ORACLE
INSURANCE

Oracle® Documaker

Rules Reference

version 11.4

Part number: E14902-01

October 2009

THIRD PARTY SOFTWARE NOTICES

This product includes software developed by Apache Software Foundation (http://www.apache.org/).

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software distributed via the Berkeley Software Distribution (BSD) and licensed for binary distribution under the Generic BSD license.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software developed by the JDOM Project (http://www.jdom.org/).

THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE JDOM AUTHORS OR THE PROJECT CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software developed by the Massachusetts Institute of Technology (MIT).

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This product includes software developed by Jean-loup Gailly and Mark Adler. This software is provided 'as-is', without any express or implied warranty. In no event will the authors be held liable for any damages arising from the use of this software.

This software is based in part on the work of the Independent JPEG Group (http://www.ijg.org/).

This product includes software developed by the Dojo Foundation (http://dojotoolkit.org).

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 2005-2009, The Dojo Foundation. All rights reserved.

This product includes software developed by W3C.

Copyright © 2009 World Wide Web Consortium, (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved. (http://www.w3.org/Consortium/Legal/)

This product includes software developed by Mathew R. Miller (http://www.bluecreststudios.com).

Copyright (c) 1999-2002 ComputerSmarts. All rights reserved.

This product includes software developed by Shaun Wilde and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

This product includes software developed by Chris Maunder and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

This product includes software developed by PJ Arends and distributed via Code Project Open License (http://www.codeproject.com).

THIS WORK IS PROVIDED "AS IS", "WHERE IS" AND "AS AVAILABLE", WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES OR CONDITIONS OR GUARANTEES. YOU, THE USER, ASSUME ALL RISK IN ITS USE, INCLUDING COPYRIGHT INFRINGEMENT, PATENT INFRINGEMENT, SUITABILITY, ETC. AUTHOR EXPRESSLY DISCLAIMS ALL EXPRESS, IMPLIED OR STATUTORY WARRANTIES OR CONDITIONS, INCLUDING WITHOUT LIMITATION, WARRANTIES OR CONDITIONS OF MERCHANTABILITY, MERCHANTABLE QUALITY OR FITNESS FOR A PARTICULAR PURPOSE, OR ANY WARRANTY OF TITLE OR NON-INFRINGEMENT, OR THAT THE WORK (OR ANY PORTION THEREOF) IS CORRECT, USEFUL, BUG-FREE OR FREE OF VIRUSES. YOU MUST PASS THIS DISCLAIMER ON WHENEVER YOU DISTRIBUTE THE WORK OR DERIVATIVE WORKS.

This product includes software developed by Erwin Tratar. This source code and all accompanying material is copyright (c) 1998-1999 Erwin Tratar. All rights reserved.

THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY. USE IT AT YOUR OWN RISK! THE AUTHOR ACCEPTS NO LIABILITY FOR ANY DAMAGE/LOSS OF BUSINESS THAT THIS PRODUCT MAY CAUSE.

This product includes software developed by Sam Leffler of Silicon Graphics.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL SAM LEFFLER OR SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL, INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE

Copyright (c) 1988-1997 Sam Leffler
Copyright (c) 1991-1997 Silicon Graphics, Inc.

This product includes software developed by Guy Eric Schalnat, Andreas Dilger, Glenn Randers-Pehrson (current maintainer), and others. (http://www.libpng.org)

The PNG Reference Library is supplied "AS IS". The Contributing Authors and Group 42, Inc. disclaim all warranties, expressed or implied, including, without limitation, the warranties of merchantability and of fitness for any purpose. The Contributing Authors and Group 42, Inc. assume no liability for direct, indirect, incidental, special, exemplary, or consequential damages, which may result from the use of the PNG Reference Library, even if advised of the possibility of such damage.

This product includes software components distributed by the Cryptix Foundation.

THIS SOFTWARE IS PROVIDED BY THE CRYPTIX FOUNDATION LIMITED AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CRYPTIX FOUNDATION LIMITED OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

Copyright © 1995-2005 The Cryptix Foundation Limited. All rights reserved.

This product includes software components distributed by Sun Microsystems.

This software is provided "AS IS," without a warranty of any kind. ALLEXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANYIMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN AND ITS LICENSORS SHALL NOT BELIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THE SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Copyright (c) 1998 Sun Microsystems, Inc. All Rights Reserved.

This product includes software components distributed by Dennis M. Sosnoski.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS'" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2003-2007 Dennis M. Sosnoski. All Rights Reserved

It also includes materials licensed under Apache 1.1 and the following XPP3 license

THIS SOFTWARE IS PROVIDED "AS IS'" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright © 2002 Extreme! Lab, Indiana University. All Rights Reserved

---

This product includes software components distributed by CodeProject. This software contains material that is © 1994-2005 The Ultimate Toolbox, all rights reserved.

This product includes software components distributed by Geir Landro.

Copyright © 2001-2003 Geir Landro (drop@destroydrop.com) JavaScript Tree - www.destroydrop.com/hjavascripts/tree/version 0.96

This product includes software components distributed by the Hypersonic SQL Group.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE

This product includes software components distributed by the International Business Machines Corporation and others.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

This product includes software components distributed by the University of Coimbra.

University of Coimbra distributes this software in the hope that it will be useful but DISCLAIMS ALL WARRANTIES WITH REGARD TO IT, including all implied warranties of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. In no event shall University of Coimbra be liable for any special, indirect or consequential damages (or any damages whatsoever) resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

This product includes software components distributed by Steve Souza.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS"' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

This product includes software developed by the OpenSymphony Group (http://www.opensymphony.com/.)"

Contents

Chapter 4, Adding Section and Field Rules

Chapter 5, Section and Field Rules Reference

Appendix B, Using Image Editor to Enter Rule Information

Chapter 1

Introduction

Welcome to the Rules Reference for Oracle Documaker. This guide serves as a reference to the various rules you can use to control how the system handles jobs, form sets, sections (images), and fields.

This chapter discusses the following topics:

Rules Overview

You can use rules to control how information is merged onto forms, how that information is then processed, and how the information and those forms are output. This guide serves as a reference to those rules.

Documaker Server uses resources you create using Documaker Studio or the older tool, Image Editor, to process information and forms. This processing includes merging external data onto forms, processing data according to rules you set up, creating print-ready files, archiving data and forms, and, if applicable, sending incomplete forms to Documaker Workstation for completion by a user.

Forms can be completed using Documaker Workstation when user input is required or, if all of your information can be extracted from external data sources, Documaker Server can be set up to process forms without requiring user input.

Documaker Server can create print-ready files for a variety of printer languages including AFP, PostScript, PCL, and Xerox Metacode printers. In addition, the system can also produce output in Adobe Acrobat PDF format.

Types of Rules

The GenData program processes these types of rules, based on this hierarchy:

- Job level rules (level 1)

  These rules define actions the system should perform for each job or work activity, such as producing a complete form set. Job level rules are global rules used to apply procedures and rules to all jobs, form sets, and forms. Most of these rules are designed to initialize, open, and close section (FAP) files, bitmap files, and data files; however, some specialized functions do exist.

  Job level rules are stored in the Job Definition Table (AFGJOB.JDT). For more information on job level rules, see Adding Job and Form Set Rules on page 5.

- Form set level rules (level 2)

  These rules let you construct and manipulate forms into form sets. Form set level rules affect the form set as a whole, not the individual components which make up the form set.

  Form set level rules are stored in the Job Definition Table (AFGJOB.JDT). For more information on job level rules, see Adding Job and Form Set Rules on page 5.

- Section level rules (level 3)

  These rules define actions to perform on single sections within a form, based on a specific transaction. Form or section (image) level rules affect the section as a whole, not the individual fields and objects which make up the section or form.

  For more information on job level rules, see Job and Form Set Rules Reference on page 29.

- Field level rules (level 4)

  These rules define actions to perform on the variable fields in a section. Field level rules provide mapping, masking, and formatting information for each variable field on a form.

  For more information on job level rules, see Job and Form Set Rules Reference on page 29.

NOTE: Only memory limits the number of rules you can add to a section, however, having a large number of forms associated with a single section can be difficult to maintain.

Chapter 2

Adding Job and Form Set Rules

Job and form set rules help you control how a processing job is run and how the system processes the various form sets.

The rules which apply to the job and form set are stored in the AFGJOB.JDT file, which is called the *job definition table*, or *JDT* file. You add these rules directly into that file using a text editor.

In this chapter you will find information about:

For reference information on individual rules, see Job and Form Set Rules Reference on page 29

Using the Job Definition Table

The rules which apply to the job and form set are stored in the job definition table, which is called the *AFGJOB.JDT* or *JDT* file. You edit this file using a text editor. When editing the AFGJOB.JDT file, you can use these types of delimiters:

Use this delimiter...	To...
backslash and asterisk (/*)	Denote comments
comma (,)	Separates the data that comprises a parameter
semi-colon (;)	Separates parameters

The base system uses the rules in the JDT file when you run the main batch system programs (GenTrn, GenData, GenPrint, GenWIP, and GenArc). You can run these programs several ways:

- Multi-step processing
- Single-step processing
- WIP transaction processing

For multi-step processing, you run each program separately. With single-step processing, you run the GenData program using rules to perform the tasks handled by the GenTrn and GenPrint programs. The AFGJOB.JDT files differ for each approach. Examples of each approach follow.

WIP transaction processing lets you add or merge WIP transactions manually approved or rejected into a GenData processing run. These transactions can be processed as new transactions or appended to an master resource library (MRL) already processed by the GenData program.

Multi-Step Processing

Multi-step processing lets you run each batch system program in turn and check the log and error messages after each step. You can learn more about the system flow and the input and output files for each processing step in Chapter 2 of the Documaker Server System Reference.

Multi-step processing AFGJOB.JDT file

```
<Base Rules>
  ;RULStandardJobProc;1;Always the first job level rule;
  ;SetErrHdr;;*:;
  ;SetErrHdr;;*:-----------------------------------------------;
  ;SetErrHdr;;*: FormMaker Data Generation (Base);
  ;SetErrHdr;;*: ;
  ;SetErrHdr;;***: Transaction:  ***PolicyNum***;
  ;SetErrHdr;;***: Symbol:  ***Symbol***;
  ;SetErrHdr;;***: Module:  ***Module***;
  ;SetErrHdr;;***: State:  ***State***;
  ;SetErrHdr;;***: Company Name (after INI conversion):
***Company***;
  ;SetErrHdr;;***: Line of Business (after INI conversion):
***Lob***;
  ;SetErrHdr;;***: Trans Type:  ***TransactionType***;
```

```
  ;SetErrHdr;;***: Run Date:     ***RunDate***;
  ;SetErrHdr;;*:-----------------------------------------------;
  ;CreateGlbVar;;TXTLst,PVOID;
  ;CreateGlbVar;;TblLstH,PVOID;
  ;JobInit1;;;
  ;LoadDDTDefs;;;
  ;InitOvFlw;;;
  ;LoadTextTbl;;;
  ;LoadTblFiles;;;
  ;SetOvFlwSym;;CGDECBDOVF,QGDCBD,1;
  ;BuildMasterFormList;;4;
```

Every form set in the base system uses these form set level rules:

```
<Base Form Set Rules>
  ;RULStandardTransactionProc;;Always the first transaction level
rule;
  ;LoadExtractData;;;
  ;GetCo;;11,HEADERREC 35,3;
  ;GetLOB;;11,HEADERREC 40,3;
  ;ResetOvFlw;;;
  ;IfRecipUsed;;BATCH1=INSURED;
  ;IfRecipUsed;;BATCH2=COMPANY;
  ;IfRecipUsed;;BATCH3=AGENT;
  ;BuildFormList;;;
  ;LoadRcpTbl;;;
  ;UpdatePOLFile;;;
  ;RunSetRcpTbl;;;
```

Every section in the base system uses these section level rules:

```
<Base Image Rules>
  ;RULStandardImageProc;;Always the first section level rule;
  ;InsNAHdr;;;
```

Every field in the base system uses this field rule:

```
<Base Field Rules>
  ;RULStandardFieldProc;;Always the first field level rule;
```

SINGLE-STEP PROCESSING

To enhance performance, you can combine the execution and functionality of the GenTrn and GenData steps into a single step. Combining these steps enhances performance by reducing the number of times files have to be opened and closed during processing. For more information, see Chapter 2 of the Documaker Server System Reference.

To combine the GenTrn and GenData steps, you place the NoGenTrnTransactionProc rule in under the <Base Form Set Rules> header in your AFGJOB.JDT file, along with several other rules. To then combine the GenData and GenPrint steps, add the following rule under the <Base Rules> header in your AFGJOB.JDT file:

```
;InitPrint;;;
```

and add this rule below the <Base Form Set Rules> header in your AFGJOB.JDT file:

```
;PrintFormset;;;
```

To use single-step processing, change the TrnFile option in the FSISYS.INI file to *NUL*, as shown below:

```
< Data >
    TrnFile = NUL
```

Once you have added the rules to your AFGJOB.JDT file and FSISYS.INI file, run the GenData program as you normally would and it will execute the GenTrn and GenPrint processing steps.

For more information on these rules, see InitPrint on page 146, PrintFormset on page 182, and NoGenTrnTransactionProc on page 168.

Single-step processing AFGJOB.JDT file

When you use single-step processing, where the GenData program runs the GenTrn and GenPrint processes as a single step, you use the following AFGJOB.JDT file. This file is also called the *performance mode JDT file*:

```
<Base Rules>
;RULStandardJobProc;1;Always the first job level rule;
;SetErrHdr;;*:-----------------------------------------------;
;SetErrHdr;;*: FormMaker Data Generation (Base);
;SetErrHdr;;*: ;
;SetErrHdr;;***: Transaction:  ***ACCOUNTNUM***;
;SetErrHdr;;***: Company Name (after ini conversion):  ***Company***;
;SetErrHdr;;***: Line of Business (after ini conversion):  ***LOB***;
;SetErrHdr;;***: Run Date:     ***RunDate***;
;SetErrHdr;;*:-----------------------------------------------;
;JobInit1;;;
;CreateGlbVar;;TXTLst,PVOID;
;CreateGlbVar;;TblLstH,PVOID;
;InitOvFlw;;;
;SetOvFlwSym;;SUBGROUPOVF,SUBGROUP,5;
;BuildMasterFormList;;4;
;PageBatchStage1InitTerm;;;
;InitSetrecipCache;;;
/*  the following is required to run GenData/GenPrint as single
step.*/
;InitPrint;;;
```

Every form set in the base system uses these form set level rules:

```
<Base Form Set Rules>
;NoGenTrnTransactionProc;;First transaction level rule when omitting
GenTrn;
;ResetOvFlw;;;
;BuildFormList;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
/*  the following is required to run GenData/GenPrint as single
step.*/
;PrintFormset;;;
;WriteOutput;;;
;WriteNaFile;;;
;WriteRCBWithPageCount;;;
;ProcessQueue;;PostPaginationQueue;
;PaginateAndPropagate;;;
;BatchingByRecipINI;;;
```

Every section in the base system uses this section level rule:

```
<Base Image Rules>
;StandardImageProc;;Always the first section level rule;
```

Every field in the base system uses this field level rule:

```
<Base Field Rules>
;StandardFieldProc;;Always the first field level rule;
```

GenData WIP Transaction Processing

GenData WIP Transaction Processing lets you process WIP transactions based on their status code. The transactions are created by one of these processes:

- Executing the GenWIP program after the GenData program to process the transactions in the manual batch. Then using Documaker Workstation to:
  - Manually view a transaction and update any required data. Then use the WIP, Save option to save the transaction with a status code such as: Approved or Accepted.
  - Manually view a transaction and then use the WIP, Save option to save the transaction with a status code of Rejected.
  - Manually view a transaction, update any required data, and save the transaction using the File, Complete, Batch Print option. This assigns a Batch Print status code to the transaction.
- Creating a new transaction using Documaker Workstation and then using the WIP, Save or File, Complete, Batch Print option to save it with a status code such as Approved, Accepted, or Rejected.

You can then process these transactions as:

- New transactions
- Transactions appended to an existing MRL recipient batch, NewTrn, NA, and POL files created by a prior run of the GenData program

GenData WIP Transaction Processing creates new recipient batch, NewTrn, NA, and POL files which you can print, archive, or both using the GenPrint and/or GenArc programs.

To do this, you execute the GenData program using a simplified AFGJOB.JDT file that contains rules to replace the existing form set, section, and field rules. In addition, you must add two rules.

Here is a list of the rules used for GenData WIP Transaction Processing. All of these rules are required in the simplified AFGJOB.JDT file.

Rule	Description
MergeWIP on page 162	This job level rule initializes WIP Transaction Processing and specifies the status codes for the transactions added or appended to a newly-created NA and POL list.
WIPTransactions on page 245	This form set level rule replaces the RULStandardTransactionProc or NoGenTrnTransactionProc rule in the AFGJOB.JDT file and starts form set processing. It also identifies the status codes for transactions to be processed. The status codes specified in this rule's parameters can include any or all of the status codes specified for the MergeWIP rule.
GVM2GVM on page 107	This form set rule to copies GVM variable data from the WIP.DBF file into GVM variables needed by the GenData program. Use the Trigger2WIP control group options to define GVM variable names
WIPImageProc on page 244	This section level rule replaces the RULStandardImageProc or StandardImageProc rule in the AFGJOB.JDT file and tells the GenData program to bypass section data processing.
WIPFieldProc on page 243	This field level rule replaces the RULStandardFieldProc or StandardFieldProc rule in the AFGJOB.JDT file and tells the GenData program to bypass field data processing.

NOTE: All WIP file transactions added to the transaction memory list by the MergeWIP rule are deleted from the WIP file after processing. You can remove specific transaction types, such as Rejected, by including the status code in the parameters for the MergeWIP rule and omitting it in the parameters for the WIPTransactions rule.

WRITING UNIQUE DATA INTO RECIPIENT BATCH RECORDS

The GenData program lets you add unique data to recipient batch records before they are written to the recipient batch files. The recipient batch record data and format is defined by the GVM variable definitions in the RCBDFDFL.DAT file.

You can use this capability if you need to add...

- Address information or other field level information to the batch record, which is typically unique for each recipient.
- Recipient information that is not handled by normal field mapping from the transaction DFD to the recipient batch DFD.
- Cumulative or calculated information not available until the document is nearly completed.

NOTE: Before the ability to add data to recipient batch records was added in version 10.2, the recipient batch records were identical except for the recipient code field which contains a unique identifier assigned to a given recipient. If additional recipient data was required, you had to write a custom rule.

Use the options in the RecipMap2GVM control group to set up this capability. Data that can be added to the recipient batch record can be:

- Contents of a variable field on the specified section or form/section
- Constant value
- Data from an existing INI built-in functions, such as ~DALRun
- Data from a custom written INI function

Here is an example of the RecipMap2GVM control group:

```
< RecipMap2GVM >
    Form    =
    Image   =
    Req     =
    Opt     =
```

Option	Description
Form	(Optional) Enter the name of the form.
Image	Enter the name of the section. You can also enter a section name root. A section name root is the first part of a name. For instance, *MAILER* is the root name for sections with names such as *MAILER A*, *MAILER_B*, or *MAILERS*.
Req *	A semi-colon delimited string that contains one of the following: - GVM variable name; variable field name; optional formatting information - GVM variable name; blank character (space); constant value - GVM variable name; INI built-in function
Opt *	A semi-colon delimited string that contains one of the following: - GVM variable name; variable field name; optional formatting information - GVM variable name; blank character (space); constant value - GVM variable name; INI built-in function

* = Repeat for each GVM variable you are setting up.

Suppressing RCBMapFromINI function warning messages

Use the WarnOnLocate option to suppress the following warning message from the RCBMapFromINI function:

```
Cannot locate image root named/image
```

Here is an example:

```
< RecipMap2GVM >
    WarnOnLocate = No
```

Option	Description
WarnOnLocate	Enter No if you want to suppress this warning message from the RCBMapFromINI function: `Cannot locate image root named/image` The default is Yes.

Optional formatting information

You can add optional formatting information as a parameter of the Opt INI option. This formatting information is comprised of four items separated by commas.

Item	Description
Input fetypes	D or d = date N or n = number
Input format mask	Date - see the FmtDate rule in the Rules Reference. Number – see the FmtNum rule in the Rules Reference.
Output fetypes	D or d = date N or n = number
Output format mask	Date - see the FmtDate rule in the Rules Reference. Number – see the FmtNum rule in the Rules Reference.

Here are some formatting examples:

```
d,"1/4", d, "4/4"
```

This converts an input date, mmddyyyy, into *month name dd, yyyy*, such as February 17, 2009.

```
n, nCAD, nUSD, "$zzz,zz9.99"
```

This converts an input numeric value in Canadian French format into a value in United States format.

Keep in mind...

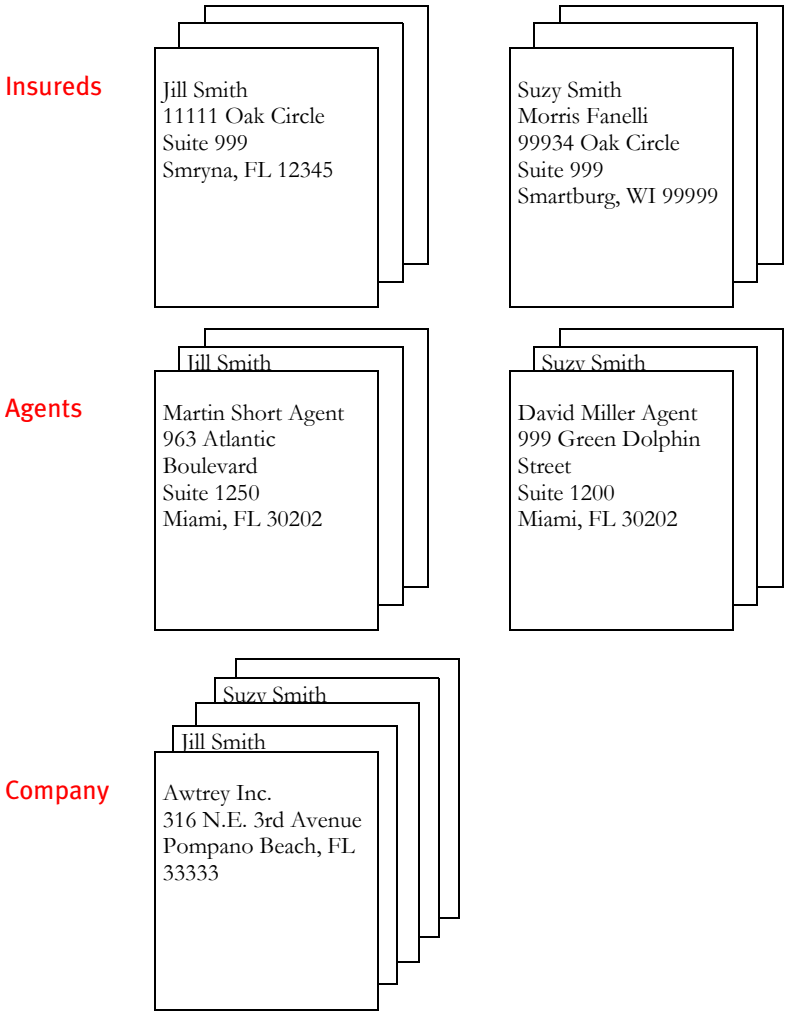
- For the Req option, if the data is missing an error occurs and the transaction is send to the error batch.
- For the Opt option, if the data is missing the system stores an empty string in the GVM variable.
- A RCB GVM variable cannot be restored to its original or default value after it has been changed using this method.
- Any RCB GVM variable not assigned using this method retains the value originally set during the transaction processing.
- Some RCB GVM variables should never be changed using this mapping technique. These include:
  - TRN_Offset
  - NA_Offset

- POL_Offset

- If the section defined in the Image option in the RecipMap2GVM control group does not name a section, the feature is disabled for all transactions.
- If the section defined in the Image option is missing from the form set being processed, the GVM data is not changed. Depending on where the GVM data is mapped, this could mean data from the prior transaction will still be in the GVM variables.
- If there are multiple sections with the same name in the form set, the form specified in the Form option is used to identify the section to use. If the Form option is omitted, the first section found in the current form set is used.
- The system assumes the specified section contains all of the unique data except for a constant value or data gathered from an INI built-in function.
- If more than one recipient is assigned to the section, all recipient batch records receive the same added data.

Example

This example creates a mailer cover page for each insured, agent, and/or company recipient per transaction. The cover page is created using banner page processing which occurs during GenPrint processing. Examples of the three different mailer cover pages are as follows.



This example assumes that the:

- Agent and company recipient batch files are sorted (agent number and company name, respectively) before the GenPrint program runs. This sorting allows for the creation of only one mailer cover page per unique agent and company.
- Unique information is contained on the form/image, Dec Page/Q1MDC1.

- The FSIUSER.INI file includes these control groups and options:

```
< RecipMap2GVM >
    Form      = Dec Page
    Image     = Q1MDC1
    Opt       = Name1;Insured Name;
    Opt       = Name2;Insured Name2;
    Opt       = Address1;Address Line1;
    Opt       = Address2;Address Line2;
    Opt       = CityCounty;prtvalue;
    Opt       = AgentName;Agent Name;
    Opt       = AgentID; Agent ID;
    Opt       = OfficeAddress;Office Address;
    Opt       = TownandState;Town And State;
< Printer >
   PrtType            = PCL
   EnableTransBanner  = True
   EnableBatchBanner  = False
   TransBannerBeginScript= PreTrans
   TransBannerEndScript= PstTrans
   TransBannerBeginForm= ;BANNER;TRANSACTION;TRANS HEADER;
   TransBannerEndForm  = ;BANNER;TRANSACTION;TRANS TRAILER;
< DALLibraries >
    LIB = Banner
```

BANNER.DAL

The DefLib directory contains this DAL script:

```
* This script obtains the required unique data from the recipient
* batch record and stores it on the mailer form.

BeginSub PreTrans

blank_gvm = Pad(" ",41," ")
SetGVM("NameA"       ,blank_gvm,,"C",41)
SetGVM("NameB"       ,blank_gvm,,"C",41)
SetGVM("AddressA"    ,blank_gvm,,"C",41)
SetGVM("AddressB"    ,blank_gvm,,"C",41)
SetGVM("CityCounty1" ,blank_gvm,,"C",41)
If Trim(RecipName()) = "INSURED" Then
   SetGVM("NameA"       ,GVM("Name1")     ,,"C",41)
   SetGVM("NameB"       ,GVM("Name2")     ,,"C",41)
   SetGVM("AddressA"    ,GVM("Address1")  ,,"C",41)
   SetGVM("AddressB"    ,GVM("Address2")  ,,"C",41)
   SetGVM("CityCounty1" ,GVM("CityCounty"),,"C",41)
   GoTo exit:
End

last_agent_id = last_agent_id
If Trim(RecipName()) = "AGENT" Then
   If last_agent_id != Trim(GVM("AgentID")) Then
    last_agent_id = Trim(GVM("AgentID"))
    SetGVM("NameA"       ,GVM("AgentName")     ,,"C",41)
    SetGVM("NameB"       ,GVM("OfficeAddress") ,,"C",41)
    SetGVM("AddressA"    ,GVM("TownandState")  ,,"C",41)
    GoTo exit:
   Else
```

```
     SuppressBanner()
     GoTo exit :
   End
End

last_company_name = last_company_name
If Trim(RecipName()) = "COMPANY" Then
   If Trim(GVM("Company")) != last_company_name Then
        last_company_name = Trim(GVM("Company"))
        If Trim(GVM("Company")) = "SAMPCO" Then;
         SetGVM("NameA"      ,"Sampco, Inc."             ,,"C",41)
            SetGVM("NameB"     ,"316 N.E. 3rd Avenue"      ,,"C",41)
            SetGVM("AddressA"  ,"Pompano Beach, FL 33333"  ,,"C",41)
            GoTo exit:
          ElseIf Trim(GVM("Company")) = "FSI"
          SetGVM("NameA"      ,"FSI Inc."                ,,"C",41)
          SetGVM("NameB"      ,"222 Newbury St."         ,,"C",41)
          SetGVM("AddressA"   ,"Northwest City, FL  99999"  ,,"C",41)
            GoTo exit:
        End
      Else
        SuppressBanner()
        GoTo exit:
   End
End
exit:
EndSub

BeginSub PstTrans
EndSub
```

The RCBDFDFL.DAT file contains the following GVM variable definitions which are defined in the RecipMap2GVM control group:

- Name1
- Name2
- Address1
- Address2
- CityCounty
- AgentName
- AgentID
- OfficeAddress
- TownAndState

Here are two recipient batch records from this example:

```
SAMPCOLB12234567SCOM1FLT1  B2199802232234567890      0   22560
******001    3724     452Jill Smith             Morris
11111 Oak Circle              Suite 999                    Smyrna,
FL  12345              Martin Short Agent           963 Main Street,
Suite 1250  Miami, FL 30202

FSI  CPP4234567FSIM1WIT1  B3199802234234567890      0   30360
******001    4667     565Suzy Smith             Morris
99934 Oak Circle             Suite 999                     SmartBurg,
WI  99999              David Miller Agent           999 Main Street,
Suite 1200   Miami, FL 30202
```

Sample AFGJOB.JDT Files and INI Options

Shown below are examples of simplified AFGJOB.JDT files and the INI options you use to process WIP transactions for specified recipients using these rules:

- IfRecipUsed on page 108
- BatchingByRecipINI on page 68
- BatchingByPageCountINI on page 49

Assume each example has these INI options:

```
< Status_CD >
    Approved    = AP
    BatchPrint  = BP
    Rejected    = RJ
```

Also assume the first two examples have the following INI options defined in the FSISYS.INI or FSIUSER INI file.

These options define the recipient batch names:

```
< Print_Batches >
    Insured     = .\batch\Insured
    Agent       = .\batch\Agent
    Company     = .\batch\Company
```

These options define the output printer names:

```
< PrinterInfo >
    Printer    = InsuredPrt
    Printer    = AgentPrt
    Printer    = CompanyPrt
```

These options define the output printer names for each recipient batch. You must have a control group for each recipient batch.

```
< Insured >
    Printer    = InsuredPrt
< Agent >
    Printer    = AgentPrt
< Company >
    Printer    = CompanyPrt
```

These options define the print-ready output file name for each recipient name:

```
< InsuredPrt >
    Port       = .\Print\Insured.PCL
< AgentPrt >
    Port       = .\Print\Agent.PCL
< CompanyPrt >
    Port       = .\Print\Company.PCL
```

Using the IfRecipUsed rule

You run the GenData program using a simplified AFGJOB.JDT file which contains an IfRecipUsed rule for each recipient. This example places print-ready output for each recipient in the following files:

Recipient	Output file
Insured	INSURE.PCL
Agent	AGENT.PCL
Company	COMPANY.PCL

Transactions with status codes defined in the WIPTransactions rule are appended to an existing MRL recipient batch, NewTrn, NA, and POL files or are appended to newly-created recipient batch, NewTrn, NA, and POL files. These files can be printed, archived, or both using the GenPrint and GenArc programs.

All transactions with a Rejected or an Approved status code are deleted from the WIP file. Here is an example of the AFGJOB.JDT file. Note that the Rejected status code is omitted from the WIPTransactions rule.

```
<Base Rules>
;RulStandardJobProc;;;
;MergeWIP;;Approved,Rejected;
;JobInit1;;;

<Base Form Set Rules>
;WIPTransactions;;Approved;
;GVM2GVM;;Trigger2WIP;
;IfRecipUsed;;Batch1=Insured;
;IfRecipUsed;;Batch2=Company;
;IfRecipUsed;;Batch3=Agent;
```

```
;UpdatePOLFile;;;
<Base Image Rules>
;WIPImageProc;;;
<Base Field Rules>
;WIPFieldProc;;;
```

Using the BatchingByRecipINI rule

You run the GenData program using a simplified AFGJOB.JDT file which contains the BatchingByRecipINI rule. The BatchingByRecip control group contains an option for each recipient. Define this control group in the FSISYS.INI or FSIUSER.INI file. This example places print-ready output for each recipient in the following files:

Recipient	Output file
Insured	INSURE.PCL
Agent	AGENT.PCL
Company	COMPANY.PCL

Transactions with status codes defined in the WIPTransactions rule are appended to an existing MRL recipient batch, NewTrn, NA, and POL files or are appended to newly-created recipient batch, NewTrn, NA, and POL files. These files can be printed, archived, or both using the GenPrint and GenArc programs.

All transactions with a Rejected or Batch Print status code are deleted from the WIP file. Here is an example of the AFGJOB.JDT file. Note that the Rejected status code is omitted from the WIPTransactions rule.

```
<Base Rules>
;RULStandardJobProc;1;;;
;JobInit1;;;
;MergeWIP;;BatchPrint,Rejected;
;InitSetrecipCache;;;
<Base Form Set Rules>
;WIPTransactions;;BatchPrint;
;WriteOutput;;;
;WriteNaFile;;;
;BatchingByRecipINI;;;
<Base Image Rules>
;WIPImageProc;;;
<Base Field Rules>
;WIPFieldProc;;;
```

Using the BatchingByPageCountINI rule

You run the GenData program using a simplified AFGJOB.JDT file which contains the BatchingByPageCountINI rule. The BatchingByRecip control group contains an option for each recipient. Define this control group in the FSISYS.INI or FSIUSER.INI file.

This example places print-ready output for each recipient into the following files based on the number of pages in each transaction processed.

File	Description
INSOVER3.PCL	Insured with more than three pages
INSUNDR4.PCL	Insured with less than three pages
AGIOVER3.PCL	Agent with more than three pages
AGIUNDR4.PCL	Agent with less than three pages
CTROVER3.PCL	Company with more than three pages
CTRUNDR3.PCL	Company with less than three pages

Transactions with status codes defined in the WIPTransactions rule are appended to an existing MRL recipient batch, NewTrn, NA, and POL files or are appended to newly-created recipient batch, NewTrn, NA, and POL files. These files can be printed, archived, or both using the GenPrint and GenArc programs.

All transactions with a Rejected or Batch Print status code are deleted from the WIP file. Here is an example of the AFGJOB.JDT file. Note that the Rejected status code is omitted from the WIPTransactions rule.

```
<Base Rules>
;RulStandardJobProc;;;
JobInit1;;;
;MergeWIP;;BatchPrint,Rejected;
;InitSetrecipCache;;;

<Base Form Set Rules>
;WIPTransactions;;BatchPrint;
;GVM2GVM;;Trigger2WIP;
;WriteOutput;;;
;WriteNaFile;;;
;BatchingByPageCountINI;;;
;WriteRCBWithPageCount;;;
;ProcessQueue;;PostPaginationQueue;
;PaginateAndPropagate;;;

<Base Image Rules>
;WIPImageProc;;;

<Base Field Rules>
;WIPFieldProc;;;
```

Here are the INI options used with the BatchingByPageCountINI rule:

```
< BatchingByRecip >
    DefaultBatch   = Default
    Batch_Recip_Def= True;"InsOver3";Insured
    Batch_Recip_Def= True;"InsUndr4";Insured
    Batch_Recip_Def= True;"AgiOver3";AddLinsd
    Batch_Recip_Def= True;"AgiUndr4";AddLinsd
    Batch_Recip_Def= True;"CtrOver3";CertHld
    Batch_Recip_Def= True;"CrtUndr4";CertHld

< Print_Batches >
    InsOver3 = .\batch\InsOver3
    InsUndr4 = .\batch\InsUndr4
```

```
    AgiOver3 = .\batch\AgiOver3
    AgiUndr4 = .\batch\AgiUndr4
    CtrOver3 = .\batch\CtrOver3
    CtrUndr3 = .\batch\CtrUndr4
    Default = .\batch\Default
< PrinterInfo >
    Printer = InsOver3Prt
    Printer = InsUndr4Prt
    Printer = AgiOver3Prt
    Printer = AgiUndr4Prt
    Printer = CtrOver3Prt
    Printer = CrtUndr4Prt
    Printer = DefaultPrt
< InsOver3 >
    Printer = InsOver3Prt
    PageRange = 4,999
 < InsUndr4 >
    Printer = InsUndr4Prt
    PageRange = 1,3
< AgiOver3 >
    Printer = AgiOver3Prt
    PageRange = 4,999
< AgiUndr4 >
    Printer = AgiUndr4Prt
    PageRange = 1,3
< CtrOver3 >
    Printer = CtrOver3Prt
    PageRange = 4,999
< CrtUndr4 >
    Printer = CrtUndr4Prt
    PageRange = 1,3
< Default >
    Printer = DefaultPrt
< InsOver3Prt >
    Port = .\Print\InsOver3.PCL
< InsUndr4Prt >
    Port = .\Print\InsUndr4.PCL
< AgiOver3Prt >
    Port = .\Print\AgiOver3.PCL
< AgiUndr4Prt >
    Port = .\Print\AgiUndr4.PCL
< CtrOver3Prt >
    Port = .\Print\CtrOver3.PCL
< CrtUndr4Prt >
    Port = .\Print\CrtUndr4.PCL
< DefaultPrt >
    Port = .\Print\Default.PCL
```

Processing Import Files

The GenData program can import and process these types of export files created by Documaker Workstation:

- Standard export
- WIP/NA/POL export
- XML export

The transactions exported to a file can be created by:

- Executing the GenWIP program after the GenData program to process any transactions in the manual batch. You then use Documaker Workstation to view the transaction and update any required data. Finally, you use the File, Complete, Export Data option to create the export file.
- Creating a transaction using Documaker Workstation. You then use the File, Complete, Export Data option to create the export file.
- Creating a transaction using iPPS.

You can then process the export files as a:

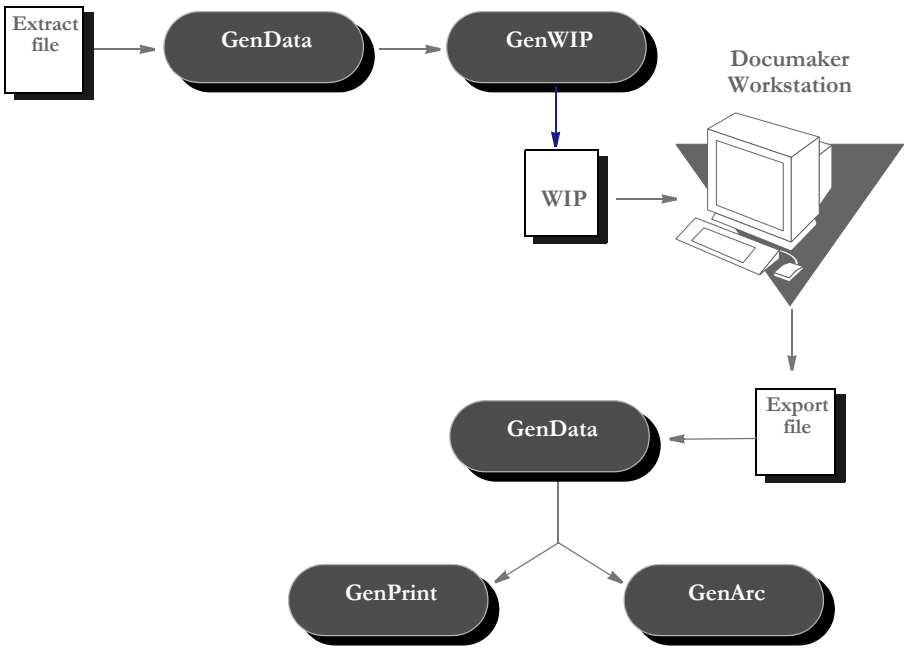
- Single transaction using the appropriate import file rule in a simplified AFGJOB.JDT. For instance, you would use one of these rules:
  - ImportFile on page 116
  - ImportNAPOLFile on page 126
  - ImportXMLFile on page 134
- Multiple transactions (one or more export files appended in a single file) using the appropriate import extract rule in a simplified AFGJOB.JDT.
  - ImportExtract on page 111
  - ImportNAPOLExtract on page 121
  - ImportXMLExtract on page 130

NOTE: Create a separate AFGJOB.JDT file for this process, instead of updating an existing one by commenting out rules and adding new one.

Here are some GenData import file processing scenarios:

Using Documaker Server

You run the GenData program using an extract file and then execute the GenWIP program to process any transactions in manual batch. These transactions are then added to the WIP file.



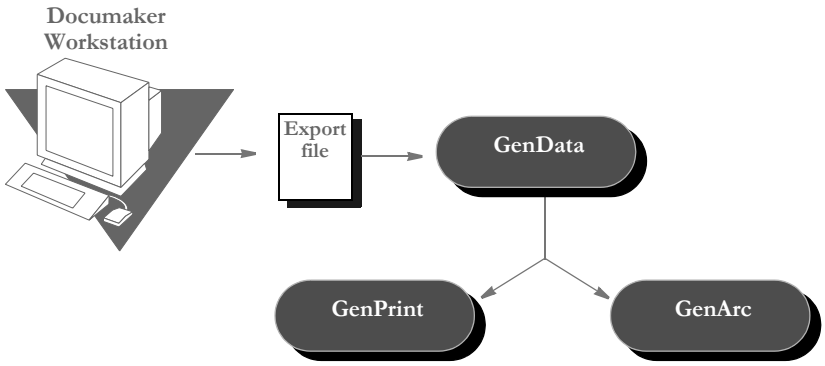
NOTE: Transactions in the manual batch file were placed there because they were flagged to go to manual batch, missing required field data, or were flagged as KickToWIP.

You then open the WIP transactions using Documaker Workstation, make necessary changes, and use the File, Complete, Export File option to create the export file.

After you finish, you run GenData Import File Processing using simplified AFGJOB.JDT and INI files. Using the export file as an import file, the GenData program then creates new recipient batch, NewTrn, NA, and POL files which you can print, archive, or both using the GenPrint and GenArc programs.

Using Documaker Workstation

You create new transactions using Documaker Workstation and then use the File, Complete, Export File option to create the export file.

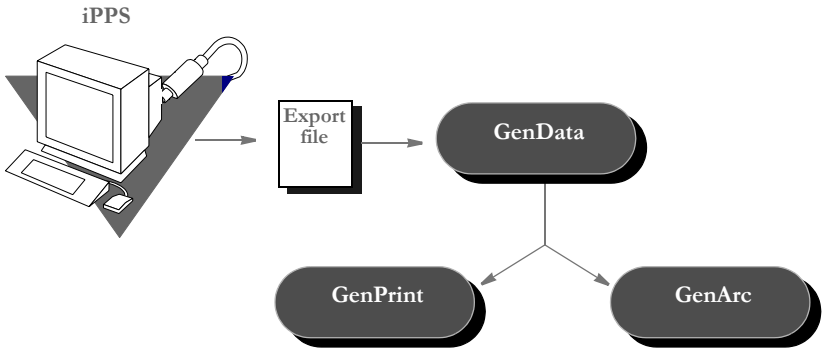


You then use GenData Import File Processing to create new recipient batch, NewTrn, NA, and POL files. These files can be printed, archived, or both using the GenPrint and GenArc programs.

NOTE: For information on setting up Documaker Workstation, see the Documaker Workstation Supervisor's Guide.

Using iPPS

You create a transaction using iPPS that is then processed by GenData Import File Processing to produce PDF files. These files can be viewed on-line and printed.



Rules Used in Single-Step Processing

Specific rules are used to combine the execution and functionality of the GenTrn, GenData, and GenPrint programs into a single step. These rules are listed below, with a brief description.

NOTE: You can find more information, including a detailed description of how processing occurs, in Chapter 2 of the Documaker Server System Reference.

Here's a list of the rules required for single-step processing.

Rule	Description
BatchingByRecipINI on page 68	Use this rule to send transactions to a batch you specify using INI options.
BatchByPageCount on page 47	Use this rule to send a transaction's form set to a specified print batch based on the number of printed pages plus the multi-mail code defined in the transaction.
BatchingByPageCountINI on page 49	Use this rule to send a transaction's form set to a specified batch based on the number of printed pages created when the system processes the transaction.
BuildMasterFormList on page 73	Use this rule to load the FORM.DAT file into an internal linked list within the GenData program. You must include this rule in the AFGJOB.JDT file because the RunSetRcpTbl rule is dependent on the list this rule creates.
InitPrint on page 146	Use this rule to load printer and recipient batch information. This rule sets up PRTLIB data, initializes print options, and loads a table which contains page totals for recipient batch files. Use this rule when you run the GenData program by itself to execute GenTrn and GenPrint processes. This rule, when combined with the PrintFormset rule, prints form sets.
InitSetRecipCache on page 147	Use this rule to set the cache the system uses to store recipient information in memory. With this rule you can tell the system the amount of memory to set aside and use for storing information in the Key1 and Key2 fields, often used to store the company and line of business.
NoGenTrnTransactionProc on page 168	Use this rule when you use the GenData program by itself to execute the GenTrn and GenData steps. When combined with the InitPrint and PrintFormset rules, it creates the output files created during the GenPrint step.
PageBatchStage1InitTerm on page 173	Use this rule to create and populate a list of records which contain page ranges and total page counts for each recipient batch file.
PaginateAndPropagate on page 174	Use this rule to paginate the form set and merge in or propagate field data.
PrintFormset on page 182	Use this rule when you run the GenData program by itself to execute GenTrn and GenPrint processes. This rule, when combined with the InitPrint rule, prints form sets.

Rule	Description
ProcessQueue on page 184	Use this rule to process the queue you specify. This rule loops through the list of functions for the queue you specify and then frees the queue when finished.
StandardFieldProc on page 235	This rule tells the system to process each field on all of the sections triggered by the SETRCPTB.DAT file. If you use the StandardFieldProc rule is in your JDT, you must also include the WriteNAFile rule.
StandardImageProc on page 236	This rule tells the system to process each section triggered by the SETRCPTB.DAT file.
WriteNAFile on page 247	Use this rule to append the NAFILE.DAT file data records for the current form set into an existing NAFILE.DAT file.
WriteOutput on page 248	Use this rule to create the POLFILE.DAT file.
WriteRCBWithPageCount on page 250	Use this rule to write page counts for each recipient.

Rules Used for 2-up Printing

The following descriptions will help familiarize you with the rules that are required to perform the 2-up printing process. All of the rules listed in the topic, Rules Used in Single-Step Processing on page 25 are required for 2-up printing, plus the additional rules listed below.

NOTE: You can find more information, including a detailed description of how processing occurs, in Chapter 2 of the Documaker Server System Reference.

Here's a list of the additional rules you can use for 2-up printing.

Rule	Description
AddLine on page 38	(Optional) Use this form set level (level 2) rule to add a line record, such as for OMR marks, to the AFP record list built by the MergeAFP rule.
AddTextLabel on page 39	(Optional) Use this form set level (level 2) rule to add a text label record to the AFP record list built by the MergeAFP rule.
GetRCBRec on page 105	Use this form set (level 2) level rule to set the current recipient batch file. This rule initializes the current recipient batch file, if necessary.
InitMerge on page 143	Use this job level (level 1) rule to create a list of printers, batches, and buffers for the comment (RCB) records. This rule also creates a list to hold AFP records and AFP fonts.
InitPageBatchedJob on page 145	Use this job level (level 1) rule to open NA and POL files.
MergeAFP on page 160	Use this form set level (level 2) rule to initialize input files. This rule populates the AFP record list, retrieves comment (RCB) records, and terminates the input files.
OMRMarks on page 169	(Optional) Use this job level (level 1) rule to generate OMR marks on 2-up documents printed on any AFP printer that supports 2-up printing.
ParseComment on page 176	(Optional) Use this form set level (level 2) rule to parse comment records into the GVM variable.
PrintData on page 181	Use this form set (level 2) rule to print the form set. This rule is used for handling 2-up printing on AFP and compatible printers.
ProcessRecord on page 185	Use this form set (level 2) rule to switch between print files as necessary when printing 2-up forms on an AFP printer. This rule updates the page count for current print file and loads and merges the form set.

Chapter 3

Job and Form Set Rules Reference

Job and form set rules help you control how a processing job is run and how the system processes the various form sets.

The rules which apply to the job and form set are stored in the AFGJOB.JDT file, which is called the *job definition table*, or *JDT* file. You add these rules directly into that file using a text editor.

NOTE: This chapter serves as a reference to job and form set rules. For information on the rules which apply to sections and fields, see Section and Field Rules Reference on page 274.

This chapter discusses rules included in the base system and supported by the support group. For information on custom rules, contact your Professional Services representative.

In this chapter you will find information about:

JDT Rules Reference

The following pages list and explain the various job and form set rules you can use. The rules are discussed in alphabetical order on the pages following this table.

The following table lists the rules by function in the first column. The Level column indicates whether the rule is a job level rule (1) or a form set level rule (2) in the AFGJOB.JDT file.

The Overflow column indicates the rules which support the overflow feature. The overflow features allow extract data to flow onto an additional page if needed.

To…	Level	Use this rule	Overflow
add a form set to a recipient batch	2	IfRecipUsed on page 108	na
add a line record, such as for OMR marks, to the AFP record list	2	AddLine on page 38	na
add a text label record to the AFP record list	2	AddTextLabel on page 39	na
add data from the extract list into global variables	2	Ext2GVM on page 93	na
add OMR marks on 1-up or on 2-up documents	1	OMRMarks on page 169	na
append a global variable to an extract file	2	AppendGblToExtr on page 42	na
append the NAFILE.DAT file data records for the current form set into an existing NAFILE.DAT file	2	WriteNAFile on page 247	na
assign form sets to specific batches	2	AssignToBatch on page 45	na
assign the recipients from a specific form to the other forms in a form set	2	MergeRecipsFromForm on page 161	na
build a form candidate list	2	PXCandidateList on page 187	na
build a form list by loading the FORM.DAT file into an internal linked list within the GenData program	1	BuildMasterFormList on page 73	na
bypass all section processing	2	ForceNoImages on page 98	na

To…	Level	Use this rule	Overflow
change the printer tray during processing.	2	SetOverflowPaperTray on page 224	na
check a field's value against another value for transactions currently in the generic linked list of objects	2	BatchByPageCount on page 47	na
check for zero font IDs	2	CheckZeroFontID on page 74	na
copy NA_Offset and POL_Offset into GVM variables	2	CreateRecordList on page 77	na
copy the data from a given GVM variable into another GVM variable	2	GVM2GVM on page 107	na
create a global variable	1	CreateGlbVar on page 76	na
create a print file that contains a set of forms filtered by form name, form description, or recipient name	2	ServerFilterFormRecipient on page 215	na
create a list of printers, batches, and buffers for the comment (RCB) records	1	InitMerge on page 143	na
create a POL file when doing 2-up printing	2	WriteOutput on page 248	na
create and populate a list of records which contain page ranges and total page counts for each recipient batch file	1	PageBatchStage1InitTerm on page 173	na
create the recipient batches when running in two-step mode	2	WriteRCBFiles on page 249	na
define an overflow variable	1	SetOvFlwSym on page 227	yes
delete records from an extract list	2	DelExtRecords on page 78	na
dump an extract list to a file	2	DumpExtList on page 90	na
dump an extract list to a file by transaction	2	DumpExtractListToFile on page 91	na

To…	Level	Use this rule	Overflow
exclude transactions from being processed in one- and two-step mode processing	1	BuildExcludeList on page 71	na
execute a DAL script	2	PostTransDAL on page 177	na
execute a DAL script	2	PreTransDAL on page 179	na
execute a DAL script if certain conditions are met	2	PXTrigger on page 189	na
execute a user function	1	RunUser on page 214	na
execute specific transactions for testing purposes	2	RULTestTransaction on page 211	na
execute regional date processing (RDP) rules on forms.	2	RegionalDateProcess on page 194	na
extract a form set from a DAP archive using an extract file.	2	LoadFormsetFromArchive on page 154	na
get a print batch name from an extract file	2	SetOutputFromExtrFile on page 221	na
get memory allocation information	1, 2	AllocDebug on page 41	na
get the company (Key1 field) from the extract data	2	GetCo on page 103	na
get the current date and use it as the run date	2	GetRunDate on page 106	na
get the line of business (Key2 field) from the extract data	2	GetLOB on page 104	na
import a single transaction from a combined NA/POL file	2	ImportNAPOLFile on page 126	na
import a single transaction from a standard import file	2	ImportFile on page 116	na
import an extract file	2	ImportExtract on page 111	na
import an XML extract file	2	ImportXMLExtract on page 130	na

To…	Level	Use this rule	Overflow
import an XML file	2	ImportXMLFile on page 134	na
import multiple transactions from a combined NAPOL extract file	2	ImportNAPOLExtract on page 121	na
initialize input files for AFP printers	2	MergeAFP on page 160	na
initialize resources such as input and output files	1	JobInit1 on page 151	na
initialize the overflow feature	1	InitOvFlw on page 144	yes
initialize the system for using the ConvertWIP rule	1	InitConvertWIP on page 142	na
load a table into a link list	1	LoadListFromTable on page 156	na
load and initialize all forms	2	BuildFormList on page 72	na
load entries from the SETRCPTB.DAT file based on the Key fields and transaction type	2	LoadRcpTbl on page 157	na
load extract data into memory for each transaction	2	LoadExtractData on page 153	na
load printer and recipient batch information	1	InitPrint on page 146	na
load text tables into the text table list	1	LoadTextTbl on page 159	na
load the field rules from the MASTER.DDT file into an internal linked list	2	LoadDDTDefs on page 152	na
load the table files listed in the tables list file	1	LoadTblFiles on page 158	na
maintain the exact data printed without LIbrary Manager	2	InlineImagesAndBitmaps on page 148	na
map data you are importing	2	ImageMapImportData on page 109	na

To…	Level	Use this rule	Overflow
nest overflow within overflow	2	RULNestedOverFlowProc on page 203	yes
open NA and POL files	1	InitPageBatchedJob on page 145	na
paginate the form set and merge field data, page ranges, and total pages	2	PaginateAndPropagate on page 174	na
parse comment records into the GVM variable field data	2	ParseComment on page 176	na
print form sets—for multi-step processing	2	PrintData on page 181	na
print form sets—for single-step processing	2	PrintFormset on page 182	na
process a queue	1	ProcessQueue on page 184	na
process each field triggered by the SETRCPTB.DAT file—for multi-step processing	1	RULStandardFieldProc on page 207	na
process each field triggered by the SETRCPTB.DAT file—for single-step processing	1	StandardFieldProc on page 235	na
process each field triggered by the SETRCPTB.DAT file—for WIP Transaction Processing	1	WIPFieldProc on page 243	na
process each section triggered by the SETRCPTB.DAT file—for multi-step processing	1	RULStandardImageProc on page 208	na
process each section triggered by the SETRCPTB.DAT file—for single-step processing	1	StandardImageProc on page 236	na
process each section triggered by the SETRCPTB.DAT file—for WIP Transaction Processing	1	WIPImageProc on page 244	na
process each transaction listed in the extract file	2	RULStandardTransactionProc on page 210	na

To…	Level	Use this rule	Overflow
process the extract file and create information created in both the GenTrn and GenData steps	2	NoGenTrnTransactionProc on page 168	na
process the groups (Key1, Key2 combinations) that exist in the form set, as opposed to only a single set of keys specified in the TRNFILE.DAT file	2	ProcessTriggers on page 186	na
process WIP transactions manually approved or rejected in Documaker Workstation	2	WIPTransactions on page 245	na
register the MapFromImportData rule which the system then uses instead of the NoOpFunc rule	2	ReplaceNoOpFunc on page 197	na
remove forms from form sets	2	FilterForm on page 94	na
remove forms from form sets based on recipients	2	FilterRecip on page 96	na
replace the LoadRcpTbl and RunSetRcpTbl rules in implementations created by Documaker Studio	2	RunTriggers on page 213	na
replace the RULStandardBaseProc rule when you use IDS to run Documaker	1	ServerJobProc on page 217	na
reset the overflow feature	2	ResetOvFlw on page 199	yes
reset the pRPS structure after the GVM variables have been remapped.	2	ResetDocSetNames on page 198	na
restart the GenData program	1	RestartJob rule on page 200	na
restart the GenData program	2	RULCheckTransaction on page 201	na
run a user function	1	RunUser on page 214	yes
run Documaker Server from another application via an XML job ticket	1	TicketJobProc on page 237	na

To…	Level	Use this rule	Overflow
run specified entries in the set recipient table	2	RunSetRcpTbl on page 212	na
run the GenArc program as part of single-step processing	2	Archive on page 43	na
run the GenArc program as part of single-step processing	1	InitArchive on page 141	na
run the GenWIP process to transfer transactions into WIP	2	ConvertWIP on page 75	na
send a transaction to a batch based on the number of pages the system generates when it processes the transaction	2	BatchingByPageCountINI on page 49	na
send transactions to a batch based on data in the extract file	2	BatchingByRecipINI on page 68	na
send a transaction to a specific print batch based on the number of page count for each recipient of all form sets the system generates when it processes the transaction	2	BatchingByPageCountPerRecipINI on page 55	na
send transactions to the manual batch when specified field errors occur	1	ErrorHandler on page 92	na
set the amount of memory you want the system to use to store the information in Key fields (speed processing of complex forms)	1	InitSetRecipCache on page 147	na
set the copy count for all forms except those listed	2	SetRecipCopyCount2 on page 229	na
set the copy count for all forms specified	2	SetRecipCopyCount on page 228	na
set the current recipient batch file	2	GetRCBRec on page 105	na
sort RCB batches before they are printed (so you can call a sort program to rearrange the order of the RCB files)	1	SortBatches on page 230	na
specify a job definition file	1	StandardFieldProc on page 235	na

To…	Level	Use this rule	Overflow
specify a print batch file for all recipients	2	AssignBatWithTbl on page 44	na
specify multiple XDBs to use across multiple key combinations	2	MultipleDataDictionaryFiles on page 166	na
specify the codes the system should look for in WIP Transaction Processing	1	MergeWIP on page 162	na
specify the text in the header of the error file	1	SetErrHdr on page 220	na
switch between print files when doing 2-up printing	2	ProcessRecord on page 185	na
terminate an XDB instance and free memory	1	Dictionary on page 79	na
translate error information	1	TranslateErrors on page 238	na
use an XML extract file	2	UseXMLExtract on page 240	na
use an XML extract file	2	XMLFileExtract on page 252	na
write out forms which contain descriptions of the other forms in the form set	2	FormDescription on page 99	na
write transactional information into each page of the print stream	1	InstallCommentLineCallback on page 150	na
write out full NA and POL information as well as certain export field information	2	DocumentExport on page 80	na
write the page count for each recipient when doing 2-up printing	2	WriteRCBWithPageCount on page 250	na
write the POL set to the POLFILE.DAT file	2	UpdatePOLFile on page 239	na

AddLine

Use this form set level (level 2) rule to add a line record, such as for OMR marks, to the AFP record list built by the MergeAFP rule.

Syntax

```
;AddLine;;Top,Bottom,Left,Right;
```

Parameter	Description
Top	location of the top edge of the line
Bottom	location of the bottom edge of the line
Left	location of the left edge of the line
Right	location of the right edge of the line

NOTE: The parameter values are all in absolute FAP coordinates. No shifting is done, which lets you place the marks anywhere on the printable area of the paper.

Example

```
;AddLine;;600,1200,600,1000;
```

This example tells the system to draw a line ¼ inch from the left edge of the page, down ¼ inch from the top of the page, for a length of ½ inch, with a width of a ¼ inch.

See also

MergeAFP on page 160

JDT Rules Reference on page 30

AddTextLabel

Use this form set level (level 2) rule to add a text label record to the AFP record list built by the MergeAFP rule. This rule is used in 2-up printing.

Syntax

```
;AddTextLabel;;Text,XPos,YPos,Orientation,Font;
```

Parameter	Description
Text	The text to be added.
XPos	The x coordinate of the text.
YPos	The y coordinate of the text.
Orientation	The optional text rotation (0, 90, 180, or 270 degrees).
Font	The AFP code font file name of the font to be used. Make sure the font has been defined in the font cross-reference (FXR) file and the font has already been used in a field, text label, or text area on that form set. In some situations, you may want to add a hidden field which uses the font you specify in this parameter of the AddTextLabel rule.

NOTE: Enter the XPos and YPos values in absolute FAP coordinates. No shifting occurs, which lets you place the marks anywhere on the printable area of the paper.

Example

```
;AddTextLabel;;Preliminary,600,600,90,X0DACOBF;
```

This example tells the system to write the text, *Preliminary*, on each page beginning ¼ inch down from top of page and ¼ inch in from the left page edge. The system rotates the text 90 degrees and uses Courier bold 16 pitch as the font.

Here is another example:

```
;AddTextLabel;;=DAL("page_1.dal"),5740,4500,0,X0DATIN9;
;AddTextLabel;;=DAL("page_1_barcode"),1800,35200,90,X0BC4N9P;
```

This example includes this DAL script:

```
BeginSub    Page_1
    #page_cnt = #page_cnt
    page_number = "*8080000001A00000" & #page_cnt & "S*"
    #page_cnt += 1
    Return(page_number)
EndSub
```

These rules tell the system to execute the DAL script named *page_1* to get the dynamic data that will be placed on each page beginning 4500 FAP units down from top of the page and 5740 FAP units in from the left page edge using the Times Roman 9 pitch font.

Here is another example:

```
;AddTextLabel;;=DAL("page_1_barcode"),1800,35200,90,X0BC4N9P;
```

This example includes this DAL script:

```
BeginSub  Page_1_barcode
    Return(new_barcode_left)
EndSub
```

This rule tells the system to execute the DAL script named *page_1_barcode* to get the data will be placed on each page beginning 35200 FAP units down from top of the page and 1800 FAP units in from the left page edge rotated down 90 degrees. The dynamic data will be displayed as a 3x9 barcode.

See also MergeAFP on page 160

Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

AllocDebug

Use this rule to get information on the number of memory bytes allocated and freed each time there is new maximum value of allocates not being freed. This rule outputs its error messages to the LOGFILE.DAT file instead of the ERRFILE.DAT file. You can use this rule to find cumulative memory allocations not reported as leaks.

The AllocDebug rule is unique in that you can use it at any level in the AFGJOB.JDT file because it is designed to run in all three processing states during rule Pre- and Post-processing.

You can place this rule in the <Base Rules>, <Base Form Set Rules>, <Base Image Rules>, and <Base Field Rules> sections in the AFGJOB.JDT file.

Syntax

```
;AllocDebug;;;
```

Example

```
;AllocDebug;;;
```

See also JDT Rules Reference on page 30

AppendGblToExtr

Use this form set level rule (level 2) to append a global variable to the extract file. You can use this rule to place selected fields in the trigger file into an extract file.

Syntax

```
;MYW32->AppendGblToExtr;2;ExtractFile_Key,GBL_Var GBL_Var
...GBL_Var;
```

In the data field, enter the name of the key the extract record should append to and the names of all global variables you want to appended to it.

You define the maximum length of the extract file record using this INI option:

```
< TRN_File >
    MaxExtRecLen =
```

If you are appending multiple GVM variables and are using an extract file key, the accumulated length should not exceed the maximum extract record length defined in the MaxExtRecLen option.

NOTE: If you enter the string *NOHEADER* as the ExtractFile_Key, the system appends global variables to the extract list without a header key.

If the accumulated length exceeds MaxExtRecLen, the rule fails and issues the following error:

```
Error in AppendGblToExtr(): Global variable <> exceeds maximum
length. Check MaxExtRecLen in INI group <TRN_FILE>
```

Example

```
;AppendGblToExtr;;;
```

See also JDT Rules Reference on page 30

Archive

Use this form set level rule (level 2), along with the InitArchive rule, to run the GenArc program as part of single-step processing.

The InitArchive rule checks the INI options in the Trigger2Archive control group, initializes the database, opens the APPIDX.DFD and CAR files, and perform other steps to initialize archive.

The Archive rule then unloads the current form set and converts field data for archive using the INI options in the Trigger2Archive control group.

Syntax

```
;Archive;2;;
```

Example

Here is an example:

```
< Base Form Set Rules >
;Archive;2;;
```

See also

InitArchive on page 141

JDT Rules Reference on page 30

AssignBatWithTbl

Use this form set level rule (level 2) to specify the print batch file for all recipients based on data found in the extract list.

Syntax

```
;MYW32->AssignBatWithTbl;2;ASSNBTCH.TBL,1,XYZ (D) (I,B,S1,H);;
```

In the data field, enter the name of the file which contains the batch assignment table. Entering the full path is optional.

After the file name, specify the search mask which if found tells the system to add the recipients (*D*) to the batch. If no record is found that matches the mask, recipients I,B,S1 and H are added to the batch.

The first set of parentheses contains the recipient list of draft recipients, the second set contains all other recipients the implementation uses.

The syntax for the batch assignment table is as follows. This is a small example with three entries, you can include more.

```
;BATCH2;*;;
;BATCH1;1,123;;
;BATCH3;1,456;;
(more lines could follow)
```

There are three semicolon-delimited fields, the first is a batch name, the second is a search mask which will be run against the extract data list for a possible match. The search mask field consists of one or more offset,data pairs. If there is data from different records, delimit the search masks by entering a pipe symbol (|).

Example

```
;BATCH1;1,123|1,546,18,XXX;HO,I,B1,B2;
```

The first record who's search mask matches a record in the extract data, moving from top to bottom through the list, will be used to determine the batch name for the processing of that transaction.

There must always be an entry which has an asterisk (*) as its search mask. This is the default batch the system uses if no matches are found. You must specify a default batch.

In the third field you can enter a set of recipient codes delimited by commas. If you enter this information, the system will only print forms for the recipients you specify. Note that the printed recipients will be a subset of the recipients specified.

NOTE: This rule can produce errors if it is run before the form set is created. You must place this rule after any rule which is used to build the form set. Normally, you would use the BuildFormList rule to build the form set.

See also JDT Rules Reference on page 30

AssignToBatch

Use this form set level rule (level 2) to identify form sets from a particular source and place those form sets into a special batch for review purposes. For instance, you can use this rule to identify policies from a particular agent, operator, or branch and place those policies into a special batch.

NOTE: You can insert several AssignToBatch rules in the base rules file (AFGJPB.JDT). All that return true will be placed in the appropriate recipient batch.

Syntax

```
;AssignToBatch;;(parameters);
```

Parameter	Description
Name	The recipient batch name to which the transaction should be assigned.
Delimiter	An equal sign (=). This is required.
Search mask	One or more pairs of offsets and data (search criteria) in a comma delimited list. Here is an example: `;AssignToBatch;;Manual=1,Patch399,31,AssignToBatch;` This example searches the records of each transaction for the string *Patch399* at offset 1 and the string *AssignToBatch* at offset 31. If a match is found then this transaction will be assigned to the Manual batch.

This table explains the order in which rules are assigned to recipient batches:

Batch	Order of precedence
Error	If an error occurs that causes the batch assignment of the transaction.
Manual	If the POWType rule exists in any triggered section.
Manual	If the KickToWIP rule exists and its condition are met.
xxxxxx	AssignToBatch rule assignment if the rule exists and its search criteria is met.
xxxxxx	Base form set rules such as: IfRecipUsed, BatchingByPageCountINI, BatchingByRecipINI, and so on.

Example

Here is an excerpt from an AFGJOB.JDT file:

```
<Base Rules>
;RULStandardJobProc;;;
…
…
<Base Form Set Rules>
;RulStandardTransactionProc;;;
…
…
;AssignToBatch;;Manual=1,Patch399,31,GVM,190,AssignToBatch;
;IfRecipUsed;2;Batch1=Customer;
…
```

...

Any transaction that has a record that matches the search criteria (character strings: 'Patch399' at offset 1, 'GVM' at offset 31 and ' AssignToBatch' at offset 190) will be assigned to the Manual batch.

See also

IfRecipUsed on page 108

JDT Rules Reference on page 30

BatchByPageCount

Use this form set level (level 2) rule to check the value of a field supplied for processing against that for transactions currently in the linked list of objects. These transactions are populated by the CreateRecordList rule. This rule is also used with multi-mail processing.

If the field has changed, the system writes the records to the recipient batch file, based on the total page count for all recipients of all form sets in the set of transactions. If you omit the parameter, the system writes records in the generic linked list of objects to the appropriate recipient batch files, based on the page count for the individual records.

If you are using multi-mail, the system updates the TotPage field of the recipient batch record to reflect the total page count for all recipients of all transactions in the multi-mail transaction set.

NOTE: Keep in mind this rule calculates the page count at the transaction level, not the recipient level.

Syntax

```
;BatchByPageCount;;(MMField);
```

Parameter	Description
MMField	(Optional) Name of the INI option in the Trn_Fields control group which defines where the multi-mail code will be found in each transaction.

NOTE: If you use this rule, you must also use the PageBatchStage1InitTerm, CreateRecordList, and WriteRCBWithPageCount rules.

Example

```
;BatchByPageCount;;MMField=MM_Field;
```

In this example, the system uses the multi-mail code defined in each transaction for batching purposes. The system checks the value in each transaction against that for transactions currently a VMMList, which is populated by CreateRecordList rule.

If the field has changed, the records are written to the recipient batch based on the total page count for all recipients for all form sets in the entire set of transactions.

NOTE: Because the end of a multi-mail transaction set is not known until after the following transaction, each multi-mail transaction set (or each transaction, in the non multi-mail situation) is written out during processing of the following transaction.

See also

Rules Used for 2-up Printing on page 27

BatchingByPageCountINI on page 49

BatchingByPageCountPerRecipINI on page 55

BatchingByRecipINI on page 68

CreateRecordList on page 77

PageBatchStage1InitTerm on page 173

PrintFormset on page 182

WriteRCBWithPageCount on page 250

JDT Rules Reference on page 30

BatchingByPageCountINI

Use this form set level rule (level 2) to send a transaction to a specific print batch based on the number of pages the system generates when it processes the transaction.

NOTE: The BatchingByPageCountINI rule uses the total page count regardless of recipient and not a specific recipient page count.

Syntax

```
;BatchingByPageCountINI;;;
```

You specify which transactions are assigned to the print batch using INI options in these control groups:

Control Group	Description
BatchingByRecip	The name is defined by the system, as shown here.
Print_Batches	The name is defined by the system, as shown here.
PrinterInfo	The name is defined by the system, as shown here.
Batch File Name	You define the name of this control group.
Printed Output File	You define the name of this control group.

You must have these control groups in your FSISYS.INI or FSIUSER.INI file.

BatchingByRecip control group

This control group must contain these INI options:

```
< BatchingByRecip >
    DefaultBatch   = DefaultOutput
    Batch_Recip_Def =
```

Use the DefaultBatch option to assign a name to the default batch, such as *DefaultOutput*. Do not enclose the name in quotation marks.

Use the Batch_Recip_Def option to define the conditions the system will use to determine which batch it should choose for each transaction. The syntax for the Batch_Recip_Def option is shown here:

```
Condition; "BatchName"; Recipient
```

You can define a series of these options to specify all of the conditions necessary to determine the desired batching for your transactions.

NOTE: For a complete description of error and manual batches, see the Documaker Server System Reference.

Parameter	Description
Condition	This lets you specify a condition that must be satisfied before the transaction is assigned to the print batch and recipients. You can use the True keyword to set the condition: *True* - Enter this keyword to specify that the condition must always be true. This tells the system to send the form set to the specified print batch if the recipient is specified in the recipient list for the batch. You can also use these user-defined options instead of the keyword: *Search mask* – the search mask consists of one or more offset, data pairs. See Search Criteria on page 270. *Error* - if the error batch flag is set by another rule, send the form set to the specified print batch (if the form's recipient is specified in the recipient list for the batch). For example, an error occurs if the Host Required field is set on a Move_It rule and the data is missing. *Manual* - if the manual batch flag is set by another rule, send the form set to the specified print batch (if the recipient is specified in the recipient list for the batch). You could also use the KickToWip rule. *Condition name* - a condition name defined in the condition table. See Using Condition Tables and the Record Dictionary on page 491.
Batch Name	You can specify the batch name by specifying the recipient name, the batch name, or using a batch name defined in the Record Dictionary. *Recipient name* - use one of the names contained in the Recip_Names control group as batch name. Enclose this name in quotes, such as *"customer"*. *Batch name* - a batch name extract from the extract file. The format is a comma-delimited field: a search mask followed by a blank space and then an offset, followed by the length of name to use. *Dict( )* – a batch name defined in the Record Dictionary. Enclose this name in quotes, such as *"batch1"*. Use the pipe symbol (|) to indicate separate items concatenated together, such as *print1|print2*.
Recipients	You can specify recipients using a keyword (All) or you can list specific recipients. All - Enter this keyword to tell the system to use all recipients in the Recip_Names control group. *list* - a list of recipient names. If you list the names, separate each recipient with a comma or space, such as *customer,agent*.

The system processes the information in this order:

1. If the conditions are met, the print batch you specified is used as the batch for the form set, provided the form set's recipients are specified in the recipient list for the batch. In addition, the system writes the batch record for the form set to the batches for the specified recipients.
2. If a transaction does not meet the condition for the first Batch_Recip_Def option, the system continues through the series of Batch_Recip_Def options until the condition for a Batch_Recip_Def option is met.

3 If a transaction does not meet any of the Batch_Recip_Def criteria, the transaction is placed in the batch specified in the DefaultBatch option. If the DefaultBatch option is not defined, an error occurs. The order in which you list the Batch_Recip_Def options determines how the system determines recipient batches. Put the most likely batches first. Use the All keyword rather than listing all recipients when appropriate.

Print_Batches control group

This control group must include at least one entry for each unique batch name in the BatchingByRecip control group.

```
< Print_Batches >
    Batch name = .\batch\page1
```

Option	Description
Batch name	You define the name and path for each batch file, such as .*batch**page1*.

PrinterInfo control group

This control group must include an entry for each unique batch file name listed in the Print_Batches control group. This control group has the following option:

```
< PrinterInfo >
    Printer = Page1Prt
```

Option	Description
Printer	You define the name for each batch file listed in the Print_Batches control group, such as *Page1Prt*.

BatchFileName control group

You define the name for this control group, such as *Page1*. You must have a BatchFileName control group for each batch name option defined in the Print_Batches control group. This control group must include at least two options. The options are shown here:

```
< Page1 >
    Printer    = (file name)
    PageRange  = 1,3
```

Option	Description
Printer	You define the name for the printed output file, such as *Page1Prt*.
PageRange	The minimum and maximum number of pages, separated by a comma. The example above show one as the minimum with three as the maximum.

PrintedOutputFile control group

You define the name for this control group, such as *Page1Prt*. This control group must include as a minimum one option. There must be a PrintedOutputFile control group for each printer option defined in each BatchFileName control group, as shown here:

```
< Page1Prt >
    Port = .\print\page1prt.pcl
```

Option	Description
Port	You define the name and path for the printed output file, such as *print**page1prt.pcl*.

Example

This single-step processing example sends transactions to print batches based on the number of printed pages the system generated as it processed each transaction. The following INI settings and JDT rules tell the system to place the transaction form sets it generates into specific print batches and printed output files based on the page counts.

If the transaction has...	Print_Batches	Printed Output File
1 page	.\Batch\Page1	.\Print\Page1Prt.pcl
2 to 3 pages	.\Batch\Pages2	.\Print\Pages2Prt.pcl
4 to 10 pages	.\Batch\Pages4	.\Print\Pages4Prt.pcl
more the 10 pages	.\Batch\DefaultOutput	.\Print\Default.pcl

Here is an example of the AFGJOB.JDT file:

```
/* JDT Rules showing use of BatchingByPageCountINI rule   */
<Base Rules>
;RULStandardJobProc;;Always the first job level rule;
….
…
;InitSetrecipCache;;;
;InitPrint;;;  required to execute GenData/GenPrint as single step;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;first transaction level rule when not
using GenTrn;
…
…
;PrintFormset;;; required to execute GenData/GenPrint as single step;
;WriteOutput;;;
;WriteNaFile;;;
;CreateRecordList;;;
;WriteRCBWithPageCount;2;;
;BatchingByPageCountINI;2;;
;ProcessQueue;;PostPaginationQueue;
;PaginateAndPropagate;;;
…
…
```

Here are examples of the FSISYS.INI file control groups and options:

```
< BatchingByRecip >
    DefaultBatch   = DefautOutput
    Batch_Recip_Def= True;"Page1";All
    Batch_Recip_Def= True;"Pages2";All
    Batch_Recip_Def= True;"Pages4";All
< Print_Batches >
    DefaultOutput  = .\Batch\DefaultOutput
    Page1          = .\Batch\Page1
    Pages2         = .\Batch\Pages2
    Pages4         = .\Batch\Pages4
< PrinterInfo >
    Printer        = DefaultOutputPrt
```

```
    Printer        = Page1Prt
    Printer        = Pages2Prt
    Printer        = Pages4Prt
< DefaultOutput >
    Printer        = DefaultOutputPrt
< Page1 >
    Printer        = Page1Prt
    PageRange      = 1,1
< Pages2 >
    Printer        = Pages2Prt
    PageRange      = 2,3
< Pages4 >
    Printer        = Pages4Prt
    PageRange      = 4,10
< DefaultOutputPrt >
    Port           = .\Print\DefaultOutputprt.pcl
< Page1Prt >
    Port           = .\Print\Page1Prt.pcl
< Pages2Prt >
    Port           = .\Print\Pages2Prt.pcl
< Pages4Prt >
    Port           = .\Print\Pages4Prt.pcl
```

In addition to selecting by page count, this rule also has all of the functionality of BatchingByRecipINI rule. Here is an example of how you set it up:

```
< BatchingByRecip >
    Batch_Recip_Def= 30,5;"BATCH1";ALL
    Batch_Recip_Def= 51,1;"BATCH2";ALL
    Batch_Recip_Def= 30,2;"BATCH3";ALL
    Batch_Recip_Def= True;"BATCH4";ALL
    Batch_Recip_Def= True;"BATCH5";ALL
< Batch1 >
    Printer        = Printer1
    PageRange      = 1,9999
< Batch2 >
    Printer        = Printer2
    PageRange      = 1,9999
< Batch3 >
    Printer        = Printer3
    PageRange      = 1,9999
< Batch4 >
    Printer        = Printer4
    PageRange      = 1,6
< Batch5 >
    Printer        = Printer5
    PageRange      = 7,9999
```

The first Batch_Recip_Def option tells the system to place into *BATCH1* all recipients which have a *5* at offset 31. If a transaction does not meet the first condition, processing continues through the INI list. Processing stops once the appropriate batch is found.

Therefore, if no condition is met by the third option, the transaction is assigned to *BATCH4* or *BATCH5* based on the page count. If the page count is less than seven, it is assigned to *BATCH4*. If the page count is sever or greater, it is assigned to *BATCH5*.

The order in which you list the Batch_Recip_Def options determines how the system determines recipient batches. Put the most likely batches first. Use *All* rather listing all recipients when appropriate.

Keep in mind that when using this rule in this manner you should *always* include the PageRange parameter in each group, even if the batch is not associated with page counts.

See also

Rules Used for 2-up Printing on page 27

BatchByPageCount on page 47

BatchingByRecipINI on page 68

KickToWip on page 372

Move_It on page 393

WriteRCBWithPageCount on page 250

JDT Rules Reference on page 30

BatchingByPageCountPerRecipINI

Use this form set level rule (level 2) to send a transaction to a specific print batch based on the number of page count for each recipient of all form sets the system generates when it processes the transaction.

Syntax

```
;BatchingByPageCountPerRecipINI;;;
```

You specify which transactions are assigned to the print batch using INI options in these control groups:

- BatchingByRecip
- Print_Batches
- PrinterInfo
- *BatchFileName* (You can define the name of this control group.)
- *PrintedOutputFile* (You can define the name of this control group.)

You must have these control groups in your FSISYS.INI or FSIUSER.INI file.

BatchingByRecip control group

This control group must contain these INI options:

```
< BatchingByRecip >
    DefaultBatch   = DefaultOutput
    Batch_Recip_Def=
```

Use the DefaultBatch option to assign a name to the default batch, such as DefaultOutput. Do not enclose the name in quotation marks.

Use the Batch_Recip_Def option to define the conditions the system should use to determine which batch it should choose for each transaction. The syntax for the Batch_Recip_Def option is shown here:

```
Condition; "BatchName"; Recipient
```

You can define a series of these options to specify all of the conditions necessary to determine the desired batching for your transactions.

Parameter	Description
Condition	This lets you specify a condition that must be satisfied before the transaction is assigned to the print batch and recipients. You can use the True keyword to set the condition: *True* - Enter this keyword to specify that the condition must always be true.This tells the system to send the form set to the specified print batch if the recipient is specified in the recipient list for the batch. You can also use these user-defined options instead of the keyword: *Search mask* - The search mask consists of one or more offset, data pairs. *Error* - If the error batch flag is set by another rule, send the form set to the specified print batch (if the form's recipient is specified in the recipient list for the batch). For example, an error occurs if the Host Required field is set on a Move_It rule and the data is missing. *Manual* - If the manual batch flag is set by another rule, send the form set to the specified print batch (if the recipient is specified in the recipient list for the batch). You could also use the KickToWIP rule. *Condition name* - a condition name defined in the condition table. *?XDB token* - a token name that equates to a named item in the XDB. *=GVM (expression)* - returns the value of a GVM symbol named in the expression. *=DAL (expression)* - returns the value of a DAL script named in the expression. *=(expression)* - returns the value of a DAL symbol represented in the expression.
BatchName	You can specify the batch name by specifying the recipient name, the batch name, or using a batch name defined in the Record Dictionary. *Recipient name* - use one of the names contained in the Recip_Names control group as batch name. Enclose this name in quotes, as shown here: `"customer"` *Batch name* - a batch name extract from the extract file. The format is a comma-delimited field: a search mask followed by a blank space and then an offset, followed by the length of name to use. *Dict( )* - a batch name defined in the Record Dictionary. Enclose this name in quotes, as shown here: `"batch1"` Use the pipe symbol (|) to indicate separate items concatenated together, as shown here: `print1|print2`
Recipients	You can specify recipients by using a keyword (All) or by listing specific recipients. *All* - Enter this keyword to tell the system to use all recipients in the Recip_Names control group. *list* - a list of recipient names. If you list the names, separate each recipient with a comma or space, such as *customer, agent*.

The system processes the information in this order:

1. If the conditions are met, the print batch you specified is used as the batch for the form set, provided the form set's recipients are specified in the recipient list for the batch. In addition, the system writes the batch record for the form set to the batches for the specified recipients.

2 If a transaction does not meet the condition for the first Batch_Recip_Def option, the system continues through the series of Batch_Recip_Def options until the condition for a Batch_Recip_Def option is met.

3 If a transaction does not meet any of the Batch_Recip_Def criteria, the transaction is placed in the batch specified in the DefaultBatch option. If the DefaultBatch option is not defined, an error occurs. The order in which you list the Batch_Recip_Def options determines how the system determines recipient batches. Put the most likely batches first. Use the All keyword rather than listing all recipients when appropriate.

Print_Batches control group

This control group must include at least one entry for each unique batch name in the BatchingByRecip control group.

```
< Print_Batches >
    Batch name = ..\batch\AllOnePageBatch.bch
```

Option	Description
Batch name	You define the name and path for each batch file, such as ..\batch\AllOnePageBatch.bch

PrinterInfo control group

This control group must include an entry for each unique batch file name listed in the Print_Batches control group.

```
< PrinterInfo >
    Printer = Printer1
```

Option	Description
Printer	You define the name for each batch file listed in the Print_Batches control group, such as Printer1.

BatchFileName control group

You define the name for this control group, such as *AllOnePageBatch*. You must have a BatchFileName control group for each batch name option defined in the Print_Batches control group. This control group must include at least these two options. The options are shown here:

```
< AllOnePageBatch >
    Printer    = (file name)
    PageRange  = 1,1
```

Option	Description
Printer	You define the name for the printed output file, such as Printer1.
PageRange	Enter the minimum and maximum number of pages, separated by a comma. The example above shows one as the minimum and the maximum.

PrintedOutputFile control group

You define the name for this control group, such as *Printer1*. This control group must include as a minimum one option. There must be a PrintedOutputFile control group for each printer option defined in each BatchFileName control group, as shown here:

```
< Printer1 >
    Port = ..\PrintFiles\AllOnePageBatch.PCL
```

Option	Description
Port	You define the name and path for the printed output file, such as ..\PrintFiles\AllOnePageBatch.PCL

INI File Examples

Here are examples of the FSISYS.INI file control groups and options for several different scenarios

Scenario 1

In this scenario, we will be showing how to set up a True condition. This tells the system to send the form set to the specified print batch if the recipient is specified in the recipient list for the batch. Here is an example from the FSISYS.INI file:

```
< BatchingByRecip >
    DefaultBatch    = Default
    Batch_Recip_Def = True;"INSURED1PAGE";INSURED
    Batch_Recip_Def = True;"COMPANY1PAGE";COMPANY
    Batch_Recip_Def = True;"AGENT1PAGE";AGENT
    Batch_Recip_Def = True;"INSUREDMULTIPAGE";INSURED
    Batch_Recip_Def = True;"COMPANYMULTIPAGE";COMPANY
    Batch_Recip_Def = True;"AGENTMULTIPAGE";AGENT
    Batch_Recip_Def = Manual;"MANUAL";ALL
    Batch_Recip_Def = Error;"ERROR";ALL
```

For each recipient, all one-page form sets go into one batch as shown here:

```
< Insured1Page >
    Printer = Printer1
    PageRange = 1,1
< Company1Page >
    Printer = Printer2
    PageRange = 1,1
< Agent1Page >
    Printer = Printer3
    PageRange = 1,1
```

All forms sets with two or more pages for each recipient go into a different batch, as shown here:

```
< InsuredMultipage >
    Printer    = Printer4
    PageRange  = 2,99999
< CompanyMultipage >
    Printer    = Printer5
    PageRange  = 2,99999
< AgentMultipage >
    Printer    = Printer6
    PageRange  = 2,99999
< Default >
    Printer    = PDefault
```

Form sets that need to go into WIP are in the manual batch:

```
< Manual >
    Printer    = Printer7
    PageRange  = 1,99999
```

Form sets with errors go into the error batch:

```
< Error >
    Printer    = Printer8
    PageRange  = 1,99999
```

This excerpt shows how to set the Print_Batches, PrinterInfo, and PrintedOutputFile control groups:

```
< Print_Batches >
    Default = default.bch
    Insured1Page = insured1page.bch
    Company1Page = company1page.bch
    Agent1Page = agent1page.bch
    InsuredMultipage = insuredmultipage.bch
    CompanyMultipage = companymultipage.bch
    AgentMultipage = agentmultipage.bch
    Manual = manual.bch
    Error = error.bch
< Printer1 >
    Port = data\insured1page.pcl
< Printer2 >
    Port = data\company1page.pcl
< Printer3 >
    Port = data\agent1page.pcl
< Printer4 >
    Port = data\insuredmultipage.pcl
< Printer5 >
    Port = data\companymultipage.pcl
< Printer6 >
    Port = data\agentmultipage.pcl
 < Printer7 >
    Port = data\manual.pcl
< Printer8 >
    Port = data\error.pcl
< PDefault >
    Port = data\pdefault
< PrinterInfo >
    Printer = Printer1
    Printer = Printer2
    Printer = Printer3
    Printer = Printer4
    Printer = Printer5
    Printer = Printer6
    Printer = Printer7
    Printer = Printer8
    Printer = PDEFAULT
```

Scenario 2

This scenario defines two simple conditions in a condition table based on information in the data dictionary. In the condition table two conditions are set which define the two company types, representing the two different company transactions in the extract file. Condition 1 (Cond1) searches for an *S* for Sampco company transactions. Condition 2 (Cond2) searches for an *F* for FSI company transactions. This scenario creates batches for recipients by company and page count. The data dictionary and condition table for this scenario are shown below:

From the data dictionary:

```
<Records>
Header = Search(11,HEADERREC)
<Variables>
CompanyType = Record(Header) Offset(1) Length(1) Type(Char)
```

From the condition table:

```
< Conditions >
Cond1 : CompanyType = "S"
Cond2 : CompanyType = "F"
```

From the FSISYS.INI file:

```
< BatchingByRecip >
    DefaultBatch    = Default
    Batch_Recip_Def = COND(Cond1);"SAMPCO1PAGE";INSURED
    Batch_Recip_Def = COND(Cond1);"SAMPCOMULTIPAGE";INSURED
    Batch_Recip_Def = COND(Cond2);"FSI1PAGE";INSURED
    Batch_Recip_Def = COND(Cond2);"FSIMULTIPAGE";INSURED
    Batch_Recip_Def = Manual;"MANUAL";ALL
    Batch_Recip_Def = Error;"ERROR";ALL
< Default >
    Printer = PDefault
```

For each recipient, all one-page form sets for each company go into a separate batch as shown here:

```
< Sampco1Page >
    Printer    = Printer1
    PageRange  = 1,1
< FSI1Page >
    Printer    = Printer3
    PageRange  = 1,1
```

For each recipient, all form sets with two or more pages go into a separate batch for each company as shown here:

```
< SampcoMultipage >
    Printer    = Printer2
    PageRange  = 2,99999
< FSIMultipage >
    Printer    = Printer4
    PageRange  = 2,99999
```

Form sets that go to WIP are put into the manual batch:

```
< Manual >
    Printer    = Printer5
    PageRange  = 1,99999
```

Form sets with errors go into the error batch:

```
< Error >
    Printer = Printer6
    PageRange = 1,99999
```

This excerpt shows how to set the Print_Batches, PrinterInfo, and PrintedOutputFile control groups:

```
< Print_Batches >
    Default = default.bch
    Sampco1Page = sampco1page.bch
    SampcoMultipage = sampcomultipage.bch
    FSI1Page = fsi1page.bch
    FSIMultipage = fsimultipage.bch
    Manual = manual.bch
    Error = error.bch
< Printer1 >
    Port = data\sampco1page.pcl
< Printer2 >
    Port = data\sampcomultipage.pcl
< Printer3 >
    Port = data\fsi1page.pcl
< Printer4 >
    Port = data\fsimultipage.pcl
< Printer5 >
    Port = data\manual.pcl
< Printer6 >
    Port = data\error.pcl
< PDEFAULT >
    Port = data\pdefault
< PrinterInfo >
    Printer = Printer1
    Printer = Printer2
    Printer = Printer3
    Printer = Printer4
    Printer = Printer5
    Printer = Printer6
    Printer = PDefault
< Tables >
    Path = .\tables\
    Recipient = reciptbl.dat
    Conditions = condition.tbl
< DataDictionary >
    Name = datadict.tbl
< SymLookup >
    MaxCache = 1000
    LeastFrequent = Yes
```

Scenario 3

This scenario looks for a token from the XDB to send form sets to agent batches. Here is an excerpt from the FSISYS.INI file:

```
< BatchingByRecip >
    DefaultBatch    = Default
    Batch_Recip_Def = ?AGENT NAME;"AGENTNAME1PAGE";AGENT
    Batch_Recip_Def = ?AGENT NAME;"AGENTNAMEMULTIPAGE";AGENT
    Batch_Recip_Def = Manual;"MANUAL";ALL
    Batch_Recip_Def = Error;"ERROR";ALL
```

If the token is not found, the system sends form sets to the Default batch:

```
< Default >
    Printer    = PDefault
```

If a token is found, the system sends all one-page transactions to an Agent batch specifically for one-page form sets:

```
< AgentName1Page >
    Printer    = Printer1
    PageRange  = 1,1
```

If a token is found, the system sends all transactions that are more than one page to an Agent batch designed to hold forms sets that consist of two or more pages:

```
< AgentNameMultipage >
    Printer    = Printer2
    PageRange  = 2,99999
```

Form sets that go to WIP are put into the manual batch:

```
< Manual >
    Printer    = Printer3
    PageRange  = 1,99999
```

Form sets with errors go into the error batch:

```
< Error >
    Printer    = Printer4
    PageRange  = 1,99999
```

This excerpt shows how to set the Print_Batches, PrinterInfo, and PrintedOutputFile control groups:

```
< Print_Batches >
    Default = default.bch
    AgentName1Page = agentname1page.bch
    AgentNameMultipage = agentnamemultipage.bch
    Manual = manual.bch
    Error = error.bch
< Printer1 >
    Port = data\agentname1page.pcl
< Printer2 >
    Port = data\agentnamemultipage.pcl
< Printer3 >
    Port = data\manual.pcl
< Printer4 >
    Port = data\error.pcl
< PDefault >
    Port = data\pdefault
< PrinterInfo >
```

```
Printer = Printer1
Printer = Printer2
Printer = Printer3
Printer = Printer4
Printer = PDefault
```

Scenario 4

Like the previous scenario, this scenario sends form sets to agent batches depending on the number of pages. This scenario, however, uses a GVM variable as the condition and an XML extract file. In the AFGJOB.JDT, you must first create the global variable you are going to use for the condition. In this scenario, it is called AGT1. To create it, use the CreateGlbVar rule. Then, use the Ext2GVM rule to map the data to the GVM variable named AGT1. This rule is placed after the LoadExtractData rule in the AGFJOB.JDT file. If the GVM variable (AGT1) holds a value, the condition is considered true and the transaction is written to the appropriate batch by the page count. If the GVM variable (AGT1) does not hold a value, the condition is considered false and the transaction will be written to the Default batch. Here is an example:

```
/* This base (this implementation) uses these rules. */
<Base Rules>
;RULStandardJobProc;1;Always the first job level rule;
;SetErrHdr;1;*:;
;SetErrHdr;1;*:-----------------------------------------------;
;SetErrHdr;1;*: FormMaker Data Generation (Base);
;SetErrHdr;1;*: ;
;SetErrHdr;1;***: Transaction:  ***PolicyNum***;
;SetErrHdr;1;***: Symbol:  ***Symbol***;
;SetErrHdr;1;***: Module:  ***Module***;
;SetErrHdr;1;***: State:  ***State***;
;SetErrHdr;1;***: Company Name (after ini conversion):
***Company***;
;SetErrHdr;1;***: Line of Business (after ini conversion):
***Lob***;
;SetErrHdr;1;***: Trans Type:  ***TransactionType***;
;SetErrHdr;1;***: Run Date:     ***Rundate***;
;SetErrHdr;1;*:-----------------------------------------------;
;CreateGlbVar;1;TXTLst,PVOID;
;CreateGlbVar;1;TblLstH,PVOID;
;CreateGlbVar;1;AGT1,CHAR_ARRAY,15;
;JobInit1;1;;
;LoadDDTDefs;1;;
;InitOvFlw;1;;
;LoadTextTbl;1;;
;LoadTblFiles;1;;
;SetOvFlwSym;1;CGDECBDOVF,Q1GDBD,5;
;BuildMasterFormList;1;4;
<Base Form Set Rules>
;RULStandardTransactionProc;2;Always the first transaction level
rule;
;LoadExtractData;2;;
;GetCo;2;11,HEADERREC 35,3;
;GetLOB;2;11,HEADERREC 40,3;
;Ext2Gvm;2;!/COMPANY/FORMS/FORM/SECTION/FIELDS/AGENTNAME 1,15,AGT1;
;ResetOvFlw;2;;
;BuildFormList;2;;
;LoadRcpTbl;2;;
;UpdatePOLFile;2;;
```

```
;RunSetRcpTbl;2;;
;BatchingByPageCountPerRecipINI;;;

Here is an example of the FSISYS.INI file:
< BatchingByRecip >
    DefaultBatch    = Default
    Batch_Recip_Def = =GVM("AGT1"),;"AGENTNAME1PAGE";AGENT
    Batch_Recip_Def = =GVM("AGT1"),;"AGENTNAMEMULTIPAGE";AGENT
        Batch_Recip_Def = Manual;"MANUAL";ALL
    Batch_Recip_Def = Error;"ERROR";ALL
< Default >
    Printer = PDefault
```

If the GVM variable holds a value, the system sends all one-page transactions to an Agent batch specifically for one-page form sets:

```
< AgenName1Page >
    Printer    = Printer1
    PageRange  = 1,1
```

If the GVM variable holds a value, the system sends all transactions that are more than one page to an Agent batch designed to hold form sets that consist of two or more pages:

```
< AgentNameMultipage >
    Printer    = Printer2
    PageRange  = 2,99999
```

Form sets that go into WIP are put in the manual batch:

```
< Manual >
    Printer    = Printer3
    PageRange  = 1,99999
```

Form sets with errors go into the error batch:

```
< Error >
    Printer    = Printer4
    PageRange  = 1,99999
```

This excerpt shows how to set the Print_Batches, PrinterInfo, and PrintedOutputFile control groups:

```
< Print_Batches >
    Default = default.bch
    AgentName1Page = agentname1page.bch
    AgentNameMultipage = agentnamemultipage.bch
    Manual = manual.bch
    Error = error.bch
< Printer1 >
    Port = data\agentname1page.pcl
< Printer2 >
    Port = data\agentnamemultipage.pcl
< Printer3 >
    Port = data\manual.pcl
< Printer4 >
    Port = data\error.pcl
< PDefault >
    Port = data\pdefault
< PrinterInfo >
    Printer = Printer1
```

```
Printer = Printer2
Printer = Printer3
Printer = Printer4
Printer = PDefault
```

Scenario 5

Like scenario 4, this scenario sends form sets to agent batches depending on the number of pages. This scenario, however, uses a DAL script called agent.dal to set the condition along with using an XML extract file. In the AFGJOB.JDT, we must use a PreTransDAL to call the DAL script to set the condition. In this scenario, it is called AGT1. This rule is placed after the RunSetRcpTbl rule in the AGFJOB.JDT file. Once the DAL script set a value to the DAL variable, the condition is considered true and the transaction is written to the respective batch for the Batch_Recip_Def condition by page count. If the DAL script does not set a value to the DAL variable, the condition is considered false and the transaction is written to the Default batch instead. Here is an example:

```
/* This base (this implementation) uses these rules. */
<Base Rules>
;RULStandardJobProc;1;Always the first job level rule;
;SetErrHdr;1;*:;
;SetErrHdr;1;*:-----------------------------------------------;
;SetErrHdr;1;*: FormMaker Data Generation (Base);
;SetErrHdr;1;*: ;
;SetErrHdr;1;***: Transaction:  ***PolicyNum***;
;SetErrHdr;1;***: Symbol:  ***Symbol***;
;SetErrHdr;1;***: Module:  ***Module***;
;SetErrHdr;1;***: State:  ***State***;
;SetErrHdr;1;***: Company Name (after ini conversion):
***Company***;
;SetErrHdr;1;***: Line of Business (after ini conversion):
***Lob***;
;SetErrHdr;1;***: Trans Type:  ***TransactionType***;
;SetErrHdr;1;***: Run Date:     ***Rundate***;
;SetErrHdr;1;*:-----------------------------------------------;
;CreateGlbVar;1;TXTLst,PVOID;
;CreateGlbVar;1;TblLstH,PVOID;
;JobInit1;1;;
;LoadDDTDefs;1;;
;InitOvFlw;1;;
;LoadTextTbl;1;;
;LoadTblFiles;1;;
;SetOvFlwSym;1;CGDECBDOVF,Q1GDBD,5;
;BuildMasterFormList;1;4;
<Base Form Set Rules>
;RULStandardTransactionProc;2;Always the first transaction level
rule;
;LoadExtractData;2;;
;GetCo;2;11,HEADERREC 35,3;
;GetLOB;2;11,HEADERREC 40,3;
;ResetOvFlw;2;;
;BuildFormList;2;;
;LoadRcpTbl;2;;
;UpdatePOLFile;2;;
;RunSetRcpTbl;2;;
;PreTransDAL;;Call("agent.dal");
;BatchingByPageCountPerRecipINI;;;
```

Here is an example of the FSISYS.INI file:

```
< BatchingByRecip >
    Batch_Recip_Def = =DAL("AGT1"),;"AGENTNAME1PAGE";AGENT
    Batch_Recip_Def = =DAL("AGT1"),;"AGENTNAME1PAGE";AGENT
    Batch_Recip_Def = Manual;"MANUAL";ALL
    Batch_Recip_Def = Error;"ERROR";ALL
    DefaultBatch    = Default
```

If the DAL variable holds a value, the system sends all one-page transactions to an Agent batch specifically for one-page form sets:

```
< AgentName1Page >
    Printer    = Printer1
    PageRange  = 1,1
```

If the DAL variable holds a value, the system sends all transactions that are more than one page to an Agent batch designed to hold form sets that consist of two or more pages:

```
< AgentNameMultipage >
    Printer    = Printer2
    PageRange  = 2,99999
```

If the DAL variable does NOT hold a value, the condition is considered false and the transaction is sent to the Default batch.

```
< Default >
    Printer = PDefault
```

Form sets that go into WIP are put in the manual batch:

```
< Manual >
    Printer    = Printer3
    PageRange  = 1,99999
```

Form sets with errors go into the error batch:

```
< Error >
    Printer    = Printer4
    PageRange  = 1,99999
```

This excerpt shows how to set the Print_Batches, PrinterInfo, and PrintedOutputFile control groups:

```
< Print_Batches >
    Default = default.bch
    AgentName1Page = agentname1page.bch
    AgentNameMultipage = agentnamemultipage.bch
    Manual = manual.bch
    Error = error.bch
< Printer1 >
    Port = data\agentname1page.pcl
< Printer2 >
    Port = data\agentnamemultipage.pcl
< Printer3 >
    Port = data\manual.pcl
< Printer4 >
    Port = data\error.pcl
< PDefault >
    Port = data\pdefault
< PrinterInfo >
```

```
Printer = Printer1
Printer = Printer2
Printer = Printer3
Printer = Printer4
Printer = PDefault
```

See also JDT Rules Reference on page 30

BatchingByRecipINI

Use this form set level (level 2) rule to send transactions to a batch you specify based on data in the extract file and conditions and recipients specified using INI options.

Syntax

```
;BatchingByRecipINI;;;
```

You pass parameters for this rule using the BatchingByRecip control group in your FSISYS.INI or FSIUSER.INI file. Make sure this control group is in the INI file and includes these options:

- DefaultBatch
- Batch_Recip_Def

The DefaultBatch option has only one parameter, a literal name such as *Default*. Do not enclose it in quotation marks.

The syntax for the Batch_Recip_Def option is shown here:

```
Condition; "BatchName"; Recipient
```

Parameter	Description
Condition	You can use these keywords: *COND* A condition name defined in the condition table. *Error* If the error batch flag is set by another rule, send the form set to the specified batch (if the form's recipient is specified in the recipient list for the batch). For example, an error occurs if the Host Required field is set on a Move_It rule and the data is missing. *Manual* If the manual batch flag is set by another rule; send the form set to the specified batch (if the recipient is specified in the recipient list for the batch). An example of another rule you could use is the KickToWip rule. *True* The condition is always true; send the form set to the specified batch (if the recipient is specified in the recipient list for the batch). *Search mask* The search mask consists of one or more offset,data pairs.
Batch name	*Batch name* from the extract file. The format is a comma-delimited field: a search mask followed by a blank space and then an offset, followed by the length of name to use. Keep in mind the batch name is limited to eight characters. You can use these keywords: *Recip_Name* means to use the names contained in the Recip_Names control group as batch names *Dict( )* – The batch name defined in the Record Dictionary. Enclose this name in quotes, such as *"Batch1"*. Use the pipe symbol (|) to indicate separate items concatenated together.
Recipients	Enter *All* for all recipients or list the recipient names. If you list the names, separate each recipient with a comma or space.

If the conditions are met, the batch you specified is used as the batch for the form set, provided the form set's recipients are specified in the recipient list for the batch. In addition, the system writes the batch record for the form set to the batches for the specified recipients.

If a transaction does not meet the first condition, processing continues through the INI list. Processing stops once the appropriate batch is found.

If a transaction does not meet one of the Batch_Recip_Def criteria, the system places it in the default batch. If you do not define the DefaultBatch option, an error occurs.

The order in which you list the Batch_Recip_Def options determines how the system determines recipient batches. Put the most likely batches first. Use *All* rather than listing all recipients when appropriate.

Example

For this example, assume you have this rule in the AFGJOB.JDT file:

```
;BatchingByRecipINI;;;
```

And these INI options:

```
< BatchingByRecip >
     DefaultBatch  = default
     Batch_Recip_Def = 4,1234567;"BATCH1";INSURED
     Batch_Recip_Def = true;"BATCH2";INSURED
     Batch_Recip_Def = true;"BATCH5";COMPANY AGENT
```

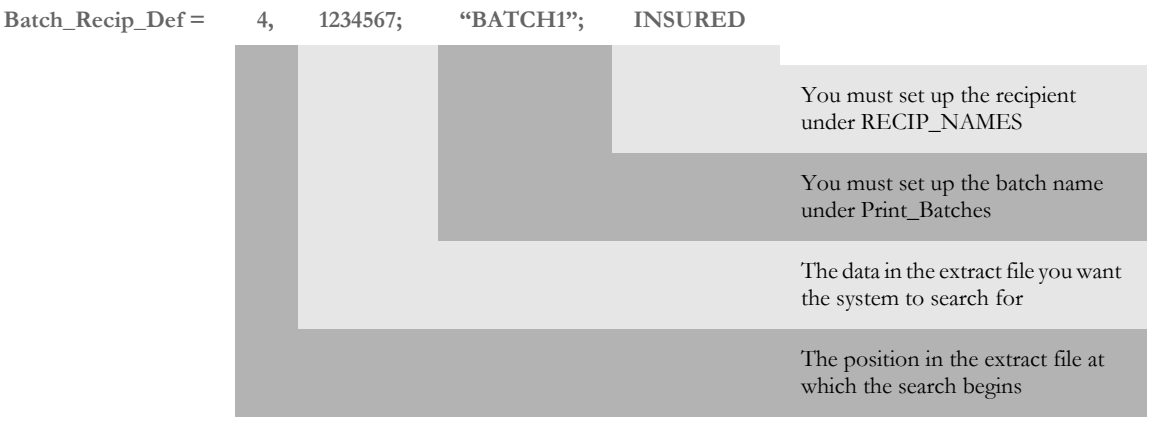
The DefaultBatch option tells the system that any output which has not already been sent to a batch by one of the Batch_Recip_Def options should be placed in the default batch:

```
DefaultBatch = default
```

You must set up the batch name under the Print_Batches control group.

The first Batch_Recip_Def option tells the system to place into *BATCH1* any output which goes to the *INSURED* recipient and has *1234567* beginning at position *4* in the extract file:

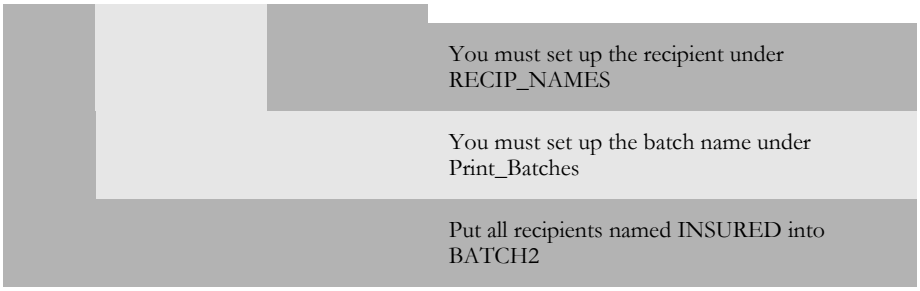
```
Batch_Recip_Def = 4,1234567;"BATCH1";INSURED
```



The next Batch_Recip_Def option tells the system to place all recipients named INSURED into BATCH2:

```
Batch_Recip_Def = true;"BATCH2";INSURED
```

Batch_Recip_Def = true; "BATCH2"; INSURED



This Batch_Recip_Def option follows the same syntax as the earlier examples, but shows how you can use the pipe symbol to place two segments on one line:

```
Batch_Recip_Def = true;"BATCH"|"5";COMPANY AGENT
```

The last Batch_Recip_Def option tells the system to place all recipients named *COMPANY* and *AGENT* into the concatenated name *BATCH5*.

As shown earlier, you have to specify the batch name under Print_Batches and the recipient name under RECIP_NAMES.

See also

BatchByPageCount on page 47

BatchingByPageCountINI on page 49

PrintFormset on page 182

SetOutputFromExtrFile on page 221

Search Criteria on page 270

Using Condition Tables on page 492

Using the Record Dictionary on page 495

JDT Rules Reference on page 30

BuildExcludeList

Use this job level rule (level 1) to selectively exclude transactions from being processed in one- and two-step mode processing.

You must include this rule in the AFGJOB.JDT file because neither one- nor two-step mode executes the GenTran program which processes transactions during multi-step processing.

NOTE: You must define the transactions to be excluded in a file specified by the Exclude option in the Data control group. In addition, this rule does not remove excluded transactions from the extract file as would occur in multi-step processing.

Syntax

```
;BuildExcludeList;;;
```

Example

Assume your master resource library has the following items defined. Your FSIUSER.INI file looks like this:

```
< Data >
    Exclude = Exclude.dat
```

You have a file named JONES.DAT in DefLib. This file contains:

```
30,Jones
```

Your AFGJOB.JDT file looks like this:

```
<Base Rules>
;RulStandardJobProc;;;
;JobInit1;;;
;BuildMasterFormList;;4;
;BuildExcludeList;;;
…
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
…
```

Your extract file looks like this:

```
RG1001 HEADER CWNGCIS 030201 Roberts      J
…
RG1002 HEADER CWNGCIS 030201 Brown        T
…
RG1003 HEADER CWNGCIS 030201 Jones        M
…
RG1004 HEADER CWNGCIS 030201 Smiths       K
…
RG1005 HEADER CWNGCIS 030201 Jones        L
…
```

In this example the system processes all of the transactions except RG1003 and RG1005. These transactions are excluded because the search mask criteria (35,Jones) defined in the EXCLUDE.DAT file was found in both transactions.

See also JDT Rules Reference on page 30

BuildFormList

Use this form set level rule (level 2) to load and initialize all forms that the processing of extract data could possibly produce.

NOTE: You can use the LoadFormsetFromArchive rule to replace the BuildFormList, LoadRcpTbl, and RunSetRcpTbl rules in the AFGJOB.JDT file.

Syntax

```
;BuildFormList;;;
```

There are no parameters for this rule.

This rule loads the entire form set into a master form set list and creates the duplicate list which is altered during processing. The system only creates the master form set list the first time it executes this rule. Subsequent calls to this rule delete the existing duplicate form set (in the working form set) and recreate it from the master form set list.

This rule makes sure the extract data exists and resets all form recipient copy counts to zero (0). It also loads the set recipient table data from file into a list. This file is defined in the SetRcpTb option in the Data control group.

NOTE: This rule erases the list of data from the SETRCPTB file.

Example

```
;BuildFormList;;;
```

See also

LoadFormsetFromArchive on page 154

LoadRcpTbl on page 157

RunSetRcpTbl on page 212

JDT Rules Reference on page 30

BuildMasterFormList

Use this job level rule (level 1) to load the FORM.DAT file into an internal linked list used by the GenData program.

You must include this rule in the AFGJOB.JDT file because the RunSetRcpTbl rule is dependent on the list this rule creates.

Syntax

```
;BuildMasterFormList;;KeyCount;FORM.DAT
```

Parameter	Description
KeyCount	This is a required integer that *must be* set to 4.
FORM.DAT	List the FORM.DAT files you want the system to load. This lets you load multiple FORM.DAT files and have them appear in memory as if they came from one large FORM.DAT file. If you do not specify the FORM.DAT file name, the system looks for the master resource library settings to find the correct file to load.

Example

```
;BuildMasterFormList;1;4;
```

The KeyCount parameter determines the number of items in the FORM.DAT line considered part of the form set key. The system organizes the form set list based on the number of items specified by the KeyCount parameter, minus one.

For example, the RPEX1 FORM.DAT file contains as its first two lines:

```
;SAMPCO;LB1;DEC PAGE;;R;;qsname|...
;SAMPCO;LB1;LETTER;;RD;;qsname|D...
```

This rule compares the lines to determine if an item is already in the list after finding the 4th (KeyCount) semicolon in each line and comparing up to the lesser position. These lines would be compared up to the following point:

```
;SAMPCO;LB1;DEC PAG;
;SAMPCO;LB1;LETTER;
```

NOTE: The KeyCount parameter should be *one more* than the number of keys in the FORM.DAT file to allow for the leading semicolon. Do not change the KeyCount parameter unless the library uses a different number of keys.

See also

RunSetRcpTbl on page 212

JDT Rules Reference on page 30

CheckZeroFontID

Use this form set level (level 2) rule to see if the form set has any fields with a zero font ID. The rule will produce an error or warning for any fields it finds which have font IDs equal to zero. You can then correct the font ID problems and restart the processing cycle.

Syntax

```
;;CheckZeroFontID;;Message;
```

Parameter	Description
Message	Enter E if you want to see error messages. Leave this parameter blank if you want to see warning messages.

Example

This example produces error messages:

```
;CheckZeroFontID;;E;
```

This example produces warning messages:

```
;CheckZeroFontID;;;
```

The error or warning message includes information about the form, section, and field:

Message	Description
DM30059	<Error> in CheckZeroFontID: zero font ID: form <PXWORKSHT> image <PXPOLICY> field <ESTFONTID>
DM30059	<Warning> in CheckZeroFontID: zero font ID: form <PXWORKSHT> image <PXPOLICY> field <TESTFONTID>

See also JDT Rules Reference on page 30

ConvertWIP

Use this form set level (level 2) rule to see if the current transaction is assigned to the MANUAL.BCH file. If it is, the rule adds the record to WIP and unloads the contents of the POLFILE.DAT and NAFILE.DAT files into new files with unique names.

The system generates unique form set IDs using a globally unique identifier (GUID) for the new files. This helps to make sure form set IDs created for WIP records do not clash even if multiple applications are generating WIP records, such as if you had multiple IDS servers generating WIP.

You can then view these WIP records using Documaker Workstation or Print Preview, which is part of the Internet Document Server (IDS).

Using this rule eliminates the need to run the separate GenWIP process to transfer transactions into WIP.

NOTE: You must have separate licenses to run Documaker Workstation and IDS. Contact your sales representative for more information.

Syntax

```
;ConvertWIP;;;
```

Example

```
;ConvertWIP;2;;
```

The order in which you place the ConvertWIP rule is important. Place it in front of the PrintFormset, WriteOutput, and WriteNAFile rules, as shown here:

```
;NoGenTrnTransactionProc;2;required to combine GenTran/GenData;
;ConvertWIP;2;;
;PrintFormset;2;;
;WriteOutput;2;;
;WriteNAFile;2;;
```

The PrintFormset rule is required to combine the GenData and GenPrint processes into a single step.

See also

InitConvertWIP on page 142

JDT Rules Reference on page 30

CreateGlbVar

Use this job level rule (level 1) to create a global variable which can be used by all code in the system. You specify the type and size of the variable. List all instances of this rule at the beginning of the AFGJOB.JDT file.

Syntax

```
;CreateGlbVar;;;
```

NOTE: This rule resets the global variable created during pre-processing.

Example

```
;CreateGlbVar;;VARIABLENAME, VARTYPE, VARSIZE;
```

You can create each global variable as shown below. VARIABLENAME is an arbitrary name of the variable which will be created, VARTYPE is the type of variable to create, and VARSIZE is an optional size of the variable to create.

The variable types are:

- SHORT
- LONG
- DOUBLE
- FLOAT
- LONG DOUBLE
- CHAR_ARRAY
- PVOID

The following example creates a character array of 20 bytes named MYCHARARRAY. Remember that the array should be large enough to include the null terminating character if it contains strings.

```
;CreateGLBVar;1;MYCHARARRAY,CHAR_ARRAY,20;
```

The next example creates a variable named MYLONGVAR that is a LONG:

```
;CreateGlbVar;1;MYLONGVAR,LONG;
```

Notice that no size was included so the variable size will be a single long value.

See also JDT Rules Reference on page 30

CreateRecordList

Use this form set level rule (level 2) to copy the NA_Offset and POL_Offset into global variables.

This rule calculates page counts for each recipient and sends transactions to error, manual, and previously assigned (such as Braille) batches, as necessary. This rule also appends RCB comment records for all other transactions into a generic linked list of objects.

This rule writes out the recipient batch records for error, manual, and previously assigned batches.

NOTE: If the current record will be appended to the generic linked list of objects, this rule takes care of filling in a number of fields in the recipient batch record. These fields would normally be filled in by a call to the RULUpdateRecips function. The RULUpdateRecips function is inappropriate for page count batching, which is why the fields are filled in manually.

Syntax

```
;CreateRecordList;;;
```

Example

```
;CreateRecordList;;;
```

See also JDT Rules Reference on page 30

DelExtRecords

Use this form set level rule (level 2) to search the extract data list and delete all data records which match the search mask you specify.

For instance, you can use this rule to remove data records that are confidential, out of date, or should never be processed on the form.

To use this rule, you must add it to the AFGJOB.JDT file after the extract data is loaded but before the extract data is used for mapping or other purposes.

Syntax

```
;DelExtRecords;;;
```

Example

This example deletes all extract records which contain the text *RECTYPE1* at offset 1, and *KEY1* at offset 50.

```
;DelExtRecords;;1,RECTYPE1,50,KEY1;
```

NOTE: Be sure that the search mask is specific enough to avoid deleting more records than intended.

See also JDT Rules Reference on page 30

Dictionary

Use this job level rule (level 1) to terminate an XDB instance and free memory. You only include this rule if you also used the GlobalFld rule.

NOTE: For information on the Dictionary Editor, see the Docucreate User Guide.

Syntax

```
;Dictionary;;;
```

Example

Here is an example of how you would use this rule:

```
/* JDT Rules for One Step Batching By Recipient           */
/*
*/
<Base Rules>
;RULStandardJobProc;;;
;SetErrHdr;;***:-----------------------------------------------;
;SetErrHdr;;***: Oracle Insurance
;SetErrHdr;;***: Company Name:  ***Company***;
;SetErrHdr;;***: Application:   ***Application***;
;SetErrHdr;;***: Account #:     ***Account_Number***;
;SetErrHdr;;***:-----------------------------------------------;
;JobInit1;;;
;CreateGlbVar;;RCBBatchName,CHAR_ARRAY,32;
;InitOvFlw;;;
;SetOvFlwSym;;MTROVF,qaIMTROV,1;
;SetOvFlwSym;;RTEOVF,qaIRTEOV,1;
;Dictionary;;;
;BuildMasterFormList;1;4;
;PageBatchStage1InitTerm;;;
;InitSetrecipCache;;;
;InitPrint;;required to execute gendata/genprint as single step;
```

See also

GlobalFld on page 340

JDT Rules Reference on page 30

DocumentExport

Use this form set (level 2) rule to write full NA and POL information as well as certain export field information. You can control how the export information the rule generates is formatted and you can specify which fields should be included.

Syntax

```
;DocumentExport;;;
```

Defining Export Options

You use these options in the ImpExpCombined control group to control this rule:

```
< ImpExpCombined >
    File =
    Path =
    Ext =
    AppendedExport =
```

Option	Description
File	Enter the name you want to assign to the file. If you omit this name, the user must specify it.
Path	Enter a path to indicate where the file should be written. The default is the current working directory. If you include the File option and the file name contains a path, that path overrides this option.
Ext	Enter the extension you want to use. The default is DS. If you include the File option and the file name includes an extension, that extension overrides this option.
AppendedExport	If you enter Yes, the data on the current form set data is appended to the existing file. The default is No.

Defining the Export Record

You must know the format of the record you intend to export. This includes having a list of all the fields that comprise the record and the lengths and formats of those fields.

If you omit fields from the ImpExpCombined control group, the export record will contain the information specified for the Trigger2WIP control group in the order it is listed. The default layout is a fixed record length. If the fields are omitted from both the ImpExpCombined and Trigger2WIP control groups, an error occurs.

Surround each field element within the export record with field separators. This helps input systems parse the field information. The default field separators are quotation marks (“), as shown here:

```
WIP = “data”“data”“data”
```

Fixed or variable record lengths

Fixed length records are always the same. In a fixed length record, all field data has a known (output) size that must be generated. Adding all the field lengths together, generally equals the fixed record length. (Remember to add two separator strings for each field to determine the actual length.)

Variable record length output typically means that some or all of the data element will not adhere to a fixed size.

With variable length records, you often need a marker that identifies where each field element begins and ends. These elements are typically constant —meaning each field begins and ends with the same value. This is the purpose of the field separator mentioned previously. The field separator defaults to a quotation mark ("), but you can specify any string of up to 39 characters.

Delimiting the fields is often necessary because the importing program needs to recognize where fields begin and end within the variable record. Although a fixed length record may not always require field separators, there is no harm caused by using them.

Listing the field source, length, and format

Build a table with this information:

- Each field that will be contained within each record
- Any length requirements of the field data
- Any special formatting requirements for writing the data to the output record.

With a fixed length record layout, each field has a specific length. In variable length records, a field may have no specific length requirement or may have a minimum or a maximum length requirement or both. Note the length requirements for each field.

Finally, note any special formatting requirements for each field. When composing this information, keep in mind that the manner in which the data is formatted within the export record may be different that the export record requirement. This would include information such as the following:

- Should the data be left or right justified in the output?
- For date fields, what format should be used (such as Month/Day/Year or Year/Month/Day)?
- For numeric data, should values include or not include commas, dollar signs, and so on?
- Should a decimal number be converted to integers (should 41.1 written as 41)?
- Should integers be written as decimals (41 becomes 41.00)?
- Is there constant or filler data that should be written for a given export record field that will not be derived from a form set field? For example, you might want to write a given value into all records for a certain field because the importing program requires such a value.

Defining the export fields and formats

Once you have identified the fields that comprise the export record, you have to define these fields and the formats required to build the defined record layout.

The record layout for the record must be defined in the INI file, in this control group:

```
< ImpExpCombined >
```

For each field that should be exported to the export record line, you must include a line in the INI file. You may export as many fields as necessary. Each line must begin with the FIELD= statement and has the following syntax.

```
FIELD = GVM Fieldname;formatstring
```

You can omit the GVM Fieldname from any record location that must contain constant or filler information not derived from the actual export record. See Format Specification Flags on page 88 for more information.

The *formatstring* is optional. If you include it, be sure to precede it with a semicolon. Whatever occurs after the semicolon is used to modify the field data in a specific manner. You can use format string flags to increase or decrease the data to a predetermined size or convert the data from one value format to another.

If a named export field is not followed by the format string, the format is derived internally by querying the GVM definition and will yield a constant string length result.

Specifying the format

If the output record area for a definition is a *filler* area — meaning that the output data is predetermined and not part of the export data — you can omit the field name. If you omit the field name in this manner, you must specify a format or nothing is written to the output record.

Constant data is often used to write header or trailer information to variable length records to help the importing system recognize where the record begins and ends. Also, you can use this method to write additional characters or data between fields such as, for example, if you need to include a comma between each field data element of a variable length record. Here are some examples:

```
FIELD = Key1;%-3.3s
FIELD = ;,
FIELD = KeyID;%-10.10s
```

In the first case, the value from Key1 is formatted as three characters in the output data. This constant text value of a comma is written next, followed by 10 characters from KeyID. If you assume Key1 contains *BOB* and KeyID contains *123456789*, the result is as follows:

```
WIP = "BOB"",""123456789 "
```

Notice that each field, whether constant or not, contains field separators. Also notice that the data for KeyID was padded to 10 characters even though the actual value only contained nine characters.

Converting dates

Date format conversions are specified using the *D* as the first character. The *D* is followed by the input format specifier for the data, a semicolon, and the format specification for output. Here is an example:

```
FIELD=CREATETIME;DX;D4
```

This example names the field CREATETIME. The *D* following the semicolon tells the system to retrieve this field and convert it from the format defined as *X* into the output format, D4. *X* represents the hexadecimal character format. *D4* is a standard YYYYMMDD format without separators.

Format Flags

If you are familiar with C programming, the data conversions provided with format flags will be familiar. Essentially, the printf function format definitions for %s, %f, and %d are supported with some limitations.

Remember that most export record data and other internal data is usually text. Therefore, to convert to a numerical format of %f or %d, the form set data must be deformatted internally and then converted into the required format.

The output written to the exported record is formatted as text. Here are some examples:

```
FIELD = DESC;%d
FIELD = ORIGUSER;%-32.32s
FIELD = APPDATA;%8.2f
```

These examples use format flags. The first example retrieves the value of the field DESC (the description) then converts that value into an integer (losing any decimal portion it might have had) and outputs it as an integer value. If the data was not a number, the result is zero (0).

The second example writes exactly 32 characters for the value taken from ORIGUSER. If the field value does not contain 32 characters, it is padded with spaces. Note also the use of the dash (-) indicator. This tells the system to left justify the field. If you omit the dash, the system pads the data with spaces on the left, right justifying it.

The last example demonstrates a floating point output with two decimal places. The field value is converted into a floating point number. The system then applies the format you specify and rounds the value if it contained more than two decimal places.

Defining the Export Record Header

The export record header occurs at the beginning of each export record output.

```
< ImpExpCombined >
    WIPHeader = WIP=
    Separator = ”
```

Option	Description
WIPHeader	Enter the text for the header. You can enter as much text as you like, but avoid exceeding 1024 characters in the entire export record line. The default is WIP.
Separator	Enter the text value used to separate fields. The default is a quotation mark (“). If a variable record layout is being used for WIP information, field separators are essential. You can enter up to 39 characters. Choose a separator that is not likely to appear in any of the data you intend to output.

Date Formats

Standard date format

You can enter dates in a variety of formats. The date format has three possible components or characters. The first character specifies the order of the date. The second character specifies the type of separator character for the date. The third character specifies the length of the year.

Date order

The first character in the date format indicates the order of the date and whether the month should be numeric or alphabetic. The first character must be a digit from 1 to 9 or an alphabetic character. The default order is format 1. The following table lists your options:

Format	Date order	Description
1	MM/DD/YY	Month-Day-Year with leading zeros (02/17/2009)
2	DD/MM/YY	Day-Month-Year with leading zeros 17/02/2009
3	YY/MM/DD	Year-Month-Day with leading zeros 2009/02/17
4	Month D, Yr	Month name-Day-Year without leading zeros (February 17, 2009)
5	bM/bD/YY	Month-Day-Year with leading zeros replaced with spaces (2/17/2009)
6	bD/bM/YY	Day-Month-Year with leading zeros replaced with spaces (17/ 2/2009)
7	YY/bM/bD	Year-Month-Day with leading zeros replaced with spaces (2009/ 2/17)
8	M/D/YY	Month-Day-Year with leading zeros suppressed (12/8/2009)
9	D/M/YY	Day-Month-Year with leading zeros suppressed (17/2/2009)
A	YY/M/D	Year-Month-Day with leading zeros suppressed (2009/8/9)
B	MMDDYY	Month-Day-Year with no separators (02172009)
C	DDMMYY	Day-Month-Year with no separators (17022009)
D	YYMMDD	Year-Month-Day with no separators (20090217)
E	MonDDYY	Month name abbreviated-Day-Year with leading zeros (Feb072009)
F	DDMonYY	Day-Month name abbreviated-Year with leading zeros (07Feb2009)
G	YYMonDD	Year-Month name abbreviated-Day with leading zeros (2009Feb07)
H	day/YY	Day of year (counting consecutively from January 1)-Year (48/2009)
I	YY/day	Year-Day of Year (counting consecutively from January 1) (often called the Julian date format) (2009/48)

Format	Date order	Description
J	D, Month Yr	Day- Month name-Year without leading zeros (7, February 2009)
K	Yr, Month D	Year-Month name-Day without leading zeros (2009 January 5)
L	Mon-DD-YY	Month name abbreviated-Day-Year with leading zeros (Feb-17-2009)
M	DD-Mon-YY	Day-Month name abbreviated-Year with leading zeros (02-Feb-2009)
N	YY-Mon-DD	Year-Month name abbreviated-Day with leading zeros (2009-Feb-17)
X	XXXXXXXX	An eight-character hexadecimal representation

Separators

The second character in the date format indicates the separator to use in the date. If you omit the separator character, the system includes a forward slash (/). You can choose from these separator characters:

Character	Example
/	02/17/2009
-	02-17-2009
.	02.17.2009
,	02,17,2009
b (blank)	02 17 2009

You specify the separator character by including it as the second character in the date format. For example, if you enter *5-*, you specify date format 5 with a dash as a separator character. The date appears as 02-17-2009. If you enter *5b*, you specify date format 5 with blanks or spaces as a separator. Your date appears as 02 17 2009.

Year length

The third character in the date format specifies the year length. The year must appear as either two or four digits. Enter 2 for a two digit year or 4 for a four digit year.

You can omit the year length character from a date format. If you do not specify the year length, the system uses the length of the original entry. For example, if you enter a date as 10/30/09 and do not specify a length, the system retains *05*. If you enter 10/30/2009, the system retains *2009*.

If you enter *5-2*, you specify date format 5, a dash (-) as a separator character and a two-digit year. Your date appears as _9-11-09. If you enter *5-4*, your date appears as _9-11-2009.

NOTE: If you do not enter a separator character the year length specification is the second digit in the date format. For example, if you enter *54*, you specify date format 5 and a four-digit year. Since the separator character is not specified the default character (/) applies. Your date appears as _9/11/2009.

Avoid two-digit year representations. For example, if you enter 5/2, you specify date format 5 and a two-digit year. Your date appears as 9/11/09.

Freeform Formats

The format argument consists of one or more codes; each formatting code is preceded by a percent sign (%). Characters not prefixed with a percent sign copied unchanged to the output buffer. Any character following a percent sign is not recognized as a valid format code is copied unchanged to the destination. Therefore, you can enter %% to include the percent sign in the resulting output string.

You can use these format codes:

Code	Description
%d	Day of month as decimal number (01 – 31)
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01 – 12)
%m	Month as decimal number (01 – 12)
%M	Minute as decimal number (00 – 59)
%p	Current locale's AM/PM indicator for 12-hour clock
%S	Second as decimal number (00 – 59)
%y	Year without century, as decimal number (00 – 99)
%Y	Year with century, as decimal number
%A	Weekday name, such as Tuesday
%b	Abbreviated month name, such as Mar
%B	Full month name, such as March
%j	Day of year as decimal number, such as 001–366
%w	Weekday as decimal number, such as 1 – 7 with Sunday as 1
%@***xxx***	Specify language locale (Where *xxx* identifies one of the supported languages. For example. A format of *%@CAD%A* might produce *mardi*, the French word for Tuesday.)

Here are some examples:

Format	Output
%m-%d-%Y	01-01-2009
The year is %Y.	The year is 2009.
Born %m/%d/%y at %I:%M %p	Born 01/01/09 at 11:57 PM

Additional format attributes

An octothorp (#) tells the system to suppress leading zeros for the following format codes. This flag is recognized on these formats and is ignored on all other format codes not listed here.

```
%#d, %#H, %#I, %#j, %#m, %#M, %#S, %#w
```

For example, if %d outputs *01*, using %#d the output will become *1*.

NOTE: This flag only affects the format code that specifies it. Any subsequent codes that are numeric are not affected unless they also specify the flag.

Enter a greater than symbol (>) to uppercase the resulting text. This flag is only recognized on these format codes:

```
%>p, %>A, %>b, %>B
```

For example, if %A results in *Tuesday*, %>A produces *TUESDAY*.

NOTE: This flag only affects the format code that specifies it. Any subsequent codes that have text are not affected unless those also specify the flag.

Enter a less than symbol (<) to lowercase the resulting text. This flag is valid for the following codes and ignored on all others:

```
%<p, %<A, %<b, %<B
```

For example, if %b results in *Mar*, %<b produces *mar*.

NOTE: This flag only affects the format code that specifies it. Any subsequent codes that have text are not affected unless they also specify the flag.

Enter <> to capitalize the first letter of the resulting text. This flag is valid for the following codes and ignored on all others:

```
%<>p, %<>A, %<>b, %<>B
```

For example, if %p results in *AM*, %<>p produces *Am*.

NOTE: This flag only affects the format code that specifies it. Any subsequent codes that have text are not affected unless they also specify the flag.

Using Locale Information

When you use the %@***xxx*** in the format string, the *xxx* represents a code that identifies one of our supported language locales.

Until a locale format code is encountered in the format string, the default locale (typically USD which is US English) is in effect. Once a locale format code is found, the locale specified remains in effect until another locale indicator is encountered.

For example: suppose the input date is 03-01-2009 (USD). This table shows the output from various formats:

Enter	To output
" %A, %B %d"	"Monday, March 01".
"%@CAD%A %@CAD%A, %B %d"	"lundi, mars 01"
"%A, %@CAD%B %d"	"Monday, mars 01"
"%@CAD%A, %@USD%B %d"	"lundi, March 01"

Format Specification Flags

The format specification, which consists of optional and required fields, is shown here:

```
%[Flags][Width][.Precision]Type
```

Each field of the format specification is a character or a number which specifies a format option. The simplest format specification contains only the percent sign and a type character, such as: *%s*. If a percent sign is followed by a character that has no meaning as a format field, that character is simply copied to the output. For example, to print a percent sign, enter *%%*.

The optional fields, which appear before the Type character, control other aspects of the formatting, as follows:

Type — Enter *s*, *f*, or *d* for this export function.

Flags — Use these flags to control justification of the output and the printing of signs, blanks, decimal points, and octal and hexadecimal prefixes. More than one flag can appear in a format specification.

Flag	Description	Default
–	Left aligns the result within the given field width.	Right align.
+	Prefixes the output value with a sign (+ or –) if the output value is of a signed type.	Sign appears only for negative signed values (–).
0	Adds zeros until the minimum width is reached. If a zero and a minus appear (-0), the system ignores the zero. If you include a zero with an integer format (d), the system ignores the zero flag.	No padding.

Flag	Description	Default
blank (' ')	Prefixes the output value with a blank if the output value is signed and positive; the blank is ignored if both the blank and + flags appear.	No blank appears.
#	When used with the f format, the # flag forces the output value to contain a decimal point in all cases.	Decimal point appears only if digits follow it.

Width

Here you can control the minimum number of characters printed. If the number of characters in the output value is less than the width you specify, the system adds blanks to the left or the right of the values — depending on whether the flag for left alignment is specified — until the minimum width is reached. If you prefix the width with a zero (0), the system adds zeros until the minimum width is reached (not useful for left-aligned numbers).

Your entry for width never causes a value to be truncated. If the number of characters in the output value is greater than the width you specify, or if you omit the width, all characters of the value are printed (subject to the **.**Precision specification).

.Precision

This optional number specifies the maximum number of characters printed for all or part of the output field, or the minimum number of digits printed for integer values.

For format *s*, the precision specifies the maximum number of characters to print. Characters in excess of precision are not printed. Characters are printed until a null character is encountered.

For format *f*, the precision specifies the number of digits after the decimal point. If a decimal point appears, at least one digit appears before it. The value is rounded to the appropriate number of digits. The default precision is six (6); if the precision is zero (0), or if a period (.) appears without a number following it, no decimal point is printed.

For format *d,* the precision specifies the minimum number of digits to be printed. If the number of digits in the argument is less than the precision value, the output value is padded on the left with zeros. The value is not truncated when the number of digits exceeds the precision. The default precision is one (1).

See also

JDT Rules Reference on page 30

DumpExtList

Use this form set level rule (level 2) to dump the extract list to a file. The result provides information about the generic linked list of objects, such as its handles and the information contained in each element.

The file the system creates is a flat text file which shows only text values—binary data is written as spaces to preserve placement.

Syntax

```
;DumpExtList;;(Name);
```

Parameter	Description
Name	Name of the file to which the extract records will be written.

NOTE: Only use this rule for test purposes, so you can inspect the contents of the extract data list. The use of this rule slows processing in proportion to the size of the extract data list.

Example

```
;DumpExtList;;ExtrListDump.txt;
```

See also JDT Rules Reference on page 30

DumpExtractListToFile

Use this form set level rule (level 2) to dump the extract list to a file. This rule is helpful if you are debugging and you want to see what is currently in the extract list.

Syntax

```
;DumpExtractListToFile;;(parameters);
```

Parameter	Description
Name	Name of the file to which the extract records will be written.
Flag	The append flag. Enter *Y* to append the records for each transaction to the file plus a header. Enter *N* to only create a dump file for the last transaction.

Example

```
;DumpExtractListToFile;;ExtrDump.txt,Y;
```

For each transaction in the extract file, the system appends to the EXTRDUMP.TXT file a transaction header plus the contents of each record in the transaction. Here is an example of this file:

```
************************************************
==> Extract data for:
    TransactionId:<Patch399>
    GroupName1:<CWNG>
    GroupName2:<CIS>
    GroupName3:<>
    External Form Name:<>
    Transaction Type:<>
************************************************
Patch399                    HEADER    CWNGCIS   Patch # …
Patch399                    GVM1      Morris    Sandra …
…
…
************************************************
==> Extract data for:

    TransactionId:<Patch400>
    GroupName1:<CWNG>
    GroupName2:<CIS>
    GroupName3:<>
    External Form Name:<>
    Transaction Type:<>
************************************************
Patch400                    HEADER    CWNGCIS   Patch # …
Patch400                    GVM1      Bob       Jane …
…
…
```

NOTE: The Y option can create a very large file, depending on the size of the extract file.

See also JDT Rules Reference on page 30

ErrorHandler

Use this job level (level 1) rule to send transactions to the manual batch when specified field errors occur. This lets normal processing continue if errors occur.

Syntax

```
;ErrorHandler;;;
```

Use the following INI option to identify the field errors which cause a transaction to be sent to the manual batch:

```
< Error2Manual >
    (CurrentError) = (NextError1),..,(NextErrorN),(M)
```

Parameter	Description
CurrentError	The error that just occurred.
NextError1	The error caused by the CurrentError failure. This continues through the list of errors until the last one.
M	This tells the system to send the transaction to the manual batch.

Errors

Here is a sample set of error messages which would appear if a field error occurred:

Error	Description
DM10513	Error in SetAddr(): Empty RuleParms for SetAddr.
DM12051	Error in RPProcessOneField(): Unable to <SETADDR>().
DM12048	Error in RPProcessFields(): Unable to RPProcessOneField(pRPS) <ADDR3>. Processing will NOT continue for image <q1addr>. See INI group:< GenDataStopOn > option: FieldErrors.
DM12083	Error in RPProcessOneImage(): Unable to RPProcessFields(pRPS).
DM12074	Error in RPProcessImages(): Unable to RPProcessOneImage(pRPS) <q1addr>. Skipping the rest of the Images for this form. See INI group:< GenDataStopOn > option:ImageErrors.

Example

Here is an example of the FSISYS.INI file:

```
< Error2Manual >
    10513 = 12051, 12048, 12083, 12074
< GenDataStopOn >
    FieldErrors = No
```

Add this rule to the < Base Rules > section in the AFGJOB.JDT file:

```
;ErrorHandler;;;
```

If field error DM10513 occurs, it will cause these errors: DM12051, DM12048, DM1283, and DM12074. This function sends these transactions to the manual batch for user-entry, continues processing, and then creates a blank field in the NAFILE.DAT file.

See also

JDT Rules Reference on page 30

Ext2GVM

Use this form set level rule (level 2) to add data from the extract list into previously defined global variables. To use the rule, you must add it to the AFGJOB.JDT file after the extract data is loaded.

You can also use this rule to get data into the NEWTRN.DAT file during GenData processing instead of using Trn_Fields. To do this you define a field for the data to be mapped in the TRNDFDFL.DFD file and then use the Ext2GVM rule to map the data from extract file to the NEWTRN.DAT file.

Syntax

```
;Ext2GVM;;(parameters);
```

You can use these parameters with this rule:

Parameter	Description
SearchMask	One or more pairs of offsets and data (search criteria) in a comma-delimited list.
DataLocation	The offset and length of the data in the extract record.
GVMName	The name of the GVM variable where the data will be stored.
SuppressFlag	Enter *S* to suppress error messages if the search mask is not found in a transaction. The default is blank, which tells the system not to suppress error messages.

Example

The following example locates the extract record that matches the search mask (1, D1) and moves the value found at position 21 for a length of 5 to the global variable TestVar. In addition, it suppresses the error messages if the search mask is not found in the transaction.

```
<Base Rules>
...     ...
...     ...
;CreateGlbVar;;TestVar,CHAR_ARRAY,5;
...     ...
<Base Form Set Rules>
...     ...
;Ext2GVM;;1,D1 21,5,TestVar,S;
...     ...
```

See also

JDT Rules Reference on page 30

NOTE: Refer to the DAL Reference for information on these related DAL functions: GVM, HaveGVM, and SetGVM.

FilterForm

Use this form set level (level 2) rule to remove all forms from a form set except those that match the filter criteria you specify.

See also

```
;FilterForm;;;
```

You can use these INI options with this rule:

```
< FilterForms >
    Form           =
    FilterByForm   =
```

Option	Description
Form	Use this option to specify the search location for form filter criteria. All occurrences of the data specified in the data are used for filtering. You can specify the search location as an XML search string or as a flat file search mask.
FilterByForm	(Optional) Use this option to turn the rule on or off by transaction. If you specify this option, the rule looks for a value of TRUE at that specified search location. If TRUE is not found, the filter logic is not executed. If you omit this option, the rule is always executed.

Here is an example of how you can use the Form option:

```
Form = 1,HEADER 20,8    (Offset,Match Offset,Length)
```

In the following example, assume you have this INI setting:

```
< FilterForm >
    Form = !/transaction/PrintForm
```

And this transaction data:

```
<transaction>
    …
    <PrintFrom>FormA</PrintForm>
    <PrintForm>FormB</PrintForm>
    …
</transaction>
```

Only forms FormA and FormB will remain in the form set after the filtering process is complete.

Example

Here is an example AFGJOB.JDT file:

```
/*  JDT Rules for Single-Step Processing Batching By Recipient. */
<Base Rules>
;RULStandardJobProc;1;Always the first job level rule;
;JobInit1;1;;
;InitPrint;1;required to execute gendata/genprint in single step;
/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;2;;
;LoadFormsetFromArchive;2;;
;PrintFormset;2;;
;WriteOutput;2;;
```

```
;WriteNaFile;2;;
;BatchingByRecipINI;2;;
;FilterForm;2;;
/* Every section in this base uses these rules. */
<Base Image Rules>
;WIPImageProc;;
/* Every field in this base uses these rules. */
<Base Field Rules>
;RULWIPFieldProc;;
```

See also FilterRecip on page 96

JDT Rules Reference on page 30

FilterRecip

Use this form set level (level 2) rule to remove all forms from a form set except those that match the recipient filter criteria you specify.

Syntax

```
;FilterRecip;;;
```

You can use these INI options with this rule:

```
< FilterRecip >
    Recip          =
    FilterByRecip  =
```

Option	Description
Recip	Use this option to specify the search location for the Recip filter criteria. All occurrences of the data specified in the data are used for filtering. You can specify the search location as an XML search string or as a flat file search mask.
FilterByRecip	(Optional) Use this option to turn the rule on or off by transaction. If you include this option, the rule looks for a value of TRUE at the specified search location. If TRUE is not found, the filter logic is not executed. If you omit this option, the rule is always executed.

Here is an example of how you can use the Recip option:

```
Recip = 1,HEADER 20,8   (Offset,Match Offset,Length)
```

Filtering is performed using all data that matches the search criteria. Assume you have this INI setting:

```
< FilterRecip >
    Recip = !/transaction/PrintRecip
```

And this transaction data:

```
<transaction>
    …
    <PrintRecip>Insured</PrintRecip>
    <PrintRecip>Agent</PrintRecip>
    …
</transaction>
```

Only recipients Insured and Agent will remain in the form set after the filtering process is complete.

Example

Here is an example AFGJOB.JDT file:

```
/*  JDT Rules for Single-Step Processing Batching By Recipient. */
<Base Rules>
;RULStandardJobProc;1;Always the first job level rule;
;JobInit1;1;;
;InitPrint;1;required to execute gendata/genprint in single step;
/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;2;;
;LoadFormsetFromArchive;2;;
;PrintFormset;2;;
```

```
;WriteOutput;2;;
;WriteNaFile;2;;
;BatchingByRecipINI;2;;
;FilterRecip;2;;
/* Every section in this base uses these rules. */
<Base Image Rules>
;WIPImageProc;;
/* Every field in this base uses these rules. */
<Base Field Rules>
;RULWIPFieldProc;;
```

See also FilterForm on page 94

JDT Rules Reference on page 30

ForceNoImages

Place this rule in your AFGJOB.JDT file to bypass all section processing. This rule prevents any section level rules from executing.

For instance, you could use this rule if the form set for each transaction is created and mapped by a higher level rule which removes the necessity for executing section or field level rules.

You can also use this rule in 2-up printing to return the msgNO_MORE_IMAGES message.

Syntax

```
;ForceNoImages( )
```

There are no parameters for this rule.

This rule prevents errors if you have no section level rules.

Example

```
< Base Image Rules >
;ForceNoImages;;
```

See also ImportNAPOLExtract on page 121

Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

FormDescription

Use this form set level (level 1) rule to write out a form or several forms, which contain descriptions of the other forms included in the form set. You can also specify a DAL script you want the system to execute to get the actual descriptions to use.

NOTE: This capability exists in Documaker Workstation. This rule provides the capability to Documaker Server.

Syntax

```
;FormDescription;;NoOverflow;
```

Parameter	Description
NoOverflow	(Optional) This parameter tells the system not to overflow the form description line section if there are not enough Form Description Line fields to include the maximum number of selected forms. Keep in mind that if this parameter is turned on and there are more forms than there are Form Description Line fields, some of the descriptive information may be lost. The default is the system will overflow to accommodate all selected forms.

Keep in mind...

- Place this rule after the UpdatePOLFile rule if you are running in multi-step mode. For single- and two-step mode, place the rule after the PaginateAndPropagate rule.
- The names of Form Description Line variable fields must begin with *FORM DESC LINE*. You can include multiple lines of these fields on a form by varying the field's name, such as *FORM DESC LINE #002*, *FORM DESC LINE #003*, and so on.
- For each form in a form set (and optionally for each Key2 grouping), the system will assign to a Form Description Line field a text description of that form. Only one text description is assigned to each From Description Line field.
- Form description lines do not wrap the description to succeeding lines. If a text is longer than the field's representation, the text can extend beyond page boundaries or into undesirable areas. Make sure the Form Description Line fields can contain the longest description. Smaller fonts generally allow more characters per line.
- Any form can contain Form Description Line fields. You can place form description lines on separate forms. You can also place forms with these fields among other fields, graphics, and so on.
- The placement of the form in the FORM.DAT file is important. Only those forms placed after the first form which contains a FORM DESC LINE field will be included in the listed forms. Please note the first form, which contains the form description lines, *is not* included in the list.

INI options

You can use these INI options to tell the system how to represent form group lines on form description lines.

```
<  FormDescTable >
    IncludeKey2     = No
    BoldKey2        = No
    Key2Prefix      =
    Key2PostInc     = 0
```

```
    IncludeDuplicateForms  = No
    IncludeFormName        = No
    StartFromFirstForm     =
    ColumnFormat           = No
    ExludedForm            =
    IncludeFormDesc        =
< FormDescription >
    Script                 =
```

Option	Description
FormDescTable control group	
IncludeKey2	Enter Yes to enable Key2 descriptions. By default, the form description lines only contain descriptions of the forms. Optionally, you can include descriptions for form groups, such as lines of business. These form groups are called Key2s.
BoldKey2	Use this option to present Key2 descriptions in a bold font. The system determines which font to use by querying the font defined on the field and selecting its bold equivalent. The fonts of normal Form Description Lines fields (not assigned a Key2 name) are changed to their non-bold counterparts.
Key2Prefix	Enter the text you want to appear before each Key2 description line. The system automatically adds a single space after the text. By default, this option is blank and does not affect the description lines. For instance, if you enter *Form Applicable* – the system prefixes all Key2 descriptions with that text. The output might look like this: `Form Applicable – General Liability Coverage`
Key2PostInc	Use this option to add blank lines between Key2 descriptions the form descriptions. For example, if you include Key2 descriptions, and set this option to one, you out put might look like this: `Form Applicable – General Liability Coverage` `Form 1Automobile Coverage` `Form 2Homeowner Coverage`
IncludeDuplicateForms	If you want the system to include duplicate forms, set this option to Yes. The system excludes the duplicate forms from the form description lines as a default.
IncludeFormName	Enter No if you want to suppress the form name from the form description lines. By default, the system includes the form names.
StartFromFirstForm	Enter Yes if you want the system to include forms starting from the first form. By default, only forms placed after the first form that contains a FORM DESC LINE field are included in the list of forms. Note that the form that contains the form description lines is *not* included in the list *unless* this option is turned on.

Option	Description
ColumnFormat	By default, the system writes out the form description lines in a columnar format with the form name on the left and pads the text based on the longest form name. To have the system write out the form description lines in a non-columnar format, set this option to No. If set to No, the system adds the form description to the end of the form name, separated by two spaces.
ExcludedForm	To exclude a certain form from the form description lines, enter the name of the form you want to exclude. Here is an example: `ExludedForm = Form Applicable` To exclude multiple forms, include a separate ExcludedForm option for each form, as shown here: `ExludedForm = Form1` `ExludedForm = Form2` `ExludedForm = Form3`
IncludeFormDesc	Set this option to No if you want to suppress the description from the form description lines. The system includes the form description by default. Do not set this option and the IncludeFormName option to No. If you do, the system writes the form description without the form name
FormDescription control group	
Script	The name of DAL script you want the system to execute.

Example

This example shows how you can call a DAL script to customize the description placed in the Form Description Line fields. To use this functionality, make sure you have the Script option set up in the FormDescription control group:

```
< FormDescription >
    Script = AddDate.DAL
```

Now suppose you want to add *(11/10)* to the end of each description line. You would set up the Script option to call the DAL script that contains the logic to do so. Here is an example of the DAL script:

```
FName = FormName( )
FDesc = FormDesc( )
return(FName  &  "     "  &  FDesc &  " (11/10)");
```

Below is an example of the description lines generated *without* calling DAL script.

```
DEC Page    Common Policy Declarations
END Page    Endorsement Page
CG DEC      General Liability Declarations
```

Here is an example of the output when you call the DAL script:

```
DEC Page    Common Policy Declarations (11/10)
END Page    Endorsement Page (11/10)
CG DEC      General Liability Declarations (11/10)
```

See also

PaginateAndPropagate on page 174

UpdatePOLFile on page 239

JDT Rules Reference on page 30

GetCo

Use this form set level rule (level 2) to get the company code (Key1 field) from the extract data, and get its equivalent value from the INI file for use by the system.

Syntax

```
;GetCo;;;
```

This rule gets the company code (Key1 field) from the extract data using the GetRecord search criteria specified in the data field. This rule also sets the company—stored in the master transaction set—to the value defined in the INI file. In the INI file, this value is stored in the Key1Table control group under the option name which equals the value returned by GetRecord.

Example

```
;GetCo;;17,PMSP0200  125,3;
```

In this example, the system searches the extract data for the first record that meets the GetRecord search criteria of having PMSP0200 at offset 17. From this record, the system extracts three characters at position 125, and uses those characters to look up the company in the INI file. It is this equivalent value from the INI file that should be used in files such as the form set definition file, as well as the set recipient table file.

These files are defined in the Data control group with the names FORMDAT and SETRCPTB respectively. For example, if the three characters at position 125 were ABC, the associated line in the Key1Table control group would look something like:

```
< Key1Table >
    ABC = THE ABC FRUIT COMPANY
```

See also

GetLOB on page 104

Search Criteria on page 270

JDT Rules Reference on page 30

GetLOB

Use this form set level rule (level 2) to get the line of business (from the Key2 field) code from the extract data, and to get its equivalent value from the INI file for use by the system.

Syntax

```
;GetLOB;;;
```

This rule gets the line of business (Key2 field) code from the extract data using the GetRecord search criteria specified in the data field. This rule also sets the line of business—stored in the master transaction set—to the value defined in the INI file. In the INI file, this value is stored in the Key2Table control group under the option name which equals the value returned by GetRecord.

Example

```
;GetLOB;;17,PMSP0200  100,3;
```

In this example the system searches the extract data for the first record which meets the GetRecord search criteria of having PMSP0200 at offset 17. From this record, the system extracts three characters at position 100, and uses those characters to look up the line of business in the INI file. It is this value in the FSISYS.INI file which should be used in files such as the FORM.DAT file, as well as the set recipient table.

These files are defined in the Data control group with the names FORMDAT and SETRCPTB respectively. For example, if in the FSISYS.INI file the three characters at position 100 were CFR, the associated line in the Key2Table control group would look something like:

```
< Key2Table >
    CFR = COMMERCIAL FIRE
```

See also

GetCo on page 103

Search Criteria on page 270

JDT Rules Reference on page 30

GetRCBRec

Use this form set (level 2) level rule to set the current recipient batch file. This rule initializes the current recipient batch file, if necessary.

This rule also sets the first printer for the current batch to be the current printer and retrieves the next record from the current recipient batch file.

Syntax

```
;GetRCBRec;;;
```

Example

```
;GetRCBRec;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

GetRunDate

Use this form set level (level 2) rule when there is no value in the transaction extract lines that can be used as the run date. This rule gets the current date and treats it as the run date. It then reformats the date based on the format you specify in the format mask.

The rule assumes the GVM variable *RunDate* is the name of the variable where the date is stored. The RunDate variable is created if it does not exist. GVM variables are automatically created from the fields defined in your TRNDFDFL.DFD file, or by using the rule that explicitly creates GVM variables. If you omit the variable from being created in one of these methods, this rule creates it for you.

Syntax

```
;GetRunDate;;;
```

The parameters you can include are the various date formats supported by FmtDate rule. The default format is D4 (YYYYMMDD).

NOTE: There are two types of format masks, pre-defined types 1-9 and A-Q and user-defined format arguments. If the pre-defined formats meet your needs, use them, otherwise, create a user-defined format. For information on using pre-defined format types, see Using Pre-defined Date Formats on page 257.

User-defined format arguments consist of one or more codes, each preceded by a percent sign (%). For more information on user-defined format masks, see Setting Up Format Arguments on page 262.

There is no limit to the length of the mask you create.

On success, msgSUCCESS is returned. If the system encounters a fatal error, msgFAIL is returned.

Example

Here are some examples:

```
;GetRunDate;;J;
```

If the system date is 20090217, the date format returned will be *17 February, 2009*.

```
;GetRunDate;;;
```

If the system date is 20090217, the date format returned will be *20090217*.

```
;GetRunDate;;;
```

If the system date is 2-17-2009, the date format returned will be *20090217*.

```
;GetRunDate;;J;
```

If the system date is 2-17-2009, the date format returned will be *17 February, 2009*.

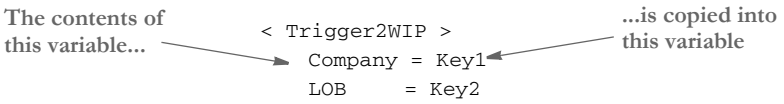
See also

FmtDate on page 337

JDT Rules Reference on page 30

GVM2GVM

Use this form set level (level 2) rule to copy the data from one GVM variable to another GVM variable. You specify the two variables using INI options, as shown in this example:



Syntax

```
;GVM2GVM;;ControlGroup;
```

Parameter	Description
ControlGroup	Specify the name of the control group in the INI file that defines the variables.

NOTE: Although this rule was created for use with GenData WIP Transaction Processing, you can also use it to map a group of GVM variables from one name to another name.

For GenData WIP Transaction Processing, this rule copies GVM data from the WIP.DBF file into GVM variables for GenData execution. You define the GVM variables in the Trigger2WIP control group.

Assume the FSISYS.INI or FSIUSER.INI file has these options:

```
< Trigger2WIP >
    Company = Key1
    LOB     = Key2
```

Example

Here is an example:

```
;GVM2GVM;;Trigger2WIP;
```

This example copies the contents of the Key1 and Key2 GVM variables found in the WIP.DBF file into the GenData Company and LOB GVM variables.

See also

MergeWIP on page 162

WIPFieldProc on page 243

WIPImageProc on page 244

WIPTransactions on page 245

GenData WIP Transaction Processing on page 9

JDT Rules Reference on page 30

IfRecipUsed

Use this form set level rule (level 2) to place a form set in a recipient batch if the recipient name matches the one you specify with this rule. For instance, if the form is triggered and the recipients are set to receive a copy of those sections, a copy of the sections that make up the form are copied to the specified batches.

Syntax

```
;IfRecipUsed;;;
```

There are no parameters for this rule.

Example

```
;IfRecipUsed;;BATCH1=INSURED;
```

If the recipient name placed in the data area (such as INSURED) is used in this form set, the system assigns this form set to the recipient batch named in the data area (such as BATCH1).

You can place multiple AssignToBatch rules in the AFGJOB.JDT file. All that return true will be placed in the appropriate recipient batch. You can assign several recipients to a single batch. This is useful if the recipients receive very few forms or you only want to manage a small number of batch files.

You must place this rule in the AFGJOB.JDT file before the BuildFormList rule.

NOTE: Powertyping takes precedence over all AssignToBatch assignments, and errors take precedence over powertyping.

See also

AssignToBatch on page 45

BuildFormList on page 72

SetOutputFromExtrFile on page 221

JDT Rules Reference on page 30

ImageMapImportData

Use this form set level (level 2) rule with the ImportFile or ImportExtract rule to map the data you are importing. You can also use this rule with any other rule, such as a custom rule, that fills in field dictionary values (like the standard V2 import methods).

Normally, when you use this type of import, you would replace the NoOpFunc rule to do mapping via the DDT files, or you would use the MapFromImportData rule on each field in the DDT file.

If, however, you have two environments — one that does imports and one that does regular batch processing — you may not want to maintain two sets of DDT files. Therefore, you could use this rule if you do not plan to execute any field level rules.

Syntax

```
;ImageMapImportData;;;
```

This rule loads the section and tries to get the data dictionary value of each field created during the import. If you do not want the field level rules to execute in the DDT file, use the JDT rule that skips field processing. Here is an example:

```
<Base Image Rules>
    ;StandardImageProc;3;;
    ;ImageMapImportData;3;;
<Base Field Rules>
    ;WIPFieldProc;4;No field processing;
```

NOTE: While the assumption is that you use this rule when you want to skip normal field processing, there is no requirement that you do so. If you omit the WIPFieldProc rule, the field level rules will execute. Depending on the rules you have assigned to your fields, this may cause errors, or may override the data that was actually imported.

Standard data importing can supply field data at various levels, such as: form set global, form global, and image local.

Each occurrence of a field with the same name and declared using the form set global scope will normally have the same value. Form global scope applies to similarly defined fields only within a given form. Image local scope means the field is specific to that section.

Import files sometimes specify all field data at the section level and do not separate out form set or form global data. This rule first tries to get each field's data at the dictionary scope level defined in the FAP file. If a form set or form global value cannot be found for a field, a second search is done at the section level.

NOTE: This approach supports both types of import files — those that specify all data at the section level and those that separate out data at the appropriately defined scope levels.

See also ImportExtract on page 111

ImportFile on page 116

WIPFieldProc on page 243

MapFromImportData on page 376

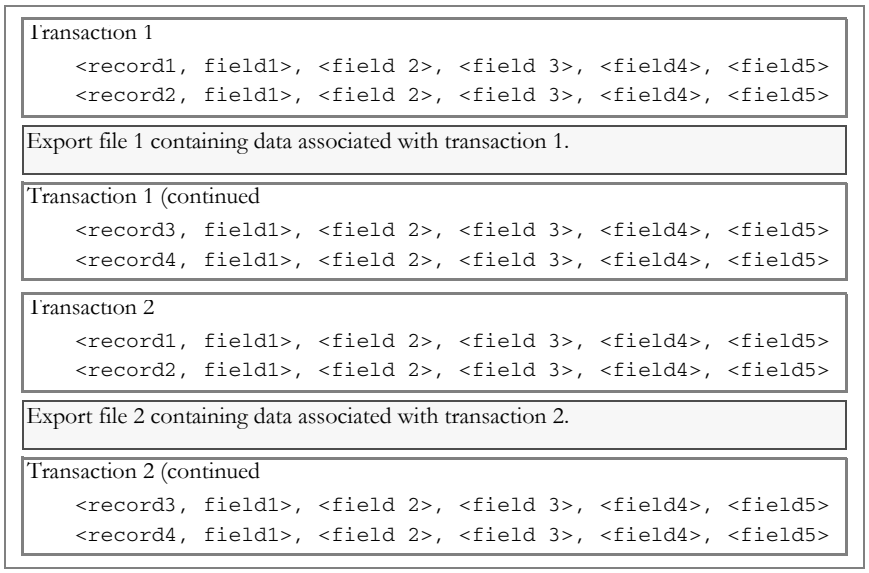
NoOpFunc on page 415

JDT Rules Reference on page 30

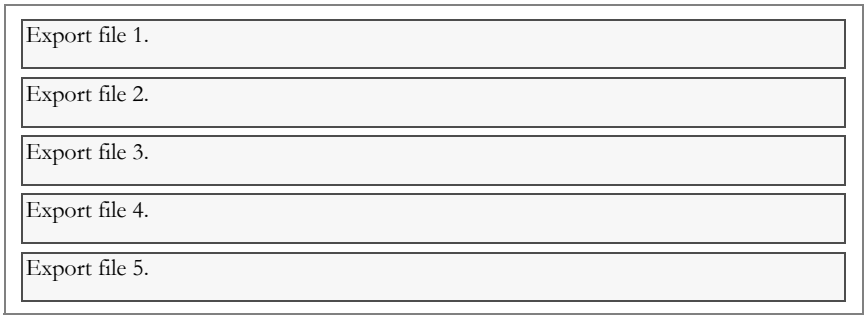
ImportExtract

Use this form set level rule (level 2) to import an extract file into GenData that is comprised of:

- Typical transactions in an extract file which have one or more Documaker Workstation export files embedded in each transaction. This illustration shows transactions with embedded export files:



- One or more appended Documaker Workstation export files. This illustration shows an extract file comprised of export files appended to one another:



Syntax

```
;ImportExtract;;;
```

Although there are no parameters for this rule. Keep in mind:

- Specify the extract file name to be imported via the Data control group using the ExtrFile option.
- In the Trn_File control group, set MaxExtRecLen option to the length of the longest record in extract file.
- Only use the SearchMask option in the ExtractKeyField control group; do not use the Key option.

- To create minimum information, such as Key1, Key2, Key ID, and so on, in the TRNFILE.DAT file for each transaction, you must define for each information item, the field name, offset, and length in the Trn_Fields control group. This definition associates the option fields in the Trn_Fields control group to the corresponding entries in the transaction DFD file (TRNDFDFL.DFD).
- For each field that comprises a section (whose data comes from the export data), you must create or have a DDT record whose field rule is set to one of the following field rules
  - MapFromImportData
  - NoOpFunc

  If you use the NoOpFunc rule for any field rule associated with the export data, you must insert the ReplaceNoOpFunc rule in the <Base Rules> section of the AFGJOB.JDT file.
- You must place the ImportExtract rule in the <Base Form Set Rules> section of the AFGJOB.JDT file after the BuildFormList rule or any custom rule that creates a form set.
- If the import extract file consists of only Documaker Workstation export files; *do not* include the LoadRcpTbl and RunSetRcpTbl or GetCo and GetLOB rules in your AFGJOB.JDT file.
- If the import extract file is comprised of normal transaction data records plus one or more embedded Documaker Workstation export files; you *must* include the LoadRcpTbl and RunSetRcpTbl or GetCo and GetLOB rules in your AFGJOB.JDT file.

Example

Here are some examples which show how this rule works:

Extract file made up of transactions with embedded export files

In this example, the extract file is made up of normal transactions with embedded export files. This example imports information from an extract file named IMPORT.DAT which is comprised of typical transactions and embedded export files. Using this information, the system creates GenData files which are input to GenPrint. The GenPrint program then creates the print output files. Keep in mind:

- The ReplaceNoOpFunc rule is not required in the AFGJOB.JDT file because no fields in the sample DDT file use the NoOpFunc rule
- The options in the Trn_Fields control group are based on items in the transaction data
- The LoadRcpTbl and RunSetRcpTbl rules are required to load and run the recipient table

Here is a sample extract file:

```
SCO1234567HEADERREC00000030198    SCOM1FP WAT1I1B119990223      804-
345-87…
SCO1234567FRMLSTREC0000010110    SCO  FP  T1            89999987
041598…                SCO1234567PRODNMREC00000David Miller
000666666600000444...
SCO1234567PRODADREC00000100 Main Street, Suite 1200          Miami,
FL 30202…
…
;SAMPCO;LB1;EXPORT FILE # 1;NB;P  ;associated w Transaction # 1;
\NA=\;SAMPCO;LB1;LETTER2;
\NA=q1snam\
\NA=q1fl2a\
DATE\October 12, 2000
LESSEE_NAME\Morris Sander
LESSEE_addr\3200 Windy Hill Road
LESSEE_city\Atlanta, GA  30339
\NA=q1b302\
\NA=q1ba36\
\NA=q1ba32\
\NA=q1sal1\
SCO1234567COVERGREC00000SPC 25000  250  Coverage Item 2…
SCO1234567GENRALREC000001    1    3    1    0
Liability1Liability2Liability3Libility4 …
…
;SAMPCO;LB1;EXPORT FILE # 2;NB;P  ;associated w Transaction # 1;
\NA=\;SAMPCO;LB1;CHARTS;
\NA=q1cht\
…
SCO999567HEADERREC00000030198    SCOM1FP WAT1I1B119990223      804-
345-87…
SCO9994567FRMLSTREC0000010110    SCO  FP  T1            89999987
041598…

…
```

Here is a sample DDT file:

```
<Image Field Rules Override>
;0;0;Date;0;0;Date;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_Name;0;9;Lessee_Name;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_addr;0;25;Lessee_addr;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_city;0;25;Lessee_city;0;25;; MapFromImportData;;N;N;N;…
```

Here is a sample AFGJOB.JDT file:

```
<Base Rules>
…
/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportExtract;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
…
```

Here is a sample INI file:

```
< Data >
    ExtrFile           = Import.dat
< ExtractKeyField >
    SearchMask         = 1,SCO
< Trn_Fields >
    Key1               = 1,3,N
    Key2               = 40,2,N
    KeyID              = 4,7,N
< Trn_File >
    BinaryExt          = N
    MaxExtRecLen       = 120
```

Extract file made up of appended export files

In this example, the extract file is made up of one or more appended export files. This example imports information from an extract file named IMPORT.DAT which is comprised of one or more appended export files. Using that information, the system then creates GenData files which are input to GenPrint. The GenPrint program then creates the print output files. Keep in mind:

- The ReplaceNoOpFunc rule is required in the AFGJOB.JDT file because one of the fields in the sample DDT uses the NoOpFunc rule
- The options in the Trn_Fields control group are based on items in the export data
- The LoadRcpTbl and RunSetRcpTbl rules not required

Here is a sample extract file:

```
;CWNG;CIS;1;NB;Export File # 1  ;;
\NA=\;CWNG;CIS;CWFBILL;
\NA=QAIBANCD\
BANNER CODE\001
BANNER CODE TXT\CWNG Company A
\NA=QAIGRAPH\
ACTUAL GJ 1\Nov
\NA=\;CWNG;CIS;CWFCRD3;
\NA=CWFCRD3\
…
;CWNG;CIS;1;NB;Export File # 2  ;;
\NA=\;CWNG;CIS;CWFBILL;
\NA=QAIBANCD\
BANNER CODE\002
BANNER CODE TXT\CWNG Company B
\NA=QAIGRAPH\
ACTUAL GJ 1\Nov
\NA=\;CWNG;CIS;CWFCRD3;
\NA=CWFCRD3\
 …
```

Here is a sample DDT file:

```
<Image Field Rules Override>
;0;0;Banner Code;75;3;Banner
Code;0;3;;MapFromImportData;31,ACCTNUM;…
;0;0;Banner Code Txt;0;11;Banner Code Txt;0;11;;noopfunc;31,Banner
Code;…
```

Here is a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportExtract;;;
…
```

Here is a sample INI file:

```
< Data >
    ExtrFile           = Import.dat
< ExtractKeyField >
    SearchMask         = 2,CWNG
< Trn_Fields >
    Key1               = 2,4,N
    Key2               = 7,3,N
    KeyID              = 16,20,N
< Trn_File >
    BinaryExt          = N
    MaxExtRecLen       = 120
```

See also

MapFromImportData on page 376

NoOpFunc on page 415

ReplaceNoOpFunc on page 197

ImportFile on page 116

ImportNAPOLExtract on page 121

JDT Rules Reference on page 30

ImportFile

Use this form set level rule (level 2) to import transactions (via a standard export file) from Documaker Workstation into GenData. This rule outputs the transaction to an NAFILE.DAT file which can then be used by the GenData program.

NOTE: You can import multiple export files if you use the SCH option.

Syntax

```
ImportFile;;option;
```

There are several ways to specify the import file in the option parameter:

Option	Description
FILE = file name	Enter the name and path of the import file.
INI = INI control group, option	Enter the INI control group and option in which the import file is defined.
SCH = offsetofmask, <searchmask> offsetofdata, lengthofdata	This indicates the file name is contained in a record of the extract file. The offsetofmask is the offset of the search mask, offsetofdata is the offset where the file name starts, and lengthofdata is the length of the file name.
GVM = GlobalVariableName	GlobalVariableName defines the GVM that contains the file name and path information.

The INI and FILE options normally import the same file for each transaction. The SCH and GVM options let you import a different file for each transaction.

Keep in mind:

- For each field that comprises a section (whose data comes from the combined standard export file), you must create or have a DDT record with the field rule set to one of these field rules:
  - MapFromImportData
  - NoOpFunc

  If you use the NoOpFunc rule for any field rule associated with the standard export file, you must insert the ReplaceNoOpFunc rule in the <Base Rules> section of the AFGJOB.JDT file.
- Place the ImportFile rule in the <Base Form Set Rules> section of the AFGJOB.JDT file *after* the BuildFormList rule or any custom rule that creates a form set.
- Do not include the LoadRcpTbl, RunSetRcpTbl, GetCo, or GetLOB rules in your AFGJOB.JDT file.

Assume you have the following items defined in your master resource library. Keep in mind:

- The ReplaceNoOpFunc rule is required in the AFGJOB.JDT file because one of the fields in the sample DDT uses the NoOpFunc rule.
- Trn_Fields control group options are based on items in the first record of the combined WIP/NA/POL Export data file.
- The LoadRcpTbl and RunSetRcpTbl or GetCo and GetLOB rules are not required if this information was assigned from the imported file. It may, however, be necessary to use other rules, such as the Field2GVM rule, to move data from the imported form set fields to relevant GVM variables.

Here is a sample import file named IMPORT.DAT file:

```
;SAMPCO;LB1;EXPORT FILE # 1;NB;P  ;;
\NA=\;SAMPCO;LB1;LETTER2;
\NA=q1snam\
\NA=q1fl2a\
DATE\October 12, 2000
LESSEE_NAME\Morris Sander
LESSEE_addr\3200 Windy Hill Road
LESSEE_city\Atlanta, GA  30339
\NA=q1b302\
\NA=q1ba36\
\NA=q1ba32\
\NA=q1sal1\
```

Here is a sample DDT file:

```
;0;0;Date;0;0;Date;0;25;; NoOpFunc;;N;N;N;…
;0;0;Lessee_Name;0;9;Lessee_Name;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_addr;0;25;Lessee_addr;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_city;0;25;Lessee_city;0;25;; MapFromImportData;;N;N;N;…
```

Here are sample INI settings:

```
< TRN_Fields >
    Key1       = 2,6,N
    Key2       = 32,2,N
    KeyID      = 13,18,N
< TRN_File >
    BinaryExt  = N
    MaxExtRecLen= 120
< ExtractKeyField >
    SearchMask = 2,SAMPCO
< Data >
    ExtrFile   = xxxxx (see the import file example above)
```

Example

The following examples illustrate the different ways you can define the import file when you use this rule.

Using the File option

This example imports information from a file named IMPORT.DAT in the \import directory and uses that information to create the GenData files which the GenPrint program uses to create the print output files.

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportFile;;File=.\Import\Import.dat;
…
```

Using the INI option

This example imports information based on the Import_File option in the Import_Data control group. Using this information, the GenData program creates the files the GenPrint program uses to create the print output files.

Here are the sample INI settings:

```
< Import_Data >
    Import_File = .\Import\Import.dat\
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportFile;;INI=Import_Data,Import_File;
…
```

Using the SCH option

This example imports multiple Documaker Workstation export files based on the content of a line in the extract file. Using this information, the GenData program creates the files the GenPrint program uses to create the print output files.

Here is an excerpt from a sample extract file named EXTRFILE.DAT:

```
;CWNG;CIS;1;NB;W ;;          TXT  .\Import\impt1file.dat
;CWNG;CIS;1;NB;W ;;          TXT  .\Import\impt2file.dat
…
```

For this example, use these INI options:

```
< Data >
    ExtrFile= extrfile.dat
< TRN_Fields >
    Key1    = 2,4,N
    Key2    = 7,3,N
    KeyID   = 35,22,N
< ExtractKeyField >
    SearchMask= 2,CWNG
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportFile;2;SCH=30,TXT 35,21;
…
```

Using the GVM option

This example imports data from an import file based on a GVM variable called *Import_File*. Using this information, the GenData program creates the files the GenPrint program uses to create the print output files. Any valid GVM variable can be used no matter how it is created or assigned.

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
....
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportFile;;GVM=Import_File;
…
```

See also

MapFromImportData on page 376

ReplaceNoOpFunc on page 197

RULNestedOverFlowProc on page 203

ImportNAPOLExtract on page 121

ImportNAPOLFile on page 126

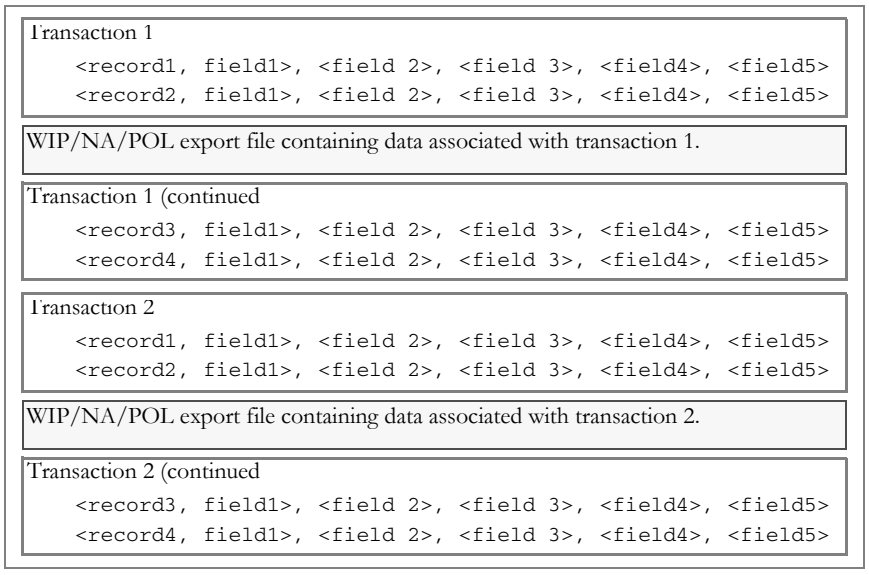
ImportExtract on page 111

JDT Rules Reference on page 30

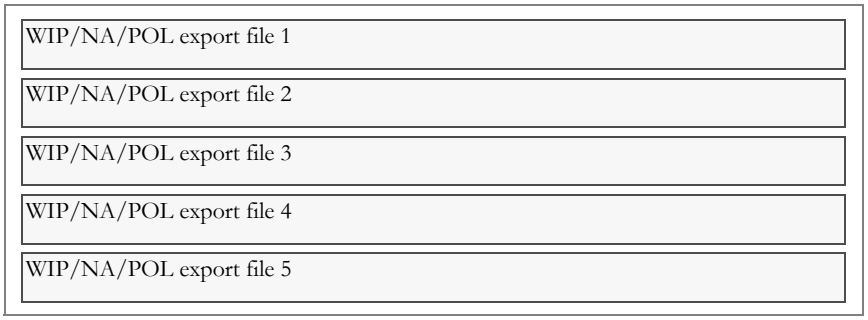
ImportNAPOLExtract

Use this form set level rule (level 2) to import an extract file into the GenData program that is made up of:

- Typical transactions with one or more combined WIP/NA/POL export files from Documaker Workstation embedded into each transaction This illustration shows transactions with embedded WIP/NA/POL export files:



One or more appended WIP/NA/POL export data files from Documaker Workstation. This illustration shows an extract file comprised of WIP/NA/POL files appended to one another:



Syntax

```
;ImportNAPOLExtract;;;
```

Although there are no parameters for this rule. Keep in mind:

- Specify the extract file name, which contains the import information for each transaction, in the ExtrFile option of the Data control group. This is the normal way to define the name of the extract file.
- Only use the SearchMask option in the ExtractKeyField control group; do not use the Key option.

- For each field that comprises a section (whose data comes from the combined WIP/NA/POL export data), you must create or have a DDT record whose field rule is set to one of the following field rules
  - MapFromImportData
  - NoOpFunc

  If you use the NoOpFunc rule for any field rule associated with the WIP/NA/POL export data, you must insert the ReplaceNoOpFunc rule in the <Base Rules> section of the AFGJOB.JDT file.
- You must place the ImportNAPOLExtract rule in the <Base Form Set Rules> section of the AFGJOB.JDT file after the BuildFormList rule or any custom rule that creates a form set.
- If the import extract file is comprised of normal transaction data records plus one or more embedded Documaker Workstation combined WIP/NA/POL export files; you can include the LoadRcpTbl and RunSetRcpTbl or GetCo and GetLOB rules in your AFGJOB.JDT file. You may need to use other rules, like Field2GVM, to move data from the imported form set fields to relevant GVM variables.
- To process without using DDT files, substitute the ForceNoImages rule for the RULStandardImageProc rule.

Example

Here are some examples which show how this rule works:

Extract file made up of transactions with embedded WIP/NA/POL files

In this example, the extract file is made up of normal transactions with embedded WIP/NA/POL export files. This example imports information from an extract file named IMPORT.DAT which is comprised of transactions with embedded WIP/NA/POL export files. Using this information, the system creates standard GenData files which are input to the GenPrint program. The GenPrint program then creates the print output files. Keep in mind:

- The ReplaceNoOpFunc rule is not required in the AFGJOB.JDT file because no fields in the sample DDT file use the NoOpFunc rule
- The options in the Trn_Fields control group are based on items in the first record of the typical transaction data
- The LoadRcpTbl and RunSetRcpTbl rules are required to load and run the recipient table

Here is a sample extract file:

```
SCO1234567HEADERREC00000030198    SCOM1FP WAT1I1B119990223      804-
345-87…
SCO1234567FRMLSTREC0000010110    SCO  FP   T1             89999987
041598…                   SCO1234567PRODNMREC00000David Miller
000666666600000444...
SCO1234567PRODADREC00000100 Main Street, Suite 1200           Miami,
FL 30202…
…
WIP="SAMPCO     ""LB1   ""NAPOL FILE # 1      ""NB ""P" associated w
Transaction # 1"
;SAMPCO;LB1;LETTER2;Second Letter;RD;;q1snam|D3<Insured>/
q1fl2a|D3S<Insured>/q1b302|D3S<Insured>/q1ba36…;
\ENDDOCSET\
\NA=q1snam,LN=1,DUP=OFF,SIZE=C,TRAY=U,X=0,Y=0,PA=1,OPT=D3\
```

```
\ENDIMAGE\
\NA=q1fl2a,LN=1,DUP=OFF,SIZE=C,TRAY=U,X=0,Y=3360,PA=1,OPT=D3S\
FDate;235;3913;12012;;;\January 1, 2000
FLessee_NAME;235;4806;12012;G;;\Morris Sander
FLessee_addr;235;5212;12012;G;;\3200 Windy Hill Road
FLessee_city;235;5636;12012;G;;\Atlanta, GA  30339
\ENDIMAGE\
…
\ENDFORM\
\ENDDOCSET\
SCO1234567COVERGREC00000SPC 25000  250  Coverage Item 2…
SCO1234567GENRALREC000001    1    3    1    0
Liability1Liability2Liability3Libility4 …
…
WIP="SAMPCO     ""LB1  ""NAPOL FILE # 2       ""NB ""P" associated w
Transaction # 1"
;SAMPCO;LB1;CHARTS;Form q1cht;RD;;q1cht|D5<Insured>;
\ENDDOCSET\
\NA=q1cht,LN=1,DUP=OFF,SIZE=L,TRAY=U,X=0,Y=0,PA=1,OPT=D5\
FFORMSET PAGE NUM;17656;25740;16008;PF;;\
FFORMSET PAGE NUM OF;18408;25740;16008;PF;;\
\ENDFORM\
\ENDDOCSET\
…
SCO999567HEADERREC00000030198    SCOM1FP WAT1I1B119990223      804-
345-87…
SCO9994567FRMLSTREC0000010110   SCO  FP  T1            89999987
041598…
…
```

Here is a sample DDT file:

```
<Image Field Rules Override>
;0;0;Date;0;0;Date;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_Name;0;9;Lessee_Name;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_addr;0;25;Lessee_addr;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_city;0;25;Lessee_city;0;25;; MapFromImportData;;N;N;N;…
```

Here is a sample AFGJOB.JDT file:

```
<Base Rules>
…
/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLExtract;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
…
```

Here is a sample INI file:

```
< Data >
    ExtrFile   = Import.dat
< ExtractKeyField >
    SearchMask = 1,SCO
< Trn_Fields >
```

```
    Key1        = 1,3,N
    Key2        = 40,2,N
    KeyID       = 4,7,N
< Trn_File >
    BinaryExt   = N
    MaxExtRecLen= 120
```

Extract file made up of appended WIP/NA/POL files

In this example, the extract file is made up of one or more appended WIP/NA/POL export files. This example imports information from an extract file named IMPORT.DAT which is comprised of one or more appended WIP/NA/POL export data files. Using that information, the system then creates GenData files which are input to GenPrint. The GenPrint program then creates the print output files. Keep in mind:

- The ReplaceNoOpFunc rule is required in the AFGJOB.JDT file because one of the fields in the sample DDT file uses the NoOpFunc rule
- The options in the Trn_Fields control group are based on items in the first record of the WIP/NA/POL export file
- The LoadRcpTbl and RunSetRcpTbl rules not required

Here is a sample extract file:

```
WIP="SAMPCO     ""LB1  ""NAPOL FILE # 1      ""NB ""P"
;SAMPCO;LB1;LETTER2;Second Letter;RD;;q1snam|D3<Insured>/
q1fl2a|D3S<Insured>/q1b302|D3S<Insured>/q1ba36…;
\ENDDOCSET\
\NA=q1snam,LN=1,DUP=OFF,SIZE=C,TRAY=U,X=0,Y=0,PA=1,OPT=D3\
\ENDIMAGE\
\NA=q1fl2a,LN=1,DUP=OFF,SIZE=C,TRAY=U,X=0,Y=3360,PA=1,OPT=D3S\
FDate;235;3913;12012;;;\January 1, 2000
FLessee_NAME;235;4806;12012;G;;\Morris Sander
FLessee_addr;235;5212;12012;G;;\3200 Windy Hill Road
FLessee_city;235;5636;12012;G;;\Atlanta, GA  30339
\ENDIMAGE\
…
\ENDFORM\
\ENDDOCSET\
WIP="SAMPCO     ""LB1  ""NAPOL FILE # 2       ""NB ""P"
;SAMPCO;LB1;CHARTS;Form q1cht;RD;;q1cht|D5<Insured>;
\ENDDOCSET\
\NA=q1cht,LN=1,DUP=OFF,SIZE=L,TRAY=U,X=0,Y=0,PA=1,OPT=D5\
FFORMSET PAGE NUM;17656;25740;16008;PF;;\
FFORMSET PAGE NUM OF;18408;25740;16008;PF;;\
\ENDFORM\
\ENDDOCSET\
```

Here is a sample DDT file:

```
<Image Field Rules Override>
;0;0;Date;0;0;Date;0;25;; NoOpFunce;;N;N;N;…
;0;0;Lessee_Name;0;9;Lessee_Name;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_addr;0;25;Lessee_addr;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_city;0;25;Lessee_city;0;25;; MapFromImportData;;N;N;N;…
```

Here is a sample AFGJOB.JDT file:

```
<Base Rules>
…
```

```
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLExtract;;;
…
```

Here is a sample INI file:

```
< Data >
    ExtrFile    = Import.dat
< ExtractKeyField >
    SearchMask = 1,WIP
< Trn_Fields >
    Key1        = 1,3,N
    Key2        = 6,6,N
    KeyID       = 26,20,N
< Trn_File >
    BinaryExt   = N
    MaxExtRecLen= 120
```

See also

MapFromImportData on page 376

NoOpFunc on page 415

ReplaceNoOpFunc on page 197

ImportFile on page 116

ForceNoImages on page 98

RULNestedOverFlowProc on page 203

JDT Rules Reference on page 30

ImportNAPOLFile

Use this form set level rule (level 2) to import a single transaction, stored in a WIP/NA/POL export file, from Documaker Workstation into the GenData program.

NOTE: If you use the SCH option, you can import multiple Documaker Workstation WIP/NA/POL export files.

Syntax

```
;ImportNAPOLFile;;option;
```

There are several ways to specify the import file in the option parameter:

Option	Description
FILE = file name	Enter the name and path of the import file.
INI = INI control group, option	Enter the INI control group and option in which the import file is defined.
SCH = offsetofmask, <searchmask> offsetofdata, lengthofdata	This indicates the file name is contained in a record of the extract file. The offsetofmask is the offset of the search mask, offsetofdata is the offset where the file name starts, and lengthofdata is the length of the file name.
GVM = GlobalVariableName	GlobalVariableName defines the GVM that contains the file name and path information.

The INI and FILE options normally import the same file for each transaction. The SCH and GVM options let you import a different file for each transaction.

Keep in mind:

- For each field that comprises a section (whose data comes from the combined WIP/NA/POL export data), you must create or have a DDT record with the field rule set to one of these field rules:
  - MapFromImportData
  - NoOpFunc

  If you use the NoOpFunc rule for any field rule associated with the combined WIP/NA/POL export data, you must insert the ReplaceNoOpFunc rule in the <Base Rules> section of the AFGJOB.JDT file.
- Place the ImportNAPOLFile rule in the <Base Form Set Rules> section of the AFGJOB.JDT *after* the BuildFormList rule or any custom rule that creates a form set.
- The LoadRcpTbl and RunSetRcpTbl or GetCo and GetLOB rules are not required unless you use the SCH option.

Assume you have the following items defined in your master resource library. Keep in mind:

- The ReplaceNoOpFunc rule is required in the AFGJOB.JDT file because one of the fields in the sample DDT uses the NoOpFunc rule.

- Trn_Fields control group options are based on items in the first record of the combined WIP/NA/POL Export data file.

Here is a sample combined WIP/NA/POL import file named IMPORT.DAT:

```
WIP="SAMPCO     ""LB1  ""NAPOL FILE # 1      ""NB ""P"
;SAMPCO;LB1;LETTER2;Second Letter;RD;;q1snam|D3<Insured>/
q1fl2a|D3S<Insured>/q1b302|D3S<Insured>/q1ba36…;
\ENDDOCSET\
\NA=q1snam,LN=1,DUP=OFF,SIZE=C,TRAY=U,X=0,Y=0,PA=1,OPT=D3\
\ENDIMAGE\
\NA=q1fl2a,LN=1,DUP=OFF,SIZE=C,TRAY=U,X=0,Y=3360,PA=1,OPT=D3S\
FDate;235;3913;12012;;;\January 1, 2000
FLessee_NAME;235;4806;12012;G;;\Morris Sander
FLessee_addr;235;5212;12012;G;;\3200 Windy Hill Road
FLessee_city;235;5636;12012;G;;\Atlanta, GA  30339
\ENDIMAGE\
…
\ENDFORM\
\ENDDOCSET\
```

Here is a sample DDT file:

```
<Image Field Rules Override>
;0;0;Date;0;0;Date;0;25;; NoOpFunc;;N;N;N;…
;0;0;Lessee_Name;0;9;Lessee_Name;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_addr;0;25;Lessee_addr;0;25;; MapFromImportData;;N;N;N;…
;0;0;Lessee_city;0;25;Lessee_city;0;25;; MapFromImportData;;N;N;N;…
```

Here are sample INI settings:

```
< TRN_Fields >
    Key1        = 1,3,N
    Key2        = 6,6,N
    KeyID       = 26,20,N
< TRN_File >
    BinaryExt   = N
    MaxExtRecLen= 120
<ExtractKeyField>
    SearchMask  = 1,WIP=
< Data >
     ExtrFile   = xxxxx
```

Example

The following examples illustrate the different ways you can define the import file when you use this rule.

Using the File option

This example imports information from the IMPORT1.DAT file in the \import directory and uses that information to create the GenData files which the GenPrint program uses to create the print output files.

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLFile;;File=.\Import\Import1.dat;
…
```

Using the INI option

This example imports information based on the Import_File option in the Import_Data control group. Using this information, the GenData program creates the files the GenPrint program uses to create the print output files.

Here are the sample INI settings:

```
< Import_Data >
    Import_File = .\Import\Import1.dat\
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLFile;;INI=Import_Data,Import_File;
…
```

Using the SCH option

This example imports multiple WIP/NA/POL export files based on the content of a line in the extract file. Using this information, the GenData program creates the files the GenPrint program uses to create the print output files.

Here is an excerpt from a sample extract file named EXTRFILE.DAT:

```
;CWNG;CIS;1;NB;W ;;          TXT  .\Import\impt1file.dat
;CWNG;CIS;1;NB;W ;;          TXT  .\Import\impt2file.dat
…
```

For this example, use these INI options:

```
< Data >
    ExtrFile= extrfile.dat
< TRN_Fields >
    Key1    = 2,4,N
    Key2    = 7,3,N
    KeyID   = 35,22,N
< ExtractKeyField >
    SearchMask = 2,CWNG
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLFile;2;SCH=30,TXT 35,22;
```

Using the GVM option

This example imports data from an import file based on a GVM variable called *Import_File*. Using this information, the GenData program creates the files the GenPrint program uses to create the print output files. Any valid GVM variable can be used no matter how it is created or assigned. Here is an excerpt from a sample AFGJOB.JDT file:

```
<Base Rules>
…
;ReplaceNoOpFunc;;;

/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLFile;;GVM=Import_File;
```

See also

ImportFile on page 116

ImportExtract on page 111

ImportNAPOLExtract on page 121

ImportNAPOLFile on page 126

JDT Rules Reference on page 30

ImportXMLExtract

Use this form set rule (level 2) to import a file which consists of one or more XML transactions into the GenData program for processing. Using this file, the GenData program creates the recipient batch, NAFile, POLFile, and NewTrn files that you can print, archive, or both using the GenPrint and GenArc programs.

NOTE: If you are running Documaker from IDS, use the ImportXMLExtract rule to bring in XML in standard Documaker XML format, such as from Documaker Workstation or iDocumaker. Use the UseXMLExtract rule to convert a loaded extract file into an XML tree, which you can then use to query data.

You append multiple export files to create the import XML file. The export files are created using the Documaker Workstation XML Export option. This illustration shows an example file comprised of export files appended to one another:

Transaction 1

```
<?xml version="1.0"?>
<Document Type="Oracle Universal" Version="5.0">
<DocSet>
<ArcEffectiveDate></ArcEffectiveDate>
<Library Name="Oracle Insurance"></Library>
<Key1 Name="Company">SkyInsur</Key1>
<KeyY2 Name="Lob">Package Policy</Key2>
<TransactionID Name="PolicyNum">1010j</TransactionID>
…
…
```

Transaction 2

```
<?xml version="1.0"?>
<Document Type="Oracle Universal" Version="5.0">
<DocSet>
<ArcEffectiveDate></ArcEffectiveDate>
<Library Name="Oracle Insurance"></Library>
<Key1 Name="Company">SkyInsur</Key1>
<KeyY2 Name="Lob">Package Policy</Key2>
<TransactionID Name="PolicyNum">1110j</TransactionID>
…
…
```

Transaction 3

```
<?xml version="1.0"?>
<Document Type="Oracle Universal" Version="5.0">
<DocSet>
<ArcEffectiveDate></ArcEffectiveDate>
<Library Name="Oracle Insurance"></Library>
<Key1 Name="Company">SkyInsur</Key1>
<KeyY2 Name="Lob">Package Policy</Key2>
<TransactionID Name="PolicyNum">1210j</TransactionID>
…
…
```

Syntax

```
ImportXMLExtract;;option;
```

Option	Description
SP	Include the SP option to suppress the pagination portion of the import. This lets you run rules and other form set manipulations before calling a rule to paginate the form set, such as the PaginateAndPropogate rule.

NOTE: You can only use this rule for single-step processing.

Keep in mind...

- Create a simplified AFGJOB.JDT file when you use this rule and omit these rules:
  - LoadRcpTbl
  - LoadExtractData
  - RunSetRcpTbl
  - CreateGlbVar
  - LoadDDTDefs
  - InitOvFlw
  - SetOvFlwSym
  - ResetOvFlw
- Use the NoGenTrnTransactionProc rule because the XML file has no transaction information on the first line.
- Place the ImportXMLExtract rule in the <Base Form Set Rules> section of the AFGJOB.JDT file after the BuildFormList rule or any rule that creates a form set.
- In the TRN_File control group, set MaxExtRecLen option to the length of the longest record in the import file.
- In the TRN_Fields control group, include only the Key1, Key2, and KeyID options. Set these options to dummy data, because the GVM variables are set to the data values in the XMLTags2GVM control group during processing.
- If you load an XML or a V2 import file as the extract file, it must conform to the extract file rules. This means that you must set the MaxExtRecLen and BinaryExt INI options appropriately.
- Define the XMLTags2GVM control group in your FSISYS.INI file as shown here:

```
< XMLTags2GVM >
    GVM = XMLTag, (Req/Opt)
```

  Where GVM is the name of the GVM variable and XMLTag is the tag name in the XML file. Include *Req* or *Opt* to specify whether it is required or optional. If it is required and is omitted from the XML file, processing stops. Here is an example:

```
< XMLTags2GVM >
    Key1    = Key1, Req
    Key2    = Key2, Req
    KeyID   = TransactionID, Opt
```

Example

Assume you have the following items defined in your master resource library. See XML File Format on page 139 for an example of an import file in the standard XML file format.

Here is an example of the INI options you need in your FSISYS.INI file:

```
< Data >
    AFGJOBFile = .\deflib\afgjob.jdt
    ExtrFile   = .\extract\extrfile.xml
< ExtractKeyField >
    SearchMask = 1,<?xml
```

```
< Key1Table >
    XML         = XML
< Key2Table >
    XML         = XML
< KeyIDTable >
    XML         = XML
< Trigger2Archive >
    Key1        = Key1
    Key2        = Key2
    KeyID       = KeyID
    RunDate     = RunDate
< TRN_Fields >
    Key1        = 3,3,N
    Key2        = 3,3,N
    KeyID       = 3,3,N
< TRN_File >
    BinaryExt   = N
    MaxExtRecLen= 175
< XMLTags2GVM >
    Key1        = Key1,Req
    Key2        = Key2,Req
    KeyID       = TransactionID,Opt
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
< Base Rules >
;RULStandardJobProc;;;
…
< Base Form Set Rules >
;NoGenTrnTransactionProc;;;
;BuildFormList;;;
;ImportXMLExtract;;;
…
…
```

See also

ImportXMLFile on page 134

PaginateAndPropagate on page 174

Processing Import Files on page 22

JDT Rules Reference on page 30

ImportXMLFile

Use this form set rule (level 2) to import an XML file into GenData for processing. Using this file, the GenData program creates recipient batch, NAFile, POLFile, and NewTrn files that you can print, archive, or both using the GenPrint and GenArc programs.

The export file to be used as import was created using the Documaker Workstation XML Export function that produces a file in the Documaker Standard XML format.

Syntax

```
;ImportXMLFile;;option;
```

Option	Description
SP	Include the SP option to suppress the pagination portion of the import. This lets you run rules and other form set manipulations before calling a rule to paginate the form set, such as the PaginateAndPropogate rule. You must place this option before the FILE option. Here is an example: `;ImportXMLFile;;SP,SCH=11,FILENAME 20,20;` This example suppresses the pagination of the import of the file designated by this search mask.
TF	Enter TF to truncate fields to the field length defined by the FAP file. Make sure you specify this parameter before FILE option.
FILE	Enter the name and path of the import file.
INI	Enter the INI control group and option in which the import file is defined. Separate the control group and option with a comma.
SCH	Enter the search criteria and the file name data, separated by a space. The name of the file, including its path, that you want to import should be contained in the record in the file indicated by the ExtrFile option in the Data control group. The search criteria are one or more comma delimited data pairs, offsets and character string, used as the search mask for finding the record in the specified file. The file name data is a comma delimited data pair that defines the offset and length of the file name in the record defined by the search criteria parameter.
GVM	Enter the global variable name (GVM) that contains the file name and path information.

Keep in mind:

- Create a simplified AFGJOB.JDT file when you use this rule. For instance, omit these rules:
  - LoadRcpTbl
  - LoadExtractData
  - RunSetRcpTbl
  - CreateGlbVar
  - LoadDDTDefs
  - InitOvFlw

  - SetOvFlwSym
  - ResetOvFlw
- Use the NoGenTrnTransactionProc rule because the XML file has no transaction information on the first line.
- Place the ImportXMLExtract rule in the <Base Form Set Rules> section of the AFGJOB.JDT file after the BuildFormList rule or any custom rule that creates a form set.
- In the TRN_File control group, set MaxExtRecLen option to the length of the longest record in the import file.
- In the TRN_Fields control group, include only the Key1, Key2, and KeyID options. Set these options to dummy data, because the GVM variables are set to the data values in the XMLTags2GVM control group during processing.
- If you load an XML or a V2 import file as the extract file, it must conform to the extract file rules. This means that you must set the MaxExtRecLen and BinaryExt INI options appropriately.
- Define the XMLTags2GVM control group in your FSISYS.INI file as shown here:

```
< XMLTags2GVM >
    GVM = XMLTag, (Req/Opt)
```

  Where GVM is the name of the GVM variable and XMLTag is the tag name in the XML file. Include *Req* or *Opt* to specify whether it is required or optional. If it is required and is not present in the XML file, processing will terminate. Here is an example:

```
< XMLTags2GVM >
    Key1    = Key1, Req
    Key2    = Key2, Req
    KeyID   = TransactionID, Opt
```

Example

These examples show the different ways you can define the import file when you use this rule. Assume you have the following items defined in your master resource library. For an example of the standard XML file format, see XML File Format on page 139.

Here are sample INI settings in your FSISYS.INI file:

```
< Data >
   AFGJOBFile      = .\deflib\afgjob.jdt
   ExtrFile        = .\extract\dummy.dat
< ExtractKeyField >
   SearchMask      = 1,XML_FILE_NAME
< Key1Table >
   XML             = xml
< Key2Table >
   XML             = xml
< KeyIDTable >
   XML             = xml
< Trigger2Archive >
   Key1            = Key1
   Key2            = Key2
   KeyID           = KeyID
   RunDate         = RunDate
```

```
< TRN_Fields >
   Key1            =  1,3,N
   Key2            =  5,5,N
   KeyID           = 10,4,N
< TRN_File >
   BinaryExt       = N
   MaxExtRecLen    = 175
< XMLTags2GVM >
    Key1           = Key1,Req
    Key2           = Key2,Req
    KeyID          = TransactionID,Opt
```

Here is a sample of the DUMMY.DAT file, pointed to by the ExtrFile option in the Data control group in your FSISYS.INI file.

```
0   1
1   5
XML_FILE_NAME This is a dummy extract file.
```

Using the TF Option

Use the TF (Truncate Field) option to truncate fields to their FAP defined field length. Make sure you specify this parameter before FILE option. Here are some examples:

This example will truncate the fields lengths:

```
;ImportXMLFile;;TF,SCH=1,XML_FILE_NAME 15,55;
```

These examples truncate fields and suppress pagination:

```
;ImportXMLFile;;SP,TF,SCH=1,XML_FILE_NAME 15,55;
```

```
;ImportXMLFile;;TF,SP,SCH=1,XML_FILE_NAME 15,55;
```

This example does not truncate fields or suppress pagination:

```
;ImportXMLFile;;SP
```

NOTE: No formatting is allowed on the multi-line text field when you include the TF option.

Using the File Option

This example imports the F_FILE.XML file from the \export directory. Using this file, the GenData program creates the recipient batch, NA, POL, and NewTrn files needed for GenPrint and GenArc processing.

Here is an excerpt from a sample AFGJOB.JDT file using the File option:

```
< Base Rules >
;RULStandardJobProc;;;
…
< Base Form Set Rules >
;NoGenTrnTransactionProc;;;
;BuildFormList;;;
;ImportXMLFile;;File=.\Export\F_File.xml;
…
```

Using the INI Option

This example imports the F_INI.XML file from the \export directory. Using this file, the GenData program creates the recipient batch, NA, POL, and NewTrn files needed for GenPrint and GenArc processing.

In addition to the INI options defined previously, you must also include the this option:

```
< Import_Data >
    Import_File = .\Export\F_File.xml\
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
< Base Rules >
;RULStandardJobProc;;;
…
< Base Form Set Rules >
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;ImportNAPOLFile;;INI=Import_Data,Import_File;
…
```

Using the SCH Option

This example imports XML files (F_SCH1.XML, F_SCH2.XML, and F_SCH3.XML) based on the content of a line in the file pointed to by the ExtrFile option in the Data control group. Using these files, the GenData program creates the recipient batch, NA, POL, and NewTrn files needed for GenPrint and GenArc processing.

This INI option differs from the one defined in the assumed MRL definition:

```
< Data >
    ExtrFile = .\extract\F_Sch.DAT
```

Here is an excerpt from the F_SCH.DAT file in the \extract directory which contains an entry (path and file name) for each XML file to import:

```
XML_FILE_NAME .\export\F_SCH1.xml
XML_FILE_NAME .\export\F_SCH2.xml
XML_FILE_NAME .\export\F_SCH3.xml
…
```

NOTE: This option lets you import and process multiple XML files because of the way the file name and path are specified—one file per entry in the file specified in the ExtrFile option in the Data control group.

Here is an excerpt from a sample AFGJOB.JDT file:

```
< Base Rules >
;RULStandardJobProc;;;
…
< Base Form Set Rules >
;NoGenTrnTransactionProc;;;
;BuildFormList;;;
;ImportXMLFile;2;SCH=1,XML_FILE_Name 15,19
…
```

Using the GVM Option

This example imports data from a XML file based on file name contained in the GVM variable called Import_File. Using this file, the GenData program creates the recipient batch, NA, POL, and NewTrn files needed for GenPrint and GenArc processing.

Any valid GVM variable can be used no matter how it is created or assigned.

This example creates the GVM variable, ImportXMLFile_GVM, by including this INI option and adding its definition to the TRNDFDFL.DFD file:

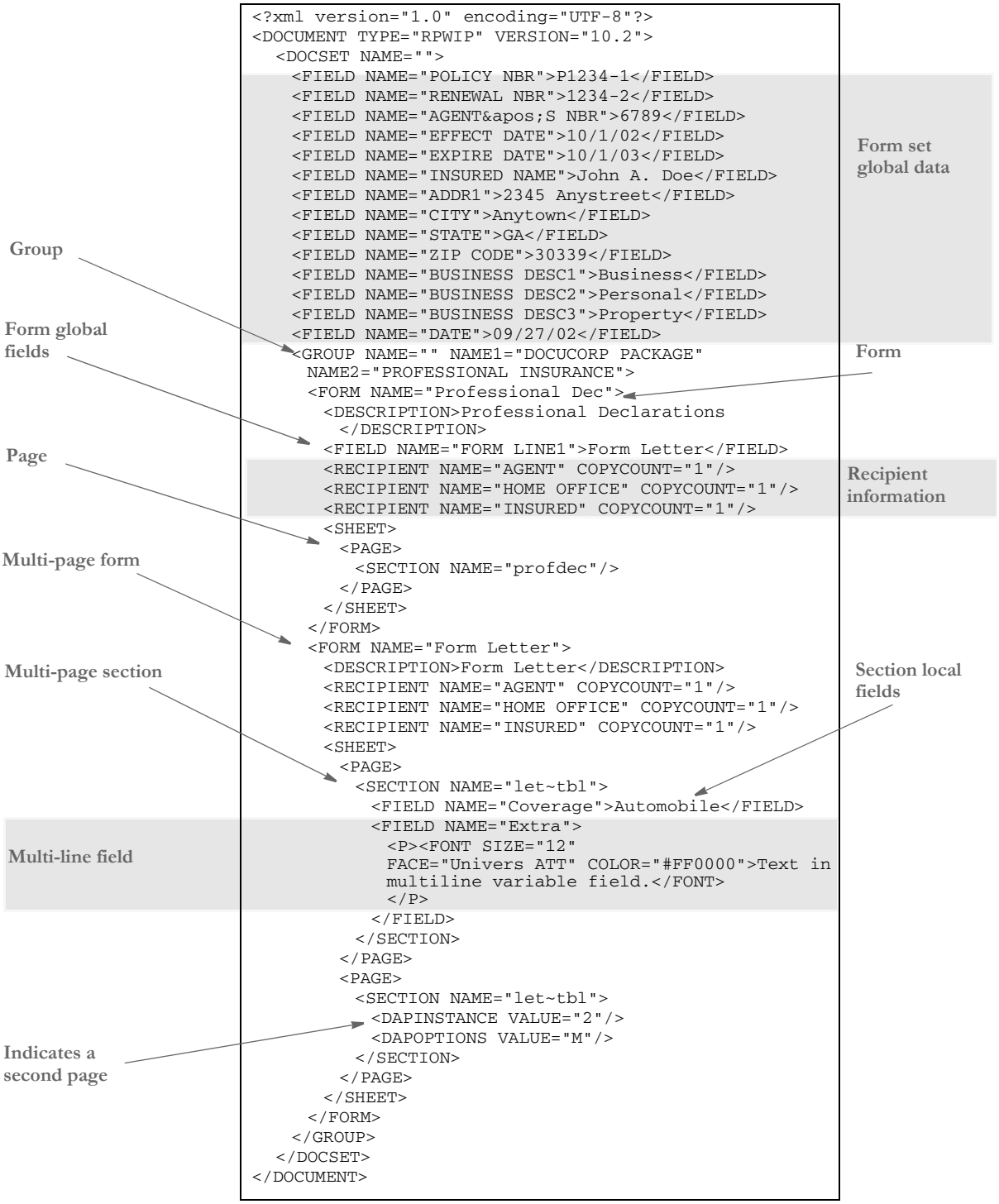
```
< GentrnDummyFields >
    ImportXMLFile_GVM = .\export\F_GVM.xml
```

Here is an excerpt from a sample AFGJOB.JDT file:

```
< Base Rules >
;RULStandardJobProc;;;
…
< Base Form Set Rules >
;NoGenTrnTransactionProc;;;
;BuildFormList;;;
;ImportXMLFile;;GVM=ImportXMLFile_GVM;
```

XML File Format

Here is an example of the format of the XML file the system creates:




```
<?xml version="1.0" encoding="UTF-8"?>
<DOCUMENT TYPE="RPWIP" VERSION="10.2">
  <DOCSET NAME="">
    <FIELD NAME="POLICY NBR">P1234-1</FIELD>
    <FIELD NAME="RENEWAL NBR">1234-2</FIELD>
    <FIELD NAME="AGENT&apos;S NBR">6789</FIELD>
    <FIELD NAME="EFFECT DATE">10/1/02</FIELD>
    <FIELD NAME="EXPIRE DATE">10/1/03</FIELD>
    <FIELD NAME="INSURED NAME">John A. Doe</FIELD>
    <FIELD NAME="ADDR1">2345 Anystreet</FIELD>
    <FIELD NAME="CITY">Anytown</FIELD>
    <FIELD NAME="STATE">GA</FIELD>
    <FIELD NAME="ZIP CODE">30339</FIELD>
    <FIELD NAME="BUSINESS DESC1">Business</FIELD>
    <FIELD NAME="BUSINESS DESC2">Personal</FIELD>
    <FIELD NAME="BUSINESS DESC3">Property</FIELD>
    <FIELD NAME="DATE">09/27/02</FIELD>
    <GROUP NAME="" NAME1="DOCUCORP PACKAGE"
     NAME2="PROFESSIONAL INSURANCE">
     <FORM NAME="Professional Dec">
      <DESCRIPTION>Professional Declarations
       </DESCRIPTION>
      <FIELD NAME="FORM LINE1">Form Letter</FIELD>
      <RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
      <RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
      <RECIPIENT NAME="INSURED" COPYCOUNT="1"/>
      <SHEET>
       <PAGE>
        <SECTION NAME="profdec"/>
       </PAGE>
      </SHEET>
     </FORM>
     <FORM NAME="Form Letter">
      <DESCRIPTION>Form Letter</DESCRIPTION>
      <RECIPIENT NAME="AGENT" COPYCOUNT="1"/>
      <RECIPIENT NAME="HOME OFFICE" COPYCOUNT="1"/>
      <RECIPIENT NAME="INSURED" COPYCOUNT="1"/>
      <SHEET>
       <PAGE>
        <SECTION NAME="let~tbl">
         <FIELD NAME="Coverage">Automobile</FIELD>
         <FIELD NAME="Extra">
          <P><FONT SIZE="12"
          FACE="Univers ATT" COLOR="#FF0000">Text in
          multiline variable field.</FONT>
          </P>
         </FIELD>
        </SECTION>
       </PAGE>
       <PAGE>
        <SECTION NAME="let~tbl">
         <DAPINSTANCE VALUE="2"/>
         <DAPOPTIONS VALUE="M"/>
        </SECTION>
       </PAGE>
      </SHEET>
     </FORM>
    </GROUP>
  </DOCSET>
</DOCUMENT>
```

Keep in mind...

- DAPOPTIONS should have a value of *M* for multi-page sections (FAP files). There are other section options, but only *M* is applicable in XML.

  Use DAPINSTANCE to provide a page number for multi-page sections. If the section does not span multiple pages, omit the DAPINSTANCE value.

- When you have multiple XML transactions within a single file, separate each transaction with a line feed. This is a requirement of Documaker software, not the XML parser.

- Although you do not have to include line feeds inside the XML for a transaction, we suggest you add a line feed after each element tag. This makes it easier to read the file and helps in debugging your XML. A message like

  ```
  Line 255, column 8, syntax is incorrect
  ```

  is easier to diagnose than

  ```
  Line 1, column 156780, syntax is incorrect.
  ```

See also ImportXMLExtract on page 130

PaginateAndPropagate on page 174

Processing Import Files on page 22

JDT Rules Reference on page 30

InitArchive

Use this job level rule (level 1), along with the Archive rule, to run the GenArc program as part of single-step processing.

The InitArchive rule checks the INI options in the Trigger2Archive control group, initializes the database, opens the APPIDX.DFD and CAR files, and perform other steps to initialize archive.

The Archive rule then unloads the current form set and converts field data for archive using the INI options in the Trigger2Archive control group.

Syntax

```
;InitArchive;2;;
```

Example

Here is an example:

```
< Base Rules >
;InitArchive;1;;
```

See also

Archive on page 43

JDT Rules Reference on page 30

InitConvertWIP

Use this job level (level 1) rule to perform the initialization necessary for the ConvertWIP rule. You use this rule when you want to include the GenWIP process in single-step mode.

Syntax

```
;InitConvertWIP;;;;
```

Example

```
;InitConvertWIP;1;;
```

See also

ConvertWIP on page 75

JDT Rules Reference on page 30

InitMerge

Use this job level (level 1) rule to create a list of printers, batches, and buffers for the RCB comment records. This rule also creates a list to hold AFP records and AFP fonts. After the system finishes running the rule, it deletes everything the rule created.

Syntax

```
;InitMerge;;;
```

NOTE: The recipient batch files are not used at this stage. The batch list must be created beforehand so the system will know which print files belong together. The skipping batch message is an artifact of the batch file loading process.

Example

```
;InitMerge;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

InitOvFlw

Use this job level rule (level 1) to initialize the overflow feature. Overflow symbols are created to keep track of the number of records processed. The overflow symbol is one of the parameters that would be initialized.

Syntax

```
;InitOvFlw;;;
```

When processing an overflow form, the overflow count must be reset back to zero, if not the processing will start with the second record in the extract.

When finished, this rule turns off the system's overflow feature which frees resources used when using the overflow feature and overflow variables.

Example

```
;InitOvFlw;;;
```

See also

WriteOutput on page 248

ResetOvFlw on page 199

SetOvFlwSym on page 227

IncOvSym on page 366

OvActPrint on page 417

OvPrint on page 419

JDT Rules Reference on page 30

InitPageBatchedJob

Use this job level (level 1) rule to open NA and POL files. This rule installs the section level callback function for inserting recipient batch records into the AFP print stream as AFP comment records.

When finished, this rule restores the original callback function and closes the NAFILE.DAT and POLFILE.DAT files.

Syntax

```
;InitPageBatchedJob;;;
```

Example

```
;InitPageBatchedJob;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

InitPrint

Use this job level (level 1) rule to load printer and recipient batch information. This rule sets up PRTLIB data, initializes print options, and loads a table which contains page totals for recipient batch files.

This rule also places a structure containing all of the above information into the GVM variable RULPRT.

Syntax

```
;InitPrint;;;
```

When finished, this rule closes any open print files.

Example

```
;InitPrint;;;
```

See also

PrintFormset on page 182

NoGenTrnTransactionProc on page 168

Rules Used for 2-up Printing on page 27

Rules Used in Single-Step Processing on page 25

JDT Rules Reference on page 30

InitSetRecipCache

Use this job level rule (level 1) to set the cache the system will use to store recipient information in memory. With this rule you can tell the system the amount of memory to set aside and use for storing information in the Key1 and Key2 fields, often used to store the company and line of business.

You can use this rule to improve processing performance for complex forms. This rule has no affect on the processing speed for static forms.

This rule is also used in multi-step processing to enhance performance.

NOTE: If you omit this rule, the system does not set aside memory for the Key1 and Key2 fields. If this rule causes any problems with your implementation, you can remove it from the AFGJOB.JDT file.

Syntax

```
;InitSetRecipCache;;Key1,Key2;
```

Parameter	Description
Key1 Key2	For Key1 and Key2, enter the amount of memory you want to set aside for storing the information contained in those fields. These fields are typically used to store information such as the company name and line of business. You can enter any number from one (1) to 500. The default is five (5). If you enter a zero (0), a negative number, or a number greater than 500, the system ignores your entry and defaults to five.

Example

```
;InitSetRecipCache;;10,15;
```

This example sets the cache for the Key1 field to 10 and sets the cache for the Key2 field to 15.

See also JDT Rules Reference on page 30

InlineImagesAndBitmaps

Use this form set level (level 2) rule if you do not want to use Library Manager to maintain forms and graphics but still need to retrieve the exact data that was printed. This rule lets you inline all FAP files and embed graphics into the NA file.

NOTE: Keep in mind the size of the NA file and archive will grow significantly if you use this rule. Furthermore, the performance of the GenData, GenPrint, and GenArc programs will degrade significantly if you use this rule.

Syntax

```
;InlineImagesAndBitmaps;;;
```

Use this rule only when necessary and when performance and the size of the output are not issues.

This rule loads all sections and graphics and ignores the LoadFAPBitmap and LoadCordFAP INI options. There are no parameters for this rule.

The return values are: msgSUCCESS or msgFAIL.

Example

Here is an example:

```
;InlineImagesAndBitmaps;;;
```

This rule must be placed (run) before the NAFILE.DAT and POLFILE.DAT files are unloaded and after the pagination rules. Here are some examples of how you would set up your AFGJOB.JDT file:

For single-step execution

```
<Base Form Set Rules>
;NoGenTrnTransactionProc;;;
;ResetOvFlw;;;
;BuildFormList;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
;PrintFormset;;;
;WriteOutput;;;
;WriteNaFile;;;
;InlineImagesAndBitmaps;;;
;BatchingByRecipINI;;;
;PaginateAndPropagate;;;
```

For multi-step execution

```
<Base Form Set Rules>
;RULStandardTransactionProc;;;
;LoadExtractData;;;
;ResetOvFlw;2;;
;IfRecipUsed;;BATCH1=INSURED;
;BuildFormList;;;
;LoadRcpTbl;;;
;RunSetRcpTbl;;;
;UpdatePOLFile;;;
;InlineImagesAndBitmaps;;;
```

See also JDT Rules Reference on page 30

InsNaHdr

The InsNAHdr rule is a legacy rule that few installations would ever need to use. This rule has no affect unless you also include this INI option:

```
< RunMode >
  NAUnload = No
```

NOTE: *Do not* set the NAUnload option to No unless you are specifically directed to do so by Oracle Insurance services or support personnel.

The InsNaHdr rule and this option tells the system the NAFILE.DAT file will not be unloaded in a single process. Instead, it will be unloaded a piece at a time. Specifically, the system unloads the section header into the NAFILE.DAT file before the remainder of the section is processed.

Using this rule implies that you will create the NAFILE.DAT file as the form set is being processed, instead of waiting until after the process has completed and creating the NAFILE.DAT file in one step.

Syntax

```
InsNaHdr( )
```

There are no parameters for this rule. This rule builds the NA header and appends it to NAFILE.DAT file.

Example

```
;InsNaHdr;3;;
```

See also JDT Rules Reference on page 30

InstallCommentLineCallback

Use this job level (level 1) rule during the AFP printing process to write transactional information into each page of the print stream. The information is written using AFP comment records and contains the recipient batch record information — the same information written into recipient batch files for each transaction.

Before adding the recipient batch record information as a comment record on each page, this rule also calculates and updates several GVM variables and structures that can be used by other rules which are executed during the print process. The values updated include the number of pages in each batch and the current page within the print stream.

The CurPage and TotPage GVM variables must be declared within the recipient batch record definition. Here is an example:

```
< FIELD:CurPage >
    INT_TYPE    = LONG
    EXT_TYPE    = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH  = 10
< FIELD:TotPage >
    INT_TYPE    = LONG
    EXT_TYPE    = CHAR_ARRAY_NO_NULL_TERM
    EXT_LENGTH  = 10
```

Normally, the CurPage variable reflects the current page number within the print stream. This is not necessarily the same as the page number that might actually print on the document. The TotPage variable reflects the total number of pages within a given transaction. Depending the other rules in use during the process, the value or meaning of these GVM variables can vary.

The comment information written into the print stream can serve multiple purposes, such as to later facilitate 2-up printing. During a 2-up printing process, you sometimes need to know whether the page on the left and the page on the right are from the same or different transactions. By having the recipient batch record information written into each page, it is possible to query that information and make the appropriate determination. You can use the ParseComment rule during the 2-up printing process to reconstruct the associated GVM variables in memory from the recipient batch record information stored in these comment records.

Syntax

```
; InstallCommentLineCallback;1; ;
```

This rule has no parameters.

Example

```
; InstallCommentLineCallback;1; ;
```

See also

ParseComment on page 176

JDT Rules Reference on page 30

JobInit1

Use this job level rule (level 1) to initialize resources such as input files, output files, and tables.

Syntax

```
;JobInit1;;;
```

This rule opens the log file, opens the extract file, creates the NA and POL files, opens forms set file, and opens and initializes recipient batch files. When finished, this rule closes the files it opened during the pre-processing stage.

Example

```
;JobInit1;;;
```

See also JDT Rules Reference on page 30

LoadDDTDefs

Use this job level rule (level 1) to load the field rules from the MASTER.DDT file into an internal linked list. You must include this rule in the AFGJDT.JDT file if your field level rules are defined in the MASTER.DDT file.

This rule is used with the Master field level rule.

Syntax

```
;LoadDDTDefs;;;
```

If you have variable fields that you use on most of your forms, such as Name and Address fields, you can use the MASTER.DDT file to store these variable field mappings.

If you use the MASTER.DDT file, add the Master rule to all variable fields on the section. The Master rule tells the system to look in the MASTER.DDT file for mapping information for those variable fields.

Using the MASTER.DDT file is helpful if you need to make a change to the variable field mapping because you only have to make changes once in the MASTER.DDT file. It's also helpful when you are setting up complicated rules since you only have to map the fields once. Test your mappings in the MASTER.DDT file before you copy them to other variable field mappings.

Example

```
;LoadDDTDefs;;;
```

See also

Master on page 379

Setting Up the MASTER.DDT File on page 507

JDT Rules Reference on page 30

LoadExtractData

Use this form set level (level 2) rule to load extract data into memory for each transaction. You must include this rule if any subsequent rules will search for or use extract data.

You must include this rule if:

- You are executing the GenTrn, GenData, and GenPrint programs as separate processes (multi-step processing), and
- Subsequent rules will search for or use extract data

If you omit this rule from the AFGJOB.JDT for multi-step processing, you will receive these error messages:

```
DM10702:  Warning in BuildFormList(): No extract records.
VMMCountList(pRPS->ExtractListH) = 0. Processing will continue.
DM12018:  Error in RPDoBaseFormsetRulesForward(): Unable to
<BUILDFORMLIST>().
```

NOTE: Do not include this rule if you are using the NoGenTrnTransactionProc rule. Doing so will cause the GenData program to go into a processing loop.

Syntax

```
;LoadExtractData;;;
```

Example

```
<Base Form Set Rules>
;RULStandardTransactionProc;;;
;LoadExtractData;;;
;ResetOvFlw;;;
;IfRecipUsed;;BATCH1=INSURED;
;IfRecipUsed;;BATCH2=COMPANY;
;IfRecipUsed;;BATCH3=AGENT;
;BuildFormList;;;
;LoadRcpTbl;;;
;UpdatePOLFile;;;
;RunSetRcpTbl;;;
;ProcessQueue;;PostPaginationQueue
```

See also

NoGenTrnTransactionProc on page 168

JDT Rules Reference on page 30

LoadFormsetFromArchive

Use this form set level (level 2) rule to extract a form set from a DAP archive based on archive keys stored in a standard extract file.

NOTE: You can use the LoadFormsetFromArchive rule to replace the BuildFormList, LoadRcpTbl, and RunSetRcpTbl rules in the AFGJOB.JDT file.

Because the LoadFormsetFromArchive rule loads a complete form set, it is usually not necessary to execute the section and field level rules. To skip section and field rule processing it is necessary to specify the appropriate rules in the AFGJOB,JDT file. See the example below. If required, the loaded form set can be modified using other transaction level rules or DAL scripts.

Syntax

```
;LoadFormsetFromArchive;;;
```

There are no parameters for this rule, but you can use these INI options:

```
< LoadFormsetFromArchive >
    Key            =
    DisplayFields  =
    TempFile       =
    Debug          =
```

Option	Description
Key	Use this option to build the search request for the APPIDX. You can specify multiple Key options if necessary. The first transaction that matches the values for the fields is extracted from archive. You can specify the Key search value as an XML or standard flat file search mask, such as *(1, HEADER)*. Here is another example: `Field(UNIQUEID) Search(!/Form/UNIQUEID) OFFSET(1) Length(40)` If the keys are not unique, the extracted matching transaction can be arbitrary.
DisplayFields	(Optional) Use this option to specify a list of archive index fields you want printed to the log file and console as the system processes the transaction.
TempFile	(Optional) Use this option for debugging purposes. If you include this option, the system writes the NA and POL files to the temp file model name you specify. For example, if you specify TEMP.TXT, the NA and POL files are written to TEMPNA.TXT and TEMPPOL.TXT, respectively.
Debug	(Optional) Enter Yes to have the system write debug information into the log file.

Example Here is an example AFGJOB.JDT file:

```
/*  JDT Rules for Single-Step Processing Batching By Recipient. */
<Base Rules>
;RULStandardJobProc;1;Always the first job level rule;
;JobInit1;1;;
;InitPrint;1;required to execute gendata/genprint in single step;
/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;2;;
;LoadFormsetFromArchive;2;;
;PrintFormset;2;;
;WriteOutput;2;;
;WriteNaFile;2;;
;BatchingByRecipINI;2;;
/* Every section in this base uses these rules. */
<Base Image Rules>
;WIPImageProc;;
/* Every field in this base uses these rules. */
<Base Field Rules>
;RULWIPFieldProc;;
```

See also

BuildFormList on page 72

LoadRcpTbl on page 157

RunSetRcpTbl on page 212

JDT Rules Reference on page 30

LoadListFromTable

Use this job level rule (level 1) to load a table specified in your FSISYS.INI file's Data control group into a link list and place the handle of the list into the GVM variable you specify.

Syntax

```
;LoadListFromTable;;;
```

This rule has these parameters:

Parameter	Description
GVM_LISTNAME	The name of the GVM variable in which the handle of the list should be stored.
INI_TABLE_OPTION	The name of the option in the Data control group in the INI file you want the system to load. This option should specify the name of the table.
COMMENT_CHARACTER	A single character which indicates the comment character. Lines beginning with this character are not loaded into the link list.

Example

```
;LoadListFromTable;;POLTYPES FORM_SCHED_POL_TYPE *;
```

If your FSISYS.INI file has these settings:

```
< Data >
    TablesPath         = ..\MSTRRES\TABLES\
    Form_Sched_POL_Type= POLTYPE.TBL
```

The LoadListFromTable rule loads the POLTYPE.TBL file into a list whose handle is stored in a GVM variable named POLTYPES. Any line in the file that starts with an asterisk (*) is omitted from the list.

See also

JDT Rules Reference on page 30

LoadRcpTbl

Use this form set level (level 2) rule to load entries from the SETRCPTB.DAT file based upon the current Key1 field (such as company), Key2 field (such as line of business), and the transaction type.

NOTE: You can use the LoadFormsetFromArchive rule to replace the BuildFormList, LoadRcpTbl, and RunSetRcpTbl rules in the AFGJOB.JDT file.

Only those entries in the SETRECPTB.DAT file that match the Key1, Key2, and transaction type will be loaded and processed. By loading only those that match, the processing becomes more efficient.

Syntax

```
;LoadRcpTbl;;;
```

Example

```
;LoadRcpTbl;;;
```

See also

BuildFormList on page 72

LoadFormsetFromArchive on page 154

RunSetRcpTbl on page 212

JDT Rules Reference on page 30

LoadTblFiles

Use this job level rule (level 1) to load one or more text tables listed in the tables list file, which is defined in the FSISYS.INI file's Data control group, in the TblFile option.

This rule lets you make available many table files for use by Documaker Server. These table files can contain lists of codes, abbreviations, and addresses which might be hard to maintain in a large extract file.

Syntax

```
;LoadTblFiles;;;
```

This rule frees memory resources used to store the tables. Here is an example of the TblFile option:

```
< Data >
    TblFile = .\deflib\TblFile.Dat
```

Example

```
;LoadTblFiles;;;
```

This example loads all the tables listed in the tables list file defined in the INI file. The table data can then be accessed using the MovTbl rule.

See also MovTbl on page 413

JDT Rules Reference on page 30

LoadTextTbl

Use this job level rule (level 1) to load all specified text tables into the text table list for use by field level rules. All text tables should be listed in the text table listing file, which is defined in the FSISYS.INI file's Data control group, in the TextTbl option.

This rule loads all defined tables and makes them available for use by Documaker Server. These text files can contain paragraphs and messages which might be hard to maintain in a large extract file.

Syntax

```
;LoadTextTbl;;;
```

When finished, this rule erases the text table list. Here is an example of the TextTbl option:

```
< Data >
    TextTbl = TextTbl.Dat
```

Example

```
;LoadTextTbl;;;
```

See also JDT Rules Reference on page 30

MergeAFP

Use this form set level (level 2) rule to initialize input files. This rule populates the AFP record list, retrieves RCB comment records, and terminates the input files.

This rule also initializes output files, and writes out the AFP record list, adding end page and end document records as necessary. The rule then terminates these output files.

Syntax

```
;MergeAFP;;;
```

Example

```
;MergeAFP;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

MergeRecipsFromForm

Use this form set level (level 2) rule to assign the recipients from a specific form to the other forms in a form set. This lets you reduce the number of recipient triggers when all recipients receive the majority of the forms in the form set.

Syntax

```
;MergeRecipsFromForm;;FormName, Z flag;
```

Parameter	Description
FormName	The name of the form from which the recipient names are copied. The other forms in that form set are assigned these recipients, if they're not already there. The form name you specify can occur multiple times in the form set and the unique recipient names from all copies are assigned to the remaining forms in the form set.
Z flag	If you want the system to copy recipients with a zero copy count, enter the character *Z* (or *z*). The default is blank.

Example

```
;MergeRecipsFromForm;;Mailer Form;
```

Assume that before processing, the recipients for this form set are set up as follows:

- Standard Form A - RECIPS=(Home Office)
- Standard Form B - RECIPS=(Home Office)
- Mailer Form - RECIPS=(Home Office, Agent 1)

After using this rule to process the form set, the recipients for each form are now set to:

- Standard Form A - RECIPS=(Home Office, Agent 1)
- Standard Form B - RECIPS=(Home Office, Agent 1)
- Mailer Form - RECIPS=(Home Office, Agent 1)

NOTE: If you want the system to copy recipients with a zero copy count, use the Z flag (it's not case sensitive). Here is an example:

```
;MergeRecipsFromForm;2;FormName,Z;.
```

The system ignores that recipient and does not copy it to the other forms in the form set if the copy count is set to zero for a recipient and the Z flag is omitted.

See also JDT Rules Reference on page 30

MergeWIP

Use this job level (level 2) rule to initialize GenData WIP Transaction Processing. This rule creates a transaction memory list to which it adds transactions from the WIP file that have status codes which match those in the rule's parameters.

The status codes identified by this rule do not have to be identified by the WIPTransactions rule. You can include status codes for transactions that you want to delete from the WIP file. Transactions with status codes not including in this rule's parameters remain in the WIP file when processing finishes.

These other rules are also used when you run WIP Transaction Processing:

- WIPTransactions – This rule replaces the RULStandardTransactionProc or NoGenTrnTransactionProc rules in the AFGJOB.JDT file. This rule starts GenData WIP Transaction Processing at the form set level. It also identifies the status codes for the transactions in the WIP file that are processed. The status codes can be a subset or all of the status codes identified on the MergeWIP rule or none.
- GVM2GVM - This rule copies GenData execution data from the Trigger2WIP INI control group.
- WIPImageProc – This rule replaces the RULStandardImageProc or StandardImageProc rule.
- WIPFieldProc – This rule replaces the RULStandardFieldProc or StandardFieldProc rule.

Using these rules in a simplified AFGJOB.JDT file and with appropriate INI files, GenData WIP Transaction Processing adds the transactions from a WIP file to a transaction memory list. It then processes the transactions from the memory list, appending the data from the WIP file to the MRL recipient batch, NewTrn, NA, and POL files. If these files do no exist, it creates them. Each transaction in the memory list is deleted from the WIP file after it is processed.

NOTE: If you are using the MergeWIP rule with the BatchingByRecipINI rule, be sure to use the =DAL and =GVM operators. For more information, see Formatting Data with the = Operator on page 267. The MergeWIP rule gets all of its data from WIP, not an extract file.

Syntax

```
;MergeWIP;;StatusCode1,StatusCode2,...;
```

Parameter	Description
StatusCode	Use the StatusCode parameters to define the status codes of the transactions in the WIP file you want to add to the memory list. Identify the status codes you want to included in the Status_CD control group. Here is an example: `< Status_CD >` `Accepted = AC` `Approved = AP` `BatchPrint = BP` `Rejected = RJ`

WIP selection performance during batch processing

The system automatically limits the result sets queried from DBMS systems by the MergeWIP rule to the rule's parameters for the STATUSCODEs to significantly improve query speeds when this rule is run against large WIP index tables.

When the WIP index table is in a DBMS, the appropriate WHERE clause for the STATUSCODE is added to the query automatically. A separate query for each STATUSCODE provided to the MergeWIP rule is used to retrieve the WIP list.

To improve performance on very large WIP tables, add an index on the DBMS WIP index table on the STATUSCODE column to avoid full table scans. An index on the WIP index table on the CURRUSER and STATUSCODE columns can also improve the performance of other queries to the WIP index table when the query to build a WIP list for a specific user is performed.

Using dates to select transactions

The MergeWIP rule can check a date field in the WIP index to determine whether to insert the WIP record into the batch. By default the field name is *ScheduleDate*. You can use this INI option to change the name of the field that is used:

```
< MergeWIP >
    ScheduleDateFieldName = ScheduleDate
```

If the data in the field is eight bytes, the system assumes the date is in YYYYMMDDD format. If the data in the field is 14 bytes, the system assumes a YYYYMMDDhhmmss format. You can change the format the MergeWIP rule expects for the date using this INI option:

```
< MergeWIP >
    ScheduleDateFieldFormat = D4%1
```

You can combine date and time formats into one string separated by percent signs (%). This table shows the date formats:

Enter	For this format
1	MM/DD/YY 99/99/99 (default)
2	DD/MM/YY 99/99/99
3	YY/MM/DD 99/99/99
4	Month DD, YY Month DD, YYYY
5	MM/DD/YY ZZ/ZZ/ZZ
6	DD/MM/YY ZZ/ZZ/ZZ
7	YY/MM/DD ZZ/ZZ/ZZ
8	MM/DD/YY LZ/LZ/LZ
9	DD/MM/YY LZ/LZ/LZ
A	YY/MM/DD LZ/LZ/LZ
B	MMDDYY ZZZZZZ
C	DDMMYY ZZZZZZ

Enter	For this format
D	YYMMDD ZZZZZZ
E	MonDDYY MonZZZZ
F	DDMonYY ZZMonZZ
G	YYMonDD ZZMonZZ
H	DOY/YY ZZZ/ZZ
I	YY/DOY ZZ/ZZZ
J	DD Month, YY DD Month, YYYY
K	YY, Month DD YYYY, Month DD
L	Mon-DD-YY Mon-ZZ-ZZ
M	DD-Mon-YY ZZ-Mon-ZZ
N	YY-Mon-DD ZZ-Mon-ZZ
O	Mon DD, YY Mon DD, YYYY
P	DD Mon, YY DD Mon, YYYY
Q	YY, Mon DD YYYY, Mon DD
R	Month Month

This table shows the time formats:

Enter	For this format
1	HH:MM:SS 99:99:99(default)(24 hour)
2	HH:MM:SS XM 99:99:99 XM (12 hour)
3	HH:MM 99:99 (24 hour)
4	HH:MM XM 99:99 (12 hour)

Returning a warning message

Instead of returning an error if it comes across an empty WIP list when merging WIP, the system can issue a warning. To have the system issue a warning instead of an error, set the WIPWarnOnEmpty option to Yes, as shown here:

```
< RunMode >
    WIPWarnOnEmpty = Yes
```

Changing the WIP Status

You can tell the system not to delete WIP records and files during the MergeWIP/WIPTransactions process if an error occurs, but instead change the WIP status to something you define.

This way, if an error occurs during batch processing, the WIP will still exist in its normal place. But since its status has changed, the system will not include it in the next batch run. You can then examine the transaction to determine what caused the error.

Use the following INI option to set up the transaction error code you want to use:

```
< Status_CD >
    TransErrCode = E
```

Option	Description
TransErrCode	You can enter up to two characters, numbers or both for the transaction error code.

Example

Here is an example:

```
;MergeWIP;; Approved, Accepted, Rejected;
```

This example adds to the memory list the transactions in the WIP file which have these status codes: Approved, Accepted, and Rejected. Those codes must be specified in the Status_CD control group.

See also

GVM2GVM on page 107

WIPFieldProc on page 243

WIPImageProc on page 244

WIPTransactions on page 245

GenData WIP Transaction Processing on page 9

JDT Rules Reference on page 30

MultipleDataDictionaryFiles

Use this form set level (level 2) rule to specify multiple data dictionaries (XDBs) to use across multiple Key1/Key2 combinations. You can specify which item (Key1, Key2) or combination of items determines the switch. If the database is not found in the list of possibilities, the system loads the default XDB, as specified by the original INI option.

Syntax

```
;MultipleDataDictionaryFiles;2;parameters;
```

You specify the parameters based on which key tells the system when to switch from one data dictionary to another. You can use the keys individually or in any combination.

Example

Here are some examples:

Example	This tells the system to switch XDBs
;MultipleDataDictionaryFiles;2;Key2;	Based on the Key2 field
;MultipleDataDictionaryFiles;2;Key1,Key2;	Based on the Key1, Key2 combination

You specify XDB files using the MultiDataDict control group. For each XDB file, use an INI option similar to the one shown here:

```
< MultiDataDict >
    File = FileName;IDFormat
```

Option	Description
File	For *FileName*, include the full file name and path followed by a semicolon. The IDFormat is based on the parameters you supplied to the rule. Note: In the first example below, the *IDFormat* would be *Key2Value*, for the second example, it would be *Key1Value;Key2Value*.

Based on the first example, assume Key2 has these possible values:

CAR, BOAT, and MISC

with corresponding XDBs of:

CARXDB.DBF, BOATXDB.DBF, and MISCXDB.DBF

The AFGJOB.DAT file would contain:

```
;MultipleDataDictionaryFiles;2;Key2;
```

The INI file would contain:

```
< MultiDataDict >
    File = CARXDB.DBF;CAR
    File = BOATXDB.DBF;BOAT
    File = MISCXDB.DBF;MISC
```

Thus whenever the Key2 ID changed to one of these values, the appropriate XDB would be loaded.

Based on the second example, assume Key1 has these possible values:

LIFE and VEHICLE

Assume Key2 has these possible values:

Under the VEHICLE Key1	CAR, BOAT, MISC
Under the LIFE Key1	SINGLE, MARRIED

The AFGJOB.DAT file would contain:

```
;MultipleDataDictionaryFiles;2;Key1,Key2;
```

The INI file would contain:

```
< MultiDataDict >
    File = CARXDB.DBF;VEHICLE;CAR
    File = BOATXDB.DBF;VEHICLE;BOAT
    File = MISCXDB.DBF;VEHICLE;MISC
    File = SINGXDB.DBF;LIFE;SINGLE
    File = MARRXDB.DBF;LIFE;MARRIED
```

Whenever the Key1 and Key2 combination changed to one of these values, the system would load the appropriate XDB.

See also JDT Rules Reference on page 30

NoGenTrnTransactionProc

Use this form set level (level 2) rule when you use the GenData program by itself to execute the GenTrn and GenData steps. In that processing environment, this rule, processes the extract file and creates the information normally created in both the GenTrn and GenData steps.

When combined with the InitPrint and PrintFormset rules, it creates the output files created during the GenPrint step.

NOTE: Do not use this rule if you are running the GenTrn, GenData, and GenPrint programs as separate processes. Do not use this rule with the LoadExtractData rule. Doing so will cause the GenData program to go into a processing loop.

This rule replaces the RULStandardTransactionProc rule in the performance mode JDT.

Syntax

```
;NoGenTrnTransactionProc;;;
```

This rule loads extract file records for the current transaction into memory. To use this rule, you must add the following options in your FSISYS.INI or FSIUSER.INI file:

```
< Data >
    TrnFile = <CONFIG:~Platform > TrnFile
< CONFIG:PC >
    TrnFile = NUL
```

...or else include a TRNFILE.DAT file, which you can leave empty.

Example

```
;NoGenTrnTransactionProc;;;
```

See also

RULStandardTransactionProc on page 210

InitPrint on page 146

PrintFormset on page 182

LoadExtractData on page 153

StandardFieldProc on page 235

StandardImageProc on page 236

Rules Used in Single-Step Processing on page 25

JDT Rules Reference on page 30

OMRMarks

Use this job level (level 1) rule to generate OMR marks on 1-up documents printed on any supported base system printer or on 2-up documents printed on any AFP printer that supports 2-up printing.

OMR marks are used to indicate ZIP code change, demand feed, inserts and so on. OMR marks are solid boxes placed on a page.

The rule loops through the pages of the form set and creates special sections for each page with the required OMR marks. The OMR marks are based on special settings in your FSISYS.INI or FSIUSER.INI file. The INI file settings use special global variable names, rule names, and conditions to trigger specific OMR marks.

Syntax

```
;OMRMarks;;Cond(LetterOMR);
```

Parameter	Description
Cond	(Optional) Indicates special conditions exist for printing OMR marks.
LetterOMR	(Optional) Refers to the data in the Condition table (CONDTBL). The true or false for this condition triggers the printing of the set of OMR marks for particular transaction.

Place this rule *after* the WriteNAFile rule in the AFGJOB.JDT file.

NOTE: Keep in mind the OMRMarks rule is a post process rule. This means that pagination and propagation takes place before the rule is called. So, this rule goes back through the form set after the forms have been created and places a mark on each page.

You must include the PaginateAndPropagate rule after the OMRMarks rule because, during post processing, the system executes the rules in the JDT file from bottom to top.

Example

Here is an example from the Condition table (CONDTBL):

```
< Conditions >
    LetterOMR: LetterCode = "0006" or LetterCode = "0039" or
       LetterCode = "0040"
```

Here is an example from the record dictionary definition:

```
< Variables >
    LetterCode = GVM(CD-LTR-TYPE)  Length(4) Type(Char)
```

LetterCode is a global variable with the name, *CD-LTR-TYPE*. This variable has a type of *Char* (character) and a length of four. In this example, any time the condition is true, the system prints OMR marks on the page created for the transaction it is processing.

You must update your FSISYS.INI or FSIUSER INI files as follows.

Enter the path for your table files in the MasterResource control group. Use the TablePath option to define the table file's path.

```
< MasterResource >
    TablePath  =  \deflib\
```

Enter the file name of your Condition table in the Tables control group. Use the Conditions option to define the Condition table's file name.

```
< Tables >
    Conditions  =  CondTbl
```

Create the OMR_Params control group with all necessary options in your FSISYS.INI or FSIUSER INI file.

Here is an example of INI settings for 1-up printing:

```
< OMR_Params >
    Mark = Cord(2100, 2140, 300, 1000), RuleParms(INSERT2),
      Rule(FlagFromGVM), When(All)
    Mark = Cord(4200, 4240, 300, 1000), Rule(Always), When(All)
    Mark = Cord(2100, 2140, 700, 1300), Rule(Always), When(All)
    Mark = Cord(6300, 6340, 700, 1300), RuleParms(A-AND-C),
      Rule(Always), When(All), Cond(ac)
```

Here is an example of INI settings for 2-up printing:

```
< OMR_Params >
    Mark =
Cord(Left(2100,2140,300,1000),Right(2100,2140,32500,33200)),
      RuleParms(Insert2),Rule(FlagFromGVM),Page(B), When(All)
    Mark = Cord(Left(4200,4240,300,1000),
      Right(4200,4240,32500,33200)), Rule(Always),Page(B), When(All)
    Mark = Cord(Left(2100, 2140, 700, 1300), Right(2100, 2140, 32500,
      33200)), Rule(Always),Page(B), When(All)
```

Parameter	Description
Mark	Definition for each OMR mark. You need a definition for each OMR mark that can be generated on a page. For instance if eight is the maximum number of OMR marks per page, you need eight mark definitions in this control group.
Cord(t,b,l,r)	Top, bottom, left, and right coordinates for the page in FAP units (1 inch = 2400 FAP units) for 1-up printing.
Cord(Left(t,b,l,r), Right(t,b,l,r))	Top, bottom, left, and right coordinates for the page in FAP units (1 inch = 2400 FAP units) for 2-up printing. Left is for the left side and right is for the right side of the 2-up paper.
Page()	Indicates which sides to print: *Both* or *B* = both left and right; *Left* or *L* = left side only; *Right* or *R* = right side only.
Rule()	The name of the OMR rule to execute.
RuleParms()	The Input parameter to the rule, as specified in the Rule() parameter.

Parameter	Description
When(All)	When to print on pages: *All* - print on all pages of the transaction output. *FirstOnly* – only print on the first page of transaction output. *LastOnly* – only print on the last page of the transaction output. *ExceptFirst* - print on all pages except the first page. *ExceptLast* - print on all pages except the last page.
Cond()	Condition, if true, then print OMR mark.

The OMR rules are:

Rule	Description
Always	Always generate the OMR mark.
Cond()	You can use condition logic for one OMR mark or for the whole set. Here is an example: *Cond(ac)*
DivertPage	Generates the OMR mark only for a special *divert* page. This only applies to 2-up printing. If you use this rule, you must also define the DivertOpt and DivertOMR options in the TwoUp control group.
FlagFromGVM	Generates the OMR mark if the GVM variable defined in the RuleParms is set to zero (0) or one (1). You control generation based on data in an extract record (using the Ext2GVM or Field2GVM rules) or based on a DAL script which uses the SetGVM function. The GVM variable must be in the RCBDFDFL and TRNDDFDFL files.
ZipCodeChange	Generates the OMR mark only when the ZIP code has changed. You must define the parameter associated with this rule as a global variable or else define it in the RCBDFDFL.DAT file.

```
< TwoUp >
    CounterTbl = Counter.tbl
    LMargin    = 0
    LShift     = -240
    RShift     = 16560
    DivertOpt  = No
    DivertOMR  = OMR20
    PageSize   = 40800
```

OMR marks are not supported in these situations:

- In 1-up printing when you have multiple copies of the same form
- In 2-up printing when you are printing duplex

See also

PaginateAndPropagate on page 174

Rules Used in Single-Step Processing on page 25

Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

Using Condition Tables on page 492

PageBatchStage1InitTerm

Use this job level rule (level 1) to create and populate a list of records which contain page ranges and total page counts for each recipient batch file.

This rule is typically used for handling 2-up printing for AFP and compatible printers. This rule is also used with multi-mail processing.

This rule creates a list (populated in another rule) to contain the recipient batch records for a multi-mail transaction set. The rule then writes out the recipient records for the final multi-mail transaction set and writes out the total page counts for each recipient batch.

- Fields must be added to the RCBDFDFL.DFD file for the file containing the total page counts for the recipient batches. Do not remove or change the BatchName and RecordCount fields.
- The name of the file containing page counts should be specified in the CounterTbl option of the TwoUp control group.
- Because the end of a multi-mail set is not signaled until the following transaction, you must write out the recipient records for the final transaction set at the job level.

Syntax

```
;PageBatchStage1InitTerm;;(MMField);
```

Parameter	Description
MMField	(Optional) Name of the INI option in the Trn_Fields control group which defines where the multi-mail code will be found in each transaction.

NOTE: If you use this rule, you must also use the BatchByPageCount and WriteRCBWithPageCount rules.

Example

If you omit the MMField parameter, the system uses standard batching by page count, as shown below:

```
;PageBatchStage1InitTerm;;;
```

If you include the MMField parameter, the system uses batching by multi-mail processing, as shown below:

```
;PageBatchStage1InitTerm;;MMField=MM_Field
```

See also

Rules Used for 2-up Printing on page 27

BatchByPageCount on page 47

WriteRCBWithPageCount on page 250

JDT Rules Reference on page 30

PaginateAndPropagate

Use this form set level (level 2) rule to paginate the form set and merge in or propagate field data.

Syntax

```
;PaginateAndPropagate;Debug FooterMode;
```

Parameter	Description
Debug	If you include this parameter, the system writes debug information about pagination to the log file. For performance reasons, you would not typically use this option unless directed by support.
FooterMode	This parameter controls how the footers are treated in regards to pagination. You can enter 0, 1, or 2. The default is zero (0) which is the standard way the rule handles footers.

Normally this rule is placed in the Base Form Set Rules section of the AFGJOB.JDT file at or near the end of the rule list. This location is important because you want the rule to execute as one of the first steps of transaction post-processing. Place this rule after the PrintFormset rule in the Base Form Set Rules section of the AFGJOB.JDT file when running in single-step mode.

This rule is a post-process rule, meaning that initial pagination and propagation takes place before the rule is called. This rule then goes back through the form set.

Example

Here are some examples:

```
;PaginateAndPropagate;;
;PaginateAndPropagate;2,Debug FooterMode(2);
;PaginateAndPropagate;2,FooterMode(1);
```

NOTE: The PaginateAndPropagate rule looks for the CanSplitImage indicator. If missing, sections are paginated in the standard method.

Footer modes 1 and 2 search the logical page for footers marked as *copy-on-overflow* and then determine the upper limit of those footers. That point becomes the lower limit of the body sections.

As the system checks the body sections to determine if they exceed the lower limit, it raises the lower limit if there is another footer with a higher limit. This prevents the a large footer at the bottom of a long logical page from affecting pages on which it does not appear.

Modes 1 and 2 differ in how they handle an overlap when a footer is encountered that raises the lower limit above a body section that is already determined to fit on the page.

With mode 1, the system increases the limit and reevaluates the section on that page. It also lets the second from the last page have a large area reserved with nothing printed.

With mode 2, the system moves the footer to the next page so the footer can appear on a page by itself.

With mode zero (0), the default, the system searches the logical page for all footers and determines the upper limits of the footers. That becomes the lower limit of the body sections. When a body section exceeds this lower limit, the system splits the logical page into two pages.

The section that exceeds the limit and all following sections are moved up and to the second page. Sections on the first page marked as copy-on-overflow are copied to the second page. Sections on the second page marked as copy-on-overflow are copied back to the first page. The system then searches the second page for all footers, determines the upper limit of that footer, and continues the process.

NOTE: Previously, this rule was known as the PaginateAndPropogate rule. You can use either spelling.

See also

CanSplitImage on page 304

Rules Used for 2-up Printing on page 27

Rules Used in Single-Step Processing on page 25

PrintFormset on page 182

JDT Rules Reference on page 30

ParseComment

Use this form set level (level 2) rule when merging two AFP print streams in a 2-up printing process. This rule parses the recipient batch record information written as an AFP comment into each printed page back into the GVM variables associated with the recipient batch DFD. You would do this, for example, if you need to know whether the page on the left and right side (as accomplished through 2-up printing) are from the same transaction.

For this rule to be useful, the appropriate comment records, matching the recipient batch record DFD, must have been added to the print streams that are being merged in the 2-up printing process. A rule such as InstallCommentLineCallback is used during the original print step is an example.

Syntax

```
ParseComment; ;Side;
```

Parameter	Description
Side	You can enter *Left* or *Right* or omit this parameter. Including *Left* or *Right* specifies that you want the system to parse the comment record from either the left or right side. The system parses the data from the comment record into the first (primary) instance of the associated GVM variables. If you omit this parameter, the associated variables from both sides are parsed and stored. The left side comment data is parsed into the first (primary) instance of the associated GVM variables. The right side comment data is parsed into the second instance of the associated GVM variables.

Example

This example shows how to use this rule to access the specific occurrences of RCB comment records retrieved from AFP files.

```
;ParseComment;;Left;
;PreTransDal;;MyScript;
;ParseComment;;Right;
;PreTransDAL;;MyScript;
```

If you include the Side parameter, be sure to finish using the values from one record before parsing the other record, because this method replaces the primary instance of the GVM variable data. This example also shows that you can use a DAL script to manipulate the parsed GVM variables.

If you want to use a DAL script and get data from both sides (omitting the Side parameter), you would specify two (2) as the optional second parameter to the GVM function to access the second (right side) set of data.

```
LeftData = GVM( name )
RightData = GVM( name, 2 )
```

See also

InstallCommentLineCallback on page 150

JDT Rules Reference on page 30

PostTransDAL

Use this form set level rule (level 2) in the AFGJOB.JDT file to execute a DAL script on the POST_PROC_A message. The PostTransDAL rule executes after other form set rules and section level rules.

You can use this rule to handle follow up tasks after form set rules and section rules are executed. For example, you can use this rule to clear or change GVM and internal DAL variables.

Syntax

```
;PostTransDAL;;string;
```

Parameter	Description
String	A character string that contains a DAL function or DAL script.

Although you can use DAL to access almost any form set or section field, keep in mind those fields may not exist, depending on where you place this rule in the transaction job rule list. And, unlike the DAL or IF rules, there is no return value from the execution of a transaction level DAL script.

Use this form to get extract data if the script is contained in the rule data. You cannot use this form in external script files.

```
A = {1,MIS257 138,1}
```

Where *A* is a DAL variable you wish to assign. The bracketed {} item can be almost any standard search mask supported by the Get Record infrastructure. In this case, *1,MIS257* is the search criteria. If the record is found, the system takes the data from position 138, length 1 as indicated by *138,1*.

This method also lets you specify an occurrence of the record by including a hyphen with a numeric value, such as *-n*, after the data length. Here is an example:

```
A = {1,MIS257 138,1-5}
```

Here the function searches for the 5th occurrence of the 1,MIS257 record. If you omit the occurrence, the system returns the first one found. If it cannot find the requested record, the system assigns the variable an empty “” value.

NOTE: To specify multiple DAL statements in the rule data area, separate the DAL statements using two colons (::). Normally, semicolons separate DAL statements, but this character is illegal in the rule data area.

Example

```
;PostTransDAL;;Call("posttran.dal");
```

This example executes the Call DAL function which executes the DAL script contained in the POSTTRAN.DAL file in the DefLib directory specified in your MRL.

```
;PostTransDAL;;If  HaveGVM("main_address") Then
SetGVM("main_address", "25 Brown St.", , "C", 20)::End;
```

In this example, the system checks to see if the GVM variable (*main_address*) exists. If not, it creates a character array GVM variable (*main_address*) 20 characters is length and stores the character string (25 Brown Street) in the array.

Here is another example:

Suppose you want any transaction that contains the following XML tag with a value of *N* to be processed and printed, but not archived:

```
INVOICE/DOCUMENT_ID/ARCHIVE
```

To accomplish this, add the following to the AFGJOB.JDT file:

```
;PostTransDAL;;a = {!/INVOICE/DOCUMENT_ID/ARCHIVE 1,1}::If a="N"
Then a="Y":: Else a="N":: End::SetGVM("SentToManualBatch", a, ,"C",
2);
```

See also PostImageDAL on page 422

PreImageDAL on page 426

PreTransDAL on page 179

JDT Rules Reference on page 30

PreTransDAL

Use this form set level rule (level 2) in the AFGJOB.JDT file to execute a DAL script on the PRE_PROC_A message. The PreTransDAL rule executes before other form set rules and before section rules.

You can use this rule to handle setup tasks which should occur before form set rules and section rules are executed. For example, you can use this rule to initialize GVM and internal DAL variables.

Syntax

```
;PreTransDAL;;string;
```

Parameter	Description
String	A character string that contains a DAL function or script.

Although you can use DAL to access almost any form set or section field, keep in mind those fields may not exist, depending on where you place this rule in the transaction job rule list. And, unlike the DAL or IF rules, there is no return value from the execution of a transaction level DAL script.

Use this form to get extract data if the script is contained in the rule data. You cannot use this form in external script files.

```
A = {1,MIS257 138,1}
```

Where *A* is a DAL variable you wish to assign. The bracketed {} item can be almost any standard search mask supported by the Get Record infrastructure. In this case, *1,MIS257* is the search criteria. If the record is found, the system takes the data from position 138, length 1 as indicated by *138,1*.

This method also lets you specify an occurrence of the record by including a hyphen with a numeric value, such as *-n*, after the data length. Here is an example:

```
A = {1,MIS257 138,1-5}
```

Here the function searches for the 5th occurrence of the 1,MIS257 record. If you omit the occurrence, the system returns the first one found. If it cannot find the requested record, the system assigns the variable an empty "" value.

NOTE: To specify multiple DAL statements in the rule data area, separate the DAL statements using two colons (::). Normally, semicolons separate DAL statements, but this character is illegal in the rule data area.

Example

```
;PreTransDAL;; trans_id={1,PrePost 1,8}::Chain("pretrans.dal");
```

This example sets the internal DAL variable, *trans_ID*, to the first eight-characters from the transaction record that matches the search mask: *1,PrePost*. Then the Chain DAL function executes the DAL script in the PRETRANS.DAL file in the DefLib directory specified in your MRL.

```
;PreTransDAL;;If (HaveGVM("main")) Then SetGVM("main_address", "25
Brown St.", , "C", 20)::End;
```

In this example, DAL checks to see if the GVM variable (*main_address*) exists. If not, it creates a character array GVM variable (*main_address*) 20 characters in length and stores the character string (25 Brown Street) in the array.

See also PostImageDAL on page 422

PostTransDAL on page 177

PreImageDAL on page 426

JDT Rules Reference on page 30

PrintData

Use this form set level (level 2) rule to print the form set. This rule is used for handling 2-up printing on AFP and compatible printers.

Syntax

```
;PrintData;;;
```

NOTE: The section handler installed by the InitPageBatchedJob rule is called during the printing stage. If you want to make any modifications to the recipient batch record, you must do so before this point.

Example

```
;PrintData;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

PrintFormset

Use this form set level (level 2) rule when you use the GenData program by itself to execute GenTrn and GenPrint processes (single-step processing). In that processing environment, this rule, when combined with the InitPrint rule, prints form sets.

NOTE: Do not use this rule if you are running the GenTrn, GenData, and GenPrint programs as separate processes.

Syntax

```
;PrintFormset;;;
```

This rule checks the recipients for the form set and then identifies the print batch in which this form set should be included. It then prints the form set.

This rule has no parameters.

When you use this rule, you must also include a BatchByPageCount or BatchingByRecipINI rule to produce print batches and the final print stream. If you omit either of these rules, you'll get the following error message:

```
Unable to <PRINTFORMSET>()
```

NOTE: You must define the BatchingByRecip control group to pass parameters to this rule. Use this control group to specify the batch names and search criteria (conditions) for the batches. If you receive this error message, also check the condition.

Example

Here is an example:

```
;PrintFormset;;;
```

You can also use the PrintFormset rule to create multiple print files when you run the GenData program in single-step mode to produce PDF or RTF output with multiple transactions. This capability is related to running Documaker under IDS (see the Internet Document Server Guide for more information). To do this, add the PrintFormset control group and these options to your INI file:

```
< PrintFormset >
    MultiFilePrint = Yes
    LogFileType    = XML
    LogFile        = (log file name and path)
```

Option	Description
MultiFilePrint	Set this option to Yes to allow multiple file print. The MultiFilePrint option should only be used with the PDF, RTF, HTML, and XML print drivers.
LogFileType	Specifies the type of the log file. Enter XML for an XML file. Any other entry results in a text file.

Option	Description
LogFile	Specifies the name of the log file. Include the full path. If you omit the path, the system uses DATAPATH. If you omit this option, the system creates a file named *TMP.LOG*. If you enter XML in the LogFileType option and a different extension here, the system uses *XML*.

NOTE: You must place this rule before the PaginateAndPropagate rule in the Base Form Set Rules section of the AFGJOB.JDT file when running in single-step mode. When running in multi-step mode, use the MultFilePrint callback functionality.

The log file that is created is either a semicolon-delimited text file, formatted like the file created by the MultiFilePrint callback function or an XML file. Here is an example of the layout of the XML file:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <LOGFILE>
- <TRANSACTION INSTANCE="1">
  <BATCH NAME="Logical Batch Name">.\data\BATCH1.BCH</BATCH>
  <GROUP1 NAME="Company">SAMPCO</GROUP1>
  <GROUP2 NAME="Lob">LB1</GROUP2>
  <TRANSACTIONID NAME="PolicyNum">1234567</TRANSACTIONID>
  <TRANSACTIONTYPE NAME="TransactionType">T1</TRANSACTIONTYPE>
  <RECIPIENT NAME="INSURED">INSUREDS COPY</RECIPIENT>
 <FILE>DATA\0rDcP7WxytE8ECp5jexhWXVqkV840Vw_F-GykT_VMfd.PDF</FILE>
  </TRANSACTION>
- <TRANSACTION INSTANCE="2">
  <BATCH NAME="Logical Batch Name">.\data\BATCH2.BCH</BATCH>
  <GROUP1 NAME="Company">SAMPCO</GROUP1>
  <GROUP2 NAME="Lob">LB1</GROUP2>
  <TRANSACTIONID NAME="PolicyNum">1234567</TRANSACTIONID>
  <TRANSACTIONTYPE NAME="TransactionType">T1</TRANSACTIONTYPE>
  <RECIPIENT NAME="COMPANY">COMPANY COPY</RECIPIENT>
 <FILE>DATA\0v3l7pBdVqHceoRL5hf2xqjJ7WMxiRVO9U70iFiIcne.PDF</FILE>
  </TRANSACTION>
</LOGFILE>
```

Use the options in the DocSetNames control group to determine which XML elements are created. The values in this control group are the same as those written to a recipient batch or TRN file.

See also

BatchByPageCount on page 47

BatchingByRecipINI on page 68

InitPrint on page 146

Single-Step Processing on page 7

JDT Rules Reference on page 30

PaginateAndPropagate on page 174

ProcessQueue

Use this job level (level 1) rule to process the queue you specify.

Syntax

```
;ProcessQueue;;(Queue);
```

Parameter	Description
Queue	The name of the queue you want to process.

This rule loops through the list of functions for the queue you specify and then frees the queue when finished.

Example

```
;ProcessQueue;;PostPaginationQueue;
```

This example tells the system to process the PostPaginationQueue.

See also

Single-Step Processing on page 7

Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

ProcessRecord

Use this form set (level 2) rule to switch between print files as necessary when printing 2-up forms on an AFP printer. This rule updates the page count for current print file and loads and merges the form set.

Syntax

```
;ProcessRecord;;;
```

Example

```
;ProcessRecord;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

ProcessTriggers

Use this form set level (level 2) rule to process the groups (Key1, Key2 combinations) that exist in the form set, as opposed to only a single set of keys specified in the TRNFILE.DAT file.

NOTE: This rule replaces the LoadRcpTbl and RunSetRcpTbl rules. You can replace those two rules with this one even if you are not using multiple lines of business (Key2s) in your document.

Syntax

```
;ProcessTriggers;;;
```

There are no parameters for this rule.

Place this rule after the BuildFormList rule in your AFGJOB.JDT file. Insert this rule after any import rule that might be used to create the starter document. For instance, insert the BuildFormList rule first, followed by your import rule, and then include the ProcessTriggers rule to add additional forms or assign recipient counts to the forms included via the import.

This rule does not trigger the Key2 (lines of business), however, if there are multiple lines of business defined at the point where triggering begins, this rule processes the triggers in each group. One way to define multiple groups is via an import file.

NOTE: The ProcessTriggers rule was added to support multiple lines of business during the triggering process. Normally, the RunSetRcpTbl file only supports the main (Key1+Key2) setting identified with the transaction. The ProcessTriggers rule, however, will process all defined lines of business (each Key1+Key2 combination) at trigger time. This means if you use an import rule to create the transaction, you can have multiple lines of business in batch processing and trigger additional forms and sections. You can use the ProcessTriggers rule with both Documaker Studio and the legacy tools.

If you are only using the Studio implementation model, you may want to use the RunTriggers rule.

See also

BuildFormList on page 72

LoadRcpTbl on page 157

RunSetRcpTbl on page 212

RunTriggers on page 213

JDT Rules Reference on page 30

PXCandidateList

Use this transaction-level rule (level 2) to build the form candidate list, based on the Form Candidate List DAL Trigger, and determine if the requested DAL script in the AFGJOB.JDT file should be evaluated.

The PXCandidateList rule acts as a functional equivalent for the PreTransDAL rule. The PXCandidateList rule executes the DAL script specified in the calling AFGJOB entry.

The DAL script then builds the State Loc table and returns control to the PXCandidateList rule. This rule next processes each of the State Loc table records against the FED table to build a Form Candidate list and a Consolidated Form State Loc table.

NOTE: The PXCandidateList and PXTrigger rules support Policy Xpress FED processing. These rules act as replacements for the PreTransDAL and DALTrigger rules when you are doing Policy Xpress FED-specific processing.

Syntax

```
PXCandidateList
```

Example

In this example, the rule calls a DAL script named Xpress_create():

```
;PXCandidateList;;Xpress_create();
```

The called DAL script then builds the State Loc and Form tables. Using these tables, the rule then builds a list of all possible forms for a given transaction. This form list is used by the PXTrigger rule to determine if the requested DAL trigger should be executed.

INI Options

You can use these INI options with this rule:

Option	Description
StateLocTable	(Optional) Enter the name of the state location table. The default is T_St_Loc.
FormLocTable	(Optional) Enter the name of the form location table. The default is T_FM_Loc.
FedTable	(Optional) Enter the name of the FED table. The default is T_F_List.
FedTable	(Optional) Enter the name of the FED table. The default is pxfed.
StatLocTableDFD	(Optional) Enter the name of the state location table DFD file. If you omit the path, the file is written to the deflib directory. The default is T_St_Loc.dfd.
FormTableDFD	(Optional) Enter the name of the form table DFD file. If you omit the path, the file is written to the deflib directory. The default is T_F_List.
FedTableDFD	(Optional) Enter the name of the FED table DFD file. If you omit the path, the file is written to the deflib directory. The default is pxfed.dfd.
CLDebug	(Optional) This option turns on debugging. The default is No.

Option	Description
CLDebugFile	(Optional) Enter the name of the debug file. If you omit the path, the file is written to the data directory. The default is cldebug.dat.
DumpCandList	(Optional) Enter Yes to have the system write the form candidate list details into the debug file. The default is No. To enable this option, you must also set the CLDebug option to Yes.
DumpFrmStTable	(Optional) Enter Yes to tell the system to write the Form-State Loc table records to the debug file. The default is No. To enable this option, you must also set the CLDebug option to Yes.
DumpFrmStLocEnt	(Optional) This tells the system to write the unloaded (by the PXTrigger rule) State Loc entries for the form. The default is No. To enable this option, set the CLDebug option to Yes.
DBTable:"Table"	(Optional) Enter the name of the table handler. The default table handler for all tables is MEM.
Fed_Processing	(Optional) Enter No to bypass FED processing. The default is Yes.

Form_List

Use this control group to enable form list processing.

Option	Description
Form_List	(Optional) Enter Yes to enable form list processing. The default is No.

XPTranslateLOB

The LOB (GROUP NAME 2) determines the type (Commercial Auto, Property Lines, Personal Lines) of FED processing. If you are using a non-standard LOB, use the XPTranslateLOB control group to translate the LOB to a standard LOB code. Here are the standard LOB codes:

Line of Business	Code
Commercial Auto	CA, CU, GL, CRIM, PR, BM, INMARC
Property Lines	BP, CP, CM
Personal Lines	HO, PP, DFire, DL, UMBRP, INMRP, PM

See also

PXTrigger on page 189

Input Tables on page 191

The Policy Xpress FED Processing Flow on page 192

PXTrigger

Use this transaction-level (level 2) rule to execute a DAL script if certain conditions are met. The PXTrigger rule replaces the DALTrigger rule. This rule is executed as part of RunSetRcpTable processing.

NOTE: The PXCandidateList and PXTrigger rules support Policy Xpress FED processing. These rules act as replacements for the PreTransDAL and DALTrigger rules when you are doing Policy Xpress FED-specific processing.

This rule does not call the requested DAL script unless the requested form is in the form candidate list. The rule performs a look up using the requested SetRecip entry form name against the Form Candidate List table.

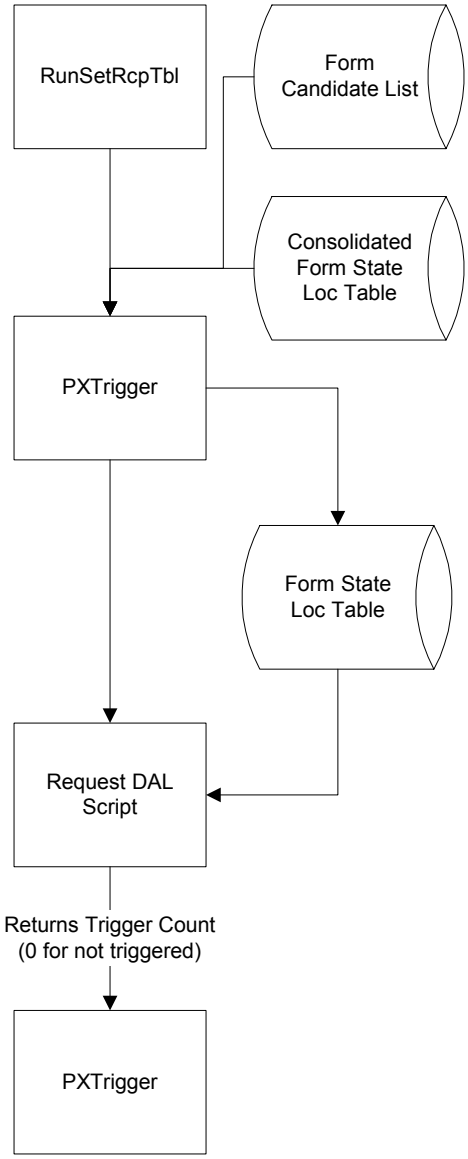
If the form is not found, the requested DAL script is not executed and the rule returns a trigger count of zero (0).

If the form is found, the rule unloads from the Consolidate Form State Loc table the StateLoc records into the FormStateLoc memory table based on the SetRecip entry form name.

Once the records are unloaded, the PXTrigger rule executes the requested DAL script. If the appropriate debug options are set, the FormStateLoc table and FormsList are unloaded to flat files.

Syntax

```
;DOCU;CA;CAINIT;;NBS;INSURED(1);;0;0;0;1;;PXTrigger;CAINIT;
```


RunSetRcpTbl
Form
Candidate List
Consolidated
Form State
Loc Table
PXTrigger
Form State
Loc Table
Request DAL
Script
Returns Trigger Count
(0 for not triggered)
PXTrigger

Input Tables

State loc table

Here are the required fields for the state location table.

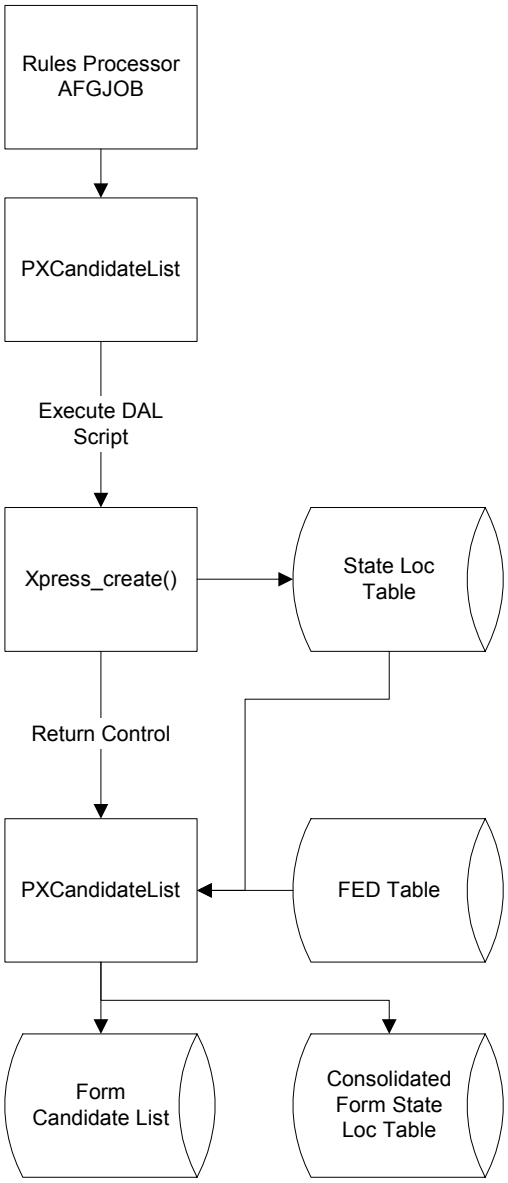
Field	Type	Length	Notes
Form	Char	36	
State	Char	8	
PolicyEffectiveDate	Char	10	CCYY-MM-DD
PolicyWrittenDate	Char	10	CCYY-MM-DD
ControllingState	Char	2	
Location	Char	8	
SubLocation	Char	8	
LocationAddrDate	Char	10	CCYY-MM-DD
TransactionEffectiveDate	Char	10	CCYY-MM-DD
ProgramCd	Char	6	

FED table

Here are the required fields for the FED table.

Field	Type	Length	Notes
FormNumber	Char	36	
FormProcessingType	Char	1	
EffectiveDateSourceCD	Char	2	“PE” or “PW”
EffectiveDate	Char	10	CCYY-MM-DD
WithdrawalDate	Char	10	CCYY-MM-DD

The Policy Xpress FED Processing Flow



FED processing

For each State Loc Record, a query is executed against the FED table using the LOB, State, and, if populated, the ProgramCd. Each of the returned FED records are then evaluated to determine if they are valid for the current State Loc table row.

FED record validation

For form list processing, the FED record must have a FormProcessingType of *R*.

The FED record validation date compares are based on the LOB groups defined above. If a LOB is not found in the standard groups (and is not translated to a standard LOB) the Commercial Auto compare is executed by default. The specific date validation/ compares are explained below.

For each FED record that passes validation, its FormNumber is added to the form candidate list (if the form does not already exist) and a row is inserted into the Form State Loc table (FormNumber plus the current State Loc Record).

Validation	Description
Commercial auto validation	If the Transaction Type is *PCH*, the date evaluation is executed using the State Loc record's TransactionEffectiveDate.If the Transaction Type is not equal to *PCH*, the date evaluation is based on the FED records EffectiveDateSourceCd value. If the EffectiveDate SourceCd is *PE,* the date evaluation is executed using the State Loc record's PolicyEffectiveDate. Otherwise, the date evaluation is executed using the State Loc Record's PolicyWrittenDate.
Personal lines validation	If the EffectiveDate SourceCd is *PE,* the date evaluation is executed using the State Loc record's PolicyEffectiveDate. Otherwise, the date evaluation is executed using the State Loc Record's PolicyWrittenDate.
Property lines validation	If the Transaction Type is *PCH*, the date evaluation is executed using the State Loc record's LocationAddrDate. If the transaction type is not equal to *PCH*, the date evaluation is based on the FED records EffectiveDateSourceCd value. If the EffectiveDate SourceCd is *PE*, the date evaluation is executed using the State Loc record's PolicyEffectiveDate. Otherwise, the date evaluation is executed using the State Loc Record's PolicyWrittenDate.
Date validation/ compare	The passed date is subjected to a trivial date validation. The date must be in a CCYY-MM-DD format. The day can not be zero (0) or greater than 31. The month can not be zero (0) or greater than 12. The year can not be zero (0). If the passed date passes validation, the system compares it to the Fed Entries Withdrawal Date and the Effective Date. If the passed date is falls before the Withdrawal Date and after the Effective Date, the date compare is passed.

See also PXCandidateList on page 187

RegionalDateProcess

Use this job-level (level 2) rule to execute regional date processing (RDP) rules on forms. You create the RDP rules via Documaker Studio.

NOTE: See the Documaker Studio User Guide for more information on creating RDP rules.

In the U.S. insurance industry, certain forms must comply with a regional authority (usually a state) to be approved for use within that area. The process of getting approval to use forms in each location is often referred to as *submitting for state compliance*.

Because of the various jurisdictions involved, you may have a form which is accepted by some states, but not by others. Alternatively, the form might be accepted by multiple states, but as of different dates. And to add another layer of complexity, states specify which document date must be used when activating this form.

To understand this last point, consider that almost all insurance policies have a date when coverage becomes *effective* – typically referred to as the *policy effective date*. Likewise, a policy usually has a *written* date that identifies when the document was actually drawn up. It is not unusual for the written date to be different from the policy effective date. For instance, you might buy your hurricane insurance today (the written date), but the policy does not become effective for 30 (or more) days. Each regional authority specifies which date determines the compliance of a given form.

This necessity to only activate the use of a form in a given region after a specific date complicates the creation of trigger conditions. Not only do you have to consider the typical transaction information that would cause you to include the form, you also have to calculate the various details to comply with the regional authorities described above.

To help you more easily manage this process, Studio lets you define regional date processing (RDP) rules that you can assign to each form. Part of the support is accomplished in Studio by defining the appropriate regional tests for each form. The remaining part occurs during the batch transactional process via the RegionalDateProcess rule.

Syntax

```
RegionalDateProcess;;;
```

There are no parameters for this rule.

Place this rule before any rule that calls the base triggering functionality, such as the RunSetRcpTbl rule. RDP rules operate as a filter that aids the normal triggering process by eliminating forms which do not meet the necessary criteria.

RDP rules are defined in group form files and assigned to individual forms. RDP rules are optional and only those forms containing one or more such rules are subject to this filtering process.

If a form has one or more RDP rules defined, these are evaluated during execution of this rule. The execution proceeds like this:

First, the system locates and evaluates the date search token associated with the rule. The resulting date value obtained from the transaction data is compared against the date range provided in the rule. If the date value is out of range, the form is flagged for possible elimination and execution moves to the next RDP rule.

If the date value is within the valid range, the next step is to iterate through the region search tokens associated with the rule and evaluate each. Each region search token might yield multiple hits. The system cross-references this list of values against the inclusion and exclusion list provided in the rule. If there is an intersection between the two lists, the rule is considered satisfied and the system moves to the next RPD rule. If the search token results in no matches for the list, then the system continues with the next search token until all search tokens are exhausted.

If the rule completes execution of the regional search tokens without finding any matches for the defined set, the form is flagged for possible elimination.

Each RDP rule executes in this manner which means that at the point where one rule considers the form eligible, no further RDP rules are execute on that form. Instead, RDP processing will immediately move to the next form.

If all RDP rules for a given form consider the form ineligible, then the form remains flagged with this state and is skipped during the subsequent triggering job rule process.

INI options

You can enter Yes for the RegionalDateProcess option to turn on debugging if you run into problems. Here is an example:

```
< Debug_Switches >
    RegionalDateProcess = Yes
```

Here's an example trace file produced by setting the RegionalDateProcess option to Yes:

```
1. Thu Sep 11 14:38:14.678 2008 pid=00029776 RDP Form <PREMIUM
CONFIRMATION> IS excluded.
2. Thu Sep 11 14:38:14.678 2008 pid=00029776 RDP Form <PREMIUM
CONFIRMATION 06152008> IS excluded.
3. Thu Sep 11 14:38:14.866 2008 pid=00029776 RDP Form <PREMIUM
CONFIRMATION> IS excluded.
4. Thu Sep 11 14:38:14.866 2008 pid=00029776 RDP Form <PREMIUM
CONFIRMATION 06152008> IS excluded.
5. Thu Sep 11 14:38:14.975 2008 pid=00029776 RDP Form <PREMIUM
CONFIRMATION> IS excluded.
6. Thu Sep 11 14:38:14.975 2008 pid=00029776 RDP Form <PREMIUM
CONFIRMATION 06152008> IS excluded.
7. Thu Sep 11 14:38:14.991 2008 pid=00029776 RDP Form <MEDICAL
HISTORY USING MEDBODY1> IS excluded.
```

Example Here is an example of how you would use this rule:

```
<Base Form Set Rules>
;NoGenTrnTransactionProc;2; single step;
;ResetOvFlw;2;;
;BuildFormList;2;;
;RegionalDateProcess;2;;
;RunSetRcpTbl;2;;
;WriteOutput;2;;
;WriteNaFile;2;;
;BatchingByRecipINI;2;;
;PaginateAndPropagate;2;;
```

See also Documaker Studio User Guide

ReplaceNoOpFunc

Use this job level rule (level 1) to register the MapFromImportData rule which the system will then use in place of the NoOpFunc rule.

Syntax

```
;ReplaceNoOpFunc;;;
```

There are no parameters for this rule. You typically use this rule with the following import rules:

- ImportFile
- ImportExtract
- ImportNAPOLFile
- ImportNAPOLExtract

NOTE: Use this rule if any of the DDT files for your sections are set to use the NoOpFunc rule. If you use the MapFromImportData rule instead of the NoOpFunc rule, you do not have to use this rule.

See also

MapFromImportData on page 376

NoOpFunc on page 415

ImportFile on page 116

ImportExtract on page 111

RULNestedOverFlowProc on page 203

ImportNAPOLExtract on page 121

ImportNAPOLFile on page 126

JDT Rules Reference on page 30

ResetDocSetNames

Use this form set level (level 2) rule to reset the pRPS structure after the GVM variables have been remapped.

Syntax

```
;ResetDocSetNames;;;
```

Normally, after it loads the transaction, the system uses the options in the DocSetNames and Trn_Fields control groups to map GVM variables into the pRPS structure for GroupName1, GroupName2, and TransactionID. There are other fields, but GroupName1, GroupName2, and TransactionID are the primary ones.

When you use the MergeWIP rule, or an rule that imports the document field information normally mapped by the transaction rule, you may need to use the GVM2GVM rule to map the options in the Trigger2WIP control group back to GVM variables. Because the names of the Key1 and Group1 fields sometimes differ, this means the mapping occurs too late to also be mapped to the pRPS structure member. Therefore, you must use the ResetDocSetNames rule to reset the pRPS structure after the GVM variables have been remapped.

When you use the EXT2GVM rule to get the values for GroupName1, GroupName2, and GroupName3, especially in an XML implementation, be sure to include the MapBeforeReset parameter to re-map the RPS structures. With the parameter, this rule gets the GroupName values from global memory and converts them into the long values using the Key1Table andKey2Table control groups. This is typically used to convert company codes to company names and so on.

Here is an example:

```
;UseXMLExtract;;;
;Ext2GVM;;!/Forms/Key1 1,10,Company;
;Ext2GVM;;!/Forms/Key2 1,15,LOB;
;ResetDocSetNames;;ConvertBeforeReset;
```

To avoid using this rule, make sure the primary keys are defined the same way in these DFD files:

- TRNDFDFL.DFD
- RCBDFDFL.DFD
- WIP.DFD

Example

Here is an example:

```
< Base Form Set Rules >
;WIPTransactions;;BATCHPRINT;
;GVM2GVM;;Trigger2Wip;
;ResetDocSetNames;;;
```

See also

Ext2GVM on page 93

GVM2GVM on page 107

MergeWIP on page 162

JDT Rules Reference on page 30

ResetOvFlw

Use this form set level rule (level 2) to reset the overflow feature.

Syntax

```
;ResetOvFlw;;;
```

This rule resets or reinitializes all of the overflow variables. In general, the overflow symbol will keep track of record counts as the extract is processed. When an overflow variable is defined, the system adds it to an overflow symbols list. This list contains several attributes for each symbol. This rule resets those attributes to the default values assigned when you initially defined the overflow symbol.

Example

```
;ResetOvFlw;;;
```

See also

InitOvFlw on page 144

IncOvSym on page 366

OvActPrint on page 417

OvPrint on page 419

WriteOutput on page 248

JDT Rules Reference on page 30

RestartJob rule

Use this job level (level 1) rule to open the restart file (RSTFILE) and reset the EXTRFILE, TRNFILE, NEWTRN, NAFILE, POLFILE, and batch files at the broken transaction. The RestartJob should be first base rule.

NOTE: If the restart file does not exist, the system skips this rule.

Syntax

```
;RestartJob;;;
```

You can set up the GenData program to restart itself at a particular transaction if it encounters a failure. To accomplish this, the system uses a restart file. You use INI options to set up the restart file.

NOTE: This rule does not apply if you are using single-step processing.

The restart file stores checkpoint information at specified intervals. If an error is encountered, the program resets itself and then checks each transaction until it isolates the transaction causing the error.

The restart file is removed at the end of a successful run. If the file exists at the start of a GenData run, the system assumes a restart is necessary and will open and read the file. The checkpoint information lets the system set internal pointers and output files in such a way that it can begin at that transaction.

You also use the RULCheckTransaction rule to restart the GenData program.

To use the restart feature, you should also set the following INI options:

```
< GenDataStopOn >
    BaseErrors         = Yes
    TransactionErrors  = Yes
    ImageErrors        = Yes
    FieldErrors        = Yes
```

Example

Here is an example:

```
;RestartJob;1;Always the first base rule;
```

See also

RULCheckTransaction on page 201

JDT Rules Reference on page 30

RULCheckTransaction

Use this form set level (level 2) rule to save the files necessary for restarting the GenData program. This rule should be the first base form set rule.

Syntax

```
;RULCheckTransaction;;;
```

The rule saves the EXTRFILE offset, TRNFILE offset, NEWTRN offset, NAFILE offset, POLFILE offset, and batch file offsets into a restart (RSTFILE) file. You can set up the GenData program to restart at a particular transaction if it encounters a failure. To accomplish this, the system uses a restart file. Use INI options to set up the restart file.

NOTE: This rule does not apply if you are using single-step processing.

The restart file stores checkpoint information at specified intervals. If an error is encountered, the program resets itself and then checks each transaction until it isolates the transaction causing the error.

The restart file is removed at the end of a successful run. If the file exists at the start of a GenData run, the system assumes a restart is necessary and will open and read the file. The checkpoint information lets the system set internal pointers and output files in such a way that it can begin at that transaction. You also use the RestartJob rule to restart the GenData program.

INI options

These offsets are updated in the post process after a specific number of transactions. You specify the number of transactions using the CheckCount option. You define the Restart file and the and check count in the Restart control group:

```
< Restart >
    RstFile    =
    CheckCount =
```

Option	Description
RstFile	Enter the name of the restart file. If you omit this option, the system uses RSTFILE.RST (DD:RSTFILE for z/OS) as the file name. The system uses the DataPath option in the Data control group to determine where to create the restart file. The default location is the current working directory.
CheckCount	Enter a number to specify the number of transactions to process before updating the offsets. For instance, if you specify two hundred (200), the system processes two hundred transactions, updates the offsets, processes two hundred more transactions, and so on. The default is 100. You can also use the */cnt* command line option with the GenData program to override the CheckCount option. Here is an example: `gendaw32 /cnt=10`

Example

Here is an example:

```
;RULCheckTransaction;2;Always the first form set rule;
```

See also

RestartJob rule on page 200

JDT Rules Reference on page 30

RULNestedOverFlowProc

Use this form level rule (level 2) to nest overflow within overflow. The nested overflow can occur on as many levels as necessary. This lets you use the system as a reporting tool. The only requirement is that the data occur in order.

This rule lets you create groups that contain group headers (lead sections), subordinate sections (list sections), and group footers (following sections).

Syntax

```
;RULNestedOverFlowProc;;;
```

Insert this rule in the AFGJOB.JDT file just after the RunSetRcpTbl rule.

To specify how the nesting occurs, you must create a file named OVERFLOW.DAT. You can create this file using any ASCII editor. By default, the system looks for this file in the DefLib directory, however, you can specify a different path and file name as the second parameter. Here is an example:

```
;RULNestedOverFlowProc;;c:\fap\dll\newfile.dat;
```

OVERFLOW.DAT file format

The file format for the OVERFLOW.DAT file is as follows:

```
;LeadIMG;LeadMask;ListIMGInfo;ListMask;FollIMGInfo;
```

Parameter	Description
LeadIMG	The group header inserted by normal triggering rules or by a previous call to the NestedOverflowProc rule. The group header is the section name without the extension of the lead section.
LeadMask	The search mask that originally triggered the lead section, in this syntax: `Offset,Record` *Offset* is the offset into the extract file. *Record* is the specific search key.
ListIMGInfo	Information about the subordinate section that overflows beneath the group header section. The format is the same as is in the FORM.DAT file: `SecName|SecAtts<Recip1(CpyCnt),Recip2(CpyCnt),…>` *SecName* is the name of the section without the extension. *SecAtts* are the attributes of the section (using the flags used in the FORM.DAT file such as D=data entry and print, S=same page, W=can grow, and so on. *RECIP1* is the recipient name. *CPYCNT* is the recipient copy count. Normally, you should set the attributes to *DS*. (*D*=data entry and print and *S*=same page). This information should not exceed 255 characters.
ListMask	(Optional) The search mask which triggers occurrences of the subordinate section beneath the group header section. Use this syntax: `Offset, Record` Where... *Offset* is the offset into the extract file. *Record* is the specific search key. If you omit this parameter, only one section is inserted.

Parameter	Description
FollIMGInfo	(Optional) Information about the group footer section which, if specified, is inserted after the last subordinate section for the current group. The format is the same as is in the FORM.DAT file: `SecName|SecAtts<Recip1(CpyCnt),Recip2(CpyCnt),…>` *SecName* is the name of the section without the extension. *SecAtts* are the attributes of the section (using the flags used in the FORM.DAT file such as D=data entry and print, S=same page, W=can grow, and so on. *RECIP1* is the recipient name. *CPYCNT* is the recipient copy count. Normally, you should set the attributes to *DS*. (*D*=data entry and print and *S*=same page). This information should not exceed 255 characters.

Before this rule is called, the lead section should already be in the form set—either through normal section triggering or by placing another call to this rule on a previous line.

The system begins by reading the OVERFLOW.DAT file line by line. The system finds the first occurrence of the lead section in the form set and then finds the first occurrence of the lead mask in the extract file.

It then counts the number of list masks between the first and second lead mask. Next, the system inserts the number of list sections after the first lead section and then inserts the follow section.

The system continues going through the form set inserting the number of list sections in between the *nth* and *(n+1)th* lead sections, according to the number of list masks in the extract file in between the (n)th and (n+1)th lead masks.

When it reaches the last lead section, the system counts the remaining list section masks after the last lead mask and inserts the appropriate number of list sections. Lastly, it inserts the *following section*, if specified.

NOTE: The *following section* and the *list masks* are optional. If there is no list mask, only one list section will be inserted.

Example

This example shows how to generate list sections subordinate to lead sections. In this example *Record1* is the lead mask, and *Record2* is the list mask.

LeadImage is the lead section and *ListImage* is the list section. The rule would count three *Record2s* between the two *Record1s*. If *Record1* is actually the fifth *Record1* in the extract file for the current transaction, the system would find the fifth *LeadImage* in the form set and insert three *ListImages* after this *LeadImage*.

Also, there is only one recipient, *Recip1*.

```
.
.
.
000000001RECORD1        data1
000000001RECORD2        data2
000000001RECORD2        data3
000000001RECORD3        data4
```

```
000000001RECORD4       data5
000000001RECORD4       data6
000000001RECORD2       data7
000000001RECORD1       data8
.   .   .   .
.   .   .   .
.   .   .   .
```

The example below shows the line in the overflow file. In the first example, the corresponding line in the overflow file for the previous example would appear as shown here:

```
;LEADIMAGE;10,RECORD1;LISTIMAGE|DS<RECIP1(1)>;10,RECORD2;;
```

The next example shows how to add a following section. If a following section was needed, the line from the file would look as shown here. Assume the following section is named *FOLLOWIMAGE:*

```
;LEADIMAGE;10,RECORD1;LISTIMAGE|DS<RECIP1(1)>;10,RECORD2;FOLLOWIMAG
E|DS<RECIP1(1)>;
```

This following example shows how to add another recipient. If a second recipient (with copy count of 3) was specified for the list section but not for the following section, the line would appear as shown here. Assume the second recipient is *Recip2*:

```
;LEADIMAGE;10,RECORD1;LISTIMAGE|DS<RECIP1(1),RECIP2(3)>;10,RECORD2;
FOLLOWIMAGE|DS<RECIP1(1)>;
```

This example shows how to change the section attributes. So far the section attributes were set as *DS—data entry and print* and *same page*. If the list section was dynamic, you must add the W (*can grow*) attribute. The line would now look as shown here:

```
;LEADIMAGE;10,RECORD1;LISTIMAGE|DSW<RECIP1(1),RECIP(2)>;10,RECORD2;
FOLLOWIMAGE|DS<RECIP1(1)>;
```

The example also shows nested/recursive functionality by iteration because this rule builds upon previous lines. This lets you have unlimited amounts of overflow within overflow.

Here are some points to consider concerning insertion logic:

- Reverse insertion logic when inserting different list sections after the same lead section.

  Because of the nature of insertions, if two lines in the overflow file use the same lead section, the list section in the second line is inserted before the list section on the first line. This requires a sort of reverse logic when creating the overflow.

- In order insertion below the lead section.

  Consider two different sections triggered by two different masks that occur in the extract file and the overflow file in the following manner:

```
(extract file)
000000001LEADREC
000000001LISTREC1
000000001LISTREC2
000000001LISTREC1
(overflow file: note reverse insertion logic)
;LEADIMAGE;10,LEADREC;LISTIMAGE2|DSW<RECIP(1);;
;LEADIMAGE;10,LEADREC;LISTIMAGE1|DSW<RECIP(1);;
```

The sections would be inserted after LEADIMAGE in the following order:

```
LISTIMAGE2…LISTIMAGE1…LISTIMAGE1
```

However the POLFILE.DAT file would look as follows:

```
…/LEADIMAGE|DSW<RECIP>/LISTIMAGE1|DSW<RECIP>/LISTIMAGE1|DSW
<RECIP>/LISTIMAGE2|DSW<RECIP>/...
```

This excerpt demonstrates that the sections are not inserted in the order of list search masks in the extract file but in the reverse order of the occurrence of list sections in the overflow file.

NOTE: If while processing the rule the system encounters invalid lines in the overflow file, it ignores those lines and adds log entries into the log file. The system then continues processing.

See also RunSetRcpTbl on page 212

JDT Rules Reference on page 30

RULStandardFieldProc

You must include this form set level rule (level 2) in the AFGJOB.JDT file as the *first* field rule. This rule tells the system to process each field on all of the sections triggered by the SETRCPTB.DAT file.

NOTE: This rule is used in multi-step processing. The StandardFieldProc rule is used in single-step processing.

Syntax

```
;RULStandardFieldProc;;;
```

There are no parameters for this rule.

Example

```
;RULStandardFieldProc;;;
```

See also Using the Job Definition Table on page 6

JDT Rules Reference on page 30

RULStandardImageProc

You must include this form set level rule (level 2) in the AFGJOB.JDT file as the *first* section rule. This rule tells the system to process each section triggered by the SETRCPTB.DAT file.

NOTE: This rule is used in multi-step processing. The StandardImageProc rule is used in single-step processing.

Syntax

```
;RULStandardImageProc;;;
```

There are no parameters for this rule.

Example

```
;RULStandardImageProc;;;
```

See also Using the Job Definition Table on page 6

JDT Rules Reference on page 30

RULStandardJobProc

You must include this rule as the first job level rule (level 1) in the AFGJOB.JDT file.

Syntax

```
;RULStandardJobProc;;;
```

There are no parameters for this rule.

Example

```
;RULStandardJobProc;;;
```

See also

ServerJobProc on page 217

JDT Rules Reference on page 30

RULStandardTransactionProc

You must include this form set level rule (level 2) in the AFGJOB.JDT file as the *first* transaction level rule. This rule tells the system to process each transaction listed in the extract file.

NOTE: Do not use this rule if you are using single-step processing. You only use this rule if you *are not* using the NoGenTrnTransactionProc rule.

Syntax

```
;RULStandardTransactionProc;;;
```

There are no parameters for this rule.

Example

```
;RULStandardTransactionProc;;;
```

See also

Single-Step Processing on page 7

NoGenTrnTransactionProc on page 168

JDT Rules Reference on page 30

RULTestTransaction

Use this form set debugging rule (level 2) before the RULStandardTransactionProc rule to have the system look for an INI group like the one shown below and execute only the transactions specified there.

Syntax

```
;RULTestTransaction;;;
```

The numbers shown below are sequence numbers, not transaction IDs. If you define the Test control group, the system will skip any transaction number not specified in the control group.

```
< Test >
    TransactionRecordNumber = 1        ; do transaction 1
    TransactionRecordNumber = 5        ; and transaction 5
```

Example

```
;RULTestTransaction;;;
```

See also

RULStandardTransactionProc on page 210

JDT Rules Reference on page 30

RunSetRcpTbl

Use this form set level rule (level 2) to run all entries in the set recipient table which pertain to the current GroupName1, GroupName2, and TransactionType to generate the form set for the current transaction.

NOTE: You can use the LoadFormsetFromArchive rule to replace the BuildFormList, LoadRcpTbl, and RunSetRcpTbl rules in the AFGJOB.JDT file.

For more information on setting recipients and copy counts, see the Documaker Server System Reference.

Syntax

```
;RunSetRcpTbl;;;
```

Place this rule in the AFGJOB.JDT file, after the BuildFormList rule.

Example

```
;BuildFormList;;;
;RunSetRcpTbl;;;
```

This example tells the system to process the SETRCPTB.DAT file to determine which forms and sections it should include in the POLFILE.DAT file for each recipient.

See also

BuildFormList on page 72

LoadFormsetFromArchive on page 154

LoadRcpTbl on page 157

Using the Job Definition Table on page 6

JDT Rules Reference on page 30

RunTriggers

Use this form set level (level 2) rule to replace the LoadRcpTbl and RunSetRcpTbl rules in implementations created by Documaker Studio.

Syntax

```
;RunTriggers;;;
```

There are no parameters for this rule.

NOTE: This rule is only available if you are running Documaker Studio. If you do not use Documaker Studio, see the discussion of the ProcessTriggers rule.

The RunTriggers rule represents an improved triggering process tailored for the file structure implemented with Documaker Studio. This process removes some of the complexity and improves performance.

For instance, in the old triggering process, if your form trigger asked for three copies of a form and then you processed the section triggers, only the last copy of the form was affected by the section triggers. The first two copies, would have been static based upon the default section configuration in the FORM.DAT file. Using the RunTriggers rule, each copy of the form will process section triggers. This means each form can differ from the prior one — assuming your section triggers check unique information to achieve that goal.

See also

LoadRcpTbl on page 157

RunSetRcpTbl on page 212

ProcessTriggers on page 186

JDT Rules Reference on page 30

RunUser

Use this job level (level 1) rule to execute a user function. You can use this rule to run a user function such as GetRecsUsed, IncRecsUsed, or ResetRecsUsed, to manipulate an overflow variable. This rule lets you manipulate an overflow variable without immediately making use of it in a field mapping.

Syntax

```
;RunUser;;;
```

Example

Overflow variables are variables you can use when a DDT rule is searching an extract record using a search mask. You can use the overflow variable to vary the occurrence of the record you are searching for.

Typically, the AFGJOB.JDT file contains this rule:

```
;ResetOvFlw;;;
```

which resets all overflow variables to zero at the beginning of each transaction. The RunUser rule lets you reset to zero at any time during field processing by running the ResetRecsUsed user function via the RunUser rule.

See also

ResetOvFlw on page 199

JDT Rules Reference on page 30

ServerFilterFormRecipient

Use this form set (level 2) rule with the DPSPrint object in IDS to generate a print file that contains a set of forms which will be filtered by

- Form name
- Form description
- Recipient name

Syntax

```
;ServerFilterFormRecipient;;;
```

The following attachment variables are created if these properties of the DPSPrint object are not null, in other words...

- DPSPrint.Forms creates DPRFORMNAME
- DPSPrint.FormDescription creates DPRFORMDESCRIPTION
- DPSPrint.Recipients creates DPRRECIPIENTNAME

The input can be a list, with items separated by commas. Here are the search conditions you can use:

- End with * , STARTS WITH
- Start with *, CONTAINS
- Text alone, EQUALS

While executing the DPSPRT request from the DPSPrint object, the RPDCreateJob rule creates the DPRFORMNAME, DPRFORMDESCRIPTION, and DPRRECIPIENT tags in the job ticket.

```
< ReqType:DPSPRT >
    function = atcw32->ATCLogTransaction
    function = atcw32->ATCLoadAttachment
    function = atcw32->ATCUnloadAttachment
    function = dprw32->DPRSetConfig
    function = dsijrule->JavaRunRule,;com/docucorp/ids/rules/
        dps;DPS;global;duplicateAttach;,RV,CUSTOMERBATCH,
        O,PRINTOUTPUTFILE,O
    function = rpdw32->RPDCheckRPRun
    function = rpdw32->RPDCreateJob
    function = rpdw32->RPDProcessJob
```

Once job ticket is created, Documaker processes the job. Documaker reads the job ticket and creates a GVM variable with this name:

```
DPRFORMNAME, DPRFORMDESCRIPTION, and DPRRECIPIENTNAME.
```

The ServerFilterFormRecipient rule looks for the GVM name DPRFORMNAME, DPRFORMDESCRIPTION, and DPRRECIPIENTNAME and filters out the mismatch condition.

Here is an example in Visual Basic:

```
Private Sub CmdPrint_Click()
    Dim oDPSVar As New DPSPrint
    Dim oDPSIDS As New DPSIDS
```

```
    oDPSVar.inputFile = FldInputFile.Text
    oDPSVar.configurationName = FldConfig.Text
    oDPSVar.outputFile = FldOutputFile.Text
    oDPSVar.outputPath = FldOutputPath.Text
    oDPSVar.printerType = FldPrinterType.Text
    oDPSVar.Forms = "ABC,DEF*,*XYZ"
    oDPSIDS.send oDPSVar
    FinalOutput.Text = oDPSVar.outputPath + oDPSVar.outputFile
End Sub
```

All form names that equal ABC or start with DEF or contain XYZ are included in the final print file.

If all inputs (DPRFORMNAME, DPRFORMDESCRIPTION and DPRRECIPIENTNAME) exist, the recipient name is evaluated first and the form name and form description are evaluated later.

NOTE: Include this rule in the AFGJOB.JDT file after the LoadRcpTbl rule.

Here is an excerpt from the AFGJOB.JDT file:

```
/* Every form set in this base uses these rules. */
<Base Form Set Rules>
;NoGenTrnTransactionProc;;required to combine gentrn/gendata into
single step;
;BuildFormList;;;
;LoadRcpTbl;;;
;ServerFilterFormRecipient;;;
;RunSetRcpTbl;;;
;PrintFormset;;required to combine gendata/genprint into single
step;
;WriteOutput;;;required to combine gentrn/gendata into single step;
;WriteNaFile;;;required to combine gentrn/gendata into single step;
;BatchingByPageCountINI;;;
;ProcessQueue;;PostPaginationQueue;
;PaginateAndPropagate;;FooterMode(2) Debug;
```

Example

```
;ServerFilterFormRecipient;;;
```

See also JDT Rules Reference on page 30

ServerJobProc

Use this job level rule (level 1) when you use the Internet Document Server (IDS) to run Documaker. This rule replaces the RULStandardJobProc rule.

NOTE: You must have a license to both IDS and Documaker to use this rule. For more information on setting up IDS and Documaker, see the Internet Document Server Guide.

Syntax

```
;ServerJobProc;;;
```

Insert this rule in the AFGJOB.JDT file as the first rule.

This rule looks for a job ticket file in the current working directory and loads it as an XML file. All of the values on the XML tree are added to or updated in the INI options. After Documaker finishes processing, the rule checks the status. If there are errors, it returns a *no more bases* return code on the next iteration. This terminates Documaker.

This rule uses a polling technique—sleep a while and check for the file existence— which you can configure using INI options. The rule loads the job ticket and sets INI options used when running subsequent rules. On the post message, this rule creates a job log XML tree and writes it to disk. If any necessary values are missing from the XML job ticket, these values are generated and changed (or appended) in the INI context.

On RP_PRE_PROC_B, this rule creates a semaphore (*gendata*), which makes it possible for the IDS RPDCheckRPRun rule to detect the status of Documaker when the next processing job starts.

This rule stays in waiting status and checks for the existence of job ticket file (JOBTICKET.XML) and the *rpdrunrp* semaphore. As soon as the job ticket file is detected, this rule loads it onto the XML tree and uses the contents of the XML tree to update INI options in memory.

If the rule does not detect the *rpdrunrp* semaphore, the rule terminates Documaker by returning a msgNO_MORE_BASES return code. It also creates a GVM variable (DSISERV) so the CUSInitPrint rule can re-initialize printers after the job process is complete. This GVM variable can be used by any of the Documaker rules to detect if the Documaker is running under IDS, if different logic is needed.

On RP_POST_PROC_B, the rule writes out the job log file and removes the job ticket file. If the RULServerJobPRoc option is set to Yes, a copy of the file will be obtained for debugging purposes.

INI options

```
< Data >
    DataPath =
    ExtrFile =
    MsgFile =
    ErrFile =
    LogFile =
    DBLogFile =
    NAFile =
    POLFile =
    NewTrn =
```

```
< PrinterInfo >
    Printer =
< Printer >
    Port =
< Print_Batches >
    Batch1 = batch1.bch
< IDSServer >
    SleepingTime =
    GENSemaphoreName =
    RPDSemaphoreName =
< Debug >
    RULServerJobProc =
< PrintFormSet >
    MultiFilePrint =
    LogFileType =
    LogFile =
```

Option	Description
Data control group	
DataPath	Used as the default path if you omit PrintPath.
ExtrFile	Enter the name and path of the extract file.
MsgFile	Enter the name and path of the message file.
ErrFile	Enter the name and path of the error file.
LogFile	Enter the name and path of the log file.
DBLogFile	Enter the name and path of the DB log file.
NAFile	Enter the name and path of the NA file.
POLFile	Enter the name and path of the POL file.
NewTrn	Enter the name and path of the NewTrn file.
PrinterInfo control group	
Printer	Enter the designated printers for print batches.
Printer control group	
Port	Enter the name of the print batch file for each designated printer. Note the group name is defined by the printer option in the PrinterInfo control group.
Print_Batches control group	
Batch1	Then name of the batch file.
IDSServer control group	
SleepingTime	Enter the amount of time in milliseconds you want the system to wait before it checks for a job ticket. The default is 1000 (1 second).

Option	Description
GENSemaphoreName	Enter the name of the semaphore. The default is *gendata*.
RPDSemaphoreName	Enter the name of the semaphore. The default is *rpdrunrp*.
Debug control group	
RULServerJobProc	Enter Yes to get a copy of the job ticket file before the system removes it.
PrintFormSet control group	
MultiFilePrint	Enter Yes to generate multiple print files which use 46-byte unique names. To identify which recipients are in which print batch, enter No or omit this option.This causes the PrintFormSet rule to save the printer for the print batch along with its recipient information. The ServerJobProc rule then adds three new tags for each print batch file and adds them to the JOBLOG.XML file. For example, for the print batch file on PRINTER1, the system creates these new tags: `<PRINTER1RECIP>Insured</PRINTER1RECIP>` `<PRINTER1CODE>001</PRINTER1CODE>` `<PRINTER1DESC>Insured Copy</PRINTER1DESC>` The MultiFilePrint option should only be used with the PDF, RTF, HTML, and XML print drivers.
LogFileType	Specify the type of print log file, such as XML or TEXT.
LogFile	Enter the name and path of the print log file. If you omit the extension, the system uses the LogFileType option to determine the extension.

Input file JOBTICKET.XML

Output file JOBLOG.XML

See also RULStandardJobProc on page 209

JDT Rules Reference on page 30

SetErrHdr

Use this job level rule (level 1) to define the header information used in the error file if an error occurs during the processing of a transaction. You can use any global variable (GVM) in the text defined to the system.

Syntax

```
;SetErrHdr;;Token:Text;
```

Parameter	Description
Token	A string of characters used to denote the beginning and end of a global variable name. Use a colon (:) to terminate the token string of characters. To substitute a global variable into the text, surround the name of the global variable with the token string. Be sure to use a string of unique characters for the token that are not defined in the text. For example, you could use '***'. You cannot use a colon (:) in the TOKEN string.
Text	A text string. The text string may include embedded tokens and global variables. Do not start the text string with a colon (:).

NOTE: Use a colon (:) to separate the token from the text. You must use a token even if there are no embedded global variables in the text string.

Example

```
;SetErrHdr;;***:        Transaction:  ***PolicyNum***;
;SetErrHdr;;***:Company Name:   ***Company***;
```

This example substitutes the global variables, *PolicyNum* and *Company*, into the error file header information. If the global variable *PolicyNum* was equal to *MVF10002* and *Company* was equal to *ABC Insurance Company,* the text output to the error file would be:

```
Transaction:   MVF10002
Company Name:   ABC Insurance Company
```

To add lines to the header, use the rule multiple times. Each time you use the rule, the system adds a line to the header, which you will see in the error file (ERRFILE.DAT).

NOTE: The global variable names must be spelled exactly as they are defined to the system.

See also JDT Rules Reference on page 30

SetOutputFromExtrFile

Use this form set level (level 2) rule to extract a print batch name from an extract file for each transaction. This capability is typically used with the MultiFilePrint callback function so you can get a print batch name from the extract file for one or more recipients per transaction.

Syntax

```
;SetOutputFromExtrFile;;RecordMask PrintBatchName;
```

Parameter	Description
RecordMask	This tells the system to locate a specific record line.
PrintBatchName	This tells the system to get the print batch name that begins in a specific location and save it to a global variable.

You must include the BatchingByRecipINI or the IfRecipUsed rule before this rule in the AFGJOB.JDT file. If you include the BatchingByRecipINI rule, also include these options:

```
< BatchingByRecip >
    Batch_Recip_Name = 39,FILENAME001;"Batch1";INVESTOR
    Batch_Recip_Name = 39,FILENAME002;"Batch2";COMPANY
    Batch_Recip_Name = 39,FILENAME003;"Batch3";AGENT
```

Make sure that FILENAME001,FILENAME002 and FILENAME003 exist at offset=39 in the records. You can see that the mask FILENAME001 is composed of FILENAME and Recipient code 001. So make sure INI control group is set.

To use multiple recipients, each transaction records should contain print multiple print batch names. Here is an example:

```
FUND/INVESTOR/TEMPLATE/TEMPLATE_LEVEL/INVESTOR_ID\JPMP0355 98PL X
FUND/INVESTOR/TEMPLATE/TEMPLATE_LEVEL/
FILENAME001\JPMP0355\JPMP035598PRP031501.pdf
FUND/INVESTOR/TEMPLATE/TEMPLATE_LEVEL/
FILENAME002\JPMP0356\JPMP035598PRP031502.pdf
FUND/INVESTOR/TEMPLATE/TEMPLATE_LEVEL/
FILENAME003\JPMP0357\JPMP035598PRP031503.pdf
```

The record mask should match the parameters for the SetOutputFromExtrFile rule and the INI options. The setup for this rule varies, depending on the mode you are running in.

3-step (GenTran, GenData, and GenPrint)

When processing using the GenTran, GenData, and GenPrint programs, you must set the INI options for the GenPrint program as shown here:

```
< Print >
    RCBDFDField    = PDFNAME
    CallBackFunc   = MultiFilePrint
    MultiFileLog   = ..\data\MFP.LOG
```

You can use the CUSMultiFilePrint function instead of the MultiFilePrint function, if you want to control the file name.

The CUSMultiFilePrint function is a print callback function that creates a new output file for each recipient and creates a log record of each. This is similar to the MultiFilePrint callback function in GenPrint except it gives you more control over the name of the file and supports long file names.

The system assumes you will use a built-in INI function to create a unique file name each time. This is important because the callback function cannot assign the first file name. You can use a DAL function to assign the first file name. Here is an example of the INI options:

```
< Printer1 >
    Port = ~DALRUN Batch1Files.dal
< Print >
    CallbackFunc = CUSMultiFilePrint
```

Here is an example of a DAL script:

```
#counter = #counter
file_name = "cusmultifileprint" & #counter & ".pdf"
#counter = #counter+1
return (file_name)
```

Please note that this function is used for multi-step processing only.

2-step (Single-step processing without the PrinfFormset rule and with GenPrint)

When you use single-step processing but omit the PrintFormset rule and instead use the GenPrint program, you must include the Print control group options:

```
< Print >
    RCBDFDField    = PDFNAME
    CallBackFunc   = MultiFilePrint
    MultiFileLog   = ..\data\MFP.LOG
```

Here is an example of the AFGJOB.JDT file you would use:

```
;NoGenTrnTransactionProc;2;required to combine GenTran/GenData;
;WriteRCBFiles;2;;
;ResetOvFlw;2;;
;BuildFormList;2;;
;LoadRcpTbl;2;;
;RunSetRcpTbl;2;;
;BatchingByRecipINI;2;;;
;SetOutputFromExtrFile;2;35,FILENAME 47,128,PDFNAME;
;WriteOutput;2;;
;WriteNaFile;2;;
;ProcessQueue;2;PostPaginationQueue;
;PaginateAndPropagate;2;;
```

Be sure to include the WriteOutput, WriteNAFile, and WriteRCBFiles rules.

Single-step (GenData only)

When you use single-step processing, the system does not use a callback function. Instead, it uses the MultiFilePrint INI option in the PrintFormset control group.

In single-step mode, you must have these INI options:

```
< PrintFormset >
    RCBDFDField    = PDFNAME
    MultiFilePrint = Yes
    LogFileType    = Text (or XML)
    LogFile        = mfp.log
```

NOTE: The MultiFilePrint option should only be used with the PDF, RTF, HTML, and XML print drivers.

Here is an example of the AFGJOB.JDT file you would use:

```
;NoGenTrnTransactionProc;2;required to combine GenTran/GenData;
;ResetOvFlw;2;;
;BuildFormList;2;;
;LoadRcpTbl;2;;
;RunSetRcpTbl;2;;
;SetOutputFromExtrFile;2;35,FILENAME 47,128,PDFNAME;
;WriteOutput;2;;
;WriteNaFile;2;;
;BatchingByRecipINI;2;;
;PrintFormset
```

Example

```
;SetOutputFromExtrFile;2;39,FILENAME 51,128,PDFNAME;
```

In this example, *39,FILENAME* is a record mask which tells the system to locate the record line that includes string *FILENAME* at offset 39. The text, *51,128,PDFNAME* tells the system to get the print batch file name at offset 51 in maximum length of 128 and save it to a global variable named *PDFNAME*. Note that *PDFNAME* must be defined in the RCBDFDFL.DFD file as a field.

See also

JDT Rules Reference on page 30

Single-Step Processing on page 7

BatchingByRecipINI on page 68

IfRecipUsed on page 108

WriteNAFile on page 247

WriteOutput on page 248

WriteRCBFiles on page 249

WriteRCBWithPageCount on page 250

SetOverflowPaperTray

Use this form set level rule (level 2), with the required INI option to change the printer tray selection during transaction processing.

This rule lets you print the first page of a form set on a special paper and the rest on different stock. For example, the first page of a utility bill is typically printed on perforated stock while the rest of the bill is printed on non-perforated stock.

Syntax

```
;SetOverflowPaperTray;;
```

There are no parameters for this rule.

The FormName option is required:

```
< OverflowPaperTray >
    FormName = Tray#
```

Option	Description
FormName	Enter the name of the form on the left and the tray you want used for the subsequent pages on the right. To specify the tray, you can only enter trays 1-9. See the example below.

Here is an example:

```
< OverflowPaperTray >
    Gas Bills   = Tray4
    Water Bills = Tray4
    Elec Bills  = Tray4
```

Keep in mind...

- You must include the PaginateAndPropagate rule in your AFGJOB.JDT file for the SetOverflowPaperTray rule to work correctly.
- When running in single- or two-step mode, place these rules in this order after the WriteOutput and WriteNAFile rules:.

  ```
  ; SetOverflowPaperTray;;
  ;PaginateAndPropagate;;
  ```

  When running multi-step mode, although UpdatePOLFile rule does pagination, you must still include the PaginateAndPropagate rule in your AFGJOB.JDT file, as pagination is needed before the SetOverflowPaperTray rule executes and before the NA and POL files are updated. Here is how the rules should be placed:

  ```
  ;UpdatePolfile;;
  ;SetOverflowPaperTray;;
  ;PaginateAndPropagate;;
  ```

- In duplex printing mode, the first page prints on stock from the original specified paper tray (for instance, containing perforated paper). If an overflow condition occurs and additional pages are printed; because you are duplexing, the first overflow page will print on the backside of the first page (on the perforated paper). The system redirects any additional overflow pages (pages 3, 4, and so on) to the paper tray you specify using the OverflowPaperTray control group.

- In simplex printing mode, the first page prints on stock from the original specified paper tray (for instance, pre-printed color letter head). If an overflow condition occurs, additional pages are printed on stock from the redirected paper tray you specify using the OverflowPaperTray control group.
- This rule *will not* work when printing in duplex mode and the first page is set as a back page.

Example

Let's assume your FORM.DAT file specifies duplex printing, tray 2 for all sections, and this INI option:

```
< OverflowPaperTray >
    UtilBill = tray4
```

This tells the system to use stock from tray 2 to print the first two pages (duplex) and stock from tray 4 for the subsequent pages 3 and 4 (duplex). Here are examples of the FORM.DAT and POLFILE.DAT files:

FORM.DAT file:

```
;RP10;CIS;UtilBill;Utility;N;;\
 billhdrp|FDLONX <Customer(1)>/\
 billsum |RDLS   <Customer(1)>/\
 billftr |RDLS   <Customer(1)>/\
 billrwtr|RDLS   <Customer(1)>/\
 billsere|RDLS   <Customer(1)>/\
 billstbr|RDLS   <Customer(1)>/\
 billrswr|RDLS   <Customer(1)>/\
 billcwt3|RDLS   <Customer(1)>/\
 blchart |RDLS   <Customer(1)>;
```

POLFILE.DAT file:

```
;RP10;CIS;UtilBill;Utility;R;;\
 billhdrp|FDLONX  <Customer>/\   page 1 - from Lower/tray 2 (L)
 billsum |RDLSN   <Customer>/\
 billftr |RDLSN   <Customer>/\
 billhdrp|RDLONX  <Customer>/\   page 2 - from Lower/tray 2 (L)
 billrwtr|RDLSN   <Customer>/\
 billsere|RDLS    <Customer>/\
 billstbr|RDLSN   <Customer>/\
 billhdrp|RD4ONX  <Customer>/\   page 3 - from tray 4 (4)
 billrswr|RDLSN   <Customer>/\
 billcwt3|RDLSN   <Customer>/\
 billhdrp|RD4ONX  <Customer>/\   page 4 - from tray 4 (4)
 blchart |RDLS    <Customer>;
\ENDDOCSET\ BillHead
```

Here is a simplex printing example:

Assume your FORM.DAT file specifies simplex printing, lower/tray 2 for all sections, and this INI control group:

```
< OverflowPaperTray >
    InsurBill = tray4
```

This tells the system to use the stock from tray 2 to print the first page and stock in tray 4 for the subsequent pages 2 and 3. Here are examples of the FORM.DAT and POLFILE.DAT files:

FORM.DAT file:

```
;RP10;CIS;InsurBill;NW Company;N;;\
 insuhdr |DLONX <Customer(1)>/\
 insuintr|DLS   <Customer(1)>/\
 insurate|DLOS  <Customer(1)>/\
 insusign|DLS   <Customer(1)>;
```

POLFILE.DAT file:

```
;RP10;CIS;InsurBill;NW Company;R;;\
 insuhdr |FDLONX <Customer>/\   page 1 - from Lower/tray 2 (L)
 insuintr|DLS    <Customer>/\
 insurate|DLOS   <Customer>/\
 insuhdr |FD4ONX <Customer>/\   page 2 - from tray 4 (4)
 insurate|DLOS   <Customer>/\
 insuhdr |FD4ONX <Customer>/\   page 3 - from tray 4 (4)
 insusign|DLS    <Customer>/\
\ENDDOCSET\ InsurHead
```

See also

PaginateAndPropagate on page 174

WriteNAFile on page 247

WriteOutput on page 248

JDT Rules Reference on page 30

SetOvFlwSym

Use this job level rule (level 1) to define an overflow variable for the overflow feature for use by the various field level rules using overflow. This rule adds the specified overflow variable to the overflow symbols list. You must use the InitOvFlw rule before you use this rule.

Syntax

```
;SetOvFlwSym;;OverflowSymbol, SectionName, MaxRecords;
```

Parameter	Description
OverflowSymbol	Name of the overflow symbol defined in the SetOvFlwSym rule.
SectionName	Name of the section that contains the fields on which overflow processing will occur.
MaxRecords	Defines the maximum number of overflow records to be processed for the section per page of output.

Example

```
;SetOvFlwSym;;Symbol,SectionName,10;
```

This example tells the system to define an overflow variable named *Symbol* for use with the section named *SectionName*, which has a maximum records per page of 10.

Another example of this rule is:

```
;SetOvFlwSym;;CGDECBDOVF,Q1GDBD,5;
```

This example tells the system to define an overflow variable named CGDECBDOVF for use with the form Q1GDBD, which has the maximum records per section set to five.

See also

WriteOutput on page 248

InitOvFlw on page 144

ResetOvFlw on page 199

IncOvSym on page 366

OvActPrint on page 417

OvPrint on page 419

SetOvFlwSym on page 227

JDT Rules Reference on page 30

SetRecipCopyCount

Use this form set level rule (level 2) in the AFGJOB.JDT file to set the number of copies for a particular recipient for all forms except those specified.

Syntax

```
;SetRecipCopyCount;;;
```

This rule includes these parameters:

```
;RULE;LEVEL;RECIPIENT,COPYCOUNT,FORM1,FORM2,...FORMn;
```

Where FORM1, FORM2, and FORMn are the names of the forms to exclude from the copy count you specify using this rule.

NOTE: This rule tests to see if the parameter value is a constant number. If not, it assumes the parameter names a GVM variable. It then uses the GVM variable to get the copy count.

Example

```
;SetRecipCopyCount;;IO,,DOC016,DOC018;
;SetRecipCopyCount;;I,3,DOCERR;
```

See also

SetRecipCopyCount2 on page 229

JDT Rules Reference on page 30

SetRecipCopyCount2

Use this form set level rule (level 2) in the AFGJOB.JDT file to set the copy count for a particular recipient for all forms specified.

NOTE: Version 10.1, Patch 109 changes the way the SetRecipCopyCount2 rule handles copy counts. Before this patch, the rule excluded the forms specified in the rule's parameters. With this patch and in subsequent versions, this rule includes those forms and changes the forms' copy counts.

Syntax:

```
;SetRecipCopyCount2;;(parameters);
```

Parameter	Description
Recipient	Enter the name of the recipient for whom the copy count will be set.
Number	Enter the number you want to set the copy count to.
Name	Enter the name of the forms for which the copy count should be set. If you have multiple forms, separate each name with a comma.

You must place this rule after the BuildFormList rule in the AFGJOB.JDT file.

NOTE: This rule tests to see if the parameter value is a constant number. If not, it assumes the parameter names a GVM variable. It then uses the GVM variable to get the copy count.

Example

```
;SetRecipCopyCount2;;Customer,2,QADesc1,QADesc2,QADesc3;
```

The copy count for recipient, *Customer*, is set to two (2) for forms: QADesc1, QADesc2, and QADesc3.

```
< Base Rules >
;RULStandardJobProc;;;
…
…
<Base Form Set Rules>
;RulStandardTransactionProc;;;
…
…
;BuildFormList;;;
;SetRecipCopyCount2;;CUSTOMER,2,Patch399;
…
…
```

See also SetRecipCopyCount on page 228

JDT Rules Reference on page 30

SortBatches

Use this job level (level 1) rule to sort RCB batches before they are printed. This rule provides a way for you to call an external sort program to rearrange the order of the recipient batch files (RCB files).

NOTE: The SortBatches rule is not available for z/OS implementations.

Syntax

```
;SortBatches;;;
```

The SortBatches rule provides two ways to sort batches:

- Single key

  This is the default sort for running under Windows. This sort command uses the Windows command line sort and builds an RCB file with a prepended sort key. Use this method if the external sort program uses a single sort field.

- Multiple keys

  Use this method when you need to create a sort command with a repeating pattern for each sort field.

Depending on the size of the recipient batch file, performance can be affected. The larger the input file, the slower the performance.

The SortBatches rule performs the required initial logic before the main job execution and executes the external sort program after execution. Place this rule immediately before the JobInit1 rule in your AFGJOB.JDT file, as shown here:

```
;RULStandardJobProc;;Always the first job level rule;
;SetErrHdr;;***:-------------------------------------------------;
...
;SortBatches;;;
;JobInit1;;;
```

Specifying Key fields

Define the key fields for the sort in SortBatches control group. Any field defined in the RCB DFD file can be used as a sort field. Each batch can have its own sort fields defined. You can also define a default sort ("SortDefault)". If you do not define a default sort, you must define a sort for each batch file written.

Here is the format of a SortBatches INI entry:

```
Batch Abbreviation = Field Name (A or D; Ascending or Descending)
```

Separate field references with semi-colons (;).

Here is an example:

```
< SortBatches >
    SortDefault = ACCOUNT_NUMBER(A);COMPANY(A);FEAT_DESCR (A)
    RegPrt = FEAT_DESCR(A);ACCOUNT_NUMBER(A)
```

In this example, the batch RegPrt will be sorted by CUSTOMER_NAME, FEAT_DESCR and ACCOUNT_NUMBER. All other batches will be sorted by COMPANY, FEAT_DESCR and ACCOUNT_NUMBER.

Sorting with a Single Key

To make it easier to set up and to support external sorts with only one key, a sort file with the with single key prepended is written and sorted by the external sort program when you use the BuildSortKey option. The batch file is written in the specified order without the prepended keys. The descending option (d) does not work with an external sort that does not support binary sorting.

Here is an example of how to set up your INI options for a single key sort. You specify the format of the external sort command using the options in the SortBatchOptions control group. This example calls the Windows command line sort:

```
< SortBatchOptions >
    BuildSortKey = Yes
    SortCommand = SORT **SourceFile** /t **WorkPath** /o
**TargetFile**
```

NOTE: The default for the BuildSortKey option is Yes on Windows and No on other platforms.

This is the default sort for running under Windows. "**SourceFile** /t **WorkPath** /o **TargetFile**" are replacement strings that are replaced with the appropriate values when the command line string is created. See Replacement Strings on page 233 for a complete list of available replacement strings.

These SortBatchOptions would produce the following sort command:

```
SORT .\data\REGPRT.tmp /t .\data\ /o .\data\REGPRT.wrk
```

Sorting with Multiple Keys

When you sort with multiple keys, the system does not use an interim file with a prepended key. Instead it writes a temporary batch file for input into the external sort. The SortCommand specified here calls a GNU Sort:

```
< SortBatchOptions >
    BuildSortKey = No
    SortCommand = sort -o **TargetFile**  *{[[ ]] -k **FieldOffset**,
**FieldLength** }* **SourceFile**
```

The data between the "*{" and "}*" (in bold) is replicated for each sort field specified in the sort batches entry. The data between the "[[" and "]]" is used as a field separators.

```
SortCommand = sort -o **TargetFile**  *{[[ ]] -k **FieldOffset**,
**FieldLength** }* **SourceFile**
```

FieldOffset and **FieldLength** are replacement strings you can use inside a repeating section. See Replacement Strings on page 233 for a complete list of available replacement strings.

Given the sample INI values defined above and the sample RCB DFD file definition, the generated sort command would appear as follows:

```
sort -o .\data\AGENT.wrk  -k 1,22 -k 23,4 -k 27,45 .\data\AGENT.tmp
```

Sorting with an OptTech Sort

OTSort by OptTech is a third-party sort utility. Here is an example of how you could set the SortCommand options to execute OTSort with the SortBatches rule:

```
< SortBatchOptions >
    BuildSortKey = No
    SortCommand = OTSW32D **SourceFile** **TargetFile** /
S(*{[[,]]**FieldOffset**,**FieldLength**,**FieldType**,**SortType**
}*)
```

INI Options

You can use these INI options with this rule:

```
< SortBatches >
    BatchFileName =
    SortDefault =
< SortBatchesOptions >
    BuildSortKey =
    SortCommand =
    LogSortCommand =
    KeepOrgFile =
    ZeroBasedOffsets =
< SortBatchSortTypes >
    a =
    b =
< SortBatchFieldTypes >
    Long =
    Char_Array =
```

Defining the sort

Use the options in the SortBatches control group to specify the name of the batch file and the fields you want to sort by.

Option	Description
BatchFileName	Enter the name of the batch file.
SortDefault	Enter the fields you want to sort by plus *A* for an ascending sort or *D* for a descending sort. The default is: `ACCOUNT_NUMBER(A);COMPANY(A);FEAT_DESCR (A)`

Sorting options

You specify all processing options for the SortBatches rule in the SortBatchOptions control group.

```
< SortBatchOptions >
    BuildSortKey       =
    LogSortCommand     =
    KeepOrgFile        =
    ZeroBasedOffsets   =
```

Option	Description
BuildSortKey	Enter Yes to specify single key processing. The default on Windows is Yes. The default on UNIX is No.

Option	Description
LogSortCommand	Enter Yes to send a copy of the sort command and associated sort options to the trace log file. The default is No.
KeepOrgFile	Enter Yes to write the original batch files in an unmodified format. The sorted batch files are written with an SRT extension. The default is No.
ZeroBasedOffsets	Enter Yes to use zero based offsets. The default is No.

Overriding the sort type

By default, the field-level sort type is written as *a* for ascending and *d* for descending. You can override these default values using the SortBatchSortTypes control group:

```
< SortBatchSortTypes >
    a = Replacement_Ascending_Type
    d = Replacement_Descending_Type
```

Overriding the field type

Field types are based on the internal field type defined in the RCB DFD (INT_TYPE). By default their types are set to *c* for character fields or *n* for numeric fields, but you can override these values. In the example below, fields defined as LONG have a field type of "num" and fields defined as CHAR_ARRAY have a field type of "char".

```
< SortBatchFieldTypes >
    Long       = num
    Char_Array = char
```

Replacement Strings

Here is a list of the non-repeating section replacement strings:

Replacement string	Description
TargetFile	The sort target file.
SourceFile	The source file name.
Key Length	The sort field length.
BeginOffset	The sort field begin offset.
EndOffset	The sort field end offset.
WorkPath	The location for temporary file (uses DataPath).

Here is a list of the repeating section replacement strings:

Replacement string	Description
FieldOffset	The field offset in the RCB file.
FieldLength	The field length.
FieldType	The field type (c or n based on INT_TYPE, values can be overridden).
SortType	The Sort type (a or d, values can be overridden).

RCB file layout

Here is the RCB file layout used in these examples:

```
< FIELDS >
   FIELDNAME = ACCOUNT_NUMBER
   FIELDNAME = FEAT_DESCR
   FIELDNAME = COMPANY
   FIELDNAME = APPLICATION
   FIELDNAME = CUSTOMER_NAME
   FIELDNAME = TRN_Offset
   FIELDNAME = X_Offset
   FIELDNAME = NA_Offset
   FIELDNAME = POL_Offset
   ...

< FIELD:COMPANY >
   INT_TYPE = CHAR_ARRAY
   INT_LENGTH = 5
   EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
   EXT_LENGTH = 4
   KEY = Y
   REQUIRED = Y

< FIELD:APPLICATION >
   INT_TYPE = CHAR_ARRAY
   INT_LENGTH = 4
   EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
   EXT_LENGTH = 3
   KEY = Y
   REQUIRED = Y

< FIELD:ACCOUNT_NUMBER >
   INT_TYPE = CHAR_ARRAY
   INT_LENGTH = 23
   EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
   EXT_LENGTH = 22
   KEY = Y
   REQUIRED = Y

< FIELD:FEAT_DESCR >
   INT_TYPE = CHAR_ARRAY
   INT_LENGTH = 46
   EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
   EXT_LENGTH = 45
   KEY = N
   REQUIRED = N

< FIELD:CUSTOMER_NAME >
   INT_TYPE = CHAR_ARRAY
   INT_LENGTH = 37
   EXT_TYPE = CHAR_ARRAY_NO_NULL_TERM
   EXT_LENGTH = 36
   KEY = N
   REQUIRED = N
...
```

See also JDT Rules Reference on page 30

StandardFieldProc

You *must* include this form set level rule (level 2) in the AFGJOB.JDT file if you are using the performance mode JDT. The StandardFieldProc rule should be the *first* field level rule in your AFGJOB.JDT file.

This rule tells the system to process each field on all of the sections triggered by the SETRCPTB.DAT file.

NOTE: If you use StandardFieldProc in your AFGJOB.JDT file, you must also include the WriteNAFile rule.

This rule is used in single-step processing. The RULStandardFieldProc rule is used in multi-step processing.

Syntax

```
;StandardFieldProc;;;
```

There are no parameters for this rule.

If the field has not yet been loaded, this rule loads it and then determines what type of field it is. If the field is not a text area, this rule sets the field data. If field is not a bar code or a text area, the rule sets field text.

If the field is a bar code, this rule validates and stores the bar code data. If there is an error with the data, the system writes a warning message to the error file, sends the transaction to the manual batch and continues the processing run.

Example

```
;StandardFieldProc;;;
```

See also

Single-Step Processing on page 7

WriteNAFile on page 247

JDT Rules Reference on page 30

StandardImageProc

This rule is a form set level rule (level 2) which you must include in the AFGJOB.JDT file. This rule is used when you are using the performance mode JDT and should be the first section level rule.

This rule tells the system to process each section triggered by the SETRCPTB.DAT file.

NOTE: This rule is used in single-step processing. The RULStandardImageProc rule is used in multi-step processing.

Syntax

```
;StandardImageProc;;;
```

There are no parameters for this rule.

This rule sets the next section. If there are no more sections, the system returns the message; *msg*NO_MORE_IMAGES.

If an error occurs, the system writes the following message in the error file and returns an error code:

```
Error in StandardImageProc(): Unable to SetNextImage(pRPS).
```

If it finds another section, the rule checks to see if the section has a corresponding DDT file and, if so, loads into memory the section and field rules included in that file.

Example

```
;StandardImageProc;;;
```

See also

Single-Step Processing on page 7

JDT Rules Reference on page 30

TicketJobProc

Use this job level (level 1) rule to run Documaker Server from another application by providing an XML job ticket. The results are returned in an XML job log file. The layout of these XML files is the same as those used by Docupresentment to run Documaker.

Specify the name of the job ticket to the GenData program on the command line using this parameter:

```
/jticket=
```

The default is JOBTICKET.XML. To prevent the job ticket file from being removed once the system finishes processing, include this INI option:

```
< Debug >
    TicketJobProc = Yes
```

Specify the name of the resulting job log file using this command line parameter:

```
/jlog=
```

The default is JOBLOG.XML.

Syntax

```
;TicketJobProc;;;
```

There are no parameters for this rule. You must include this rule as the first job level rule in the AFGJOB.JDT file. This rule replaces the RULStandardJobProc rule.

NOTE: Documaker must be set up in single step mode. Only the GenData program is executed.

For a single-transaction job process, you receive recipient information if you have this option set:

```
< PrintFormset >
    MultiFilePrint = No
```

Example

```
;TicketJobProc;;;
```

See also

RULStandardJobProc on page 209

JDT Rules Reference on page 30

TranslateErrors

Use this rule to extract error information from the message file and translate it into an error file using the message INI file, named TRANSLAT.INI, with group standard as the translation key. The system reads errors from the MSGFILE.DAT file, translates them, and places the translated error messages in the ERRFILE.DAT file.

NOTE: Using this rule can slow processing by creating additional tasks for the system to perform at the end of the processing cycle.

Syntax

```
;TranslateErrors;;;
```

Example

```
;TranslateErrors;;;
```

See also JDT Rules Reference on page 30

UpdatePOLFile

Use this form set level rule (level 2) to write the names of the forms to the POLFILE.DAT file. This list of forms, and the sections that comprise those forms, is sometimes called the POL set.

NOTE: Do not use this rule when you are doing 2-up printing on AFP printers. In that situation, you use the PaginateAndPropagate and WriteOutput rules instead.

Do not use this rule when you use the GenData program by itself to execute the GenTrn, GenData, and GenPrint steps.

Syntax

```
;UpdatePOLFile;;;
```

This rule writes the POL set to the POLFILE.DAT file, deletes the duplicate form set, updates the TRN file, and updates the recipients.

Example

```
;UpdatePOLFile;;;
```

See also Rules Used for 2-up Printing on page 27

JDT Rules Reference on page 30

UseXMLExtract

Use this form set level (level 2) rule when the extract list loaded by the transaction is also the source of the XML tree.

NOTE: The extract list and the XML tree are separate items. Even if you are only using an XML file as the source of the transaction, there will be two copies of the information in memory — one as the extract list and one as the XML tree.

If you are running Documaker from IDS, use the ImportXMLExtract rule to bring in XML in standard Documaker XML format, such as from Documaker workstation or iDocumaker. Use the UseXMLExtract rule to convert a loaded extract file into an XML tree, which you can then use to query data.

Syntax

```
;UseXMLExtract;;
```

Each XML transaction must begin with the XML declaration. The system assumes each transaction is a separate entity and requires that each transaction begin with an XML declaration.

Keep in mind that you will not be able to load this *appended file* as one large XML file. The system does not load the entire file before it processes the first transaction. Instead, it loads one transaction and then processes that transaction. Make sure there are line feeds between transactions. The line feed requirement is not an XML issue, but rather tells the system where one transaction ends and the next begins.

You place the rule in different locations in the AFGJOB.JDT file, depending on the mode in which you are running. For multi-step mode, place the XMLFileExtract rule after the LoadExtractData rule, as shown here:

```
;LoadExtractData;;
;UseXMLExtract;;;
```

For single or two-step mode, place the XMLFileExtract rule after the NoGenTrnTransactionProc rule, as shown here:

```
;NoGenTrnTransactionProc;;
;UseXMLExtract;;;
```

Remember that the system decides whether to search the extract list or the XML tree by checking to see if the search mask starts with an exclamation mark (!). The exclamation mark indicates that this is an XML path string. The system ignores the exclamation mark when it performs the XML path search.

To preserve the space when mapping data, use two exclamation marks (!!). Otherwise, the system assumes it should remove the leading white space.

Use these INI options with this rule:

- In the TRN_File control group, set the MaxExRecLen option to the optimal read size for your system. If you set this too large, it will consume too many resources. If you set it too small, it will perform too many reads. Check with your system administrator for guidance on setting this option.

```
< TRN_File >
    MaxExRecLen =
```

- In the Data control group, make sure the ExtrFile option points to the location of your XML file. Here is an example:

```
< Data >
    ExtrFile = .\extract\Sample.xml
```

- In the ExtractKeyField control group, set the SearchMask option as shown here:

```
< ExtractKeyField >
    SearchMask = 1,<?xml
```

- In the RunMode control group, include the XMLExtract option as shown here:

```
< RunMode >
    XMLExtract = Yes
```

- When running NoGenTrnTransactionProc (single or two-step mode) and the INI option is set to load the XML file, so there is no need to place XMLFileExtract rule in the AFGJOB.JDT file. Doing so makes the system load the XML file twice.

Mapping Fields

You can map TRN_Fields fields using the Ext2GVM rule or by using XPath.

Using Ext2GVM

When you use the Ext2GVM rule to get key information, make sure you only include the Key1, Key2, and KeyID options it the TRN_Fields control group. Set these options to dummy data, because the GVM variables are set to the data values during GenData processing.

To re-map these values from the global variables you get using Ext2GVM rule to the RPS structures (GroupName1, GroupName2, and GroupName3), include this rule in the AFGJOB.JDT file:

```
ResetDocsetNames;;ConvertBeforeReset;
```

The ConvertBeforeReset parameter gets the GroupName values from global memory and converts them to the long values using the Key1Table and Key2Table control groups.

Here is an example of the INI options:

```
< TRN_File >
    Company     = 3,3,N
    LOB         = 3,3,N
    PolicyNum   = 3,3,N
```

Here is an excerpt from the AFGJOB.JDT file:

```
;Ext2GVM;;!/Forms/Key1 1,10,Company;
;Ext2GVM;;!/Forms/Key2 1,15,LOB;
;Ext2GVM;;!/Forms/PolicyNum 1,12,PolicyNum;
;ResetDocsetNames;;ConvertBeforeReset ;
```

Using XPath

When you use XPath to map to the fields in the TRN_Fields control group, be sure to place an exclamation mark (!) in front of the XPath. Here is example:

```
< TRN_Fields >
    Company = !/Forms/Key1
    LOB = !/Forms/Key2
    PolicyNum = !/Forms/PolicyNum
    RunDate = !/Forms/RunDate;DM-4;D4
```

The format for the options in the TRN_Fields control group is:

```
(Field in the transaction DFD file) = XPath;Field Format
```

Although the exclamation mark (!) is not part of the actual search routine, the XML path search must begin with an exclamation mark. Do not specify whether a field is a key. The system does not support a multiple (search) keys with the XML implementation.

To selectively exclude transactions, place the XPath with a leading exclamation mark of what you want to exclude in your exclude file. Here is an example:

```
!/Forms[PolicyType="OLD"]
```

Overflow in XML

Here is how overflow works in XML. First, the system scans the search text to see if a replacement is needed for the overflow value. Here is one approach:

```
@GETRECSUSED,IMAGE1,STARS/!/Forms/Form/Car[****]/Driver
```

The system inserts the current overflow value, then performs the actual XML search for the requested XPath.

With the following approach, you can omit the use of @GETRECSUSED to declare which overflow variable to use and instead include the overflow name directly into the XPath, as shown here:

```
!/Forms/Form/Car[**INDEX**]/Driver
```

This method lets you support overflow within overflow.

Be aware that with either method, you still have to declare and use the overflow variables. The difference is that for the second method [**OverFlowSymbol**], the form name has to be *XML*, while for the first example [****], the form name is the actual name of the section for which you created the overflow symbol.

Also, remember to include the IncOvSym rules at the section level to increment the values to the next index. When doing overflow within overflow, you may also have to include an additional dummy section to do the IncOvSym for the symbol that represents the outer-most loop index.

See also

Ext2GVM on page 93

LoadExtractData on page 153

IncOvSym on page 366

XMLFileExtract on page 252

JDT Rules Reference on page 30

WIPFieldProc

Use this form set level rule (level 2) in place of the RULStandardFieldProc or StandardFieldProc rule in the AFGJOB.JDT file when you are using GenData WIP Transactions Processing. Using this rule tells the GenData program to bypass normal field processing.

NOTE: You cannot include in the AFGJOB.JDT file this rule and the RULStandardFieldProc or StandardFieldProc rule.

These other rules are also used when you run WIP Transaction Processing:

- WIPTransactions – This rule replaces the RULStandardTransactionProc or NoGenTrnTransactionProc rules in the AFGJOB.JDT file. This rule starts GenData WIP Transaction Processing at the form set level. It also identifies the status codes for the transactions in the WIP file that are processed. The status codes can be a subset or all of the status codes identified on the MergeWIP rule or none.
- GVM2GVM - This rule copies GenData execution data from the Trigger2WIP INI control group.
- WIPImageProc – This rule replaces the RULStandardImageProc or StandardImageProc rule.
- MergeWIP - This rule initializes GenData execution of WIP Transaction Processing at the job level. It creates a memory list and adds the transactions from the WIP file that match the status codes in its parameters.

Using these rules in a simplified AFGJOB.JDT file and with appropriate INI files, GenData WIP transaction processing adds the transactions from a WIP file to a transaction memory list. It then processes the transactions from the memory list, appending the data from the WIP file to the MRL recipient batch, NewTrn, NA, and POL files. If these files do no exist, it creates them. Each transaction in the memory list is deleted from the WIP file after it is processed.

Syntax

```
;WIPFieldProc;;;
```

There are no parameters for this rule.

Example

```
;WIPFieldProc;;;
```

See also

GVM2GVM on page 107

MergeWIP on page 162

WIPImageProc on page 244

WIPTransactions on page 245

GenData WIP Transaction Processing on page 9

JDT Rules Reference on page 30

WIPImageProc

Use this form set level rule (level 2) in place of the RULStandardImageProc or StandardImageProc rule in the AFGJOB.JDT file when you are using GenData WIP transactions processing. Using this rule tells the GenData program to bypass normal section processing.

NOTE: You cannot include in the AFGJOB.JDT file this rule and the RULStandardImageProc or StandardImageProc rule.

These other rules are also used when you run WIP Transaction Processing:

- WIPTransactions – This rule replaces the RULStandardTransactionProc or NoGenTrnTransactionProc rules in the AFGJOB.JDT file. This rule starts GenData WIP Transaction Processing at the form set level. It also identifies the status codes for the transactions in the WIP file that are processed. The status codes can be a subset or all of the status codes identified on the MergeWIP rule or none.
- GVM2GVM - This rule copies GenData execution data from the Trigger2WIP INI control group.
- MergeWIP - This rule initializes GenData execution of WIP Transaction Processing at the job level. It creates a memory list and adds the transactions from the WIP file that match the status codes in its parameters.
- WIPFieldProc – This rule replaces the RULStandardFieldProc or StandardFieldProc rule.

Using these rules in a simplified AFGJOB.JDT file and with appropriate INI files, GenData WIP Transaction Processing adds the transactions from a WIP file to a transaction memory list. It then processes the transactions from the memory list, appending the data from the WIP file to the MRL recipient batch, NewTrn, NA, and POL files. If these files do no exist, it creates them. Each transaction in the memory list is deleted from the WIP file after it is processed.

Syntax

```
;WIPImageProc;;;
```

There are no parameters for this rule.

Example

```
;WIPImageProc;;;
```

See also

GVM2GVM on page 107

MergeWIP on page 162

WIPFieldProc on page 243

WIPTransactions on page 245

GenData WIP Transaction Processing on page 9

JDT Rules Reference on page 30

WIPTransactions

Use this form set level (level 2) rule to process WIP transactions (manually approved or rejected in Documaker Workstation) in place of the RULStandardTransactionProc or NoGenTrnTransactionProc rules.

This rule processes WIP transactions and places them into batch files for GenPrint and GenArc processing. To specify which transactions, you use codes which you define in the Status_CD control group. If a WIP transaction has a status code that is not among the list of codes to be processed, the system deletes the transaction and WIP file.

The parameters for this rule are the status codes for which the user needs to process WIP transactions. The codes are located in the Status_CD control group. If you are using more than one status code, separate the codes with commas, as shown here:

```
Approved,Accepted
```

NOTE: Do not use this rule with the RULStandardTransactionProc or NoGenTrnTransactionProc rule.

The following rules are also used with this rule:

- MergeWIP - specifies the codes to look for
- GVM2GVM - copies the data from one GVM variable to another
- WIPImageProc - used in place of RULStandardImageProc or StandardImageProc
- WIPFieldProc - used in place of RULStandardFieldProc or StandardFieldProc

Using these rules in a simplified AFGJOB.JDT file and with INI options, you can input or merge WIP transactions (manually approved or rejected in Documaker Workstation) into a GenData processing run as new data or data appended to an existing GenData processed MRL (one that already has NEWTRN.DAT, NAFILE.DAT, and POLFILE.DAT files). These new or merged transactions can then be printed, archived, or both.

For instance, a typical use of these rules would is to take the results of a GenData run (NEWTRN.DAT, NAFILE.DAT, POLFILE.DAT, and print batch files) and process those files using the GenWIP program. You then open in the Documaker Workstation transactions sent to WIP and manually approve or reject them. Next, you run those WIP transactions (form sets) through the GenData process. The result is files ready for the GenPrint and GenArc programs.

Syntax

```
;WIPTransactions;;StatusCode1,StatusCode2,...;
```

Use the *StatusCode* parameters to define the status codes you want the system to use as it selects the WIP transactions to process.

After a transaction is processed, the system deletes it from the WIP list.

If you include a slash (/) before the StatusCode parameter, it tells the system not to delete the transactions with that status after it processes then, but instead assign them a new status. Here is an example:

```
;WIPTransactions;;APPROVED,FINAL,/PRINTED;
```

In this example, the slash (/) tells the system to process WIP transactions with an APPROVED or FINAL status and then change their status to PRINTED. The WIP transactions are not deleted.

Example

```
;WIPTransactions;;Approved;
```

Here is an example of how to define the codes in the Status_CD control group:

```
< Status_CD >
    Approved = AP
    Rejected = RJ
```

If the batch system does not need to process transactions with a certain code, such as *Rejected*, omit that code from the parameter list for this rule. When the system encounters a code not on the list, it deletes that transaction.

See also

GVM2GVM on page 107

MergeWIP on page 162

WIPFieldProc on page 243

WIPImageProc on page 244

GenData WIP Transaction Processing on page 9

Changing the WIP Status on page 165

JDT Rules Reference on page 30

WriteNAFile

Use this form set level rule (level 2) to append the NAFILE.DAT file data records for the current form set into an existing NAFILE.DAT file.

When you use the NoGenTrnTransactionProc rule, which replaces the RULStandardProc rule in the performance JDT, you must include the WriteNAFile rule to write the data (records) to the NAFILE during the GenData step.

In addition, you must also include the WriteOutput rule to write the data (records) to the POLFILE.DAT and NEWTRN.DAT files during the GenData step.

Syntax

```
;WriteNAFile;;;
```

There are no parameters for this rule.

Example

```
;WriteNAFile;;;
```

If an error occurs, the system returns this message:

```
Error in WriteNaFile: Unable to PurgeOutput(pRPS).
```

See also

WriteOutput on page 248

SetOutputFromExtrFile on page 221

NoGenTrnTransactionProc on page 168

Single-Step Processing on page 7

JDT Rules Reference on page 30

WriteOutput

Use this form set level (level 2) rule to create the POL file when executing single and two-step processing.

You also use this rule when you are using the GenData program by itself to execute the GenTrn, GenData, and GenPrint processing steps.

Syntax

```
;WriteOutput;;;
```

NOTE: If you use this rule, do not use the UpdatePOLFile rule.

Example

```
;WriteOutput;;;
```

See also

Rules Used for 2-up Printing on page 27

UpdatePOLFile on page 239

SetOutputFromExtrFile on page 221

Single-Step Processing on page 7

JDT Rules Reference on page 30

WriteRCBFiles

Use this form set level (level 2) rule to create the recipient batches when running in two-step mode. This mode is similar to single-step processing but omits the PrintFormset rule and instead uses the GenPrint program.

NOTE: Studio includes the WriteRCBFiles rule in the default AFGJOB.JDT file that Test manager produces.

Syntax

```
;WriteRCBFiles;;;
```

There are no parameters for this rule.

Example

```
;WriteRCBFiles;;;
```

Here is an example of a AFGJOB.JDT file you could use:

```
;NoGenTrnTransactionProc;2;required to combine GenTran/GenData;
;WriteRCBFiles;2;;
;ResetOvFlw;2;;
;BuildFormList;2;;
;LoadRcpTbl;2;;
;RunSetRcpTbl;2;;
;BatchingByRecipINI;2;;;
;SetOutputFromExtrFile;2;35,FILENAME 47,128,PDFNAME;
;WriteOutput;2;;
;WriteNaFile;2;;
;ProcessQueue;2;PostPaginationQueue;
;PaginateAndPropagate;2;;
```

Be sure to include the WriteOutput and WriteNAFile rules.

See also

SetOutputFromExtrFile on page 221

WriteNAFile on page 247

WriteOutput on page 248

JDT Rules Reference on page 30

WriteRCBWithPageCount

Use this form set level rule (level 2) to write the page count for each recipient. This rule is typically use for handling 2-up printing on AFP and compatible printers. This rule is also used for multi-mail processing.

Syntax

```
;WriteRCBWithPageCount;;;
```

There are no parameters for this rule.

You must include the following data in your RCBDFDFL.DFD file when you use this rule:

```
< Fields >
    ....
    ....
    FieldName = CurPage
    FieldName = TotPage
    FieldName = AccumPage
    ......
    ......
< FIELD:CurPage >
    INT_Type = LONG
    EXT_Type = CHAR_ARRAY_NO_NULL_TERM
    EXT_Length = 10
    Key = N
    Required = N
< FIELD:TotPage >
    INT_Type = LONG
    EXT_Type = CHAR_ARRAY_NO_NULL_TERM
    EXT_Length = 10
    Key = N
    Required = N
< FIELD:AccumPage >
    INT_Type = LONG
    EXT_Type = CHAR_ARRAY_NO_NULL_TERM
    EXT_Length = 10
    Key = N
    Required = N
```

Example

```
;WriteRCBWithPageCount;;;
```

This rule gets a pointer to the global variable NA_Offset. If the pointer is NULL, the system returns this message to the error file:

```
Error in WriteRCBWithPageCount: RCB field NA_Offset not found.
```

The rule then assigns the pointer it retrieved to the pRPS->POL_Offset and gets a handle to the print batch list. It then loops through this list to free the contents of the corresponding PRINT_BATCH structure if that print batch has been assigned a recipient.

The rule then checks the handle to the counter list. This handle is a global handle created by the PageBatchStage1InitTerm rule. If the handle is equal to VMMNULLHANDLE, the system returns this message to the error file:

```
Error in WriteRCBWithPageCount: PageBatchStage1InitTerm has not been
called
```

The rule then checks the handle to the list of transaction records for the current batch. This handle is a global handle also created by the PageBatchStage1InitTerm rule. If this handle is equal to VMMNULLHANDLE, the system returns this message to the error file:

```
Error in WriteRCBWithPageCount: PageBatchStage1InitTerm has not been
called
```

The rule then sets recipient page counts. If an error occurs, the system returns this message to the error file:

```
Error in WriteRCBWithPageCount:Failed to set recipient page counts
```

Next, the rule gets the handle of the global recipient list for the current form set and loops through the recipient list to add page counts to the GVM. The rule also gets the pointer to global variable TotPage. If this pointer is NULL, the system returns this message to the error file:

```
Error in WriteRCBWithPageCount: RCB field TotPage not found
```

Then rule then gets the handle to the print batch and recipient lists and loops through these lists. Finally, the rule loops through the list of page counts. If it cannot get the handle to the given page count, the system returns this message to the error file:

```
Error in WriteRCBWithPageCount: Cannot locate batch <Batch Name>.
```

See also

Rules Used for 2-up Printing on page 27

BatchByPageCount on page 47

BatchingByPageCountINI on page 49

PageBatchStage1InitTerm on page 173

SetOutputFromExtrFile on page 221

JDT Rules Reference on page 30

XMLFileExtract

Use this form set level (level 2) rule when the extract list loaded by the transaction contains the name of an external file source for the XML tree.

NOTE: The extract list and the XML tree are separate items. Even if you are only using an XML file as the source of the transaction, there will be two copies of the information in memory — one as the extract list and one as the XML tree.

Syntax

```
;XMLFileExtract;;parameter;
```

For the parameter, you can use one of the following:

- FILE=<filename>

  Where *filename* is the name of the XML file, including path information.

- INI=group,option

  Where *group* and *option* are defined in the INI files

- SCH=offsetofmask,<searchmask> offsetofdata,lengthofdata

Parameter	Description
offsetofmask	the offset of the search mask
offsetofdata	the offset where the path and the file name start
lengthofdata	the length of the file name

- GVM=<globalvariablename>

  Where *globalvariablename* is the name of a GVM that contains the file name.

Here are examples of how you can use this rule:

```
;XMLFileExtract;2;FILE=SAMPCO.XML;
;XMLFileExtract;2;INI=Group,Option;
;XMLFileExtract;2;SCH=11,FILENAME 20,20;
;XMLFileExtract;2;GVM=FileNameVar;
```

Keep in mind...

- Begin each XML transaction with the XML declaration.
- In the RunMode control group, set the XMLExtract option as shown here:

```
< RunMode >
    XMLExtract = Yes
```

- You place the rule in different locations in the AFGJOB.JDT file, depending on the mode in which you are running. For multi-step mode, place the XMLFileExtract rule after the LoadExtractData rule, as shown here:

```
;LoadExtractData;;
;XMLFileExtract;2;FILE=SAMPCO.XML;
```

For single or two-step mode, place the XMLFileExtract rule after the NoGenTrnTransactionProc rule, as shown here:

```
;NoGenTrnTransactionProc;;
;XMLFileExtract;2;FILE=SAMPCO.XML;
```

- Remember that the system decides whether to search the extract list or the XML tree by checking to see if the search mask starts with an exclamation mark (!). The exclamation mark indicates that this is an XML path string. The system ignores the exclamation mark when it performs the XML path search.

  To preserve the space when mapping data, use two exclamation marks (!!). Otherwise, the system assumes it should remove the leading white space.

- When running NoGenTrnTransactionProc (single or two-step mode) and the INI option is set to load the XML file, so there is no need to place XMLFileExtract rule in the AFGJOB.JDT file. Doing so makes the system load the XML file twice.

Mapping Fields

You can map the fields listed in the TRN_Fields control group using either offset/length, XPath, or a combination of both methods. In the RunMode control group, be sure to set these INI options:

Option	Description
XMLExtract	Enter Yes to tell the system you are using the XML file.
XMLFileExtract	Enter Yes to tell the system your extract file contains a list of pointers pointing to the XML file to be processed.
XMLFileExtractName	Use this option to tell the system how to find your XML file. Enter the method you use to point to your XML file. This should be exactly the same as how you would set up the rule parameter for the XMLFileExtract rule in your AFGJOB.JDTfile.

Here is an example:

```
< RunMode >
    XMLExtract         = Yes
    XMLFileExtract     = Yes
    XMLFileExtractName = SCH=1,XML 20,60
```

Also set the TRN_Fields options as shown in this example:

```
< TRN_Fields >
    Company     = !/Forms/Key1
    PolicyNum   = !/Forms/PolicyNum
    RunDate     = !/Forms/RunDate;DM-4;D4
    LOB         = 30,15,N
    Cust_Name   = 46,30,N
```

The format for the options IN the TRN_Fields control group is:

(Field in the transaction DFD file) = XPath;Field Format

(Field in the transaction DFD file) = offset, length, Key;Field Format

An XML path search must begin with an exclamation mark (!). The exclamation mark is not part of the actual search routine. Do not specify whether a field is a key. The system does not support a multiple (search) keys with the XML implementation.

To selectively exclude transactions, use either an offset/SearchMask, the XPath, or a combination of the two in your exclude file. Here is an example:

```
!/Forms[PolicyType="OLD"]
20,ABC
```

Overflow in XML

Here is how overflow works in XML. First, the system scans the search text to see if a replacement is needed for the overflow value. Here is one approach:

```
@GETRECSUSED,IMAGE1,STARS/!/Forms/Form/Car[****]/Driver
```

The system inserts the current overflow value, then performs the actual XML search for the requested XPath.

With the following approach, you can omit the use of @GETRECSUSED to declare which overflow variable to use and instead include the overflow name directly into the XPath, as shown here:

```
!/Forms/Form/Car[**INDEX**]/Driver
```

This method lets you support overflow within overflow.

Be aware that with either method, you still have to declare and use the overflow variables. The difference is that for the second method [**OverFlowSymbol**], the form name has to be *XML*, while for the first example [****], the form name is the actual name of the section for which you created the overflow symbol.

Also, remember to include the IncOvSym rules at the section level to increment the values to the next index. When doing overflow within overflow, you may also have to include an additional dummy section to do the IncOvSym for the symbol that represents the outer-most loop index.

See also LoadExtractData on page 153

IncOvSym on page 366

UseXMLExtract on page 240

JDT Rules Reference on page 30

Chapter 4

Adding Section and Field Rules

This chapter discusses adding section and field level rules. These rules link the section's variable fields to external data.

NOTE: You create variable fields using Documaker Studio or the Image Editor. For more information, see the Documaker Studio User Guide or the Docucreate User Guide.

The section and field level rules are executed during data generation and merger procedures. This occurs in Documaker Server.

In this chapter you will find information about:

For reference information on individual rules, see Chapter 5, Section and Field Rules Reference on page 274.

Storing Rule Information

Documaker Studio stores sections (images) in a FAP file, along with the section and field rule assignments you assign to it. This differs from the way rule information is stored when using the older document creation tool. Image Editor.

Image Editor stores sections in a FAP file which only contain the section's objects and object attributes. The Image Editor stores section and field rule assignments in a separate file, called a *data definition table* (DDT) file. While DDT files originally offered high performance, advanced formatting needs made it necessary for the FAP files to be available at runtime to handle dynamic composition. This made the DDT file approach less of an advantage, and even a stumbling block within some implementations.

With the release of version 11.0 and the introduction of Documaker Studio's FOR file, section and field-level rules previously stored in the DDT file are, in Studio implementations, either unnecessary or are stored in the FAP file. Having section level rules (such as SetOrigin) in the FOR file makes it easier to do visual form design. Having field level rules in the FAP file eliminates synchronization worries.

NOTE: For more information on Image Editor, see Using Image Editor to Enter Rule Information on page 503.

Formatting Data

The system provides several ways to format dates and numbers using the FmtDate, RunDate, SysDate, DateFmt, and FmtNum rules. The system includes several pre-defined formats from which you can choose and you can set up *format arguments* to handle any special needs.

The following topics explain your options.

NOTE: The DateFmt rule accepts a mask which includes an input and an output format. See DateFmt on page 324, for more information.

Using Pre-defined Date Formats

In this example...

```
d,"1/4",
```

…the *d* indicates it is a date format, as opposed to a number format (n). The first digit (1) indicates the date format (MM/DD/YY). The forward slash (/) indicates the separator character (/) and the third digit (4) indicates the number of digits in the year. See DateFmt on page 324 for the complete list of date formats.

NOTE: Because of year 2000 considerations, use four-digit years.

In cases where you do not need a separator, such as format 4 or B, you can specify the date as *"4/2"* for format 4 with a two-digit year. The system ignores the slash (/).

NOTE: This example shows the date format as it looks in the FAP file. The easiest way to enter date formats is through the Image Editor, on the Attributes tab of the variable field's Properties window. The Image Editor will then create the date format in the FAP file for you. The following discussion is based on using the Image Editor to select the date format.

When you choose Date Format as the type, you can choose from this list of date formats in the Image Editor on the Attributes tab of the field's Properties window. The table also shows the corresponding date format code the Image Editor creates in the FAP file:

In the Format field, select this format	To see this code in the FAP file	To get dates formatted as shown below (all examples are for January 2, 2013)
MM/DD/YY	1	01/02/13 (default)
DD/MM/YY	2	02/01/13
YY/MM/DD	3	13/01/02
Month D, Yr	4	January 2, 2013
M/D/YY	5	1/2/13
D/M/YY	6	2/1/13
YY/M/D	7	13/1/2
bM/bD/YY	8	1/ 2/13 (space before 1/ and 2/)
bD/bM/YY	9	2/ 1/13 (space before 2/ and 1/)
YY/bM/bD	A	13/ 1/ 2
MMDDYY	B	010213
DDMMYY	C	020113
YYMMDD	D	130102
MonDDYY	E	Jan0213
DDMonYY	F	02Jan13
YYMonDD	G	13Jan02
DAY/YY	H	002/13
YY/DAY	I	13/002
D Month, Yr	J	02 January, 2013
Yr, Month D	K	2013, January 02
Mon-DD-YYYY	L	Jan-02-2013
DD-Mon-YYYY	M	02-Jan-2013

In the Format field, select this format	To see this code in the FAP file	To get dates formatted as shown below (all examples are for January 2, 2013)
YYYY-Mon-DD	N	2013-Jan-02
Mon DD, YYYY	O	Jan 02, 2013
DD Mon, YYYY	P	02 Jan, 2013
YYYY, Mon DD	Q	2013, Jan 02
(hexadecimal)	X	Eight-character hexadecimal representation of the system date. Valid dates range from 12/31/1969 to 01/18/2038. Valid dates may differ depending on the type of machine (PC or host) and the type of CPU chip.

These date formats affect processing in Documaker Workstation, not Documaker Server.

Here is a list of the separators you can choose from in the Separators field on the Attributes tab of the variable field's Properties window.

In the Separator field, choose...	To use this character as the separator...
00/00/00 (default)	/ (a slash appears in the FAP file)
00-00-00	- (a dash appears in the FAP file)
00.00.00	. (a period appears in the FAP file)
00,00,00	, (a comma appears in the FAP file)
00 00 00	blank (a "b" appears in the FAP file)

In the Year Size field on the Attributes tab of the variable field's Properties window, you can choose from these options...

To use...	Select...
a two-digit year such as 01/01/13 (use only if the year is in current century)	2 (a "2" appears in the FAP file)
only a two-digit year such as 01/01/13 (if you enter anything other than a two-digit year, you will receive an error)	3 (a "3" appears in the FAP file)
a four-digit year such as 01/01/2013	4 (a "4" appears in the FAP file)

only a four-digit year such as 01/01/2013 (if you enter anything other than a four-digit year, you will receive an error)	5 (a "5" appears in the FAP file)
The year as entered without changing it	Default (a blank space appears in the FAP file)

NOTE: You can force 2-digit years when you use the FmtDate rule, even if doing so means the date may not be interpreted correctly when it is compared to the century cut-off date. To force a 2-digit year, you must specify the output format as "1/3" instead of "1/2". Here is an example:

```
;0;0;DRVR-BIRTH-DT;1022;8;DRVR-BIRTH-DT;0;8;d,"B4",d,"1/3";
FmtDate;5,DRVRREC01,;N;N;N;N;3367;3600;11011;
```

The 3 is a format mask (normally used for input) which means 2-digits and only 2-digits.

NOTE: The century cut-off date is used to determine the century for 2-digit years. This date defaults to 50, but you can change it using this INI option:

```
< Control >
    DateFMT2To4Year =
```

Anything less than or equal to the cut-off year is considered to fall in the current century. For instance using the default of 50, *13* would be interpreted as *2013*. Anything greater than the cut-off year is considered to fall in the previous century. For instance, again using the default of 50, *88* would be interpreted as *1988*.

This is important when you have to determine the years or days between two dates.

There is a scenario where the system overrides a 2-digit year output. This only happens when the input has 4-digits and the output has 2-digits and the resulting 2-digit output does not yield the same results when read in again.

For instance, suppose your input is 01/01/1927 and the cutoff year is 50. Normally any 2-digit year with a value less than 50 is considered part of the current century. So if the system outputs the data as 01/01/27 and then tries to read this date back in, you would get 01/01/2025 and not 01/01/1927.

The system changes its normal behavior because it is designed to be able to read its own output and come up with the result originally provided in the original input.

If, however, you specifically tell the system you only want two digits, you will get that output, but the system may not be able to read it back in and get the same results.

Using Pre-defined Numeric Formats

For numbers, you can use these format masks:

To...	Use...
place a number in that space (0-9)	9
place any number except a zero	Z
indicate the number is an amount	$
place a currency symbol to the right of the amount (the second $indicates which symbol)	$$
place a minus sign (-) beside the amount if it is negative	-
place a minus (-) or plus (+) sign beside the amount	+
indicate a credit (accounting format)	CR
indicate a debit (accounting format)	DB
indicate a debit (accounting format)	()
include an asterisk ($*999)	*
place a percent sign (%) after the number	%

You determine whether the minus (-) or plus (+) signs appear before or after the amount when you choose the field's format on the Attributes tab of the Properties window in the Image Editor.

When you choose the format in the Image Editor, the system lets you choose from a list of examples, such as:

```
+$ZZZZZZZZZ9.99
$$ZZ,ZZZ,ZZZ,ZZZ
$ZZZZZZZZZ9.99CR
$*ZZZZZZZZZZZ.ZZ
```

Suppressing Decimals with the FmtNum Rule

The FmtNum rule can use a pre-defined numeric format to suppress decimals. The format is 0 (zero). You can only use this format after the decimal and at the end of the value. You cannot place format code 9 or Z after you specify the zero (0) format code.

Here are some examples of how the Z format and the zero (0) format work together.

Format Z,ZZZ.0Z	Format: Z,ZZZ.00	Format Z,ZZZ.Z0
Input = 9999.00	Input = 9999.00	Input = 9999.00
Output = 9,999	Output = 9,999	Output = 9,999

Format Z,ZZZ.0Z	Format: Z,ZZZ.00	Format Z,ZZZ.Z0
Input = 9999.90	Input = 9999.90	Input = 9999.9
Output = 9,999.90	Output = 9,999.9	Output = 9,999.9
Input = 9999.09	Input = 9999.09	Input = 9999.09
Output = 9,999.09	Output = 9,999.09	Output = 9,999.09

If you have decimals and you want to see two decimal places, you should use the first format style shown (ZZZ.0Z). With this format style, an input value of 1.1 will yield 1.10.

If you use ZZZ.00 and input 1.1, you will get 1.1. In most cases, the values using format Z will be displayed. However, if you are using Z,ZZZ.0Z format, Z will be suppressed if the Z value contains zero and if the value next to the decimal is suppressed.

The values using the zero (0) format will be displayed unless it is zero. However, the zero will be displayed if it is followed by another decimal position with a format of Z. If there is another decimal position that follows and it has a format of zero (0) and the value is also zero, both zeros will be suppressed along with the decimal. Lastly, if the input value contains more decimal places than the output value, the number will be truncated.

Using the ZeroText Option with the FmtNum Rule

Use the ZeroText option to insert text you define instead of the zero value, if the result is zero. Here is an excerpt from a DDT file:

```
;0;0;FmtNum Field;10;12;FmtNum
Field;1;20;n,n,"$zz,zzz,zzz,zzz,zz9.99";fmtnum;1,ABC
ZeroText("INCL");N;N;N;N;7399;2675;11010;
```

Add the ZeroText option after the search mask. It should be preceded by a space. Place the text you want to print inside quotation marks and parentheses, as shown in the excerpt.

Setting Up Format Arguments

The FmtDate, RunDate, and SysDate rules let you design the format of the output. You tell the system how to format the output using format arguments. Format arguments consists of one or more codes, separated by a percent sign (%).

Characters that do not begin with a percent sign are copied unchanged to the output buffer. This lets you include static text. Any character following a percent sign that is not a format code is copied unchanged to the destination. For example, to include a percent sign in the output, add two percent signs (%%).

You can enter up to 80 characters in the mask and you can use these format codes:

Code	Description
%A	Name of the weekday, such as Tuesday
%w	Number of the weekday, (Sunday is 1, Saturday is 7)
%b	Month abbreviation, such as Mar

Code	Description
%m	Month number, (January is 1, December is 12)
%B	Month name, such as November
%d	Number of the day of the month (01 – 31)
%j	Number of the day of the year (001 – 366)
%Y	Year with the century, such as 2013
%y	Year without the century, such as 13
%H	Hour in 24-hour format (00 – 23)
%I	Hour in 12-hour format (01 – 12)
%M	Minute (00 – 59)
%S	Second (00 – 59)
%p	Current locale's AM/PM indicator for 12-hour clock
%@xxx	*xxx* identifies the locale. For example, *%@CAD%A* might produce *mardi*, the Canadian French word for Tuesday. The default locale is USD, which is US English. Once it finds a local format, the system uses that locale until it finds another locale indicator.
#	Suppress leading zeros for the following format codes. The system recognizes this flag *only* with these formats: %#d, %#H, %#I, %#j, %#m, %#M, %#S, %#w
>	Uppercase the resulting text. The system recognizes this flag *only* with these formats: %>p, %>A, %>b, %>B
<	Lowercase the resulting text. The system recognizes this flag *only* with these formats: %<p, %<A, %<b, %<B
<>	Capitalize the first letter of the resulting text. The system recognizes this flag *only* with these formats: %<>p, %<>A, %<>b, %<>B

* - This flag only affects the format code that specifies it. Any subsequent codes that have text are not affected unless they also include the flag.

NOTE: Keep in mind the system can only work with the information it receives as input.

The formats for week, hour, minute, AM, and PM (%A,%w,%H,%I,%M,%S, %p) are useful with the SysDate rule, but do not make sense for RunDate and FmtDate rules since those rules seldom see week or time information as an input.

Furthermore, you would not want to use the zero suppress format option (#) on input—especially if there are no separators in the data. For instance, the date indicated by 010109 or 1/1/09 is clear, but 1109 could indicate several things.

For example, assume the date is 03-01-2009, which was a Monday, and the time is 11:57 am. This table shows you results using various formats.

Example	Output
%m-%d-%Y	03-01-2009
The year is %Y.	The year is 2009.
Born %m/%d/%y at %I:%M %p	Born 03/01/09 at 11:57 AM
%d	01
%#d	1
%A	Monday
%>A	MONDAY
%b	Mar
%<b	mar
%p	AM
%<>p	Am
%A, %B %d	Monday, March 01
%@CAD%A %@CAD%A, %B %d	lundi, mars 01
%A, %@CAD%B %d	Monday, mars 01
%@CAD%A, %@USD%B %d	lundi, March 01

Field Format Types (fetypes)

An *fetype* defines the field format type. You can have an input and an output fetype. For example, an input fetype with the FmtNum rule tells the system where the decimal goes in the number. The output fetype tells the system how to format the output amount. An fetype can consist of either one or four characters.

NOTE: In Image Editor, you can display the Properties window for a variable field and then click the Attributes tab to enter this information in the Locale field. For the Locale field, you pick from a list of countries/languages, instead of entering one of the codes shown in the following table. The Locale field only appears if you chose Date Format, Numeric, or (Y)es or (N)o format in the Type field.

The first character of an fetype defines the field format type. There are several types defined in the system such as a *d* for dates and an *n* for numbers. You can add three additional characters to override the default locale, which is the United States (English). Here is a list of the currently supported localities:

For this country	And this language	Use this code in the FAP file:
Argentina	Spanish	ARS
Australia	English	AUD
Austria	German	ATS
Belgium	Dutch	BED
Belgium	French	BEF
Bolivia	Spanish	BOB
Brazil	Portuguese	BRC
Canada	English	CAN
Canada	French	CAD
Chile	Spanish	CLP
Columbia	Spanish	COP
Denmark	Danish	DKK
Ecuador	Spanish	ECS
European Union	English	EUR
France	French	FRF

For this country	And this language	Use this code in the FAP file:
Finland	Finnish	FIM
Germany	German	DEM
Guatemala	Spanish	GTQ
Iceland	Icelandic	ISK
Indonesia	Indonesian	IDR
Italy	Italian	ITL
Ireland	English	IEP
Liechtenstein	German	CHL
Luxembourg	French	FLX
Luxembourg	German	LUF
Mexico	Spanish	MXN
The Netherlands	Dutch	NLG
New Zealand	English	NZD
Norway	Norwegian	NOK
Panama	Spanish	PAB
Paraguay	Spanish	PYG
Peru	Spanish	PES
Portugal	Portuguese	PTE
South Africa	English	ZAR
Spain	Spanish	ESP
Sweden	Swedish	SEK
Switzerland	German	CHF
Switzerland	French	CHH
Switzerland	Italian	CHI
United Kingdom	English	GBP

For this country	And this language	Use this code in the FAP file:
United States	English	USD
Uruguay	Spanish	UYU
Venezuela	Spanish	VEB

Formatting Data with the = Operator

You can include an equals sign (=) in the data area of field-level rules, such as Move_It and MoveNum, so those rules can format data returned by the = operation.

NOTE: The system lets you use the = operator to reference GVM and DAL expressions before it rebuilds XPath search masks.The format is as follows:

```
=XXX(expression)
```

where *XXX* is one of the supported ways of finding data from a symbol, such as DAL or GVM.

This table shows your options:

This usage	Tells the system to
=()	Return the contents of the DAL variable named the same as the root name of the current source name of the current DDT field-level rule.
=("constant")	Return the value of the DAL variable that is named by the string constant.
=GVM(variable)	Return the value of the DAL variable whose name is stored in the specified DAL variable.
=(expression)	Resolve the DAL expression and use the result as the name of a DAL variable to access and return the value
=GVM()	Return the value of the GVM variable named the same as the root name of the current source name of the current DDT field-level rule.
=GVM("constant")	Return the value of the DAL variable that is named by the string constant.
=(variable)	Return the value of the GVM variable specified by the contents of the named DAL variable.
=GVM(expression)	Resolve the DAL expression and then use the result as the name of a GVM variable to access and return the value.
=DAL()	Execute a DAL script named the same as the root name of the current source name of the current DDT field-level rule and return the results.

This usage	Tells the system to
=DAL("constant")	Execute the DAL script named by the string constant and return the results.
=DAL(variable)	Execute the DAL script named by the contents of the specified DAL variable and then return the results.
=DAL(expression)	Resolve the DAL expression and use the results as the name of a DAL script to execute, and then return the results in XPATH.

Here are some examples:

```
=(....)
```

Retrieves the value of a DAL variable specified by a DAL expression

```
=("ABC")
```

Returns the contents of the DAL variable *ABC*.

```
=(ABC)
```

Returns the contents of some other DAL variable that is specified by the contents of the DAL variable *ABC*.

```
=("A" & "B" & "C")
```

Returns the same result as the *ABC* example — the contents of the DAL variable *ABC*.

```
=()
```

This retrieves the contents of a DAL variable that is, by default, the root name of the source name of the current DDT field. For example, assume, the current DDT field has destination name *MYFIELD #003* and the source name *MYFIELD #003,* then...

```
=()
```

Means to return the contents of the DAL variable *MYFIELD*. This is useful because it lets you write general purpose XDB rules.

```
=DAL(...)
```

Returns the results returned by DAL script named by the expression. For example, assume a DAL script named ABC.DAL contains:

```
MYVARIABLE = 100
RETURN(MYVARIABLE)
```

Then, =DAL("ABC") returns *100*.

Formatting imported data

The system lets you load data from a standard import file in XML, V2, or DS format. During this process, the system creates a form set and loads the imported data onto the fields on the appropriate forms.

To be able to use the various formatting rules when you have no extract file, include the following rule mask symbolic lookup operators. These operators, which begin with an equals sign (=), provide a way to access the contents of a variable field as if it were found in an extract file. For instance...

This operator	Tells the system to
=@()	Return the contents of the variable field that is the same as the current source field name.
=@(expression)	Evaluate the DAL expression to get the name of the variable field and then get its contents.

NOTE: For more information, see also information about the @ function in the DAL Reference.

Search Criteria

The GetRecord function lets the system get data records from an extract list. It searches the extract list for particular records based on search criteria formatted as shown here:

```
offset,data offset,data (and so on)
```

The search criteria is defined by one or more pairs of offsets and data. The number of pairs is limited by the size of the data field in a MEM_DDT_REC. All offsets are based on the first character in a record being character 1 (base 1)—not character zero (0).

It is not necessary for offsets to increase from left to right, but it makes for better readability. It is necessary, however, to specify your search string in the correct case. Searches are performed in a case-sensitive manner.

Because many of the section and field rules use calls to GetRecord, search criteria is often needed wholly or as part of the data field in the DDT file.

Here are some examples:

This search criteria	Finds the record...
20,HeaderRec	with the text *HeaderRec* starting at offset 20.
10,ABC 50,XYZ	with *ABC* at offset 10 and *XYZ* at offset 50.
11,~ABC 25,Header	that has a string starting at offset 11 which is not equal (~) to *ABC* and is equal to *Header* at offset 25
11,(Electric,Pwr)	that has a string starting at offset 11 which is equal to *Electric* or *Pwr*.

Overflow and User Functions

Many of the rules support the use of overflow symbols and user functions which work together. An overflow symbol can be thought of as a block of memory that holds a counter. This counter, or overflow variable, tracks the number of records processed which helps the system determine which record to start with after it handles an overflow situation.

To use overflow, you must include specific data in the DDT file. This overflow data consists of the…

- @GetRecsUsed function
- Name of the form
- Overflow symbol

The @GetRecsUsed function is a function the rule runs to access information about a pre-defined overflow symbol. The overflow symbol is stored in the DDT file with the field level rules. You must define all overflow symbols using the SetOvFlwSym rule, which is a job level rule (level 1) stored in the AFGJOB.JDT file.

The second part is the name of a form, which is retrieved from the form definition file (FORM.DAT) specified in the INI file.

The third part is the overflow symbol itself.

The format of these data items in the DDT data field are as follows:

```
;@GETRECSUSED,FORMNAME,SYMBOL/ADDITIONAL_DATA;
```

NOTE: The first three data items are separated by commas. These items are separated from the rest of the data that the rule requires by a forward slash (/).

Here is an example:

```
@GETRECSUSED,DETAILS,Symbolnm/11,DETAILREC;N;N;N;
```

Chapter 5

Section and Field Rules Reference

Section (image) and field rules help you control how data is processed and generated to fill a field on a form.

NOTE: This chapter serves as a reference to the section and field rules. For information on the rules which apply to jobs and form sets, see Adding Job and Form Set Rules on page 5.

This chapter discusses rules included in the base system and supported by the Oracle Documaker support staff. For information on custom rules, contact your Services representative.

For a summary of these rules, see Section and Field Rules Reference on page 274.

Section and Field Rules Reference

The following pages list and explain the various section and field rules you can use. The rules are discussed in alphabetical order on the pages following this table.

NOTE: You can also see information about the section and field rules while using Studio when you select the rule on the Rule Properties window.

When you select a rule, information about that rule appears here:

If you are using Image Editor, select the Help, Explain Rule option.

The following table lists the rules discussed in this chapter by type (section or field) and purpose.

The Level column indicates whether the rule is a section level rule (3) or a field level rule (4). The Overflow column indicates the rules which support the overflow feature. The overflow features allow extract data to flow onto an additional page if needed.

To…	Level	Use this rule	Overflow
add a page break before the system begins processing the current section	3	PaginateBeforeThisImage on page 421	na
add sections to the current form set based on conditions in the SETRCPTB.DAT file	3	SetRecipFromImage on page 466	na
add TIFF images contained in a single TIFF file in a form set	3	AddMultiPageTIFF on page 292	na
allow a chart's series data to be retrieved via reference to variable fields defined on the same section	3	FieldVarsToChartSeries on page 335	na
check to see if the FAP file is loaded, and if not, load the FAP file	3	CheckImageLoaded on page 307	na
create a temporary extract list which contains similar records in a transaction	3	CreateSubExtractList on page 317	na
create a GVM variable from fields in a section	3	Field2GVM on page 333	na
create custom axis labels for a chart	3	SetCustChartAxisLabels on page 455	na
define the first section in a group of sections	3	GroupBegin on page 343	yes
define the last section in a group of sections	3	GroupEnd on page 355	yes
delete a page from a form set	3	DontPrintAlone on page 329	na
delete a specific occurrence of a section	3	DelImageOccur on page 328	na
draw an underline beneath a variable field	3	UnderlineField on page 484	na
execute a DAL script	3	PostImageDAL on page 422	na

To…	Level	Use this rule	Overflow
execute a DAL script	3	PreImageDAL on page 426	na
get data from extract records and include it as series data in a chart	3	CreateChartSeries on page 315	na
import PDF or TIFF files as bitmap images.	3	AddMultiPageBitmap on page 283	na
increment an overflow variable	3	IncOvSym on page 366	yes
map fields in the XDB database	4	XDB on page 485	na
map fields in the XDD database	4	XDD on page 488	na
merge data for embedded variable fields in a text area with text	3	TextMergeParagraph on page 483	na
move and align field text so the data elements are connected.	3	ConnectFields on page 312	na
move sections from the current page to a page you specify	3	MoveMeToPage on page 401	na
remove a series from a chart if the series contains no data	3	PurgeChartSeries on page 433	na
remove series data from the series you specify	3	DeleteDefaultSeriesData on page 327	na
remove the white space from between fields.	3	RemoveWhiteSpace on page 434	na
reset an overflow variable	3	ResetOvSym on page 438	yes
reset section dimensions	3	ResetImageDimensions on page 436	na
set group options	3	SetGroupOptions on page 439	yes
set the dimensions of a section	3	SetImageDimensions on page 457	na
set the section overlay/page segment X and Y coordinates using FAP units	3	SetOrigin on page 458	na
set the section overlay/page segment X and Y coordinates using inches	3	SetOriginI on page 462	na

To…	Level	Use this rule	Overflow
set the section overlay/page segment X and Y coordinates using millimeters	3	SetOriginM on page 464	na
set the send copy to variable	3	SetCpyTo on page 454	na
span a field's width between two other fields, filling in with a fill character	3	SpanAndFill on page 470	na
Field Level Rules			
add a placeholder which causes no operation to occur (used in testing)	4	NoOpFunc on page 415	na
add a variable from more than one occurrence of a particular record type	4	AccumulateVariableTotal on page 280	na
add two fields and insert the result into a new field	4	MoveSum on page 411	na
call a DAL function	4	If on page 360	na
concatenate strings and place the result in the field you specify	4	ConCat on page 311	yes
copy alphanumeric data from a table using the source record field as a key	4	TblLkUp on page 476	yes
copy and format numeric data in an extract record	4	MoveNum on page 402	yes
copy data from an external record into the output buffer	4	Move_It on page 393	yes
copy data from an SAP Raw Data Interface (RDI) extract file	4	SAPMove_It on page 443	yes
copy data from the table list of records into the output buffer	4	MovTbl on page 413	na
copy data if a source record exists	4	MoveExt on page 399	yes
count the *total* number of overflow records that could be processed per transaction	4	OvPrint on page 419	yes
create lists of data for populating section lists or columns	4	BldGrpList on page 301	na

To…	Level	Use this rule	Overflow
display the difference between two dates	4	DateDiff on page 322	na
emulate TerSub entry functionality	4	TerSubstitute on page 480	yes
execute the MoveNum rule if an external record is found	4	MNumExt on page 390	yes
force a transaction to manual batch (WIP)	4	KickToWip on page 372	na
force a transaction to manual batch (WIP)	4	PowType on page 424	na
format a date	4	DateFmt on page 324	yes
format a date (for international localities)	4	FmtDate on page 337	yes
format a number	4	FmtNum on page 338	yes
format the system date	4	SysDate on page 474	yes
get a text table item based on a key built from the source field name concatenated with the data retrieved from the source record	4	TblText on page 478	yes
get data from an extract record, look up the data in a table, and copy the table data to the destination field	4	LookUp on page 374	na
get information from an extract file based on conditions you specify	4	If on page 360	yes
get information from an extract file based on conditions you specify	4	DAL on page 320	yes
get the current system date	4	FfSysDte on page 331	yes
get the run date from the TRNFILE.DAT file and format it using the mask you specify	4	RunDate on page 440	na
insert a specific value	4	Mk_Hard on page 388	na
insert a value in a field only if a record is found in the extract data using the search criteria you specify	4	HardExst on page 356	yes
justify a field (right, left, or center)	4	JustFld on page 367	na

To…	Level	Use this rule	Overflow
print information in a field if the data matches the numeric value you specify	4	PrtIfNum on page 430	na
print information in a field if the data matches the string you specify	4	PrintIf on page 428	yes
process multi-page sections	4	EjectPage on page 330	na
replace the NoOpFunc rule	4	MapFromImportData on page 376	na
report the *actual* number of overflow records that could be processed per transaction	4	OvActPrint on page 417	yes
retrieve a message from an extract file	4	MessageFromExtr on page 380	yes
retrieve and format a string	4	StrngFmt on page 472	yes
select the largest value of multiple packed decimal fields located on the same record to populate a variable field	4	CompBin on page 308	na
speed the processing of fields used repeatedly throughout a form set	4	GlobalFld on page 340	na
store and retrieve subsequent lines of a multiple line address	4	SetAddr on page 445	yes
store and retrieve subsequent lines of a multiple line address	4	SetAddr2 on page 448	yes
store and retrieve subsequent lines of a multiple line address	4	SetAddr3 on page 451	yes
tell the system the field has been mapped to the master DDT file	4	Master on page 379	na
translate a numeric ISO state code into the actual state name	4	SetState on page 468	yes

AccumulateVariableTotal

Use this field level rule (level 4) when you need to sum a variable from more than one occurrence of a particular record type. This rule only works with the Record Dictionary.

Syntax

```
output format;AccumulateVariableTotal;Record( ) Variable ( ) Cond (
);;
```

Parameter	Description
Record	Name of the record pointer defined in the Records group of the Record Dictionary file (entitled DataDict). This record pointer defines the column to search, the text to look for in the starting column, and option flags.
Variable	Name of the variable pointer defined in the Variables group of the Record Dictionary. This variable pointer defines offset into the record where the data to be accumulated is located, the length and type of the data, and formatting flags.
Cond	(Optional) Name of the condition defined in the Conditions group of the Condition table. The condition consist of combinations of comparisons, parenthesis, ANDs, and ORs to verify the correct results.

To format the output, you can also include any of the following format options in the Mask field on the Field Options window in Studio or in the Mask field on the Edit DDT tab of the field's Properties window in Image Editor. Separate each option with a comma.

Option	Description
- (one dash)	If the number is negative, this option places a minus sign (-) in the left most position. For example, if the format mask is (9.2,12.2,C,$,-), the result is: "-$2,100.00".
-- (two dashes)	If the number is negative, this option places a minus sign (-) immediately before the amount. For example, if the format equal is (9.2,12.2,C,$,--), the result is " -$2,100.00", with a full length of 12.
+	Tells the system to always include a sign with all numbers.
%	Appends a percent sign (%) at the end of the number.
$	Adds a dollar sign. Cannot be the first character in the format mask. This limitation arises from the Move_It format option, where a dollar sign ($) in the first character of the mask means to perform a sprintf.
C	Adds commas.
C**	Adds commas if in US English format or spaces if in Canadian French format.
CR	Appends CR to the end of the number.
CS1 CS2 CS731	Enter one of these options to indicate the checksum method. The system appends a check digit (mod 10) of 0 through 9 to the end of the number. This is typically used in accounting to make sure a number, such as an account number, is correct by performing a formula on each digit. For details, see the discussion on page 406.

Option	Description
D	Dollars (a combination of B, C, and $. You must modify GEN_FMT_FmtMaskSaysBinary to recognize this format.)
E	Stops a calculation if the search condition is false. The Move_It rule may return a null output buffer if: - no record was found; a record was found, but the search mask contained a pairing (offset,data) which extended past the end of the record - a record was found, but the mapped data was blank.
F	Adds a dollar sign ($) and places it in the first position. If the value is negative, it moves the minus sign (-) to the last position.
L	Left justifies the number.
-L (or --)	Tells the system to use a floating negative sign on negative values.
+L (or++)	Tells the system to use a floating sign and to always show that sign.
Lang	Selects a language for spelling out the number. This flag is used with the V flag and mask parameters. Here is an example: US, CFR.
M	Money (This format is a combination of formats C and $.)
N	Leaves the output buffer blank if the number is zero or negative.
NM	Adds a minus sign (-) to the number.
P	Print leading zeros. You cannot use this format with $, -, C, and F.
P**	Prints leading zeros if used without character or symbol enclosed with single quote.
SLZ	Suppresses leading zeros. For example, 00.25 becomes .25.
T	Used with the NegText, Text, and ZeroText data options. Adds text before or after a number. Use the less than (<) symbol for inserting before, the greater than (>) symbol for inserting after. Use the comma as a separator. You can also use this option to place currency symbols before or after amounts. For instance, T>£ places the British pound sterling symbol (ALT+0163) before an amount.
V	Spells out the numeric value in US English.
X	Adds an *x* before the number.
Z	Prints a number even if it is zero.
Z2	Prints two zeros.

Image Editor example

Assume the content of the Record Dictionary is as follows:

```
<Records>
Detail =  Search(61,18) Repeating
TotalDtl = Search(61,18,96,~00625,101,(01,02)) Repeating
Usage = Search(61,08) Repeating
RTP = Search(61,21) Repeating
```

```
<Variables>
DTAT = Record(Detail) Offset( 163) Length(10) Type(Zone)
Format(14.2,C,Z) Precision(2)
DTAT$ = Record(Detail) Offset( 163) Length(10) Type(Zone)
Format(14.2,C,Z,$) Precision(2)
```

And you add this rule to the DDT file:

```
;0;0;DetailTotal;0;15;DetailTotal;0;15;10.2,14.2,C,Z,$;AccumulateVa
riableTotal;Record(Detail) Variable(DTAT$)
Cond(Type);N;N;N;N;25947;15921;16006;
```

Each time this rule encounters a record which matches the search criteria (18 starting in column 61), it accumulates a total for the variable DTAT$ (the data found at offset 163 for a length of 10 formatted to the specification stated in the variable description).

The conditional parameter (Cond(Type)) is an optional parameter defined in the Condition Table. This parameter is used to limit your search criteria.

See also Section and Field Rules Reference on page 274

Using Condition Tables on page 492

Using the Record Dictionary on page 495

AddMultiPageBitmap

Use this section level (level 3) rule to import PDF or TIFF files as bitmap images. If the file consists of multiple pages, the system inserts the first page on the triggering form or section. For each subsequent page in the file, the system generates additional pages and appends them to the form after the triggering section.

See Using the Type Option on page 289 for information on importing specific file types.

NOTE: When you use this rule with TIFF files, it performs the same task and works just like the AddMultiPageTIFF rule. The first TIFF in the file is inserted on the triggering form/section. Subsequent TIFF images trigger additional pages which are appended to the form after the page which contains the first TIFF image.

Syntax

```
;AddMultiPageBitmap;Options;;
```

For the Options parameter, this table describes your choices:

Option	Description
Opt	(Optional) Enter Yes to indicates this rule is optional and you do not want error messages generated if the file naming parameters fail to produce a valid name. The default is No. This option lets you use multiple named parameters. The first parameter that provides a usable file name is used. Make this option the first rule parameter.
Use one of the following options (DAL, File, GVM, or SRCH) to specify the file name.	
File(file name)	Enter the name and path of the file you want to import. See Using the File Option on page 286 for more information.
DAL(script name)	Enter the name of the DAL script you want to execute to return the name of the file you want to import. You must enter the name of a script file or DAL library routine. Do not include DAL statements. See Using the DAL Option on page 287 for more information.
SRCH(search criteria name data)	The name and path of the file you want to import is contained in a record in the file specified by the ExtrFile option in the Data control group. The search criteria are one or more comma-delimited data pairs, offsets, and character strings, used to as the search mask to find the record in the file you specified. The name data is a comma-delimited data pair that defines the offset and length of the file name in the record defined by the search criteria. Separate the search criteria and name data by a space. See Using the SRCH Option on page 288 for more information.
GVM(variable name)	Enter the GVM variable name that contains the name and path of the file you want to import. The GVM variable data is mapped by some other means before this rule is executed. See Using the GVM Option on page 288 for more information.

Option	Description
Embed	(Optional) Include Embed to add the image data into the NA file. This is necessary when the file that contains the scanned images is temporary and you need to archive the NA/POL information. Upon retrieval, if you have not embedded the bitmap information directly into the form set, you will not be able to view or reprint the original images. The default is No. Keep in mind that embedding bitmap data can make the resulting NA file much larger and also affects the size of the archives generated.
Only (Odd or Even)	(Optional) By default, all images in the file are included. Only include this option to specify that you want only the odd or even numbered images. You can use this option to reduce the size of the output when you know blank pages are included in the scanned images on every other page. Choose Odd when you know that the first image is not blank. This includes images 1, 3, 5, and so on. Choose Even to start with the second image. This includes images 2, 4, 6, and so on. If you include both Only (Odd) and Only (Even), you exclude all images.
IN(top,left)	(Optional) Specifies the coordinate for the top-left corner of the bitmaps, in inches. The default is position 0,0 within the image.
MM(top,left)	(Optional) Specifies the coordinate for the top-left corner of the bitmaps, in millimeters. The default is position 0,0 within the image.
Top(top,left)	(Optional) Specifies the coordinate for the top-left corner of the bitmaps, in FAP units (2400 per inch). The default is position 0,0 within the image.
Type	(Optional) Enter T (TIFF) or P (PDF). If you omit this option, the system first looks for TIFF files. If it cannot find a TIFF file, it looks for a PDF file. Including this option will speed processing. You can also include this option if the target directory contains both TIFF and PDF files. For instance, if the directory contains import1.tif and import1.pdf, the TIFF file is included by default. If you want to include the PDF file, use the Type option. See Using the Type Option on page 289 for more information.
Scale(height[in|mm],width[in|mm])	(Optional) Use this option to resize the loaded graphics while maintaining the aspect ratio (height to width), so the graphic fits within the provided height and width dimensions. If you only provide the height or width, the system sizes the graphic to fit that dimension and automatically calculates the other dimension to preserve the aspect ratio. See Using the Scale Option on page 289 for more information.
Crop(height[in|mm],width[in|mm])	(Optional) Use this option to remove all parts of the graphic that extend beyond the specified distances from the top left corner. If you omit one of the arguments, the graphic is not modified in that dimension. See Using the Crop Option on page 290 for more information.

Keep in mind:

- In z/OS environments, you can import TIFF files or import only the bitmap data contained in PDF files. Under z/OS, this rule imports the bitmaps contained in the PDF file, puts them at the position you specified with the position options (IN, MM, or TOP) and scales them to fit the page.

  Importing bitmap data from inside PDF files is useful because some fax drivers take TIFF data and place it inside a PDF file. Therefore, by reading the bitmap data from the PDF file, you are importing all the valuable information in that file.

- You can specify several AddMultiPageBitmap rules, as shown here, but realize that each subsequent rule reuses the document pages added by previous rules.

  ```
  <Image Rules>
  …
  ;AddMultiPageBitmap;DAL(TIF_DAL.dal),Only(ODD);
  ;AddMultiPageBitmap;DAL(TIF_DAL.dal),Only(ODD);
  …
  ```

  For instance, suppose you have declared two rules. The first has a 4-page file. The second has a 5-page file.

  After executing the two rules, there will be five pages in the form. The first four pages will have two images each (one from the first rule and one from the second) and the final page will contain the last image from the 5-page file.

  Be aware that the placement of those bitmap images on the page can make them overlap.

NOTE: This rule supports long file names on 32-bit Windows operating systems.

- The system supports these types of TIFF images:

Type	Description
Type 1	uncompressed
Type 2	Huffman
Type 3	CCITT group 3 FAX
Type 4	CCITT group 4 FAX
Type 5	LZW
Type 32773	Packbits

- When importing TIFF or PDF files, keep in mind you can only include one of the image positioning parameters, Top, In, or MM. The value specified is relative to the FAP file's origin as specified by a SetOrigin rule. If there are more than one positioning parameters, subsequent definitions override prior ones.

If you omit the positioning parameters, the default top/left coordinate is taken from the margin defined for the FAP file. If the FAP file is not loaded and the margins are unknown, the default is 0,0 (aligned with the top of the image).

- For TIFF and PDF files, if either the LoadFAPBitmap option or the Embed parameter are set to Yes, the bitmap is loaded into memory. If neither are enabled, the system opens the file to get the bitmap size, resolution, and number or pages, but the bitmap data is not loaded. The system then assumes all of the bitmap images are the same size as the first image in the file.

  For single step mode, set LoadFAPBitmap option to Yes.

- You can use the PDFImportDPI option to set the resolution at which PDF files are imported.

```
< BitmapLoaders >
    PDFImportDPI =
```

Option	Description
PDFImportDPI	Enter the resolution in dots per inch (DPI) at which you want to import PDF files. The default is 100 DPI. A higher DPI gives you better fidelity, but the import process will take longer and the output files will be larger.

Example

These examples show how you can define the file to import when you use this rule. Assume that your MRL has these sub-directories which contain these PDF files:

Directory	File name
PDF_DAL	A_DAL.PDF
PDF_File	A_FILE.PDF
PDF_GVM	A_GVM.PDF
PDF_SRCH	A_SRCH.PDF

Using the File Option

This example imports the A_FILE.PDF file from the PDF_File directory. Using this file, the GenData program adds the PDF images contained in the single PDF file to the form set. Each image in the PDF file causes a duplicate of the original FAP image to be appended to the form. This duplicate contains the bitmap image.

Here is an excerpt from a sample DDT file using the File option:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageBitmap;Opt(Y), File(.\PDF_FILE\A_File.pdf);;
…
```

NOTE: Keep in mind that if the OPT option is set to No, which is the default, the system expects you to provide a file name, otherwise you get an error.

If you set the OPT option to Yes, this tells the system that if the data for the file name is not provided it should skip to the next rule without creating an error message. Setting OPT to Yes simply tells the system that if no file name is provided, regardless of the mapping method you are using, it should not be considered an error. Here is an example:

```
;AddMultiPageBitmap;OPT(Y), SRCH(1,PDF 10,25);
```

You get no error if the PDF record does not exist in the extract file or if there is PDF record but as offset 10 for 25 bytes, there is nothing but spaces. If the OPT(Y) option is omitted, you get one of these messages, depending on your situation:

```
SRCH() A record matching the search mask <1,PDFF> could not be
 located.
SRCH() Filename location within search record <1,PDF> is blank.
 Offset <10,> Length <25>.
```

Here is another example:

```
;AddMultiPageBitmap;OPT(Y), GVM(PDF_GVM);
```

If PDF_GVM contains no data and the OPT(Y) option is specified, you get no error. If the OPT(Y) option is omitted, the system generates an error similar to this one:

```
GVM(<PDF_GVM>) Global variable does not exist or is empty.
```

Here is another example:

```
;AddMultiPageBitmap;OPT(Y), DAL(AddPDF.dal);
```

If processing the AddPDF.dal script results in an empty string and the OPT(Y) option is specified, you get no error. If the OPT(Y) option is omitted, the system generates an error similar to this one:

```
DAL(<AddPDF.dal>) script returned no result or result was blank.
```

The thing to remember is that if no data exists and the OPT option is set to Yes, no error message appears.

Using the DAL Option

This example executes the PDF_NAME.DAL DAL script which returns the file name, *F_DAL.PDF*. Using this file name, the GenData program adds the images contained in the single PDF file to the form set. Each image in the PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the bitmap image. Only the odd images in the PDF file are included because the Only option is set to *Odd*.

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
```

```
;AddMultiPageBitmap;DAL(PDF_DAL.dal),Only(ODD);;
…
```

Using the SRCH Option

This example imports PDF files (*F_SCH1.PDF, F_SCH2.PDF*, and *F_SCH3.PDF*) based on the content of lines in the file designated by the ExtrFile option in the Data control group. Using this file, the GenData program adds the images contained in the three PDF files to the form set. Each image in the PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the bitmap image. The bitmap images are embedded in the NA file because the Embed option is set to Yes.

Here is an example of the extract file records pointed to by the ExtrFile option:

```
0          1
1          1
SCOxxxxxxxHEADERREC
…
…
PDF_File_Name .\PDF\F_SCH1.PDF
…
…
```

NOTE: This option lets you import and process multiple PDF files because of the way the file name and path are specified — one file per entry in the file pointed to by the ExtrFile option.

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageBitmap;SRCH(1,PDF_File_Name 15,17),Embed(Y);;
```

Using the GVM Option

This example imports a PDF file based on file name contained in the GVM variable called *PDF_File_GVM*. Using the PDF file name and path in the GVM variable, the GenData program adds the PDF images contained in the single PDF file to the form set. Each image in the PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the bitmap image.

NOTE: Keep in mind you can use any valid GVM variable, no matter how it is created or assigned.

To create the *PDF_File_GVM* variable, you would include the following INI option in your FSISYS.INI file and add its definition in the TRNDFDFL.DFD file.

```
< GenTrnDummyFields >
    PDF_File_GVM = .\PDF_gvm\A_GVM
```

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageBitmap;GVM(PDF_File_GVM);;
```

Using the Type Option

You can use the Type option to specify the type of file you want to import to speed processing. Enter **T** (TIFF) or **P** (PDF). If you omit this option, the system first looks for TIFF files. If it cannot find a TIFF file, it looks for a PDF file. Keep in mind:

- Pages imported from a PDF file are placed at coordinates (0,0) in the output by default. You can use the position options (IN, MM, or TOP) to specify another position.
- You can specify several AddMultiPageBitmap rules, as shown here, but realize that each subsequent rule reuses the document pages added by previous rules. Here is an example:

  ```
  <Image Rules>
  …
  ;AddMultiPageBitmap;DAL(PDF_DAL.dal),Only(ODD),TYPE(P);
  ;AddMultiPageBitmap;DAL(PDF_DAL.dal),Only(ODD),TYPE(P);
  …
  ```

  Assume the first PDF file contains four pages and the second PDF file contains five pages.

  After executing the two rules, there will be five pages in the form. The first four pages will have two images each (one from the first rule and one from the second) and the final page will contain the last image from the 5-image PDF file.

  Be aware that the placement of those bitmap images on the page can make them overlap.

NOTE: This rule supports long file names on 32-bit Windows operating systems.

Using the Scale Option

The Scale option resizes the loaded graphics, maintaining the aspect ratio (height to width), so the graphic fits within the provided height and width dimensions. If you only provide the height or width, the system sizes the graphic to fit that dimension and automatically calculates the other dimension to preserve the aspect ratio. For example (assuming the imported graphic is originally 8 ½" x 11"), this rule.

```
;AddMultiPageBitmap;SRCH(1,AddMultiPageBitmap,40,.\tif_srch 40,19),
SCALE(4in);;
```

tells the system to scale the graphic so that it is 4 inches high and 3.09 inches wide.

This rule....

```
;AddMultiPageBitmap;SRCH(1,AddMultiPageBitmap,40,.\tif_srch
40,19),SCALE(4in,3in);;
```

tells the system to scale the graphic to 3.88 inches tall and 3 inches wide, because if it scaled the height to be 4 inches, while maintaining the aspect ratio, the width would exceed the specified 3 inches.

This rule.

```
;AddMultiPageBitmap;SRCH(1,AddMultiPageBitmap,40,.\tif_srch
40,19),SCALE(,5in);;
```

tells the system to scale the graphic to 6.47 inches tall and 5 inches wide.

Keep in mind...

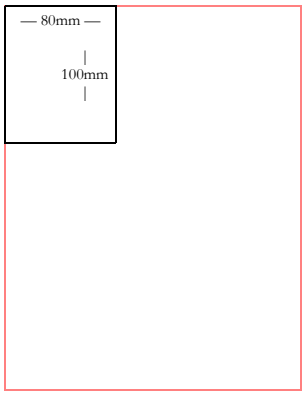
- If you omit the height, you must include a comma (,) as a placeholder.
- If no units, such as inches (in) or millimeters (mm), are provided, the system assumes your entry is in FAP units (2400 per inch).
- Do not include both the Scale and Crop options. If you include both options, the system ignores the Crop option and only uses the Scale option.
- Only the PDF Print Driver supports the Scale option.

Using the Crop Option

The Crop option removes all parts of the graphic that extend beyond the specified distances from the top left corner. If you omit one of the arguments, the graphic is not modified in that dimension. For example, this rule...

```
;AddMultiPageBitmap;SRCH(1,AddMultiPageBitmap,40,.\tif_srch
40,19),CROP(100mm,80mm);;
```

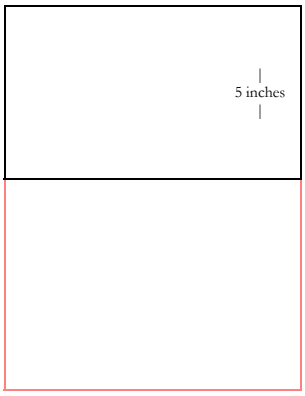
tells the system to only include in the print stream an area of the graphic that is 100 mm tall by 80 mm wide, beginning in the top, left corner.



This rule...

```
;AddMultiPageBitmap;SRCH(1,AddMultiPageBitmap,40,.\tif_srch
40,19),CROP(12000);;
```

tells the system to only include the top five inches (12000 FAP units) of the full-width graphic in the print stream.



Keep in mind...

- If no units, such as inches (in) or millimeters (mm), are provided, the system assumes your entry is in FAP units (2400 per inch).
- Do not include both the Scale and Crop options. If you include both options, the system ignores the Crop option and only uses the Scale option.
- Only the PDF Print Driver supports the Crop option.

See also Section and Field Rules Reference on page 274

AddMultiPageTIFF

NOTE: Beginning with version 11.3, the AddMultiPageTIFF rule was replaced by the AddMultiPageBitmap rule. If the system comes across the AddMultiPageTIFF rule, it auto ma ti ally runs the AddMultiPageBitmap rule.

See AddMultiPageBitmap on page 283 for more information.

Use this section level (level 3) rule to include multiple TIFF images contained in a single TIFF file in a form set. The first TIFF in the file is inserted on the triggering form/section. Subsequent TIFF images trigger additional pages which are appended to the form after the page which contains the first TIFF image.

This rule can also extract TIFF, JPEG, or bitmap images from PDF files. For instance, if you have a PDF file that includes scanned images, typically in TIFF format, you can use this rule to extract those images from the PDF file.

NOTE: For more information on using this rule to extract TIFF, JPEG, or bitmaps images from PDF files, see Using the Type Option on page 297.

Syntax

```
;AddMultiPageTIFF;Options;;
```

For the Options parameter, you have these options:

Option	Description
Opt	(Optional) Enter Yes to indicates this rule is optional and you do not want error messages generated if the file naming parameters fail to produce a valid name. The default is No. This option lets you use multiple named parameters. The first parameter that provides a TIFF file name is used, but if the name of the TIFF file returned by the search criteria is blank, the system ignores it and does not generate an error. Make this option the first rule parameter.
DAL(script name)	Enter the name of the DAL script you want to execute to return the name of the TIFF file you want to import. You must enter the name of a script file or DAL library routine. Do not include DAL statements.
File(file name)	Enter the name and path of the TIFF file to import.
GVM(variable name)	Enter the GVM variable name that contains the name and path of the TIFF file. The GVM variable data is mapped by some other means before this rule is executed.

Option	Description
SRCH(search criteria name data)	The name and path of the TIFF file is contained in a record in the file specified by the ExtrFile option in the Data control group. The search criteria are one or more comma-delimited data pairs, offsets, and character strings, used to as the search mask to find the record in the file you specified. The name data is a comma-delimited data pair that defines the offset and length of the file name in the record defined by the search criteria. Separate the search criteria and name data by a space.
Embed	(Optional) Include Embed to add the image data in the NA file. This is necessary when the file that contains the scanned images is temporary and you need to archive the NA/POL information. Upon retrieval, if you have not embedded the bitmap information directly into the form set, you will not be able to view or reprint the original images. The default is No. Keep in mind that embedding bitmap data can make the resulting NA file much larger and also affects the size of the archives generated.
Only (Odd or Even)	By default, all images in the multi-page TIFF file are included. Only include this option to specify that you want only the odd or even numbered images. You can use this option to reduce the size of the output when you know blank pages are included in the scanned images on every other page. Choose Odd when you know that the first image is not blank. This includes images 1, 3, 5, and so on. Choose Even to start with the second image. This includes images 2, 4, 6, and so on. If you include both Only (Odd) and Only (Even), you exclude all images.
IN(top,left)	(Optional) Specifies the coordinate for the top-left corner of the bitmaps, in inches.
MM(top,left)	(Optional) Specifies the coordinate for the top-left corner of the bitmaps, in millimeters.
Top(top,left)	(Optional) Specifies the coordinate for the top-left corner of the bitmaps, in FAP units (2400 per inch).
Type	(Optional) Enter T (TIFF) or P (PDF). If you omit this option, the system first looks for TIFF files. If it cannot find a TIFF file, it looks for a PDF file. Including this option will speed processing. You can also include this option if the target directory contains both TIFF and PDF files.

Keep in mind:

- The system supports these types of TIFF images:

Type	Description
Type 1	uncompressed
Type 2	Huffman
Type 3	CCITT group 3 FAX

Type	Description
Type 4	CCITT group 4 FAX
Type 5	LZW
Type 32773	Packbits

- You can only include one of the section positioning parameters, Top, In, or MM. The value specified is relative to the FAP file's origin as specified by a SetOrigin. If there are more than one positioning parameters, subsequent definitions override prior ones.

  If you omit the positioning parameters, the default top/left coordinate is taken from the margin defined for the FAP file. If the FAP file is not loaded and the margins are unknown, the default is 0,0 (aligned with the top of the section).

- If either the LoadFAPBitmap option or the Embed parameter are set to Yes, the bitmap is loaded into memory. If neither are enabled, the system opens the TIFF file to get the bitmap size, resolution, and number or pages, but the bitmap data is not loaded. The system then assumes all of the bitmap images will be the same size as the first image in the file.

  For single step mode, set LoadFAPBitmap option to Yes.

- If you include several options that serve similar purposes, the last one to provide a result is used.

- You can specify several AddMultiPageTIFF rules, as shown here, but realize that each subsequent rule reuses the document pages added by previous rules.

```
<Image Rules>
…
;AddMultiPageTIFF;DAL(TIF_DAL.dal),Only(ODD);
;AddMultiPageTIFF;DAL(TIF_DAL.dal),Only(ODD);
…
```

  For instance, suppose you have declared two rules. The first has a 4-page TIFF. The second has a 5-page TIFF.

  After executing the two rules, there will be five pages in the form. The first four pages will have two images each (one from the first rule and one from the second) and the final page will contain the last image from the 5-page TIFF.

  Be aware that the placement of those bitmap images on the page can make them overlap.

NOTE: This rule supports long file names on 32-bit Windows operating systems.

Image Editor example

These examples show how you can define the TIFF file to import when you use this rule. Assume that your MRL has these sub-directories which contain these TIFF files:

Directory	File name
TIF_DAL	T_DAL.TIF
TIF_File	T_FILE.TIF
TIF_GVM	T_GVM.TIF
TIF_SRCH	T_SRCH.TIF

Using the File Option

This example imports the T_FILE.TIF file from the TIF_File directory. Using this file, the GenData program adds the TIFF images contained in the single TIFF file to the form set. Each image in the TIFF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the bitmap image.

Here is an excerpt from a sample DDT file using the File option:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTiff;Opt(Y), File(.\TIF_FILE\T_File.tif);;
…
```

NOTE: Keep in mind that if the OPT option is set to No, which is the default, the system expects you to provide a file name, otherwise you get an error.

If you set the OPT option to Yes, this tells the system that if the data for the file name is not provided it should skip to the next rule without creating an error message. Setting OPT to Yes simply tells the system that if no file name is provided, regardless of the mapping method you are using, it should not be considered an error. Here is an example:

```
;AddMultiPageTiff;OPT(Y), SRCH(1,TIFF 10,25);
```

You get no error if the TIFF record does not exist in the extract file or if there is TIFF record but as offset 10 for 25 bytes, there is nothing but spaces. If the OPT(Y) option is omitted, you get one of these messages, depending on your situation:

```
SRCH() A record matching the search mask <1,TIFF> could not be
 located.
```

```
SRCH() Filename location within search record <1,TIFF> is blank.
 Offset <10,> Length <25>.
```

Here is another example:

```
;AddMultiPageTiff;OPT(Y), GVM(TIFF_GVM);
```

If TIFF_GVM contains no data and the OPT(Y) option is specified, you get no error. If the OPT(Y) option is omitted, the system generates an error similar to this one:

```
GVM(<TIFF_GVM>) Global variable does not exist or is empty.
```

Here is another example:

```
;AddMultiPageTiff;OPT(Y), DAL(AddTiff.dal);
```

If processing the AddTiff.dal script results in an empty string and the OPT(Y) option is specified, you get no error. If the OPT(Y) option is omitted, the system generates an error similar to this one:

```
DAL(<AddTiff.dal>) script returned no result or result was blank.
```

The thing to remember is that if no data exists and the OPT option is set to Yes, no error message appears.

Using the DAL Option

This example executes the TIF_NAME.DAL DAL script which returns the file name, *F_DAL.TIF*. Using this file name, the GenData program adds the TIFF images contained in the single TIFF file to the form set. Each image in the TIFF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the bitmap image. Only the odd images in the TIFF file are included because the Only option is set to *Odd*.

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;DAL(TIF_DAL.dal),Only(ODD);;
…
```

Using the SCH Option

This example imports TIFF files (*F_SCH1.TIF, F_SCH2.TIF,* and *F_SCH3.TIF*) based on the content of lines in the file designated by the ExtrFile option in the Data control group. Using this file, the GenData program adds the TIFF images contained in the three TIFF files to the form set. Each image in the TIFF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the bitmap image. The bitmap images are embedded in the NA file because the Embed option is set to Yes.

Here is an example of the extract file records pointed to by the ExtrFile option:

```
0          1
1          1
SCOxxxxxxxHEADERREC
…
```

```
…
TIF_File_Name .\tiff\F_SCH1.tif
…
…
```

NOTE: This option lets you import and process multiple TIFF files because of the way the file name and path are specified—one file per entry in the file pointed to by the ExtrFile option.

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;SRCH(1,TIF_File_Name 15,17),Embed(Y);;
```

Using the GVM Option

This example imports a TIFF file based on file name contained in the GVM variable called *TIFF_File_GVM*. Using the TIFF file name and path in the GVM variable, the GenData program adds the TIFF images contained in the single TIFF file to the form set. Each image in the TIFF file causes a duplicate of the original FAP image to be appended to the form. This duplicate contains the bitmap image. Note the Top option is set to *0,400*. This sets the top/left coordinate for each bitmap to 0,400 FAP units.

NOTE: Keep in mind you can use any valid GVM variable, no matter how it is created or assigned.

To create the *TIFF_File_GVM* variable, you would include the following INI option in your FSISYS.INI and add its definition in the TRNDFDFL.DFD file.

```
< GenTrnDummyFields >
    TIFF_File_GVM = .\tif_gvm\T_GVM
```

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;GVM(TIFF_File_GVM),Top(0,400);;
```

Using the Type Option

Use the Type option to tell the AddMultiPageTIFF rule that the target image is a PDF file which contains TIFF, JPEG, or bitmap images. The first image found in the PDF file is inserted on the triggering form or image. Subsequent images in the PDF file trigger additional pages which are appended in the order in which they appear in the PDF file.

NOTE: If the PDF file contains anything else, such as text, those items will be discarded. Only TIFF, JPEG, and bitmap images are added.

Keep in mind...

- The images in the PDF file are sized to fit the defined page dimensions for the form, beginning at coordinates 0,0 in the output.
- If you include several options that serve similar purposes, the last one to provide a result is used.
- You can specify several AddMultiPageTIFF rules, as shown here, but realize that each subsequent rule reuses the document pages added by previous rules. Here is an example:

```
<Image Rules>
…
;AddMultiPageTIFF;DAL(PDF_DAL.dal),Only(ODD),TYPE(P);
;AddMultiPageTIFF;DAL(PDF_DAL.dal),Only(ODD),TYPE(P);
…
```

  Assume the first PDF file contains four TIFF images. The second PDF file contains five TIFF images.

  After executing the two rules, there will be five pages in the form. The first four pages will have two TIFF images each (one from the first rule and one from the second) and the final page will contain the last TIFF image from the 5-image PDF file.

  Be aware that the placement of the images on the page can make them overlap.

NOTE: This rule supports long file names on 32-bit Windows operating systems.

Image Editor example

These examples show how you can define the PDF file to import when you use this rule. Assume that your MRL has these sub-directories which contain these PDF files:

Directory	File name
PDF_DAL	T_DAL.PDF
PDF_File	T_FILE.PDF
PDF_GVM	T_GVM.PDF
PDF_SRCH	T_SRCH.PDF

Using the File option with the Type option

This example imports the T_FILE.PDF file from the PDF_File directory. Using this file, the GenData program adds the images in the PDF file to the form set. Each image in the PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the image from the PDF file. If the T_FILE.PDF file does not exist, no error messages appear because the OPT option is set to Yes.

Here is an excerpt from a sample DDT file using the File option:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;File(.\PDF_File\T_File.PDF),Opt(Y,P),TYPE(P);;
…
```

Using the DAL option with the Type option

This example executes the PDF_NAME.DAL DAL script which returns the file name, *F_DAL.PDF*. Using this file name, the GenData program adds the images contained in the PDF file to the form set. Each image in the PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the image from the PDF file. Only the odd-numbered images in the PDF file are included because the Only option is set to *Odd*.

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;DAL(PDF_DAL.dal),Only(ODD),TYPE(P);;
…
```

Using the SCH option with the Type option

This example imports PDF files (*F_SCH1.PDF, F_SCH2.PDF*, and *F_SCH3.PDF*) based on the content of lines in the file designated by the ExtrFile option in the Data control group. Using this file, the GenData program adds the images contained in the three PDF files to the form set. Each image in each PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the image from the PDF file. The images from the PDF files are embedded in the NA file because the Embed option is set to Yes.

Here is an example of the extract file records pointed to by the ExtrFile option:

```
0         1
1         1
SCOxxxxxxxHEADERREC
…
…
PDF_File_Name .\PDF\F_SCH1.PDF
…
…
```

NOTE: This option lets you import and process multiple PDF files because of the way the file name and path are specified—one file per entry in the file pointed to by the ExtrFile option.

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;SRCH(1,PDF_File_Name 15,17),Embed(Y),TYPE(P);;
```

Using the GVM option with the Type option

This example imports a PDF file based on file name contained in the GVM variable called *PDF_File_GVM*. Using the PDF file name and path in the GVM variable, the GenData program adds the images contained in the PDF file to the form set. Each image in the PDF file causes a duplicate of the original FAP file to be appended to the form. This duplicate contains the image from the PDF file.

NOTE: Keep in mind you can use any valid GVM variable, no matter how it is created or assigned.

To create the *PDF_File_GVM* variable, you would include the following INI option in your FSISYS.INI and add its definition in the TRNDFDFL.DFD file.

```
< GenTrnDummyFields >
    PDF_File_GVM = .\PDF_gvm\T_GVM
```

Here is an excerpt from a sample DDT file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;0,0,26400,20400,0,0,0,0;
;AddMultiPageTIFF;GVM(PDF_File_GVM),TYPE(P);;
```

See also AddMultiPageBitmap on page 283

Section and Field Rules Reference on page 274

BldGrpList

Use this field level rule (level 4) to build lists of data you can use to populate section lists or columns during GroupEnd processing for each transition. This rule lets you specify which data will be collected in the list for GroupEnd processing.

Syntax

```
BldGrpList; ListFunction (ListParameter (ListSubParameter)) Rule
(FieldRule) Data (FieldRuleParameters);
```

This rule lets you use one of these list functions:

- Array
- MultiArray
- MultiOccur

For each list function you can specify the *Rule( )* and *Data( )*. These parameters let you specify any standard field level rule that can be used to format the data gathered with this rule, such as the MoveNum and Move_It rules.

NOTE: You must include the following rule in the AFGJOB.JDT file to clear the queues created by the BldGrpList after it processes each transaction:

```
;ProcessQueue;;PostPaginationQueue;
```

For more information, see ProcessQueue on page 184.

Using the Array function

This list function retrieves data from the first extract record encountered that meets the search mask criteria in a transaction. The data defined as an array in the record is used to populate fields on the section. The sub parameters for this function are:

Parameter	Description
Search	Search masks used to locate the extract record which contains the array of data.
Count	The offset and length of the field which contains the number of entries in the array.
Entry	The offset for the start of the array data and length of each data entry in the array.

Here is an example of the Array function:

```
BldGrpList; Array (Search (31, CWIARRAY), Count (39, 2), Entry (41,
5)) Rule (MoveNum) Data ( );
```

Parameter	Description
BldGrpList; Array	Calls the BldGrpList rule using the Array function.
Search(31, 68, 14, 2)	Searches for an extract record with *68* in offset 31 and *2* in offset 14.
Count(39, 2)	The field at offset 39 for a length of 2 in the extract record contains the number of entries in the array.

Parameter	Description
Entry(41, 11)	The array starts in offset 41 in the extract record and each entry is 11 characters long.
Rule(MoveNum) Data( )	The MoveNum rule is called using the parameters defined by *Data ( )*. If there are no *Data* parameters, you do not have to define a *Data* sub parameter.

Here is an example of a record from a transaction:

```
31                              39   41
RG00000028030219281501          CWIARRAY0400.0011.1122.2233.3344.44
```

Using the MultiArray function

Use this function to retrieve data from multiple extract records that meet the search mask criteria. The multiple records define the array data that is used to populate fields on the section. The MultiArray sub parameters are:

Parameter	Description
Search	Search masks used to locate the extract records which contain the array of data.
Count	The offset and length of the field which contains the number of entries in the array.
Entry	The offset for the start of the array data and length of each data entry in the array.

Here is an example of the MultiArray function:

```
BldGrpList; MultiArray (Search (1, CWIARRAY, 14, ~90), Count (38, 1),
Entry (41, 11)) Rule (MoveNum) Data ( );
```

Parameter	Description
BldGrpList;MultiArray	Calls the BldGrpList rule using the MultiArray function.
Search(1, CWIARRAY, 14, ~90)	Searches the extract records for each transaction with *CWIARRAY* in offset 1 and which is not equal to (~) *90* in offset 14.
Count(38, 1)	The field at offset 38 for a length of 1 in the extract record contains the number of entries in the array
Entry(41, 11)	The array starts in offset 41 in the extract record and each entry is 11 characters long.
Rule (MoveNum) Data ( )	Calls the MoveNum rule using the sub parameters defined by *Data ( )*. If there are no *Data* parameters, you do not have to define a *Data* sub parameter.

Here is an example of a record from a transaction:

```
31                              39   41
RG00000028030219281501          CWIARRAY0400.0011.1122.2233.3344.44
RG00000028030219281501          CWIARRAY0499.9688.4534.2176.4504.05
```

Using the MultiOccur function

Use this function to retrieve data from multiple extract records that meet the search criteria for each transition, which is used to populate the fields on the section. The MultiOccur sub parameters are:

Parameter	Description
Search	Search masks used to locate the extract records that contain the data to populate the field.
Field	The offset and length of the data in the extract record.

Here is an example of the MultiOccur function:

```
BldGrpList;
MultiOccur(Search(31,CWICURR,39,~90,39,~95,39,~99),Field(41,38))Rul
e(Move_It)
```

Parameter	Description
BldGrpList;MultiOccur	Calls BldGrpList rule using the MultiOccur function.
(Search(31,CWICURR,39,~90,39,~95,39,~99)	Searches the extract records for *CWICURR* at offset 31 and offset 39 which is not equal to *90*, *95*, or *99* for each transaction.
Field(41, 38)	Collects data from extract record starting at offset 41 for 38 positions.
Rule(Move_It)	Calls the Move_It rule to collect the data.

Here is an example of records from a transaction:

```
31        39 41
RG000000220302192815O1   CWICURR 02   ENERGY CHARGE    4.93
RG000000230302192815O1   CWICURR 03   FIXED CHARGE    14.00
RG000000240302192815O1   CWICURR 04   REVENUE FEE      2.10
RG000000250302192815O1   CWICURR 90      21.03
RG000000260302192815O1   CWICURR 95       1.47
RG000000270302192815O1   CWICURR 99      22.50
```

See also

GroupBegin on page 343

GroupEnd on page 355

Move_It on page 393

MoveNum on page 402

ProcessQueue on page 184

Section and Field Rules Reference on page 274

CanSplitImage

Use this section level rule (level 3) to identify the segments of sections (images) that can be dynamically split.

Syntax

```
;CanSplitImage;Debug;
```

Parameter	Description
Debug	Optional. Include this parameter to tell the system to include debug information about when and how the section is split in the LOG.DAT file.

Keep in mind:

- You must include the PaginateAndPropagate rule in the AFGJOB.JDT file. The PaginateAndPropagate rule looks for the CanSplitImage indicator. Without this rule, sections are paginated normally.
- If the print driver produces output for a non-edge printer, such as PCL, you must have a header and footer that are copied on overflow. Otherwise, the data that falls into the non-print area is lost.
- Text area objects must be flagged as *Can span pages* or the section is not split.
- Text area sections that are greater 26,400 FAP units must be defined as *Custom* in the Paper Type field on the Page Properties tab. To add information to one of these sections, increase the paper height on the Page Properties tab to accommodate the increased size before you add the new information. Otherwise, the system may create a multi-page section, which is not supported.
- Text areas that are multi-page sections are not supported.
- This rule cannot split a section that includes a text area with an inserted file object.
- This rule is not supported when you are using a group rule. The group pagination logic does not check for split sections, nor does it call the section split logic.

This table shows how each type of object is handled within a text area:

Object	Supported
Different fonts	Yes
Borders	Yes.
Background shading	Yes
Bullets or numbers	No
Columns	Yes
Boxes	Yes *
Fields	Yes
Files	No

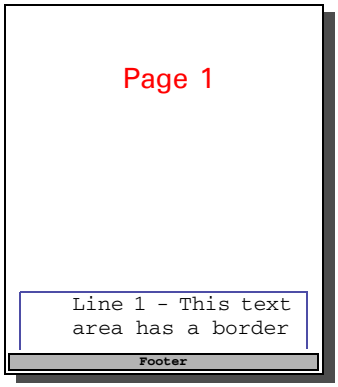
Object	Supported
Graphics (LOG, BMP, TIF, and PNG files)	Yes *
Charts	Yes *
Vectors	Yes *
Shaded areas	Yes
Bar codes	Yes *
Lines	Yes *
Boxes	Yes *

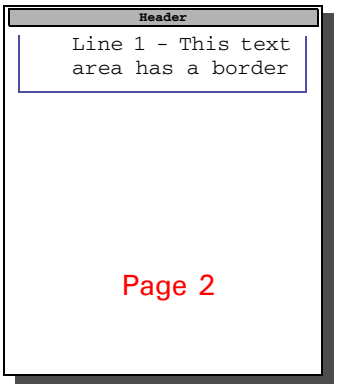
* If the object falls on the page to be split, the system moves it to the next page.

Here is how the system splits the section:

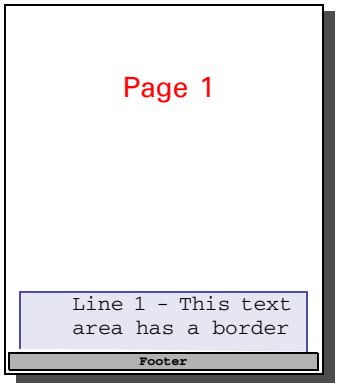
1. The system loads the section and flags it as *inline*.
2. The system duplicates the section. The new section follows the original section.
3. The system moves up the bottom of the original section to just above the footer.
4. The system moves the objects from the original section to the new section if the bottom of the object extends beyond the bottom of the original section.
5. If the object is a text area and is defined as *Can Span*, the system splits the text area, as described here:
   - The system makes sure the original section is not set to *Can Grow*.
   - The system creates the new text area in the new section and copies the object from the original text area into the new text area.
   - The system creates a list of the text areas to divide and repositions all objects moved to the new section. All objects are moved up as much as possible but maintain their relative positions.
   - The system then moves the section down as far as the objects are moved up. This preserves the relative position of the objects.
   - The system turns on the *Can grow and shrink* attribute and turns off the *Can span pages* attributes in the text areas earlier saved in the list. It then resizes the new section to its minimum size and turns the *Can Span* attribute back on. Pagination continues as usual with two sections, instead of one.

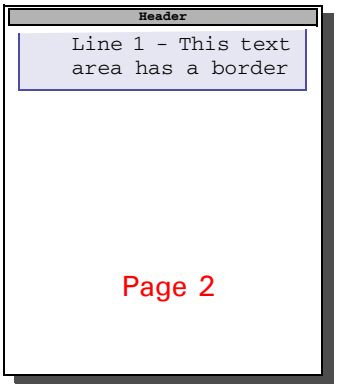
Here are some examples that show what happens when a text area with a border is split:





Shaded areas are split in a similar manner:





See also PaginateAndPropagate on page 174
Section and Field Rules Reference on page 274

CheckImageLoaded

Use this section level rule (level 3) to see if the FAP file is loaded, and if not, load the FAP file. You would typically use this rule if there is information needed in the FAP file that is not present in the DDT file, such as bar code information, or variable field rotation information.

By default, the GenData program loads FAP files. If the LoadCordFAP option is turned on, the GenData program loads all FAP files. Avoid turning on this INI option as it slows performance. For example, make sure this option is set in the FSISYS.INI file as follows:

```
< RunMode >
    LoadCordFAP = No
```

The GenData program should only write information about dynamic data, such as variables, into the NAFILE.DAT file from the DDT files. You can do this more efficiently by loading the DDT files instead of the FAP files.

There are, however, situations which require you to load FAP files. This rule and the TextMergeParagraph rule handle these situations. These rules let you load data for a single FAP file. Keep in mind that the TextMergeParagraph rule affects a *single* FAP file while the LoadCordFAP option affects *all* FAP files.

Since, in some cases, you must load FAP files, the system includes utilities which let you pre-compile FAP files and FXR files. By pre-compiling these files into CFA (FAP) and CFX (FXR) files, you can speed performance by eliminating parsing operations. The system is set up to use pre-compiled FAP and FXR files. You can see this setting in the FSISYS.INI file:

```
< RunMode >
    CompiledFAP = Yes
```

To turn off this setting, change the *Yes* to *No*. For best results leave it set to *Yes*.

NOTE: Using this rule slows performance. Use only as necessary.

Syntax

```
;CheckImageLoaded;;;
```

The CheckImageLoaded rule checks to see if the FAP file associated with the DDT file has already been loaded into memory. If the FAP file has not been loaded, the CheckImageLoaded rule loads it.

Image Editor example

```
;CheckImageLoaded;;;
```

Rotated fields

If you have a section (FAP file) which contains four variable fields, each with a different rotation and the fields are not rotated when you run the GenPrint program, make sure you include the CheckImageLoaded rule. This rule is required in this situation.

Bar code variables

If you are using the EAN (European Article Numbering) system to represent bar code variables and you are using the Move_It rule to map the bar code variable field to your data, include the CheckImageLoaded rule if your LoadCordFAP option is set to *No*.

See also

TextMergeParagraph on page 483

Section and Field Rules Reference on page 274

CompBin

Use this field level rule (level 4) to select the largest value of multiple packed decimal fields located on the same record to populate a variable field.

For each given offset in the data section, the source offset is set and the MoveNum rule is called. The highest value is kept in a buffer. After comparing all numbers, the system copies the highest number into the variable field. There are two optional parameters:

- Compare the largest value with the given value. If it is larger, return it. If smaller, use option two.
- Take the field with the index specified in the next parameter. If the second parameter is not specified, return a SKIP message.

NOTE: The system ignores the fields of the DDT entry which usually contain the source offset and the source length. Instead, the system uses the offset and length specified in the data section. The search criteria and the extract field descriptors must be delimited by a single space. No other spaces are allowed.

Studio example

You could make the following entries in Studio in the Rule section of the Field Options panel:

In this field...	Enter...
Rule	CompBin
Destination offset	1
Source name	REC-MAXFINE
Source offset	45
File	*
Length	6
Record	*
Required	*
Overflow Multiplier	*
Overflow	*
Mask	11.0,18.0,B
Data	100,XYZ 45,4,67,4

* no entry required for this field in this example

The rule compares the packed decimals on the locations 45-4 bytes and 67-4 bytes in the record identified by a XYZ at location 100 and returns the highest value.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	MAXFINE
Offset	1
Length	6
Source name	REC-MAXFINE
Offset	45
Length	4
File	*
Record	*
Required	*
Rule	CompBin
Mask	11.0,18.0,B
Data	100,XYZ 45,4,67,4

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-MAXFINE;45;4;MAXFINE;1;6;11.0,18.0,B;CompBin;100,XYZ
45,4,67,4;;;;;
```

The rule compares the packed decimals on the locations 45-4 bytes and 67-4 bytes in the record identified by a XYZ at location 100 and returns the highest value.

Here's another example:

In this field...	Enter...
Destination name	MAXFINE
Offset	1
Length	6
Source name	REC-MAXFINE
Offset	45
Length	4

File	*
Record	*
Required	*
Rule	CompBin
Mask	11.0,18.0,B
Data	100,XYZ 45,4,67,4 500,1

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-MAXFINE;45;4;MAXFINE;1;6;11.0,18.0,B,CompBin;100,XYZ
45,4,67,4 500,1;;;;;
```

The example compares the packed decimals on the locations 45-4 bytes and 67-4 bytes in the record identified by a XYZ at location 100 and returns the highest value if it is larger than 500. If it is not larger, the system returns the data from the first offset/length pair (45,4).

See also Section and Field Rules Reference on page 274

ConCat

Use this field level rule (level 4) to concatenate two or more text strings contained in an extract record and place the result in a field you specify. This rule supports overflow.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	POL NUMBER
Offset	1
Length	15
Source name	REC-POL NUMBER
Offset	30
Length	12
File	*
Record	*
Required	*
Rule	ConCat
Mask	*
Data	17,PMSP0200 30,3,33,7,40,2

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;REC-POL NUMBER;30;12;POL NUMBER;1;15;;ConCat;17,PMSP0200
30,3,33,7,40,2;;;;;
```

This rule tells the system to get the first occurrence of a record matching the search criteria of PMSP0200 at offset 17. The output consists of 12 characters retrieved from three locations. The 14 character output consists of three characters from offset 30, seven characters from offset 33, and two more characters from offset 40, as well as a single space between the first and second and the second and third fields.

The mask is used for character compression of concatenated fields. The default is one space. You can enter zero (0) or any positive value—just make sure the destination length can accommodate the spacing of the fields.

See also

Section and Field Rules Reference on page 274

ConnectFields

Use this section level rule (level 3) to move and align field text so the data elements appear to be connected.

NOTE: Previously, this rule was known as the ConcatFields rule. You can use either spelling.

Syntax

```
ConnectFields F=FixedField L=LeftField R=RightField;;
```

Parameter	Description
F	Enter the name of the field you want used as the *fixed field*. The system will move the other fields in relation to this field. The system does not move the fixed field. The first field you list in the rule's parameters is always considered the fixed field.
L	Enter the names of the fields you want moved to the left of the fixed field.
R	Enter the names of the fields you want moved to the right of the fixed field.

You must enter a fixed field and at least one field to move to the left or right side of the fixed field. By default, the system places the other fields one space character from the fixed field, unless you indicate that you do not want spacing between the fields.

NOTE: Include the word *No* after the movement parameters (L or R) to tell the system not to add spacing between the two fields. For example,

```
F=FIELD1,RNO=FIELD2
```

tells the system to place the contents of FIELD2 immediately after to the end of FIELD1 with no intervening spaces.

As the system places another field next to the fixed field, the *fixed rectangle* grows. This lets you define additional fields based upon where the last field was added.

Keep in mind...

- This rule does not move fields vertically. Fields are only moved horizontally.
- This rule loads the section (FAP or compiled FAP) if it is not already loaded.
- If you use the Move_It rule, or other rules that support right justification by padding the data with spaces, your results will be incorrect. This rule calculates the width of a field based upon its entire contents and does not remove any white space in the field.
- If you specify a field which does not contain data or is invalid, then no space, or space holder, is included.
- Do not try to move the same field multiple times. The final location of a given field's data is determined by the last movement of that field.
- The field you specify as the fixed field, cannot be included as one of the fields to be moved.

- This rule does not work with barcode or multi-line text fields. If you try to name such a field, you will get an error. This rule does not handle rotated fields.

Image Editor example

For these examples assume FIELD1 contains *ABC*, FIELD2 contains *DEF*, and FIELD3 contains *XYZ*. With this rule in the DDT file:

```
;ConnectFields;F=FIELD1,R=FIELD2;;
```

The result is:

```
ABC DEF
```

With this rule in the DDT file:

```
;ConnectFields;F=FIELD1,L=FIELD2,R=FIELD3;;
```

The result is:

```
DEF ABC XYZ
```

Here, the rule appended FIELD2 to the left side of FIELD1 and appended FIELD3 to the right of FIELD1. Note, that in this example, it is difficult to see that the fixed field, FIELD1, did not move. FIELD2 and FIELD3 moved to align with FIELD1. During this operation, FIELD1 did not move at all.

With this rule in the DDT file:

```
;ConnectFields;FIELD1,LNO=FIELD2,RNO=FIELD3;;
```

The result is:

```
DEFABCXYZ
```

This rule is defined similar to the last one but uses the *No* space parameter.

With this rule in the DDT file:

```
;ConnectFields;F=FIELD1,R=FIELD2,R=FIELD3;;
```

The result is:

```
ABC DEF XYZ
```

Notice there are two fields appended to the right of the fixed field. The first one appended expanded the rectangle which allows the next one to append after the last.

With this rule in the DDT file:

```
;ConnectFields;F=FIELD1,R=FIELD2,F=FIELD2,R=FIELD3;;
```

The result is:

```
ABC DEF XYZ
```

Notice that the result of this rule is the same as the previous example. In this case, the fixed field was changed to FIELD2 after FIELD2 had been moved adjacent to FIELD1. Then FIELD3 was moved adjacent to FIELD2 in its new location.

With this rule in the DDT file:

```
;ConnectFields;F=FIELD1,R=FIELD2,R=FIELD2;;
```

The result is:

```
ABC      DEF
```

This example illustrates one of the earlier cautions. In this case, FIELD2 is defined to move twice. Since the operations are sequential, the field is first moved adjacent to FIELD1. This movement expands the fixed rectangle used by subsequent movements. When the field is named again, it moves relative to the current rectangle, making the field appear farther to the right a distance equal to the size of the text in the field plus the width of two spaces.

See also Move_It on page 393

SpanAndFill on page 470

Section and Field Rules Reference on page 274

CreateChartSeries

Use this section level rule (level 3) to get data from extract records and include it as series data in a chart. The system uses the data exactly as it exists in the extract record. There is a syntax you can use to create a search mask which will search for and get an extract record. You can also specify where in that record the data resides. Additionally, you must specify the chart and series to which the data will be added.

Typically, you will use the DeleteDefaultSeriesData rule, on page 327 *before* you use this rule. You should always use the PurgeChartSeries rule, on page 433 *after* you use this rule.

Syntax

```
CreateChartSeries (Chart)(Series)(SearchMask)(DataDefinitions)
```

Parameter	Description
Chart	The name of the chart (as defined in the FAP file) to which the series data will be added.
Series	The name of the series that data is to be added to. If the series is not defined in the FAP, a default series is created.
SearchMask	The search mask to be used to find the extract record from which data is retrieved. The search mask should contain one or more *offset,data* pairings. For example, if you want to find the extract record with the text HEADERREC at offset 20 (base 1), the search mask would be 20,HEADERREC. Additional offset length pairings can be appended. For example: `20,HEADERREC,50,XYZ` means find the record with HEADERREC at offset 20 and XYZ at offset 50. See the topic Search Criteria on page 270 for more information.
DataDefinitions	One or more *offset,length* pairings used to obtain data values from the extract record defined by the search mask. For example, if five data values existed at offsets 110, 120, 130, 140 and 150 each one 10 characters in length, the DataDefinitions field would look like this: `110,10 120,10 130,10 140,10 150,10.` There is a space between each *offset,length* pair, unlike the SearchMask which has all its *offset,data* pairs separated by a comma (,).

Image Editor example

```
<Image Rules>
;CreateChartSeries;{VBarChart}{Ser1}{11,TESTXXREC,25,2}{30,10 40,10
50,10 60,10};
;CreateChartSeries;{VBarChart}{Ser2}{11,TESTXXREC,25,3}{30,10 40,10
50,10 60,10};
;CreateChartSeries;{VBarChart}{Ser3}{11,TESTXXREC,25,4}{30,10 40,10
50,10 60,10};
;PurgeChartSeries;;
```

This example gets data for three different chart series in the same chart. The chart name is VBarChart and the three data series are Ser1, Ser2, and Ser3.

The series data for series Ser1 is retrieved from the record found with search mask 11,TESTXXREC,25,2. There are four series data values which are added to this series, each one 10 characters in length. These are found at offsets 30, 40, 50 and 60 in the extract record.

The series data values are added to the series, one after another, in the order that they are listed in this rule. Series data for series Ser2 and Ser3 are created in similar manner, but get data from other extract data records.

If multiple data items are contained in a single record, it is best to use the rule as described above where the extract data record is searched for only once.

In the following example, we see the extract data records searched individually. If each series data value is contained in a unique extract data record then there is no choice but to search for each individually. Here is an example:

```
;CreateChartSeries;{VBarChart}{Ser1}{11,TESTAAREC,25,2}{30,10};
;CreateChartSeries;{VBarChart}{Ser1}{11,TESTBBREC,25,2}{40,10};
;CreateChartSeries;{VBarChart}{Ser1}{11,TESTCCREC,25,2}{50,10};
;CreateChartSeries;{VBarChart}{Ser1}{11,TESTDDREC,25,2}{60,10};
;CreateChartSeries;{VBarChart}{Ser2}{11,TESTEEREC,25,3}{30,10};
;CreateChartSeries;{VBarChart}{Ser2}{11,TESTFFREC,25,3}{40,10};
;CreateChartSeries;{VBarChart}{Ser2}{11,TESTGGREC,25,3}{50,10};
;CreateChartSeries;{VBarChart}{Ser2}{11,TESTHHREC,25,3}{60,10};
;CreateChartSeries;{VBarChart}{Ser3}{11,TESTIIREC,25,4}{30,10};
;CreateChartSeries;{VBarChart}{Ser3}{11,TESTJJREC,25,4}{40,10};
;CreateChartSeries;{VBarChart}{Ser3}{11,TESTKKREC,25,4}{50,10};
;CreateChartSeries;{VBarChart}{Ser3}{11,TESTLLREC,25,4}{60,10};
```

See also FieldVarsToChartSeries on page 335

Search Criteria on page 270

Section and Field Rules Reference on page 274

CreateSubExtractList

Use this section level rule (level 3) to process multiple items of the same type within a single transaction. This rule produces a temporary extract list which contains each record of the same type within a transaction.

The temporary extract list is based on the search mask and key you define. The system can then populate variable fields using the data in the temporary extract list.

NOTE: The CreateSubExtractList rule is used with the SetRecipFromImage rule.

Syntax

```
;CreateSubExtractList; Search( ) Keys( );;
;CreateSubExtractList; From( ) To ( );;
;CreateSubExtractList; Drop;;
;CreateSubExtractList; Append;;
```

Parameter	Description
Search	Defines a search criteria; see Search Criteria on page 270.
Keys	This variable defines the scope of the search: `Keys((offset1,length1),(offset2,length2),….(offsetN, lengthN))`
From( ) To ( )	The From(mask) includes the record specified in the from mask. The To(mask) indicates where to stop. The To(mask) does not include the record specified in the To(mask).
Drop	This parameter drops the temporary sub-extract list that was created and removes the records from the cached extract records for that transaction. You will no longer have access to these records.
Append	This parameter adds the temporary sub-extract list that was created to the end of the extract list.

Image Editor example

In the following example, the CRTSUB section triggers the CreateSubExtracList rule. Here is an excerpt from the CRTSUB.DDT file:

```
<Image Rules>
;CreateSubExtractList;Search(1,18) Keys((24,10));
```

The first line creates a list based on the defined search criteria. The *Keys* variable defines the scope of the search. In this example, *Keys=Service Number*. Each time the service number changes, the system begins a new grouping.

Once a list of records has been grouped by this rule, a second form (sub-form) is called and processed against that group of data. A unique key, such as a LOB key, must define each sub-form called by this rule.

The second line in the DDT file defines the key for the sub-form it will call. Key2 triggers the appropriate sub-form. In this example, CRTSUB calls on LOB2. Therefore, each sub-group of data created by CRTSUB will be used to process all of the sections in LOB2. And, unless otherwise specified, the LevelCheck value should always be set to zero.

Here is an example of a FORM.DAT file:

```
;Company;LOB;TEST;;N;;CRTSUB|DS<CUSTOMER(0)>
                       /IMAGE1|DS<CUSTOMER(0)>;
;Company;LOB2;TEST;;N;;IMAGE2|DS<CUSTOMER(0)>
                        /IMAGE3|DS<CUSTOMER(0)>
                       /IMAGE4|DS<CUSTOMER(0)>
                        /IMAGE5|DS<CUSTOMER(0);
```

Like all forms under a separate line of business in a form set, a sub-form called by the CreateSubExtracList must have a trigger in the SETRCPTB.DAT file. You have to make sure you have a unique trigger for the section that calls the CreateSubExtractList rule. Otherwise, you will end up with a repeating group of data (see the sample extract file below.) Here is an example:

```
;COMPANY;LOB;TEST;;;;;0;0;0;0;;
;COMPANY;LOB;TEST;CRTSUB;;CUSTOMER;1,18,9,HEADER,16,ELEC11;1;1;0;1;
;
;COMPANY;LOB;TEST;IMAGE1;;CUSTOMER;;0;0;0;1;1,18,16,ELEC11;
;COMPANY;LOB2;TEST;;;CUSTOMER;;0;0;0;0;;
;COMPANY;LOB2;TEST;IMAGE2;;CUSTOMER;;0;0;0;1;1,18,16,ELEC11;
;COMPANY;LOB2;TEST;IMAGE3;;CUSTOMER;1,18,16,ELEC11;1;1;0;1;;
;COMPANY;LOB2;TEST;IMAGE4T;;CUSTOMER;;0;0;0;1;1,18,16,ELEC11;
;COMPANY;LOB2;TEST;ENDSUB3;;CUSTOMER;;0;0;0;1;1,18,16,ELEC11;
```

NOTE: You must set up the calling section for overflow so the CreateSubExtractList rule is recalled and can group the next set of records. If you do not set up overflow, the system processes only the first group of records for the first key field. Also note the form name *TEST* does not change.

Section	Overflow	Sub-form called
CRTSUB.FAP	Yes	LOB2;TEST

Here is an excerpt of the extract file:

```
01SAMPCO STARTR ELEC01
18SAMPCO HEADER ELEC11 1111111111
18SAMPCO DETAIL ELEC11 1111111111
18SAMPCO DETAIL ELEC11 1111111111
18SAMPCO DETAIL ELEC11 1111111111
18SAMPCO HEADER ELEC11 2222222222
18SAMPCO DETAIL ELEC11 2222222222
18SAMPCO DETAIL ELEC11 2222222222
02SAMPCO END ELEC02
```

The DDT file corresponding to the last section in the sub-form called by this rule must contain either *Drop* or *Append*. In this example the last section is IMAGE5.FAP. This lets the rule knows that it has reached the end of the form. Here is an excerpt from the IMAGE5.DDT file:

```
<Image Rules>
;CreateSubExtractList;DROP;
```

See also SetRecipFromImage on page 466

Section and Field Rules Reference on page 274

DAL

Use this field level (level 4) rule to get information from an extract file if certain conditions are met. In addition, this rule lets you call most of the DAL functions.

NOTE: Version 11.4 changed the way entries that specify a DAL rule are processed in the Extract Dictionary (XDD).

Normally, the ancestry Data fields are appended together to form a complete data representation for a field. When using the DAL rule, this behavior was often undesired. Now, the system assumes that the entry specifying the DAL rule contains all the data information required to resolve the necessary field value and ignores any values specified by an ancestor.

The DAL rule is similar to the IF rule except it returns only data, not another rule. For more information, see If on page 360.

When you use this rule in a DDT statement, end every statement with two colons (::). Instead of writing the statement to the DDT file, you can specify a DAL script name by adding *Call ("scriptname")* in the Data field. In the DDT file, it would look like this:

```
;0;0;DALTEST;0;18;DALTEST;0;18;;DAL;Call ("script.dal");N;N;N;N;
```

It is also possible to specify the rule data for the DAL rule in an external file. Name the file to include with a leading ampersand (&) in the data area, as shown here:

```
;0;0;DALTEST;0;18;DALTEST;0;18;;DAL;&data.inc;N;N;N;N;
```

This loads the file, *data.inc*, from the current directory and inserts its contents into the rule's data area. Note that the file should contain a single line, just as it would appear if you had typed the data directly into the rule data parameter. Also, if the include file is not in the current directory, the name must specify the correct path to locate the file.

NOTE: If the include file is in DefLib, then change &*data.inc* to &*deflib**data.inc*.

If you encounter this error message:

```
DM10558: Error in GetFieldRuleData(): Condition exceeds buffer length
```

This means the content of your include file is too large to fit into the rule data area. To resolve this problem, place the data in a DAL script file and use the CALL or CHAIN command to execute the DAL script.

Image Editor example

Here is an example:

```
;0;0;AUTONUREC-TOT;90;9;TOTAL PREMIUM;0;10;;dal;
$A = {11,AUTONUREC 25,9}::
$B = {11,AUTONUREC 35,9}::
$C = {11,AUTONUREC 45,9}::
$D = {11,AUTONUREC 55,9}::
$E = {11,AUTONUREC 65,9}::
$F = {11,AUTONUREC 75,9}::
$DENOM = 100.00::
$RESULT1 = ($A + $B + $C + $D + $E + $F)::
```

```
$RESULT2 = ($RESULT1 / $DENOM)::
if ($RESULT2=0)::
RETURN("0.0")::
ELSE::
RETURN($RESULT2)::
END::
;N;N;N;N;15373;16426;12012;
```

NOTE: You can use curly braces { } to tell the system to apply a search mask before executing the DAL script. Here is an example:

```
$A = {11,AUTONUREC 25,9}::
```

The use of curly braces is not part of DAL syntax, but rather is a Documaker Server notation that is preprocessed before the DAL script is executed.

Please note that you can only use curly braces in this manner if the DAL script is written into the rule data area. External DAL script files cannot contain such syntax. To retrieve extract data within an external DAL script file, you have to use the GETDATA function.

See also If on page 360

Section and Field Rules Reference on page 274

DateDiff

This field level rule (level 4) lets you display the difference between two dates. The dates do not have to be in the same format in the extract file, but the format of each must be supported. (The valid formats 1,7,10,11 will be described later.)

The dates and their formats are separated by a comma (,) and are defined in the Data field of the rule. For formats supporting a two-digit year, the 20th century (19xx) is assumed for date difference calculations. Date differences as a result of using this rule are not true in the sense that they are calculated under the following guidelines. These are sometimes referred to as *bankers* dates.

- Month =30 days
- Year = 360 days

You must also specify the output format and output format data in the format mask of the rule. (The only valid format and data currently supported is 1 3 / 3.)

Syntax

```
FORMATMASK;rule;RULEDATA;...
```

The FormatMask must be in the form of:

```
OUTPUTFORMAT OUTPUTFORMATDATA
```

where OUTPUTFORMATDATA varies, depending on the output format. You must separate these parameters with a space.

OUTPUTFORMAT. Shows the difference between the two dates, in months and days. The only output format is 1. You must separate these parameters with a space.

OUTPUTFORMATDATA. The OutputFormatData must be in the form of:

```
MONTHLENGTH SEPARATORSTRING DAYLENGTH
```

For example, if you enter **3 / 3** and there are 365 days difference, the output would be *012/005*. Here is an example:

```
1 3 / 3 (1 is OUTPUTFORMAT)
```

The RULEDATA consists of two groups of search criteria and extract field descriptors for the two dates. These two groups of data are separated by a colon (:). Within each group, search criteria and extract field descriptors are separated by a single space. Within search criteria, offset and length are separated by a comma (,). Extract field descriptors are also separated by a comma (,).

The RULEDATA must be in this form:

```
RECOFFSET1,RECDATA1 OFFSET1,LENGTH1,DATEFORMAT1:RECOFFSET2,RECDATA2
OFFSET2,LENGTH2,DATEFORMAT2
```

Parameter	Description
RECOFFSET1	The record offset of first date.
RECDATA1	The search data for record of first date.
0FFSET1	The offset of first date in record.
LENGTH1	The length of first date in record.

Parameter	Description
*DATEFORMAT1	The format of first date.
RECOFFSET2	The record offset of second date
RECDATA2	The search data for record of second date
OFFSET2	The offset of second date in record
LENGTH2	The length of second date in record
*DATEFORMAT2	The format of second date

* The format you specify here must be supported.

You can use these formats:

Enter	For this format:
1	MM/DD/YY
7	YYYY-MM-DD
8	Julian_AD date 1 through 1000034 (01/01/0001 – 12/31/2738)
10	Month Date, Year (such as February 17, 2002)
11	MMDDYYYY

NOTE: The data for the two dates is separated by a colon (:). Here is a RULEDATA example: **1,TRANS 73,10,7:1,TRANS 83,10,7**

Image Editor example

```
;1 3 / 3;DateDiff;1,TRANS 73,10,7:1,TRANS 83,10,7;...
```

NOTE: :The input to the DateDiff rule can be any of the supported date formats. The system converts each format to the Julian_AD date and then returns the difference.

Julian_AD dates are simply the number of days since 1 AD. 1 AD is presumed to be 01/01/0001 because there was no year 0000. That's why the millennium does not actually start until 2001. For example, 730179 is Julian_AD for 02/29/2000.

See also Section and Field Rules Reference on page 274

DateFmt

NOTE: You should use the FmtDate rule, on page 337, instead of this rule. This rule is included in this version of the system only for legacy system support.

Use this field level rule (level 4) to format a date retrieved from an extract record based on the mask you select. A list of date masks appears below. This rule supports overflow.

Date masks

ID	Source	Destination	Also supported by FmtDate
1	YYMMDD	MMDDYY	yes
2	YYYYMMDD	MMDDYYYY	yes
3	YYYYMMDD	MMDDYY	yes
4	YYMMDD	MM-DD-YY	yes
5	YYMMDD	MM/DD/YY	yes
6	YYYYMMDD	MM-DD-YY	yes
7	YYYYMMDD	MM/DD/YY	yes
8	MMDDYY	MM-DD-YY	yes
9	MMDDYY	MM/DD/YY	yes
10	YYYMMDD	MM/DD/YY	yes
11	MMDDYYYY	Month D, YYYY	yes
12	MMDDYYYY	Mon D, YYYY	no
13	MMDDYYYY	MONTH D, YYYY	no
14	YYYY-MM-DD	Month D, YYYY	yes
15	YYYY-MM-DD	Mon D, YYYY	no
16	YYYY-MM-DD	MM/DD/YY	yes
17	DD/MM/YY	Mon D, YYYY	no
18	DD/MM/YY	Month D, YYYY	yes
19	DD/MM/YY	MM/DD/YY	yes
21	YYYY-MM-DD	Month DD, YYYY	no

ID	Source	Destination	Also supported by FmtDate
22	DD/MM/YY	Month DD, YYYY	no
23	MM/DD/YY	Month DD, YYYY	no
24	YYMM	MM/YY	no
25	MM/DDYY	Month DD, YYYY	no
26	DD/MM/YY	Mon DD, YYYY	yes
27	YYYY-MM-DD	MM/DD/YYYY	yes

Destination formats with a single letter, such as *D*, indicate that the system will omit leading zeros or spaces. Also, please note that *Month* indicates both upper- and lowercase letters will be used while *MONTH* indicates only uppercase letters will be used. *Mon* indicates the month will be abbreviated, in upper- and lowercase letters.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	EFFECTIVEDATE
Offset	1
Length	10
Source name	REC-EFFECTIVEDATE
Offset	75
Length	6
File	*
Record	2
Required	*
Rule	DateFmt
Mask	4
Data	17,PMSP0200

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;2;REC-
EFFECTIVEDATE;75;6;EFFECTIVEDATE;1;10;4;DateFmt;17,PMSP0200;;;;;
```

This rule gets the second occurrence of a record in the extract list which has PMSP0200 starting at its 17th character. It then takes the six characters starting at location 75 (which should be of the format YYMMDD since we are using format mask 4). The DateFmt rule will reformat the date to be MM-DD-YY before placing the date in the output buffer.

This example shows the use of a user function and overflow symbol:

In this field...	Enter...
Destination name	EFFECTIVEDATE
Offset	1
Length	10
Source name	REC-EFFECTIVEDATE
Offset	75
Length	6
File	*
Record	2
Required	*
Rule	DateFmt
Mask	4
Data	@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;2;REC-EFFECTIVEDATE;75;6;EFFECTIVEDATE;1;10;4;DateFmt;
@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200;;;;;
```

See also FmtDate on page 337

Formatting Data on page 257

Section and Field Rules Reference on page 274

DeleteDefaultSeriesData

This section level rule (level 3) removes all unnamed series data from the specified set of series in a chart. If there are no series specified, this rule removes all series for the chart.

This rule is designed for the special circumstance where unnamed series data may have been defined for the chart during composition, but this series data is not used in every production. In other words, the series data is dynamically created as the result of running rules such as the CreateChartSeries or FieldVarsToChartSeries rules.

Syntax

```
;DeleteDefaultSeriesData;chart,series,...;;
```

The first semicolon-delimited field contains the rule name. The second semicolon-delimited field should contain one or more comma-delimited items. The first item must always be a chart name. Any successive items must be series names that belong to that chart.

Image Editor example

If you had a chart named MYCHART which contained three different series named SERIES1, SERIES2, and SERIES3, all the following would be valid uses of this rule:

Delete all series data (either method will work)

```
;DeleteDefaultSeriesData;MYCHART;
;DeleteDefaultSeriesData;MYCHART,SERIES1,SERIES2,SERIES3;
```

Delete series data for SERIES1

```
;DeleteDefaultSeriesData;MYCHART,SERIES1;
```

Delete series data for SERIES1 and SERIES3

```
;DeleteDefaultSeriesData;MYCHART,SERIES1,SERIES3;
```

See also

CreateChartSeries on page 315

FieldVarsToChartSeries on page 335

Section and Field Rules Reference on page 274

DelImageOccur

Use this section level rule (level 3) to delete a specific occurrence of a section on a form.

Syntax

```
;DelImageOccur;Occurrence,Form,KillSpace;;
```

Parameter	Description
Occurrence	The section occurrence to delete. Positive numbers indicate the count is from the beginning. Negative numbers indicate the count is from the end.
Form	The form name. If there are multiple sections, separate the section names with commas, such as `form1,form2,form3,` If you specify a form name, the rule only works on those forms. If you omit the form names, the rule affects all forms that include the section.
KillSpace	(Optional) If you include this parameter, the system removes the space after the specified occurrence of the section. If you include this parameter, you must include it before the form name. Here is an example: `;DelImageOccur;1,KillSpace;;` `;DelImageOccur;1,KillSpace, Form1;;` Any form specified before the KillSpace parameter is not affected by this parameter. Consider this example: `;DelImageOccur;1,Form1,KillSpace,Form2,Form3` The first occurrence of the current section in Form1 is removed but the space where this section was placed is not removed. However, the first occurrence of the current section is removed from From2 and From3. Furthermore, the system also removes the space after the specified occurrence of the section.

To use this rule, you must also add the following rule to the AFGJOB.JDT file:

```
;ProcessQueue;;PostPaginationQueue;
```

Add this rule after the RunSetRcpTbl rule. You can omit the rule level number.

Image Editor example

```
;DelImageOccur;3,form1,form2;;
```

This example removes the third occurrence of the current section on form1 and form2.

```
;DelImageOccur;-2;;
```

This example removes the second-to-last occurrences of the current section for all forms.

See also

ProcessQueue on page 184

RunSetRcpTbl on page 212

Section and Field Rules Reference on page 274

DontPrintAlone

Use this section level rule (level 3) if you need to delete a page from a form set and there is only one section on that page.

Syntax

```
DontPrintAlone( )
```

If, when the system determines pagination, it determines the section is the only section on the page, the system omits the page from the printed output.

Image Editor example

Here is an example from the DDT file for the QAIMSGOP.FAP file:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,298,15965,0,600,0,480;;
;SetOrigin;REL+0,MAX+0;;
;PaginateBeforeThisImage;;;
;ResetImageDimensions;MinHeight;;
;DontPrintAlone;;;
```

This example will not include the section QAIMSGOP in the printed output if it is the only section on a page.

See also

PaginateBeforeThisImage on page 421

Section and Field Rules Reference on page 274

EjectPage

Use this field level (4) rule to process multi-page FAP files (sections). With multi-page sections, when the section is added to the form set only one section object is created. When the section gets loaded, the system creates the other sections.

Syntax

```
EjectPage( )
```

This rule makes sure the pre- and post-section level processing is run for each section of the multi-page FAP file, not just the pre-processing on the first section and the post-processing on the last.

When you run the FAP2MET utility on a multi-page FAP file, the utility creates a FAP file for each page. In the DDT file, you should then make an entry for the FAP file and add an EjectPage rule for each additional page that makes up the form. The system knows by the EjectPage rules to look for additional FAP files for this form.

Image Editor example

Here is an example of the rule as it would appear in a DDT file:

```
;0;0;zip;75;5;zip;0;5;;move_it;11,ProdARec;N;N;N;N;16168;5070;11010
;
;;;;;;;;;;EjectPage;;;;;;;;;
;0;0;f p num;0;2;f p num;0;2;;mk_hard;X;N;N;N;N;13925;25842;11006;
```

Image Editor automatically inserts this rule into the DDT file for a multiple page FAP file.

See also Section and Field Rules Reference on page 274

FfSysDte

Use this field level rule (level 4) to get the current date from the system and place that date in the destination field. This rule supports overflow.

There are several output formats you can specify by entering a format ID in the DDT data field. Here's a list of the format IDs you can use:

ID	Output Format
M	month only, output formatted as MM
D	day only, output formatted as DD
Y	year only, output formatted as YY
MS	month spelled out
MS1	month spelled out, followed by the two-digit year. (24 June 02)
MS2	month spelled out, followed by the date and a four-digit year (February 17, 2002)
Default	no option specified in the data field, will use the format MM/DD/YY

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	DATE
Offset	1
length	20
Source name	REC-DATE
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	FfSysDte
Mask	*
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-DATE;;;DATE;1;20;;FfSysDte;;;;;;
```

This example places the current system time stamp in the destination field, according to the default format of MM/DD/YY.

NOTE: Place the format ID immediately *after* the rule. This differs from other rule masks.

See also

SysDate on page 474

Formatting Data on page 257

Section and Field Rules Reference on page 274

Field2GVM

Use this section level rule (level 3) to create a GVM variable from the fields in the current section. For instance, you can use this rule to store the system date for later use.

Syntax

```
Field2GVM; FieldName, GVMName
```

Parameter	Description
FieldName	Name of the field on the current section from which the data is retrieved.
GVM name	Name of the GVM variable in which the retrieved data will be stored.

You can use this rule to output data into one of the batches or the NEWTRN.DAT file if the GVM variable name matches the field name in the DFD file.

GVM variables are essential part of Documaker Server. For example, the fields in the NEWTRAN.DAT file, or in recipient batch records are all GVM variables during runtime. This rule lets you take data from a field and place the data into a GVM variable. If that GVM variable happened to be one of the fields in a recipient batch record, it would be written out to the RCB file.

You can also use the *O* parameter to identify fields the system should consider as optional.To flag a field as optional, include \O at the end of the field name. Here is an example:

```
;Field2GVM;Date\O,CurrentDate,DTE_CLOSED,DTEACCTCLSD;
```

This example will not generate an error if *Date* cannot be located on the section. An error will be generated if *DTE_CLOSED* is missing.

If the system cannot find a field marked as optional, it will not change the destination GVM variable. This behavior supports situations where you map any of several fields that could be generated to the same GVM variable.

Note, that this rule creates a GVM variable if necessary. Therefore, be sure to check the spelling of the GVM variable name if you intend to use one created by a prior process. Otherwise, a new variable is created.

Keep in mind...

- If the GVM variable you specify does not exist, it will be created. This differs from other rules which expect you to have defined the GVM variable by some other means. This means that if you misspell the variable name, you will not get an error because a GVM variable will be created for that name.
- Designating a field as optional does not change the value of the GVM variable if the field is missing. The system does not clear the GVM variable of data just because a field is missing.

Image Editor example

Here is an example from a DDT file:

```
;Field2GVM;BANNER TEXT #002,DATAOUT;;
```

The data contained in the BANNER Text #002 field is retrieved and stored in the GVM variable called DATAOUT.

You can also use this rule to place the system date in the NEWTRN.DAT file. To do this, in the FAP file you create a field, such as System_Date, which will always be triggered. You then add a field name, such as SystemDate in the TRNDFDFL.DFD file. Next, add...

```
Field2GVM,System_Date,SystemDate
```

in the DDT file and use the SysDate rule to place system date in the System_Date field. This will place the system date in the NEWTRN.DAT file.

If you then want to place the system date into archive, define a field, such as DAPDate, in the APPIDX.DFD file and add...

```
< Trigger2Archive >
    dapdat = systemdate
```

Once the system date is in APPIDX.DBF file, you can display it by adding...

```
< AfeArchiveDisplay >
    Field = dapdata,DD%m/%d/%Y
```

See also Section and Field Rules Reference on page 274

FieldVarsToChartSeries

Use this section level rule (level 3) to allow a chart's series data to be retrieved via references to the section's variable fields. Since Documaker Server does no field propagation, as would an entry system, you must handle field propagation.

You can assign a name for the series data in the FAP file definition. You can then use the series data name to associate the series with a variable field with the same name. This rule propagates data mapped to a variable field into series data with the same name. What this rule does that the CreateChartSeries rule cannot is manipulate extract data *before* it is assigned to a series.

Syntax

```
FieldVarsToChartSeries( )
```

Typically, you will need to use the DeleteDefaultSeriesData rule *before* you use this rule. Always use the PurgeChartSeries rule *after* you use this rule, as a means of cleanup. If you find incorrect data in a chart, you may be missing one of these rules.

NOTE: Use Studio or Image Editor to select the fields the system will then use to assign the minimum, maximum, increment/label, and tick mark values. See the Documaker Studio User Guide or the Docucreate User Guide for more information.

Image Editor example

This example uses the Move_It rule, which does not manipulate the extract data. You can also use other rules, such as the MoveNum rule, to scale the data mathematically. For example, if a chart has four series, each with four series data values, you could build it as shown below with data gathered by field level rules. You can then use the mapped variable field data to populate the chart series data fields listed in the FAP file.

Here is how it would look in the DDT file:

```
<Image Rules>
;FieldVarsToChartSeries;;;
<Image Fields>
<Image Field Rules Override>
;0;0;FIELD10;30;10;FIELD10;0;10;;Move_It;11,TESTXXREC,25,1;N;N;N;N;
;0;0;FIELD11;40;10;FIELD11;0;10;;Move_It;11,TESTXXREC,25,1;N;N;N;N;
;0;0;FIELD12;50;10;FIELD12;0;10;;Move_It;11,TESTXXREC,25,1;N;N;N;N;
;0;0;FIELD13;60;10;FIELD13;0;10;;Move_It;11,TESTXXREC,25,1;N;N;N;N;

;0;0;FIELD20;30;10;FIELD20;0;10;;Move_It;11,TESTXXREC,25,2;N;N;N;N;
;0;0;FIELD21;40;10;FIELD21;0;10;;Move_It;11,TESTXXREC,25,2;N;N;N;N;
;0;0;FIELD22;50;10;FIELD22;0;10;;Move_It;11,TESTXXREC,25,2;N;N;N;N;
;0;0;FIELD23;60;10;FIELD23;0;10;;Move_It;11,TESTXXREC,25,2;N;N;N;N;

;0;0;FIELD30;30;10;FIELD30;0;10;;Move_It;11,TESTXXREC,25,3;N;N;N;N;
;0;0;FIELD31;40;10;FIELD31;0;10;;Move_It;11,TESTXXREC,25,3;N;N;N;N;
;0;0;FIELD32;50;10;FIELD32;0;10;;Move_It;11,TESTXXREC,25,3;N;N;N;N;
;0;0;FIELD33;60;10;FIELD33;0;10;;Move_It;11,TESTXXREC,25,3;N;N;N;N;

;0;0;FIELD40;30;10;FIELD40;0;10;;Move_It;11,TESTXXREC,25,4;N;N;N;N;
;0;0;FIELD41;40;10;FIELD41;0;10;;Move_It;11,TESTXXREC,25,4;N;N;N;N;
;0;0;FIELD42;50;10;FIELD42;0;10;;Move_It;11,TESTXXREC,25,4;N;N;N;N;
```

```
;0;0;FIELD43;60;10;FIELD43;0;10;;Move_It;11,TESTXXREC,25,4;N;N;N;N;
```

See also

CreateChartSeries on page 315

DeleteDefaultSeriesData on page 327

PurgeChartSeries on page 433

Move_It on page 393

MoveNum on page 402

Section and Field Rules Reference on page 274

FmtDate

Use this field level rule (level 4) to format dates. Using this rule you can format dates for different localities. This rule supports overflow.

The DDT mask for the FmtDate rule takes these values:

- input fetype
- input format mask
- output fetype
- output format mask

NOTE: There are two types of format masks, pre-defined types 1-9 and A-Q and user-defined format arguments. If the pre-defined formats meet your needs, use them, otherwise, create a user-defined format. For information on using pre-defined format types, see Using Pre-defined Date Formats on page 257.

User-defined format arguments consist of one or more codes, each preceded by a percent sign (%). For more information on user-defined format masks, see the Setting Up Format Arguments on page 262.

You can enter up to 80 characters in the mask.

Image Editor example

Assume the data in extract file is *04/01/2001* (which is fixed type 1 – MM/DD/YYYY) and you want to convert it to April 1, 2001 (which is Mon D, Yr), use this DDT format mask:

```
d,"1/4",d,"4/4"
```

To produce a Canadian French date, add the CAD locality code, as shown here:

```
d,"1/4",dCAD,"4/4"
```

To produce a date such as Apr 1, 2001 (format -- Mon D, Yr which does not fall into any fixed type), use the following DDT format mask.

```
d,"1/4",d,"%b %#d, %Y"
```

Here is another example:

```
;0;0;datefield1;80;6;datefield1;0;12;d,"B",d,"4/
4";FmtDate;11,HEADERREC;N;N;N;;;;;
```

See also

DateFmt on page 324

Formatting Data on page 257

Field Format Types (fetypes) on page 265

Section and Field Rules Reference on page 274

FmtNum

Use this field level rule (level 4) to format numbers. Using this rule you can format amounts for different localities. This rule supports overflow.

The DDT mask area takes these values:

- input fetype
- output fetype
- output format mask

An *fetype* defines the field format type. You can have an input and an output fetype. For example, an input fetype with the FmtNum rule tells the system where the decimal goes in the number. The output fetype tells the system how to format the output amount. An fetype can consist of either one or four characters.

The first character of an fetype defines the field format type. There are several types defined in the system such as a *d* for dates and an *n* for numbers. You can add three additional characters to override the default locale, which is USD (United States, English). Please see Field Format Types (fetypes) on page 265 for a list of the supported localities. If your defined fetype is not one that is supported, the system uses USD.

Image Editor example

For example, if you have this DDT format mask...

```
n,nCAD,"$ZZZ,ZZ9.99"
```

...and the extract data has an amount such as 123,456.78, the system uses this rule to read the amount and then create 123456.78. The system then formats this number (123456.78) to produce the result, 123 456,78$.

Here is another example. If you use this DDT format mask...

```
nCAD,nUSD,"$ZZZ,ZZ9.99"
```

...and the extract data has a Canadian French amount such as 123 456.78$, the system uses this rule to read the number and then create 123456.78. The system then formats this number (123456.78) to produce the result in US dollars, $123,456.78.

Left justifying numbers

You can include the L parameter to left justify numbers after they have been formatted. This parameter is the fourth optional flag in the DDT mask area. For example, if you have this DDT format mask

```
n,n,"z,zzz,zzz,zzz,zz9.99",L
```

and the extract data has an amount such as 123456.99, the system uses this rule to read the amount and formats this number to produce the result, *123,459.99*. The rule will then left justify the field. Below is an example of the result with and without the L parameter.

Without the L parameter:

```
n,n,"z,zzz,zzz,zzz,zz9.99"
          123,456.99
```

With the L parameter:

```
n,n,"z,zzz,zzz,zzz,zz9.99",l
123,456.99
```

Like the Move_It and MoveNum rules, this type of left justification simply removes leading spaces. It does not provide a *positional* justification as is provided by the JustFld rule.

See also

Using Pre-defined Numeric Formats on page 261

Section and Field Rules Reference on page 274

Suppressing Decimals with the FmtNum Rule on page 261

Using the ZeroText Option with the FmtNum Rule on page 262

JustFld on page 367

GlobalFld

Use this field level rule (level 4) to speed the processing of fields used repeatedly throughout a form set.

Frequently a field rule is called to retrieve the same record over and over, which slows batch processing. Using this rule helps you avoid unnecessary repetition and therefore speeds processing.

NOTE: The name of the field in the FAP file (the field name in the XDB file) and the source name of the field in the DDT file (the source name in the XDB file) must be identical for this rule to work correctly.

Syntax

```
GlobalFld( )
```

To use this rule, you must have an XDB.DBF file, which is similar to a data definition table (DDT) file. You set up the XDB and DDT files as shown in the examples below.

The Record field column in an XDB record can be just about anything. If you are importing the fields from a DDT file, the system defaults the record field to the name of the DDT file you are importing.

If you import the fields from a COBOL Copybook, the record column field is assigned the name of the higher level group that owns the field.

Keep in mind that you cannot use the GlobalFld rule if the field data is not going to be global in scope. For instance, overflow and sub-extract situations where you expect the next occurrence of the field to get a different result are not candidates for the GlobalFld rule.

The SourceFile record member is not used in looking up XDB records. XDB records are looked up by the destination field name alone.

NOTE: When using the GlobalFld Rule, the XDB record replaces the entire DDT record. Remember, it is a *global* field, therefore the assumption is that all DDT references to the field are identical.

If you want to make *common* rule definitions, but override certain members in individual DDT files, use the Master DDT Editor instead of the GlobalFld rule.

Image Editor example

Assume the InsuredName field is used many times throughout the form set and the name of the form set is *QMDC1*. Make these entries in the XDB.DBF file:

Field	Enter
Source Name	InsuredName
Source File	QMDCL.DDT
Record	QMDCL
Offset	25
Length	30
Field Name	InsuredName
Required	Not
Rule	Move_It
Mask	
Data	11,INSNAMREC

And make these entries in the QMDC1.DDT file, using the Edit DDT tab of the field's Properties window in Image Editor:

Field	Enter
Destination Name	InsuredName
OffSet	0
Length	30
SourceName	InsuredName
OffSet	0
Length	0
File	0
Record	0
Required	Not
Rule	GlobalFld
Mask	
Data	

When the system executes this rule, it first checks the Dictionary rule by the field name key. If the record exists, it returns the value—here InsuredName. If not, it looks into XDB and gets the original field rule, such as Move_It.

Then the system executes the field rule to get the value and returns it. Finally, it stores the record in the dictionary. The next time that record is required, the system gets the value from the dictionary.

NOTE: After you run the GlobalFld rule, you must run the Dictionary rule to terminate the XDB and free memory.

See also Dictionary on page 79

Section and Field Rules Reference on page 274

GroupBegin

Use this section level rule (level 3) to define the first section in a group of sections. A *group* is a set of sections delimited by a *begin section* and an *end section* that is processed as a single unit.

Using this rule, you specify which sections are grouped on the printed pages. Each GroupBegin rule must have a corresponding GroupEnd rule. With these rules you can:

- Expand boxes to surround a section group with user-defined margins
- Keep a group of sections together on a page
- Paginate vertically with headers, footers, and overflow sections at the group level
- Paginate horizontally with left and right margins that can contain lists
- Format fields with currently used rules
- Create nested groups
- Vary row heights by the tallest field size or set a standard height for all rows
- Pre-define the spacing between rows
- Set a minimum number of lines to be left on the first or last page
- Create a columnar layout

Syntax

```
GroupBegin;GroupFunction(parameters(sub parameters))
```

The group functions include:

- Box
- GroupPagination
- List
- StayTogether
- Column

Using the Box Function

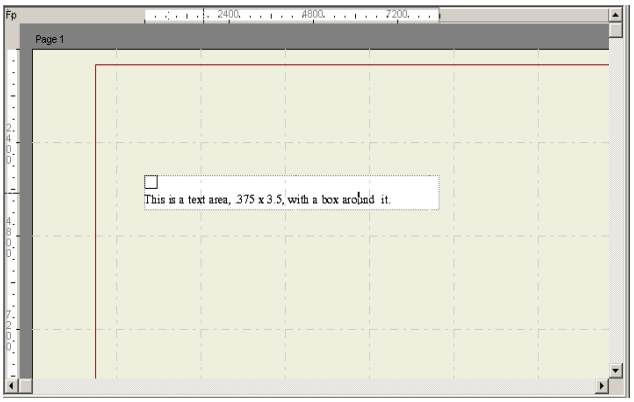
Use this function to expand the first box defined in the group to fit around all sections in the group. The *Margin* parameter lets you define the extra space to be added between the edge of the section and the box edges. The *Margin* sub parameters are:

Parameter	Description
Left	Left margin size in FAP units
Top	Top margin size in FAP units
Right	Right margin size in FAP units
Bottom	Bottom margin size in FAP units

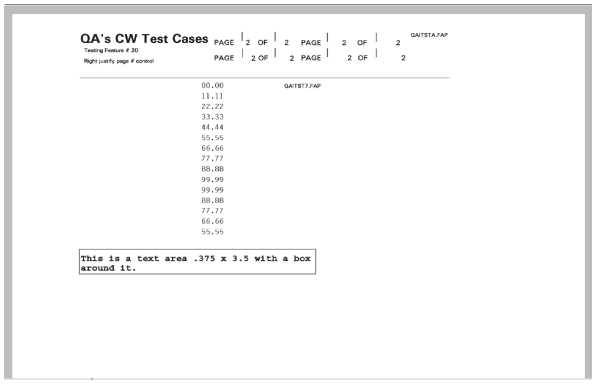
Here is an example:

```
;GroupBegin; Box(Margin(20,20,20,20));;
```

The section would look like this:



The section must include a box that will be expanded by the GroupBegin rule's Box function around the text, as shown below:



This example expands the box around a section group and sets the margin to 20 FAP units between the outer edge of the section and the outer edge of the box. There are 2400 FAP units per inch.

Using the GroupPagination Function

Use this group function to define the requirements for keeping certain sections (groups) together on pagination. The GroupPagination parameters are:

Parameter	Description
MinImagesOnCurrent	Defines the minimum number of sections required on the current page. The default is zero (0). This rule counts all sections triggered in the group, even if a section has no size. It totals the section sizes to determine the minimum number of sections which can be placed in the remaining space on the page. If a section has no size or is flagged as *view only*, the section is placed on the page.
MinImagesOnNext	Defines the minimum number of sections required for new and next page. The default is one (1).
NeverSplit	Requires that all sections within the group must remain together on same page—pagination can never occur within the group. The default is No.
CheckNextPage	Requires that the next page be checked to confirm that the entire group cannot fit on the next page before splitting can occur. The default is No.

The following example requires that a minimum of two sections appear on the current page, and a minimum of three sections appear on any subsequent pages. This example also requires that the next page be checked to confirm that the entire group cannot fit on the next page before splitting can occur. In addition, the second section is defined as the header for the group and is to be copied on overflow. Plus the fourth section is defined as the footer for this group.

Here's an excerpt from the DDT file for the first section:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,936,19718,0,0,0,0;
;SetOrigin;Rel+0,Max+100;
;GroupBegin;GroupPagination(MinImagesOnCurrent(2),MinImagesOnNext(3
));
… … …
```

Here's an excerpt from the DDT file for the second section:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,1142,19718,0,0,0,0;
;SetOrigin;Rel+0,Max+100;
;SetGroupOptions;header,copyonoverflow;
```

Here's an excerpt from the DDT file for the third section:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,357,19699,0,0,0,0;
;SetOrigin;Rel+0,Max+100;
… … … …
```

Here's an excerpt from the DDT file for the fourth section:

```
/* This section uses these rules */
<Image Rules>
```

```
;SetImageDimensions;98,0,621,6124,0,0,0,0;
;SetOrigin;Rel+0,Max+100;
…. … … …
;SetGroupOptions;footer;
;GroupEnd;;
```

Using the List Function

A list is a column of data on a section that is defined as a single field in the data definition table (DDT) and is populated by the BldGrpList rule.

The List function works with the BldGrpList rule to print sections containing lists, or columns, side by side in rows. The tallest field in the row and the GroupBegin:List parameters, *MinSpacing* and *AddSpacing*, determine the row height. The List sub parameters are:

Parameter	Description
AddSpacing	Adds additional spacing in FAP units between rows of data. There are 2400 FAP units per inch.
MinSpacing	Defines the minimum size in FAP units for each row of data.
MinLines	Defines the minimum number of lines to be printed on the first page of a section. When pagination occurs, if the number of lines printed on the first page is less than the *MinLines* amount, the entire section is moved to the second page.
MinLinesCont	Defines the minimum number of lines to be printed on the last page of a section. When pagination occurs, if the number of lines printed on the last page is less than the *MinLinesCont* amount, lines are taken from the preceding page to meet the minimum.

Here is an example:

```
GroupBegin;List (MinSpacing(800) AddSpacing(200) MinLines(12)
MinLinesCont(5) );;
; GroupBegin;List(AddSpacing(65));;
```

This example causes 65 FAP units to be inserted between the groups of sections.

Using the StayTogether Function

Use this function if you do not want the group of sections to be split between pages and overflow onto a new page if there is room on the current page for the entire group. The dimensions of the group of sections *cannot* be larger than the dimensions of the page.

NOTE: Also keep in mind that you cannot nest a StayTogether with a column to keep the column section together. If you try to use a StayTogether over all the sections you want to organize into columns, the results will not be what you expect.

Here is an example:

```
;GroupBegin;StayTogether;;
```

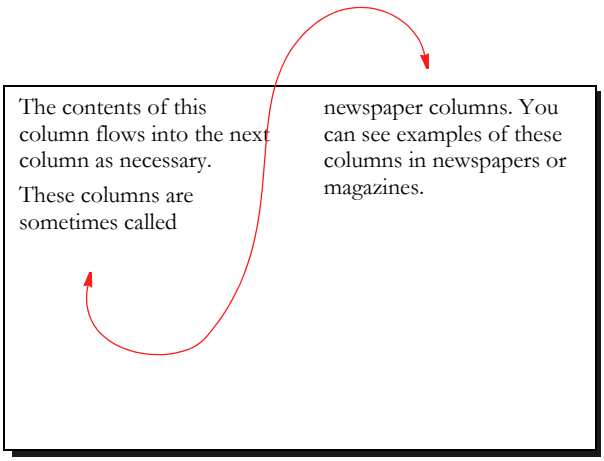
This example keeps a group of sections together when overflow forces them onto a new page.

Using the Column Function

Use the Column group function to create wrapping or and straight columns.

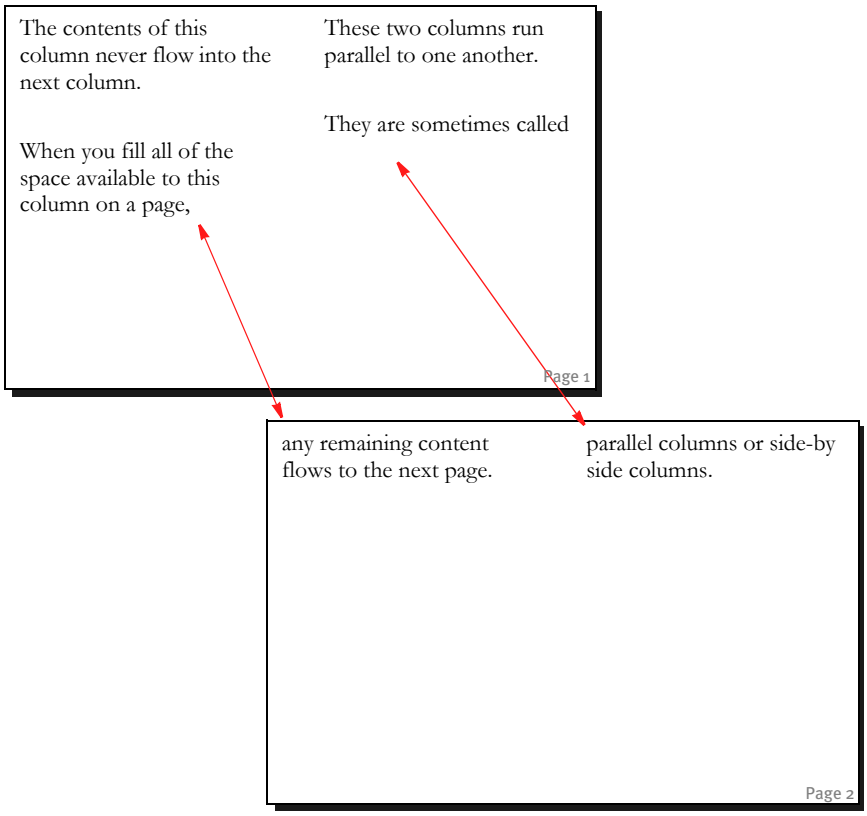
Creating wrapping columns

Use the Wrap parameter to create newspaper style columns where the column contents flow from the top of a column to the bottom and then to the top of the next column. All columns have the same width and the same amount of space between them. There are a fixed number of columns on the page.



Creating straight columns

Use the Straight parameter to create columns whose contents do not flow from one column to the next. Instead, these columns are not connected and run parallel to one another. Straight columns are paginated independently. If the contents of one column exceed the page, the remaining contents appear in that same column on a second page. All the usual overflow, header, and footer considerations still apply.



Column function parameters

Here is a list of the parameters you can use with the Column function.

Parameter	Description
Wrap	Indicates the text in the columns will wrap. No other parameters are required. Wrapping is done by default, unless you use the Multiple or Straight parameter.
Straight	Indicates the text will not be wrapped from one column to the next. No other parameters are required. The section definition controls the width and separation of the columns. When you use straight columns, you define the starting columns with a GroupBegin and the ending column with an GroupEnd.

Parameter	Description
Balanced( )	The balanced sub-parameters determine how sections are processed if there is less than a full page of sections. The default is Left. *Left* Use this sub-parameter to equally divide the sections between the columns on the page. If there is a remainder, the left most column will be the longest column. *Unbalanced* Use this sub-parameter to add sections to a column until there is no more room in that column on that page. The system then places remaining sections in the second column of that same page. The system repeats this process until all columns on the page are filled. The system then places any remaining sections in the first column of the next page and continues filling the columns in this manner.
ColCount( )	Defines how many columns will be on a page. You must enter a positive number.
ColSeparation( )	Defines, in FAP units, how much space is between columns. You must enter a positive number. The default is zero. There are 2400 FAP units per inch.
ColWidth( )	Defines, in FAP units, the width of each column. If you omit the width, the system uses the width of the widest section in the group. You cannot enter a negative number. There are 2400 FAP units per inch.
Single	Indicated there will be a single straight column on the page. Single is the default unless you specify Multiple.
Multiple	Indicates there will be multiple straight columns on the page. You must embed the straight groups within a group. You do this using the Multiple group parameter.
Debug	Use this parameter to write column-processing information into the log file for debugging purposes.

Keep in mind that...

- Wrap and Straight are mutually exclusive.
- Multiple and Single are mutually exclusive.
- Straight is mutually exclusive with Balanced, ColCount, ColWidth, and ColSeparation.
- Multiple is mutually exclusive with Wrap, Straight, Balanced, ColCount, ColWidth, and ColSeparation.
- You cannot nest a Wrap within a Wrap, Straight, or Multiple.
- You cannot nest a Straight within a Wrap or Straight.
- You cannot nest a Straight within a Multiple.
- You cannot nest a Multiple within a Multiple.

Example 1: Wrapped balanced columns

Assume you have a long list of narrow sections which you want to flow down the page until they reach the bottom of the page. The next section should appear at the top of the second column on that page.

In addition, you want the sections to move over to the right so they do not overlap the sections in the first column and you want to repeat this layout until there are three columns of sections.

The columns should be 6000 FAP units wide, balanced as much as possible, and separated by 1/4 inch (600 FAP units) gap. Each section is 1800 x 5200 FAP units. Section margins are 600 FAP units for the top, bottom, left, and right. The text area is 4000 FAP units in width and can grow and shrink. Up to two hundred characters of data can be moved into each text area.

Based on this criteria, here is an excerpt from the DDT file for the first section:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;…
;SetOrigin;Rel+0,Max+0
;GroupBegin;Column(Wrap Balanced(Left) ColCount(3) ColWidth(6000)
ColSeparation(600));
…
```

Here are excerpts from the DDT file for the second section:

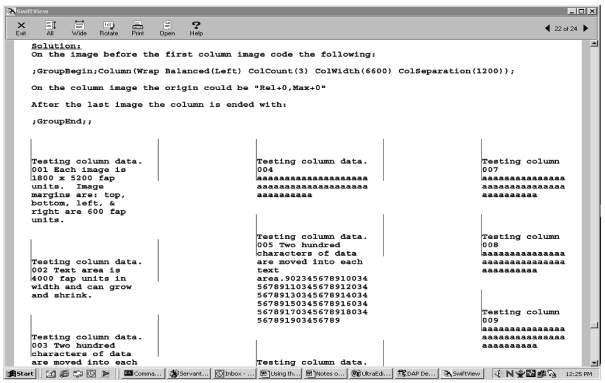
```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;…
;SetOrigin;Rel+0,Max+0;
;IncOvSym;QAICOL2ASYM,QAICOL2A;
;TextMergeParagraph;;
….

/* The following fields override the lower level definitions for this
section only.*/
<Image Field Rules Override>
;0;1;Column Input Area A;40;200;Column Input Area A; 0;200;;Move_It;
@GETRECSUSED,QAICOL2A,QAICOL2ASYM/
31,MOVEITA;N;N;N;N;650;1020;12110;
```

After the last section, the column ends with:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;…
;SetOrigin;Rel+0,Max+0;;
;GroupEnd;;;
```

This is how the section should look:



Example: 2 - Multiple straight columns

Assume you have three columns of sections which you want to output as straight columns. When either of the columns runs into a page footer or off the bottom of the page, you want the sections continued on the next page.

The data in the extract file is contained in three separate groups of continuous overflow records, which may have up to two hundred characters of information.

Column anchor point	Column anchor point	Column anchor point
Record 1, column 1	Record 1, column 2	Record 1, column 3
Record 2, Column 1	Record 2, Column 2	Record 2, Column 3
Record 3, column 1	Record 3, column 2	Record 3, column 3
Record 4, column 1	Record 4, column 2	Record 4, column 3
Record 5, column 1	Record 5, column 2	Record 5, column 3
---	---	---
End of column 1	End of column 2	End of column 3

Based on this criteria, here is a sample solution:

```
/* Excerpt from the DDT file for the first column anchor point. */
<Image Rules>
;SetImageDimensions;98,0,2098,6600,600,600,600,600;
;SetOrigin;Abs+0,Abs+0,,Store(VAR1);
;GroupBegin;Column(Debug Multiple );
;GroupBegin;Column(Straight);

/* Excerpt from the DDT file for the second column anchor point. */

<Image Rules>
;SetImageDimensions;98,0,2098,6600,600,600,600,600;
;SetOrigin;VAR1.right+600,VAR1.top+0,,Store(VAR2);;
;GroupBegin;Column(Straight);

/* Excerpt from the DDT file for the third column anchor point. */

<Image Rules>
;SetImageDimensions;98,0,2098,6600,600,600,600,600;
;SetOrigin;VAR2.right+600,VAR2.top+0;;
;GroupBegin;Column(Straight);
```

```
/* Excerpt from the DDT file for the first column section*/

<Image Rules>
;SetImageDimensions;98,0,1749,6634,600,600,600,600;
;SetOrigin;Abs+0,Max+0;
;IncOvSym;QAICOL2C11SYM,QAICL2C1;
;TextMergeParagraph;;;

<Image Fields>

<Image Field Rules Override>
;0;1;Column Input Area C1;40;200;Column Input Area
C1;0;200;;Move_It;@GETRECSUSED,QAICL2C1,QAICOL2C11SYM/
31,MOVEITC1;N;N;N;N;733;917;16010;

/* Excerpt from the DDT file for the second column section */

<Image Rules>
;SetImageDimensions;98,0,1749,6634,600,600,600,600;
;SetOrigin;VAR1.right+600,Max+0;;
;IncOvSym;QAICOL2C21SYM,QAICL2C2;
;TextMergeParagraph;;;

<Image Fields>

<Image Field Rules Override>
;0;1;Column Input Area C2;40;200;Column Input Area
C2;0;200;;Move_It;@GETRECSUSED,QAICL2C2,QAICOL2C21SYM/
31,MOVEITC2;N;N;N;N;733;917;16010;

/* Excerpt from the DDT file for the third column section */

<Image Rules>
;SetImageDimensions;98,0,1749,6634,600,600,600,600;
;SetOrigin;VAR2.right+600,Max+0;;
;IncOvSym;QAICOL2C31SYM,QAICL2C3;
;TextMergeParagraph;;

<Image Fields>

<Image Field Rules Override>
;0;1;Column Input Area C3;40;200;Column Input Area
C3;0;200;;Move_It;@GETRECSUSED,QAICL2C3,QAICOL2C31SYM/
31,MOVEITC3;N;N;N;N;733;917;16010;

/* Excerpt from the DDT file for the End of the first column. */

<Image Rules>
;SetImageDimensions;98,0,1749,6634,600,600,600,600;
;SetOrigin;Abs+0,Max+1200;
;GroupEnd;;

/* Excerpt from the DDT file for the End of the second column. */

<Image Rules>
```

```
;SetImageDimensions;98,0,1749,6634,600,600,600,600;
;SetOrigin;VAR1.right+600,Max+1200;;
;GroupEnd;;

/* Excerpt from the DDT file for the End of the third column. */

<Image Rules>
;SetImageDimensions;98,0,1749,6634,600,600,600,600;
;SetOrigin;VAR2.right+600,Max+1200;;
;GroupEnd;;;
;GroupEnd;;;
```

Here is an excerpt from the FORM.DAT file:

```
;CWNG;CIS;QaiColD;Testing
StraightColumns;N;;QAICLST1|DS<Customer(1)> /
QAICL2C1|DSW<Customer(1)>/EndCol1|DSW<Customer(1)>/QAICLST2|DS
<Customer(1)>/QAICL2C2|DSW<Customer(1)>/EndCol2|DSW<Customer(1)> /
QAICLST3|DS<Customer(1)>/QAICL2C3|DSW<Customer(1)>/EndCol3|DSW
<Customer(1)>;
```

Here is an excerpt from the SETRCPTB.DAT file:

```
;CWNG;CIS;QaiColD;;;Customer;;0;0;0;0;11,HEADER,31,030167994401;
;CWNG;CIS;QaiColD;QAICLST1;;Customer;;0;0;0;1;31,MOVEITMC1;
;CWNG;CIS;QaiColD;QAICL2C1;;Customer;31,MOVEITC1;1;0;0;1;;
;CWNG;CIS;QaiColD;EndCol1;;Customer;;0;0;0;1;31,MOVEITMEND;
;CWNG;CIS;QaiColD;QAICLST2;;Customer;;0;0;0;1;31,MOVEITMC2;
;CWNG;CIS;QaiColD;QAICL2C2;;Customer;31,MOVEITC2;1;0;0;1;;
;CWNG;CIS;QaiColD;EndCol2;;Customer;;0;0;0;1;31,MOVEITMEND;
;CWNG;CIS;QaiColD;QAICLST3;;Customer;;0;0;0;1;31,MOVEITMC3;
;CWNG;CIS;QaiColD;QAICL2C3;;Customer;31,MOVEITC3;1;0;0;1;;
;CWNG;CIS;QaiColD;EndCol3;;Customer;;0;0;0;1;31,MOVEITMEND;
```

Keep in mind these requirements and restrictions when defining groups:

- Each GroupBegin section must have a corresponding GroupEnd section. Either of these sections can be a blank section.
- If you are using conditional sections, make sure the triggers in the SETRCPTB.DAT file for the GroupBegin and GroupEnd sections are the same.
- Group footers do not have to be defined as the first section as form footers in the form definition file (FORM.DAT).
- Do not use an absolute Y coordinate for a group header or group footer.
- When a section contains both a GroupBegin rule and a SetGroupOptions rule, the GroupBegin rule must come first.
- When a section contains both a GroupEnd rule and a SetGroupOptions rule, the SetGroupOptions rule must come first.
- Variable field data inside overflow group header sections will propagate to the new page during group pagination if the field scope is set to Form.

You must set all group pagination section options (footer, header, and copyonoverflow) using the SetGroupOptions rule.

See also GroupEnd on page 355

BldGrpList on page 301

SetGroupOptions on page 439

Section and Field Rules Reference on page 274

GroupEnd

Use this section level rule (level 3) to define the last section in a group of sections. This rule triggers the formatting of gathered data into section lists, or columns. Vertical pagination occurs while the system processes this rule.

Syntax

```
GroupEnd( )
```

There are no parameters for this rule.

Image Editor example

```
;GroupEnd;;
```

This example causes the group of sections defined with a GroupBegin rule to be ended and triggers the formatting for those sections.

See also

GroupBegin on page 343

SetGroupOptions on page 439

Section and Field Rules Reference on page 274

HardExst

Use this field level rule (level 4) to place a value into a field only if a record is found in the extract data using the search criteria you specify in the data field. This rule supports overflow.

Syntax

```
HardExst ( )
```

For instance, you could use this rule to see if there is a record in an extract file that corresponds to a field designating whether or not the applicant is a home owner. If the data exists in the extract record, the rule could then place an X in the Home Owner field.

You can use these format flags:

Flag	Description
C	Center
R	Right justify

The system justifies the data by adding spaces in front of the text. If you are using a proportional font, do not use these flags to align the data. Use the JustFld rule for that.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	CHECK BOX
Offset	*
Length	1
Source name	REC-CHECK BOX
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	HardExst
Mask	*
Data	17, PMSP0200 X

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;REC-CHECK BOX;;;CHECK BOX;;1;;HardExst;17,PMSP0200 X;;;;;
```

This example puts an *X* into the destination buffer for the field named CHECK BOX if a record is found in the extract list using the search criteria of 17,PMSP0200.

NOTE: Source offset and length do not apply to this rule.

Keep in mind that the HardExst rule, when working with overflow, *does not* return data in the same order the search criteria appears in the extract file.

For example, suppose you want to return the value X in a variable field called CHECKBX based on the search criteria:

```
11,AUTOREC,40,CHECKBX
```

The variable field is set up for overflow.

```
;0;1;CHECKBX;0;0;CHECKBX;0;0;;hardexst;@GETRECSUSED,OVFSYM1,MYIMAGE
/11,AUTOREC,40,CHECKBX  X;;;;
```

In the extract file, there are five occurrences of 11,AUTOREC. The first, third, and fifth occurrence of 11,AUTOREC does not have the value CHECKBX at offset 40 but the second and fourth occurrences of 11,AUTOREC do have CHECKBX at offset 40.

```
SCO12345678AUTOREC
SCO12345678AUTOREC                CHECKBX
SCO12345678AUTOREC
SCO12345678AUTOREC                CHECKBX
SCO12345678AUTOREC
```

The system does not leave the first, third, and fifth occurrences of the variable field blank and populate the second and fourth occurrence.

The system finds the two occurrences of 11,AUTOREC,40,CHECKBX, populates the first two occurrences of the variable field with the value X, and leaves the last three occurrences of the field blank.

Search masks and overflow

Before version 10.1, the HardExst rule did not support overflow. Overflow affects how the search mask is used. Keep in mind that the rule uses the *entire* search mask, not just part of it. In this way, the HardExst rule differs from the PrintIf rule.

For example, if you specify a search mask like

```
11,DETAILREC,28,Y
```

it appears that the system checks to see if the record contains a Y in the 28th position. Instead, this mask really tells the system to find a row with *DETAILREC* at offset 11 and with a *Y* in the 28th position. This may sound like the same thing, but it is not.

Before the rule supported overflow, the answer could only be Yes or No—either you have such a record in your extract file or you don't. For example, suppose you have these rows in an extract file:

```
HEADERREC0
DETAILREC0Y
```

And, suppose you specify the HardExst rule without overflow and with this search mask

```
1,DETAILREC,11,Y
```

In this case, the answer would be Yes—you do have such a record.

Now suppose you have these rows in your extract file and you are still not using overflow with the rule:

```
HEADERREC0
DETAILREC0N
DETAILREC0Y
```

The HardExst rule, even without overflow, will find the record that matches the search mask. Therefore, the answer is still Yes—you do have a row with *DETAILREC* at offset 1 and a *Y* in offset 11.

If you introduce overflow the result does not change. There is still only one record that has *DETAILREC* at offset 1 and a *Y* in offset 11. The first overflow variable will have the value you assign, while all the rest will not.

Although it may seem like you are searching for *DETAILREC* and you want to know if there is a *Y* in the 11th position, this is not what you are specifying. You are specifying that a row must have *both* criteria to match.

That is the difference between the PrintIf rule (which also now supports overflow) and the HardExst rule. For the PrintIf rule, you would use a less specific search mask of *1,DETAILREC* and then use the *if* part of the rule to determine if the row contains the value you want. There are two records that match the less specific search mask of *1,DETAILREC*. The first does not have a *Y* in the designated position, but the second does.

What you have to note is that the HardExst rule, like any other rule, uses the entire search mask to find matching rows—not just part of the mask. Therefore, only use the HardExst rule in overflow conditions to determine how many matching rows are found—rather than to try to find out if a row does or does not match the criteria.

How data is returned

The HardExst rule with overflow does not return data in the same order that the search criteria appears in the extract file. For example, suppose you would like to return the value *X* in a variable field named *CHECKBX* based on this search criteria:

```
11,AUTOREC,40,CHECKBX
```

The variable field is set up for overflow.

```
;0;1;CHECKBX;0;0;CHECKBX;0;0;;hardexst;@GETRECSUSED,OVFSYM1,MYIMAGE
/11,AUTOREC,40,CHECKBX  X;;;;
```

In the extract file, assume there are five occurrences of 11,AUTOREC. The first, third, and fifth occurrence of 11,AUTOREC does not have *CHECKBX* in offset 40 but the second and fourth occurrence of 11,AUTOREC does have *CHECKBX* at offset 40.

```
SCO12345678AUTOREC
SCO12345678AUTOREC                          CHECKBX
SCO12345678AUTOREC
SCO12345678AUTOREC                          CHECKBX
SCO12345678AUTOREC
```

The system does not leave the first, third, and fifth occurrences of the variable field blank and populate the second and fourth occurrence. The system finds the two occurrences of 11,AUTOREC,40,CHECKBX, populates the first two occurrences of the variable field with *X*, and leaves the last three occurrences of the field blank.

See also

JustFld on page 367

Mk_Hard on page 388

Search Criteria on page 270

Section and Field Rules Reference on page 274

If

Use this field level (level 4) rule to get information from an extract file if certain conditions are met. In addition, the IF rule lets you call most of the DAL functions. For more information, see the FieldRule function in the DAL Reference.

When you use this rule in a DDT statement, end every statement with two colons (::). Instead of writing the statement to the DDT file, you can specify a file name by adding **&filename** in the Data field. In the DDT file, it would look like this:

```
;&filename;
```

The IF rule supports the FieldRule function to call every field rule in the IF rule. The FieldRule function requires as many parameters as are required for a field level rule. Not all fields must contain data, but you must include the correct number of delimiters.

Here is a list of field rule parameters. An asterisk indicates the parameter is generally required, depending on the rule you are using. If you leave a parameter blank, be sure to include two colons as delimiters (::) to indicate the parameter is blank.

Parameter	Description
File number	(required by TblLkUp)
Record number	(required for overflow)
Source field name	(required by TblText)
Source field offset	*
Source field length	*
Destination field name	*
Destination field offset	
Destination field length	*
Format mask	*
Field rule name	*
Rule parameters	* (also called “data”)
Flag1	(also called “not required”)
Flag2	(also called “host required”)
Flag3	(also called “operator required”)
Flag4	(also called “either required”)
X position	

Parameter	Description
Y position	
Font ID	

The IF rule and overflow

You can use overflow variables if the field-level rule you used supports overflow. Generally, the IF rule does not support overflow—it can only be supported through the use of the FieldRule function. Here is an example. Suppose you want to move multiple lines of text from *N* number of specific external extract records to the output buffer when the HEADERREC record (at offset 11) contains an *F* in position 1.

For this scenario, you could use the FieldRule function to call the MoveExt rule and use the standard IF rule to do the rest. The DAL script for this example would look like this:

```
CON={11,HEADERREC 1,1}:: A=FIELDRULE("::0::1::E::45::4::PREM/OPS
RATE1::0::4:::moveext::@GETRECSUSED,QCPVR5,OVSYM1/
11,CLSSCDREC::N::N::N::N::::::::"):: if(CON='F')::return("^" & A &
"^")::end ;N;N;Y;N;12461;2119;16010
```

Overflow variables used in the search mask have a syntax which looks like this:

```
@GETRECSUSED,CPDEC1,CPDEC1OVF/11,CLSSCDREC
```

Writing DAL scripts

Keep in mind that writing DAL scripts is like coding rules. You must write the script using the correct syntax and make sure you correctly handle the variables you use. This table shows the different types of variables:

Type	Description
A	String variable. Use quotation marks for comparisons with this variable.
$A	Numeric variable with decimal places. Omit the quotation marks and include only numbers.
#A	Numeric variable without decimal places. Omit the quotation marks and include only numbers.

In an IF condition, the data type of the variable on the left side of the operator determines the data type used during the comparison. This means the variable/number/string on the right is converted to the data type of the variable/number/string on the left. After this conversion occurs, the comparison is performed.

If you encounter this error message:

```
DM10558: Error in GetFieldRuleData(): Condition exceeds buffer length
```

Use this statement:

```
CALL("logo.dal")
```

NOTE: You can use curly braces { } to tell the system to apply a search mask before executing the DAL script. Here is an example:

```
$A = {11,AUTONUREC 25,9}::
```

The use of curly braces is not part of DAL syntax, but rather is a Documaker Server notation that is preprocessed before the DAL script is executed.

Please note that you can only use curly braces in this manner if the DAL script is written into the rule data area. External DAL script files cannot contain such syntax. To retrieve extract data within an external DAL script file, you have to use the GETDATA function.

For more detailed information on writing DAL scripts and using DAL functions, see the DAL Reference.

Examples

Here are some examples which will help you understand how you can use the IF rule. While the IF rule can be very useful, its use affects performance. For production purposes, you may want to use other rules and triggers to perform the same tasks.

Image Editor example 1

Suppose you want to print the town and state information on the form only if a record PRODADREC at offset 11 contains a string of four characters of *0000* starting at position 20. The town and state information is stored in the extract file record *PRODADREC* starting at position 65 for 25 characters. You want to trim off any trailing spaces or characters for the town and state information.

You can define a variable to be used in the IF rule with the following syntax:

```
{search criteria  variable attribute}
```

The search criteria and the variable attributes are separated by one space. The search criteria is a series of offset and data pairs. You can have as many pairs as you want to define, as long as they refer to the same record.

The variable attributes are offset and length for the variable you want to define. The offset of the variable must be for the same record as the search criteria.

The return statement does not support the DAL function. Therefore, the variable must be trimmed first, before it can be used in the return statement. For this example, we will define two variables, *A* and *B*:

```
A={11,PRODADREC 20,4} and B={11,PRODADREC 65,25}
```

The complete script for the IF rule in the semicolon-delimited field would look like this:

```
::A={11,PRODADREC 20,4}::B={11,PRODADREC 65,25}::
if(A='0000')::B=TRIM(B)::RETURN("^" & B & "^")::END::
```

Image Editor example 2

Suppose you want the transaction sent to WIP when the record PRODAREC, at offset 11, contains a string of four characters (*0000*) starting at position 20. And, you always want the system to get 25characters of data from PRODAREC, starting at position 65. Furthermore, you want the system to remove any trailing spaces.

For this scenario, you would use the FieldRule DAL function to call the KickToWIP rule and use the standard IF rule to do the rest. The script for this example would look like this:

```
::A={11,PRODAREC 20,4}::B={11,PRODAREC 65,25}:: if(A='0000')::
FIELDRULE("::0::1::TOWN_STATE::55:9:REC-TOWN_STATE::0::25::::
KICKTOWIP::::N::N::Y::N::3001::5602::11010::")::END::B=TRIM(B)::
RETURN("^" & B & "^")::END::
```

Image Editor example 3

For a record where there *SUN* at offset 23; if *00308* is found in this record at offset 54 and *LIFE* at offset 80, then put *Indiana* into the variable named *StateCode*, else put *OutOfState* into the variable.

The DDT file should look like this:

```
;0;0;StateCode;;;StateCode;;10;;if;A={23,SUN 54,5}::B={23,SUN
80,4}::if((A='00308') AND
(B='LIFE'))::RETURN("^Indiana^")::ELSE::RETURN("^OutOfS
tate^")::END::;;;;;;2128;11592;5;
```

(Syntax for A or B in this example is "offset,data offset,length")

Image Editor example 4

For a record with *SUN* at offset 23; if *00308* is found in this record at offset 54, then put *Indiana* into the variable named *StateCode*, else if *00400* is found, then put *Georgia* into it, else put *OutOfState* into it.

The DDT file should look like this:

```
;0;0;StateCode;;;StateCode;;10;;IF;A={23,SUN
54,5}::IF(A='00308')::RETURN("^Indiana^")::
elseif(A='00400')::RETURN("^Georgia^"):ELSE::RETURN("^OutOfState^")
::END ::;;;;;2128;11592;5;
```

Image Editor example 5

For a record where there is a *SUN* at offset 23; if *00308* is found in this record at offset 54 and *LIFE* at offset 80, then put the data from the offset 63 for 8 bytes in the record where *MOON* is found at offset 23 into the variable called *StateCode*.

The DDT file should look like this:

```
;0;0;StateCode;;;StateCode;;10;;IF;A={23,SUN 54,5}::B={23,SUN
80,4}::IF((A='00308') AND (B='LIFE'))::RETURN("^{23,MOON
63,8}^")::END::;;;;;;2128;11592;5;
```

Image Editor example 6

This example shows how to specify the occurrence of a record.

In every place where you want to use variable data from the extract file, specify a string like the one shown here:

```
{1,MIS257 138,10-5}
```

In this string, *1,MIS257* is the search mask. If the record is found, the function takes ten characters starting from position *138*. The *-5* tells the function to search for the fifth occurrence of the 1,MIS257 record. If you omit the *-5*, the function searches for the first occurrence.

Image Editor example 7

This example shows you how to use the Trim function with the IF rule:

```
;0;0;TOWN AND STATE;65;25;TOWN AND STATE;0;40;;IF;A={11,PRODADREC
20,4}::B={11,PRODADREC 65,25}::IF(A='0000')::B=TRIM(B)::RETURN("^"
& B & "^")::END::;N;N;N;N;
```

Note that the Trim function does not work in the Return statement. Also, make sure you define a variable first, and then trim the variable after the IF statement and before the Return statement. This is the only way it works.

NOTE: Similarly, the JCenter function works the same way.

Image Editor example 8

If you are using the Trim function in an IF rule and you can get either the first or second portions of the Return statement, but not both, this example may help you understand how the system works.

For instance, the following statement…

```
Y={1,MODELDES 94,45}::if((A='ACV') AND B='1'))::Y=TRIM(Y)::return
("{1,VUNITNBR 112,20} + {1,VUNITNBR  81,10}")::elseif((A='APV') AND
(C='1')) or ((A='APA') AND  (C='1'))::return("^"&Y&"^ + {1,MODELDES
49,45}")::end;N;N;N;N;2719;9699;16008
```

…returns the value of *Y* only.

There are two types of errors in this DDT file example. One was a logic error in the IF rule. The *Y* was trimmed in the IF branch where *Y* was never used. The elseif branch used *Y*, but the *Y* was not trimmed there.

The other errors were syntax errors. The correct syntax is as follows:

```
Y={1,MODELDES 94,45}::if((A='ACV') AND (B='1'))::X={1,VUNITNBR
112,20}::Z={1,VUNITNBR 81,10}::return("^" & X & "" & Z &
"^")::elseif((A='APV') AND (C='1')) or ((A='APA') AND
(C='1'))::Y=TRIM(Y)::return("^" & Y & "^+{1,MODELDES
49,45}")::end::;N;N;N;N;2719;9699;16008
```

This example tells the system that…

- if you have a variable A=ACV and a variable B=1, then trim the data from the record, where there is VUNITNBR at position 1, for 20 bytes starting at position 112 and the data from the same record for 10 bytes starting at position 81.
- if (variable A=APV and variable B=1) or (variable A=APA and variable C=1), then trim the data from the record, where there is MODELDES at position 1, for 45 bytes starting at position 94, and combine this data with the data from the same record for 45 characters starting at position 49.

The system does not allow ("{offset,data offset,length} + {offset,data offset,length}") in one Return statement. You must define the variables first and use the variables in the Return statement, as shown above.

Image Editor example 9

This example shows you how to format a date from YYYYMMDD to MM/DD format. To do this, you would need to include the IF rule in your DDT file. The syntax is shown here:

```
;IF;A={11,POLICYREC 21,8}::B=( sub(date2date( A, "D4"), 1,
5))::return("^" & B & "^")::end::;
```

A={11,POLICYREC 21,8} is just an example. The syntax is…

Variable Name = {offset,data offset,length}

In this example, it means that variable A is equal to 8-byte characters starting at position 21 for a record in the extract which meets the search criteria of the string of *POLICYREC*, which is found at position 11.

Image Editor example 10

If, in the extract file, you have a date in this format…

```
YYYYMMDD
```

…and would like to reformat it as…

```
MM/DD/YYYY
```

…here is how to do it:

NOTE: Format options 1 through 27 provide many format choices you can use, but these options truncate the year into a two-digit year, such as MM/DD/YY.

No format in the DateFmt rule handles this task, but you can use the IF rule to reformat the date. Assuming that the date in the EXTRFILE.DAT file is located at offset 77 in a record with FMMDECREC at offset 11, here is the syntax for the IF rule:

```
 IF;A={11,FMMDECREC 77,8}::B=(sub(date2date(A
,"D4"),1,10))::return("^" & B & "^")::END::;
```

Image Editor example 11

To format a salutation so you can have the text *Dear*, a space, and then a variable field followed by a comma, use this syntax:

```
If; A={1,Criteria 10,5}::RETURN("^"&TRIM(A)&","&"^")
```

The result is something similar to:

```
Dear XXXX,
```

Image Editor example 12

This example shows how the IF rule can support overflow by using the FieldRule within the IF rule. In this case, the FmtNum rule is used to get the locale and Trim is used to left-justify the text.

NOTE: In this example the locale is the Netherlands, so the text is in Dutch.

Since the FmtNum rule uses double quotes "-ZZZ,ZZZ,ZZZ.ZZ", the FieldRule must use single quotes (' ') to avoid parsing errors. Note the semicolons (;) within the FieldRule are replaced with double colons (::).

Assume the section name is IMGNAME and overflow symbol is OVFSYM:

```
;0;0;KORTOT-Totaalkorting-Bed;92;10;KORTOT-Totaalkorting-
Bed;0;0;;if;val=fieldrule('::0::0::KORTOT-Totaalkorting-
Bed::92::10::KORTOT-Totaalkorting-Bed::0::0::n,nNLG,"-
ZZZ,ZZZ,ZZZ.ZZ"::fmtnum::@GETRECUSED,IMGNAME,OVFSYM/
18,KORTOT::N::N::N::N::0::0::0')::return("^"&TRIM(val)&"^");N;N;N;N
;0;0;0;
```

See also

DAL on page 320

Section and Field Rules Reference on page 274

IncOvSym

Use this section level rule (level 3) to increment an overflow variable. The overflow variable is a counter that tracks of the number of overflow values processed for a section. This overflow variable is incremented as records are processed and as the overflow increases.

This rule increments the overflow symbol you specify in the data field. Use the ResetOvSym rule to reset the variable for the next transaction that might overflow.

Syntax

```
;IncOvSym;OVERFLOWVAR,IMAGENAME,X;;
```

Use the X parameter to limit the IncOvSym rule to a single execution and also determine when the rule is executed. You control the execution of the rule by including the X parameter, as shown here:

Parameter	Description
X	(Optional) Here you specify when the system should execute the IncOvSym rule by choosing one of these options: F - Tells the system to execute the IncOvSym rule after the first page. L - Tells the system to execute the IncOvSym rule after the last page. 1-9 - Tells the system to execute the IncOvSym rule after the corresponding page. For instance, enter 3 to tell the system to execute the rule after the third page. Keep in mind you can enter only one option. If you enter more than one character, the system evaluates the first character and ignores the rest.

If the requesting section is not a multi-page section, the system ignores this parameter. If you enter a character other than F, L, or 0-9, an error message appears. If you enter zero (0), nothing happens because there is never a page zero.

NOTE: Be sure to thoroughly test your environment when you use this parameter. Different results are created depending on the number of pages in the FAP file, the type of overflow, when pagination occurs, and type of data fields on the different pages in the FAP file.

Image Editor example

```
;IncOvSym;OVERFLOWVAR,IMAGENAME;;
```

This rule increments the overflow variable OVERFLOWVAR by 1 for the form called IMAGENAME.

See also

Overflow and User Functions on page 271

PurgeChartSeries on page 433

SetImageDimensions on page 457

ResetOvSym on page 438

Section and Field Rules Reference on page 274

JustFld

Use this field level rule (level 4) to justify (left, right, or center) a variable field by modifying its field coordinates.

Syntax

```
JustFld (Mode,Cord,XPos,Achr,Rota,Font,NoClip,Rule)
```

This rule calls either the Move_It, MoveNum, FmtDate, FfSysDte, MoveSum, ConCat, TblLkUp, SAPMove_It, MoveExt, FmtNum, TblText, Mk_Hard, StrngFmt rules, or other similar rules.

The first parameter used by the JustFld rule must be the Mode parameter. Here is a discussion of the parameters:

Parameter	Description
Mode	Enter **L** (left), **R** (right), or **C** (center). If you omit this parameter, this rule will call the Move_It rule and generate an error message. You must also include the Cord or Xpos parameter. The other parameters are optional. The mode parameters (left, right, and center) tell the system to remove leading and trailing blanks before it justifies the data. Use the NoClip parameter if you do not want the system to do this.
Cord	Enter the top, bottom, left, and right coordinates to define where the field appears on the page. In Studio, you specify a field's coordinates on the Field Properties window. In Image Editor, you use the General tab of the Properties window. The coordinates are specified as shown here: • The top coordinate is specified in the Y field. The bottom coordinate is the entry in the Y field, plus the entry in the Height field. • The left coordinate is specified in the X field. The right coordinate is the entry in the X field, plus the entry in the Width field. In the DDT file, each coordinate must be in FAP units (2400 per inch) and separated by commas. Here is an example: `CORD=113,441,5714,9314`
XPos	Enter the X coordinate used to align the field. If Mode=R this will be the right most position of the field, likewise if Mode=C this will be the center of the field. Here is an example: `MODE=R,XPOS=5000` If the data is 12345, character 5 will be at position 5000.
Achr	Enter a string of characters found in the data and used to align the field. When you use this parameter, you *must* define the XPos parameter, otherwise the system ignores the Achr parameter. With the correct setup, the rule aligns the field so the characters you specify in this parameter overlay the XPos. You can include up to 10 characters in the string. Here is an example: `MODE=R, XPOS=5000, ACHR=.` If the data is 123.45, the decimal will be at position 5000.

Parameter	Description
Rota	Specifies the field rotation. Enter **0**,**90**,**180**, or **270**. For example: `ROTA=270` This tells the system to rotate the field 270 degrees.
Font	Specifies the font ID. Here is an example: `FONT=11010`
NoClip	Tells the system not to remove trailing spaces from the field.
Rule	Enter the name of the rule you want to use to load the data from the extract file. If you omit this parameter, the system uses the Move_It rule. You can choose from any of the date function rules, such as FmtDate, FfSysDte, MoveSum, ConCat, TblLkUp, SAPMove_It, MoveExt, FmtNum, TblText, Mk_Hard, StrngFmt, and so on. For instance, enter **MoveNum** to have the JustFld rule call the MoveNum rule. This lets you use the formatting capabilities of the MoveNum rule. Here is an example: `MODE=R,XPOS=5000,RULE=MoveNum,FONT=11010`

Be sure to separate the parameters in the data area with commas.

Essentially, you first define the necessary information as though you were not using the JustFld rule, but were going to use the underlying rule to get the data. Then, at the end of the list of parameters, add the Mode parameter and follow with any other JustFld rule parameters you need—including the Rule parameter if you want to use a rule other than Move_It.

Errors

If the call to the rule fails, this rule returns an error. If you omit the Mode parameter, this rule calls the Move_It rule and generates an error. The Mode parameter separates the portion of the data parameter passed to the specified rule.

Using the LoadCordFAP option

Use of the LoadCordFAP option also affects the JustFld rule. The following table shows how this INI option affects this rule:

If set to...	Then...
Yes	The system gets the coordinates (Cord), font (Font), and rotation (Rota) from the FAP file.
No	You must define the coordinates, font, and rotation using the Cord, Font, and Rota parameters. Otherwise the field will not be positioned properly.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	qaij10ab
Offset	0

In this field...	Enter...
Length	15
Source name	qaij10
Offset	21
Length	15
File	*
Record	*
Required	*
Rule	JustFld
Mask	Optional, see the discussion of the rule you specified.
Data	11,JUSTRIGHTA,MODE=R,CORD=12800,13288,5000,10760,FONT=11016,ROTA=0,CLIP,XPOS=5000,ACHR=.

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;qaij10;21;15;qaij10ab;0;15;;JustFld;11,JUSTRIGHTA,MODE=R,CORD=
12800,13288,5000,10760,FONT=11016,ROTA=0,CLIP,XPOS=5000,ACHR=.;N;N;
N;N;5000;13288;11016;
```

This example shows all possible parameters, as they would appear in the DDT file.

Here are some other examples of how to use the JustFld rule:

This example shows how the DDT file would look if you enter a mask and data value for the MoveNum rule:

```
'12.2,12.2,M,S,R,Z,SLZ,T'
'43,Gval NegText("" R"),mode=L,FONT=16116, ROTA=0, Rule=MoveNum'.
;0;0;k1;53;12;k1;0;15;12.2,12.2,M,S,R,Z,SLZ,T;JustFld;43,Gval
NegText("" CR"), mode=L,FONT=16116,ROTA=0,RULE=MoveNum;
N;N;N;N;4999;2500;16116;
```

NOTE: The data value for the MoveNum rule must follow the search mask for the JustFld rule. Include a space to separate the parameters.

Here is an example of how you would use a DDT line to map a field using the Move_It rule:

```
;0;0;FIELD;30;10;FIELD;0;10;;move_it;11,TVBR2DREC,25,1;N;N;N;N;1977
;3763;11006;
```

If the then decided to right-justify the data using the JustFld rule. Simply change the DDT line as shown here:

```
;0;0;FIELD;30;10;FIELD;0;10;;JustFld;11,TVBR2DREC,25,1,MODE=R,XPOS=
5000;N;N;N;N;1977;3763;11006;
```

Notice that most of the line did not change. You simply changed the rule name from Move_It to JustFld and appended the JustFld parameters, starting with the Mode parameter. In this example, the Rule parameter was omitted because the default rule used by JustFld is Move_It.

Now assume you are using the DDT line to map a numeric field using the MoveNum rule:

```
;0;1;PRM;25;9;PRM;0;;12;9.2,12.2,C;movenum;11,AREC;N;N;N;N;13082;34
72;12012;
```

If you decide that you want the result of this to be right-justified, you could change the line as shown here:

```
;0;1;PRM;25;9;PRM;0;;12;9.2,12.2,C;JustFld;11,AREC,MODE=R,RULE=Move
Num,XPOS=14500;N;N;N;N;13082;3472;12012;
```

Again notice that most of the line did not change. You simply changed the rule from MoveNum to JustFld and then included the Rule=MoveNum parameter after the Mode=R parameter.

If you want the number to be decimal-aligned over the X position instead of right-justified, include the ACHR parameter.

Here is an example of using a rule other than the Move_It and MoveNum rules:

```
;1;0;TBL;0;48;TBL;0;48;;TblLkUp;11,HDRREC 53,1 1,CLCODE,12
20,11;N;N;N;N;7831;24236;12012;
```

Although the TblLkUp rule has more information in the rule parameter area than the typical Move_It or MoveNum rule might have, the same approach is used to convert to using the JustFld rule:

```
;1;0;TBL;0;48;TBL;0;48;;JustFld;11,HDRREC 53,1 1,CLCODE,12
20,11,Mode=R,Rule=TblLkUp,XPOS=11000 ;N;N;N;N;7831;24236;12012;
```

The JustFLd parameters are appended to the end of the existing rule parameters, starting with Mode. Then the Rule parameter names the original rule (TblLkUp) and the other JustFld rule parameters further control the resulting output.

This example demonstrates how to use the JustFld rule in an overflow situation using the @GetRecsUsed function.

```
<Image Rules>
          ...
;IncOvSym;JOVF,QAIJUST1;
          ...
          ...
<Image Field Rules Override>
;0;1;qaij1;53;15;qaij1;0;15;;JustFld;@GETRECSUSED,QAIJUST1,JOVF/
43,JUSTRIGHT,MODE=R,FONT=16116,ROTA=0,CLIP,XPOS=5000,ACHR=.;N;N;N;N
;724;727;16114;
```

See also

ConCat on page 311

FfSysDte on page 331

FmtDate on page 337

FmtNum on page 338

Mk_Hard on page 388

Move_It on page 393

MoveExt on page 399

MoveNum on page 402

MoveSum on page 411

SAPMove_It on page 443

StrngFmt on page 472

TblLkUp on page 476

TblText on page 478

Overflow and User Functions on page 271

Section and Field Rules Reference on page 274

KickToWip

Use this field level rule (level 4) to force a transaction to manual batch (WIP). You can use the KickToWIP rule for situations when data is not available in the extract file or the data changes, requiring entry by a data entry operator. This rule makes those fields available for entry.

Syntax

```
KickToWip( )
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	ADDR1
Offset	*
Length	*
Source name	INSNAMEREC-INSNAME1
Offset	*
Length	*
File	*
Record	*
Required	Operator
Rule	KickToWip
Mask	*
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;INSNAMEREC-
INSNAME1;0;0;ADDR1;0;0;;KickToWip;;N;N;Y;N;87;1406;12010
```

The KickToWip rule tells the system to set the manual batch flag to true. Also, to edit the field associated with the KickToWip rule in the Entry module of Documaker, you must set the Required field to Operator.

In this example, the operator required flag for the field INSNAMEREC-INSNAME1 must be set if this field is to be editable in the entry system when retrieved from manual batch.

Suppressing Warning Messages

Use the ShowWIPWarning option to suppress the Sent to Manual Batch warning messages:

```
< RunMode >
    ShowWIPWarning = No
```

Option	Description
ShowWIPWarning	Enter No to suppress warning messages included the error logs when using the KickToWIP or POWType rules, or the KickToWIP DAL function. The default is Yes, which tells the system to include the messages in the error logs.

See also Section and Field Rules Reference on page 274

LookUp

Use this field level rule (level 4) to take data from an extract record. Next, use the data as a key name to look up the key data in a table. Then, copy the table data to the destination field. You must specify the offset of the key name in the data field, as well as the offset and length of the key data in the data field. This rule uses the same table as the MovTbl rule.

Syntax

```
LookUp( )
```

You can use one or more files to keep the tables used by this rule. You must list each table (file) in the TABLEFILE.DAT file. The table list file must be in the following format:

```
TABLEFILENAME1.EXT <crlf>
TABLEFILENAME2.EXT <crlf>
```

where each table is listed on a single line followed by a carriage return/line feed.

The format of the tables is key name followed by key data. The key need not be a specific length. The data can also be any length, which allows a single table or group of table files to contain table entries of varying lengths.

You specify the table list file using the TblFile option under the Data control group in the FSISYS.INI file.

For example, suppose a form contains the names and numbers of agents for calling purposes, but these names and numbers change on a regular basis. For this situation you could create a text table called AGENTS.TBL which contains entries such as...

```
AGENT001 JOE MILLER <crlf>
```

and another table, called AGENTPHO.TBL, with phone number entries such as...

```
AGENT001PHONE404 111-2222 <crlf>
```

You could then make these tables available to Documaker Server by including them in the files specified in the Data control group of the FSISYS.INI file, as shown here:

```
< Data >
    TblFile=.\deflib\TblFile.Dat
```

You must load the tables into memory before the system can use them. To do this, include these rules in the AFGJOB.JDT file:

```
;CreateGlbVar;1;TblLstH,PVOID;
;LoadTblFiles;1;;
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	TEXT01
Offset	1
Length	80
Source name	REC-TEXT01

In this field...	Enter...
Offset	100
Length	5
File	*
Record	2
Required	*
Rule	LookUp
Mask	*
Data	17,PMSP0200 1 14,80

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;2;REC-TEXT01;100;5;TEXT01;1;80;;LookUp;17,PMSP0200 1 14,80;;;;;
```

This example searches for the second extract record matching the search criteria of 17,PMSP0200. It then takes the five characters at offset 100 in the extract record and uses them as a key name into the table data.

The table data is searched for a match with the five characters from the extract record starting at offset 1, the first character of each table entry. If a match is found, the key data of 80 characters starting at offset 14 are copied into the destination field.

See also MovTbl on page 413

TblLkUp on page 476

Section and Field Rules Reference on page 274

MapFromImportData

Use this field level (level 4) rule to map imported data from an internal dictionary to a field, as opposed to mapping from an extract file. Normally, you use this rule with either the ImportFile or ImportExtract rules, however, you can use this rule with any preceding rule that fills in field dictionary values.

By default, this rule checks for a dictionary value starting with the section dictionary, then the form dictionary, and finally the form set (global) dictionary. The search ends as soon as the rule finds a value for the field.

If no dictionary entry is found for the field, the field remains blank. Use the Required flags in the rule definition to control whether an empty field is considered an error.

NOTE: For some legacy implementations, this rule was registered under the name *NoOpImp*.

If you do not use this rule, you must use the ReplaceNoOpFunc rule and make sure that all of the fields for each DDT file are set to NoOpFunc.

Syntax

```
MapFromImportData( )
```

You can use the optional INDEX parameter to specify a particular dictionary instance of the field to use. This is only useful if you use the ImportFile or ImportExtract rules to import form set data.

For the ImportFile rule, to support field instances you must include this INI option:

```
< ImportFile >
    IndexDuplicateFields = Yes
```

For the ImportExtract rule, to support field instances you must include this INI option:

```
< ImportExtract >
    IndexDuplicateFields = Yes
```

Normally, duplicate field entries found in the import file are ignored. If, however, you enter Yes, each instance is stored in a separate dictionary entry.

NOTE: Field instance (indexing) only applies to field data stored in the form set (global) dictionary.

The parameter value to specify indexing has this syntax:

```
INDEX(option,...)
```

Where *option* is a keyword that indicates how to calculate the dictionary instance to use or a constant value to use as the index.

Here is a list of the keywords you can use:

Keyword	Description
FORM	Use the occurrence of this form as the instance index. For example, if this field is contained on the second copy of this form, then get the second instance of field data.
IMG	Use the occurrence of this section within the current form as the instance index. For example, if this field is contained on the third copy of section on the current form, then get the third instance of field data.
IMGFSET	Use the occurrence of this section within the entire form set as the instance index. For example, if this is the fifth occurrence of this section within the form set (without considering what forms contain the other copies of this section), then get the fifth instance of field data.
FLD	Use the occurrence of this field within the current form as the instance index. For example, if this field is the third occurrence within the same form, then get the third instance of field data.
FLDFSET	Use the occurrence of this field within the entire form set as the instance index. For example, if this is the tenth occurrence of this field within the entire form set (without considering what forms and sections contain the other copies of this field), then get the tenth instance of field data.

A successful return does not indicate whether the field was assigned a value.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	Issue
Offset	0
Length	8
Source name	Issue
Offset	0
Length	8
File	*
Record	*
Required flags	*
Rule	MapFromImportData
Mask	*

*** no entry required for this field in this example**

In this field...	Enter...
Data	Keywords if desired

*** no entry required for this field in this example**

In the DDT file, this information looks like this:

```
;0;0;ISSUE;0;0;ISSUE;0;8;;MapFromImportData;;N;N;N;N;1167;652;1201;
```

The above example would try to map the field by querying the section dictionary first; then the form dictionary, and finally the global dictionary.

Suppose you had this line in the DDT file:

```
;0;0;ISSUE;0;0;ISSUE;0;8;;noopimp;INDEX(IMG);N;N;N;N;1167;652;1201;
```

This example would use the occurrence number of this section on the form to index the global dictionary for this field.

See also

NoOpFunc on page 415

ReplaceNoOpFunc on page 197

ImportExtract on page 111

ImportFile on page 116

Section and Field Rules Reference on page 274

Master

Use this field level rule (level 4) to tell the system the field has been mapped in the MASTER.DDT (data definition table) file. Use this rule when you have variable fields which are used on multiple sections.

Instead of mapping these identical variable fields, like Name and Address, each time they are used, you can map them once in the MASTER.DDT file and then map the individual fields to the Master rule. This tells the system to look in the MASTER.DDT file for the complete mapping information for those variable fields.

Syntax

```
Master( )
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	NAME
Offset	*
Length	*
Source name	REC-NAME
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	Master
Mask	*
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-NAME;;;NAME;;;MASTER;;;;;;15960;4000;17200;
```

In the AFGJOB.JDT file, you must use this rule:

```
;LoadDDTDefs;1;;
```

See also

Setting Up the MASTER.DDT File on page 507

Section and Field Rules Reference on page 274

MessageFromExtr

Use this field level rule (level 4) to retrieve a message from an extract file and place the message into a field on the form. Default formatting information comes from the definition of the field which you set up using Studio or Image Editor. Specific formatting information is embedded within the message using tags. This rule can also contain variable blocks of text.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	The field name
Offset	The offset within a record where the key can be found
Length	The length of the key
Source name	*
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	MessageFromExtr
Mask	The field, defined in the Record Dictionary, contains the grouping code. For more information about the Record Dictionary, see Using the Record Dictionary on page 495.
Data	The record, defined in the Record Dictionary, contains the message key.

* no entry required for this field in this example

In the DDT file, this information could look like this:

```
<Image Field Rules Override>
;0;0;ExplChrg;323;38;
ExplChrg;0;0;;MessageFromExtr;Detail;N;N;N;N;338;18551;16008;
;0;0;Mess;238;93;Mess;0;0;MsgLinePriority;MessageFromExtr;Message;N
;N;N;N;338;25975;16112;
```

For this rule, the main components are the *message*, the *message tags*, the *Record Definition Dictionary*, and the *INI options*. These components are discussed in the following topics.

Creating Messages

The *message* is the text retrieved from the extract file. This text can contain *message tags* which control how the text is formatted. The tags control the justification, spacing, font, variable insertion and other functions. All formatting information is contained in the tags, which serve as a mark-up language. The message itself is straight text. The message is a single line or record with a maximum length of 2000 bytes.

Setting up the field

You attach this rule to a field in the section's DDT file. You can do this by editing the DDT file in a text editor or by using Image Editor. In the section (FAP file), you define the field to which the message is attached as a *multi-line text* field.

If you also select the Can Grow option for the field, the system combines all messages with the same group code in this field, letting the field *grow* to accommodate the entire message. If you do not select the Can Grow option, the system only includes the messages that fit within the defined area for the field. Messages that would not fit are ignored.

Adding messages

The system adds messages in the order they appear in the extract file or database. It tests every message to see if the message fits in the available space. All the messages of a group must fit in the available space or none of the messages appear.

Grouping messages

You can group and format messages as a single message by including a *group code*. You define grouping codes in the Record Dictionary. For more information about the Record Dictionary, see Using the Record Dictionary on page 495. All the messages in a group are treated as a single message. Group codes cannot be blank. All messages comprising a group must be located together in the extract list.

Formatting messages

The formatting information you select for the field serves as the default formatting information for all messages, so make sure you set up the field in the section (FAP file) to use the default font you want for all messages.

Message tags override the default formatting you set up for the field and let you control the appearance of the message text. You can add tags to the text of the message to describe and control the justification, spacing, fonts and other formatting information. The tags affect all text which follows the tag. You can also use message tags to insert variable data fields.

NOTE: Place the variable (VAR) tag within the text where you want the system to insert the variable.

Tags are enclosed within brackets (< >). The text between the brackets describes the formatting action or the reference to the variable name referenced in the Record Dictionary. The tag itself does not appear in the formatted text.

The following table describes the tags you can use:

Tag	Description
<Justify:value>	You can enter *Left*, *Right*, or *Center* to have the system left, right, or center-justify the text. If you omit the value, the system uses the previously defined justification. You can abbreviated tags if the line size is limited. For example, <J:L> or <Justify:L> provides left justification and <J:C> or <Justify:C> provides center justification. Note: If the justification tag is in the message—not the first entry on the line—then you must insert the carriage return tag before the justification tag. For example, this message places the word, *age*, to the right of the second line: `<CR><Justify:Right>Age`
<Font:value>	You can enter any valid font ID (00000-99999) or *Default*. The value is a numeric reference to the font cross reference file (FXR) font ID. If the value is left blank, the system uses the previously defined font ID. For example, <Font:16210> changes the text to the font identified with font ID 16210 in the FXR file while <Font> changes the font back to the default font as font defined in the FAP file. You cannot abbreviate this tag.
<Var:var-name>	*var-name* refers to a variable name defined in a Record Dictionary. The Definition Dictionary describes the variable data. The description defines each record and the fields within each record, such as the record name, offset, length, format, and so on. It does not include formatting information, such as the font ID. You can abbreviate this tag as <***V:var-name***>. For example, <***V:DTAT***> references the variable *DTAT* as defined in the Record Dictionary and <***VAR:DTV1***> references the variable *DTV1* as defined in the Record Dictionary
<Spacing:value>	You can enter *Single*, *Double*, or a numeric value in FAP units (2400 per inch in place of *value*). Single indicates the following text should be single-spaced. Double indicates the following text should be double-spaced. A numeric value tells the system the number of FAP units to use for spacing. If you omit the value, the system returns to the previously defined spacing. You cannot abbreviate this tag. The spacing option you choose applies to the entire message grouping. You cannot change spacing within a grouping (a single message). For example, <Spacing:Double> tells the system to double space the message lines within the message or message group while <Spacing> returns the spacing to the default format. Note: If you change spacing in the text of a message—not the first items in the message—you must insert a carriage return tag before the spacing tag. For example, this changes spacing to double lines: `<CR><Spacing:Double>`

Tag	Description
<Tab>	Use the Tab tag to have the system indent the text from the left margin by a specified number of FAP units. You can abbreviate the tag to <T> if the line size is limited. You can justify the text relative to the tabbed position by specifying Left, Center, or Right. You can abbreviate the tags by using the first character (L, C, or R) if the line size is limited. The default is to left justify the text. You can also use different types of fill (leader) characters if the text does not fill the entire space. You can use these leader characters: no leader (spaces), dashes (---), periods (…), or underlines (___). Except for no leader, you can abbreviate the tags using the first character if the line size is limited. For no leader, you must use the word *nolead* for spaces. Spaces are the default fill characters.
<CR>	This tag tells the system to insert a hard return (carriage return), or forced line break. For example, Residential <CR> rate will look like this: Residential rate You cannot abbreviate this tag.

Here are some examples using the Tab tag…

```
<CR><T:9600,Left,nolead>Tabbing in 4 inches with no leader.
<CR><T:9600,Left,dash>Tabbing in 4 inches with dashes.
<CR><T:9600,Left,period>Tabbing in 4 inches with periods.
<CR><T:9600,Left,underline>Tabbing in 4 inches with underline.

0   1   2   3   4   5   6   7   8
                  Tabbing in 4 inches with no leader.
-----------------Tabbing in 4 inches with dashes.
.................................................Tabbing in 4 inches with periods.
_____________________Tabbing in 4 inches with underline.


<CR><T:9600,C,D>4"dashes & text centered.
<CR><T:9600,R,P>4"periods & text right.

0   1   2   3   4   5   6   7   8
----------------4" dashes & text centered.
.................4" periods & text right.
```

Here is another example. The following tags...

```
<Justifiy:Center><Font:23712>Example<CR><Justify><Font>This is a
sample message.<CR>Name<Justify:Right>Age
```

... produce this message:

Example

This is a sample message.

Name Age

Using the Record Dictionary

The Record Dictionary provides the information for identifying and locating records and fields within records. The Record Dictionary is an ASCII file you can create using any ASCII editor. Used with the Condition table, any variable in the Record Dictionary can be used in a conditional evaluation. For more information about the Condition table, see Using Condition Tables on page 492.

Record Dictionary information is divided into two sections:

- the Record section, which describes the records
- the Variable Definition section which describes the fields contained within the records

For more information about the Record Dictionary, see Using the Record Dictionary on page 495.

Record definition syntax

```
Record Name = Search(Column, Search Mask) {Repeating}
```

Parameter	Description
Record Name	The name the record will use in the future. A record name begins with an alpha character and can have a maximum of 30 characters. You can have only one description for a given record name. Record names are not case sensitive—you cannot define both *BASE* and *Base*.
Search	Keyword
Column	Starting column number to search.
Search Mask	Text to search for in the record columns.
Repeating	(Optional) Keyword used to indicate there may be multiple records of that type in a transaction.

Here are some examples:

```
< Records >
    Message    =  Search(61,01) Repeating
    Account    =  Search(61,02)
```

Variable definition syntax

```
Variable = Record(name) Offset(n) Length(n) Type(x)
Variable = GVM(name) Offset(n) Length(n) Type(x)
Variable = Record(name) Offset(n) Length(n) Type(x) Rule(name)
Format(flags)
Variable = Record(name) Offset(n) Length(n) Type(x) Format(flags)
Precision(n)
```

Parameter	Description
Variable	The name future references to this variable will use. A variable name begins with an alpha character and can have a maximum of thirty (30) characters.
Record(name)	(Optional) Identifies the record in which this variable will be found. This parameter is mutually exclusive when using GVM.
GVM(name)	(Optional) The name of the global variable to use. This parameter is mutually exclusive when using Record. Also keep in mind a rule is not necessary with the global variable.
Offset	The offset into the record where the data is located.
Length	The length of the data.
Type(x)	(Optional) May be either *Char*, *Num*, *Zone*, or *Packed*. *Char* is character data. Can be any string of alphanumeric characters and symbols. *Num* is numeric data. Can have a sign in front and a decimal place. *Zone* is zoned decimal. Looks like a numeric value except the sign is added to the last digit. *Packed* is packed decimal. A binary format used mainly on z/OS systems.
Format(flags)	(Optional) Similar to the flags used with the MoveNum rule except the input flags (such as input length and precision, S, and B) are not needed.
Rule(name)	(Optional) You can include any field rule such as DateFmt or SetAddr2. The Move_It and MoveNum rules are inherent to the Record Dictionary, so you do not need to call them. If you omit a rule, the Move_It rule functionality is the default.
Precision(n)	(Optional) The number of decimal places for a numeric variable.

Here are some examples:

```
< Variables >
MSGTYPE  = Record(Message)Offset(41) Length(1) Type(Char)
MSGGID   = Record(Message)Offset(37) Length(2) Type(Zone)
GRPHID   = Record(Graph)  Offset(31) Length(8) Type(Packed)
PRTCOND1 = Record(Graph1) Offset(31) Length(8) Type(Num) Format(C)
PRTCOND2 = Record(Graph2) Offset(31)Length(8)Type(Num)Precision(5)
Total    = Record(Address)Offset(50)Length(50)Rule(SetAddr2)

* OMR

RCBBATCH = GVM(RCBBatchName)Length(32)Type(Char)
```

INI options

This INI option is required. Place the Record Dictionary file in your DEFLIB directory.

```
< DataDictionary >
    Name  = (file name of the Record Dictionary)
```

Sample Record Dictionary

Here is a sample Record Dictionary definition:

```
*
*  This is the Record Dictionary
*
* These are the Records
*    The only parameter is the record search mask. This can only be
*    used for non-repeating records.
<Records>
Message   = Search(61,01) Repeating
Account   = Search(61,02)
MeterRead = Search(61,03) Repeating
Detail    = Search(61,18) Repeating
*
*
* These are the variable definitions
*    The required fields:
*       Record name defined in the above section
*       Offset into the record where the data begins
*       Length of the data
*    Optional fields:
*       Formatting routine (data as is will be the default)
*       Type or input format (not currently used)
<Variables>
*****  The following are examples. All white space is ignored. ****
*
*     AcctNum      = Record(Header) Offset(4)   Length(15) Type(Char)
*     MessageText  = Record(Message) Offset(53) Length(38) Type(Char)
*     CompanyCode  = Record(Header) Offset(24)  Length(2)  Type(Char)
*     CustomerName = Record(Client) Offset(22)  Length(21) Type(Char)
*     NoticeDate   = Record(Client) Offset(79)  Length(8)  Type(Num)
Rule(Date) Format( )
*    CashReceived = Record(Client) Offset(313) Length (10) Type(Zone)
Rule(MoveNum)   Format(10.2,13.2,$,S-)
*
ACSA  = Record(Account) Offset(487) Length(10) Type(Zone)
Rule(MoveNum) Format(10.2,14.2,S-,C)
ACSC  = Record(Account) Offset(497) Length(10) Type(Zone)
Rule(MoveNum) Format(10.2,14.2,S-,C)
ACBB  = Record(Account) Offset(436) Length(10) Type(Zone)
Rule(MoveNum) Format(10.2,14.2,S-,C)
BMV1  = Record(Message) Offset(159) Length(15) Type(Char)
BMV2  = Record(Message) Offset(174) Length(15) Type(Char)
BMV3  = Record(Message) Offset(189) Length(15) Type(Char)
BMV4  = Record(Message) Offset(204) Length(15) Type(Char)
BMV5  = Record(Message) Offset(219) Length(15) Type(Char)
DTAT  = Record(Detail)  Offset(163) Length(10) Type(Zone)
Rule(MoveNum) Format(10.2,14.2,S-,C)
DTV1  = Record(Detail)  Offset(181) Length(18) Type(Zone)
Rule(MoveNum) Format(18.8,18.2,S-,C)
DTV2  = Record(Detail)  Offset(199) Length(18) Type(Zone)
Rule(MoveNum) Format(18.8,18.2,S-,C)
DTV3  = Record(Detail)  Offset(217) Length(18) Type(Zone)
Rule(MoveNum) Format(18.8,18.2,S-,C)
```

```
DTV4  = Record(Detail)  Offset(235) Length(18) Type(Zone)
Rule(MoveNum) Format(18.8,18.2,S-,C)
DTV5  = Record(Detail)  Offset(253) Length(18) Type(Zone)
Rule(MoveNum) Format(18.8,18.2,S-,C)
DTVM1 = Record(Detail)  Offset(271) Length(15) Type(Char)
DTVM2 = Record(Detail)  Offset(286) Length(15) Type(Char)
DTVM3 = Record(Detail)  Offset(301) Length(15) Type(Char)
*
***The following is the grouping that is defined for messaging*
*
MsgLinePriority = Record(Message) Offset(96) Length(5) Type(Zone)
*
*
================================================================
```

See also Section and Field Rules Reference on page 274

Using Condition Tables on page 492

Using the Record Dictionary on page 495

Mk_Hard

Use this field level rule (level 4) to insert or *hard code* a value into a variable field. For instance, you can use this rule to place an *X* in a check box or to insert the text *Same as above* in a field on a form.

Syntax

```
Mk_Hard( )
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	CHECK BOX
Offset	*
Length	1
Source name	REC-CHECK BOX
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	Mk_Hard
Mask	*
Data	X

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;REC-CHECK BOX;;;CHECK BOX;;1;;Mk_Hard;X;;;;;
```

This example puts an uppercase *X* into the destination buffer for the field named CHECK BOX. This field has a destination length of one character, and begins at offset 1 by default (since none was specified).

NOTE: Source offset and length have no significance to this rule since the source text is coming from the Data field in the mapping.

See also HardExst on page 356

SetCpyTo on page 454

Section and Field Rules Reference on page 274

MNumExt

Use this field level rule (level 4) to perform the function of the MoveNum rule if an external record is found. Enter the external record search criteria after the MoveNum search criteria in the Data field. You can also enter a calculation after the external search criteria. This rule supports overflow.

The format mask must contain the input numeric format, followed by the output numeric format. These formats are in the form of *X.Y*, where *X* is the size of the number, including any commas, currency symbols, and decimal points, and *Y* represents the number of digits after the decimal point, such as:

```
10.2,12.2
```

The first pairing of *X.Y* describes the input. The second pairing describes the output. In this example, 10.2 is the description of the data in the extract record and the output for this would be 12 digits before and two digits after the decimal.

The format mask can contain any of these formats after the output numeric format:

Format	Description
L	Left justify the number
C	Add commas
B	Translate BCD number to decimal
Z	Print number even if it equals zero (0)
$	Add a dollar sign ($)

NOTE: You *cannot* use the dollar sign ($) as the first character in the format mask because this conflicts with the use of this character in the Move_It rule.

The data may contain a calculation to be performed upon the number obtained from the extract record. The calculation must be separated from the search criteria by a space, enclosed in parentheses, and contain spaces to separate each element (including parentheses) in the equation string.

An *X* in the calculation is replaced by the value moved from the extract file. You must place parentheses around each operator and its accompanying operands.

NOTE: This rule does not support an OR condition in the search mask. You can, however, run multiple searches.

Image Editor example 1

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	1PAYAMT
Offset	*
Length	18
Source name	REC-1PAYAMT
Offset	64
Length	6
File	*
Record	*
Required	*
Rule	MNumExt
Mask	6.0, 11. 2, B
Data	17,PMSPAR01 17,PMSP0200,99,0 ((X * 50) + 2.50)

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-1PAYAMT;64;6;1PAYAMT;;18;6.0,11.2,B;mnumext;17,PMSPAR01
17,PMSP0200,99,0 ( ( X * 50 ) + 2.50 );N;N;N;N;
```

The move occurs only if the record defined by the external search criteria is found. If not found, zero is used in place of *X*.

If the external extract record matching the search format 17,PMSP0200 and 99,0 is found, the current record defined by the search format 17,PMSPAR01 is searched to get the numeric data from offset 64 for length 6.

The six-character BCD value located at offset 64 is then multiplied by 50. The system adds 2.50, as specified by the operation ((X * 50) + 2.50), before copying the result to the destination field. The destination field can contain up to 11 characters, including the decimal point and two characters after.

Image Editor example 2

This example shows the use of a user function and overflow symbol. If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	TOTAL PREM
Offset	*
Length	12
Source name	TOTAL PREM
Offset	87
Length	6
File	*
Record	1
Required	*
Rule	MNumExt
Mask	11.0,7.2,$,L
Data	@GETRECSUSED,FORMABC,OVSYM/17,PMSPAR01 17,PMSP0200,99,0 ((X * 50) + 2.5)

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;1;TOTAL PREM;87;6;TOTAL PREM;;12;7.2,11.0,$,L;MNumExt;
@GETRECSUSED,FORMABC,OVSYM/17,PMSPAR01 17,PMSP0200,99,0 ( ( X * 50 )
+ 2.5 );N;N;N;N;
```

See also

MoveNum on page 402

Using Pre-defined Numeric Formats on page 261

Section and Field Rules Reference on page 274

Move_It

Use this field level rule (level 4) to move text from an extract record to the output buffer. This rule supports overflow.

The format mask can contain these options:

Format	Description
B	Used when mapping XML extract data. Here is an example: `;0;0;FIELD;0;1024;FIELD;0;1024;B;move_it;!/ descendant::My_Extract_Data/ FIELD.xml();N;N;N;N;3715;2899;11010;`
C	Centers the data.
D	Converts the data returned by the rule into lowercase.
F	Converts to sentence style where the first character is capitalized and the remaining characters are lowercased.
G	If you include this flag, the system always returns the data at the source length and ignores the destination length. If you omit the G flag, the Move_It rule returns data truncated to either the source or destination length, whichever is shorter. For instance, if you use the Move_It rule with a source length of 20 and a destination length of 19, including the G flag returns 20 characters. Omitting the G flag returns 19 characters. If you include the ChkDestLenExceeded option in the RunMode control group and set the option to Yes, the system reports any occurrence where the destination length is less than the source length.
L	Left justifies the data.
K	Removes leading and tailing spaces.
N	Searches for the next record in the transaction list instead of starting with the first record. For example, assume the current transaction has five records (A, B, C, D, and E) and the last record processed is C. If the next rule is Move_It and it has the *N* flag set, the D record will be searched instead of starting at the top (record A). When you include the Move_It rule and you are using overflow (@UserFuncName options) do not use the *N* flag.
R	Right justifies the data (for non-proportional fonts only).
SR	Same record flag. This flag is similar to the N (next record) flag except it assumes the data is in the record returned by the previous Move_it rule. Note that SR *only* applies to the prior execution of a Move_It rule. You must have at least one Move_It rule *without* the SR flag before you can add a Move_It rule which uses this flag.
T	Include this flag to format the text in title case. This flag tells the system to capitalize the first letter in each word in the string and lowercase the rest of the letters in each word. See the example on page 397 for more information
U	Converts the data returned by the rule into uppercase.

Format	Description
$	A string proceeded by a dollar sign ($) is used as a sprintf format for the output data.
8	Indicates the extract data for this field is stored in UTF-8 format. UTF-8 (Unicode Transformation Format, 8-bit encoding form) is a format for writing Unicode data in text files. See Using Unicode Support for more information.
Blank	Default, trims trailing spaces.

NOTE: Before version 10.0, this rule did not permit multiple flags. Beginning with version 10.0, flags are executed in sequence, thus the particular order may cause a difference in the formatted string output. When you use multiple format mask flags, use a comma as a separator.

Do not use *C*, *R*, or *L* with *K*. The system intentionally skips *K* after *C*, *R*, or *L* is mapped. If *K* occurs before *C*, *R*, or *L*, it will not affect *C*, *R*, or *L* mapping.

For example, if a *K* flag occurs first, the system clips the heading and tailing spaces and then formats the string. If a *K* flag occurs second in the format string, the system formats the string and then clips the heading and trailing spaces.

You can do some interesting things by handling the flags in sequence, you can clip (K) the input data, format ($) the string, right justify (R) the result, and format ($) it again.

On the other hand, you can do some things that don't make sense, like center justify (C) the data and then clip (K) the result. This sequence negates the center justification. The same applies to right (R) and left justification (L) if you put a *K* in the format afterwards.

Furthermore, what order would you expect *R*, *C*, and *L* applied especially when mixed with the format ($)? Just like the clip flag (K), if you right justify (R) first and then format ($), you will likely get different results than if you format ($) first and then right justify (R).

NOTE: If you apply this rule to a multi-line variable field, make sure destination length is greater than one (1). Otherwise, no data will be mapped. This happens because when you create a multi-line variable field, its length is zero (0) and its destination length in the DDT file is also set to zero. While some rules, such as the Mk_Hard rule, map data even if the destination length is zero, the Move_It rule will not.

Handling currency symbols

Let's assume you have variable fields that represent amounts. The extract data is pre-formatted as character text (left justified), which represents the correct currency format. Unfortunately, the extract data does not include the currency symbol. You have to add the currency symbol.

Depending on the nationality, the currency symbol can appear at the beginning of the amount, like the dollar sign, or it can appear at the end of the amount, such as *FF* for French Francs.

Beginning with version 10.0, you can use a format of *K,$%sFF* to add the currency symbol.

The *K* would come first to indicate the space before and after the input data should be removed, then *FF* would be appended.

So *K,$%sFF* is not the same as *$%sFF,K*. The former clips the input data *before* the format is applied. The latter clips the data *after* the format has been applied.

For currency symbols that appear at the beginning of the data, such as British pound sterling, you could use the Move_It rule with a format mask of *$£%s*. This works because trailing spaces are trimmed. For currency symbols at the end of the data, this will not work.

User functions

The Move_It rule supports the use of @UserFuncName functions. User functions let you move data from the source record to the output buffer based on the outcome of a user-defined function. User functions and their parameters are specified in the data field before the search criteria.

Image Editor example

In this example, the user function name is *GetRecsUsed* and has two parameters. *FORM NAME* and *VAR;* / denote the end of the parameter list and the start of the search criteria.

```
;Move_It;@GETRECSUSED,FORM NAME,VAR/17,00,15,B;;;;
```

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	POLNUMBER
Offset	1**
Length	10
Source name	REC-POLNUMBER
Offset	50
Length	10
File	*
Record	* (generally not required unless you are also using overflow)
Required	*
Rule	Move_It
Mask	*
Data	17,PMSP0200

* no entry required for this field in this example

**if you set the offset field to zero (0), the system automatically defaults to 1.

In the DDT file, this information looks like this:

```
;;;REC-POLNUMBER;50;10;POLNUMBER;1;10;;Move_It;17,PMSP0200;;;;;
```

This rule gets the first occurrence of a record matching the search criteria of PMSP0200 at offset 17. From the extract record, ten characters from offset 50 are moved to the output buffer (which also happens to be 10 characters in length).

This example shows the use of a user function and overflow symbol:

In this field...	Enter...
Destination name	POLNUMBER
Offset	1
Length	10
Source name	REC-POLNUMBER
Offset	50
Length	10
File	*
Record	1 (required with overflow)
Required	*
Rule	Move_It
Mask	*
Data	@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200

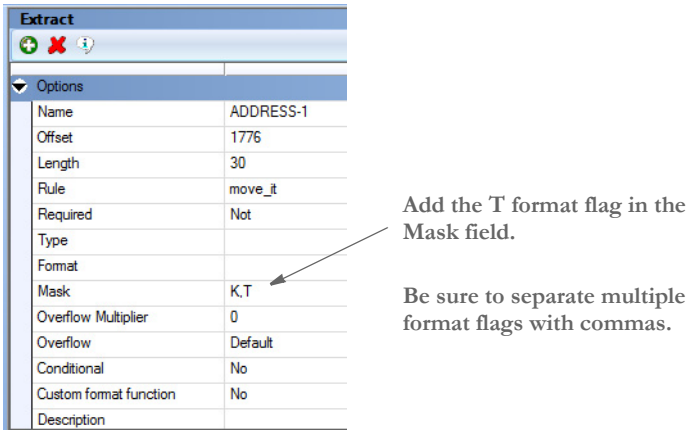
* no entry required for this field in this example

In the DDT file, this information looks like this:

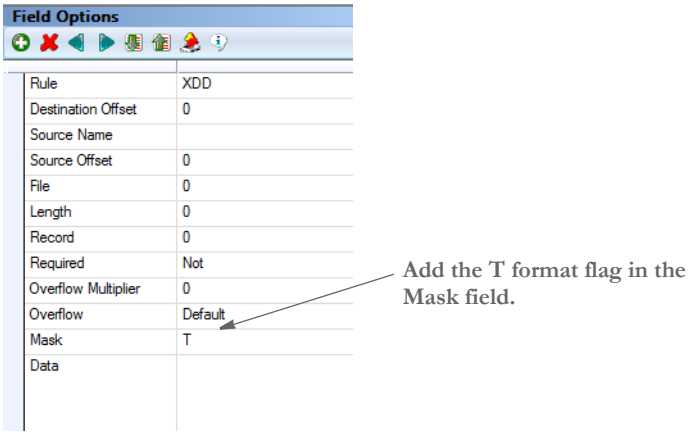
```
;;1;REC-
POLNUMBER;50;10;POLNUMBER;1;10;;Move_It;@GETRECSUSED,FORMABC,OVSYM/
17,PMSP0200;;;;;
```

Studio example

Here is an example of how you can specify the T (title case) format flag in an extract dictionary definition:



You can also specify this format flag in the field's rule mapping in the Section manager.



NOTE: The Section manager example shows the T format flag used with the XDD rule. The presumption is that when the dictionary element is found, the resulting rule will be the Move_It rule. Adding the T format flag here overrides any mask defined in the XDD definition.

Here are some examples of what happens when you include the T format flag:

This text	Is changed to
11 paces ferry road	11 Paces Ferry Road
SAM DOE	Sam Doe

This text	Is changed to
Marquis de Lafayette	Marquis De Lafayette
George O'Brien	George O'brien
This is the Title Case Option	This Is The Title Case Option

Keep in mind:

- In some cases, the use of this formatting flag can result in unwanted changes.
- The T format flag will work on Unicode text that has upper and lower equivalents. If the text characters are for a language that does not have such distinctions, like certain Asian character sets, then those Unicode characters will not be modified.
- You can enter several format flags in the Mask field of the Move_It rule. If you include conflicting format flags, the last one determines the results. For instance, if you specify both this flag and the Uppercase format flag (T,U) in that order, the result is upper cased, because the U is the last flag specified.

See also

MoveExt on page 399

Extracting data on page 409

Search Criteria on page 270

Section and Field Rules Reference on page 274

MoveExt

Use this field level rule (level 4) to move text from a specific external extract record to the output buffer *if* a specific external record is found. This rule calls the Move_It rule if the specified extract record is found. This rule supports overflow.

A string in the format mask that is preceded by a dollar sign ($) is used as a sprintf format for output data. You cannot use this feature with a format mask ID.

User functions let data move from the source record to the output buffer based on the outcome of a user-defined function. User functions and their parameters are specified in the data field *before* the search criteria, and follow the syntax described below.

In this example the user function name is GetRecsUsed. It has two parameters FORM NAME, and VAR. The slash (/) denotes the end of the parameter list and the start of the search criteria.

```
;MoveExt;@GETRECSUSED,FORM NAME,VAR/17,00,15,B;;;;;
```

Image Editor examples

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	POLNUMBER
Offset	1
Length	10
Source name	REC-POLNUMBER
Offset	50
Length	10
File	*
Record	*
Required	*
Rule	MoveExt
Mask	*
Data	17,PMSP0200 17,IPMSR01,99,T1

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;REC-POLNUMBER;50;10;POLNUMBER;1;10;;MoveExt;17,PMSP0200
17,IPMSR01,99,T1;;;;;
```

The move only occurs if the record defined by the external search criteria is found. If the extract record matching the search format 17,IPMSR01,99,T1 is found, the record defined by the search format 17,PMSP0200 is searched to get the text from offset 50 for the length of 10 characters.

This example shows the use of a user function and overflow symbol:

In this field...	Enter...
Destination name	POLNUMBER
Offset	1
Length	10
Source name	REC-POLNUMBER
Offset	50
Length	10
File	*
Record	1
Required	*
Rule	MoveExt
Mask	*
Data	@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200 17,IPMSR01,99,T1

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;1;REC-
POLNUMBER;50;10;POLNUMBER;1;10;;MoveExt;@GETRECSUSED,FORMABC,OVSYM/
17, PMSP0200 17,IPMSR01,99,T1;;;;;
```

See also Move_It on page 393

Search Criteria on page 270

Section and Field Rules Reference on page 274

MoveMeToPage

Use this section level rule (level 3) to move the entire page the section resides on to a designated page.

Syntax

```
;MoveMeToPage;;PageNumber;
```

Parameter	Description
PageNumber	Page number to which the sections will be moved. To move sections to the last page, enter zero (0).

If you use the rule, you must have the following set in the AFGJOB.JDT:

```
;ProcessQueue;;PaginationQueue;
```

This rule only works with forms that have multiple pages.

Image Editor example

```
;MoveMeToPage;;3;
```

This example moves the section to the third page of the form set.

```
;MoveMeToPage;;0;
```

This example moves the section to the last page of the form set.

See also Section and Field Rules Reference on page 274

MoveNum

Use this field level rule (level 4) to move numeric data from an extract record to the output field and, if necessary, reformat the data. This rule supports overflow.

NOTE: The numeric data this rule handles is limited to 15 significant digits. This is a total of all the digits, both to the right and left of the decimal. Here are some examples:

```
999,999,999,999,999.
.999,999,999,999,999
999,999,999.999,999
```

The system tries to honor almost any format you supply, but when a conversion has to occur, it can only guarantee 15 significant digits in the result.

The first part of the format mask must contain the input numeric format (*X.Y*) followed by the output numeric format (*X.Y*), where *X* is the size of the number, including any commas, currency symbols, and decimal places, and *Y* is the number of digits after the decimal. For example, a simple format mask can look like this:

```
10.2,15.2
```

This tells the system the input string consists of ten characters and the last two characters are decimals, such as *1234567890*. The output string should consist of 15 characters, including two decimals.

To format the output, you can also include any of the following format options after the output numeric format (separate each option with a comma).

Format mask

Mask	Description
-	(*one dash*) If the number is negative, this option places a minus sign (-) in the *left most* position. For example, if the format mask is (9.2,12.2,C,$,-), the result is: "-$2,100.00".
--	(*two dashes*) If the number is negative, this option places a minus sign (-) *immediately* before the amount. For example, if the format is (9.2,12.2,C,$,--), the result is " -$2,100.00", with a full length of 12.
+	Tells the system to always include a sign with all numbers.
%	Appends a percent sign (%) at the end of the number.
$	Adds a dollar sign. The dollar sign cannot be the first character in the format mask. This limitation arises from the Move_It format option, where a dollar sign ($) in the first character of the mask means to perform a sprintf.
A	Removes the trailing spaces after an extract value if the input data type is neither BCD nor Packed Decimal. For example, assume the data value is "100000 " (a one followed by five zeros and two spaces). If you omit this flag and select a 12.2 output format with commas, the value generated will be " 100,000.00". If you include this flag, the result will be " 1,000.00".

Mask	Description
B	Translates a BCD number into a decimal. If the data *is* in EBCDIC format, use this flag instead of the BA flag.
BA	Translates a BCD number into a decimal. Use this flag for ASCII signed numbers.
C	Adds commas to the output.
C**	Adds commas if the data is in US English format or spaces if the date is in Canadian French format.
CR	Appends *CR* (credit) to the end of the number.
CS1 CS2 CS731	Enter one of these options to indicate the checksum method. The system appends a check digit (mod 10) of 0 through 9 to the end of the number. This is typically used in accounting to make sure a number, such as an account number is correct by performing a formula on each digit. For details, see the discussion on page 406.
D	Dollars (a combination of B, C, and $). You must modify GEN_FMT_FmtMaskSaysBinary to recognize this format.
E	Stops a calculation if the search condition is false. The Move_It rule may return a null output buffer if: no record was found; a record was found, but the search mask contained a pairing (offset,data) which extended past the end of the record; or a record was found, but the mapped data was blank.
F	Add a dollar sign ($) and place it in the first position. If the value is negative, move the minus sign (-) to the last position.
G	Tells the MoveNum rule not to use the Move_It rule to get the data from the extract file. See Extracting data on page 409 for more information.
L	Left justifies the number in the variable field.
-L (or --)	Tells the system to use a floating negative sign on negative values.
+L (or ++)	Tells the system to use a floating sign and to always show that sign.
Lang	Selects a language for spelling out the number. This flag is used with the V flag and mask parameters. Here is an example: US, CFR.
M	Money (This format is a combination of formats C and $.)
N	Leave the output buffer blank if the number is zero or negative.
NM	Adds a minus sign (-) to the number.

Mask	Description
-O	Places a negative sign outside the right side of the field definition. This allows positive and negative numbers to right align on the page if you use a fixed font. Here is an example using this input format: 10.2,10.2,-O: `input data: 0000009.99` `-000012.25` `output: 12345678901234567890` `**********` `9.99` `12.25-`
On	Sets the output field size to n and overrides the output size of the field. Here is an example using this input format: 10.2,10.2,O8: `input data: 0000009.99` `-000012.25` `output: 12345678901234567890` `**********` `9.99` `-12.25` `input format: 10.2,10.2,O12` `input data: 0000009.99` `-000012.25` `output: 12345678901234567890` `**********` `9.99` `-12.25`
P	Print leading zeros. You cannot use this format with $, -, C, and F.
Pn	Pads the output zeroes to n total width. This parameter only works with whole numbers, not decimals. Here is an example using this input format: 10.0,10.0,P4: `input data: 0000000001` `0000000025` `0000012345` `output: 12345678901234567890` `**********` `0001` `0025` `12345`
P**	Prints leading zeros if used without character or symbol enclosed with single quote.
R	Tells the system to retain the minus sign (-) if the result is less than zero. Use with signed numbers.

Mask	Description
-R	Places a negative sign on the right side of the field (within the field). Here is an example using this input format: 10.2,10.2,-R: `input data: 0000009.99` `-000012.25` `output: 12345678901234567890` `**********` `9.99` `12.25-`
R**	Retains the sign when translating a signed data value into decimal. Use with the S flag.
S	Translates signed data to a decimal.
SLZ	Suppress leading zeros. For example, **00.25** becomes **.25**.
T	Adds text before or after a number. Use the less than (<) symbol for inserting before, the greater than (>) symbol for inserting after. Use the comma as a separator. Use with the NegText, Text, and ZeroText data options. You can also use this option to place currency symbols before or after amounts. For instance, **T>£** places the British pound sterling symbol (ALT+0163) before an amount.
TA	Same as T>
TB	Same as T<
SP	Same as E
V	Spells out the numeric value in US English.
X	Adds *X* to the front of the number.
Z	Print a number even if it is zero.
Z2	Prints two zeros.

For example…

Input string	Format mask	Output string
1234567890	10.2,15.2,C,$,L	$12,345,678.90

…tells the system to take a ten-character input string with two decimals (10.2) and output it as a 15-character string with two decimals (15.2), commas (C), a dollar sign ($), and left-justified (L).

This rule respects the number of decimals in the source. For instance, if you have the number “ 1.2" defined as using a mask of 6.2, the system outputs 1.20 instead of 0.12.

NOTE: :The MoveNum and AccumulateVariableTotal rules support three checksum methods. These methods only work on the integer portion of a number. The system ignores the decimal portion of the number.

CS1 works from right to left. CS2 works from left to right. These two algorithms are exactly the same except for the direction in which they work. The calculation works like this:

The odd number digits are multiplied by 2. If that result is greater than 9, then 9 is subtracted from the value. The result is added to the sum. The even number digits are simply added to the sum.

Once all the digits values have been summed, the total is divided by 10. The remainder of this division is subtracted from 10 and that becomes the *check-digit*. If the resulting value is 10, then zero (0) will be the check-digit.

Here are some examples. In all cases, assume the value is 346,100.99. The CS1 calculation works like this: (notice the digits are addressed backwards)

```
(0 x 2) + 0 + (1 x 2) + 6 + (4 x 2) + 3
0 + 0 + 2 + 6 + 8 + 3 = 19
( 19 mod 10) = 9
10 - 9 = 1
```

The resulting number will be 346,100.991.

The CS2 calculation works like this:

```
(3 x 2) + 4 + ((6 x 2)-9) + 1 + (0 x 2) + 0
6 + 4 + 3 + 1 + 0 = 14
(14 mod 10) = 4
10 - 4 = 6
```

The resulting number will be 346,100.996

Note that in the CS1 example, the third digit—an odd number digit—is multiplied by 2 and exceeds 9. Therefore 9 is subtracted from that result before proceeding to the next number).

CS731 is the other checksum method. This method works from left to right. Unlike the other two methods which use an even/odd multiplier, this method has three multipliers. The first digit is multiplied by 7, the next by 3, and the next by 1. This process is repeated until all digits have been multiplied. Unlike the other methods, it does not matter if a the result of a digit multiplication exceeds 9.

CS731 calculation works like this:

```
(3 x 7) + (4 x 3) + (6 x 1) + (1 x 7) + (0 x 3) + (0 x 1)
21 + 12 + 6 + 7 + 0 + 0 = 46
(46 mod 10) = 6
10 - 6 = 4
```

The resulting number will be 346,100.994

Data

The data can contain a calculation to be performed on the number in the extract record. Separate the calculation from the search criteria with a space, enclosed in parentheses. Use spaces to separate each element (including parentheses) in the equation string.

An *X* in the calculation is replaced by the value moved from the extract record. You must place parentheses around each operator and its accompanying operands.

NOTE: If you have zeros in your extract file, the MoveNum rule converts these zeros into blanks unless you include the *Z* option.

Option	Description
NegText	If the value is negative, this option lets you insert user-defined text before and/or after the negative value.
RPN (x)	Allows a calculation to be performed using reverse Polish notation.
Text	Lets you print text before and after the number.
X	Adds *X* to the front of the number.
ZeroText	If the result is zero, this option lets you insert user-defined text instead of the zero value.

Image Editor example

This example...

```
;0;0;AMOUNT;75;10;AMOUNT;0;10;10.2,10.2,T;MoveNum;11 HEADERREC
Text("French $" "FF");N;N;N;N;
```

...produces: French $ 123FF.

This example...

```
;0;0;AMOUNT;75;10;AMOUNT;0;10;10.2,10.2,T;MoveNum;11 HEADERREC
NegText("" "CR");N;N;N;N;
```

...produces: 123 CR.

This example...

```
;0;0;AMOUNT;75;10;AMOUNT;0;10;10.2,10.2;MoveNum;11 HEADERREC
ZeroText("ZERO");N;N;N;N;
```

...produces *ZERO* if the number is 0 (zero).

This example...

```
;0;0;AMOUNT;75;10;AMOUNT;0;10;10.2,10.2;MoveNum;11 HEADERREC RPN(X X
+ 2 /) "FF");N;N;N;N;
```

...produces the number if it is a positive number.

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	TOTALPREM

In this field...	Enter...
Offset	*
Length	12
Source name	REC-TOTALPREM
Offset	87
Length	6
File	*
Record	*
Required	*
Rule	MoveNum
Mask	11.0,7.2,$,L
Data	17,PMSP0200 ((X * .50) + 2.50)

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-
TOTALPREM;87;6;TOTALPREM;;12;7.2,11.0,$,L;MoveNum;17,PMSP0200 ( ( X
* .50 ) + 2.50 );N;N;N;N;
```

This example takes the numeric value represented by the six characters at offset 87 in the record found using the search criteria of 17,PMSP0200. That value is then multiplied by .50.

Then the system adds 2.50, left justifies the output, and adds a leading dollar sign ($). The numeric output can contain up to 11 characters before the decimal point, and zero (0) after.

This example shows the use of a user function and overflow symbol:

In this field...	Enter...
Destination name	TOTALPREM
Offset	*
Length	12
Source name	REC-TOTALPREM
Offset	87

Length	6
File	*
Record	1
Required	*
Rule	MoveNum
Mask	11.0,7.2,$,L
Data	@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200 ((X * .50) + 2.50)

* no entry required for this field in this example

Extracting data

Typically, the MoveNum rule uses the Move_It rule to get numeric data from the extract record before formatting the numeric data. The Move_It rule only copies the least number of characters possible. If the destination length is shorter than the source length, this means that the destination length is used instead of the source.

Normally, this is fine with numeric processing because extract files typically contain unformatted data. For example, you might pick up 123456 and turn it into $$$$$1,234.56 or some other valid format. In these cases, the destination is almost always longer than the source length.

There are cases, however, where the extract data is already formatted in some fashion that's longer than the expected destination. For example 00000001234 might appear in the extract and yet the desired format expected for output is known to never exceed 6.2, such as 9999.99.

In this case, if you use the Move_It rule to get the data the result would be 0000000 because the destination length is only seven characters—shorter than the 11 character source length.

The G flag tells the MoveNum rule not to call the Move_It rule and instead use an alternate function that retrieves the entire source length before it formats the data for the destination length.

In the DDT file, this information looks like this:

```
;0;1;REC-TOTALPREM;87;6;TOTALPREM;;12;7.2,11.0,$,L;MoveNum;
@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200 ( ( X * .50 ) + 2.50 )
;N;N;N;N;
```

The system substitutes zero (0) for *X* and calculates the format when the search condition is not met.

NOTE: You can use format *E* to stop the calculation when the search condition is false.

See also MNumExt on page 390

Search Criteria on page 270

Section and Field Rules Reference on page 274

MoveSum

Use this field level rule (level 4) to add two fields and insert the result into a new field.

This rule uses the Record Dictionary table to get the search criteria, offset, length and type for the variables specified in the Data field. It then performs an addition on the information retrieved from extract file. The output sum is formatted according to the format specified in the format mask in the DDT file.

To apply this rule, you must define the record, offset, length, and type for the variables in the Record Dictionary table. For more information about the Record Dictionary, see Using the Record Dictionary on page 495. An example of the Record Dictionary table is as follows:

```
* This is the Record Dictionary table
<Records>
Account= Search(PMSP0200,17)
<Variables>
TBAL  = Record(Account) Offset(18) Length(12) Type(Zone) Precision(2)
TBAL2 = Record(Account) Offset(38) Length(8) Type(Zone) Precision(2)
```

The name of the data dictionary file must be defined in the DataDictionary control group as shown here:

```
< DataDictionary >
    Name = DataDict.Tbl
```

The path for the table files must also be defined in the MasterResource control group as shown here:

```
< MasterResource >
    TablePath = .\MstrRes\TblLib\
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	NEW_BAL
Offset	0
Length	15
Source name	NEW_BAL
Offset	0
Length	0
File	0
Record	0
Required	*
Rule	MoveSum

In this field...	Enter...
Mask	15.2
Data	TBAL,TBAL2

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0; NEW_BAL;0;0;NEW_BAL;0;15;15.2;MoveSum;TBAL,TBAL2;N;N;N;N;
```

This rule tells the system to look up the variable *TBAL* in the Record Dictionary table named DATADICT.TBL located in the TblLib directory. Then the system gets from the EXTRFILE.DAT file the first occurrence of a record matching the search criteria of PMSP0200 at offset 17 as defined in Account under < Records >.

The output consists of 12 characters from offset 18. It is a zoned decimal with two position precision. The same procedure is applied to the second variable *TBAL2*. The system gets from the EXTRFILE.DAT the first occurrence of a record matching the search criteria of PMSP0200 at offset 17 as defined in Account under < Records >.

The output consists of eight characters from offset 38. It is a zoned decimal with two position precision. Finally, these two numbers are added and the result is stored in the variable NEW_BAL. The length of the sum number is 15 and the precision is two decimals.

See also Section and Field Rules Reference on page 274

MovTbl

This field level rule (level 4) works similarly to the Move_It rule, except records are taken from the table list of records stored in memory instead of the extract records list from which many of the other rules get data.

One or more files may be used to keep tables used by this rule. Each table (file) must be listed in the table list file specified in the Data control group with the name TBLFILE. This table file list file must be in the following format:

```
TABLEFILENAME1.EXT <crlf>
TABLEFILENAME2.EXT <crlf>
```

Each table file name is listed on a single line followed by a carriage return/line feed. The format of the table itself is a key name followed by key data. The key need not be a specific length nor the data, which allows for a single table or group of table files to contain table entries of varying lengths.

For example, suppose a form contains the names and numbers of agents for calling purposes, but these names and numbers change on a regular basis, this situation lends itself to the use of text tables. A table might be created called AGENTS.TBL that contains table entries such as the following:

```
AGENT001 JOE MILLER <crlf>
```

and another table called agentpho.tbl with phone number entries such as:

```
AGENT001PHONE404 111-2222 <crlf>
```

You could then make these tables available to Documaker Server by including them in the file specified by the TblFile setting in the FSISYS.INI file.

You specify the table file name in the Data control group of the FSISYS.INI file as follows:

```
< Data >
    TblFile = .\deflib\TblFile.Dat
```

These tables must first be loaded into memory before the system can use them. To do so, the following rules must be in the AFGJOB.JDT file:

```
;CreateGlbVar;1;TblLstH,PVOID;
;LoadTblFiles;1;;
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	AGENTNAME1
Offset	1
Length	30
Source name	REC-AGENTNAME1
Offset	10
Length	20
File	*
Record	*
Required	*
Rule	MovTbl
Mask	*
Data	1,AGENT001

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-AGENTNAME1;10;20;AGENTNAME1;1;30;;MovTbl;1,AGENT001;;;;;
```

Here, the system will find a maximum of 20 characters at offset 10 in the first record in the table with a key value of AGENT001 at offset 1. It then moves the information to the destination field.

See also

Move_It on page 393

LookUp on page 374

TblLkUp on page 476

Section and Field Rules Reference on page 274

NoOpFunc

This field level rule (level 4) is useful when you are developing new forms because it lets you map all fields and systematically test each field by replacing the NoOpFunc rule with the actual rule you want to use.

If a particular DDT rule keeps failing, you can use NoOpFunc to temporarily replace the original rule and process the form without error until you can evaluate and solve the problem.

NOTE: The field on the form will be blank after processing with this rule.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	DESTNAME
Offset	*
Length	*
Source name	SRCNAME
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	NoOpFunc
Mask	*
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;SRCNAME;;;DESTNAME;;;;NoOpFunc;;;;;;
```

NOTE: When you use the NoOpFunc rule, many of the fields which would otherwise be required for processing are not needed. If, however, any of the other fields contain data, this will not affect the operation of the NoOpFunc rule. The system lets the NoOpFunc rule replace any rule on an existing line in a DDT file.

See also Section and Field Rules Reference on page 274

OvActPrint

Use this field level (level 4) rule to report the *actual* number of overflow records that could be processed per transaction for the overflow section.

Syntax

```
OvActPrint (Section, OvSymbol)
```

Parameter	Description
Section	Name of the overflow section
OvSymbol	Name of the overflow symbol defined by the SetOvFlwSym rule

For instance, assume an overflow section can handle five overflow records before being forced to another page and a transaction contains seven overflow records. This rule would state the output as 7—five for the first page, plus two for the second page.

This rule supports only automatic overflow.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	INC
Offset	*
Length	15
Source name	REC-INC
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	OvActPrint
Mask	*
Data	FORMABC,ITEMSYM **

* no entry required for this field in this example

** The data field contains a form name and overflow symbol, separated by a comma.

The information in the DDT file will look like this:

```
;0;0;REC-INC;0;0;INC;0;15;;OvActPrint;FORMABC,ITEMSYM;;;;;
```

This example outputs to a destination field the actual number of overflow records processed for the form/overflow symbol combination of FORMABC, ITEMSYM.

See also

Overflow and User Functions on page 271

PurgeChartSeries on page 433

SetImageDimensions on page 457

Section and Field Rules Reference on page 274

OvPrint

Use this field level (level 4) rule to report the *maximum* number of overflow records that could be processed per transaction for the overflow section.

Syntax

```
OvPrint (Section, OvSymbol)
```

Parameter	Description
Section	Name of the overflow section
OvSymbol	Name of the overflow symbol defined by the SetOvFlwSym rule

For instance, assume an overflow section can handle five overflow records before being forced to another page and a transaction contains seven overflow records. This rule would state the output as 10—five for the first page, plus five for the second page.

This rule works with the IncOvSym rule.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	INC
Offset	*
Length	15
Source name	REC-INC
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	OvPrint
Mask	*
Data	FORMABC,ITEMSYM**

* no entry required for this field in this example

**The data field contains a form name and overflow symbol, separated by a comma.

In the DDT file, this information looks like this:

```
;0;0;REC-INC;0;0;INC;0;15;;ovprint;FORMABC,ITEMSYM;;;;;
```

This example counts the maximum number for the overflow records used for the form, overflow symbol combination of FORMABC, ITEMSYM.

See also IncOvSym on page 366

SetImageDimensions on page 457

Section and Field Rules Reference on page 274

PaginateBeforeThisImage

Use this section level rule (level 3) to force the system to perform a pagination before it processes this section. Normally, pagination does not occur until the system has finished processing the entire form set - meaning that all data is complete.

If pagination occurs because an earlier section exceeded a page boundary, the internal references for page coordinates are reset for the page that now contains this section.

This rule makes it possible for a section or field rule that occurs later to *know* what page the section (or field) occupies. You can also use this rule if you want to know how much of the page is occupied so you can conditionally include or exclude data. If you waited until normal pagination would occur, it would be after all normal section and field level rules had been executed and it would be too late.

Syntax

```
PaginateBeforeThisImage ( )
```

There are no parameters for this rule. This rule is sometimes used with the ResetImageDimensions and DontPrintAlone rules.

Image Editor example

```
;PaginateBeforeThisImage;;;
```

See also

ResetImageDimensions on page 436

DontPrintAlone on page 329

Section and Field Rules Reference on page 274

PostImageDAL

Use this section level rule (level 3) in the DDT file to execute a DAL script on the POST_PROC_B message. The PostImageDAL rule executes after all field level rules are run.

You can use this rule to handle follow-up tasks after the section and field level rules are executed.

Syntax

```
; PostImageDAL;string;
```

Parameter	Description
String	A character string that contains a DAL function or DAL script.

Although you can use DAL to access almost any form set or section field, keep in mind those fields may not exist, depending on where you place this rule in the transaction job rule list. And, unlike the DAL or IF rules, there is no return value from the execution of a section level DAL script.

Use this form to get extract data if the script is contained in the rule data. You cannot use this form in external script files.

```
A = {1,MIS257 138,1}
```

Where *A* is a DAL variable you wish to assign. The bracketed {} item can be almost any standard search mask supported by the Get Record infrastructure. In this case, *1,MIS257* is the search criteria. If the record is found, the system takes the data from position 138, length 1 as indicated by *138,1*.

This method also lets you specify an occurrence of the record by including a hyphen with a numeric value, such as *-n*, after the data length. Here is an example:

```
A = {1,MIS257 138,1-5}
```

Here the function searches for the 5th occurrence of the 1,MIS257 record. If you omit the occurrence, the system returns the first one found. If it cannot find the requested record, the system assigns the variable an empty “” value.

NOTE: To specify multiple DAL statements in the rule data area, separate the DAL statements using two colons (::). Normally, semicolons separate DAL statements, but this character is illegal in the rule data area.

Image Editor example

```
; PostImageDAL;;Chain("posttran.dal");
```

This example executes the Chain DAL function which then executes the DAL script contained in the POSTTRAN.DAL file in the DefLib directory specified in your MRL.

```
; PostImageDAL;;If  HaveGVM("main_address") Then
SetGVM("main_address", "25 Brown St.", , “C”, 20)::End;
```

In this example, the system checks to see if the GVM variable (*main_address*) exists. If not, it creates a character array GVM variable (*main_address*) 20 characters in length and stores the character string (25 Brown Street) in the array.

See also

PreImageDAL on page 426

PostTransDAL on page 177

PreTransDAL on page 179

Section and Field Rules Reference on page 274

PowType

Use this field level rule (level 4) to force a transaction to manual batch (WIP). The PowType rule sets the manual batch flag to true. To edit the field associated with the PowType rule in the Entry system, you must set the required flag to *operator* for the field.

Syntax

```
PowType ( )
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	ADDR1
Offset	*
Length	40
Source name	INSNAMEREC-INSNAME1
Offset	*
Length	*
File	*
Record	*
Required	Operator
Rule	PowType
Mask	*
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;INSNAMEREC-
INSNAME1;0;0;ADDR1;0;40;;PowType;;N;N;Y;N;87;1406;12010
```

In this example, the operator required flag for the INSNAMEREC-INSNAME1 field must be set if you want this field to be editable in the entry system when it is retrieved from manual batch (WIP).

Suppressing Warning Messages

Use the ShowWIPWarning option to suppress the Sent to Manual Batch warning messages:

```
< RunMode >
    ShowWIPWarning = No
```

Option	Description
ShowWIPWarning	Enter No to suppress warning messages included the error logs when using the KickToWIP or POWType rules, or the KickToWIP DAL function. The default is Yes, which tells the system to include the messages in the error logs.

See also Section and Field Rules Reference on page 274

PreImageDAL

Use this section level rule (level 3) in the DDT file to execute a DAL script on the PRE_PROC_B message. The PreImageDAL rule executes before section or field level rules.

You can use this rule to handle setup tasks which should occur before image and field level rules are executed.

Syntax

```
; PreImageDAL;string;
```

Parameter	Description
String	A character string that contains a DAL function or DAL script.

Although you can use DAL to access almost any form set or section field, keep in mind those fields may not exist, depending on where you place this rule in the transaction job rule list. And, unlike the DAL or IF rules, there is no return value from the execution of a section level DAL script.

Use this form to get extract data if the script is contained in the rule data. You cannot use this form in external script files.

```
A = {1,MIS257 138,1}
```

Where *A* is a DAL variable you wish to assign. The bracketed {} item can be almost any standard search mask supported by the Get Record infrastructure. In this case, *1,MIS257* is the search criteria. If the record is found, the system takes the data from position 138, length 1 as indicated by *138,1*.

This method also lets you specify an occurrence of the record by including a hyphen with a numeric value, such as *-n*, after the data length. Here is an example:

```
A = {1,MIS257 138,1-5}
```

Here the function searches for the 5th occurrence of the 1,MIS257 record. If you omit the occurrence, the system returns the first one found. If it cannot find the requested record, the system assigns the variable an empty “” value.

NOTE: To specify multiple DAL statements in the rule data area, separate the DAL statements using two colons (::). Normally, semicolons separate DAL statements, but this character is illegal in the rule data area.

Image Editor example

```
; PreImageDAL;; service_id={1,PrePost,22,Elect
1,8}::Call("postimage.dal");
```

This example executes the Call DAL function which executes the DAL script contained in the POSTIMAGE.DAL file in the DefLib directory in your MRL.

This example sets the internal DAL variable, *service _id*, to the first eight-characters in the transaction record that match the search mask:

```
1,PrePost,22,Elect
```

Then the Call DAL function executes the DAL script in the POSTIMAGE.DAL file, which resides in the DefLib sub-directory in your MRL.

```
; PreImageDAL;;If (HaveGVM("main")) Then SetGVM("main_address", "25
Brown St.", , "C", 20)::End;
```

In this example, the system checks to see if the GVM variable (*main_address*) exists. If not, it creates a character array GVM variable (*main_address*) 20 characters in length and stores the character string (25 Brown Street) in the array.

See also

PostImageDAL on page 422

PostTransDAL on page 177

PreTransDAL on page 179

Section and Field Rules Reference on page 274

PrintIf

Use this field level rule (level 4) to determine what text should be placed into the output buffer. The PrintIf rule compares a character string from the extract record to the character string specified in the user-defined condition contained in the data field.

This rule does not support comparison of data strings that contain all numeric characters. This rule does supports overflow.

NOTE: The PrtIfNum rule does support comparison of data strings that contain all numeric characters.

The user-defined condition is comprised of one or more user-defined definitions separated by a colon (:). A user-defined definition is comprised of two parameters separated by an equal sign (=). User-defined definition parameters contains the...

- Character string to be compared against
- Character string to be placed in the output buffer, if the comparison is true

Here are some examples:

```
Inc=Extra premium due to age is included.
Exc=Age premium has been excluded.
Y=Age premium is not applicable.
Inc=Age premium is included.:Exc=Age premium excluded.:Y=N/A
```

You can use these format flags:

Flag	Description
C	Center
R	Right justify

The system justifies the data by adding spaces in front of the text. If you are using a proportional font, do not use these flags to align the data. Use the JustFld rule for that.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	PREMB1
Offset	*
Length	4
Source name	REC-PREMB1
Offset	285

In this field...	Enter...
Length	1
File	*
Record	*
Required	*
Rule	printif
Mask	*
Data	17,ASBLCPL1 Y=INCL:N=EXCL

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-PREMB1;285;1;PREMB1;;4;;PrintIf;17,ASBLCPL1
Y=INCL:N=EXCL;N;N;N;N;
```

This will put *INCL* into the field if record ASBLCPL1 offset 285 length 1 = "Y", or *EXCL* if record ASBLCPL1 offset 285 length 1 = "N".

NOTE: Separate the record search criteria and the user-defined condition criteria using a blank space in the data field of the rule.

See also

JustFld on page 367

PrtIfNum on page 430

Section and Field Rules Reference on page 274

PrtIfNum

This field level rule (level 4) is similar to the PrintIf rule. The difference is the PrtIfNum rule compares the data to a number while PrintIf compares data to a character string.

A MoveNum action is performed on the value from the extract record and the resulting value is compared to the value in the user-defined conditions to determine what text should be placed in the output buffer. This rule supports overflow processing.

A user-defined *condition* is comprised of one or more user-defined definitions separated by a colon (:). A user-defined *definition* is comprised of these two items separated by an equal sign (=):

- This item is comprised of a logical operator and numeric value to be used in the comparison. The logical operators supported are:

Operator	Description
=	equal to
>	greater than
<	less than
<>	not equal to
Blank	(default) No comparison occurs, text is moved to output buffer

- Character string (inside quotation marks) to be placed in the output buffer if the comparison is true.

Here are some examples of user-defined conditions:

```
=40="He is forty years old."
```

The logical operator is *equal to*, the numeric value is *40*, and the character string if the comparison is true is *He is forty years old.*

```
>50="He is greater than 50 years old."
```

The logical operator is *greater than*, the numeric value is *50*, and the character string if the comparison is true is *He is greater than 50 years old.*

```
<30="He is less than 30 years old."
```

The logical operator is *less than*, the numeric value is *30*, and the character string if the comparison is true is *He is less than 30 years old.*

```
<>20="He is not 20 years old."
```

The logical operator is *not equal to,* the numeric value is *20* and the character string if the comparison is true is *He is not 20 years old.*

```
=40.0="Forty years old.":>50="Greater than 50":<30="Less than
30.":<>20.00="Is not 20.":29="29 years old."
```

This user-defined condition is comprised of five user-defined definitions.

- If the value from the extract record is equals to 40.0, the string *Forty years old.* is moved to the output buffer.

- If the value from the extract record is greater than 50, the string *Greater than 50* is moved to the output buffer.
- If the value from the extract record is less than 30.0, the string *Less than 30* is moved to the output buffer.
- If the value from the extract record is not equal to 20.00, the string *Is not 20* is moved to the output buffer.
- In this definition, the logical operator does not exist so no comparison is made. If one of the other four user-defined condition is not true, the string *29 years old.* is moved to the output buffer.

NOTE: You must define the MoveNum parameters (format mask) in the PrtIfNum rule mask field. As a minimum, you must define the MoveNum input numeric format (X.Y) followed by the output numeric format (X.Y).

If data (offset, length) does not exist for the search mask, the value returned to PrtIfNum for the comparison is zero (0). Therefore, you may want to include a zero compare in the user-defined conditions.

For example, suppose you left the No check box blank if the data is three and an *X* if the data is a one, two, or four. These user-defined conditions...

```
=3=" ": <>3="X"
```

would not produce the desired results if the data was missing (blank). These conditions...

```
=0=" ":=3=" ": <>3="X"
```

would insert a blank if the data was missing.

Image Editor example

If you want a six-character packed decimal located at offset 200 in a record identified by an XYZ at location 100, and base it on the numeric value, you would do the following:

- If it equals 40, print *MIDDLE AGE*
- If less than 40, print *YOUNGSTER*
- Otherwise (default) print *SENIOR*

Your entries on the Edit DDT tab on the field's Properties window would look similar to the following:

In this field...	Enter...
Destination name	AGEDESC
Offset	1
Length	18
Source name	REC-AGEDESC

In this field...	Enter...
Offset	200
Length	6
File	*
Record	*
Required	*
Rule	prtifnum
Mask	11.0, 18.0, B
Data	100,XYZ =40=”MIDDLE AGE”:<40=”YOUNGSTER”:”SENIOR”

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-AGEDESC;200;6;AGEDESC;1;18;11.0,18.0,B;PrtIfNum;100,XYZ
=40="MIDDLE AGE":<40="YOUNGSTER":"SENIOR";;;;;
```

NOTE: A space separates the record search mask “100,XYZ” and the following logic.

See also

MoveNum on page 402

PrintIf on page 428

Section and Field Rules Reference on page 274

PurgeChartSeries

Use this section level rule (level 3) to remove a series from a chart which contains no series data. The cleanup performed by this rule affects all charts on the section. If a series is defined for a chart, but there is no data to fill that series, in most cases you would not want to include the empty series in the chart.

A series may exist but contain no series data because you may need to add series data on an *as provided* basis. For instance, the customer extract data may contain a variable number of records containing data that goes into a chart. Each of these records may contain data for a single series.

The FAP file would be designed to accommodate the maximum number of series that could be included. If the extract data does not contain records to build the maximum number of series the chart can accommodate, you may want to exclude those series from the chart. This eliminates white space on the chart.

Syntax

```
;PurgeChartSeries;;;
```

You typically use this rule with the CreateChartSeries or FieldVarsToChartSeries rules. Always place this rule *after* any rule which gathers chart data.

NOTE: If you are using Image Editor, add this rule under the Image Rules section in the DDT file.

See also

CreateChartSeries on page 315

FieldVarsToChartSeries on page 335

Section and Field Rules Reference on page 274

RemoveWhiteSpace

Use this section level rule (level 3) to remove the white space from between fields. This rule works similarly to the SetAddr rules, but is not address specific.

Syntax

```
;RemoveWhiteSpace;FIELD1,FIELD2,FIELD3;.../NoWarning
```

The parameters to this rule will include a list of fields that should exist on the section. Because fields are typically not created when no data maps, you must load the section to make sure the empty fields exist.

Separate the fields in the list with commas. Each subsequent field with data will be mapped into the earliest named prior field that did not contain data.

This rule moves field data from one field to a prior named field to *compress* out the space between the fields. Typically, you would use this rule to compress the vertical space, as in address lines, but the rule does not really care whether the space is vertical or horizontal.

Unlike the SetAddr type rules, this rule does not actually map the original data. You must use field mapping rules, like Move_It, to do that. Also note that only the data moves between the fields. The location of each physical field remains the same.

Also, unlike the SetAddr rules, you do not have to compress the space *up*. If you specify the fields in the reverse vertical order, you can compress the space *down*.

Using the NoWarning parameter

If the system cannot locate the field, you get a warning. If you include the NoWarning parameter, however, you can suppress the warning. Add this parameter after the last field in the list. Use a forward slash (/) to separate it from the previous parameter and end it with a semicolon (;). This parameter is optional. Here is an example:

```
;RemoveWhiteSpace;PAGE1_FIELD1, PAGE1_FIELD2, PAGE1_FIELD3,
PAGE1_FIELD4/NoWarning;
```

The warning message includes addition information to help you resolve the problem:

```
DM10190: Warning in <REMOVEWHITESPACE>: Unable to locate field
<FieldName> in image <ImageName>.
```

This rule does not work with bar code fields or multi-line text fields.

If the system cannot locate the listed field and the section is a multi-page section and you omitted the NoWarning parameter, you get message DM10189. If you included the NoWarning parameter, you get message DM10190. Here are some examples:

```
DM10189:  Warning in <REMOVEWHITESPACE>:  Unable to locate field
<FieldName> on page <PageNo> of a multi-page image <ImageName>.

DM10190:  Warning in <REMOVEWHITESPACE>:  Unable to locate field
<FieldName> in image <ImageName>.
```

NOTE: Naming the same field to move more than once in the parameters may cause unreliable results. The final location of a field's data is determined by the last movement of that field.

Also, this rule does not work with barcode type fields or multi-line text fields.

Image Editor example

Here is an example. Suppose you have these fields and data:

```
FIELD_A = ABCDEFG
FIELD_B =
FIELD_C =
FIELD_D = TUVWXYZ
```

Further suppose you name the fields in this order,

```
;RemoveWhiteSpace;FIELD_A,FIELD_B,FIELD_C,FIELD_D;
```

FIELD_A does not move because there is no earlier named field. FIELD_B and FIELD_C are empty. The data from FIELD_D will move to FIELD_B — the earliest field that is still empty. The result is:

```
FIELD_A = ABCDEFG
FIELD_B = TUVWXYZ
FIELD_C =
FIELD_D =
```

Now suppose you specify the field parameters like this:

```
;RemoveWhiteSpace;FIELD_D,FIELD_C,FIELD_B,FIELD_A;
```

The result is:

```
FIELD_A =
FIELD_B =
FIELD_C = ABCDEFG
FIELD_D = TUVWXYZ
```

See also

Section and Field Rules Reference on page 274

ResetImageDimensions

Use this section level rule (level 3) to reset the top or bottom dimensions of the current section based on the parameters you specify. This rule makes the objects on the section fit under the bottom of the header, over the top of the footer, or compresses the section to the smallest height possible, including all of its objects.

Syntax

```
;ResetImageDimensions;NewSize;;
```

Parameter	Description
NewSize	Choose one of these options: FooterTop - Makes the objects on the section fit over the top of the footer by resetting the bottom dimensions. HeaderBottom - Makes the objects on the section fit under the bottom of the header by resetting the top dimensions. MinHeight - Resizes the section to occupy the least amount of space. No objects are omitted or resized, but unused space is removed. LastField - Resizes the section to occupy the least amount of space. Only the fields with data are examined when the system looks for the lowest point on the section. Use this option on sections that have only fields or no objects lower than the last few fields you expect to map

The parameters are not case sensitive.

In legacy implementations, where only DDT files were loaded and not FAP files, fields are the only objects on a section the ResetImageDimension rule recognizes. That meant section bottoms were most likely being assigned at the last mapped field. In subsequent releases and because of new features and the new Studio model of development, FAP files are loaded during batch runs. Therefore, to get behavior similar to what you had in legacy implementations, you must either change the ResetImageDimension rules to use the LastField parameter, or use the RID_LastMapField INI option to change the behavior of the MinHeight parameter.

When you use the LastField option, the bottom of the section is moved to a position below the lowest mapped field on the section. This is what you want in situations where the next section should be placed immediately below where the last field was mapped. For instance, assume you have a small section used for addresses. It can contain up to eight lines, but depending upon the address only two or three lines might be used and you would like to set the bottom of the section below where the last field was mapped. Here is an example:

```
<Image Rules>
    ;ResetImageDimension;LastField;
```

NOTE: Depending upon your print or display method, changing the bottom of the section with this parameter could mean that any objects below this point will not be visible and may not print. Or it could mean those objects will simply overprint the next section in sequence on the same page.

You can also use the RID_LastMapField INI option to change the MinHeight parameter to work like the new LastField option described above.

```
< RunMode >
    RID_LastMapField = Yes
```

(*RID* is an abbreviation for ResetImageDimension.)

The default is No. Enter Yes if you want to modify the behavior of the MinHeight parameter on all sections that would use the ResetImageDimension rule.

NOTE: Depending upon your print or display method, changing the bottom of the section with this option could mean that any objects below this point will not be visible and may not print. Or it could mean those objects will simply overprint the next section in sequence on the same page.

Also note that the term *last field* refers to the *lowest* field mapped on a section and not the physical sequence in which the fields are mapped. The lowest field is the one that is the greatest distance from the top of the section.

Image Editor example

This example shows a DDT file excerpt:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,298,15965,0,600,0,480;
;SetOrigin;REL+0,MAX+0;
;PaginateBeforeThisImage;;
;ResetImageDimensions;MinHeight;
;DontPrintAlone;;
```

This example resets the section dimensions to occupy the least amount of space.

See also

SetImageDimensions on page 457

PaginateBeforeThisImage on page 421

Section and Field Rules Reference on page 274

ResetOvSym

Use this section level rule (level 3) to reset an overflow variable during the processing of a document set. Use this rule on the section level if you can not wait until the job level rule, ResetOvFlw, resets the entire set of overflow variables.

Syntax

```
;ResetOvSym;OverflowSymbol, SectionName;;
```

Parameter	Description
OverflowSymbol	The name of the overflow symbol defined in the SetOvFlwSym rule.
SectionName	The name of the section that contains the fields on which overflow processing will occur.

Image Editor example

```
;ResetOvSym;;Symbol_A,Section_AA;
```

This example tells the system to reset the overflow variable named *Symbol_A* which is used in the section named *Section_AA*.

See also

PurgeChartSeries on page 433

IncOvSym on page 366

OvActPrint on page 417

OvPrint on page 419

SetImageDimensions on page 457

Section and Field Rules Reference on page 274

SetGroupOptions

Use this section level rule (level 3) to set group options similar to forms. This rule lets you define the section as a header or footer and lets you specify whether or not the section should be copied onto the overflow section if overflow occurs.

Syntax

```
SetGroupOptions; (Header or Footer), CopyOnOverFlow;;
```

Parameter	Description
Header	Defines the sections that appear before the group.
Footer	Defines the sections that appear after the group.
CopyOnOverflow	Defines the sections that are copied to the new page if group pagination splits the group.

NOTE: The header and footer parameters are mutually exclusive.

Keep in mind...

- When a section contains both a GroupBegin rule and a SetGroupOptions rule, the GroupBegin rule must come first.
- When a section contains both a GroupEnd rule and a SetGroupOptions rule, the SetGroupOptions rule must come first.
- You must set all group pagination section options (footer, header, and CopyOnOverflow) using the SetGroupOptions rule.

Image Editor example

This DDT file excerpt defines this section as the header which should be copied to the new pages if the group pagination splits the group because of overflow.

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,1142,19718,0,0,0,0;
;SetOrigin;Rel+0,Max+100;
;GroupBegin;GroupPagination();;
;SetGroupOptions;header,copyonoverflow;
```

This DDT file excerpt defines this section as the footer which should be copied to the new pages if the group pagination splits the group because of overflow.

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,1142,19718,0,0,0,0;
;SetOrigin;Rel+0,Max+100;
;GroupBegin;GroupPagination();;
;SetGroupOptions;footer,copyonoverflow;
```

See also

GroupBegin on page 343

GroupEnd on page 355

Section and Field Rules Reference on page 274

RunDate

Use this field level rule (level 4) to get the run date from the transaction data (TRNFILE.DAT file) and format that date using the mask you specify.

The mask on the RunDate rule supports the following syntax:

A number (between 1 and 10 for compatibility with prior releases) and this format: *DinFmt:outFmt*

The *D* indicates a date conversion using the new method. Here is a list of the date formats you can choose:

Enter	To take a date in this format...	And output it in this format...
1	YYMMDD	MMDDYY
2	YYYYMMDD	MMDDYYYY
3	YYYYMMDD	MMDDYY
4	YYMMDD	MM-DD-YY
5	YYMMDD	MM/DD/YY
6	YYYYMMDD	MM-DD-YY
7	YYYYMMDD	MM/DD/YY
8	MMDDYY	MM-DD-YY
9	MMDDYY	MM/DD/YY
10	YYYMMDD	MM/DD/YY
D	*inFmt* is one of the standard date formats which consists of a format character, optional date separator, and an optional year size (2 or 4).	*outFmt* is also a standard date format for the destination field and is separated from the *inFmt* by a colon (:).

For compatibility with prior releases, masks (1 through 10) and the destination formats with a single letter, such as D, indicate the system will omit leading zeros or spaces. Also, please note that *Month* indicates both upper- and lowercase letters are used while *MONTH* indicates only uppercase letters are used. *Mon* indicates the month will be abbreviated in upper- and lowercase letters.

Using locales

If you use one of the standard formats, use the @XXX (without the percent). For example, D44@CAD is the standard format for Month DD, YYYY in Canadian French. If you are creating your own format, use %@???. For instance, D%@CAD%B %#d, %Y yields the same result as the standard format D44@CAD.

Keep in mind that the run date is typically stored in YYYYMMDD format and therefore does not require any locale information on the input format.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	Date
Offset	*
Length	10
Source name	REC-Date
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	RunDate
Mask	DM-4:44
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-Date;0;0;Date;0;10;DM-4:44;RunDate;;N;N;N;N;
```

So, if your RunDate extract (Input) field is in the format *DD-Mon-YYYY* and you want a form (output) field to have the format *Month D, YYYY*, you would define the mask like this:

```
DM-4:44
```

The *D* tells the system what conversion method to use. *M-4* indicates the input format is *DD-Mon-YYYY*. And *44* indicates the output format is *Month D, YYYY*.

Or, if you have used the new TRN_FIELDS conversion support to have the GenTrn program change the RunDate to *YYYYMMDD*, you can use this mask definition:

```
DD4:44
```

Here everything is the same except the input format of *D4*, which indicates *YYYYMMDD* as the format of RunDate.

Here is another example:

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	DATE
Offset	*
Length	8
Source name	REC-DATE
Offset	*
Length	*
File	*
Record	*
Required *	
Rule	RunDate
Mask	6
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;REC-DATE;0;0;DATE;0;8;6;RunDate;;N;N;N;N;
```

See also Formatting Data on page 257

Section and Field Rules Reference on page 274

SAPMove_It

Use this field level rule (level 4) for a *Move_It* type of operation on an SAP Raw Data Interface (RDI) extract file. This rule supports overflow.

Format mask

The format mask can consist of these options:

C Center the text

R Right justify the text (for non-proportional fonts only)

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	CAUFVD_P-MATNR
Offset	0
Length	10
Source name	CAUFVD_P-MATNR
Offset	173
Length	3
File	*1
Record	1 (generally not required unless you are also using overflow)
Required	*1
Rule	SAPMove_It
Mask	*1
Data	42,CAUFVD_P-MATNR

*1 means that no entry is required for this example

In the DDT file, this information looks like this:

```
;;;Caufvd_P-Matnr;173;3;Caufvd_P-Matnr;0;10;;SAPMove_It;42,
Caufvd_P-Matnr;;;;;
```

In this example, the rule gets the first occurrence in the extract file of a record matching the search criteria of CAUFVD_P-MATNR at offset 173. From the extract record, 3 characters (which contains the length of the data that will follow it) are moved to the output buffer.

The rule then reads the extract record again, this time from offset 176, and copies x characters (where x is the 3-byte length that was just read) from the extract record to the output buffer (which in this case is defined to be 10 characters in length).

This example shows the use of a user function and overflow symbol:

In this field...	Enter...
Destination name	CAUFVD_P-MATNR
Offset	0
Length	10
Source name	CAUFVD_P-MATNR
Offset	173
Length	3
File	*1
Record	1 (required for overflow)
Required	*1
Rule	SAPMove_It
Mask	*1
Data	@GETRECSUSED,S4TOP,S4TOPOVF/42,CAUFVD_P-MATNR

*1 means that no entry is required for this example

In the DDT file, this information looks like this:

```
;;1;Caufvd_P-Matnr;173;3;Caufvd_P-Matnr;0;10;;SAPMove_It;
@GetRecsUsed,S4Top,S4TopOvf/42,Caufvd_P-Matnr;;;;;
```

See also Move_It on page 393

Section and Field Rules Reference on page 274

SetAddr

Use this field level rule (level 4) to store and retrieve subsequent lines of a multiple line address. This rule is useful if you are setting up an address which may have three or four lines of information. For instance, some addresses include a suite or apartment number. If one of the middle address lines is missing, the SetAddr rule will format the address to omit any white space or blank lines. This rule supports overflow.

The first time the rule is called, the format mask field must contain an *F*. This initializes the function and loads the address lines into an array. The system then returns the first line of the address data. Subsequent address variable fields should contain an *N* in the format mask field and return the next available non blank address line from the array.

The data element of the DDT structure should contain the parameters necessary to obtain the multiple lines that make up the entire address record. Use of overflow with this rule only pertains to calls which have the format mask field set to *F*.

The data field of the DDT has two parts, the first is the search criteria to get the extract record which contains the address information. The second part, separated from the first by a space, consists of *offset,length pairs* of address information.

When the address table is built, if one of the address lines consists of all blanks no entry is made in the address table. This lets you remove blank lines in an address.

Here are the optional format mask parameters you can use:

Parameter	Description
C	Converts to sentence style where the first character is capitalized and the remaining characters are lowercased.
S	Suppresses the state from the last address line.
U	Converts the data returned by the rule into uppercase.

Image Editor example

For example, if an address record contained four fields such as name, P.O. Box, city & state, and ZIP code and these four data items were loaded into the table, if P.O. Box was empty, only the other three items would be loaded into the table.

NOTE: An address line cannot exceed 256 characters.

When the address table items are retrieved, the first three contain data and the fourth returns nothing. This lets you print the address on successive lines without having a blank line in the middle where the P.O. Box would be.

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter for field 1...	Enter for field 2...	Enter for field 3...
Destination name	ADDRESS1	ADDRESS2	ADDRESS3

In this field...	Enter for field 1...	Enter for field 2...	Enter for field 3...
Offset	1	1	1
Length	50	50	50
Source name	REC-ADDRESS1	*	*
Offset	*	*	*
Length	*	*	*
File	*	*	*
Record	*	*	*
Required	*	*	*
Rule	SetAddr	SetAddr	SetAddr
Mask	F	N	N
Data	17,ADDRECORD 35,40,75,40,115,40	*	*

* no entry required for this field in this example

In the DDT file, this information looks like this for the first variable field:

```
;0;0;REC-ADDRESS1;;;ADDRESS1;1;50;F;SetAddr;17,ADDRECORD
35,40,75,40,115,40;;;;;
```

The example above fills the address table and copies the first address line to the destination field. The first record matching the search criteria of 17,ADDRECORD is obtained and from it three separate entries are made into the address table. The first is the 40 characters starting at offset 35 of the record, the second for 40 characters starting at offset 75, and the last for 40 characters starting at offset 115.

To get the second and third address lines from the table, subsequent calls to the SetAddr rule must be made using format mask *N*, with nothing in the data field:

```
;0;0;ADDRESS2;;;ADDRESS2;1;50;N;SetAddr;;;;;;
;0;0;ADDRESS3;;;ADDRESS3;1;50;N;SetAddr;;;;;;
```

NOTE: In this example, the city, state, and ZIP code are together in the extract file and would be found by the entry for field 3. If the city, state, and ZIP are not formatted, see the SetAddr2 rule.

This example shows the use of a user function and overflow symbol:

In this field...	Enter...
Destination name	ADDRESS1
Offset	1
Length	50
Source name	REC-ADDRESS1
Offset	*
Length	*
File	*
Record	1
Required	*
Rule	SetAddr
Mask	F
Data	@GETRECSUSED,FORMABC,OVSYM/17,ADDRECORD 35,40,75,40,115,40

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;1;REC-ADDRESS1;;;ADDRESS1;1;50;F;SetAddr;@GETRECSUSED,
FORMABC,OVSYM/17,ADDRECORD 35,40,75,40,115,40;;;;;
```

See also

SetAddr2 on page 448

SetAddr3 on page 451

Section and Field Rules Reference on page 274

SetAddr2

Use this field level rule (level 4) to store and retrieve subsequent lines of a multiple line address. This rule is similar to the SetAddr rule in that it also omits blank lines from an address. The SetAddr2 rule, however, also formats the city, state, and postal code and adds a dash if you have a 10-digit ZIP code (ZIP+4). For instance, this rule automatically formats the city, state, and ZIP code as follows:

AtlantaGA 30333 (one space between state and ZIP code)

You can also specify additional formatting. For instance '^,'^' in the Data field (where ^ represents a space) tells the system to format the text as shown here:

Atlanta, GA 30333 (one space between the comma and the state, two spaces between the state and ZIP code)

In addition, you can also specify an S flag in the Data field to tell the system to suppress the state from the last address line. If you include this flag, the text is formatted as shown here:

Atlanta, 30333 (state code is suppressed)

The first time you call this rule, the format mask field must contain an *F*. This initializes the function and loads the lines of the address into an array. The system then returns the first line of the address data. Subsequent address variable fields should contain an *N* in the format mask field and return the next available non blank address line from the array.

The data element of the DDT structure contains the parameters necessary to get the address record. The last three fields (city, state, postal code) are stored in one field. The various address data element mapping comes from the first DDT record's data element (after the record mapping). This rule supports overflow.

NOTE: An address line cannot exceed 256 characters.

Here are the optional format mask parameters you can use:

Parameter	Description
C	Converts to sentence style where the first character is capitalized and the remaining characters are lowercased.
D	Converts the data returned by the rule into lowercase.
U	Converts the data returned by the rule into uppercase.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field…	Enter for field 1…	Enter for field 2…	Enter for field 3…
Destination name	ADDRESS1	ADDRESS2	ADDRESS3

In this field…	Enter for field 1…	Enter for field 2…	Enter for field 3…
Offset	0	0	0
Length	30	30	25
Source name	REC-ADDRESS1	REC-ADDRESS2	REC-ADDRESS3
Offset	45	65	0
Length	20	20	0
File	*	*	*
Record	*	*	*
Required	*	*	*
Rule	SetAddr2	SetAddr2	SetAddr2
Mask	F	N	N
Data	11,INSADRREC 25,20,45, 20,65,9,75,3,78,10 ',',5,'-'	*	*

* no entry required for this field in this example

In the DDT file, this information looks like this for the three variable fields:

```
;0;0;REC-ADDRESS1;45;20;ADDRESS1;0;30;F;SetAddr2;11,INSADRREC
25,20,45,20,65,9,75,3,78,10 ',',5,'-';;;;;
;0;0;REC-ADDRESS2;65;20;ADDRESS2;0;30;N;SetAddr2;;;;;
;0;0;REC-ADDRESS3;0;0;ADDRESS3;0;25;N;SetAddr2;;;;;
```

The first set of offsets (25,20) is for address line 1. The second set of offsets (45,20) is for address line 2. The third set of offsets (65,9,75,3,78,10) is for address line 3, which is normally used for the city (65,9), state or province (75,3), and postal code (78,10).

NOTE: Use a single space to separate the offsets from the format parameters (‘,’,5,’-’).

For the third line of address data, this example uses “11,INSADRREC” as the search criteria and “65,9,75,3,78,10”as the data mapping parameters. These parameters are used to format the city, state or province, and postal code: ', ',5,'-'. The comma (,) is placed between city and state or province. The dash (-) is placed after the 5th position in the postal code.

If the mapping parameters are: ',','5','-' or ',','5',' ' and the input data from the extract data contains a ZIP code of nine digits *with* a dash, then:

Input	Output
12345-6789	12345-6789
12345-0000	12345 (*without the dash*)
12345-(*four spaces*)	12345 (*without the dash*)

If the mapping parameters are: ',','5','-' or ',','5',' ' and the input data from the extract data contains a ZIP code of nine digits *without* a dash, then:

Input	Output
123456789	12345-6789
123450000	12345 (*without the dash*)
12345	12345 (*without the dash*)

See also SetAddr on page 445

SetAddr3 on page 451

Section and Field Rules Reference on page 274

SetAddr3

Use this field-level (level 4) rule to handle a three-line address with six components, as shown here:

- Address1 (placed on line 1)
- Address2 (placed on line 2)
- Address3 (placed on line 2)
- City (placed on line 3)
- State (placed on line 3)
- ZIP (placed on line 3)

Address1 is required. Address1 is handled using the Move_It rule.

Address2 and Address3 are placed on line 2. The components of line 2 can vary. If both Address2 and Address3 exist, both are placed on line 2 with the delimiter passed in by the rule. If only one exists, no delimiter is used. If neither Address2 or Address3 exists, the line 3 (City, State and ZIP) is moved up to line 2.

Line 3 contains the City, State and ZIP code with a comma placed between the city and state, and a space added after the state and before the ZIP code. The ZIP code is formatted based on the format flag.

This rule is similar to the SetAddr2 rule. If Address2 or Address3 are not applicable, the remaining lines move up into their places. The City, State, and ZIP always remain on the same line.

NOTE: An address line cannot exceed 256 characters.

The format mask must contain one of these options:

Option	Description
F	Initializes the function, fills the addr_ln array with the address components, builds the address lines, and returns the first line from the addr_ln array. The first time you call this rule the format mask field must contain an F.
N	Returns the next available non-blank address line from the addr_ln array. This mask is required for all subsequent SetAddr3 calls.

The various address data element mappings come from the first field rule record's Data element (after the record mapping).

The rule expects five fields to be mapped. If one is missing you will receive an error. This rule also assumes all address components are in the same record.

Here are the format parameters you can use:

Parameter	Description
C	Converts to sentence style where the first character is capitalized and the remaining characters are lowercased.
D	Converts the data returned by the rule into lowercase.
U	Converts the data returned by the rule into uppercase.

Image Editor example

Here are some examples based on this sample data:

Component	Text
Address1	Oracle Insurance
Address2	3353 Peachtree Road
Address3	Suite II - 900
City	Atlanta
State	GA
ZIP	30326

If all fields have data, the layout looks like this:

Oracle Insurance
3353 Peachtree Road
Atlanta, GA 30326

If Address3 does not print, the layout should look like this:

Oracle Insurance
3353 Peachtree Road
Atlanta, GA 30326

If Address2 does not print, the layout should look like this:

Oracle Insurance
Suite II - 900
Atlanta, GA 30326

If Address2 and Address3 do not print, the layout should look like this:

Oracle Insurance
Atlanta, GA 30326

Here are the DDT file entries for a SetAddr3 rule that has record address elements in the following locations:

Element	Description
100,30	The field offset and length for the addr2 field

Element	Description
130,30	The field offset and length for the addr3 field
160,18	The field offset and length for the city field
178,3	The field offset and length for the state field
181,9	The field offset and length for the ZIP field
', '	The format used between the city and state and between addr2 and addr3
Y	The ZIP format flag. If Y and the ZIP field is 9 positions, a dash appears after the fifth position. If N or the field is not 9 positions, it is mapped as is.

NOTE: The Mask field in the first SetAddr3 rule would contain the character *F*. The Data field for this rule would contain the following:

```
100,30,130,30,160,18,178,3,181,9 ', 'Y
```

The Mask field for the second and third SetAddr3 rules would contain the character *N* and the Data field would be blank.

Here is how it looks in the DDT file:

```
;0;0;ADDRESS LINE1;70;30;ADDRESS
LINE1;0;30;;Move_It;1,001;N;N;N;N;2412;2147;11011;
;0;0;ADDRESS LINE2;0;0;ADDRESS LINE2;0;30;F;SetAddr3;
100,30,130,30,160,18,178,3,181,9 ', 'Y;N;N;N;N;2400;2566;11011;
;0;0;ADDRESS LINE3;0;0;ADDRESS
LINE3;0;30;N;SetAddr3;;N;N;N;N;10488;2566;11011;
```

See also

SetAddr on page 445

SetAddr2 on page 448

Section and Field Rules Reference on page 274

SetCpyTo

Use the section level rule (level 3) to set the value of the SendCopyTo variable to a field name in the DDT file that definitely contains data.

Syntax

```
SetCpyTo (FieldName)
```

If the SendCopyTo variable contains a value, the GenPrint program will print at the bottom of a form the following text:

```
Send copy to ######
```

where ###### is the current recipient name.

Since the field you specify for this rule must contain data for the system to print the text *Send copy to* on the form, you will typically map the field with the Mk_Hard rule in the DDT file.

Image Editor example

```
;SetCpyTo;ExampleField;;
```

In this example, if *ExampleField* contains data, the system will print the text *Send copy to* on the form.

See also

Mk_Hard on page 388

Section and Field Rules Reference on page 274

SetCustChartAxisLabels

Use this section level rule (level 3) to take data values that have been mapped to variable fields and use them as custom axis labels on a chart.

Syntax

```
;SetCustChartAxisLabels;ChartName,FieldNames;;
```

Parameter	Description
ChartName	Name of the chart
FieldNames	Names of the variable fields you want to use as custom axis labels

This rule tells you if there are discrepancies in the number of axis labels you create using this rule and the number the system would create on its own.

NOTE: The system creates place holder labels if you do not specify enough labels and ignores others if you specify to many.

Image Editor example

Here is an example, shown in a DDT file excerpt:

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,6960,7029,0,600,0,0;
;SetOrigin;ABS+0,MAX+0;
;CusSetDynamicScaleAxis;Chart(CHART1), Search(51,GRFMTHYR),
Min(82,6), Max(88,6), Increment (94,6);
;SetCustChartAxisLabels;CHART1,Axis1,Axis2,Axis3,Axis4,Axis5,Axis6,
Axis7,Axis8,Axis9,Axis10,Axis11,Axis12,Axis13;
;FieldVarsToChartSeries;;

/* These fields override the lower level definitions for this */
/* section only.                                              */
<Image Field Rules Override>
;0;0;Axis1;61;1;Axis1;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;6367;4958;16
006;
;0;0;Axis2;62;1;Axis2;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;6427;4564;16
006;
;0;0;Axis3;63;1;Axis3;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2884;4324;16
006;
;0;0;Axis4;64;1;Axis4;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2884;4324;16
006;
;0;0;Axis5;65;1;Axis5;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;16
006;
;0;0;Axis6;66;1;Axis6;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;16
006;
;0;0;Axis7;67;1;Axis7;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;16
006;
;0;0;Axis8;68;1;Axis8;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;16
006;
;0;0;Axis9;69;1;Axis9;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;16
006;
;0;0;Axis10;70;1;Axis10;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4351;
16006;
```

```
;0;0;Axis11;71;1;Axis11;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;
16006;
;0;0;Axis12;72;1;Axis12;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;
16006;
;0;0;Axis13;73;1;Axis13;0;1;;Move_It;51,GRFMTHYR;N;N;N;N;2886;4326;
16006;
```

See also Section and Field Rules Reference on page 274

SetImageDimensions

Use this section level rule (level 3) on forms which are made up of *floating* sections.

NOTE: The system automatically inserts this rule for you if you save your DDT file in Image Editor. If you later resize the section, go to Image Editor and save the DDT file again to have the system update the section dimensions.

If you used the SetOrigin rule, the system automatically includes this rule in the section level rules section of your DDT file.

NOTE: Always let the system take care of this rule for you.

Image Editor example

This example shows an excerpt from a DDT file:

```
/*This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,298,15965,0,600,0,480;
;SetOrigin;REL+0,MAX+0;
;PaginateBeforeThisImage;;
;ResetImageDimensions;;
;DontPrintAlone;;
```

See also

SetOrigin on page 458

ResetImageDimensions on page 436

Section and Field Rules Reference on page 274

SetOrigin

Use this section level rule (level 3) to set the section overlay/page segment X and Y coordinates. Using this rule, you specify where the page segment will be placed on the printed page.

NOTE: You can also use the SetOriginI and SetOriginM rules. SetOriginI works just like SetOrigin except you enter X and Y coordinate information in inches, instead of FAP units. There are 2400 FAP units per inch.

SetOriginM works just like SetOrigin except you enter X and Y coordinate information in millimeters. There are approximately 98 FAP units per millimeter.

Use the SetOrigin rule if you prefer to enter these coordinates in FAP units.

Syntax

```
;SetOrigin;Fixed,X,Y,Form,Store(),ImageName;;
```

Parameter	Description
FIXED	(Optional) Anchors the section at the specified X and Y coordinates. This *must* be the first parameter listed. See Fixing a section's position on page 461 for more information.
X	Sets the X coordinate for a section. This determines the section's horizontal position.
Y	Sets the Y coordinate for a section. This determines the section's vertical position.
Form	(Optional) The form name on which the section exists. This parameter lets you define more than one SetOrigin rule for a section and specify which one applies based on the name of the form.
Store( )	(Optional) This parameter lets you store the current section coordinates in prefix-name variables for later use. The syntax is: `Store(prefix-name variable)` For Windows, the coordinates are stored in: `prefix-name.left, prefix-name.right, prefix-name.top, and prefix-name.bottom` The stored coordinates can be referenced by any subsequent SetOrigin rule used to place a section on the same form.

The X and Y coordinates are specified using a combination of the following parameter prefixes plus the addition or subtraction of FAP units. There are 2400 FAP units per inch. Here are the prefixes you can use:

Prefix	Description
Abs	Absolute page position based on 2400 units per inch. (supports overflow)
Rel	Relative to the last section's top left coordinate. (supports overflow)

Prefix	Description
Max	Relative to the last section's maximum edge. For example, *Rel+0,Max+600* would position the current section ¼ inch below the last section. (supports overflow)
Mpg	Relative to any section at the maximum edge. For example, *Abs+0,Mpg+600* places the current section ¼ inch below the lowest object currently on the page. (does not support overflow)
T2T	Top edge is placed relative to the last section's top edge. (Similar to Rel)
T2B	Top edge is placed relative to the last section's bottom edge. (Similar to Max+)
B2T	Bottom edge is placed relative to the last section's top edge. (Similar to Max-)
B2B	Bottom edge is placed relative to the last section's bottom edge.
L2L	Left edge is placed relative to the last section's left edge. (Similar to Rel)
L2R	Left edge is placed relative to the last section's right edge. (Similar to Max+)
R2L	Right edge is placed relative to the last section's left edge. (Similar to Max-)
R2R	Right edge is placed relative to the last section right edge.
Ctr	X/Y dimensions are centered on the last section's X/Y dimensions.

Image Editor example

```
;SetOrigin;Abs+0,Abs+1200;;
```

This example sets the origin for the section to the X, Y values of 0, 1200.

NOTE: There are no spaces between Abs+0,Abs+1200.

```
;SetOrigin;Rel+0,Max+0;;
```

This example shows you how to set up a section, which will print after the section, which precedes it. You define the order of the sections which make up a form in the FORM.DAT file using the Form Set Manager. The GenData program reads this information as it builds the print batches.

For instance, suppose IMAGE_A appears on two separate forms in the form set: FORM_1 and FORM_2.

```
;SetOrigin;ABS+0,ABS+0,FORM_1
;SetOrigin;ABS+0,ABS+2400,FORM_2
```

This example places the section at coordinate 0,0 when it appears on FORM_1 and an inch (0,2400) down the page when it appears on FORM_2.

Here is another example:

```
;SetOrigin;Abs+0,Abs+12000,Image1,Store(var1);;
```

This example sets the origin for Image1 to zero (0) FAP units for the X value (from the edge of form) and 12000 FAP units for the Y value (5 inches down form the top of the form). Plus, it stores the section coordinates into the prefix-name variable *VAR1*.

Here is another example:

```
;SetOrigin;Rel+0,VAR1.bottom+600;;
```

In this example, the current section would be positioned such that its left edge would be at the same x coordinate of the previous section's left edge but the top of the section would be 1/2 inch below the saved coordinates of a previous section—its coordinates were saved in the variable VAR1 using this rule:

```
;SetOrigin;x+n,y+n,external form name,Store(var1);;
```

Here is another example:

```
;SetOrigin;Img1.right+600,Img1.bottom-1200;;
```

This example places the current section's top left corner ¼ inch to the right and ½ inch up from Img1's bottom right corner.

Here is another example:

```
;SetOrigin;Rel+0,Max+300;;  (same as L2L+0,T2B+300)
```

In this example the current section is placed 1/4 inch below the previous section. The sections are aligned with their left edges.

Here is another example:

```
;SetOrigin;Rel+0(IMG1),Max+300;;  (same as L2L+0(IMG1),T2B+300)
```

In this example the current section is placed 1/4 inch below the previous section. The left edge of the current section is aligned with the left edge of section IMG1.

Here is another example:

```
;SetOrigin;Max+600,Rel+0;; (same as L2R+600,T2T+0)
```

The current section is placed 1/2 inch to the right of the previous section. The sections are aligned with their top edges.

Here is another example:

```
;SetOrigin;Abs+0,Mpg+600;;
```

The current section is placed 1/4 inch below the lowest object currently positioned on the page. The current section is aligned with the left edge of the page.

Here is another example:

```
;SetOrigin;Ctr+0,Max+0;; (same as Ctr+0,T2B+0)
```

The current section is centered immediately below the previous section.

Here is another example:

```
;SetOrigin;Ctr+0,Ctr+0;;
```

The current section is centered above the previous section.

Here is another example:

```
;SetOrigin;Ctr+0(IMG1),Ctr+0(IMG1);;
```

The current section is centered above IMG1.

Here is another example:

```
;SetOrigin;FIXED,ABS+2400,ABS+9731;
```

This fixes the current section at the coordinates 2400, 9731. Even if it is the last section triggered, the system will fix the section at those coordinates.

Fixing a section's position

The SetOrigin rule lets you designate a section as a *fixed* section. Fixed sections cannot be moved from their set position.

You declare a section to be fixed by including FIXED as the first parameter of the section's SetOrigin rule. Regardless of when the section is triggered, it will keep the coordinates assigned to it throughout pagination.

Fixed sections can also be designated as CopyOnOverflow sections. This will cause the fixed section to be copied onto subsequent pages at the same coordinates as those used for the first page.

The SetOrigin rule should always define the X,Y coordinates as an absolute position or a position relative to a section that will always be on the page. Do not anchor the fixed section relative to a section that can overflow onto a new page. If you do, the fixed section will never move to the second page. Instead, it will always appear on the first page with coordinates that place it below the bottom of the page.

Here is an example of how you would use the FIXED parameter:

```
;SetOrigin;Fixed,X,Y,Form,Store(),ImageName;;
```

NOTE: You must use the SetImageDimensions rule when you use this rule.

See also

SetOriginI on page 462

SetOriginM on page 464

SetImageDimensions on page 457

Section and Field Rules Reference on page 274

SetOriginI

Use this section level rule (level 3) to set the section overlay/page segment X and Y coordinates. Using this rule, you specify where the page segment will be placed on the printed page.

NOTE: You can also use the SetOrigin and SetOriginM rules. SetOrigin works just like this rule except you enter X and Y coordinates in FAP units, instead of inches. There are 2400 FAP units per inch.

SetOriginM works just like this rule except you enter X and Y coordinates in millimeters. There are approximately 25.4 millimeters per inch.

Syntax

```
;SetOriginI;Fixed,X,Y,Form,Store(),ImageName;;
```

Parameter	Description
FIXED	(Optional) Anchors the section at the specified X and Y coordinates. This *must* be the first parameter listed. See Fixing a section's position on page 461 for more information.
X	Sets the X coordinate for a section. This determines the section's horizontal position.
Y	Sets the Y coordinate for a section. This determines the section's vertical position.
Form	(Optional) The form name on which the section exists. This parameter lets you define more than one SetOrigin rule for a section and specify which one applies based on the name of the form.
Store( )	(Optional) This parameter lets you store the current section coordinates in prefix-name variables for later use. The syntax is: `Store(prefix-name variable)` For Windows, the coordinates are stored in: `prefix-name.left, prefix-name.right, prefix-name.top, and prefix-name.bottom` The stored coordinates can be referenced from any page in the form set.

Specify the X and Y coordinates using a combination of the following parameter prefixes plus the addition or subtraction of measurements you specify in inches. Here are the prefixes you can use:

Prefix	Description
Abs	Absolute page position based on inches. (supports overflow)
Rel	Relative to the last section's top left coordinate. (supports overflow)
Max	Relative to the last section's maximum edge. For example, *Rel+0,Max+.25* would position the current section ¼ inch below the last section. (supports overflow)

Prefix	Description
Mpg	Relative to any section at the maximum edge. For example, *Abs+0,Mpg+.25* places the current section ¼ inch below the lowest object currently on the page. (does not support overflow)
T2T	Top edge is placed relative to the last section's top edge. (Similar to Rel)
T2B	Top edge is placed relative to the last section's bottom edge. (Similar to Max+)
B2T	Bottom edge is placed relative to the last section's top edge. (Similar to Max-)
B2B	Bottom edge is placed relative to the last section's bottom edge.
L2L	Left edge is placed relative to the last section's left edge. (Similar to Rel)
L2R	Left edge is placed relative to the last section's right edge. (Similar to Max+)
R2L	Right edge is placed relative to the last section's left edge. (Similar to Max-)
R2R	Right edge is placed relative to the last section right edge.
Ctr	X/Y dimensions are centered on the last section's X/Y dimensions.

Image Editor example

```
;SetOrigin;Abs+1,Abs+1;;
```

This example sets the origin for the section to the X, Y values of 1 inch, 1 inch. Keep in mind there are no spaces between *Abs+1,Abs+1*.

NOTE: For additional example, see the examples for the SetOrigin rule on page 459. The only difference between this rule and the SetOrigin rule is that you use inches instead of FAP units.

See also

SetOrigin on page 458

SetOriginM on page 464

SetImageDimensions on page 457

Section and Field Rules Reference on page 274

SetOriginM

Use this section level rule (level 3) to set the section overlay/page segment X and Y coordinates. Using this rule, you specify where the page segment will be placed on the printed page.

NOTE: You can also use the SetOrigin and SetOriginI rules. SetOrigin works just like this rule except you enter X and Y coordinates in FAP units, instead of millimeters. There are approximately 98 FAP units per millimeter.

SetOriginI works just like this rule except you enter X and Y coordinates in inches. There are approximately 25.4 millimeters per inch.

Syntax

```
;SetOriginM;Fixed,X,Y,Form,Store(),ImageName;;
```

Parameter	Description
FIXED	(Optional) Anchors the section at the specified X and Y coordinates. This *must* be the first parameter listed. See Fixing a section's position on page 461 for more information.
X	Sets the X coordinate for a section. This determines the section's horizontal position.
Y	Sets the Y coordinate for a section. This determines the section's vertical position.
Form	(Optional) The form name on which the section exists. This parameter lets you define more than one SetOrigin rule for a section and specify which one applies based on the name of the form.
Store( )	(Optional) This parameter lets you store the current section coordinates in prefix-name variables for later use. The syntax is: `Store(prefix-name variable)` For Windows, the coordinates are stored in: `prefix-name.left, prefix-name.right, prefix-name.top, and prefix-name.bottom` The stored coordinates can be referenced from any page in the form set.

Specify the X and Y coordinates using a combination of the following parameter prefixes plus the addition or subtraction of measurements you specify in millimeters. Here are the prefixes you can use:

Prefix	Description
Abs	Absolute page position based on millimeters. (supports overflow)
Rel	Relative to the last section's top left coordinate. (supports overflow)
Max	Relative to the last section's maximum edge. For example, *Rel+0,Max+1* would position the current section 1 millimeter below the last section. (supports overflow)

Prefix	Description
Mpg	Relative to any section at the maximum edge. For example, *Abs+0,Mpg+1* places the current section 1 millimeter below the lowest object currently on the page. (does not support overflow)
T2T	Top edge is placed relative to the last section's top edge. (Similar to Rel)
T2B	Top edge is placed relative to the last section's bottom edge. (Similar to Max+)
B2T	Bottom edge is placed relative to the last section's top edge. (Similar to Max-)
B2B	Bottom edge is placed relative to the last section's bottom edge.
L2L	Left edge is placed relative to the last section's left edge. (Similar to Rel)
L2R	Left edge is placed relative to the last section's right edge. (Similar to Max+)
R2L	Right edge is placed relative to the last section's left edge. (Similar to Max-)
R2R	Right edge is placed relative to the last section right edge.
Ctr	X/Y dimensions are centered on the last section's X/Y dimensions.

Image Editor example

```
;SetOrigin;Abs+1,Abs+1;;
```

This example sets the origin for the section to the X, Y values of 1 millimeter, 1 millimeter. Keep in mind there are no spaces between *Abs+1,Abs+1*.

NOTE: For additional example, see the examples for the SetOrigin rule on page 459. The only difference between this rule and the SetOrigin rule is that you use millimeters instead of FAP units.

See also

SetOrigin on page 458

SetOriginI on page 462

SetImageDimensions on page 457

Section and Field Rules Reference on page 274

SetRecipFromImage

Use this section level rule (level 3) to conditionally add sections to the current form set based on conditions in the SETRCPTB.DAT file. You set up parameters which instruct the rule to generate a new set of keys (Key1, Key2, and TranID).

A new set of items from the current SETRCPTB.DAT file is generated and those items are run through the RunSetRcp feature to generate a temporary form set. This temporary form set is merged with the current form set to create the final form set.

NOTE: With version 11.3, Documaker Studio lets you create subforms. Using subforms you can include forms within forms which eliminates the need to use the SetRecipFromImage rule. This simplifies triggering and populating data on sections (images) when you are processing repeating patterns of hierarchical or nested data. Previously, you had to use the SetRecipFromImage rule, the sub extract rules, and overflow symbols to achieve the same result.

Syntax

```
;SetRecipFromImage (Key1) (Key2) (TranID) (AtEnd );;
```

Parameter	Description
Key1	Value
Key2	Value
TranID	Value to set new key values
AtEnd	If you set this parameter to True, the system adds the forms in the order you listed in your FORM.DAT file. The default is False, which tells the system to load the form backwards from the way it is referenced in the FORM.DAT file.

The string *$(Key1)* within *value* equals the current transaction's Key1 value. *$(Key2)* equals the current transaction's Key2 value. *$(TranID)* equals the current transaction ID.

Image Editor example

```
;SetRecipFromImage;Key1=JNLCS1 Key2=VARA1;
;SetRecipFromImage;Key1=$(Key1)1 Key2=$(KEY2)1;
```

The result from the second line is the same as the result from the first line if the current transaction's value for Key1 and Key2 are *JNLCS* and *VARA*.

Here is an example that shows the AtEnd parameter. Assume the CALLIMAGE.DDT contains this rule:

```
;SetRecipFromImage;Key1=SMPCOM1 Key2=SMPLOB1;
```

And this is the excerpt from the FORM.DAT file:

```
;SAMPCO;LOB;GAS BILLS;Gas Light;N;;CALLIMAGE|D(48)<CUSTOMER(0)>;
;SAMPCO;LOB;GAS BILLS2;Gas Bills2 Desc;N;;GASBILLS2|D<CUSTOMER(0)>;
;SMPCOM1;SMPLOB1;FORM1;FORM1 Desc;N;;FORM1a|D(48)<CUSTOMER(0)>/
FORM1b|D(48)<CUSTOMER(0)>;
;SMPCOM1;SMPLOB1;FORM2;FORM2 Desc;N;;FORM2a|D(48)<CUSTOMER(0)>/
FORM2b|D(48)<CUSTOMER(0)>;
;SMPCOM1;SMPLOB1;FORM3;FORM3 Desc;N;;FORM3a|D(48)<CUSTOMER(0)>/
FORM3b|D(48)<CUSTOMER(0)>;
```

Within the new Key1/Key2 combination, to which the SetRecipFromImage rule will jump, three forms are listed: FORM1, FORM2 and FORM3. By omitting the AtEnd parameter, these forms are added, assuming all forms/images are triggered, backwards from the way they are referenced in the FORM.DAT file. Therefore, the POLFILE.DAT will look like this:

```
;SAMPCO;LOB;GAS BILLS;Gas Light;R;;\
FORM3A|D48<CUSTOMER>/\
FORM3B|D48<CUSTOMER>/\
FORM2A|D48<CUSTOMER>/\
FORM2B|D48<CUSTOMER>/\
FORM1A|D48<CUSTOMER>/\
FORM1B|D48<CUSTOMER>;
;SAMPCO;LOB;GAS BILLS2;Gas Bills2 Desc;R;;\
GASBILLS2|D<CUSTOMER>;
\ENDDOCSET\ 11111-11111-1-11111
```

If you include the AtEnd parameter and set it to True, as shown here:

```
;SetRecipFromImage;Key1=SMPCOM1 Key2=SMPLOB1 AtEnd=True;
```

The POLFILE.DAT will look like this:

```
;SAMPCO;LOB;GAS BILLS;Gas Light;R;;\
FORM1A|D48<CUSTOMER>/\
FORM1B|D48<CUSTOMER>/\
FORM2A|D48<CUSTOMER>/\
FORM2B|D48<CUSTOMER>/\
FORM3A|D48<CUSTOMER>/\
FORM3B|D48<CUSTOMER>;
;SAMPCO;LOB;GAS BILLS2;Gas Bills2 Desc;R;;\
GASBILLS2|D<CUSTOMER>;
\ENDDOCSET\ 11111-11111-1-11111
```

See also CreateSubExtractList on page 317

Section and Field Rules Reference on page 274

SetState

Use this field level rule (level 4) to translate a numeric ISO code retrieved from an extract record into its equivalent state text. This rule supports overflow.

State table

Code	State	Code	State	Code	State
01	Alabama	54	Alaska	02	Arizona
03	Arkansas	04	California	05	Colorado
06	Connecticut	07	Delaware	08	District of Columbia
09	Florida	10	Georgia	52	Hawaii
11	Idaho	12	Illinois	13	Indiana
14	Iowa	15	Kansas	16	Kentucky
17	Louisiana	18	Maine	19	Maryland
20	Massachusetts	21	Michigan	22	Minnesota
23	Mississippi	24	Missouri	25	Montana
90	Nationwide	26	Nebraska	27	Nevada
28	New Hampshire	29	New Jersey	30	New Mexico
31	New York	32	North Carolina	33	North Dakota
34	Ohio	35	Oklahoma	36	Oregon
37	Pennsylvania	58	Puerto Rico	38	Rhode Island
39	South Carolina	40	South Dakota	41	Tennessee
42	Texas	43	Utah	44	Vermont
45	Virginia	46	Washington	47	West Virginia
48	Wisconsin	49	Wyoming		

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	STATE
Offset	1
Length	30
Source name	REC-STATE
Offset	30
Length	2
File	*
Record	*
Required	*
Rule	SetState
Mask	*
Data	17,PMSP0200

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;REC-STATE;30;2;STATE;1;30;;SetState;17,PMSP0200;;;;;
```

This example gets the two characters at offset 30 from the record obtained using the search criteria specified in the data field. The rule then looks in the table and returns the associated state text in the output buffer.

See also

Section and Field Rules Reference on page 274

SpanAndFill

Use this section level rule (level 3) to take a field and span its width between two other fields, filling the field with a fill character.

Syntax

```
;SpanAndFill SpanField, LeftField, RightField;;
```

Parameters	Description
SpanField	Enter the name of the field you want to span. This field must be the first parameter.
LeftField	Enter the field you want to be on the left.
RightField	Enter the field you want to be on the right.

NOTE: This rule does not move the field vertically. Only the width and horizontal location are changed.

The filler character is used to span the width between the end of the text in the left field and the beginning of the text in the right field. If either field is empty, the left coordinate of the field is used.

You can use any rule to map the fill character into the SpanField. For example, you can use the HardExst rule to map a character such as a period or asterisk. Only the first character of the mapped data is used as the filler character. If no data is found in the SpanField, the system uses periods (.) as the fill character.

NOTE: You may want to use the JustFld rule on your right-most field to make sure the field is right justified.

Keep in mind...

- If you use the Move_It rule, or other rules that support right justification by padding the data with spaces, your results will be incorrect. This rule calculates the width of a field based upon the entire contents and will not remove space from the fields.
- This rule loads the section (FAP or compiled FAP) if it is not already loaded.
- The font ID assigned to the SpanField is used for calculating the number of characters required to fill the width of the field.
- If there is fractional space remaining, the system place the extra white space to the left of the SpanField.

Image Editor example

Assume that...

For this parameter	You have this entry
LeftField	"ABCDEFG"
RightField	"$123.45"
SpanField	"."

And your DDT file contains...

```
;SpanAndFill; SPANFIELD, LEFTFIELD, RIGHTFIELD;;
```

The result will be...

```
ABCDEFG..........$123.45
```

Remember the horizontal location of the SpanField is moved to fill the gap between the left and right fields. The section designer handles the vertical alignment of fields.

See also

ConnectFields on page 312

Section and Field Rules Reference on page 274

StrngFmt

Use this field level rule (level 4) to format a string retrieved from an extract record, based on a given format string. The StrngFmt rule is useful for formatting Social Security numbers and phone numbers. This rule supports overflow.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	NUMBER
Offset	1
Length	15
Source name	REC-NUMBER
Offset	30
Length	9
File	*
Record	*
Required	*
Rule	StrngFmt
Mask	3,-,3,**
Data	17,PMSP0200

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;;REC-NUMBER;30;9;NUMBER;1;15;3,-,3,**;StrngFmt;17,PMSP0200;;;;;
```

This example gets the nine characters at offset 30 from the record selected using the search criteria you specified in the data field. The format mask is interpreted in pairs of offset and insert data. In this example, insert a dash (-) after the 3rd character and insert two asterisks (**) after the 3rd character from the previous insertion point.

The offset numbers that refer to insertion points always pertain to those points in the source data relative to the previous insertion.

For example, if the nine characters from the extract record were 123456789, the output buffer would contain 123-456**789. The format can be read as:

*skip 3, insert "-", skip 3, insert "**"*

This example shows the use of a user function and overflow symbol.

In this field...	Enter...
Destination name	NUMBER
Offset	1
Length	15
Source name	REC-NUMBER
Offset	30
Length	9
File	*
Record	1
Required	*
Rule	StrngFmt
Mask	3,-,3,**
Data	@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;1;REC-NUMBER;30;9;NUMBER;1;15;3,-
,3,**;StrngFmt;@GETRECSUSED,FORMABC,OVSYM/17,PMSP0200;;;;;
```

See also Section and Field Rules Reference on page 274

SysDate

Use this field level rule (level 4) to format the system date. Using this rule you can format the system date for different localities. This rule supports overflow.

The DDT mask area for the SysDate rule takes these values:

- output fetype
- output format mask

NOTE: There are two types of format mask, pre-defined types 1-9 and A-Q and user-defined format arguments. If the pre-defined formats meet your needs, use them, otherwise, create a user-defined format. For information on using pre-defined format types, see Using Pre-defined Date Formats on page 257.

User- defined format arguments consist of one or more codes, each preceded by a percent sign (%). For more information on user-defined format masks, see the Setting Up Format Arguments on page 262.

Image Editor example

Assume the system date is 03-01-2009, which is a Monday, and the time is 11:57 am.

The DDT format mask of:

```
d, "4/4
```

formats the system date using format 4, with month spelled out, such as March 1, 2009.

To produce a Canadian French date, such as mars 1, 2009, use the following DDT format mask:

```
DCAD, "4/4
```

The following format, which uses format arguments, will produce the same output:

```
dCAD, "%B %#d, %Y
```

Format arguments let you include the day of the week, hour, minute, second, and so on. This table shows you the results using various formats:

Format	Result
%m-%d-%Y	03-01-2009
The year is %Y.	The year is 2009.
Born %m/%d/%y at %I:%M %p	Born 03/01/09 at 11:57 am
%d	01
%#d	1
%A	Monday
%>A	MONDAY
%b	Mar

Format	Result
%<b	mar
%p	AM
%<>p	Am
%A, %B %d	Monday, March 01
%@CAD%A %@CAD%A, %B %d	lundi, mars 01
%A, %@CAD%B %d	Monday, mars 01
%@CAD%A, %@USD%B %d	lundi, March 01

See also

FfSysDte on page 331

Field Format Types (fetypes) on page 265

Formatting Data on page 257

Section and Field Rules Reference on page 274

TblLkUp

Use this field level rule (level 4) to find the record in a table that matches the first specification. After the system finds that record, it uses the offset you specify to get a key. The key is used to look up a final record and return the result.

NOTE: The size of a table row is set in the MaxExtRecLen option in the Trn_File control group. The maximum size is 1024 characters.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	CMP.NAME
Offset	*
Length	9
Source name	CO.NAME
Offset	*
Length	*
File	1
Record	*
Required	*
Rule	TblLkUp
Mask	*
Data	1,POLDTL 61,2 1,CO,7 11,9

* no entry required for this field in this example

NOTE: For this rule, the Mask field specifies a default look up value. It is not a standard mask like that used in the Move_It rule.

If you want the system to use the table files to look up a final record and return the result, specify a source file equal to any number except zero (0). If you leave the Source File field equal to zero (0), the system uses the extract file as your table.

Keep in mind that all tables you specify in your *Tblfile* are loaded into memory sequentially. This table is then used to search for the final record. If your search mask and key are not unique, you may end up with an incorrect result.

In the DDT file, this information looks like this:

```
;1;0;CO.NAME;;;CMP.NAME;;9;;TblLkUp;1,POLDTL   61,2   1,CO,7
11,9;;;;;6888;6208;7112
```

The rule does a search in the extract data for a record that matches the search mask 1,POLDTL. The 61,2 gets the two characters at offset 61 from the record found with the 1,POLDTL search mask.

Assume these characters turned out to be XY. The tables are then searched for a match on the search mask 1,CO,7,XY. If a matching record is found, the system maps the nine characters starting at offset 11.

You specify the table file in the Data control group in the FSISYS.INI file as follows:

```
TblFile=.\deflib\TBLFILE.DAT
```

In the TBLFILE.DAT file you would see a list of the fields in which the rule will search to find a match:

```
.\DEFLIB\AGENCY.TBL
.\DEFLIB\COMPCODE.TBL
```

To use this rule, the following rules must be in the AFGJOB.JDT file:

```
;CreateGlbVar;1;TblLstH,PVOID;
;LoadTblFiles;1;;
```

See also

MovTbl on page 413

LookUp on page 374

Section and Field Rules Reference on page 274

TblText

Use this field level rule (level 4) to get a text table item based on a key built from the source field name concatenated with the data retrieved from the source record. This rule supports overflow. Keep in mind these considerations, which pertain to the external ASCII text table referenced by this rule.

- Keys can be up to 12 characters in length.
- The key begins in position 1 in the text file.
- The returned text begins in position 14 in the text file.
- Only the first occurrence of the match is returned to the caller.

Each data line in the text table file must follow the following format:

```
KEY;ENTRY
```

where *KEY* is a value of up to 12 characters, padded right with spaces. *ENTRY* is the text data associated with the key entry.

NOTE: All support for this rule resides in a single table file. You specify the name of this file in the FSISYS.INI file in the TEXTTBL option in the Data control group.

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	TYPE OF POLICY
Offset	0
Length	20
Source name	TRANTYPE
Offset	16
Length	1
File	*
Record	1
Required	*
Rule	TblText
Mask	*
Data	17,00,15,A

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;1;TRANTYPE;16;1;TYPE OF POLICY;0;20;;TblText;17,00,15,A;;;;;
```

In this example, a key into the table is formed by concatenating the source field name, TRANTYPE, with the first character found at offset 16 from the record retrieved using the search criteria of 17,00,15,A. The source field name comes first in the key. The table is searched for a key match and the data associated with that key is written to the destination field TYPE OF POLICY for up to 20 characters.

This example shows the use of a user function and overflow symbol.

In this field...	Enter...
Destination name	TYPE OF POLICY
Offset	0
Length	20
Source name	TRANTYPE
Offset	16
Length	1
File	*
Record	1
Required	*
Rule	TblText
Mask	*
Data	@GETRECSUSED,FORMABC,OVSYM/17,00,15,A

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;;1;TRANTYPE;16;1;TYPE OF POLICY;0;20;;TblText;
@GETRECSUSED,FORMABC,OVSYM/17,00,15,A;;;;;
```

In the AFGJOB.JDT file, you must use the following:

```
;CreateGlbVar;1;TXTLst,PVOID;
:LoadTextTbl;1;;
```

See also Section and Field Rules Reference on page 274

TerSubstitute

Use this field level (level 4) rule to emulate TerSub entry functionality. You add this rule to a multi-line text field that has been designated as one which can grow.

Syntax

```
TerSubstitute; Key1  Key2  FormName  Recipient
```

Parameter	Description
Key1	Enter the name of Key1, such as Key1=AccountNo.
Key2	Enter the name of Key2, such as Key2=Name.
FormName	Enter the name of the form, such as FormName=XYZ.fap
Recipient	Enter the name of the recipient, such as Recipient=Agent.

NOTE: For all parameters, you can use the names of GVM variables instead of actual values.

The text to include in TerSubstitution is stored in the text areas of the sections. The sections are listed in the FORM.DAT file under a dummy Key1, Key2, and FormName. The rule finds the entry in the FORM.DAT file for the Key1,Key2,FormName parameters and uses the text from any sections with the supplied recipient.

Image Editor example

Assume you have the following data defined in your master resource library.

- Here is the FORM.DAT file:

```
;TerSub;Data;Form1;;N;;image1|D<CLIENT(0)>/
image2|D<CUSTOMER(0)>/image3.|D<CLIENT(0),CUSTOMER(0)>/
image4|D<CUSTOMER(0)>/image5|D<CLIENT(0),CUSTOMER(0)>/
image6|D<CUSTOMER(0)>/image7|D<CLIENT(0)>;

;TerSubA;DataA;FormA;;N;;image1|D<CLIENT(0)>/
image2|D<CUSTOMER(0)>/image3.|D<CLIENT(0),CUSTOMER(0)>/
image4|D<CUSTOMER(0)>/image5|D<CLIENT(0),CUSTOMER(0)>/
image6|D<CUSTOMER(0)>/image7|D<CLIENT(0)>;

;TerSubB;DataB;FormB;;N;;imagea|D<CLIENT(0)>/
imageb|D<CUSTOMER(0)>/imagec.|D<CLIENT(0),CUSTOMER(0)>/
imaged|D<CUSTOMER(0)>/imagez|D<CLIENT(0),CUSTOMER(0)>/
imagef|D<CUSTOMER(0)>/imageg|D<CLIENT(0),<AGENT(0)>;
```

- Here is an AFGJOB.JDT file which includes the Ext2GVM rule:

```
;CreateGlbVar;;Key1GVM,CHAR_ARRAY,10;
;CreateGlbVar;;Key2GVM,CHAR_ARRAY,10;
;CreateGlbVar;;RecipGVM,CHAR_ARRAY,10;
;CreateGlbVar;;FormGVM,CHAR_ARRAY,10;
;Ext2GVM;;11,GVMREC 20,10,Key1GVM;
;Ext2GVM;;11,GVMREC 30,10,Key2GVM;
;Ext2GVM;;11,GVMREC 40,10,RecipGVM;
;Ext2GVM;;11,GVMREC 50,10,FormGVM;
```

- You also have extract file transactions with a record which has a search mask equal to *11,GVMREC* and data at the following offsets and length.
- Each transaction could have different data in the specified record that would cause each transaction to have different sections in the multi-line text area.

```
Offset,Length

20,10= TerSubA
30,10= DataA
40,10= Customer
50,10= FormA
```

- A field prior to the multi-line text area that calls an external DAL script (SELIMAGE.DAL) plus a record (11,FlagRec) in each transaction that has a 1, 2, or 3 at offset 40. Here is an example of the SELIMAGE.DAL script:

```
flag = (11,FlagRec 40,1);
If flag = 1 Then;
        SetGVM(Key1GVM,  "TerSub");
        SetGVM(Key2GVM,  "Data");
        SetGVM(FormGVM,  "Form1");
        SetGVM(RecipGVM, "CLIENT");
                    ElseIf flag = 2;
        SetGVM(Key1GVM,  "TerSubA");
        SetGVM(Key2GVM,  "DataA");
        SetGVM(FormGVM,  "FormA");
        SetGVM(RecipGVM, "CUSTOMER");
                    ElseIf flag = 3;

        SetGVM(Key1GVM,  "TerSubB");
        SetGVM(Key2GVM,  "DataB");
        SetGVM(FormGVM,  "FormB");
        SetGVM(RecipGVM, "AGENT");
End;
```

Based on these assumptions, here are some examples:

- If your DDT file contains...

  ```
  ;0;0;SETFIELDTEST;0;0;SETFIELDTEST;0;0;;TerSubstitute;Key1=TerSub
  Key2=Data Recipient=CLIENT FormName=Form1;N;N;N;N;4513;2194;11010;
  ```

  only the sections (image1, image3, image5, and image7) are placed in the multi-line text field.

- If your DDT file contains the following and your AFGJOB.JDT file includes the above rules...

  ```
  ;0;0;SETFIELDTEST;0;0;SETFIELDTEST;0;0;;TerSubstitute;Key1=Key1GVM
  Key2=Key2GVM Recipient=RecipGVM
  FormName=FormGVM;N;N;N;N;4513;2194;11010;
  ```

  only the sections (image2, image3, image4, image5, and image6) are included in the multi-line text field.

- If your DDT file contains the following, has a record (11,FlagRec) which has a 3 at offset 40, and the external DAL script (SELIMAGE.DAL) is in the DefLib directory...

  ```
  ;0;0;SETFIELDTEST;0;0;SETFIELDTEST;0;0;;TerSubstitute;Call("SELIMAG
  E.DAL");N;N;N;N;4513;2194;11010;
  ```

  only the section named *imagez* is included in the multi-line text field.

See also Section and Field Rules Reference on page 274

TextMergeParagraph

Use this section level rule (level 3) to merge data for embedded variable fields in a text area with text. The Move_It or MoveNum rules are most often used with this rule to move data from the extract file into embedded variable fields. The system then rewraps the text area.

The system writes the FAP information into the NAFILE.DAT file, which is used by the GenPrint program. If a section includes this rule, the Can Grow attribute setting in the section's FORM.DAT file must match the text area's Can Grow attribute in the FAP file.

NOTE: If you include the Can Grow attribute for a multi-line text field, be aware the field can both grow and shrink, depending on the data. If you do not want the text area to change sizes, turn off the Can Grow attribute.

Using this rule can slow performance, so use it only as necessary. If you must use this rule, it is better not to mix other objects with the text area in this FAP file. The more objects are mixed, the worse your performance, because all of the information about these objects will be written to NAFILE.DAT file also. If these objects are separated into another FAP file, they can be part of the compiled overlay and need not be loaded into the NAFILE.DAT file.

Performance is affected even more if you include graphics in the FAP file *and* you are sending the data stream to an AFP printer. This is because the LoadFAPBitmap option in the RunMode control group is set to Yes and is needed to print the graphics. This INI option also affects performance. Avoid it as much as possible.

NOTE: System variables, such as Send Copy To:, cannot be used as an embedded variable field text area.

Syntax

```
TextMergeParagraph( )
```

No parameters are necessary. Only include this rule if your section has embedded variable fields in a text area.

Image Editor example

```
;TextMergeParagraph;;;
```

Keep in mind that if you use this rule and the MoveNum rule, you should left-justify the data to avoid leading spaces.

See also

Move_It on page 393

MoveNum on page 402

Section and Field Rules Reference on page 274

UnderlineField

Use this section level rule (level 3) to draw an underline beneath a variable field.

The system does not store the line in the NAFILE.DAT file. Instead, it turns on the underline attribute (U) in the option field of the NAFILE record. You will not see the underline in the NAFILE.DAT file.

Syntax

```
;UnderlineField;field name;
```

To underline multiple variable fields, you must make an entry for each field.

NOTE: This rule does not work with sections which have the copy on overflow attribute enabled.

Image Editor example

```
/* This section uses these rules */
<Image Rules>
;SetImageDimensions;98,0,3000,20400,0,0,0,0;
;SetOrigin;Rel+0,Max+0;
;IncOvSym;OVSYM3,QCPV5;
;UnderlineField;CLASSIFICATN 1;

/* By default, this section contains the following fields */
<Image Fields>

/* The following fields override the lower level definitions for */
/* this section only.                                           */
<Image Field Rules Override>
;0;1;CLCODE;25;6;CLASSIFICATN
1;0;30;;Move_It;@GETRECSUSED,QCPV5,OVSYM3/11,CLSSCDREC;...
;0;1;CLSSCDREC-CODE;25;6;CODE NO.
1;0;6;;Move_It;@GETRECSUSED,QCPV5,OVSYM3/11,CLSSCDREC;...
```

See also Section and Field Rules Reference on page 274

XDB

Use this field level rule (level 4) to tell the system the field has been mapped in the XDB database. Use this rule when you have variable fields which are used on multiple sections.

NOTE: You should use the XDD rule, on page 488, instead of this rule. This rule is included in this version of the system only for legacy system support.

Syntax

```
XDB
```

Instead of mapping these identical variable fields, like Name and Address, each time they are used, you can map them once in the XDB database and then map the individual fields to the XDB rule. This tells the system to look in the XDB database for the complete mapping information for those variable fields.

Keep in mind, however, that these fields do not exist in the dictionary:

- SrcFile (source file)
- SrcRec (source record number)

In the DDT mapping, the SrcFile is saved as a number — not an actual file name. It is used in the TblLkup rule and becomes the index to use to find the table you want to look into for this rule.

So, if you want to use the TblLkup rule, you must define this source file variable within the field map definition.

Similarly, to reference a specific source record, you must define SrcRec in the field's mapping. For example, you may have an overflow detail record which is identified by *1,Detail*. For a certain field, however, you want the data from the second detail record to be mapped. In this case, the DDT for this particular field must contain a SrcRec. Otherwise, the data from the first record will be used.

Mapping

You can include an asterisk (*) to tell the system to add a space before it concatenates the search masks. This makes the XDB and token lookup more flexible and lets you use XML for parent/child mapping.

The child can also be another search mask or XPath, instead of just being the rule parameter. To maintain the same search mask for the Child as that shown in the above example, however, you must add an asterisk (*) in front of the Child's data if it was used as a rule parameter.

You must also set up a correct search mask (XPath) syntax if a child's parent references another parent.

NOTE: The use of an asterisk was added in version 11.0 and patched back to version 10.3. Prior to this change, the system automatically added a comma for you. To make this work for all implementations, the system cannot assume a comma is always needed. For example, an XML implementation would not want a comma added before the two XPaths are appended together.

Name	Parent	Rule	Data
Child1	Parent2	PrintIf	*A=Accident:C=Casualty
Child2	Parent1	Move_It	,35,ZZZ
Parent2	Parent1		,20,ABC
Parent1			1,HEADREC

```
Child1 - 1,HEADREC,20,ABC A=Accident:C=Casualty
```

Because there is no asterisk in Child2's data, the complete search mask for Child2 becomes:

```
1,HEADREC,35,ZZZ.
```

Here is an example for the XML implementation:

Name	Parent	Rule	Data
Child1	Parent2	PrintIf	*A=Accident:C=Casualty
Child2	Parent1	Move_It	/JHI
Parent1			!/ABC
Parent2	Parent1		/DEF

In this example, the mapping for Child1 becomes:

```
!/ABC/DEF A=Accident:C=Casualty
```

The mapping for Child2 becomes:

```
!/ABC/JHI
```

In addition, you can name a parent within each child. The set up is similar to that used for token lookup. When using token lookup, the data's portion in the DDT line contains the set up for this. Here is an example:

```
?Child/Parent
```

For the XDB rule, the source field name would contain this set up without a question mark. Here is an example:

```
;0;0;Child/Parent1;0;0;Child;0;20;;XDB;;N;N;N;N;7577;2273;11114;
```

Name	Parent	Rule	Data
Child	Parent1	PrintIf	*A=Accident:C=Casualty
Child	Parent2	Move_It	/JHI
Parent1			!/ABC
Parent2			!/ABC/DEF

The mapping for Child/Parent1 is:

```
!ABC A=Accident:C=Casualty
```

The mapping for Child/Parent2 is:

```
!/ABC/DEF/JHI
```

Image Editor example

If you make the following entries on the Edit DDT tab of the field's Properties window in Image Editor:

In this field...	Enter...
Destination name	Name
Offset	*
Length	32
Source name	Name
Offset	*
Length	*
File	*
Record	*
Required	*
Rule	XDB
Mask	*
Data	*

* no entry required for this field in this example

In the DDT file, this information looks like this:

```
;0;0;Name;;;Name;;;XDB;;;;;;15960;4000;17200;
```

See also

Formatting Data with the = Operator on page 267

TblLkUp on page 476

XDD on page 488

Section and Field Rules Reference on page 274

XDD

Use this field level rule (level 4) to tell the system the field has been mapped in the XDD database. Use this rule when you have variable fields which are used on multiple sections.

NOTE: . The XDD and XDB rules are synonymous. When encountered in a Studio MRL, the XDD is used from the library. If these rules are used in a MRL that is legacy-based, the XDB database is used.

Syntax

```
XDD
```

Instead of mapping these identical variable fields, like Name and Address, each time they are used, you can map them once in the XDD database and then map the individual fields to the XDD rule. This tells the system to look in the XDD database for the complete mapping information for those variable fields.

In the DDT mapping, the SrcFile is saved as a number — not an actual file name. It is used in the TblLkup rule and becomes the index to use to find the table you want to look into for this rule.

So, if you want to use the TblLkup rule, you must define this source file variable within the field map definition.

Similarly, to reference a specific source record, you must define SrcRec in the field's mapping. For example, you may have an overflow detail record which is identified by *1,Detail*. For a certain field, however, you want the data from the second detail record to be mapped. In this case, the DDT for this particular field must contain a SrcRec. Otherwise, the data from the first record will be used.

NOTE: SrcRec is only necessary if you know the specific instance of the data that you wish to use. Typical overflow can be mapped in the XDD and does not involve the SrcRec mapped at the field level.

Mapping

You can include an asterisk (*) to tell the system to add a space before it concatenates the search masks. This makes the XDD rule and the token lookup more flexible and lets you use parent/child mapping.

The child can also be another search mask or XPath, instead of just being the rule parameter. To maintain the same search mask for the Child as that shown in the above example, however, you must add an asterisk (*) in front of the Child's data if it was used as a rule parameter.

You must also set up a correct search mask (XPath) syntax if a child's parent references another parent.

NOTE: The use of an asterisk was added in version 11.0 and patched back to version 10.3. Prior to this change, the system automatically added a comma for you. To make this work for all implementations, the system cannot assume a comma is always needed. For example, you would not want a comma added before the two XPaths are appended together.

Name	Parent	Rule	Data
Child1	Parent2	PrintIf	*A=Accident:C=Casualty
Child2	Parent1	Move_It	,35,ZZZ
Parent2	Parent1		,20,ABC
Parent1			1,HEADREC

```
Child1 - 1,HEADREC,20,ABC A=Accident:C=Casualty
```

Because there is no asterisk in Child2's data, the complete search mask for Child2 becomes:

```
1,HEADREC,35,ZZZ.
```

Here is an example:

Name	Parent	Rule	Data
Child1	Parent2	PrintIf	*A=Accident:C=Casualty
Child2	Parent1	Move_It	/JHI
Parent1			!/ABC
Parent2	Parent1		/DEF

In this example, the mapping for Child1 becomes:

```
!/ABC/DEF A=Accident:C=Casualty
```

The mapping for Child2 becomes:

```
!/ABC/JHI
```

In addition, you can name a parent within each child. The set up is similar to that used for token lookup. When using token lookup, the data's portion contains the set up for this. Here is an example:

```
?Child/Parent
```

For the XDD rule, the source field name would contain this set up without a question mark.

Name	Parent	Rule	Data
Child	Parent1	PrintIf	*A=Accident:C=Casualty

Name	Parent	Rule	Data
Child	Parent2	Move_It	/JHI
Parent1			!/ABC
Parent2			!/ABC/DEF

The mapping for Child/Parent1 is:

```
!ABC A=Accident:C=Casualty
```

The mapping for Child/Parent2 is:

```
!/ABC/DEF/JHI
```

Studio example

You could make the following entries in Studio in the Rule section of the Field Options panel:

In this field...	Enter...
Rule	XDD
Destination offset	*
Source name	Name
Source offset	*
File	*
Length	*
Record	*
Required	*
Overflow Multiplier	*
Overflow	*
Mask	*
Data	*

* No entry is required unless you intend to override the setting that will be inherited when the source field is found in the XDD.

See also

Formatting Data with the = Operator on page 267

TblLkUp on page 476

Section and Field Rules Reference on page 274

Appendix A

Using Condition Tables and the Record Dictionary

In this appendix you will find information about...

Using Condition Tables

Condition tables provide a simple and efficient way to set conditions. The system reads the conditions from an input file and then uses those conditions to trigger sections. When the system receives a file of conditions which are used by the rules, it then...

- Compiles the conditions for evaluation
- Evaluates the conditions for each transaction

Setting Up the INI Files

To use the Condition tables, you must make these changes to your FSISYS.INI file.

- Enter the path for your table files in the MasterResource control group. Use the TablePath option to define your table files path:

```
< MasterResource >
    TablePath =  \T4\UtilTest\MstrRes\TblLib\
```

- Enter the name of your Condition table file in the Tables control group. Use the Conditions option to define the Condition table's file name:

```
< Tables >
    Conditions =  CondTbl.tbl
```

Using a Record Dictionary File

Condition tables use the Record Dictionary to resolve variables. See the Using the Record Dictionary on page 495 for more information.

Here is an example from the Record Dictionary:

```
* These are the Record definitions

<Records>
Account   = Search(61,02)
MeterRead = Search(61,03) Repeating
Detail    = Search(61,18) Repeating

* These are the variable definitions

<Variables>
ACSA = Record(Account) Offset(487) Length(10) Type(Zone)
Format(14.2,C)
BMV1 = Record(Message) Offset(159) Length(15) Type(Char)
DTV1 = Record(Detail)  Offset(181) Length(18) Type(Zone)
Format(18.2,C)
CustomerType = Record(Detail) Offset(100) Length(1) Type(Char)
BIGTEST = Record(Detail) Offset(253) Length(18) Type(Zone)
Format(18.2,C)
               RPN(BIGTEST 5 * 30.55 + DTV3 + DTV5 -)
```

CREATING A CONDITIONS FILE

Conditions consist of combinations of comparisons, parentheses, and ANDs, and ORs to verify the correct results. Conditions are stated in this format:

```
ConditionName : {valid conditions}
```

Conditions can use the following:

- Variable names from the Record Dictionary
- Quoted strings
- Numeric constants
- Comparison operators, such as <, >, =, <=, >=, <>, !=, !<, and !>
- ANDs
- ORs
- Parentheses ( )
- Reserved words, such as *ZERO* and *SPACES*

Here are some examples:

```
Cond1 : ACSA > 9900 OR  (ACSC = 4173 AND DTAT = ZERO)
Cond2 : (DTV1 = 3936.50 OR DTAT > 1) AND (DTV1 > -2 OR DTAT = 0)
Cond3 : (BMV1 <> SPACES AND ( DTVM1 = "CREDIT" OR DTAT > 0 AND BMV1
= "PAYMENT"
Cond4 : CustomerType  = "A"
```

NOTE: The variables used above are defined in the Record Dictionary example.

Occurrence Counting

Occurrence counting uses the following format:

```
OccurName : OCCURRENCE(RecordName,ConditionName) MAX(Count)
```

Here is an example:

```
Occur1 : OCCURRENCE(Detail,Cond4)
```

The record *Detail* and the condition *Cond4* are used above as defined in the previous examples. The occurrence condition *Occur1* is driven by the record named *Detail*. The record must be of *Repeating* type.

The condition *Cond4* references the variable *CustomerType*. *CustomerType* is defined on the *Detail* record. There must be a connection between the record and a variable in the condition for the occurrence count to work correctly.

```
<Conditions>
MSG1        : LNPRTY < 10000
MSG2        : LNPRTY >= 10000
MSGTRGR1    : Occurrence(Message, MSG1)
totcurrchrg: DTLSECTION = "01" and LNPRIORITY != 00625
totamtdb    : TOTAL >= 0 AND BBFLAG = "N"
env         : EDIVERT ="0"
emitgrapha  : GRAPHTRUE = "28" AND SCALETRUE = "29" AND BUDBLTRUE =
"N"
*Triggers grouping for Tariff, Rider and Detail records
IAMBDTLA    : DTLKeyProd = TARKeyProd and DTLTarSeqNo = TARTarSeqNo
REMAINDER   : USAGEREM != 0
REGTARIFF   : Occurrence(Tariff, REGBUS)
REGBUS      : CDBUS != "0700" and CDBUS != "0100"
```

Setting a maximum count to return

Include the MAX parameter if you want to set a maximum count to be returned. Here is an example of the format for occurrence counting. Here is an example:

```
Occur1 : OCCURRENCE(Detail,Cond4) MAX(5)
```

Assume Cond4 is defined as shown here:

```
Cond4 : CustomerType = "A"
```

The occurrence condition Occur1 is driven by the record named Detail. The record must be of Repeating type.

Condition tables and the RecipCondition rule

One example of using Condition tables is to call the RecipCondition rule. In the SETRCPTB.DAT file, call the rule, as shown here:

```
;ORACLE;REGION2;REG;;01;;;M0;0;0;0;;RecipCondition;Cond1;
;ORACLE;REGION2;REG;;01;;;M0;0;0;0;;RecipCondition;Occur1;
```

Using the Record Dictionary

The Record Dictionary lets you define and access variables easily and efficiently. Variables are loaded from the extract file according to their definitions in the Record Dictionary file. You can use the Record Dictionary any time you need data from the extract list. The data can be in a numeric, character, or date format.

The Record Dictionary definitions are loaded from a text file. The variables can be referenced by name once the dictionary file has been loaded. For instance, used with Condition tables, any variable in the Record Dictionary can be used in a conditional evaluation.

Setting Up the Record Dictionary

Enter the path for your table files under the MasterResource control group in the FSISYS.INI file. Use the TablePath option to define your table files path:

```
< MasterResource >
    TablePath =  \T4\UtilTest\MstrRes\TblLib\
```

Enter the name of your Record Dictionary file in the DataDictionary control group. Use the Name option to define the Record Dictionary file name:

```
< DataDictionary >
    Name =  DataDict.Tbl
```

Record Dictionary File

The Record Dictionary must be populated with the variables you want to use. The file consists of two parts:

- <Records> section
- <Variables> section

Records

The record parameters are defined in the format:

```
RecordName = SEARCH(Column,SearchMask) {Repeating}
```

Parameter	Description
RecordName	The name that future references to this record will use.
Column	The column number that will be searched.
SearchMask	The text to look for in the column.
Repeating	(Optional) Can be set for any record that is of repeating type. You must set this flag when you are using the pointer to reference multiple records.

Variables

The variable parameters are defined using this format:

```
VariableName = Record(RecordName) GVM(GVM_Variable) Offset(Offset)
    Length(Length) Type(TypeVariable) Format(FormatFlags)
    Rule(RuleName) Data(RuleData) Precision(Precision)
    RPN(RPN Equation)
```

Parameter	Description
VariableName	The name future references to this variable will use. A variable name begins with an alpha character and can consist of up to 30 characters.
RecordName	The previously defined record (from the Record section) on which this variable will be found.
GVM_Variable	The name of the global variable to use.
The RecordName and GVM_Variable parameters are mutually exclusive.	
Offset	The offset into the record where the data is located.
Length	The length of the data.
TypeVariable	(Optional) Char, Num, Zone, or Packed. Char is character data. Character data can be any string of alphanumeric characters and symbols. Num is numeric data. Numeric data can have a sign in front and a decimal place. Zone is zoned decimal. Zoned decimal looks like a numeric value except the sign is added to the last digit. Packed is packed decimal. Packed decimal is a binary format used mainly on z/OS systems.
FormatFlags	(Optional) Similar to the flags used with MoveNum rule except the input flags, such as input length and precision, S, and B, are not needed.
RuleName	(Optional) You can include any field rule such as DateFmt or SetAddr2. The Move_It and MoveNum rules are inherent to the Record Dictionary, so you do not need to specify them. If you omit the rule, the Move_It rule functionality is the default.
RuleData	(Optional) Any required rule data for the RuleName entry.
Precision	(Optional) The number of decimal places for a numeric variable.
RPN Equation	Reverse Polish Notation function. See the RPN Function section below.

NOTE: Include a single space between variable parameters. A carriage return indicates the end of the variable definition. If you omit the length of a GVM-based Record Dictionary variable in the Record Dictionary entry, the system uses the length of the source GVM variable.

RPN Function

The RPN (Reverse Polish Notation) function handles mathematical operations in the Record Dictionary. The RPN function is used as a parameter of a variable in the Record Dictionary. Use Reverse Polish Notation to express your equation. Any variables that are referenced must be previously defined in the Record Dictionary.

- Compile the RPN equation into a linked list.
- Retrieve information from Record Dictionary for each variable.
- Evaluate the equation and return the resulting value.

Use the format:

```
RPN(valid RPN equation)
```

A valid RPN equation can include: variables, numeric constants, arithmetic operators (+,-,*,/,%), and several functions (MOD, ABS, DUP, SWAP, POW, SQRT, CEIL, FLOOR). When using a function, place a '#' sign before the function name (example: #MOD). This distinguishes a function name from a variable name.

RPN can also be used with date format variables. This can be useful when adding to a date or calculating an age. Here are some examples:

- BIGTEST = Record(Detail) Offset(253) Length(18) Type(Zone) Format(18.2,C) RPN(BIGTEST 5 * 30.55 + DTV3 + DTV5 -)
- LittleTEST = Type(NUM) RPN(BIGTEST 5 *) Format(18.2,C)
- SumTest = Type(num) RPN(BigTest LittleTest + #ABS)

RPN or Reverse Polish Notation is an arithmetic method that performs calculations from left to right. A stack is created to hold numeric values until an operation is performed. For instance, a simple equation such as " 1 + 2 " would be represented as " 1 2 + ". During computation, the stack would first hold "1", then it would be given "2". When the "+" is reached, the "1" and "2" are taken off the stack and added together. A slightly more complicated equation such as " (1 + 2) * 5 " would be represented as " 1 2 + 5 * ".

NOTE: No parentheses are needed in RPN logic.

Available RPN functions

These are the available functions in RPN. When using them, remember to place a "#" sign in front of the function name. This distinguishes a function name from a variable name.

Function	Description
ABS	References the most recent value and returns the absolute value of that number.
CEIL	Returns the next largest integer value of a number (round up).
DUP	Creates a duplicate of the top value in the stack.
FLOOR	Returns the next smallest integer value of a number (round down).
MAX	Compares the top two values on the stack and returns the larger.
MIN	Compares the top two values on the stack and returns the smaller.
MOD	Performs a division with the top two values on the stack and returns the remainder.
POW	Removes the top two values in the stack. Calculates the first to the power of the second.
SQRT	Returns the square root of the number.
SWAP	Removes the top two values in the stack and replaces them in reverse order.

Record Dictionary Rules

You can use the following rules to reference the Record Dictionary and its contents. The system loads variables from the extract file based on the variable definitions in the Record Dictionary file.

You can use the Record Dictionary any time you need data from the extract list. The data can be in a numeric, character, or date format.

Base_FromDataDictToGVM

Use this rule to copy a Record Dictionary value into a global variable. Place this rule in the AFGJOB.JDT file.

Syntax

```
;Base_FromDataDictToGVM;; GVM(GlobalVariableName)
DATA(DataDictVariableName);
```

Example

```
;Base_FromDataDictToGVM;;GVM(STATION1) DATA(OMR1);
```

FromDataDict

Use this rule to get data from variable fields from the Record Dictionary. Place this rule in the DDT file.

Syntax

```
;FromDataDict;DataDictVariableName {MoreOptionalVariables};;
```

Example

```
;0;0;KWH-ON-COM;0;0;KWH-ON-COM;0;10;;FromDataDict;ComkWh "and "
ComkWh2;N;N;N;N;2446;1218;16229;
```

FromDataDictToGVM

Use this rule to copy a Record Dictionary value into a global variable. Place this rule in the DDT or JDT file.

Syntax

```
;Base_FromDataDictToGVM;; GVM(GlobalVariableName)
DATA(DataDictVariableName);
```

Example

```
;Base_FromDataDictToGVM;;GVM(STATION1) DATA(OMR1);
```

Image_FromDataDictToGVM

Use this section level rule to copy a Record Dictionary value into a global variable. Place this rule in the DDT file.

Syntax

```
;Base_FromDataDictToGVM;;
GVM(GlobalVariableName)DATA(DataDictVariableName);;
```

Example

```
;Base_FromDataDictToGVM;;GVM(STATION1)DATA(OMR1);;
```

IncDataDictRecPtr

Use this section level rule to increment to the next occurrence of a Record Dictionary record. Place this rule in the DDT file.

Syntax
```
;IncDataDictRecPtr;RecordName {,MoreOptionalRecords};;
```

Example
```
;IncDataDictRecPtr;Tariff , Tarriff2;;
```

PosDataDictRecPtr

Use this section level rule to advance the record pointer until the condition is true. Place this rule in the DDT file.

Syntax
```
;PosDataDictRecPtr;Record(RecordName) Cond(ConditionName);;
```

Example
```
;PosDataDictRecPtr;Record(Meter) Cond(CompareMeterTariff);;
```

PostIncDataDictRecPtr

Use this section level rule to increment to the next occurrence of a Record Dictionary record. Place this rule in the DDT file.

Syntax
```
;PostIncDataDictRecPtr;RecordName {,MoreOptionalRecords};;
```

Example
```
;PostIncDataDictRecPtr;Tariff , Tarriff2;;
```

PostPosDataDictRecPtr

Use this section level rule to advance the record pointer until the condition is true. Place this rule in the DDT file.

Syntax
```
;PosDataDictRecPtr;Record(RecordName) Cond(ConditionName);;
```

Example
```
;PostPosDataDictRecPtr;Record(Meter) Cond(CompareMetrTarif);;
```

PreIncDataDictRecPt

Use this section level rule to increment to the next occurrence of a Record Dictionary record. Place this rule in the DDT file.

Syntax
```
;PreIncDataDictRecPtr;RecordName {,MoreOptionalRecords};;
```

Example
```
;PreIncDataDictRecPtr;Tariff , Tarriff2;;
```

PrePosDataDictRecPtr

Use this section level rule to advance the record pointer until the condition is true. Place this rule in the DDT file.

Syntax

```
;PosDataDictRecPtr;Record(RecordName) Cond(ConditionName);;
```

Example

```
;PrePosDataDictRecPtr;Record(Meter)Cond(CompareMeterTariff);;
```

ResetDataDictRecPtr

Use this rule to reset the pointer of a Record Dictionary record. Place this rule in the DDT file.

Syntax

```
;ResetDataDictRecPtr;RecordName;;
```

Example

```
;ResetDataDictRecPtr;Meter;;
```

Appendix B

Using Image Editor to Enter Rule Information

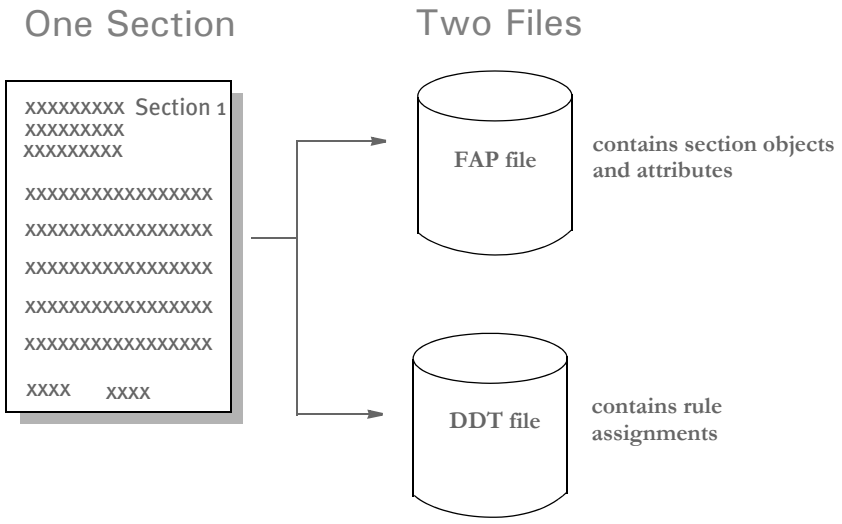
This appendix explains how to add, remove, and edit rule assignments using Image Editor. It also explains how to generate information reports. You should only set up rules if you fully understand mapping procedures, rules, and if you are using Documaker Server.

In this appendix you will find information about:

For reference information on individual rules, see Chapter 5, Section and Field Rules Reference on page 274.

Storing Rule Information in DDT Files

The Image Editor stores sections (formerly known as *images*) in a FAP file. FAP files created with Image Editor only contain the section's objects and object attributes. The Image Editor stores section and field rule assignments in a separate file, called a *data definition table* (DDT) file.



Remember, as you assign rules with the Image Editor you affect the DDT file, *not* the FAP file.

NOTE: Documaker Studio stores sections (images) in a FAP file, along with the section and field rule assignments you assign to it. This differs from the way rule information is stored in Image Editor.

With the release of version 11.0 and the introduction of Documaker Studio's FOR file, section and field-level rules previously stored in the DDT file are, in Studio implementations, either unnecessary or are stored in the FAP file. Having section level rules (such as SetOrigin) in the FOR file makes it easier to do visual form design. Having field level rules in the FAP file eliminates synchronization worries.

Using the Data Definition Table

When you use Image Editor, the system stores your section and field level rule assignments in a separate, semi-colon delimited file called a data definition table (DDT). This file contains all the rule assignments that apply to a specific section. The system creates the DDT file and stores it in the master resource library.

NOTE: The Image Editor stores the section in a FAP file and it stores the section rule assignments in a DDT file. The file names correspond, but the extensions differ, for example, *IMAGE.FAP* and *IMAGE.DDT*. Storing information separately makes it easier to apply and modify rules. This also helps the system process your information faster.

You can set up section and field rules directly in the DDT file using any ASCII text editor or you can use the Image Editor to make the same assignments. For information on using the Image Editor, see Assigning Rules with the Image Editor on page 515.

In the DDT file examples below, each semicolon delimited field represents a distinct piece of variable field formatting, mapping, or data information. There are several levels of rules:

- Field Rules - Field rules affect individual fields on a section. These rules are stored in the DDT file, along with the section rules. You run field level rules by including the rule in a section's DDT file under the <Image Field Rules Override> section.
- Section Rules - Section rules are stored in the DDT file. These rules affect specific sections. Section rules associate the rule name with the data required by the rule. You run section level rules by including the rule in a section's DDT file under the <Image Rules> section.

Here is an example of the DDT file format:

```
<Image Rules>
;IMAGERULE;IMAGERULEDATA;
;IMAGERULE;IMAGERULEDATA;

<Image Field Rules Override>
;#1;#2;#3;#4;#5;#6;#7;#8;#9;#10;#11;#12;#13;#14;#15;#16;#17;#18
```

Where:

In the <Image Rules> section:

```
IMAGE          the section rule name
IMAGERULEDATA  the data required by the section rule
```

In the <Image Field Rules Override> section:

```
#1      the source file number
#2      the source record index number in the record section
#3      the source field name
#4      the source field offset
#5      the source field length
#6      the destination field name
#7      the destination field offset
#8      the destination field length
#9      the field format mask
#10     the field rule name (DDT rule)
#11     the data for the field rule
#12     flag 1 (not required)
#13     flag 2 (host required)
#14     flag 3 (operator required)
#15     flag 4 (either required)
#16      x offset
#17      y offset
#18      font
```

Line #11 contains the data for the field rule. This data varies, depending on the rule. The data can consist of the following, in the order shown below:

```
Search Criteria, Extract Field Descriptors, Additional Parameters
```

For some rules, the data field (#11) is empty, such as for the Master and KickToWip rules. For other rules, the data field can consist of only search criteria, such as for the SetState or Move_It rules (when used without overflow data).

Each item must be separated by a single space—no other spaces are allowed. If the rule supports overflow, you should place the overflow data *before* the search criteria.

NOTE: For details about overflow, see Overflow and User Functions on page 271. For more information about search criteria, see Search Criteria on page 270.

The way the delimiters are used in rules is almost the same. There are four types of delimiters, as shown here:

Delimiter	Description
space	A single space separates different groups of data as mentioned above.
backslash (/)	A backslash separates overflow data and other data.
colon (:)	A colon separates conditions in rules such as PrintIf and PrtIfNum. It is also used to separate two dates in the rule, such as in the DateDiff rule.
comma (,)	A comma is used in search criteria to delimit offset and data. It is also used to separate extract field descriptors, such as offset and length.

Setting Up the MASTER.DDT File

The MASTER.DDT file helps you save time by letting you set up common fields used on many forms. For instance, if you have a field called *Name* which appears on many forms, you can use the MASTER.DDT file to set it up once and then use that information as necessary.

For instance, suppose you have a section named MYIMAGE.FAP, which has a corresponding DDT file named MYIMAGE.DDT.

In MYIMAGE.DDT you have a field called Name. For the Name field, you select the rule, Master. This tells the system to look in the MASTER.DDT file for a field called Name and use the settings for length, rule, mask, data, and so on, stored there.

To use this capability, follow these steps:

1. Select the Edit, Master DDT option in Docucreate to open the MASTER.DDT file and map the extract data to the variable fields in the MASTER.DDT file.
2. In the individual DDT files, enter either *blank* or *master* in the Rule field. The variable name must be the same in the master and subordinate DDT files.
3. In the AFGJOB.JDT file, add this rule:

```
;LoadDDTDefs;;;
```

Taking precedence

If a field in the FAP file is specified in the master DDT, the system uses the information in the master DDT.

When using the master DDT during rules processing, every field defined in a DDT file will fill in any blank item (or zero in the case of numeric items) from the matching master template for that field. So any field rule component that you leave blank, or zero, will be filled in from the master DDT if that field is declared within the master DDT.

Image DDT lines that specify Master as the rule or have no rule specified also accept the rule from the master DDT. But like all other DDT rules, if a rule name other than Master is specified in the DDT, it will be used. Here are some examples.

Suppose the master DDT contains these lines:

```
;0;0;NAME;25;50;NAME;0;50;;move_it;11,INSNAMREC;N;N;N;N;0;0;15412
;0;0;PRICE;15;10;FINAL
```

Now suppose the section DDT contains these lines.

```
;0;0;NAME;0;0;;0;0;;master;;;;;;12400;2340;0
;0;0;PRICE;0;0;PRICE;0;10;;movenum;;N;N;N;N;13330;1900;12310
;0;0;LOCATION;75;50;LOCATION;0;50;;move_it;;N;N;N;N;14100;2100;0
;0;0;STATE;0;0;;0;0;;;;;;;;0;0;0
```

After the master template is applied, the section DDT rules would look like this:

```
;0;0;NAME;25;50;NAME;0;50;;move_it;11,INSNAMREC;N;N;N;N;12400;2340;15
412
;0;0;PRICE;15;10;PRICE;0;10;C,10.2;movenum;11,INSNAMREC;N;N;N;N;13330
;1900;12310
```

Note the items in bold.

The rule line for field NAME was basically blank in the DDT and specified *Master* as the rule. Therefore, any item left blank, or zero, was filled in from the matching NAME field in the master DDT. In addition, the rule name of *Master* was replaced with the rule name from the template.

The rule line for field PRICE does not specify the Master rule. Note, however, that the zero and blank items were still filled in from the master template. In this example, that included the source offset and length, the field rule flags, and the search mask data.

Also look at the required flags. In the master DDT for this field, these items are specified as *;N;Y;N;N;* but because the section DDT was not left blank for those items (;N;N;N;N;), they did not get copied and were left intact. Also note that the Source Field name *FINAL PRICE* was not copied into the section DDT line, because data *PRICE* already occupied that space on the section DDT line.

For the field LOCATION, nothing about the section DDT line changed because the field does not occur in the master DDT.

For the field STATE, nothing on the DDT line changes because it does not appear in the master DDT. But unlike the field LOCATION, because the rule name was specifically declared as *Master*, it was expected to be found in the Master. This will cause an error indicating that the field was missing in the master DDT.

Using the Master DDT Editor

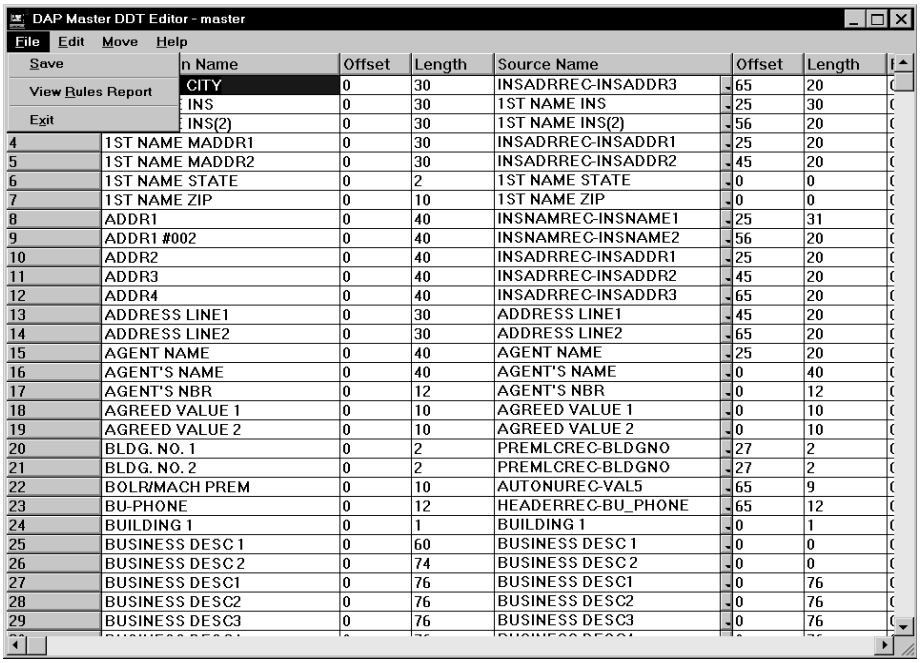
You can use the Master DDT Editor to work with your master DDT file. This tool only edits the Master DDT file. You cannot use it to edit other DDT files.

The Master DDT Editor presents the information stored in the Master DDT file in a spreadsheet-like format, as shown below. You can use the various menu options to view a report, save your work, make changes, move assignments up or down, and perform other tasks.

The following topics discuss the tasks you can perform using the menu option for the Master DDT Editor.

Using the File Menu

The File menu options let you save your changes, generate a report that documents all field rule assignments in the master DDT file, or exit the Master DDT Editor. To display the File menu, choose File. The File menu appears.



	n Name	Offset	Length	Source Name	Offset	Length
	CITY	0	30	INSADRREC-INSADDR3	65	20
	INS	0	30	1ST NAME INS	25	30
	INS(2)	0	30	1ST NAME INS(2)	56	20
4	1ST NAME MADDR1	0	30	INSADRREC-INSADDR1	25	20
5	1ST NAME MADDR2	0	30	INSADRREC-INSADDR2	45	20
6	1ST NAME STATE	0	2	1ST NAME STATE	0	0
7	1ST NAME ZIP	0	10	1ST NAME ZIP	0	0
8	ADDR1	0	40	INSNAMREC-INSNAME1	25	31
9	ADDR1 #002	0	40	INSNAMREC-INSNAME2	56	20
10	ADDR2	0	40	INSADRREC-INSADDR1	25	20
11	ADDR3	0	40	INSADRREC-INSADDR2	45	20
12	ADDR4	0	40	INSADRREC-INSADDR3	65	20
13	ADDRESS LINE1	0	30	ADDRESS LINE1	45	20
14	ADDRESS LINE2	0	30	ADDRESS LINE2	65	20
15	AGENT NAME	0	40	AGENT NAME	25	20
16	AGENT'S NAME	0	40	AGENT'S NAME	0	40
17	AGENT'S NBR	0	12	AGENT'S NBR	0	12
18	AGREED VALUE 1	0	10	AGREED VALUE 1	0	10
19	AGREED VALUE 2	0	10	AGREED VALUE 2	0	10
20	BLDG. NO. 1	0	2	PREMLCREC-BLDGNO	27	2
21	BLDG. NO. 2	0	2	PREMLCREC-BLDGNO	27	2
22	BOLR/MACH PREM	0	10	AUTONUREC-VAL5	65	9
23	BU-PHONE	0	12	HEADERREC-BU_PHONE	65	12
24	BUILDING 1	0	1	BUILDING 1	0	1
25	BUSINESS DESC 1	0	60	BUSINESS DESC 1	0	0
26	BUSINESS DESC 2	0	74	BUSINESS DESC 2	0	0
27	BUSINESS DESC1	0	76	BUSINESS DESC1	0	76
28	BUSINESS DESC2	0	76	BUSINESS DESC2	0	76
29	BUSINESS DESC3	0	76	BUSINESS DESC3	0	76

Saving your work

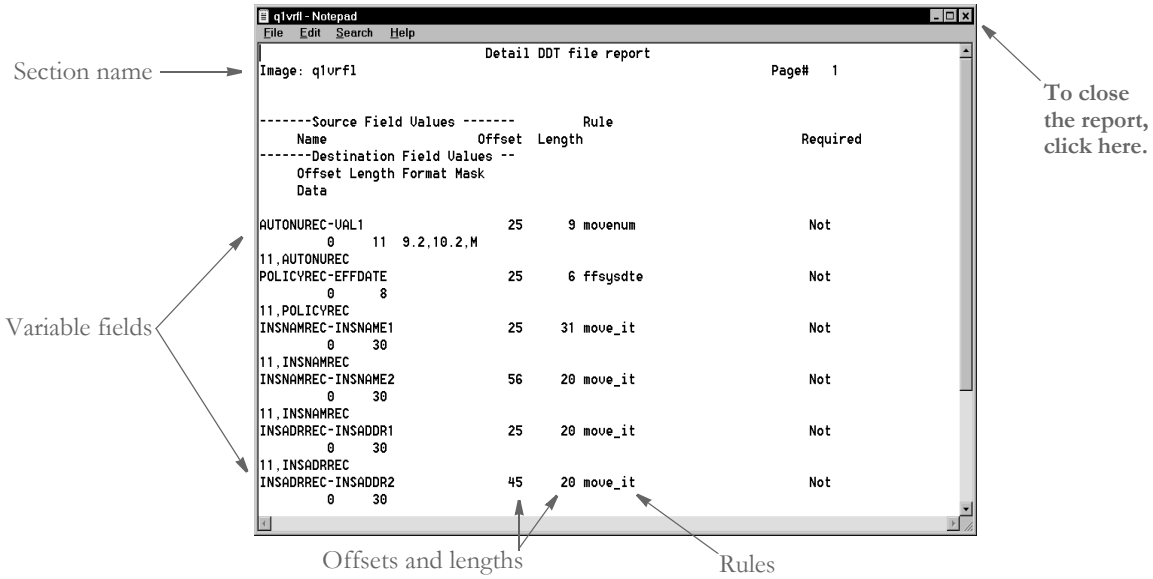
The system saves your work as you move through the rows displayed on screen. You can also use the Save option to save all of the additions, changes and deletions you make to rule assignments in the Master DDT file.

To save rule assignments, choose File, Save.

Viewing the Rules Report

The View Rules Report option shows you a listing of all fields in the DDT file and pertinent information about each field. The listing shows the information in a tabular fashion. The variable fields' offsets, lengths, assigned rules, and data requirements appear.

The system creates this report from data stored in memory when you open the Master DDT Editor. To view the Rules Report, choose File, View Rules Report. The Rules Report appears.



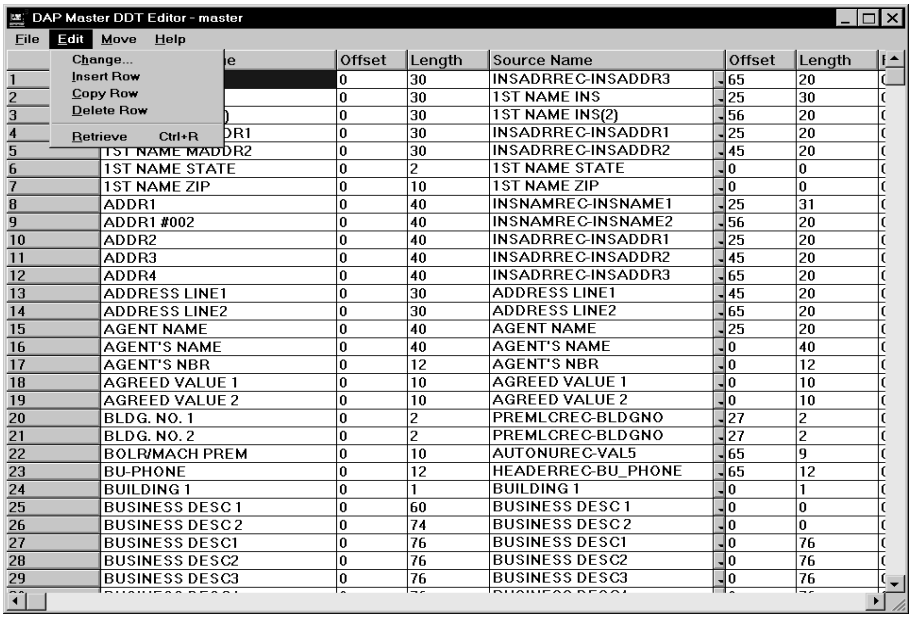
After you finish viewing the rules report information, you can close the report by double clicking on the icon in the top left corner of the window.

Exiting the Master DDT Editor

To exit the Master DDT Editor, choose File, Exit.

Using the Edit Menu

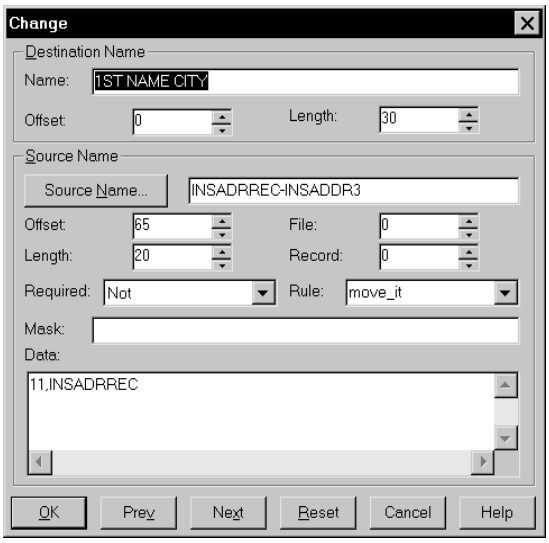
The Edit menu options let you change, insert, copy, or delete rows, and retrieve rule assignments. To display the Edit menu, choose Edit. The Edit menu appears.



		Offset	Length	Source Name	Offset	Length
1		0	30	INSADRREC-INSADDR3	65	20
2		0	30	1ST NAME INS	25	30
3		0	30	1ST NAME INS(2)	56	20
4	DR1	0	30	INSADRREC-INSADDR1	25	20
5	1ST NAME MADDR2	0	30	INSADRREC-INSADDR2	45	20
6	1ST NAME STATE	0	2	1ST NAME STATE	0	0
7	1ST NAME ZIP	0	10	1ST NAME ZIP	0	0
8	ADDR1	0	40	INSNAMREC-INSNAME1	25	31
9	ADDR1 #002	0	40	INSNAMREC-INSNAME2	56	20
10	ADDR2	0	40	INSADRREC-INSADDR1	25	20
11	ADDR3	0	40	INSADRREC-INSADDR2	45	20
12	ADDR4	0	40	INSADRREC-INSADDR3	65	20
13	ADDRESS LINE1	0	30	ADDRESS LINE1	45	20
14	ADDRESS LINE2	0	30	ADDRESS LINE2	65	20
15	AGENT NAME	0	40	AGENT NAME	25	20
16	AGENT'S NAME	0	40	AGENT'S NAME	0	40
17	AGENT'S NBR	0	12	AGENT'S NBR	0	12
18	AGREED VALUE 1	0	10	AGREED VALUE 1	0	10
19	AGREED VALUE 2	0	10	AGREED VALUE 2	0	10
20	BLDG. NO. 1	0	2	PREMLCREC-BLDGNO	27	2
21	BLDG. NO. 2	0	2	PREMLCREC-BLDGNO	27	2
22	BOLR/MACH PREM	0	10	AUTONUREC-VAL5	65	9
23	BU-PHONE	0	12	HEADERREC-BU_PHONE	65	12
24	BUILDING 1	0	1	BUILDING 1	0	1
25	BUSINESS DESC 1	0	60	BUSINESS DESC 1	0	0
26	BUSINESS DESC 2	0	74	BUSINESS DESC 2	0	0
27	BUSINESS DESC1	0	76	BUSINESS DESC1	0	76
28	BUSINESS DESC2	0	76	BUSINESS DESC2	0	76
29	BUSINESS DESC3	0	76	BUSINESS DESC3	0	76

Changing rule assignment settings

Use the Edit, Change option to display the Change window. From this window you can quickly change all of the information for a rule.



You can also access the Change window by double clicking on the row number to the left of the destination name. This window contains the same fields you see on the editor window—it's just another way to make changes to an existing rule.

From this window, you can use the Next and Prev buttons to move from row to row.

You can also click Reset to return the various properties back to the original settings. This however does not revert back to changes made before the file was saved. The Cancel button also resets the properties, but exits the window too.

Click Ok to save your changes and exit the window.

Inserting a row

The Insert Row option lets you insert a blank rule assignment record at the end of the list. Use Insert Row if you have deleted a rule assignment and you want to add it back to the rule assignment list or if you want to add more rule assignments.

In very unique situations you can use this option to execute a variable field level rule without attaching the rule to a field in the section. If you need to execute a rule without attaching it to a section field, consult your system administrator.

To insert a rule assignment, choose Edit, Insert Row. The Master DDT Editor adds a blank rule assignment record to the end of the list. The record appears in red. For each assignment, you must make entries in the Destination Name and Rule fields.

Coping a row

The Copy Row option lets you copy an existing rule assignment record to the end of the list. Use Copy Row if you want to base a new rule assignment on an existing rule assignment.

To copy a rule assignment, click the variable field row that you want to copy. Choose Edit, Copy Row. The Master DDT Editor copies the highlighted row and adds it to the end of the list.

Deleting a row

The Delete Row option lets you remove a rule assigned to a field. You remove the field and its assigned rule from the DDT file not from the FAP file.

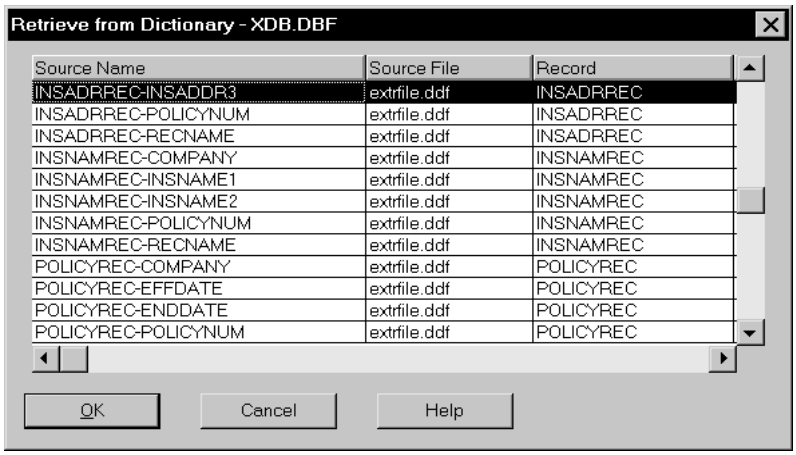
1. To delete a rule assignment, click the variable field for which you want to delete a rule. Choose Edit, Delete Row. The Confirm Delete message appears.
2. Click Yes or No to confirm or cancel the delete.

Retrieving information from the data dictionary

The Retrieve option lets you retrieve a record from the data dictionary for use in a rule assignment. When you choose this option the system creates a list of all the source name entries in the data dictionary.

After selecting the Retrieve option, you can copy source information from the data dictionary into your rule assignment. The Master DDT Editor places the information from the Source Name, Source Offset, Source Length, Record, Required, Rule, Mask, and Data fields into the current row. The destination name, destination length, and destination offset information are not copied.

1 To retrieve source information, click within the row assignment where you want to place source information. Choose Edit, Retrieve. The Retrieve from Dictionary window appears.



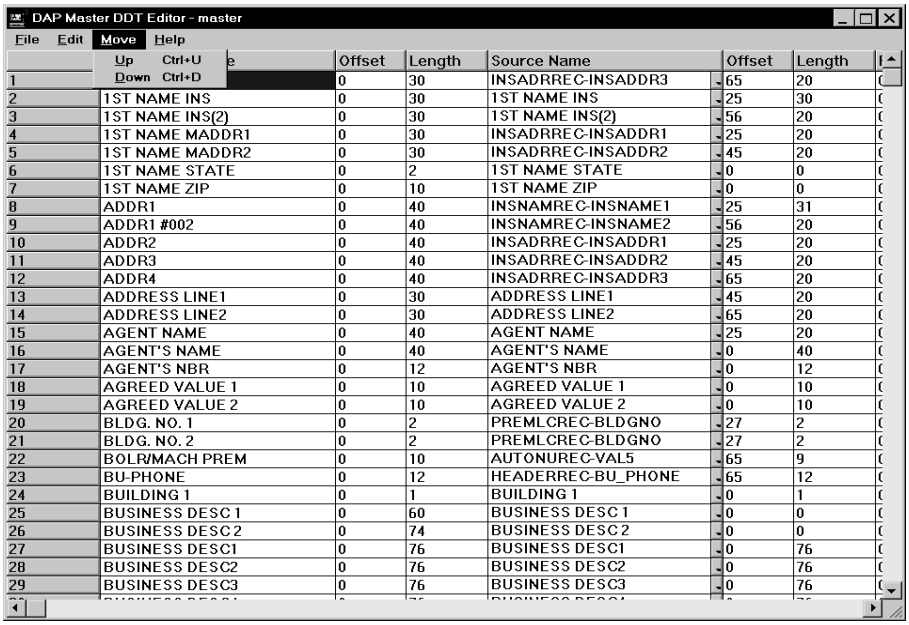
2 Click the source record that you want to retrieve as a rule assignment.

3 Click Ok. The Master DDT Editor places the information from the source record into the rule assignment.

Using the Move Menu

The Move menu lets you change the sequence of rule assignments. The row number for the variable field indicates its sequence number. The sequence number defines the order that the system processes the rule.

To display the Move menu, choose Move. The Move menu appears.



Using the Up option

The Up option lets you move a rule assignment up one sequence number at a time. Use Up when you want to alter the order in which a rule is executed during processing.

To move a rule assignment up, click the variable field rule assignment you want to move up. Choose Move, Up. The rule assignment moves up one sequence number.

Using the Down option

The Down option lets you move a rule assignment down one sequence number at a time. Use Down when you want to alter the order in which a rule is executed during processing.

To move a rule assignment down, click the variable field rule assignment you want to move down. Choose Move, Down. The rule assignment moves down one sequence number.

Assigning Rules with the Image Editor

Although you can enter rules directly into the DDT file, you will probably find it easier to set up your section and field level rules as you create the section.

You set up section level rules using the Image Properties window. Similarly, you set up field level rules using the field's Properties window.

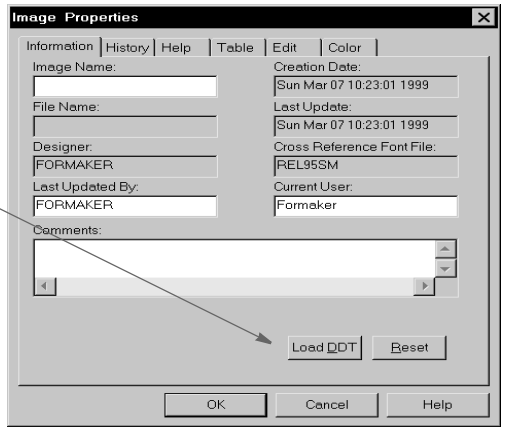
Adding Section Rules

To insert section rules, follow these steps:

1 With the section open, select the Format, Image Properties option. The Image Properties window appears.

No DDT file has been created for this section or the option to automatically update the DDT file is turned off.

Click here to create or load a DDT file.

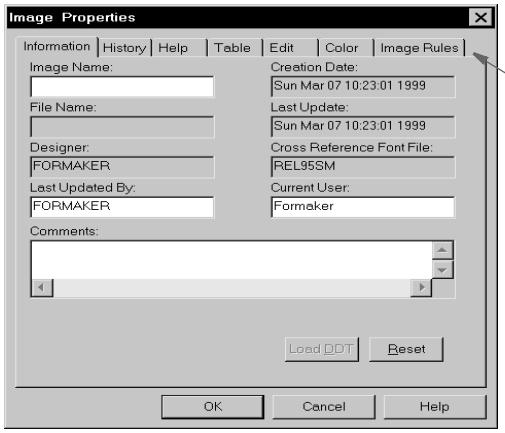


If you have no rules for the section and the option to automatically update the DDT file is turned off, the system shows the Load DDT button. This lets you create a DDT file for the section, into which you can add rules.

Also, if you have rules for the section and the option to automatically update the DDT files is turned off, the system shows the Load DDT button.

NOTE: To turn on or off the option to automatically update the DDT file, choose Options, Editor Properties while in the Image Editor. Then select the Save tab.

If you have already created and opened a DDT file for the section or the option to automatically update the DDT file is turned on, the Load DDT button is unavailable and you see an additional tab named Image Rules, as shown on the following window.



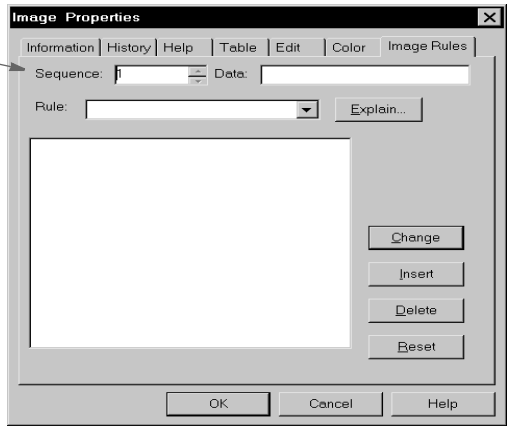
A DDT file exists and has been opened for this section or the option to automatically update the DDT file is turned on.

Click here to add, change, or delete section rules.

2 Either click the Load DDT button to create or load a DDT file or click the Image Rules tab. The Image Rules tab appears.

The system assigns a default sequence number for you.

You can change this number to change the order in which the rule appears in the DDT file and is executed by the system.



3 On the Image Rules tab, the system automatically assigns the next available sequence number for you. You can change the sequence number if necessary. Use the following fields to set up the rule's parameters.

Field	Description
Sequence	The number you enter in this field determines where the rule is placed in the DDT file and the order in which it is processed.
Rule	Use this field to select the section rule you want to insert.
Data	Enter data necessary to process the rule. Refer to the rule descriptions for more information about the data necessary for individual rules.

4 Once you set up the information for the rule, click Insert to add the rule to the DDT file.

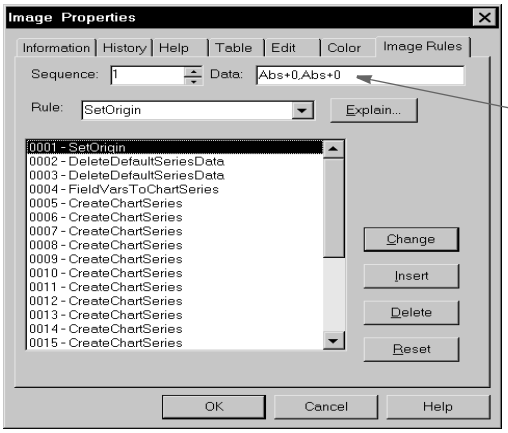
Changing a Section Rule

The Change button on the Image Rules tab lets you change a rule and its information. For instance, you can use this button to make changes to the data required for the rule.

1 To make changes to a section rule, select the Image Rules tab. A window similar to the one shown below appears.

All of the section rules for the section are shown here.

Click a rule to substitute another rule or make changes to a rule's sequence or data.



Here you can see the data for the SetOrigin rule.

2 Click the rule you want to change.

- To change the sequence of the rule, enter a new number in the Sequence field.
- To assign a different rule, select a rule in the Rule field.
- To change the data for a rule, click in the Data field and enter the new data.

3 Click Change to record your changes. Click Reset to cancel your changes. Click Cancel to cancel your changes and return to the Image Editor.

Deleting a Section Rule

Use the Delete button to remove a section level rule assignment. The system removes the rule from the DDT file. The FAP file is not affected.

1 Select the Image Rules tab and then click the rule you want to delete.

2 Click Delete and then click Yes or No to confirm or cancel the delete.

NOTE: Once you click Delete, you cannot click Reset to cancel the deletion.

ASSIGNING FIELD RULES

You can assign rules to fields two ways:

- Using the Edit DDT tab on the field's Properties window.
- Using the Edit DDT window.

Using the Edit DDT tab lets you work with a single field, while the Edit DDT window lets you work with all of the fields on the section at the same time. Choose the approach that best fits your working habits.

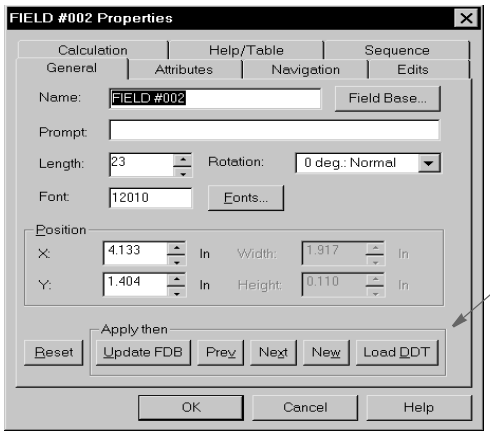
Using the Edit DDT Tab

To insert field rules using the Edit DDT tab, follow these steps:

1 With the section open, double click on the variable field or click once and select the Format, Object Properties option from the main menu.

NOTE: As with any object. you can also click on the variable field to select it and then right click to choose the Object Properties option from a pop-up menu.

The field's Properties window appears.



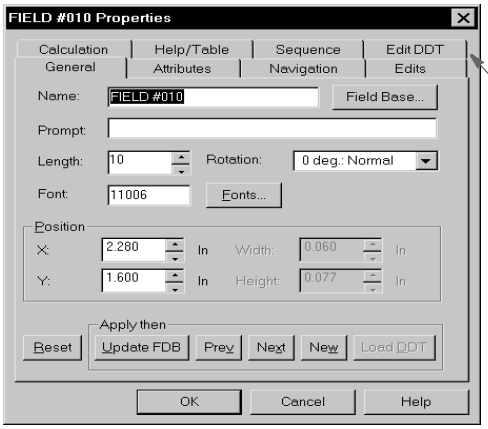
No DDT file has been created and opened for this section or the option to automatically update the DDT file is turned off.

Click here to create or load a DDT file.

If you have no rules for the section and the option to automatically update the DDT file is turned off, the system shows the Load DDT button. This lets you create a DDT file for the section, into which you can add rules.

NOTE: To turn on or off the option to automatically update the DDT file, choose Options, Editor Properties while in the Image Editor. Then select the Save tab.

If you have already created a DDT file for the section or the option to automatically update the DDT file is turned on, the Load DDT button is unavailable and you see an additional tab named Edit DDT, as shown on the following window.



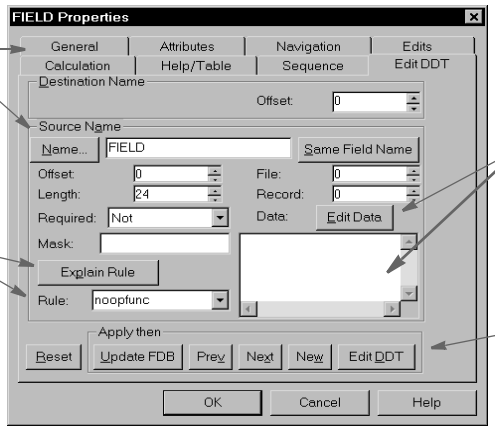
A DDT file exists and has been opened for this section or the option to automatically update the DDT file is turned on.

Click here to add, change, or delete a field rule.

2 Either click the Load DDT button to create or load a DDT file or click the Edit DDT tab. The Edit DDT tab appears.

The name of the Destination field is set on the General tab. This name serves as the default for the name of the source field.

Click the Explain Rule button to see a description of the rule selected in the Rule field.



You can enter data for the rule in this field or you can click here to enter the data in a larger window.

Click the Edit DDT button to work with a spreadsheet-like list of all the field rules for this section.

3 On the Edit DDT tab, use the following fields to supply information about the destination field.

Field	Description
Offset	Indicates the position in which the data begins in the destination field. This field is not a required field.
Length	Required. Indicates the number of characters in the destination field. The system automatically fills this field based on the length of the variable field.

4 On the Edit DDT tab, use the following fields to define the source field.

Field	Description
Name	Name of the source field. You can enter the source name by entering the name or by clicking the Name button. This button lets you retrieve and insert source names and associated source information from the data dictionary file. Assigning a source name to specific source information gives you a shortcut retrieval method and eliminates having to re-enter source information. Use the Same Field Name button to make the name you enter here apply to the destination field. If you changed the name on the Edit DDT window, you can click this button to tell the system to change that name to the one you entered on this tab. This is not a required field.
Offset	Required. Indicates the position in which the source field data begins.
Length	Required. Number of characters in the source field.
File	Number of the source file. Specify the number of the table file in the TBLFILE.DAT file you want to use as the data source.
Record	Number of the source record. This number tells the system which record to retrieve from the source file.
Required	Select one of the following options. These options control whether data must merge in the section's variable field during the merge procedure. *Not* required at processing time. Missing data does not result in an error message. *Host Required* from a source data file during a batch run. Missing data results in an error message and the system places the form set in the error batch. Print the error batch to review and correct any errors. *Operator Required* as a manual entry in the WIP. Missing data results in a warning message and the document is kicked to WIP. *Either Required* as a manual entry in WIP. Missing data results in a warning message and the document is kicked to WIP.
Rule	Select the name of the rule to execute. This is a required field. See Section and Field Rules Reference on page 274 for a list and brief description of the rules from which you can choose.
Mask	If required, enter the mask necessary to execute the rule.
Data	Enter the data required to execute the rule. Click Edit Data to enter the data in the Edit DDT Data window. This gives you a larger window in which to make your entries.

5 When you finish entering the information for the field, click Update FDB to update the field database with any applicable information. Click Ok to finish and close the Properties window. Click Reset to cancel your entries. The system saves your entries when you save the section.

NOTE: The Field Database Editor lets you store and retrieve variable field information to make setting up and creating sections faster and more consistent. The field database contains a record for each unique variable field. Each record contains information such as the field name, font, type, and so on. No DDT information is stored in the field database.

Changing a Field Rule

To make changes to a field rule, simply double-click the field and select the Edit DDT tab. Then change the various fields as necessary.

Click Ok to finish and close the Properties window, Reset to cancel your changes, or Update FDB to update the field database with any applicable information.

If you need to make changes to several field rules, there are two ways to do this.

- You can double click one of the fields, select the Edit DDT tab, make your changes, and then use the Next and Prev(ious) buttons to move to the next variable field which requires changes.
- You can click the Edit DDT button on the Edit DDT tab and make all of your changes from the Edit DDT window.

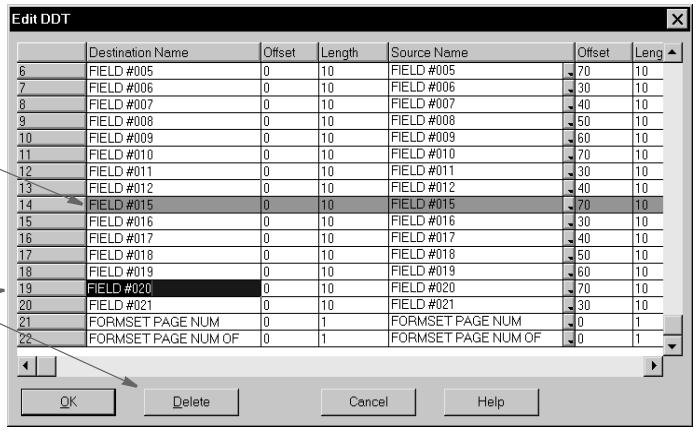
NOTE: If you are making changes to several fields and using the Next and Prev buttons to move from field to field, be aware that Next and Prev save your changes to a field. If you click Reset, the system only cancels your changes for the current field.

Deleting a Field Rule

You delete field rules from the Edit DDT window. To display this window, double click on any field *other than the one* you want to delete and select the Edit DDT tab from the Properties window. Then click the Edit DDT button. The Edit DDT window appears.

This is the assignment for the current field. On most screens, it appears in green. You cannot delete the assignment for a field if it's the field you are working with.

To delete a rule assignment for another field, click the row and then click Delete.



Click the row you want to delete and then click Delete. The system *removes* the rule assignment from the DDT file and the field from the FAP file to make sure *both* files are in sync.

NOTE: The Automatically Update DDT option does not affect deletions.

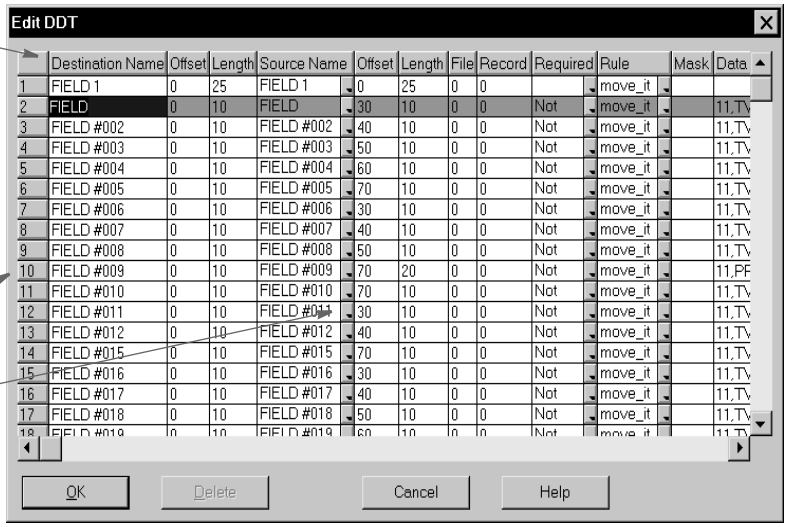
USING THE EDIT DDT WINDOW

You can enter field rule information for a specific field on the Edit DDT tab or you can use the Edit DDT window to work with all the fields on a section at once. To display this window, double click a variable field, select the Edit DDT tab, and then click the Edit DDT button. The Edit DDT window appears.

With the exception of the Destination Length, the columns on this window correspond to the fields on the Edit DDT tab—it's just a different way to present the information.

Click on these buttons to select a field. For example, click here to select FIELD #10.

These drop downs let you select from a list. For example, click here to retrieve source field information from the data



NOTE: The columns on this window were resized to show all of the fields. You can resize any column in the window.

The first column lists the fields that exist on the section, or FAP file. This column always remains visible on the window. Use the scroll bar at the bottom of the window to scroll left or right and display other columns.

ASSIGNING A RULE

The Edit DDT window lets you quickly and easily assign rules to the variable fields. You simply move from column to column and row to row to make your rule assignments. You can also change destination names.

When you select a new row, any changes you made to a previous row are saved in memory but not written to the DDT file. The information is saved when you save the section.

1. To assign a rule, highlight the variable field you want to assign a rule to by clicking on it. The field name automatically defaults for you in the Destination Name field. This is the section field that receives data during processing. Do not change this name.
2. Enter the offset for the destination field in the Offset field. This field indicates the beginning position of a piece of data in a variable field. For example, if you need the data to be indented or offset in the variable field once processing has occurred, you would enter that offset in this field.
3. Enter the length for the destination field in the Length field. The system defaults to the variable field's length.
4. Enter the name of the source field in the Source Name field or click the drop down arrow to retrieve source field information from the data dictionary. Assigning a source name to specific source information eliminates having to re-enter source information.
5. Enter the offset for the source field in the Offset field. This indicates the first position of the data in the extract file.
6. Enter the length for the source field in the Length field. This indicates the length of the data in the extract file.
7. If you are using the TblLkUp rule, specify the number of the table file in the File field. Table file information is stored in the TBLFILE.DAT file. This file tells the system where to find individual table (TBL) files. Here is an example:

   ```
   .\DEFLIB\AGENCY.TBL
   .\DEFLIB\COMPCODE.TBL
   ```

   This information is used by the TblLkUp rule. The system looks at the number in the File field to determine which *TBL* file to use. Based on the search mask information, the system then looks in that *TBL* file for the text.

NOTE: If you are not using the TblLkUp rule, no entry is required in this field.

8. Enter the record number in the Record field. The record number tells the system which record to retrieve from the source file—a one (1) means the first record found, a two (2) means the second record, and so on.
9. Select one of these options in the Required field. Data requirements control whether data must merge into the field during the merge procedure.

Option	Description
Not	Not required at print time. Missing data does not result in an error message.
Host	Required from a source data file during a batch run. Missing data results in an error message and the document kicks to the error batch.
Operator	Required as a manual entry in the WIP module. Missing data results in a warning message and the document is kicked to WIP.
Either	Required as manual entry in the WIP module. The system first tries to fill the field with data from the extract file. If the data is missing, the system kicks the document to WIP and creates a warning message.

NOTE: By customizing your INI file settings, you can have the system send all transactions it could not process to a specific file, commonly called the error batch (ERROR.BAT). You can print the error batch after Documaker Server finishes. Transactions listed in this file are not sent to WIP or archived. These transactions must first be corrected before they can be sent to WIP or archive. For information about error batches, refer to the Documaker Server System Reference.

10 Select a rule from the list in the Rule field. See Section and Field Rules Reference on page 274 for a list and brief description of the rules from which you can choose.

11 If a mask is required during the processing of the rule, enter the mask in the Mask field.

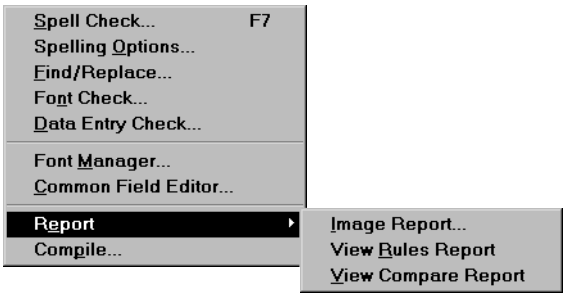
12 Enter the data in the Data field. The data string is the information that points to the record location that identifies the record in the extract file.

After you make all the rule assignments, click OK to return to the Properties window. Click Reset to undo your changes and display the Properties window. Click Cancel to undo your changes and close the Properties window.

Displaying Rule Reports

You can generate a report that documents all field rule assignments in the current DDT file. You can also generate a report that compares all fields in the section's DDT file to all fields in the section's FAP file.

To view these reports, first select the Reports option from the Image Editor's Tools menu. This option shows you the available reports.



NOTE: These reports are only available if the DDT file has been loaded. If the section has no DDT file or if the DDT file has not been loaded, these reports are unavailable.

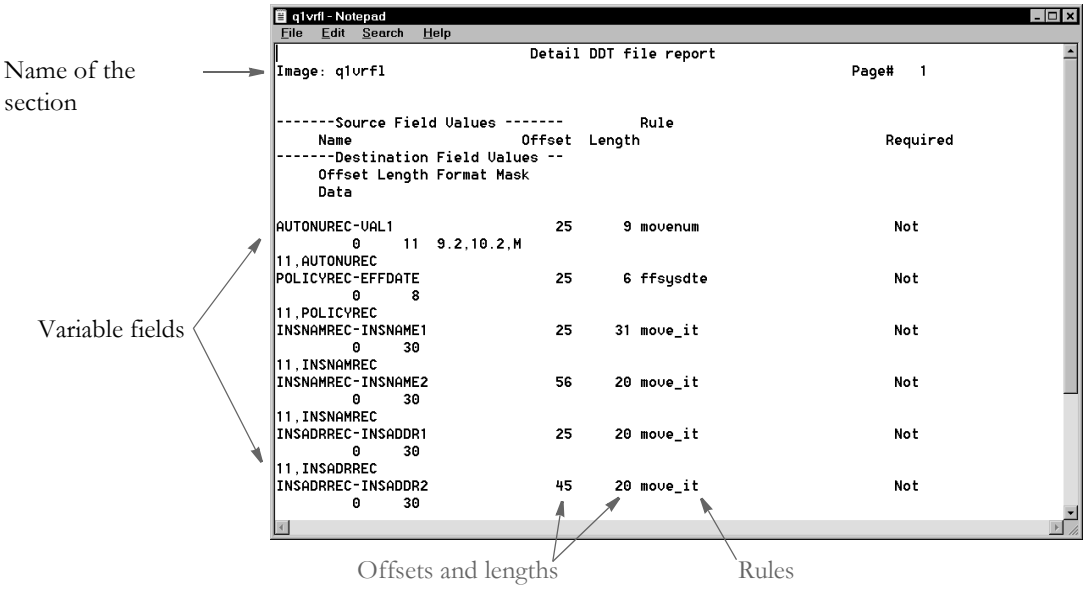
Image Report

The Image Report option lets you view a listing of fonts used, section dimensions, variable field information, and so on. This report does not include information about rules.

View Rules Report

The View Rules Report option lets you view a listing of all fields in the DDT file and pertinent information about each field. The Image Editor shows the information in a tabular fashion. The variable fields' offsets, lengths, assigned rules and data requirements appear.

The system creates this report from data stored in memory when you open the Image Editor. The system creates the report as a text file and then displays it using your default text editor, such as Notepad.



After you finish viewing the rules report information, you can use Notepad to save, print, edit, or perform other tasks. For instance, you can...

- Save the report by choosing File, Save.
- Print the report by choosing File, Print.
- Close the report by double clicking on the icon in the top left corner of the window.

View Compare Report

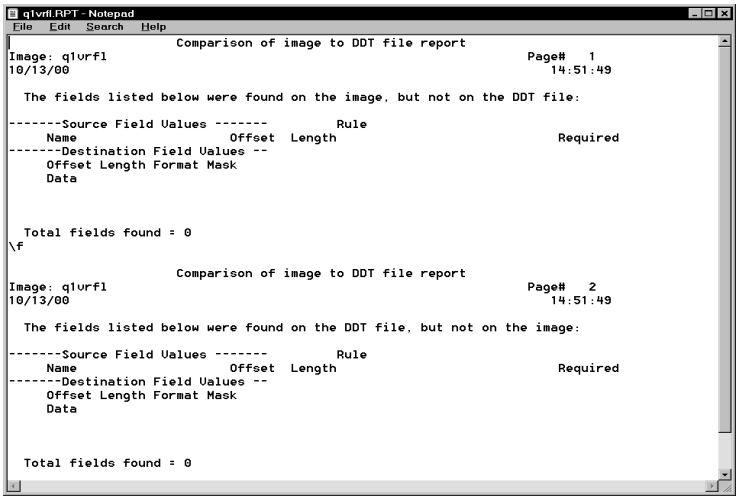
The View Compare Report option lets you view and compare a listing of all the fields stored in the section's FAP file with all the fields in the section's DDT file. Differences between the two files appear in the report.

The system displays a message stating that the rules (DDT) file will be saved before running the report. Click Ok to save the DDT file and create the report. Click Cancel to return to the Image Editor.

The system creates the report as a text file and then displays it using your default text editor, such as Notepad.

The report provides the names of variable fields found in the section (FAP) file but not found in the DDT file. The fields' offsets, lengths, rules, and data flags also appear.

Name of the section



You can use this report to locate fields that need to be mapped or to make sure variable field lengths are long enough to contain the data described by the destination length in the DDT file.

After you finish viewing the compare report information, you can use Notepad to save, print, edit, or perform other tasks. For instance, you can...

- Save the report by choosing File, Save.
- Print the report by choosing File, Print.

Close the report by double clicking on the icon in the top left corner of the window.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z