Zerlegen der	13	Recognition	40
Eingangsdatenstrom	11	Is_Var_Known	
Elemente		Recognition	40
ConversionBase	7, 15, 16, 19	Is_Var_Not_Eql	
EndOfJob		Recognition	40
Form	30, 32	Is_Var_Not-Eql	
Erkennung		Recognition	40
siehe auch Recognition .	7, 9, 10, 12, 15, 21, 27, 70	Is_Var_Unknown	
Exit	47, 67, 84, 90	Recognition	40, 41
FeldName		JF_Command.....	46, 58
Read_Field.....	70	JfPageNo	
Fill	47, 66, 84, 91	Block.....	73
Form... 7, 9, 11, 12, 13, 15, 18, 19, 27, 29, 30, 32, 33, 36		Design	76
Comment.....	30	StartSubForm	70, 71, 73, 74, 76
EndOfJob.....	30	StopSubForm	70
Lines	30	SubForm	73
NewSpoolFile.....	30	Job 7, 9, 10, 15, 16, 18, 19, 21, 22, 25, 26, 28, 36	
Pages	30	Comment.....	26, 30
Recognition.....	30	Forms.....	26
Sort.....	30	Recognition	26
FormName		StandardForm	26
Page.....	30, 32, 35, 36	JobName	
Forms		Page.....	26, 32, 35, 36
Job.....	26	Jobs	
siehe auch ChoiceList.....	26, 27, 28, 30	siehe auch ChoiseList..	9, 10, 13, 15, 21, 22, 23, 26
Formularerkennung		System	21
siehe auch Recognition	9, 10	Kill	46, 57
Get_Substitute.....	46, 63	Konvertierungstabelle	102
i		Layout.....	10, 11, 16
Inline.....	22	Lines	
i0		Block	73
Inline.....	22	Form	30, 33, 73
Inline		Link.....	9, 12, 16, 18, 32
System	21, 22, 23	MDF_FileName	
<u>Input-Daten</u>	7, 9, 33	Block	73
Input-Datenstrom.....	Siehe Input-Daten, Siehe Input-Daten	Design	76
InputFile.....	6	StartSubForm	70, 71, 73, 74, 76
Input-Strom		StopSubForm	70
siehe Input-Daten.....	16	SubForm	73
		Move.....	46, 56
		NAND	

Recognition	42	Set_Var	46, 60, 84, 92
siehe auch Recognition	15, 42	Set_Var_Fix	46, 61, 84, 93
NewSpoolFile		Set_Var_Fix_If_Unknown	61, 84, 94
Form	30, 32	Set_Var_Fix_If_Unknown	46
NOR		Set_Var_If_Unknown	46, 60, 84, 93
Recognition	42	Sort	
siehe auch Recognition	15, 42	Form	30, 32
ObjektList	9, 15, 18	Spalte	
Operator		Recognition	27
Job	26	SpoolFile	9
OptionenListe	6	StandardBlock	
OR		DynBlock	70
Recognition	42	siehe auch Link	16, 70, 71
siehe auch Recognition ...	15, 28, 40, 42, 43, 44	StandardForm	
Outtray	10, 35, 37	Job	26
Page	35	siehe auch Link	16, 26, 27, 28
Page	10, 11, 13, 18, 33, 34, 35, 71, 74, 76	StandardJob	
Comment	35	siehe auch Link	16, 21, 22, 23
Conversion	35	System	21
Copy	35	StartSubForm	
Design	35	DynBlock	36, 70, 71
Duplex	35	StopSubForm	
FormName	35	DynBlock	36, 70, 71
Intray	35	Strip	47, 65, 84, 94
JobName	35	SubForm	16
Outtray	35	Block	16, 18, 32, 36, 71, 73, 74, 76, 77
Printer	35	Design	76
Pages		Subi	46, 59, 84, 86
Form	30	Substitute	46, 64, 78, 84, 95
siehe auch ObjektList	11, 15, 30	Suchfenster	
PositionField		DynBlock	13, 70
Block	73	System	8, 9, 10, 18, 19, 20, 21, 23, 24, 33
Design	76	Comment	21
StartSubForm	70, 71, 73, 74, 76, 77	ConvertUnderlines	21
StopSubForm	70	Inline	21
SubForm	73	Jobs	21
Printer		StandardJob	21
Page	35, 37	Take	46, 53
Read	49	Take_Field	46, 54
Read_Field	46, 51	Take_Field_Minimized	46, 55
Read_Field_Minimized	46, 52	Take_Global	46, 54
Read_Global	51	Take_Global_Minimized	46, 56
Read_Global_Minimized	52	Take_Minimized	46, 53
Read_Minimized	46, 50	TCI-File	6
Read_Vars_From_File	62, 84, 91	Token	46, 65, 84, 96
Read_Vars_From_File	46	ul	
Recognition ..	8, 13, 15, 18, 21, 22, 26, 27, 28, 30, 39, 40, 43, 44, 70, 71, 73	Inline	22
Block	73	ul0	
Form	30	Inline	22
Is_Text_At	30	Value	8, 16, 18, 71, 74, 77
Schriftattribute		VarExec	46, 83, 84
Inline	22, 23	Write_Vars_To_File	46, 63, 84, 92
		Zeichenkette	
		Is_Text_At	15, 70

Zeile

Recognition 27